JAIST Repository
https://dspace.jaist.ac.jp/

Title	ターン制戦略ゲームへの深層学習の適用
Author(s)	木村，富宏
Citation	
Issue Date	2020-12
Type	Thesis or Dissertation
Text version	ETD
URL	http://hdl.handle.net/10119/17049
Rights	
Description	Supervisor:池田　心，情報科学研究科，博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

博　士　論　文

ターン制戦略ゲームへの深層学習の適用

木村　富宏

主指導教員　池田 心

北陸先端科学技術大学院大学

情報科学研究科

令和２年12月

概要

本論文では，ゲーム研究に新しく採用されるようになってきた深層学習をターン制戦略ゲームに適用し強いプレイのできる AI を作ることを通じて，そのためには何が必要であるのか，旧来のアルゴリズムではどうして性能を発揮できなかったのか，そして深層学習で性能を発揮するためにはどのような技術的課題があり解決する必要があるのかについて明らかにすることを試みた．

人工知能は長年にわたって研究され，広い応用範囲を持つ重要な学問分野である．その中でもゲームは再現や評価が比較的容易であるなどといった性質が取り扱いやすい，という事情から人工知能研究の題材として多く取り上げられてきており，木探索やモンテカルロ法や機械学習といった技術が適用または提案されてきた．近年，AlphaGo や AlphaZero といったブレイクスルーにより，人工知能研究はさらに活発となっている．

本論文で題材とするターン制戦略ゲームは人気のある分野の一つであり，囲碁や将棋と比べても複雑で新しい課題を多く持つことから研究の意義は大きい．特に各プレイヤーが盤上の全ての自分の駒を任意の順番で動かすことができるために行動の選択肢が指数的に増加し，いわゆる計算爆発を起こすことはターン戦略ゲーム用 AI の設計上の解決が困難な課題であった．また，ターン制戦略ゲームでは駒や地形とった複雑な要素があるために扱うデータが複雑かつ多層的であることも解析を困難にしていた．これらの事情のためにターン制戦略ゲームの AI 研究は顕著な進展がみられていなかった．本論文では AlphaZero によって提案された深層学習を使うアルゴリズムを適用して，上記の困難を解決する提案を行いつつ，ターン制戦略ゲームの AI 研究を進展させることを図った．

まず最初に本論文にかかわるゲームの既存研究について，ゲーム研究でよく使用されるアルゴリズムであるモンテカルロ木探索（Monte Carlo Tree Search: MCTS）についてや，確率を使用した探索である実現確率探索や盤面評価関数の機械学習について調査を行い，さらに，深層学習と強化学習を組み合わせた AlphaGo/ AlphaGo Zero/ AlphaZero 等の手法についても考察した．

ターン制戦略ゲームは長い歴史を持ち，いくつもの要素が追加削除されながら発展してきた．ターン制戦略ゲームは AlphaZero が適用されたチェス，将棋，囲碁と比較しても行動空間が非常に大きく，AlphaZero を直接適用できない意義深く挑戦的な課題であると言える．そこで本論文では，このゲームジャンルを題材とし，その代表的な要素のみを持つ研究用フレームワーク TUBSTAP を用いて実験を行うこととした．

TUBSTAP での研究を効率的に行うため，まずベンチマークマップを多数準備し

た．ベンチマークマップとは，作成したアルゴリズムの基本的な性能をチェックするための，人間からみて比較的簡単な問題のマップであり，2016 年ゲームプログラミングワークショップにて発表されたものである．これは TUBSTAP 向けに設計・制作され，全部で 8 グループ，43 個のマップが含まれている．ベンチマーク問題によって既存アルゴリズムの性能と限界について調査した結果，大会上位入賞レベルのプログラムを用いても，人間が容易に解ける問題すら解けないことが頻繁にあることがわかった．

ターン制戦略ゲームで扱っている複雑なデータ構造や行動表現を深層学習のニューラルネットワークで表現できるようにすることは，実際に適用する上では課題となる．本研究ではまず，複雑で入れ子状態であるデータ構造を簡潔に表現し，ニューラルネットワークの出力数を削減するために時分割でデータ出力できるリカレントネットワークを導入した．このネットワークを評価するため，強化学習によるコンピュータプレイヤーの作成を試みた．強化学習手法としては疎な報酬に対して鋭い立ち上がりを示す Profit Sharing 法を使用し，ベンチマーク問題で検討した経路探索と追跡マップについて 20 個以上のマップについて学習を行った結果，エピソード成功割合において 20% から 30% という正答率となり，これは DQN アルゴリズムと比較しても高い性能である．

さらに，2016 年 UEC-GAT 大会で上位入賞した MCTS アルゴリズム同士の対戦による大量の棋譜データを生成し，これによって policy network を訓練した．生成された棋譜データの量は試合数で約 10 万試合，局面数で約 80 万局面であった．policy network の性能評価を棋譜作成に利用した MCTS アルゴリズムとの対戦によって行い，50% 以上の勝率を示し，探索なしで推論のみで動作するにもかかわらず探索しているアルゴリズムに勝ち越したことになった．

続いて，AlphaZero によって提案された深層学習を利用したアルゴリズムをターン制戦略ゲームに適用するための課題を検討し，整理した形で適用した．このアルゴリズムを本論文では PV-MCTS と呼んでいるが，PV-MCTS の実現にあたってニューラルネットワークのデータ構成をユニット選択についてニューラルネットワークの出力から入力に配置して出力層の削減を図り，また PUCT 値に攻撃バイアス項を付け加えることで探索の高速化をしている．ニューラルネットワークにはリカレントネットワークではなく AlphaZero で採用されていた Residual ブロックを 8 段積層した設計としている．この設計では多層の積層を行うことで複雑な状況における表現力の強化を図っている．PV-MCTS アルゴリズムの諸パラメータについての性能評価と対戦評価を行った．対戦評価では独自に作成した対戦用 MCTS アルゴリズムとの対戦で勝率 80% 以上，学習をしていない未知のマップ上でも高い勝率を示した．

本研究では，ターン制戦略ゲームに深層学習を適用する手法を示し，複雑なデータ構造を持ち，将棋や囲碁よりも広大な探索空間を持つゲームに対して，深層学習が適用できることを実証した．深層学習を適用する上での技術的な課題について明確にし解決のための手法を示し，適用したプログラムの性能評価を行った．

目 次

図 目 次

表 目 次

第1章　はじめに

人工知能研究が学問分野として確立したのは，1956 年夏にダートマス大学のキャンパスで開催された会議がきっかけとされている．その会議の参加者達が後に研究のリーダーとしてその後の人工知能研究を牽引していくことになった．

長年，人工知能研究の一分野としてゲームの研究が行われてきたが，その理由としては，ルールが単純で明確に定義されていること，再現が容易なことが多いこと，多くの人にとって身近で評価しやすいこと，特にチェス，囲碁，将棋などは人智の象徴のような意味合いがあったこと，などがあげられる．題材としては多種類のゲームが取り上げられており，上記ボードゲームのほかチェッカーやオセロなどいったものがあった．人工知能研究によってさまざまなアルゴリズムの開発や検討がなされてきており，新しく生まれたアイデアがまたさらに新しいアルゴリズムを生み出したりすることもあった．そして近年は，アルゴリズム研究とデータの蓄積，そしてハードウェアの進歩によって PC によって動作するゲーム用 AI の強さが人間の強さを凌駕する事態が起こるようになってきている．

米 Google 社の子会社である DeepMind 社が開発した AlphaGo は 2015 年 10 月に囲碁のヨーロッパチャンピオンであった Fan Hui に対して 5 勝 0 敗という成績で勝ち越し，19 路盤のハンディキャップなしの勝負でプロ囲碁棋士に勝利した初めての囲碁プログラムとなった．AlphaGo の技術的詳細は 2016 年 1 月に明らかにされたが [1]，この時点ではまだ AlphaGo の強さについて懐疑的な意見もあった．しかし，2016 年 3 月に囲碁の世界トップ棋士である Lee Sedol と対戦し 4 勝 1 敗という成績で勝ち越したことからその性能を疑うことはできなくなった．AlphaGo ではそれまでさまざまな研究などで人類が蓄積してきたニューラルネットワークの技術や強化学習の技術等の過去の技術を巧みに組合せ，さらに膨大な計算を実行することで，きわめて優秀な方策評価関数（policy network）と盤面評価関数（value network）を作り出し，モンテカルロ木探索技術と複合させることで高い性能を実現したのである．

AlphaGo 登場以前の囲碁プログラムの強さについてのおおかたの見解はチェスと将棋ではプログラムの強さがプロプレイヤーをこえたものの，囲碁についてはプロプレイヤーをこえるのはまだ当分先のことであろうと予想されていたことからこれは衝撃的なことであり，深層学習を用いたゲーム研究や機械学習の研究が世界的に活発に行われるようになった．

AlphaGo の後継として 2017 年 10 月に発表された AlphaGo Zero [2] は，自己対戦のみで自らを強くしていく囲碁プログラムである．このバージョンではニューラルネットワークにおける policy network と value network の統合が果たされた．さらに

は同様のアルゴリズムを囲碁だけではなくチェスと将棋にも適用できる一般化アルゴリズムを目指した AlphaGo Zero [3] が 2017 年 12 月に発表された．

AlphaGo に先立って DeepMind 社は 2013 年に強化学習の Q-learning に深層学習を取り入れた Deep Q-Learning (DQN) [4] を発表している．この発表では Arcade Learning Environment (ALE) [5] の Atari 2600 の 7 種類のゲームのうち 6 種類において比較した他の強化学習手法より優れた成績を修めている．こののちに内容的に追加と更新された論文がネイチャー誌に掲載 [6] され，DQN は Atari 2600 の 49 種類のゲームのうち 29 種類のゲームで人間のプレイヤーをこえる優れた成績を示した．

このように，ゲーム研究においてはチェス，将棋そして囲碁などにプロ以上の強さのプログラムを作るためのアルゴリズムや手法が広く知られるようになり，強化学習や深層学習を使用したまざまな研究が行われている．しかしながら，なんらかの難しい特徴を持っているためにまだ研究が進んでいないゲームの分野もいくつかあり，ターン制戦略ゲームと呼ばれる分野もその一つである．

ターン制戦略ゲームはボードゲーム時代から「Tactics」などといった有名なタイトルが存在する歴史と人気のあるゲーム分野であり，PC の発達に伴い多くの戦略ゲームが登場したが，そのようなゲームに内蔵されていたコンピュータプレイヤーはゲームの複雑性などから強いものを設計するのが困難であるため，概して単純な動作しかしないものが多く，人間に比較して弱いレベルであったり，さらにそのためにハンデキャップとして AI 側の駒が多いなど有利な設定がなされていたりした．このような状況から強いプレイができる AI アルゴリズムの開発が望まれていたが，ターン制戦略ゲームに適したアルゴリズムの研究は，探索空間も広大で，また初期画面や盤の大きさが可変であるといった諸条件 [7] により，チェスや将棋などと同様の手法を直接適用するのが困難になっており大幅には進展してこなかった．

このような状況ではあったものの，ゲームアルゴリズムの進歩や研究の進化によってターン制戦略ゲームにおいても人間を満足させるようなプレイができるアルゴリズムを制作する材料はそろいつつある．強化学習と深層学習の技術を応用した新たなアルゴリズムを開発して，ターン制戦略ゲームに適用するという研究は，ゲームのアルゴリズムの進歩の観点でも，世に数多くあるターン制戦略ゲームに使用される AI アルゴリズムの性能を向上させる意味でも，またアルゴリズムをゲーム用途以外で社会的に戦略研究に応用していく上でも意義があることである．

本論文では，ターン制戦略ゲームに深層学習を適用することにより強いプレイのできる AI を実際に作成することで，ターン制戦略ゲームに旧来のアルゴリズムがなぜ適用できなかったのか，また適用できても性能が低かったのか，そして新しいアルゴリズムを適用するために必要な課題とはどのような点になるのか，について明らかにしていく．

まず，第二章では本論文に関連する既存研究やアルゴリズムについて概説する．第三章では使用するアルゴリズムの性能評価を適切に行い比較評価をしやすくするためにベンチマークマップ問題集の提案を行う．そして第四章では，ターン制戦略ゲームの歴史と本論文で使用する TUBSTAP について説明し，しかるのち，第五章では

深層学習をターン制戦略ゲームに適用する上での課題や困難な点を明らかにし，リカレントネットワークを使用した簡潔なデータ表現について提案する．さらに，第六章では既存のアルゴリズムのゲーム記録（棋譜）を使用した学習により policy network の設計に取り組む．そして，第七章では AlphaZero で採用された policy network と value network を統合した形の深層学習を取り込んだ強化学習がターン制戦略ゲームにどのように適用できるかについて論じ，学習パラメータの分析をこころみる．第八章で全体をまとめ研究の展望について述べる．

第2章　既存研究

この章では，本論文にかかわる既存研究について述べる．本研究にかかわる深層学習と AlphaGo/ AlphaGo Zero/AlphaZero に至るまでの過去の既存研究や関連研究は数多いが，深層学習にかかわる研究や強化学習にかかわるもの，さらには将棋や囲碁で使用されていた探索技術や評価関数を制作する手法などについて概説する．

近年は戦略ゲームでもターン制以外のゲームとしてリアルタイム制のもの，チーム制のもの，不完全情報のもの，RPG と組み合わせたものなど多種多様なものについて研究され，強いアルゴリズムを作ることに成功しているものもあるが，ここではターン制のものに関連することに絞る．

2.1　AlphaGo 登場以前の既存研究

2.1.1　Minimax 探索法

Minimax 探索とは，自分と相手が交互に最善手を取るような探索法のことを指す．ゲーム木のように場合分けをしたグラフにおいて，局面の評価値が相手が最小値をかつ，自分が最大値の評価値を取った場合の探索法となる．

Minimax 探索が適用できる条件としてターン制であること，ゼロサムゲームであること，行動が有限であること，二人で行う完全情報ゲームであること，ゲーム結果が確定することが挙げられる．

Minimax 探索は，1928 年の von Neumann によるミニマックス定理の証明から，1949 年の Shannon によるコンピュータチェスに関する論文 [8] で　評価関数に基づいた Minimax 法がチェスに導入されたことから始まったと考えられる．

ある局面の評価値を何らかの手法で求める評価関数の手法によって，Minimax 探索で途中の局面において探索を打ち切り，その局面の評価値を返すことでゲームの終端まで探索する必要がなくなり，探索を高速化することができる．

Minimax 探索法は探索する末端の局面の評価値を全て求めなくてはならないため，動作に時間のかかる欠点があった．それより効率の良い探索方法として経験的にチェスのプログラム用に開発されたのがアルファベータ法であり [9]，これをアルゴリズムとしてまとめたのが Knuth [10] である．Knuth は末端の局面の数が N 個のときに，アルファベータ法は最も効果が高い場合に $\sqrt{N}$ 個だけ評価値を求めればいいことを明らかにした．

アルファベータ法の効果が高くなるのは，展開した探索木が評価関数の値の大きい順番になっているときである．したがって 1 手先を読むたびに，何らかの指標によって評価値を推定して大きい順に並べ替えておくのがよいことになる．将棋で言えば,「駒を取る手」や「王手をする」などと言った重要度の高い手に高い指標を与える．このように指し手を何らかの指標によって順序付けを行うことを move ordering と言う．

2.1.2 Df-pn 探索

Depth First Proof Number (Df-pn）探索 [11] は将棋ソフトにおいては詰将棋を解くプログラムによく使われるアルゴリズムである．その大本の概念として証明数と反証数がある．詰将棋でいうと，直観的には証明数は玉の逃げ方の総数を，反証数は攻め方の王手の総数を表す．この証明数と反証数をしきい値として使った深さ優先探索法が Df-pn 探索となる．証明数と反証数を使って最良優先探索を行うのが PN 探索 [12] である．

Df-pn 探索は AND/OR 木のようなゲームの勝敗がはっきりしていて引分けがないようなゲーム木に適用するのに向いている．主に詰将棋や詰碁の詰め手順の探索 [13] やチェッカーの完全求解 [14] に使用されたりしている．

詰将棋のような詰むか詰まないか，のみを探索すればよい状況ではゲーム木は AND/OR 木に置き換えて考えることができる．詰めようとする側は詰みそうな局面，逃げる側は詰まなそうな局面から優先して探索することで探索の効率を向上させる．この時の詰みそうな指標，詰まなそうな指標となるのが証明数と反証数という数値である．

Df-pn 探索は基本的に AND/OR 木を想定しており，引分けがあるゲームの探索においては勝ちと負けと引分けを分けて探索しなければならないため，二度探索するなどの付加的な手順が必要になる．またゲーム木に合流やループがあるとこれに対応するための処理が必要になる．

2.1.3 実現確率探索

AlphaGo 以前に確率を使用した探索として実現確率探索 [15] がある．実現確率探索は将棋の研究において導入された手法であり，局面の実現確率に基づく探索範囲の制御方式である．実現確率は次のような式で再帰的に表される．

$$P_x = P_m \cdot P_{x\prime} \tag{2.1}$$

ここで P_x はノード x の実現確率，$P_{x\prime}$ は親ノード $x\prime$ の実現確率，P_m は位置 $x\prime$ から x へ移動する際の指し手 m による遷移確率である．遷移確率は親局面からその局面に遷移する確率であるといえる．この式により探索ノードの実現確率を求め，局面

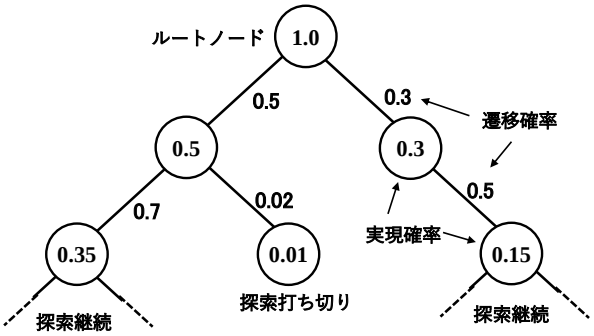


図 2.1: 実現確率探索

の実現確率があるしきい値を下回ったら探索を打ち切ることにより実現確率の高い手だけを深く読むようにする．通常のアルファベータ探索が探索の深さで探索を打ち切るのに比較してこの点が異なった探索打ち切り手法である．

実現確率探索についての概念図を図 2.1 に示す．この探索木ではノードが各局面，エッジが各局面に対する指し手を，ノードに記入された数値が各ノードの実現確率，エッジに記入された数値がノード間の遷移確率をそれぞれ表している．あるノードの実現確率は親ノードの実現確率に遷移確率をかけた値になる．例えば，0.5 のノードから 0.02 の遷移確率の指し手によって遷移するノードは $0.5 \times 0.02 = 0.01$ となる．ここで探索打ち切りの設定が実現確率が 0.01 以下であった場合，このノードの探索は打ち切りとなる．実現確率が 0.01 より大きなノードはさらに探索を継続していく．

将棋の探索ではアルファベータ探索が多く使われていたが，探索の打ち切りの制御は主に探索の木の深さで行なわれていた．この方式では現実の将棋では実現しないような局面も，重要な局面も同等に探索してしまうことで探索の資源を消費することになる．ここで実現確率探索を導入することによって指される確率の高い手を優先的に探索ができるようになり，より効率的な探索ができるようになった．このアルゴリズムを採用した将棋アルゴリズムである激指は 2005 年の第 15 回コンピュータ将棋選手権で優勝している．

しかし実現確率探索のためには遷移確率について事前に計算しておく必要がある．当初は特定の駒の動き,「駒得でとりかえす」といった指し手のカテゴリを将棋プロ棋士の棋譜約 600 局から得られた統計データが利用された [16]．やがてロジステック回帰モデル確率モデルによる指し手のさまざまな属性や特徴量を用いて棋譜 16000 局の将棋プロ棋士の棋譜データから遷移確率を得るようになった．

遷移確率を計算する重みの学習の特徴量の策定は人間が行い，特徴量設計の自動化はまだされていない．このように確率を利用して効率的に良い手を選択していく概念は AlphaGo が発表される以前より存在していたといえる．

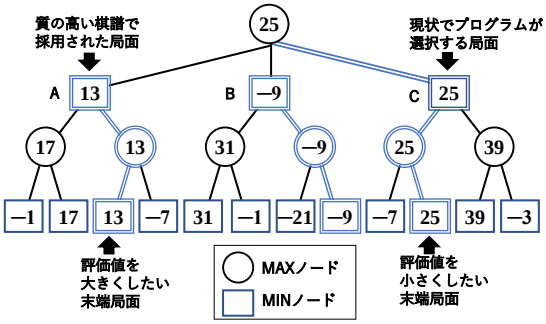


図 2.2: 棋譜を用いた機械学習

2.1.4 盤面評価関数の機械学習

将棋では盤面評価関数の自動的な学習を行う試みは TD(λ) 法 [17] などがあったがその性能は高くなかったため，Bonanza 登場以前には将棋での盤面評価関数の自動調整は困難と考えられていた [18]．将棋以外で機械学習が成功していた例として，オセロ [19] とバックギャモン [20] があった．

しかし 2006 年に登場した Bonanza [21] は将棋において盤面評価関数の機械学習を実現し世界コンピュータ将棋選手権で初出場で初優勝を果たし，このことにより将棋でも機械学習による盤面評価関数の調整が可能であること，そしてその有効性が示された．

Bonanza における自動学習について図 2.2 で説明する [16]．図において将棋の現局面に三つの指し手 A, B, C がありそれぞれ評価値が $13, -9, 25$ であり，質の高い棋譜において採用された指し手が A であったとする．この時，アルファベータ探索アルゴリズムの動作であれば最も評価値の高い指し手 C が選択されることになる．Bonanza の自動学習は条件において Minimax 探索結果の最適制御による特徴ベクトルの学習によって A の評価値を上げるか C の評価値を下げることで指し手 A が指されるような制御を行う．結果的にこれは棋譜の指し手を教師データとする教師付き学習と同じことになる．

盤面評価関数の設計法は将棋の全 40 枚の駒から王を含む 3 駒の組合せを網羅的に選びそれぞれの組合せの持つ得点をすべて加えて評価結果とする．各組合せの得点はプロ棋士の棋譜を教師データとする教師付き学習で調整する．これらの組合せの評価の項目は 5 千万個という膨大な数になっている．

Bonanza で設計された盤面評価関数はそれまでの手調整であった盤面評価関数に比較して優秀な性能を示したものの欠点も存在する．ひとつは教師付き機械学習であるため，ある程度大きなサイズのかつ良質なデータ，主にプロ棋士の棋譜が必要であること，入玉のような棋譜に現れにくい状況は学習が困難であること，局面の優劣認識の性能において元の棋譜のレベルを大きくこえることが困難であること，などがあげられる．

学習に用いる棋譜は開発当初では将棋プロ棋士の公式戦 3 万局と将棋倶楽部の棋譜

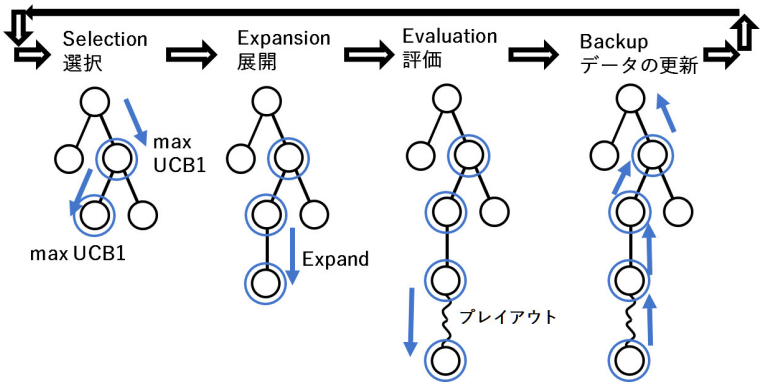


図 2.3: MCTS の基本動作

集から 3 万局の計 6 万局を使用していたが，後のバージョンでは将棋倶楽部の棋譜は使用されなくなった [16]．これは当初の開発版では入玉形の学習をするために将棋倶楽部の棋譜を使用したものの，最終的にはこのようなことをしない方が強くなるためである．このように学習に使用するデータとその質が学習結果に密接に関連していることがうかがわれる．

このように大量の特徴量を学習することで精密な盤面評価関数を設計することは AlphaGo 以前にも行われていたが特徴量の設計はまだ人間が行っていた．

2.1.5 MCTS と囲碁アルゴリズム

囲碁は合法手の数が多いために探索空間が巨大になってしまい深く探索するのが難しいだけではなく，チェスや将棋のように精度の高い盤面評価関数を設計するのが困難であった [22]．精密な盤面評価関数を設計するのが困難であった理由としては，チェスや将棋には駒に個性があり持ち駒の価値などの手がかりに比較的容易に評価関数を作ることができるが，囲碁の場合は駒に個性がなく評価関数を作るための手がかりが乏しいこと，領域の広さによる評価も単純にどちらが有利という判断がしにくい，局所的な最善手が全局的な最善手になりにくい，といったことが挙げられる．

このため将棋やチェスで成功した手法であるアルファベータ探索を適用して強いプログラムにすることができなかった．そのようななかで囲碁に適したアルゴリズムとして注目されたのがモンテカルロ木探索（MCTS）であった．

モンテカルロ木探索の原型といえる原始モンテカルロ法のアイデアは 1993 年の Brugmann にさかのぼるとされており [22]，これにさらに木探索を組み合わせたものが 2006 年の第 11 回 Computer Olympiad の囲碁 9 路盤部門で優勝した CrazyStone とその後に登場した MoGo である．これが MCTS の始まりであると言われている．モンテカルロ木探索の基本アルゴリズムは，Selection（選択） / Expansion（展開） / Evaluation（評価）/ Backup（データ更新）の四つのステップから成っている（図 2.3）．Selection で現局面からの枝を選択し，Expansion で枝の展開を行い，Evaluation では

プレイアウトによるランダムシミュレーションを行って，Backup で各ノードの諸データを更新する．木探索におけるどの枝を選択するかという基準として Multi-Armed Bandit 問題に対する一つの提案である UCB1 アルゴリズム [23] は次式による指標を用いる．

$$UCB1 = \overline{x_j} + C \cdot \sqrt{\frac{2 \log n}{n_j}} \tag{2.2}$$

ここで $\overline{x_j}$ はノード j における平均勝率，C は定数，n は総訪問回数， n_j はノード j の訪問回数である．この式により探索（Exploration）と利用（Exploitation）のトレードオフのバランスを取ることを目指している．UCB1 式を改良した別の式を用いる場合もある．

2000 年後半に MCTS が登場して囲碁プログラムは一挙に強くなり，アマチュア有段者レベルからアマチュア高段者レベルになった．しかしながら技術の進歩は徐々に停滞し囲碁プログラムのレベルの向上は鈍化してきた．2013 年には第一回電聖戦で Crazy Stone がプロ棋士を相手に 4 子のハンディキャップで勝利したが，囲碁のプログラムがプロ囲碁棋士にハンディキャップなしで勝利するのはまだ当分先のことであろうと考えられていた．

2.1.6 深層学習

AlphaGo の飛躍的な性能向上を支えた要素技術として深層学習（ディープラーニング： Deep Learning）がある．深層学習は多層のニューラルネットワークを用いた機械学習の方法論である．ニューラルネットワークの研究は過去に何度かブームになったが下火になっていた所で HinTon らの Deep Belief Network (DBN) [24] の研究によって注目されるようになった [25]．その後，音声認識や画像認識のベンチマークテストで過去の記録が塗りかえられ，多層ニューラルネットワークの有効性が確かめられたことで深層学習が広く研究されるようになった．深層学習は音声や画像といった高次元空間に存在する複雑なデータから特徴量を抽出して認識することに優れていると言われている．

従来は 4 層をこえる多層ニューラルネットワークは過学習や勾配消失問題などがあり性能を発揮できなかったが，研究の深化によってこれらの問題を解決するための手法が多く提案されてきた．深層学習で使用される主な手法として活性化関数として使用される ReLU [26] ，ランダムに任意のニューロンを無視して計算することで過学習を防止する dropout [27] ，バッチ正則化，データ拡張（Data Augmentation） [28]，

規模の大きなニューラルネットで効率的に学習を進めるためのミニバッチ法など他にも多数の手法がある．

深層学習で使用されるネットワークの種類としては，順伝播型ネットワーク（Feed-forward Neural Network），畳込みニューラルネットワーク（Convolutional Neural Network: CNN），再帰型ニューラルネットワーク（Recurrent Neural Network: RNN），

などがある．RNN の中には自然言語処理によく使われる Long Short-Term Memory(LSTM) がある．

2.2 深層強化学習

ここでは深層学習を強化学習に取り入れた新しい手法についてまとめる．

2.2.1 Deep Q-Network

Deep Q-Network (DQN) [4, 6] は DeepMind 社が提案した手法であり，深層学習と強化学習（Q 学習）を組み合わせることで，ベンチマークに使用した Atari 2600 のゲーム群において優れた性能を示した．DQN の特徴として次が挙げられる．

- Q 学習の Q 関数をディープニューラルネットワーク（DNN）で実装した．DNN にはゲームの画面で 80 ピクセル × 80 ピクセル × 4 フレーム の画像を入力し，出力はゲームの操作信号（ジョイスティックの方向とボタン垂下情報）と単純な構成として，DNN の画像認識能力を活用した．
- Experience Replay の手法を採用して学習中の行動結果を一定サイズのリプレイメモリに保存しておき，DNN の学習時にランダムサンプリングしてミニバッチに投入することでデータの相関関係を取り除き過学習を避ける工夫をした．
- 不要な振動現象を避け学習の収束を速めるため Q-network の固定を行い，定期的な更新をするようにした．
- ゲーム内容によらない学習を進めるために報酬を正なら +1，なしなら 0，負なら −1 に固定してクリッピングを行った．これによりどのゲームでも同一のパラメータを使用できるようにした．
- 探索と利用のトレードオフ（Exploration-Exploitation trade-off）のため ε-Greedy 法を使用している．

DQN が報告されてから深層学習と強化学習を適用したゲーム研究が多く行われるようになったが，DQN には課題もあった．DQN において高い得点を取ることができなかったゲームにはこの枠組みではうまく扱うことができない疎な報酬の仕組みのゲームがあることである．現在ではこれらのゲームにおいて高い性能を目指す強化学習の研究がさかんとなっている．また Atari 2600 の枠組みでは出力がゲーム画面で入力がジョイスティックとボタン一個のみであった．これはニューラルネットワークからの観点では画面の画像が入力でジョイスティックとボタンは出力側になる．ニューラルネットワークへの画像入力は 84 ピクセル × 84 ピクセル × 4 フレーム と複雑であるものの，ニューラルネットワークに必要な出力はジョイスティックの方向が 8 方

向にボタンの垂下情報を合わせて $8 \times 2 = 16$ の出力となるので，比較的簡単な出力構造となっていた．

つまり DQN が発表された段階では複雑なデータ出力を必要とするゲームはまだ取り扱っておらず，基本的にはアーケードゲームのようなシングルプレイヤーのゲームを想定しており，どちらかというとシングルエージェント環境向けで，二人プレイするゲームに適用するにはエージェントを二つ用意する必要があった．

2.2.2 AlphaGo

ディープマインド社の開発した AlphaGo [1] の技術的な要素技術は個々の歴史を見ると新しいとはいえない場合が多いが，これらを巧みに組み合わせて膨大な計算資源を投入することで囲碁のプログラムの強さを一気にプロ棋士を越えるレベルに押し上げた．

AlphaGo で使用されている技術としては大きく分けて深層学習と強化学習，そして探索の部分に分けられる．

深層学習の部分は，任意の局面における手の予測を行う予測器であるポリシーネットワークと，局面の勝率を高速で出力できるバリューネットワークの二つをニューラルネットワーク技術を用いて実現している．

ポリシーネットワークの作成手順はまず，KGS 囲碁サーバーから取り出した囲碁対局データからの約 3000 万局面を教師データとして学習した Supervised Learning (SL) policy network を作りあげる．さらにそれを自己対戦によって学習データを作成し，これによって Reinforcemnet Learning (RL) policy network を訓練する．バリューネットワークの作成は SL policy network と RL policy network を利用したシミュレーションによる学習を行うことで作成し，MCTS でよく使用されるロールアウトの評価値との加算によって最終的な局面の value 値としていた．

探索プログラムは以前から囲碁に適用されていた MCTS をベースとしているものの独自の拡張が為された Asynchronous Policy and Value MCTS (APV-MCTS) アルゴリズムと呼ばれているものである．基本的な動作は　Selection（選択）/ Expansion（展開）/ Evaluation（評価）/ Backup（データ更新） の動作から成り広く使われている MCTS と同一であるが，シミュレーションにおける次の一手の選択には policy network の確率出力による補正のある確率探索を行い，また選択するノードの指標としては，他の MCTS でよく使われている UCB1 式ではなく，次のような独自の PUCT 式を使用している．

$$PUCT(s,a) = Q(s,a) + Cpuct \cdot P(s,a)\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \tag{2.3}$$

ここで状態 s ，行動 a に対して $Q(s,a)$ はノードの勝利確率，$Cpuct$ は探索補正係数，$P(s,a)$ は各ノードの選択確率，$N(s,a)$ はノードの訪問回数，を表している．PUCT 式は UCB1 式と第一項の勝率に関しては同じであるが，第二項のバイアス項に

関する部分が異なっており，ここにPUCT式では選択確率が乗じられており，UCB1式では総訪問回数の対数をとったものを使用していたが簡単化されている．また，分母に+1があるので訪問回数がゼロの場合でも分数が発散しない．この選択確率を使用することによりある局面でのより有望と思えるノードを優先的に探索するアルゴリズムになっている．最終的には訪問回数の最も多いノードが選択され次の一手が決定する．

AlphaGoによっていままで特徴量の策定や設計について人間が行ってきた部分が深層学習によってこの部分も自動化された．つまり，特徴量の抽出と策定に関する労力は不要となったかわりにどのような特徴量が使用されているのか不明確になってしまい，アルゴリズム内部のブラックボックス化が進行した．

2.2.3 AlphaGo Zero と AlphaZero

AlphaGoに続いて発表されたAlphaGo Zero [2] では多くの改良点が加えられた．さらにAlphaZero [3] ではチェスと将棋への拡張が為された．AlphaGo Zeroでは人間の棋譜から学習することを止め，自己対戦によるデータ生成のみで自らを強くしていく方式をとるようになり，また分離していたpolicy networkとvalue networkが一つに統合された．value networkの性能向上によりロールアウトが不要となり廃止され，また以前強化学習の中で行っていたニューラルネットワーク同士のトーナメントによるニューラルネットワークの交代を止め，常に同一のニューラルネットワークを訓練するように変更された．

またAlphaGo Zeroになってから温度減衰付きボルツマン確率探索が採用され初手から30手程度まで指し手の選択にランダム性が加えられている．これは，ロールアウトが廃止されてランダム性がなくなり，このままでは同一の局面のみ探索することになってしまうことや，自己対戦に特有の局所解に落ち込むような挙動を避けるために導入されたものである．さらには，MCTSシミュレーション中には根ノードにDirichletノイズがpolicy出力に付加され，広い範囲の手を探索する工夫が為されている．

ここでAlphaGoの各バージョンと性能についてまとめたものが表2.1である．AlphaGo Fanは最初のバージョンでありネイチャー誌に掲載された内容であるが訓練時間が明確に記載されていないがこれは各ネットーワークが個別に訓練されたためため簡単に記述できないためと思われる．またAlphaGo LeeとAlphaGo Masterについては公式な論文が存在せず訓練時間も公表されていないようである．また，AlphaZeroについては論文にグラフの形でレーティング値が示されているが数値で示されていなかったため記載しなかった．しかし，AlphaZero GoはAlphaGo Zeroに対して勝率61 %を示しているのでレーティングは明らかにAlphaGo Zeroより上であるといえるだろう．

この表のレーティングの性能の伸びからAlphaGoの性能が飛躍的に向上したことがわかる．また，計算のために使用したTPU [29] は一台あたりの性能がきわめて高

表 2.1: AlphaGo のバージョンと性能

バージョン	対戦相手と 試合結果	計算資源	訓練時間	Elo レーティング
AlphaGo Fan	Fan Hui 5 勝 0 敗	CPU1202 GPU176	-	3144
AlphaGo Lee	Lee Sedol 4 勝 1 敗	TPU48	-	3739
AlphaGo Master	Ke Jie 3 勝 0 敗	TPUv2 4	-	4858
AlphaGo Zero	AlphaGo Master 89 勝 11 敗	TPUv2 4	40 日	5185
AlphaZero Chess	Stockfish 155 勝 6 敗 839 分	TPU5000	9 時間	-
AlphaZero Shogi	Elmo 勝率 98.7%	TPU5000	12 時間	-
AlphaZero GO	AlphaGo Zero 勝率 61%	TPU5000	3 日	-

表 2.2: AalphaGo ネットワークの設計の概略

バージョン	ネットワーク構成
AlphaGo	◯SL/RL ポリシーネットワーク　全 13 層 (CNN + ReLU)×12 + Softmax ◯バリューネットワーク 全 15 層 (CNN + ReLU)×13 + FC(1) + tanh
AlphaGo Zero	全 40 層以上 Residual 19 段 ◯ポリシー出力 CNN + ReLU + FC(362) ◯バリュー出力 CNN + ReLU + FC(256) + ReLU + FC(1) + tanh

く訓練時間だけでは計算量の推定が困難であるが，TPU の性能は通常の GPU の数倍から数十倍であることから推測すると膨大な計算量であることは明確である．

AlphaGo と AlphaGo Zero のニューラルネットワークの構成を比較したのが表 2.2 である．AlphaZero は AlphaGo Zero とほぼ同一であるので省略した．AlphaGo では SL ポリシーネットワークと RL ポリシーネットワークの設計は同一であるものの個別のネットワークとして用意されていた．しかし，AlphaGo Zero となってからバリューネットワークとともにこれらは一つに統合され Residual ネットワークを採用されたものになった．Residual ネットワークの採用が性能の向上に貢献していることが予想される．

AlphaGo のニューラルネットワークの出力段の設計について検討するためにまとめたものが表 2.3 である．バリューネットワークの出力が tanh 関数を活性化関数とす

表 2.3: AalphaGo ニューラルネットワークの出力構成

バージョン	出力構成
AlphaGo	◯ SL/RL ポリシーネットワーク　$19 \times 19 = 361$ （盤面位置） ◯バリューネットワーク 1　(tanh)
AlphaGo Zero AlphaZero GO	◯ポリシーネットワーク $19 \times 19 + 1 = 362$ （盤面位置 + パス） ◯バリューネットワーク　1 (tanh)
AlphaGo Chess	◯ポリシーネットワーク $8 \times 8 \times 73 = 4672$ （盤面位置 × 層数） ◯バリューネットワーク　1 (tanh)
AlphaGo Shogi	◯ポリシーネットワーク $9 \times 9 \times 139 = 11259$ （盤面位置 × 層数） ◯バリューネットワーク　1 (tanh)

るスカラー値の単一出力であることは各バージョンで共通である．AlphaGo ではポリシーネットワークの出力は囲碁の盤面を表現する確率出力のみであったが，AlphaGo Zero となってから盤面の確率出力にパスの出力が加わった．これは MCTS シミュレーション中にパスの手順を表現することをできるようにするためと思われる．シミュレーション中の手順でパスが二回連続すると囲碁のゲームは終了することから，より高速でゲームの終了を検出するためにパス出力の追加が行われたと推測される．

AlphaGo Chess では出力数が大幅に増加している．これはチェスの盤面上の駒の表現でクイーンタイプの移動を表現するために 56 層，キングタイプの移動を表現するために 8 層，成りを表現するために 9 層でトータル 73 層必要なため，結局 $8 \times 8 \times 73 = 4672$ の出力が必要となっているためであろう．

AlphaGo Shogi ではさらに出力数が増加し 11259 となっている．これは将棋の盤面上 (9×9) で駒の移動を表現するために，駒の移動の上下左右と右上と左上と右下と左下の 8 方向を 8 種類の駒（王，金，銀，桂馬，香車，飛車，角，歩）で 8×8 の 64 層，さらに桂馬飛びの方向を表現するために 2 層，駒の移動と駒の成りを表現するために上と同様に，上下左右と右上，左上，右下，左下の 8 方向に 8 種類の駒で 8×8 の 64 層，さらに桂馬が成る場合の二層，そして持ち駒を盤上に打つことを表現するために王以外の 7 種類の駒を表現する 7 層でトータル 139 層必要となる．結局必要な出力は $9 \times 9 \times 139 = 11259$ となると思われる．

このように扱うゲームの種類や出力行動の複雑さによって，設計するニューロンの出力は大幅に増加することがわかる．AlphaZero の方式であればチェスや将棋でも複数の確率出力を表現することができるが，同時に複数の指し手の動作を表現することを想定した設計となるため盤面のサイズ，駒の種類や行動の種類が増加するにしたがって必要な出力数は増加していく．

このようにAlphaGo Zeroの構成においてはチェスや将棋のような複雑な駒の動きや盤面の表現ができるようなニューラルネットワークの出力設計となっていると言える．

2.3 AlphaGo 以降の状況

2019年4月，OpenAIが開発したAI「OpenAI Five」が，対戦型リアルタイムストラテジーゲーム「Dota 2」の2018年度世界大会覇者に勝利を収めた [30]．OpenAI FiveはDota 2が5人でチームを組んで戦うマルチプレイヤーシステムのため5つのニューラルネットワークになっている．

2019年10月DeepMind社は同社の開発したAlphaStarがリアルタイムストラテジーゲーム「StarCraft II」において人間プレイヤーの上位0.2％にランクインしたとが発表 [31] した．StarCraft IIはきわめて複雑な不完全情報ゲームであり，AlphaStarはマルチエージェントシステムで大型のニューラルネットワークを組み合わせて動作する仕組みとなっておりAlphaGoのシステムとは単純に比較ができない．

OpenAI FiveとAlphaStarではRNNの一種であるLSTMが多数使用されている．

2.4 強化学習の展開

ゲーム研究として深層学習を利用した強化学習のアルゴリズムの研究はAlphaGo以降にも非常に活発に行われている．

TD誤差学習の手法の一つであるactor-critic法をベースとしながら，エージェントを複数で並列に動作させることでサンプリング効率を高めデータの相関関係を解消し，1ステップ先の報酬だけではなく数ステップ先の報酬まで使用して学習を進めるA3C [32] では学習の速度が向上した．

PPO [33] はA3CとTRPO [34] の手法を受け継ぎつつ目的関数にクリッピングを採用することで，方策が大きく変化することを防止して，実装を容易にしつつAtari 2600での性能も向上させたものである．Dota2ゲームにおいて人間プレイヤーに勝利したAI「OpenAI Five」はPPOアルゴリズムを採用している．

優先度付き経験再生を分散処理で高速化したアルゴリズムApe-X [35] はAtari 2600において競合アルゴリズムに対して優秀な性能を示した．

さらにはApe-Xの技術をベースとしてRNNの一種であるLSTMをExperience Replayに使用する技術を確立したR2D2 [36] は，より良い時系列データを扱うことができるようになりApe-Xの成績をAtari 2600上で大幅に上回った．

第3章　ターン制戦略ゲームと TUBSTAP

この章ではターン制戦略ゲームについて概説しさらに TUBSTAP フレームワークについて説明する．

3.1　ターン制戦略ゲーム

ここではターン制戦略ゲームをウォーシミュレーションゲームと読み替えてその歴史と分類，そして社会的な活用について紹介する．

3.1.1　ターン制戦略ゲームの歴史

ターン制戦略ゲームおよびウォーシミュレーションゲームの歴史について述べる．

現代的なウォー・シミュレーションゲームの先祖にあたるものはボードゲーム等の伝統的な「戦場を模したゲーム」である．ターン制であること，戦略があることをウォーシミュレーションゲームの条件にするとチェス，将棋，囲碁などの古典ボードゲームも入ることになる．広義のウォーシミュレーションという意味ではこれらを含めて考える場合もあるが，ここでは狭義のウォーシミュレーションゲームの定義として具体的な戦争や戦闘を模擬したもので近代のボードゲームに近いものを限定して考える．

近代におけるウォーシミュレーションゲームの嚆矢は，陸においてはプロイセン陸軍で用いられたクリークシュピール，海においてはイギリス海軍で用いられたジェーン氏のネイヴァルウォーゲームとされる．それは来るべき戦いに備えた教育，訓練のツールであると同時に，新たな作戦・戦術の検証ツールであった [37]．日本海軍でも，秋山真之の提案でシミュレーションウォーゲームを取り入れ，日露戦争前，海軍大学校で繰り返し行った逸話は有名である．

古くからウォーゲームは存在した [38] が．娯楽ゲームとして広まるのは 1950 年代以降である．第二次世界大戦後の 1954 年，アメリカのチャールズ・S・ロバーツが，ミニチュア・ウォーゲームや兵棋演習の戦略的な部分のみに着目して，マス目が印刷された紙製の地図と，厚紙で作られた駒を使う自作のボードゲーム「Tactics」 [39] を自費出版で販売開始した．このゲームは彼我の戦力比によって戦闘結果が多様に変

化するという概念を初めてホビー用のゲームに持ち込んだんものでもあり，ウォーシミュレーションゲームと呼ばれるジャンルの元祖ともされており，RED と BLUE に分かれて架空の大陸で戦うものであった．

Tactics は現在のボード・ウォーゲームでも標準的に使われる多くのシステムの先駆者となった．例えば以下のようなものが挙げられる．

- 個々の戦闘部隊をあらわす厚紙の駒，移動力と戦闘力を表示
- オッズ比による戦闘結果表
- いろいろな地形をあらわすマス目で表現されるマップとマス目毎に異なる移動コスト

ただし，マップは，後のボード・ウォーゲームで一般的となる六角形の升目（ヘクス）ではなく，正方形の升目（スクエア）であった．その後，多数のウォーシミュレーションゲームが開発されファンを増やしていった．

1970 年代後半から登場した家庭用コンピュータによりビデオゲームの時代が始まり，ウォーシミュレーションゲームもボードゲームから移植されるようになった．PC の普及によって以前は数表を見ながら計算していた攻撃による損害などの面倒な計算や数値の処理をコンピュータにまかせることができるようになって労力は軽減しまた，ルール上の確認しながら行う操作もプログラムが自動で行ってくれるようになったことでゲームの進行は飛躍的にスムーズになった．

「信長の野望シリーズ」は 1983 年に発売された，日本の戦国時代をテーマとする歴史ウォーシミュレーションゲームで，戦闘における戦術よりも戦略や内政に重きをおくゲーム構成となっている．

「ファミコンウォーズ」は 1988 年に任天堂ファミコン用に発売されたウォーシミュレーションゲームで将棋のように整理・簡単化された兵種とマップでランダム要素を排した初心者向けを意識した内容であった．ファミコンウォーズは人気シリーズとなり，多数の後継作品を生み出した．

「大戦略シリーズ」は 1988 年の現代大戦略から始まるウォーシミュレーションゲームシリーズで，六角形のマス（ヘックス）の採用，細かい兵器の性能の再現や，生産システムの採用，史実を再現したマップなどボードゲームのウォーシミュレーションに近い機能が盛り込まれた本格的なものであった．このシリーズは国内売り上げ 80 万本をこえたヒットシリーズとなった．

ウォーゲームを戦術レベルか戦略レベルの規模によって分類することがあるが，この場合は信長の野望は戦略レベルでファミコンウォーズや大戦略は戦術レベルであると言える．

もちろんビデオゲームが勃興していくなかでターン制戦略ゲーム以外の分野のゲームもさまざまなものが開発され人気を獲得し，発展していった．大戦略などのゲームの流れとしては，RPG と組み合わせた「ファイヤーエンブレム」，リアルタイム制

をとった「StarCraft」，文明を発展させていく「Civilization」のように複雑な構成のものなど，多種多様のものが存在している．

3.1.2　ターン制戦略ゲームの活用

ターン制戦略ゲームは，戦史研究や戦術の研究，戦争の模擬という目的から出発したものが，すぐに娯楽のためのゲームへと変化していった．

そして，ゲーム一般についても社会活動への応用が活発になってきているが，その中にもターン制戦略ゲームの要素を持つものもある．明示的に戦略という用語を使用しなくても内部的には結果的に戦略を扱うものは戦略ゲームの延長線上にあるゲームといえる．

「シヴィライゼーション」(Civilization) は，文明をモチーフとしたターン制のシミュレーションゲーム（ストラテジーゲーム）シリーズであるが，教育用向けバージョンはアメリカの高校などで教育用に使用されている [40]．

またシリアスゲームと呼ばれる新しいゲーム分野は社会的用途でゲームを開発・利用する取り組みといえる．ターン戦略ゲームも経営シミュレーションのようなシリアスゲームへの応用が考えられる．

人工知能研究においてはゲームは当初からその研究のしやすさや知能を研究するための題材として有用であった．その中でもターン制戦略ゲームの持っているいろいろな要素（不完全情報性，協力，毎回違う環境，行動の多層性）は現代社会に頻繁に登場するものであって，取り組むべき課題を模擬して研究していくのに都合のよい性質を持っており研究をしていく意義は大きい．

3.2　ターン制戦略ゲーム研究用プラットフォーム: TUBSTAP

TUBSTAP は Turn-Based Strategy Games as an Academic Platform の略称である．ここでは本論文で題材として使用するターン制戦略ゲーム研究用フレームワークである TUBSTAP について説明する．

3.2.1　TUBSTAP とは

ターン制戦略ゲームは歴史が長く市場で人気もありながらも，学術的な研究としてはゲームが持つ複雑性のためもあり活発に行われていなかった．そのようななか「大戦略」や「ファミコンウォーズ」といった既存のターン制戦略ゲームを分析しつつ，ターン制戦略ゲームに向けた学術研究用基盤プラットフォームとして使用できる「ターン制戦略ゲーム　学術用基盤プロジェクト： TUBSTAP 」 [41] が提唱された．

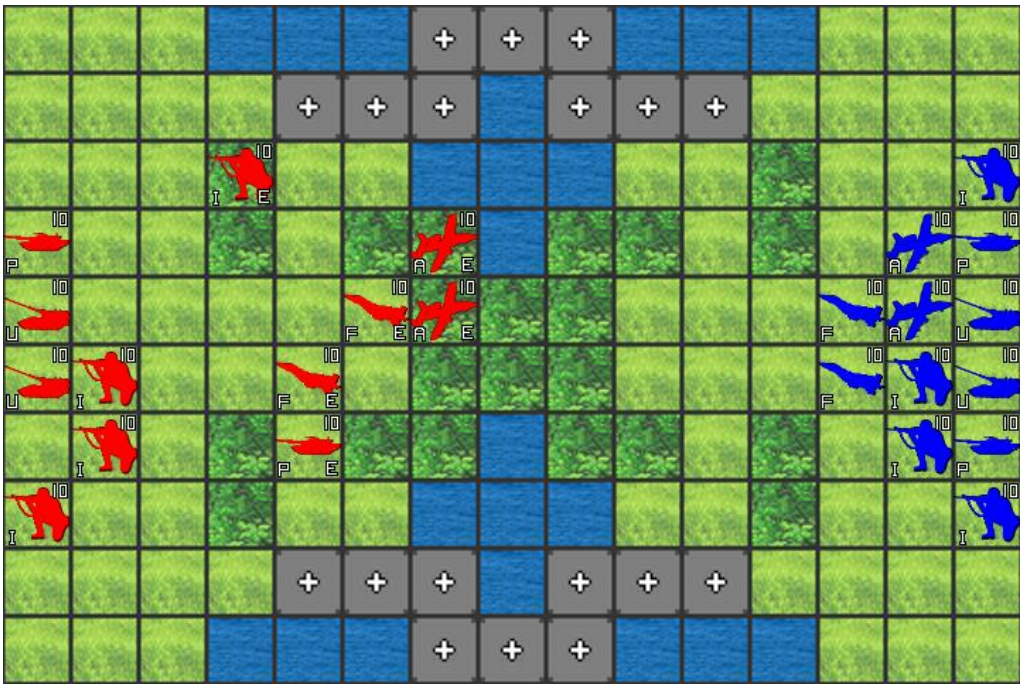


図 3.1: TUBSTAP のマップ

分析のために使用されたゲームタイトルは市場で有名なもの 17 本であり，主なものとして「ファミコンウォーズ DS2」「大戦略」「スーパーロボット大戦」「Arimaa」「StarCraft2」「チェス」「将棋」「軍人将棋」などがある．これらの中から特徴量を分析し最も共通するルールを抽出して，既存ゲームとの親和性を踏まえ洗練させた形でルールを定めたものが TUBSTAP である．TUBSTAP は上記のなかで「ファミコンウォーズ DS2」に最も類似している．ルールは，次節で詳細に説明する．

TUBSTAP は戦術級シミュレーションか戦略級シミュレーションかの分類上は戦術級に分類されるが便宜上，戦略ゲームとしている．またリアルタイム性はなく，ロールプレイングゲームのような成長部分やランダム性は含まれない．この意味ではチェスや将棋のような二人確定完全情報ゲームである．

3.2.2 TUBSTAP のルール

TUBSTAP で提案された基本的なルールは主に次のようなものである．

- 盤面は四角形のマス目状のマップでサイズ設定は自由
- 二人プレイ方式
- 駒の種類制限
- 1 ターンで複数ユニットの動作

(a) 歩兵　(b) 戦車　(c) 自走砲　(d) 対空戦車　(e) 戦闘機　(f) 攻撃機

図 3.2: TUBSTAP の駒の種類

(a) 草原　(b) 進入禁止　(c) 森　(d) 山　(e) 海　(f) 道路　(g) 陣地

図 3.3: TUBSTAP の地形の種類

- HP システム有り
- 攻撃と反撃有り
- 地形と移動コスト有り
- 移動の際は味方ユニットはすり抜けるが敵ユニットはすり抜けられない

TUBSTAP の初期画面はマップごとに異なり様々なものがあるが一例としてマップを示す（図 3.1）.

TUBSTAP で用意している駒は図 3.2 に示すように，歩兵（I），戦車（P），自走砲（U），対空戦車（R），戦闘機（F），攻撃機（A）の 6 種類となっており，多くのターン制戦略ゲームと同じように駒をユニットと呼ぶことが多い．そして用意されている地形は図 3.3 に示すように草原，進入禁止，森，山，海，道路，陣地の 7 種類となっている．駒や地形はできる限り簡略化し種類を最小化している．これは研究用としてできる限りゲーム条件を単純化し必要最小限の要素にしぼっているためである．

ゲームは各ターンごとに進行し RED 軍と BLUE 軍の交代で各駒を動かす．ターンの定義としてはゲームによって異なる場合があるが TUBSTAP においては，相手の行動が全て終了して自分側の手番となってから，全ての自軍の駒を移動させるまでを一ターンとしている．ゲームは基本的に RED 側が先攻で開始する．

各駒は表 3.1 にあるように種類に応じてそれぞれ固有の移動力を持っており移動の際には移動コストを消費しながら移動する．例として歩兵は移動力 3 であるので，移動コスト 1 のマスの平原や道路は 3 マス移動できるが，山のマスを通過するには 2 の移動力を消費し，かつ海には入れない．移動力を消費して 0 になったらそれ以上の移動はできない．

各駒は攻撃する際には必ず攻撃する相手の駒に隣接マスに移動してから攻撃する．例外は自走砲ユニットで自走砲の攻撃は隣接するマスにはできず，距離 2 か 3 のマス

表 3.1: TUBSTAP 攻撃力と移動コスト

ユニット	攻撃力						移動力	移動コスト					
	F	A	P	U	R	I		道路	平原	森	山	海	陣地
戦闘機 F	55	65	0	0	0	0	9	1	1	1	1	1	1
攻撃機 A	0	0	105	105	85	115	7	1	1	1	1	1	1
戦車 P	0	0	55	70	75	75	6	1	1	2	-	-	1
自走砲 U	0	0	60	75	65	90	5	1	1	2	-	-	1
対空戦車 R	70	70	15	50	45	115	6	1	1	2	-	-	1
歩兵 I	0	0	5	10	3	55	3	1	1	1	2	-	1

のみとなる．これは自走砲は大砲を模擬しているもので間接攻撃ユニットであるためである．

各軍で攻撃することにより相手の駒の HP をダメージを与えて減らすことができるが同時に反撃を受けて自分の駒の HP もダメージにより減少する．駒の HP がゼロになったらその駒はマップから取り除かれる．

攻撃の際のダメージは次の式で計算される．

$$\text{ダメージ} = \frac{\text{攻撃力} \times \text{攻撃ユニット } HP + 70}{100 + \text{地形効果} \times \text{防御ユニット } HP} \tag{3.1}$$

ここで攻撃力は表 3.1 で示されるように駒の相性に応じて種類ごとにあらかじめ決められており，また，地上ユニット（歩兵，戦車，自走砲，対空戦車）は，平原で 1，林で 3，山と陣地で 4 の地形効果を受ける．戦闘機と攻撃機は航空ユニットであるため地形効果を受けない．

ゲームの終了条件

ゲームの終了条件はマップで規定されたターン数に達するか，どちらかの軍が全滅した場合である．

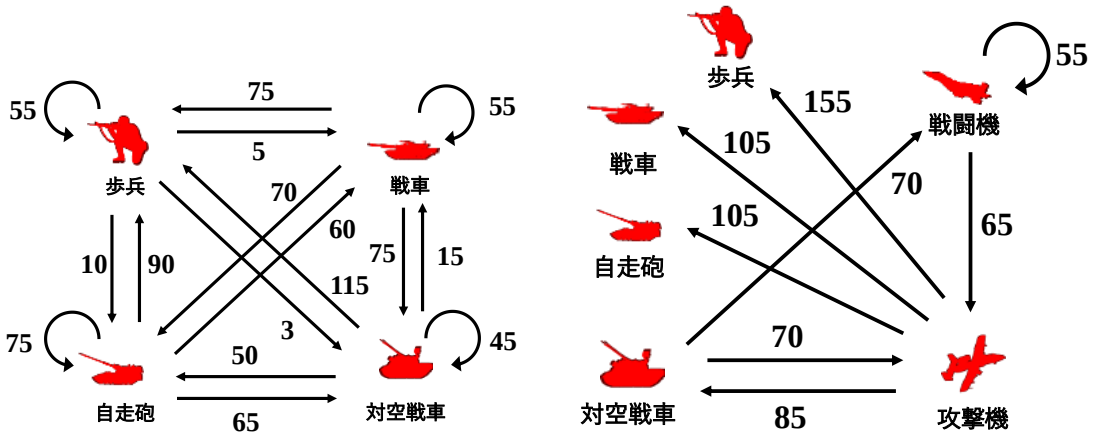
ゲームの勝利条件

ゲームの勝利条件は相手を全滅させるか，ゲーム終了時に自軍の総 HP が相手の総 HP よりもマップことに規定されるしきい値よりも大きい場合に勝利となる．ゲーム終了時に総 HP の差が規定以上でない場合は引分けである．

TUBSTAP で採用されなかったルール

TUBSTAP では解析のしやすさやルールの単純化を図って採用されなかったルールが多数ある．採用されなかったルールとしては，占領，生産，燃料，経験値，天候，索敵，ZOC(Zone of Control, 隣接効果)，支援効果，包囲効果等である．

さらにくわしいルールについては TUBSTAP Web Page [42] を参照されたい．



(a) 地上ユニットの相性関係　　(b) 航空ユニットの相性関係

図 3.4: ユニットの相性関係

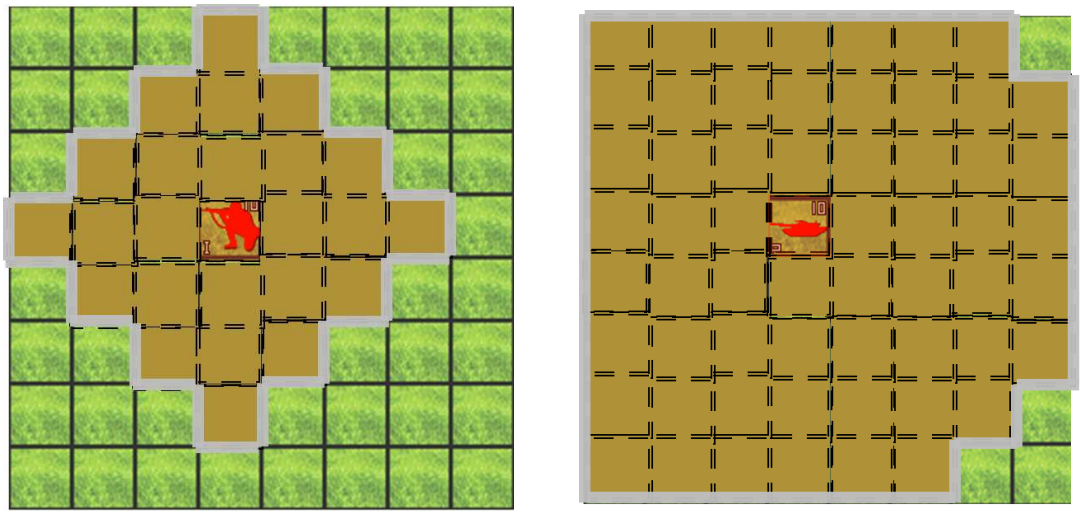
3.2.3 TUBSTAP の探索を困難にする要因

TUBSTAP でアルゴリズムが探索を行う際に，探索を困難にする要因としてユニットの移動の順序や可能移動先の多さ，ユニットの相性の問題，初期画面であるマップの多様さなどがある．それぞれについて以下に説明する．

ユニットの相性関係

ユニットの相性関係について述べる．TUBSTAP のユニットには相性関係があり，これがゲームを多彩なものにしているが同時に複雑なものにもしている．図 3.4 は表 3.1 に記述された攻撃力の関係をグラフで表現したものである．グラフで矢印の方向で攻撃が可能であること，矢印がない場合は攻撃ができないことを表している．矢印に付加された数値が攻撃力で，数値が大きいほど攻撃力が大きいことになる．地上ユニット同士はすべて相互に攻撃可能であるが，攻撃力はユニットによって異なっている（図 3.4(a)）．例として歩兵は歩兵に対して 55 の攻撃力を持つが戦車に対しては 5 しかなく，逆に戦車は歩兵に対して 75 もの攻撃力を持っている．このように攻撃力は現実の兵器のある程度の模擬になっている．図 3.4(b) の航空ユニットのグラフでは，戦闘機は航空ユニットのみ攻撃可能で，攻撃機は地上ユニットのみ攻撃可能となっている．航空ユニットを攻撃できる地上ユニットは対空戦車のみである．ユニットの相性関係は現実の兵器をモデル化，抽象化して模擬するための重みづけと言えるが，この関係があるためにゲームでの攻撃や戦略が複雑化する．

ユニットの移動する順番と可能移動先数の多さについて



(a) 歩兵の可能な移動範囲　　(b) 戦車の可能な移動範囲

図 3.5: ユニットの可能な移動範囲

探索における計算爆発を起こす原因となるのがユニットを選択する順番とユニットの可能な移動先の多さである．マップ上に移動可能なユニットが一個であればユニットを移動する順番は一つ通りしかなく問題はない．しかし，ユニット数が 8 個になった場合は，ユニットの選択する順番は $8! = 40320$ 通りに増加する．全体の 1 ターンにおける合法手の数は，各ユニットの合法手の数の積にユニットの選択順番の数をかけたものになる．

さらには，TUBSTAP においてはユニットの可能な移動先も多い．図 3.5 は歩兵と戦車の 8×8 マスのマップでの可能な移動範囲を表しているが，歩兵の場合で 25 か所，戦車の場合で 61 か所となっている．これは歩兵の移動力が 3，戦車の移動力が 6 であるからである．ユニットの移動力が大きすぎる印象があるのはマップサイズが 8×8 マスであるからで，通常の戦略ゲームはもっと大きなサイズ，16×16 マスや 64×64 マスのマップを使用するのが一般的である．8×8 マスサイズのマップは解析やアルゴリズムの開発を容易にするための機能を制限したマップといえる．

ここで，8 個の歩兵がマップに存在した場合の 1 ターンの合法手の数を推定してみる．歩兵 1 個の合法手が 25 程度として，順番まで考慮すると $25^8 \times 8! \simeq 6.15 \times 10^{15}$ という膨大な数となる．現実には地形の影響や合流などもあるのでここまで大きくはならないが少ない数ではない．

将棋における平均分岐数の数は一説によると平均 80 手程度 [43] といわれているが，かりに将棋の手数と比較すると $80^8 \simeq 1.6 \times 10^{15}$ 程度であるので TUBSTAP の歩兵 8 個の 1 ターンの合法手における読みは将棋での 8 手程度の先の読みに相当すると推定される．通常は相手のターンも含めて予想しないといけないことを考えると倍の 16 手程度の将棋の読みが TUBSTAP での 2 ターンといえる．

マップの多様性について

戦略ゲームは将棋や囲碁のように初期画面が固定されておらず，TUBSTAP においてもさまざまなマップを制作して検討しうる．たとえば，空戦マップや陸戦マップ，陸空が入り乱れたものやパズル的なものや特定の兵種にこだわったものなどを設計できるようになっている．また同一の地形配置であっても，配置されるユニットの数や兵種が異なるとゲームの展開が異なってしまうため，異なったマップとなる．異なった地形や相手のユニットの構成によってユニットの価値は変化することがありまた，ゲームの進行によってゲームの途中でも自軍のユニットの価値や役割が変化することもある．このことはチェスや将棋のように駒の価値を機械学習しておくことで評価関数に使用するといった手法が使えないことを意味する．

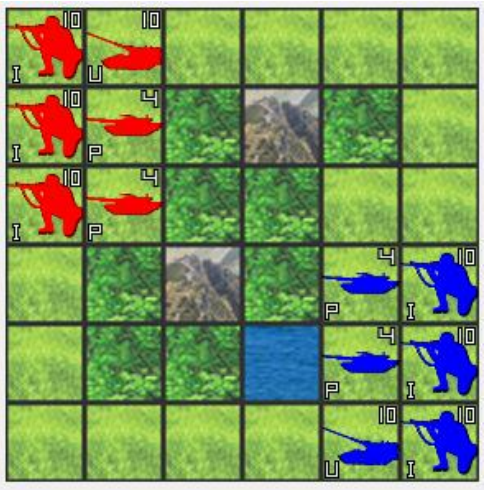
つまりはターン制戦略ゲームにおいては，一つのマップについて学習をしたことが他のマップで応用できるとは限らず，地形や相手と自軍の兵力構成など多岐にわたる条件を合わせた形で同時に学習しなければならない．

3.2.4 TUBSTAP のマップ

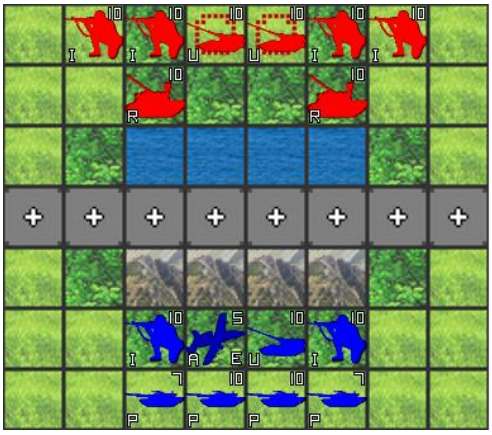
ユーザーは TUBSTAP のマップを自由に設計しかつ，TUBSTAP のフレームワーク上で設計したマップを使い自ら開発した AI の性能を確認することができる．マップ上で使用する駒の数やサイズ的には制限はない．しかし，PC のメモリや動作時間による制約は存在する．実際に制作されているマップには様々なものがある．

TUBSTAP で使用されている戦闘用マップの例を図 3.6 に示す．図 3.6(a) の map01 は 6×6 のサイズの標準的なマップであり，自走砲の使い方が焦点となる．図 3.6(b) の map02 は 7×8 のサイズで，初期配置において双方に各 8 個のユニットは配置されてはいるものの，BLUE 側には戦車が 4 台あるにもかかわらず，RED 側には戦車が含まれておらず，兵種的には BLUE 側に有利な戦力設定のマップである．図 3.6(c) の map03 は 6×6 のサイズで，中央の山をはさんだ自走砲による間接攻撃の使い方が勝敗を分けるようになるマップである．図 3.6(d) の map05 は 8×6 と広めのサイズで，航空ユニットの使い方が焦点となるマップである．

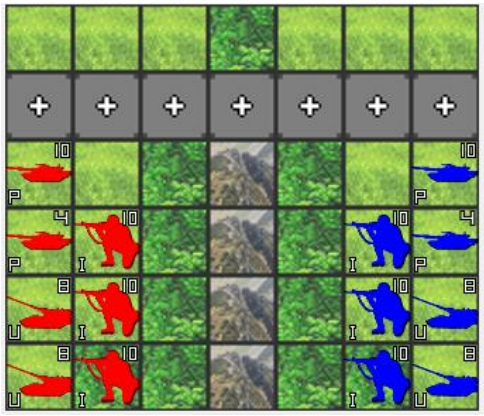
ユーザーがマップを設計するには TUBSTAP のマップファイル仕様に準じたマップファイルを制作する必要がある．TUBSTAP マップファイルには，マップサイズ，最大ターン数，引分けを規定する HP しきい値，RED のユニット数，BLUE のユニット数，各ユニットごとにユニットの位置，兵種，HP，移動済みフラグ，さらにはマップの地形情報がテキストファイルの形で含まれている．ユーザーはマップファイルを編集することで新しいマップを容易に作成することができる．



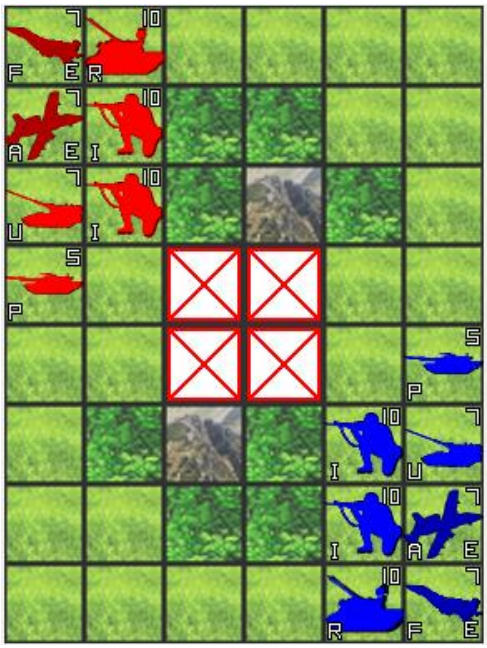
(a) map01



(b) map02



(c) map03



(d) map05

図 3.6: 戦闘用マップ

3.2.5　合法手が多い戦略ゲームと関連研究

Arimaa [44] は駒を 1 ターンに 4 回動かせるゲームであり，駒の間に強弱の相性関係がありターン制戦略ゲームと近い点があるゲームであるが，1 ターンにおける合法手が多いという点や初期配置が一つに限らず様々なものがあるという点でも類似している．Arimaa の 1 ターンあたりの平均合法手は約 18000 通りであり [45]，これはチェスの 35 通り，将棋の 80 通り [43]，囲碁の 250 通りと比較しても圧倒的に大きい．

Arimaa を題材とした研究はさまざまなものがあるが，主に Move ordering に関するものや，棋譜の比較学習による評価関数の作成 [46] などがあった．Arimaa の研究コミュニティでは人間より強いプレイができる AI の開発を目指していたが，2015 年に Arimaa Challenge において人間に勝利したプログラムが登場している [47]．

ターン制ストラテジーゲームとして Civilization シリーズ [48] があり，研究も多いがゲーム内容として「経済」や「外交」といった要素も多く，研究しやすいとはいえない．

探索空間の巨大さやゲームの持つ複雑性やなどから TUBSTAP に適用されたアルゴリズムは限られたものしかなかったが，MCTS によるものが多い．

ターン制戦略ゲームでは第一ターン目から多数の合法手が存在することが多く，合法手について移動の部分で大きく枝刈りする手法を導入して効率化を図るや手法 [49]，UCT の探索優先度にファジイ制御による判断を加えることで効率的な探索を目指す手法 [50, 51] などがあった．

モンテカルロ法を使用するものとして深さと着手を制限した DLMC 法 [52] を提案したものがある．

また Min-Max 法によるものとして探索空間を類似の局面ごとに分割して探索していく手法 [7] があったが性能的には大きな進展はなかったようである．

その他，TUBSTAP の 1 ターンにおける展開される膨大なノード数を削減するために行動の組合せを一つのノードとして展開する UCT 探索方式 [53] も提案された．

3.2.6　幅広探索木問題と行動データ表現問題

可能な行動の数について考えると，将棋は平均して局面あたりに 80 手といわれており，囲碁の場合は着手する可能な点を考えると最大でも $19 \times 19 = 361$ 手までしか存在しない．しかし，TUBSTAP においては駒の数が増えるにつれて 1 ターンにおける可能行動数は増大し，行動順序まで考慮に入れると膨大な数となる．例えば，4 駒の盤面で 1 駒あたり 20 か所の移動先があるものとし，攻撃については考えないとする．この時，可能な行動数は,（駒の順列）×（移動先）×（移動先）×（移動先）×（移動先）となるので， $4! \times 20 \times 20 \times 20\times = 3840000$ という数になってしまう．膨大な可能行動数の爆発によって数手先の先読みすら困難になることがある．

このように Arimaa や TUBSTAP といったゲームにおいては探索木の横幅が爆発的に大きくなってしまうため，この問題を解決するための研究が上記のように行われ

ているが，本論文ではこの問題を「幅広探索木問題」と呼ぶことにする．

同様の特徴を持つゲームは多数存在するためこのような課題のあるゲームを同一の問題としてとらえて研究することには意義があると思われる．

そして，TUBSTAP にはもう一つの課題として「行動データの表現問題」がある．TUBSTAP のように行動空間が広い課題においては駒のとりうる行動をどのような形でデータとして表現するのがプログラム上やアルゴリズムの実行上で都合が良いかという問題である．

具体的には 8×8 マスのマップ上での駒の行動を表現するためには，（移動元）×（移動先）×（攻撃先）の三つの情報の組を 8×8 の位置情報で表現するために $8 \times 8 \times 8 \times 8 \times 8 \times 8$ といったように大きな空間を表現できる形式を取る必要がある．

行動データの表現問題は TUBSTAP に限らず，入れ子構造を持つデータや複雑で広大な行動空間を持つゲームを扱う際には，解決する必要がある問題である．

3.2.7 TUBSTAP における課題を解決する手順

前節において指摘された幅広探索木問題と行動データの表現問題に取り組み研究を進めるために，本論文では段階を追って進めることにする．

まずは TUBSTAP の持つ戦略性やゲーム性についての解析と簡単なアルゴリズムの性能を確認するための基本問題を多数作成しベンチマーク問題として定義する．ベンチマーク問題は駒の数やマップの状況を制限することでマップごとの課題を理解しやすくし，アルゴリズムがマップの課題が解けたかどうがをわかりやすく判定できるようにする．

次に「行動データの表現問題」に取り組む．ターン制戦略ゲームの複雑で多層性があり入れ子構造になったりする行動表現データをニューラルネットワークで扱いやすくするにはどのような表現が良いのかを実際にニューラルネットワークを設計して動かすことで検証をする．強化学習については報酬が入りにくい場合においても立ち上がりが早い手法によりすばやく結果が得られることを目指す．

さらには，「幅広探索木問題」や報酬のはいりにくいターン制戦略ゲームの課題について取り組みを行う．駒の数を一個から二個へと増加させた場合でも探索がうまくいくようにすることや報酬がはいりやすくするための工夫としてランダムマップや手筋マップなどを導入などを検討する．ゲーム研究において重要な方策関数の設計と研究を前の章で行った RNN の設計技術を応用し，設計が複雑なターン制戦略ゲームの方策関数の作成を棋譜学習の形で実現させる．

最終的には前章までに検討した内容を基にしながら，「行動データの表現問題」を解決するための工夫や，入力ユニットの情報をニューラルネットワークの入力部へと移動させるといった「幅広探索木問題」に対応するための工夫を導入して，policy network と value network の双方を使用して探索を行うことで棋譜学習だけでは到達できない強いプレイができる AI プレイヤーを AlphaZero によって提案されたアルゴリズムを応用することで作成する．

本来であればいきなり AlphaZero レベルの AI プレイヤーを作成することは容易にはできない．しかしながら，基本的なベンチマーク問題の作成から出発してニューラルネットワークの設計，幅の狭い探索木から幅の広いものへと広げ，棋譜学習で方策関数の設計に関する基礎を積み重ねることで可能になる．これは囲碁における AlphaGo にまでいたる歴史と同様である．

第4章　ベンチマークマップの提案

この章では，TUBSTAP のアルゴリズムの基本的な性能をチェックすることができるベンチマーク問題群についてその意義や活用法について述べる．なお，これは 2016 年ゲームプログラミングワークショップ (GPW-16) [54] において発表された内容を骨子としている．

4.1　概要

4.1.1　さまざまなゲーム研究におけるベンチマーク問題群

将棋や囲碁では本来のゲームの部分問題として練習用に詰将棋や詰碁問題群が存在し，初心者の練習用や棋力判定，将棋や囲碁のコンピュータアルゴリズムの性能評価などに用いられてきた．また最適化問題や数理計画法の世界でもベンチマーク問題はしばしば用いられ，アルゴリズムの特定の能力・総合的な能力を誰もが評価できるよう整備されてきた．たとえば巡回セールスマン問題のベンチマーク問題集 [55] は有名である．将棋のソフト開発では次の一手問題 [56] などが使用されたことがあり，囲碁でもベンチマーク問題 [57] が用意されてアルゴリズムの性能を評価することが行われた．

このようなことから TUBSTAP について，アルゴリズムに必要なさまざまな能力ごとに，難易度の低いものから高いものまでを含むベンチマーク問題群を提案し，既存プログラムでそれらが解けるのかを確認する．

4.1.2　背景

TUBSTAP ではターン制戦略ゲームの基本的なプレイができるようになっているが，2016 年の時点において用意されている対戦用のマップの数は 10 程度で，十分とはいえなかった．対戦用マップをアルゴリズムの性能評価に使用する場合，マップごとに駒の価値や取るべき戦略も異なってくることに注意する必要がある．それゆえ，各アルゴリズムにはマップごとに得手不得手があることが頻繁に生じ，性能評価を公平に行うためにはできるだけ多様かつ多数のマップが準備されていることが望ましいと言える．

加えて，いままで使用されてきたマップは概して全探索が困難なほどに大規模であり，アルゴリズムに必要とされる能力が複合的であった．これらのマップは人間が遊んだり，プレイヤの総合的な能力を測るのには問題ないが,「どんな能力が原因で勝率差が出ているのか」「最善手は打てているのか」「最善結果は出せているのか」などを分析することは困難であった．

TUBSTAP で動作する，コンピュータアルゴリズムには,「広い探索空間を適切に削減する」「相手の壁役を排除しながら正しい順番で攻撃する」「重要ユニットを同定し，それを壁役で守りながら進軍する」「相手の逃げ道を塞ぐように回り込む，逆に回り込まれないように逃げる」などなど様々な能力が要求される．これらの総合能力を測るだけであれば，人間がプレイしても面白いような大規模で対等に近いようなマップを複数用意すればよいかもしれない．しかし，アルゴリズムの開発あるいは評価のためには，各要素それぞれを評価できるようなマップが整備されていることが望ましい．

つまりベンチマーク問題として必要なマップの条件として，簡単で評価がしやすいこと，難易度がレベル的に設定されていること，問題定義がはっきりしているという事が挙げられる．

4.1.3　ベンチマークマップの意義

通常のゲーム AI 開発においてゲーム AI の性能評価には対戦評価が行われることが多い．対戦による評価では対戦相手との相対的な強さの評価を測定していることになる．対戦相手が弱い場合は測定 AI は強く測定され，対戦相手が強い場合は弱く測定される．対戦による評価では相手との相対的な強さのみの測定になる．用意する対戦相手との強さの差が極端な場合，対戦結果は一方の勝利ばかりとなり，対戦結果から情報を多く取り出すことができず対戦相手より強いか弱いか程度の情報しか得る事ができず，どの程度の差があるのかについてはよくわからない．

これに比較して，ベンチマーク問題は特定状況での課題をこなすことができるかどうかをみているので，ベンチマーク問題は絶対評価であると言える．絶対評価ができるようになることで，作成した AI がどの程度の性能向上したのかをより正確にそして簡単に測定することができるようになる．

これらことや前節で述べた理由などから，本論文では小規模で解析が容易で，必要とされる能力が限定的で，対戦による相対評価ではなく絶対評価が行えるベンチマークマップを，多様にかつ多数用意する．また，既存の比較的強いコンピュータプレイヤに実際に問題を解かせることで，現時点でのアルゴリズムの課題を見つけることを目指す．

本章で紹介するベンチマークマップのファイルは TUBSTAP のホームページ [42] にて公開されている．

4.1.4　ベンチマークマップの設定等

各マップは，全て評価したい側を先攻（RED 軍）とし，主に 8 × 8 のサイズを用いた．TUBSTAP のマップ設定ファイルには「最大ターン数」と「HP しきい値」の設定があり，マップごとに異なった設定となっている．TUBSTAP では勝敗の決定条件は，相手を全滅させるかまたは，最大ターン数に達した時に，RED と BLUE のそれぞれで HP の総和を求め，その差が HP しきい値をこえて大きい方が勝利となる．総 HP の差が HP しきい値より小さい場合は引分けと判定される．本章では，HP しきい値の設定をマップで用いる HP の総和より大きな値を設定して，最大ターンに達した場合は必ず引分けになるようにした．この設定は，もし HP しきい値の設定が小さな値の場合は，マップによっては最大ターンまで生きのびたにもかかわらず，HP の少ない設定側が自動的に負けの判定となってしまうのを防止するためである．マップによっては引分けが最善の結果である．

評価に用いたプログラムは，2016 年 3 月の UEC-GAT 大会 [58] で上位 3 位までに入賞した Military, M-UCT [59], M3Lee [60] の三つである．これらのプログラムはソースコードも含め現在 TUBSTAP のパッケージに標準搭載されており，誰でも参考にすることができる．

ただし，これらのプログラムはもともといわゆる対戦用マップ向けに開発されたものであるため，今回のベンチマークマップに最適化されておらず性能的には高くならないとしても不思議はない．逆にベンチマークマップを解けるように最適化した設定にした場合，通常の対戦で性能を発揮できるかどうかは別の問題である．

敵側（BLUE 軍側）をどのように動かすかは，最適な敵としての行動が自明ではないため難しい問題であるが，評価用のプログラムのうち最強と思える行動をするものを適時選定した．

さらに，いくつかの問題は「ターン最大まで逃げ切れれば正解」「最大ターンまでに倒しきらないと不正解」といったやや通常とは趣の異なる条件を持つため，これに対応していないプログラムも存在する．具体的には，Military はモンテカルロシミュレーションの結果評価として“ 合計 HP が相手より 1 でも大きければ勝ち ”と判定しており，これは「最大ターンまでに倒しきらないと不正解」の問題には不適切な対応である．従って，Military については，モンテカルロシミュレーションの結果評価を“ 合計 HP が相手よりマップで指定された閾値以上大きければ勝ち ”に変更して用いた．

各評価プログラムごと 10 試行を行い，最善の結果を導いた回数を成功としてカウントした．

本章では，マップの命名規則としては，「chase_A1」のように,「目的＋ _ ＋テーマ番号（ A,B,C ）＋レベル（ 1,2,3 ）」を用いる．

表 4.1: 追跡・逃走能力検証用マップにおける搭載 AI の成績

マップ名	目標	最大ターン	Military	M-UCT	M3Lee
run_A1	赤逃げ切り（引分け）	39	0	0	6
run_A2	赤逃げ切り（引分け）	39	0	0	2
run_A3	赤逃げ切り（引分け）	19	0	2	0
chase_A1	青全滅（勝ち）	19	0	10	9
chase_A2	青全滅（勝ち）	19	0	7	10
chase_A3	青全滅（勝ち）	19	0	2	0
run_B1	赤逃げ切り（引分け）	10	10	1	0
run_B3	赤逃げ切り（引分け）	30	0	0	0
chase_B1	青全滅（勝ち）	19	4	8	7

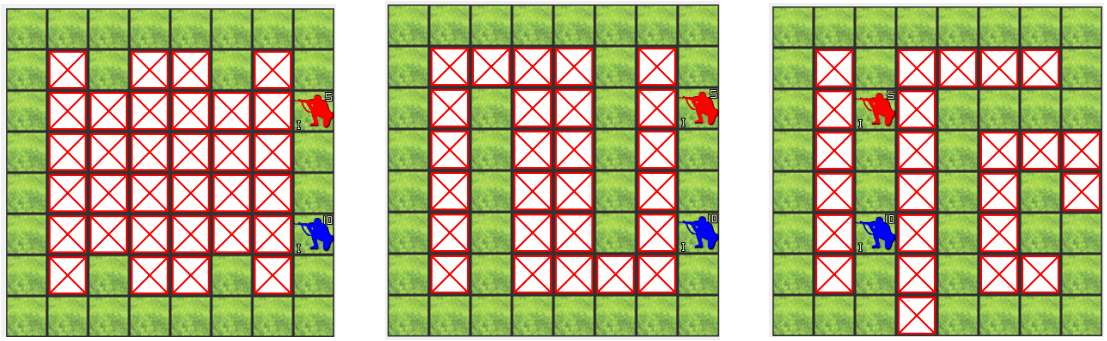
4.2 追跡・逃走能力検証用マップ

追跡アルゴリズムはゲームアルゴリズムのなかでも重要かつ基本的なものであってゲームアルゴリズムのテキスト [61] などでも紹介されることが多い．戦略ゲームの中でも，相手を逃がさないように追跡することが必要な局面は頻繁に出現する．また，相手の抵抗を考えながら追跡するアルゴリズムを正しく作るには，逃走を正しく行えることも必要である．ここでは，2 つの異なるテーマを持つ追跡 (chase) および逃走 (run) のマップセットを紹介する．run シリーズでは，RED 軍は戦力的に不利な状況にあり，戦闘を行うと敗北する設定であるが，障害物を利用して正しく逃走すれば，いくらでも逃げ延びることができる．最大ターンに達すれば，引分け，すなわち成功となる．chase シリーズでは，RED 軍が戦力的に優位な状況にあり，正しく追いつめれば最大ターン内で BLUE 軍を全滅させることができる．最大ターンを超えれば，引分け，すなわち失敗である．

4.2.1 製作したマップと評価結果

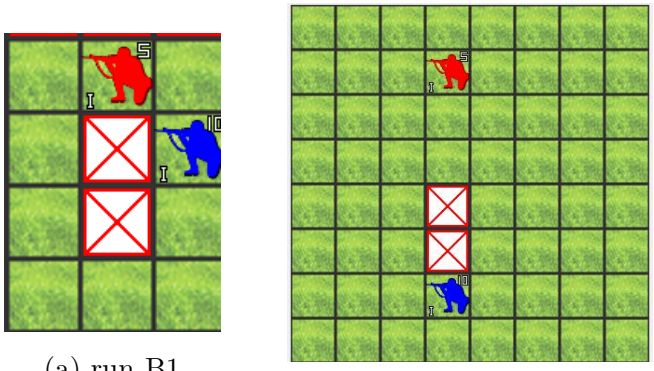
表 4.1 に制作したマップの搭載 AI による成績を示す．マップの目標は，run シリーズならば最大ターンまで逃げ切ればよく，chase シリーズならば最大ターンまでに全滅させる必要がある．

図 4.1(a) および図 4.1(b) は袋小路に入らぬように逃げ回れば良いマップで，人間ならば容易に引き分けに持ち込める．しかし M3Lee が時に正解する以外は逃走に失敗している．途中までうまく逃げても， 39 ターンという比較的長期間の間にアルゴリズムの乱数性により失敗するようである．基本的に逃走マップでは相手の行動番をしっかり読む必要があり，それは自分手番だけでも膨大な行動数になる戦略ゲームでは既存アルゴリズムが苦手とするようである．図 4.1(c) はターン数が短く設定されている代わりにマップ右半分方面への遠い逃げ道が袋小路となっており，アルゴリズ



(a) run_A1　(b) run_A2　(c) run_A3

図 4.1: 逃走マップ A



(a) run_B1　(b) run_B3

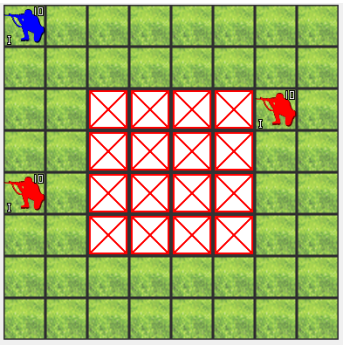
図 4.2: 逃走マップ B

ムが騙されやすい．

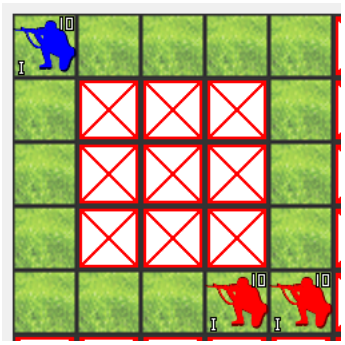
図 4.2(a) はある意味 run A1 などよりも単純な問題で，相手の裏側に位置して逃げ延びることを目指す．この問題は今まで成績の良くなかった Military が正解しており，アルゴリズムごとに得手不得手があることが分かる．図 4.2(b) は run B1 の応用問題であり，2 歩下に下がれば同様の方法で逃げ続けることができる．しかしコンピュータは例えば右上方面など，一見安全度が高いと思われる方向に逃げてしまい，最後は追いつめられてしまう．シミュレーションの中ではなかなか正しく BLUE 側の追いつめが成功しないことも，右上方面を安全と思ってしまう原因であろう．

図 4.4 は非常に単純な追いつめ問題であり， run B3 の部分問題と言える．アルゴリズムは 19 ターンという十分な時間を与えられても必ずしも正解しない．よくある失敗は，敵歩兵が自分から見て（3 歩右，3 歩下）にあるような状況で，下に 3 歩進んでしまうことである．続いて敵歩兵が上に 3 歩進んだときにも上に 3 歩進んでしまえば，これの繰り返しで追いつめられないことがある．

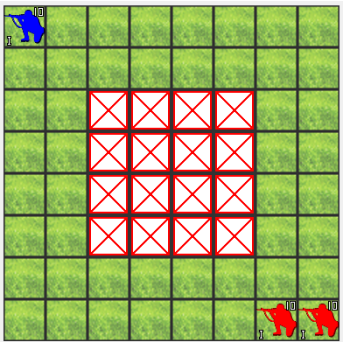
図 4.3 は RED の 2 つの歩兵が二手に分かれて BLUE 歩兵を挟み打ちにしなければ



(a) chase_A1



(b) chase_A2



(c) chase_A3

図 4.3: 追跡マップ

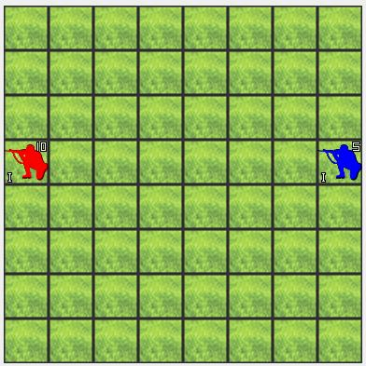


図 4.4: 追跡マップ chase_B1

勝てないマップである．これも人間であれば挟み打ちの必要性が分かるのだが，コンピュータには chase_A3 は難しいマップでほとんど正解されていない．そもそも，run_A1 などの結果から,「片方面だけから攻めた場合，BLUE 歩兵が正しく抵抗すれば逃げ切られてしまう」ということが読めていないのが一つの原因であろう．図 4.3(a) および 図 4.3(b) は 図 4.3(c) を「二手に分かれる」「正しく仕留める」という二つの部分問題に分けたものである．探索空間が小さくなることもあるせいか，これならば完全とは言えないまでも解けるようである．

4.2.2　マップ chase_A3 の全滅ターン数解析

図 4.3(c) は RED 手番で開始すれば 14 ターンで BLUE を全滅できると，手作業であるが解析できていたものだが，本章では，BLUE 軍の全ての抵抗手段に対して必ず 14 ターンで全滅に至ることを確認するための木探索を行った．問題設定は少し変更し，BLUE 手番で開始することにし，14 ターン内に RED 軍が BLUE 軍を全滅できることを確認した．

解析の手順と条件は，

- BLUE 軍側はすべての逃走経路についての場合について展開

表 4.2: chase_A3 の解析結果

ターン数	9	10	11	12	13	14
ゲーム途中のノード数	127	52	437	15	92	0
同一局面の数	33	11	113	5	29	0
ゲームが終了したノード数	0	289	0	1236	0	290

表 4.3: 経路探索能力検証用マップにおける搭載 AI の成績

Map 名	最大ターン	Military	M-UCT	M3Lee
path_A1	29	0	10	10
path_A2	11	0	9	3
path_A3	39	0	1	2

- RED 軍側は挟み撃ち戦略で青軍側の逃走経路をせばめていく手順のみを読む
- MINMAX 木探索と同様にノードをすべての場合について展開する
- RED 軍は 2 つのユニット同時に青軍と戦闘に入り，逃がさないようにする
- 同一ターンで同じ局面ノードが出現したらそのノードは同じノードへと集約する

のように設定した．解析した結果は表 4.2 のようになった．ここでは，9 ターン以降のデータを示している．データでは，10 ターン目または 12 ターン目でゲームが終了する場合も多いが，最長でも 14 ターンでゲームはすべて終了した．RED 軍側はヒューリスティックを用いて全探索にはなっていないので，今回の結果は「14 ターン以内に全滅はできる」ことを示したものであり,「12 ターンでは全滅できない」ことは示せていない．

4.3 経路探索能力検証用マップ

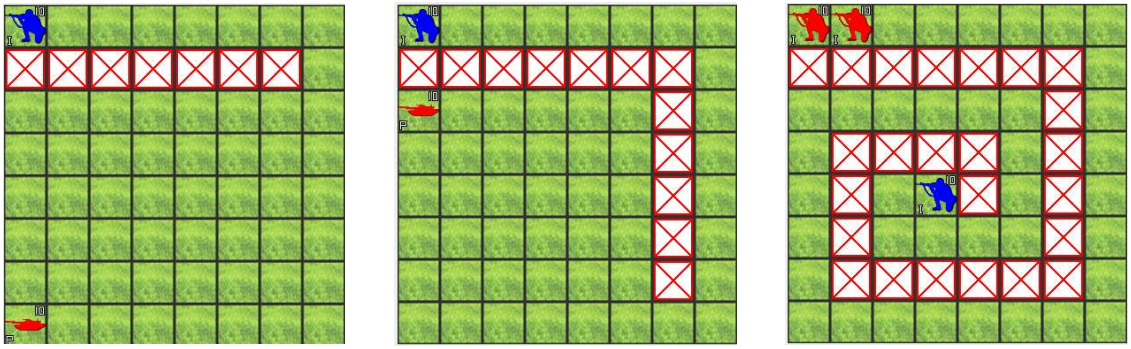
追跡能力の発展形として，経路探索能力が必要になる場合もある．現在 TUBSTAP で用いられているような比較的小規模のマップではあまり問題にならないが，生産や占領があるゲームの大規模なマップ，要塞攻略のような趣のあるゲームのマップでは，入り組んだ道筋を辿って目的地に到達する能力が必要である．

本節ではこれを経路探索能力検証用マップとして，pathfind シリーズを準備した．

4.3.1 製作したマップと評価結果

表 4.3 に製作したマップの搭載 AI での成績を示す．

図 4.5(a) は敵歩兵の前に壁があり，回り込んで倒す必要があるマップである．非常に単純であるが，敵との距離をマンハッタン距離で詰めるような移動アルゴリズム



(a) path_A1　(b) path_A2　(c) path_A3

図 4.5: 経路探索マップ

表 4.4: 封鎖能力検証用マップにおける搭載 AI の成績

Map 名	最大ターン	Military	M-UCT	M3Lee
block_A1	9	0	6	0
block_A2	9	0	3	0
block_A3	9	0	0	0
block_B1	9	10	4	0
block_B2	9	0	0	0

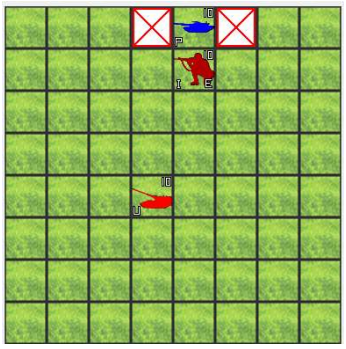
の場合，失敗してしまう．図 4.5(b) は遠回りをしなければいけない上に，最大ターン数もぎりぎりに設定されており，比較的難易度が上がっており，成績も若干低下している．図 4.5(c) は歩兵が長い道のりを周回して目的の敵に到達する必要があり，最大ターン数の設定もきつく，既存プログラムではなかなか正解できない．これらの問題は人間であれば，正解手数を正しく読めるかはともかくとして，正解手順を見つけることは難しくない．特にモンテカルロシミュレーションを使うような手法ではこういった問題は課題であると言える．

4.4　封鎖能力検証用マップ

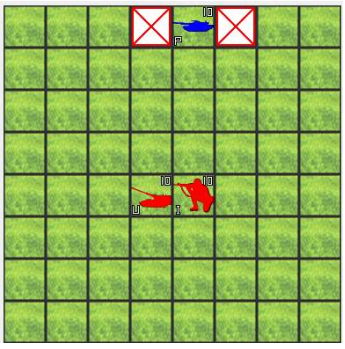
将棋や囲碁でもうまく相手の活動を邪魔するような行動は必要になるが，ターン制戦略ゲームでは駒の移動範囲が広いだけに，周囲を取り囲んで封鎖する，重要地点を押さえて侵攻を免れることは必須の能力になる．

4.4.1　製作したマップと評価結果

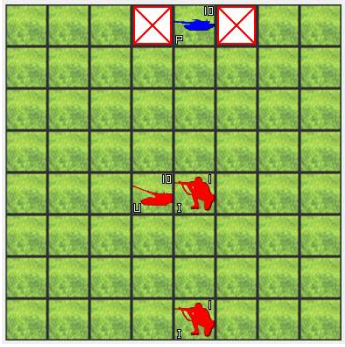
表 4.4 に製作したマップの搭載 AI での成績を示す．図 4.6 は，BLUE 戦車の出口を歩兵が犠牲になって足止めし，その間に自走砲が遠距離攻撃して倒すというテーマのマップである．図 4.6(a) は初手の歩兵の移動を完了させてある簡易バージョン



(a) block_A1
歩兵は移動済み

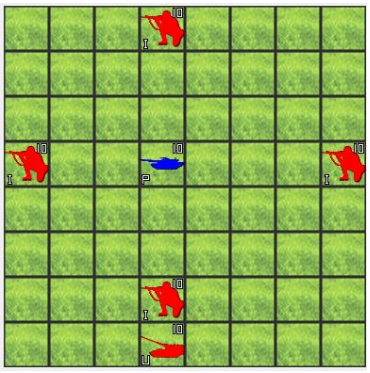


(b) block_A2

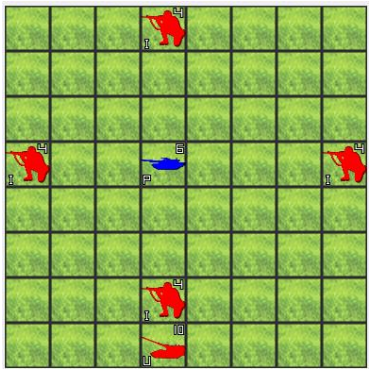


(c) block_A3

図 4.6: 封鎖マップ A



(a) 封鎖マップ block_B1



(b) 封鎖マップ block_B2

図 4.7: 封鎖マップ B

であるが，これでもこの問題を正解するアルゴリズムは少なかった．逃走マップでも述べたように，相手の移動や攻撃を読み切れておらず，マップ下側に逃走して引分けを試みるような挙動が多く見られた．図 4.6(b) は足止めを 1 回，図 4.6(c) は足止めを 2 回にわたり行うテーマのマップであり，この程度であれば人間は簡単に正解を導くが，block_A2 を成功した AI があるものの block_A3 を成功できる搭載 AI はなかった．

図 4.7(a) は，障害物のないマップで歩兵 4 個が戦車の上下左右を封鎖する必要があるマップである．これには Military が全正解するなど好成績となったが，その理由は HP10 の歩兵が戦車を攻撃すると HP を 1 減らせるため,「封鎖」ではなく「有効な攻撃」としてこの行動が選択されるためと思われる．一方，図 4.7(b) では，封鎖して 1 回攻撃するだけで（深さ 3 まで読むだけで）正解するにも関わらず，搭載 AI では正解できなかった．

基本的に戦車や自走砲といった移動力が高く，移動可能なマスの多い駒が出現すると AI の読みは正確さを失う場合が多かった．

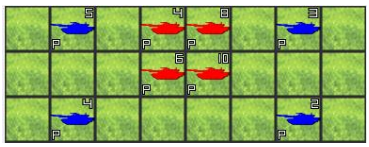
表 4.5: 選択能力検証用マップにおける搭載 AI の成績

Map 名	最大ターン	Military	M-UCT	M3Lee
select_A1	1	0	10	2
select_B1	1	0	10	2
select_B2	1	0	2	0
select_C1	9	9	7	10

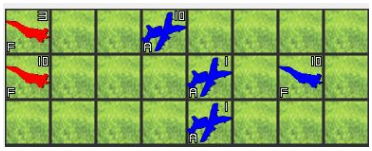
4.5 選択能力検証用マップ

攻撃できる範囲や相手，さらには順番の選択手が多いターン制戦略ゲームでは，どの順序でどの相手を攻撃するのかを適切に読む必要がある．select シリーズでは，全てのマップで，正しく行動を選択できれば開始ターンで直ちに敵を全滅できるように設計してある．相手番を読む必要がない代わりに，自分の手番は正確に読む必要がある．

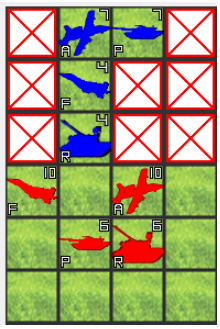
4.5.1 製作したマップと評価結果



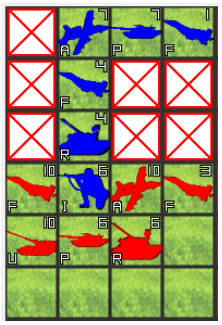
(a) 選択マップ select_A1



(b) 選択マップ select_C1



(c) 選択マップ select_B1



(d) 選択マップ select_B2

図 4.8: 選択マップ

表 4.5 に作成したマップの搭載 AI での成績を示す．

図 4.8(a) は，双方 HP の異なる各 4 個の戦車を持ち，全ての赤戦車が全ての青戦車に攻撃できる状況である．組み合わせとしては 24 通りの攻撃パターンがあり得るが，正解は 1 通りである．これに M-UCT は毎回正解したが，他のアルゴリズムはほぼ失敗している．図 4.8(c) は，多様なユニットが用いられ，敵を 1 個ずつしか攻撃でき

表 4.6: 護衛・タッグ能力検証用マップにおける搭載 AI の成績．tag_A1 については最大ターン数を 3 から 9 まで変更してそれぞれ評価した

Map 名	最大ターン	Military	M-UCT	M3Lee
guard_A1	9	9	0	4
guard_B1	19	1	5	1
tag_A1	**9**	10	10	10
-	**7**	7	5	6
-	**5**	3	6	6
-	**3**	0	0	0
tag_B1	9	0	0	0

ないマップとなっている．このような「敵陣を掘っていく」ような読みは実戦では非常に頻繁に生じる．これも M-UCT 以外はほぼ失敗している．図 4.8(d) は同じテーマでユニット数が 6 に増えたものであり，人間でも多少迷うかもしれない．M-UCT の正解率も低下している．図 4.8(b) はテーマが異なり,「正しく価値の高い相手を攻撃できるか」を調べるものとなっている．ターン制戦略ゲームではマップごとに駒価値が異なるが，本マップでは敵攻撃機は放っておいてもよいために価値が低く，敵戦闘機を攻撃しなければならない．搭載 AI はほぼ正解を出したが，場合によっては攻撃機の側を攻撃してしまう．

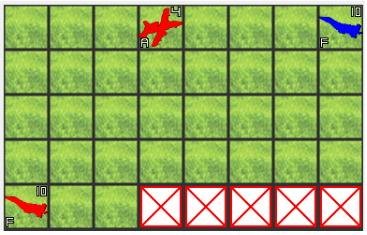
4.6 護衛・タッグ能力検証用マップ

ターン制戦略ゲームでは，特定のユニットを防護しながら進軍することが求められることが多い．例えば占領のために歩兵を戦車で守るであるとか，逆に虎の子の戦車を守るために歩兵が壁になるとか，与えられる役割や必要な戦略は場合によって異なる点が難しい．guard シリーズでは，あるユニットを護衛したうえでそのユニットが敵に先制することで勝利に導くようなマップを作成した．また，単独では敵に先制されたり射程範囲を外されたりしやすい自走砲を，壁役で守るのではなく，複数協調して運用することが必要になるケースもある．これにより，反撃の可能性を見せることで敵の先制を防いだり，攻撃範囲の網を広くかぶせることが可能になる．このような能力を検証するためのマップとして tag シリーズを作成した．

4.6.1 製作したマップと評価結果

表 4.6 に作成したマップの搭載 AI での成績を示す．

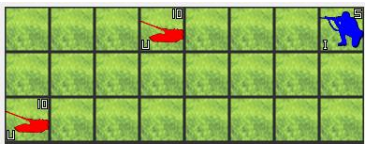
図 4.9(a) は，戦闘機同士のにらみ合いであり，先制した側が勝利する．移動力が同じなので，単独では先制できず手詰まりとなるが，壁役の攻撃機ユニットがあるため，戦闘機が 2 歩右に進み，攻撃機がその直上を護衛すれば勝利することができる．



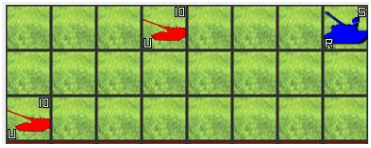
(a) 護衛マップ guard_A1

(b) 護衛マップ guard_B1

図 4.9: 護衛マップ



(a) タッグマップ tag_A1



(b) タッグマップ tag_B1

図 4.10: タッグマップ

図 4.9(b) は，歩兵を壁にして自走砲が戦車を追い詰めるテーマである．

図 4.10(a) は，自走砲がマンハッタン距離 2 （1 マスナナメや，上下左右 2 マス先）にならないように進行することで，歩兵の逃げ先である死角を作らず全滅できるマップである．正しく操作すれば 3 ターンで BLUE 歩兵を全滅させることができる．これを搭載 AI に解かせた場合，最大ターン数を 9 とした場合は全 AI が毎回 BLUE を全滅させたが，最大ターン数を 7, 5, 3 と厳しくしていくうちに正解率は下がった．何度か述べている通り，自走砲の行動については必ずしもうまくできていない．

図 4.10(b) は相手が移動力 5 の対空戦車に変わったものである．これは追いつめ方を間違えると包囲をすり抜けられ，最大ターン 9 では倒せなくなる（最短は 5 ターン）．それなりに正しい先読みが必要で，搭載 AI には解けなかった．

4.7 パズル型マップ

これまでの問題も，ある意味でパズル性の強いマップであった．ここでは，やや必要能力の分類が困難なマップについて紹介する．いずれも，これまで同様，人間には簡単に解けるが，既存のコンピュータには難しいものが多い．

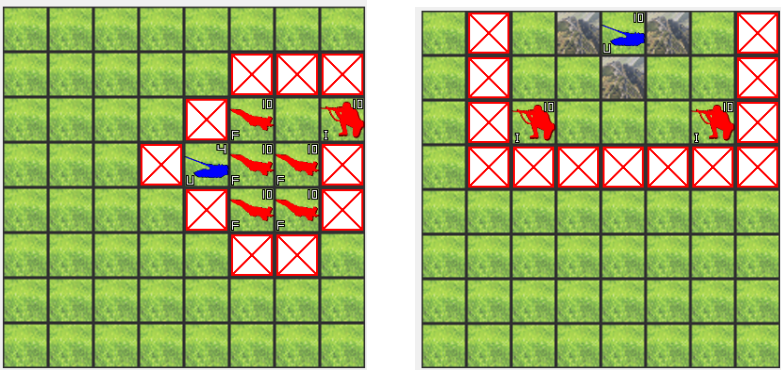
4.7.1 製作したマップと評価結果

表 4.7 に製作したマップと搭載 AI での成績を示す．

図 4.11(a) は，歩兵が敵の自走砲を倒すために，邪魔な戦闘機がスペースを空ける必要がある問題である．select B シリーズでも攻撃の順番が重要になったが，このマップでは攻撃だけでなく移動も必要である点が要点である．「攻撃できるユニット

表 4.7: パズル型マップにおける搭載 AI の成績

Map 名	最大ターン	Military	M-UCT	M3Lee
puzzle_A2	19	1	10	10
puzzle_B1	29	0	8	10



(a) パズルマップ puzzle_A2　　(b) パズルマップ puzzle_B1

図 4.11: パズルマップ

から行動する」といったアルゴリズムはしばしば用いられるが，それでは正解が見つからないということになる．

puzzle_B シリーズは，歩兵 2 個で山に囲まれた自走砲を攻撃するというテーマの問題である．片方が犠牲になるという意味では tag シリーズにも関連する．図 4.11(b) はシリーズ中で最も簡単なマップである．

4.8　おわりに

この章では TUBSTAP の開発されたアルゴリズムの性能を簡単にチェックすることができるベンチマーク問題を開発し，性能を既存プログラムで評価した．

これらのベンチマーク問題によって，アルゴリズムの基本的な戦術や戦略を実行する能力を評価するができる．ベンチマークの問題は一見簡単ではあるものの，アルゴリズムの設計の仕方によっては正解するのが困難だったり，正しい行動ができなかったりするため，アルゴリズムのバグのチェックや成熟の度合いを実際に確認ができて便利である．

第5章　リカレントネットワークと Profit Sharing によるターン制戦略ゲームの強化学習

この章では，深層学習をターン制戦略ゲームへと適用する場合に課題となる設計部分と，実際に適用した例について述べる．ターン制戦略ゲームを研究する上での課題である「行動データの表現問題」を本章における提案で解決し，すばやい学習ができる手法について考察する．

この章はゲーム情報学研究会 (IPSJ-SIG-GI 2019) [62] において発表された内容を骨子としている．

5.1　深層学習と強化学習

5.1.1　深層学習をターン戦略ゲームに適用するにあたって

この章で説明するのは，DQN や AlphaGo において適用された深層学習をターン制戦略ゲームに適用する上での問題点を明らかにし，解決するための提案を行う．ターン制戦略ゲームと，囲碁や DQN で使用された Atari 2600 のゲームなどを深層学習を適用する観点から比較すると，大きな違いとなるのがゲームの性質の違いからくる扱うデータの構造の違いである．ターン制戦略ゲームの扱うデータは複雑で盤面の大きさも固定ではなく，1 ターン内で複数の駒が動作し，また駒の可能な移動先も多い．このような事情から AI エージェントは複数の駒の行動についての指示の出力をしなければならないし，また，1 つの駒の行動指示についても最低限のデータを含んだ形でなければならない．

5.1.2　強化学習

強化学習とは以下の評価関数 V_t を最大化するように，各状態での適切な行動選択の方策 π を学習することである [63].

$$V_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^n r_{t+n} + \cdots , \tag{5.1}$$

ここで γ は割引率（discount rate） r_t は時刻 t で得られる報酬（reward）である．つまり，現時点から未来にわたって得られる報酬の重み和の最大化を目標とする．

Q-learning

強化学習アルゴリズムのひとつに Q-learning がある．Q-learning では状態と行動のペアである個々のルールの評価値として Q 値と呼ばれる値を使用する．さらにさまざまな状態で実際に行動を実行して得られる報酬を基に Q 値を更新していく．Q 値の更新式は次のようになる．

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma V(s_{t+1})), \tag{5.2}$$

$$V(s) = \max_{b \in A} Q(s, b) \tag{5.3}$$

この式は時刻 t において状態が s_t であって行動 a_t を実行した結果，状態は s_{t+1} に遷移し，報酬 r_t が得られた時に適用され，α は学習率（learning rate）である．Q-learning は特定の方策モデルを必要としないため方策オフ型学習と呼ばれている．Q-learning はエージェントの行動選択において，全ての行動を十分な回数選択し，かつ学習率 α が $\sum_{t=0}^{\infty} \alpha(t) \to \infty$ かつ $\sum_{t=0}^{\infty} \alpha(t)^2 < \infty$ ．を満たす時間 t の関数となっているとき， Q-learning のアルゴリズムで得る Q 値は確率 1 で最適な Q 値に収束する（概収束） [64]．ただし，環境はエルゴード性を有する離散有限マルコフ決定過程であることを仮定する．

DQN

Q-learning における関数近似を深層学習によって行うのが DQN（Deep Q-network） [4, 6] であり，深層学習によって高次元で非線形な Q 関数を高精度で近似している．しかし，ニューラルネットワークの学習のためにデータと学習時間を大量に必要とする課題があり，このような DQN の課題を解決するために後継アルゴリズムが多数開発・研究されている．

5.1.3 Profit Sharing 法

Profit Sharing 法

Profit Sharinig 法（利益共有法） [63, 65, 66] では，実行されたルールの履歴を保存しておき，報酬が得られるたびに過去にさかのぼってルールの強さを更新する．更新はある時間ステップ t において報酬 r_t が得られるときエピソードに参加したルール R_{t-i} の強さ S_i は次式で行う．

$$S_i(t) \leftarrow (1-\alpha)S_i(t) + \alpha f_i(r) \tag{5.4}$$

$f_i(r)$ はエピソードの最後のステップ t から数えて i ステップ前のルールに分配する報酬の大きさを決定する関数で強化関数と呼ばれる．

Profit Sharing 法は少ないサンプルデータで高速で学習ができる長所があるが，短所としては探索性が弱く経験に固執して性能が上がりにくくなることがある点と，問題環境が大きくなり必要な行動選択回数が増えると学習が進まなくなることがある点が挙げられる [67].

Profit Sharing 法においても強化関数の方策の局所合理性を保証する合理性定理が知られている [68] .

先行研究

DQN への Profit Sharing 法の適用には DQNwithPS[69] があるが，良いゲームシミュレータを使用し価値の高い経験を用意することが重要であることが述べられている．この手法は各エピソードの終わりにエージェントが報酬が得られた場合にすべてのルールに報酬を規定の式に応じて分配しリプレイメモリに保存する．しかるのちにリプレイメモリから規定のサンプルをランダムに取り出し、通常の DQN と同様に勾配を学習する．通常の DQN との違いは報酬が得られた時は Profit Sharing 方式の学習を行い，報酬が得られなかった場合には通常の DQN の学習を行うところである．どちらの場合もペナルティ（罰）は使用している．

Profit Sharing のターン制戦略ゲームへの適用を考える際にはデータ構造に関する考察と解決が必要になってくる．

5.2 ニューラルネットワークについて

5.2.1 ニューラルネットワークの出力部のデータ表現

複雑なデータ構造を扱うようになってニューラルネットワークの設計は難易度が増している．

ニューラルネットワークの出力構成の問題

通常のニューラルネットワークの出力の形式は二値分類形式か多値クラス分類形式に分けられる．複雑な構造があったり階層構造や入れ子構造になっているデータの表現には二値分類形式か多値クラス分類形式を組み合わせて表現するか，テータをすべて展開して多数のニューロンで表現しなければならない．複数の形式を組み合わせて表現することはマルチラベル問題 [70] ではしばしば用いられ，そのための手法を参考にすることもできると考える．

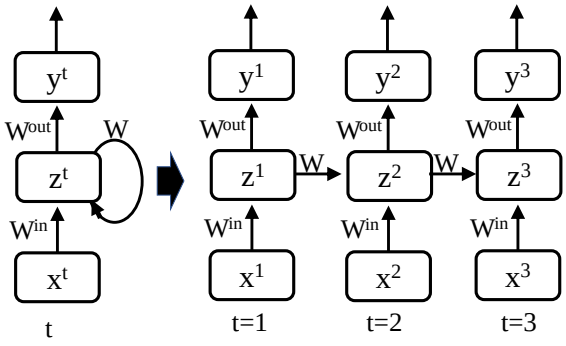


図 5.1: RNN と時間方向への展開

TUBSTAP に深層学習を導入し駒の動作を表現するためには相互に依存するデータを同時に表現しなければならないためマルチラベル問題とまったく同一ではないものの似たような表現上の課題がある．

ゲームでのニューラルネットワークの出力表現

ゲームにおけるニューラルネットワークの出力部の構成について以下にまとめる．

- DQN での Atari 2600 のゲーム群は統一された操作部を想定しており，ジョイスティックによる 8 方向の方向入力とボタン入力のみの $8 \times 2 = 16$ となっていた [6]．
- AlphaZero[3] において囲碁の出力構成は，囲碁の盤面の 19×19 マスとパスを表示する一個で合計で $19 \times 19 + 1 = 362$ 個であった．
- AlphaZero Chess の場合はチェスのすべての指し手を表現する $8 \times 8 \times 73 = 4672$ の出力があった．
- AlphaZero Shogi のニューラルネットワークの出力は将棋のすべての指し手を表現する $9 \times 9 \times 139 = 11259$ となっていた．

このようにゲームで扱われているニューラルネットワークの出力構成は将棋については複雑化してきているものの，まだ比較的単純ものが多いといえる．

5.2.2 RNN

RNN(Recurrent Neural Network: 再帰型ニューラルネットワーク) とは内部的に閉路を持つことで情報を一時的に記憶することができるニューラルネットワークである（図 5.1）．旧来の RNN では深いネットワークでの誤差逆伝搬アルゴリズムでの勾配消失問題や長い系列での関連性をとらえられないといった問題があった [25]．これに

対応するためゲート構造を内蔵したRNNがLSTM(Long Short Term Memory)である．RNN，特にLSTMは時系列データの解析とりわけ自然言語処理で多く使われているが近年ゲーム分野での活用も進んでいる．またLSTMの内部構造を簡略化し高速化を図ったGRU(Gated recurrent unit)もある [71].

5.3 提案システム

複雑なデータ構造でワンホット出力に適していないニューラルネットワークの出力段を適切な分割をすることで扱いやすくし，さらに各出力の関連性をRNNを採用することで高めたシステム提案を行う．さらに，TUBSTAPにおいて強化学習を行うには使用できるマップの数やデータ量に限りがあり，何が価値のある行動であるか判断するのが困難な場合が多いためProfit Sharingによる少ないサンプル数による効率的な学習を試みる．そして，経験を重視するのがProfit Sharingであるが深層学習による汎化能力による未知のマップへの対応能力の向上を図る．

5.3.1 ニューラルネットワークの出力段の設計の課題

TUBSTAPのようなターン制戦略ゲームでは1ターンで複数の駒が動作することを想定した行動の表現が必要となる．具体的にはユニット1個分の行動について

- 移動元：どの駒についての行動であるかを示す
- 移動先：どの位置への移動であるかを示す
- 攻撃先：どの駒を攻撃するかを示すかまたは攻撃しない

のような情報を含んでいなければならない．

ニューラルネットワークにこれらの情報を出力させる設計を考える際に必要なニューロン数を検討する．この場合，8×8マスのマップをそのままエンコードするとすると移動元で64個，移動先で64個，攻撃先で攻撃しないフラグを含めて65個のニューロンが必要となる．これをそのまま通常のクラス分類問題としての出力段と考えると$64 \times 64 \times 65 = 266240$　ものニューロンを用意しなければならない．これはAlphaZero Shogiの11259やAlphaZero Chessの4672と比較しても非常に多い．ここでは1ユニット分についての計算なのでもし仮に，多ユニットを同時に表現しようとする際にはもっと増加することになる．これだけの数の設計では通常GPUのメモリにはのらないので動作しないか，動作しても学習がきわめて遅くなってしまう．本章の取り組みでは出力段を移動元，移動先，攻撃先として三つに分割することで必要なニューロン数を低減させる．

図5.2にこの場合の行動出力の構造でデータ表現について摸式的に示す．ここではマップは8×8マスであって，移動元のマスが決定して，しかるのちに移動先のマス

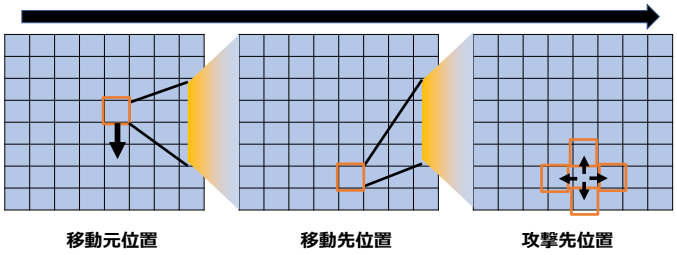


図 5.2: 行動出力の構造とデータ表現

を決定し，最終的に攻撃する先のマスを決定するという手順を表現している．攻撃するマスでは攻撃しない，という選択した場合を考えると 5 つの選択肢があることになる．各行動が合法手となるかどうかは，各ユニットの配置やマップの状況と各行動との組み合わせによって変化する．例えば，移動元の行動表現はマス目にユニットが配置されていなければならない．また移動先のマスは移動元からユニットがルールにてらして移動可能なマスでなければならない．このように移動先が合法手であるか確定するのは移動元の行動が決定してからなので，この行動の列には時間関係が存在する．そして，攻撃先が合法手かどうか確定するのは移動元と移動先が合法手であって，ルールにてらして合法な行動であった時に決定するため，同様に時間関係が存在する．

この移動元と移動先と攻撃先のデータ表現は一つのデータの中に次のデータが影響を受ける階層的または入れ子のような構造となっている．これを単純化して表現することを考えた場合には RNN のように時分割できるニューラルネットワークが適している．1 つの行動につき 1 ステップごとの出力を割り当てることで単純な表現となる．他にも可能な入出力の形式はさまざまありうる．例えば，3 つのネットワークを用いて，1 つ目が状態を入力として移動元を出力し，2 つ目が移動元と状態を入力として移動先を出力し，3 つ目が移動先も入力とするようなものなどである．しかしながら，これは性能はともかくとして 3 つの異なるネットワークを持つ必要があり，標準的な実装では GPU が 3 つ必要になるため，広い活用を考えたときには利便性が悪いと思われる．

5.3.2 RNN の TUBSTAP への適用

出力段のニューロン数を抑えることだけを考えるのであれば，出力を上記の項目ごとに分割してニューラルネットワークを多出力の構成にすることで実現できる．しかしながら，多出力のニューラルネットワークの構成では学習において特定の出力段の学習のみ改善されてしまうことで学習が偏ったり相互の出力の関連性を持たせるのが難しくなったりする問題がある．このような各出力の相関を取り学習の偏りを防止するために 図 5.3 のようにリカレントネットワークを使用することができると考えて提案する ．ここでは t=1 で駒の移動元の出力 z^1 が RNN のフィードバックによっ

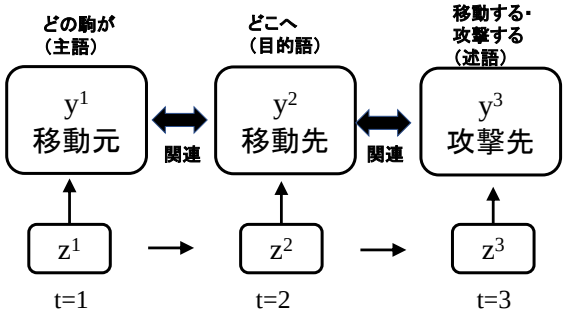


図 5.3: 出力段の RNN による構成

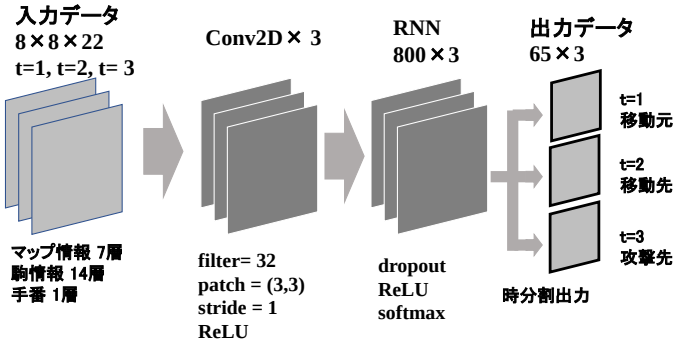


図 5.4: 使用したニューラルネットワーク構成

て次の段に入力され，z^2 の出力に影響を与えこれが移動先となる．さらに z^2 の出力が次の段に入力され z^3 の出力に影響を与えこれが攻撃先となる．実際に使用したユニットは LSTM ではなく実行速度を考慮して GRU ユニットを採用した．

5.3.3 使用したニューラルネットワーク

今回使用したニューラルネットワークの内部構造を図 5.4 に示す．入力データはマップ情報の地形情報 7 種類と駒情報（兵種 6 種類＋移動済みフラグ）を RED と BLUE で 7 × 2，これに加えて手番情報が 1 で，トータルで 7 + 7 × 2 + 1= 22 の情報を 8 × 8 のマス目状に変換してエンコードした 22 チャンネルのニューラルネットワーク入力用データである．駒の HP は [0, 1] の範囲に正規化され駒情報層に浮動小数点の形式で入力される．入力層には通常の自然言語処理用のものは Embedding 層が採用されるが今回は畳み込み層を採用して三層積層している．ついでリカレントネットワーク層が積層されて三層あり，各層のニューロン数が 800 となっている．入力データは t=1, t=2, t= 3 で同一データを入力する．出力は駒の行動情報を時分割 (t=1 移動元, t=2 移動先, t= 3 攻撃先) で出力している．

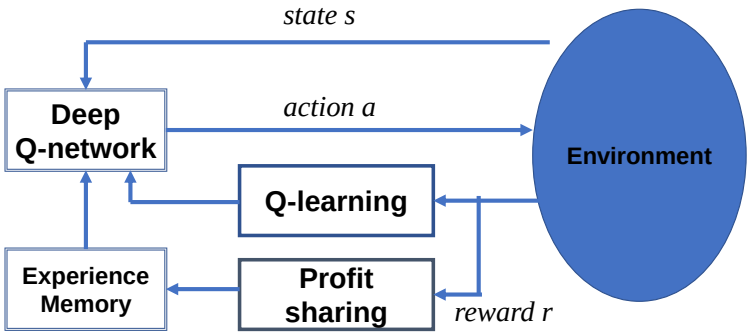


図 5.5: ブロック図

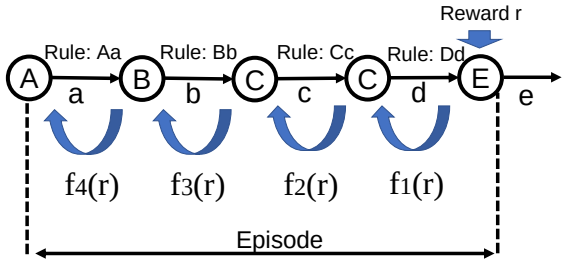


図 5.6: Profit Sharing の報酬の分配

5.3.4 Profit Sharing の TUBSTAP への適用

提案手法では図 5.5 のように DQN システムについて報酬の与え方を Profit Sharing 法 を用いて DQN に適用するために DQNwithPS [69] の考え方に基づいたものを TUBSTAP に応用できるように変更し報酬が与えられた時点で報酬を 1 としそのエピソードを Experience Replay のリプレイメモリへと格納する．TUBSTAP の 1 つのマップのゲーム開始から終了までを 1 エピソードとみなすことができる．エピソードが終了した際に Profit Sharing の強化関数に基づいて報酬を分配する（図 5.6）．TUBSTAP の各ターンにおける局面と駒の特性が状態であり，各状態におけるルールが行動となる．ニューラルネットワークの Q-network には局面と駒の情報が与えられ行動を出力しているものとみなす．したがって Q-network は各局面におけるルールを経験に応じて個別に学習していくことになる．

採用した強化関数

TUBSTAP における各ステップはどの局面においても価値はほぼ同じであると今回は仮定して次式のように設定した．

$$f_i(r) = 1 \tag{5.5}$$

この強化関数の設定による報酬の与え方では，対象問題によっては無効ルールが獲

得されるといった不都合な学習が為される可能性があるが，本実験では用いるマップの特性を考慮し，深層学習の汎化能力に期待してこのような設定とした．

5.4 実験設定について

ここでは提案したニューラルネットワークの出力段の設計手法について確認するための性能評価と上記に加えて Profit Sharing 法を適用した場合の強化学習の性能について DQN と比較した数値実験を行う．

5.4.1 環境（Environment）の設計

ここで環境プログラムの入力となるのはシステムの状態 s_t ，次の行動 a_t であり，出力となるのは次のシステムの状態 s_{t+1} と報酬 r_t である．s_t はマップの地形情報と駒の位置情報や HP の情報が，a_t には駒の現在位置情報と次の移動先と攻撃先情報が入っている．ここでは 1 つの状態から次の状態に至るまでで 1 ステップと呼ぶことにする．ひとつのマップにつきマップごとに決められたターン数に達するかどちらかの駒が全滅することでゲーム終了となる．これを 1 エピソードとしている．

対戦相手側の方策設計

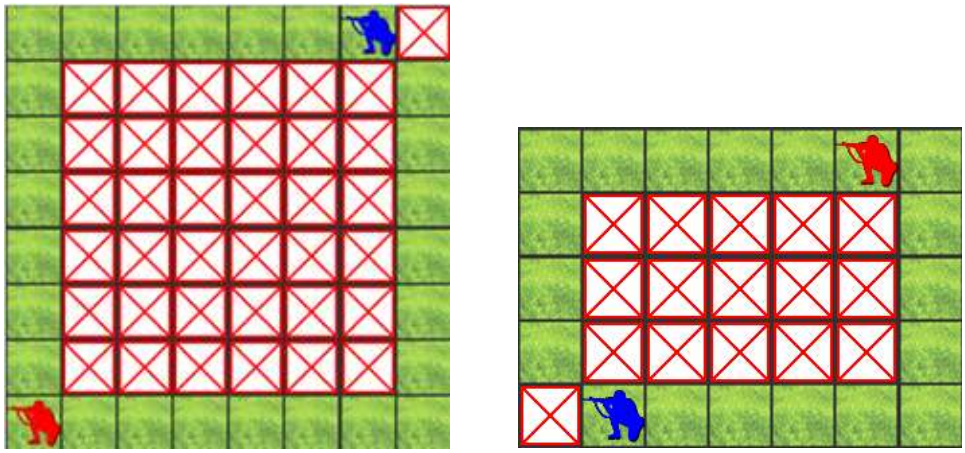
学習する側と対戦する側の駒の動作は環境に含まれている．今回使用するマップは簡単であるため容易に追跡・逃走プログラムを設計できる．これを最適方策 $\pi*$ として採用し相手側の駒の動作として常に使用している．この方策はマップ設定で相手の HP と自らの HP を比較して敗北する場合は逃走し，勝利する場合は追跡して可能なら攻撃する．逃走する場合はできる限りターン数が長くなる方向で道のり距離が遠い方向へと移動し，追跡する場合は道のり距離が最短の方向に移動する．先読み探索は行っていない．複数の最適経路がある場合はランダムに選択する．

報酬の与え方

エピソードに対して報酬を与えるのはマップにより規定されたターン数以内でゲームが終了した場合で

- HP の比較から戦力的に勝てるマップ設定であってかつ，相手を全滅させた
- HP の比較から戦力的に負けるマップ設定であってかつ，自軍が全滅しなかった（逃げ切った）

の時に，与えられる報酬は 1 となる．上記以外の場合は報酬は 0 となる．



(a) run 8 × 8　　(b) run 7 × 5

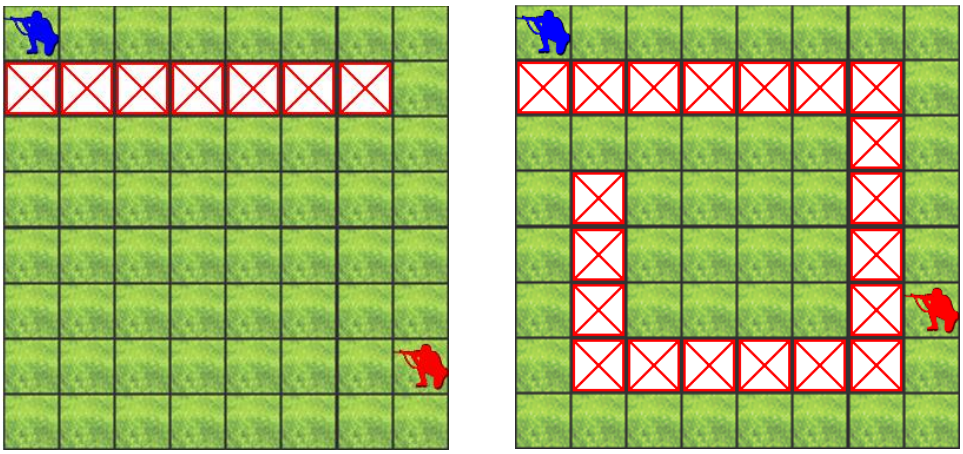
図 5.7: 追跡・逃走マップの例

5.4.2　評価に使用したマップ

本章で使用したマップは駒の挙動や正解を確認しやすくするため，すべて 8 × 8 のサイズとし，使用する駒は RED が歩兵一個，BLUE が歩兵一個に限定した．駒の数を一個に制限することで解析しやすくなるがゲームの持つ複雑性は制限されるので限定的な条件での解析となる．なお図のマップではすべて RED 側が学習側であるが，RED と BLUE を入れ替えたマップを生成しても学習は成立する．ここでベンチマーク問題に登録されている逃走・追跡マップと異なって経路が一か所進入禁止ブロックで封鎖されている形になっているのは，追跡のケースにおいて逃げ切りが発生しないようにして，プログラムが適切に動作するなら，1 エピソード中に確実に報酬が入るようにするためである．進入禁止ブロックの配置により追跡マップと経路探索マップのテーマ的には類似の動作の確認となっている．

追跡・逃走マップ（run マップ）

追跡・逃走問題は人工知能の研究課題として重要である．ここで用意したマップは図 5.7 のように回転型の経路上を追跡または逃走する設定のマップである．マップの地形は外周が角 1 つを除いて全て平地で四角形の経路のみである．ここでは run マップと呼び，run 8 × 8 のようにマップの横と縦のマス目のサイズに応じて数値で名称を付けている．例えば図 5.7 (a) は横 8 マス × 縦 8 マスのマップである．ここで追跡と逃走としているのはランダムな HP の設定によって，追跡が正しい行動である場合と逃走するのが正しい行動である場合の二種類が存在し，エージェントはそれを適確に判断しなくてはならないためである．



(a) pathfind01　　(b) pathfind06

図 5.8: 経路探索マップの例

表 5.1: 評価に使用したマップ

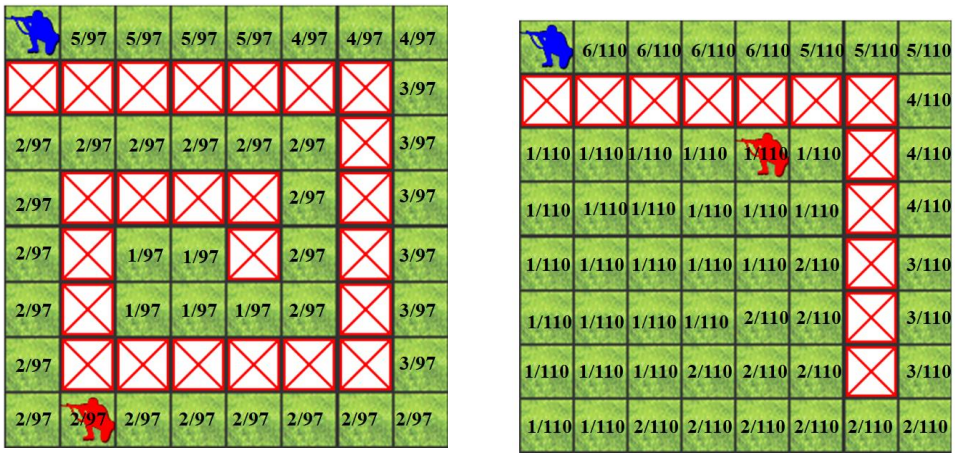
学習用（train）	検証用（test）
8 × 8, 8 × 7, 8 × 6, 8 × 5, 8 × 4, 8 × 3 7 × 7, 7 × 6, 7 × 5, 7 × 4, 6 × 6, 6 × 5, 6 × 4, 6 × 3, 5 × 5, 5 × 3, 4 × 4, 4 × 3, pathfind01, pathfind02, pathfind03, pathfind05	7 × 3, 5 × 4, pathfind06

経路探索マップ（pathfind マップ）

経路探索もまた人工知能やロボット研究などでよく扱われる課題である．ここで用意したマップは図 5.8 のように障害物となる壁を避けた経路を探索して相手に到達できるかどうかを課題としている．ここでは壁の形を変えた種類を用意した．

学習用マップと検証用マップ

用意したマップをまとめたものが表 5.1 である．ここでは学習用（train）に使用するマップと検証用（test）に使用するマップを明確に区別して使用する．各グラフ上で使用したマップは学習用が _train，検証用が _test を付記したものになる．学習と検証とも選択されるマップはランダムに選択される．追跡・逃走マップについては経路長が長いほど出現確率が高くなるよう調整している．



(a) pathfind02　　(b) pathfind03

図 5.9: 駒出現確率分配データ

駒の配置について

相手側の駒は左上の隅のマスに固定して配置される．経路探索マップにおいては学習する側のエージェントの駒は図 5.9 のようにマップごとに決められた確率に応じてランダムに配置される．基本的には近いほど重要度が高いとして高い確率に定め，さらに重要性の高いマス目に多く駒を出現させることで学習を速めている．追跡・逃走マップの駒の配置確率はマス目ごとに等確率である．

駒の HP と勝敗について

駒の HP は自分も相手の駒も 1 から 10 までのランダムな整数値が選ばれる．つまり HP の組み合わせは 10×10 の 100 通りとなる．駒が戦闘した場合は TUBSTAP のルールに基づいて確定的に勝敗が決定される．

制限ターン数について

各マップにおいては制限ターン数がそれぞれ設定されている．制限ターン数の決め方は極端にエージェントが冗長な行動を取ることにならないようにするためにターン数が長くならないようにかつ，短すぎてマップを成功させることができないことがないように設定する必要がある．この場合は，各陣営の駒の配置が決定した時点で双方が最善を尽くした場合にマップの課題を成功させるのにかかるターン数を計算して設定している．ターン数の設定は数ターン程度の余裕が含まれるため冗長な行動をわずかに許容している場合もあるが，最善の行動をしている限り成功に至るようになっている．

Data Augmentation

マップデータの多様性を確保するため，マップを生成する過程においてData Augmentation（データ拡張） [28] と呼ばれる手法によってデータ量を増加させている．具体的にはマップを生成時に鏡映および回転をランダムに行ってマップと駒配置を変化させている．

5.4.3 使用したソフトウェアとハードウェア

今回の研究に使用したソフトウェアとハードウエアは以下である．

- ソフトウェア：Python 3.6, Keras 2.2
- ハードウェア：CPU Intel i7 3.4GHz, GPU NVIDIA GTX1050Ti

また学習に使用したプログラムはOpenAI Gym[72] におけるDQNプログラムをもとに改変を加えたものである．

5.4.4 学習の進展の評価法について

学習の進捗程度の評価法を考える．

今回の実験設定ではエピソードが規定のターンまで到達するか，相手を全滅させた段階になって初めてエピソードが成功したか，失敗したかが判明するようになっている．したがって，出力された行動が正しいかどうかはエピソードの途中では判断することができない．また，行動出力が複数ある場合においては正しい行動がただ一つのみであるとは限らず，複数の行動が同様な正しい結果をもたらすことがあるため，正解ラベルを用意して学習したり評価することが困難である．

そして対戦相手として設定したAIである最適方策$\pi*$はこのマップにあわせて設計され，このマップに限ってではあるものの最適化されている．このため対戦結果は対戦相手の強いか弱いかなどによる影響を受けないことになる．この条件により評価するAIの性能は対戦相手の性能によらず絶対評価が可能となっている．

このようなことから，学習の進展度合いとしての絶対評価もできるエピソードの成功率を測定することにした．引分けが最善である設定のマップもあるため勝率とは呼ばず，成功率とここでは呼んでいる．

成功率による評価

エピソードが成功するためには当然ながら合法手が出力されていることが前提となる．しかし出力手が合法手であるからといって，それが各局面で正しい勝つための行動であるとは限らない．

そこで成功率による評価を考える前の評価としてまずは合法手を出力している割合 (合法手出力割合) での評価も行う.

同時にエピソードが成功した割合（エピソード成功割合）でも評価を行うことで複数の観点からの学習の進展の様子を確認できるようにした.

5.5 ニューラルネットワークの性能評価

設計した強化学習のシステムにおいて，双方のエージェントを同一方策を用いて強化学習を行い，学習用のデータを作成し，このデータによってニューラルネットワークを学習させることで，学習の性能評価を行う．つまり環境と二つのエージェントを通じてデータを作成して，ニューラルネットワークを学習させる.

設計したニューラルネットワークの性能を比較するために，単純な分割出力した場合と，ワンホット出力形式の場合について同様な方法で学習しデータを示す.

5.5.1 設定と実験結果

比較するネットワークとして次のものを用意する.

- 図 5.4 の RNN のもので，出力は時分割出力で移動元 65+ 移動先 65 + 攻撃先 65（RNN_）
- 図 5.4 の RNN の部分を 3 層の全結合層に置き換え（CNN + Fully Connected Layer）かつ，出力を移動元 64 +移動先 64 + 攻撃先 65 の 3 分割形式にしたもの（CNN_）
- 図 5.4 の RNN の部分を 3 層の全結合層に置き換え（CNN + Fully Connected Layer）かつ，出力を移動先 64 × 攻撃先 65 = 4160 のワンホット表現で出力するもの (One_).

上記の一つに集約したワンホット表現出力の場合は本来なら $64 \times 64 \times 65$ のサイズのニューロンが必要だがこのサイズでは GPU にのらないため移動元の出力を省略したタイプに変更し 64×65 のサイズとした．本来なら機能が異なるものであるが比較のために掲載する.

使用した方策は学習する側も対戦相手と同様に最適方策 $\pi*$ として双方が最適方策の状態でデータを生成して学習した.

実験結果が合法手出力割合が図 5.10 でエピソードの成功割合が図 5.11 である． 1 イタレーションあたりに 4000 ステップ実行している.

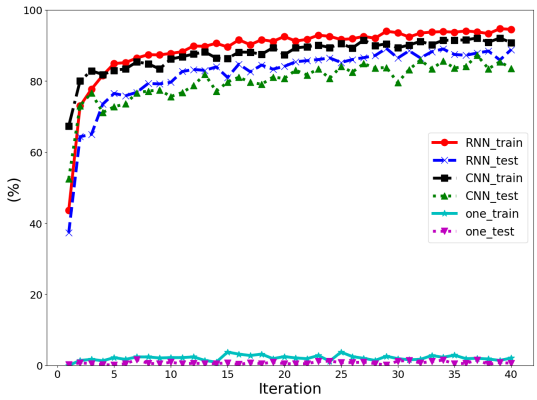


図 5.10: 合法手出力割合

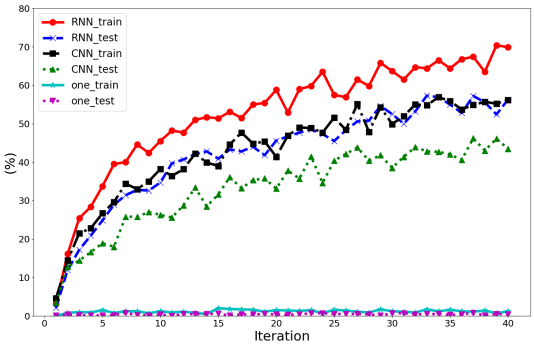


図 5.11: エピソード成功割合

5.6 強化学習

DQN のシステムに Profit Sharing を組み込み，学習をさせて Profit Sharing 法と DQN について比較をする．さらに，DQN に優先度付き経験再生（Prioritized Experience Replay） [73] の思想を参考にしてリプレイメモリからの取得確率をすべて等確率ではなく，報酬が得られた時は取得確率を高くした場合のデータも比較する．

5.6.1 設定と実験結果

探索方策は ε-Greedy 法であり，これは通常は Q-network の Q 値の最大値に基づいた行動選択を行い，確率 ε でランダムな合法手を打つものである．ここでは $\varepsilon = 0.25$ とした．もし Q-network の行動出力が合法手ではなく TUBSTAP のルール上実現できないものであった場合，報酬は -1 となり，ただちにエピソードを終了してそこまでの情報はリプレイメモリへと格納されない．報酬が 0 の場合もリプレイメモリへと情報は格納されない．ε-Greedy 法は学習時にのみ動作し，検証時には動作しない．検証時では Q 値のみによって行動決定をしている．

実験結果は，合法手出力割合が図 5.12，エピソード成功割合が図 5.13 であり，Profit Sharing が PS_train と PS_test，DQN が DQN_train と DQN_test がそれぞれ学習と検証データの精度を表している．ここでは 1 イタレーションあたり 20000 ステップ実行している．

さらには通常の DQN に優先度付き経験再生（Prioritized Experience Replay） [73] の機能に対応させることで学習能力を向上させたものとも比較する．優先度付き経験再生は基本的には TD 誤差に応じてリプレイメモリからの取得確率を変更するものであるが，ここでは仕様を変更し，報酬があったエピソードについてリプレイメモリから選択される確率を 4 倍にすることで学習を加速している．グラフでは Ex_train と Ex_test と記載している．

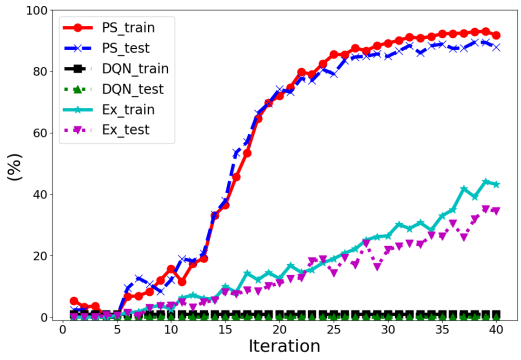


図 5.12: 合法手出力割合（強化学習）

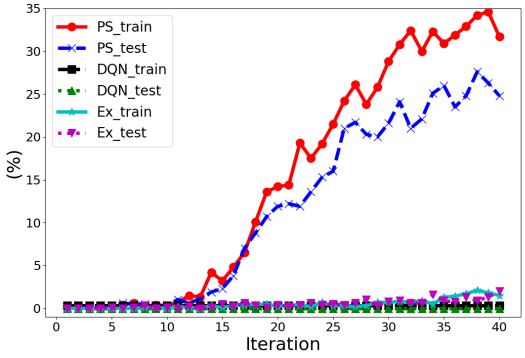


図 5.13: エピソード成功割合（強化学習）

5.7 考察

ニューラルネットワークの設計においては RNN で設計した場合が CNN で設計した場合に比較して成績が上回った．また，RNN と CNN の双方のネットワーク設計のどちらについても，検証用に使用されたマップは学習用マップと似ているものではあるが、学習中にはまったく使用されていないにもかかわらず，検証用のテストでも高い合法手出力率とエピソード成功率を示した．これは深層学習による高い汎化性能による未知のマップへの対応能力によるものと言える．

出力をひとつに集約した場合は学習速度的が遅く精度的にも低い成績となった．これは出力のニューロン数が多くなりすぎたために学習の効率が低下しているためと思われる．RNN の設計が CNN をより良いのは，RNN において時系列出力として t=1 の出力を t=2 にて活用し，t=2 の出力を t=3 で活用するというように相互に関連性を持たせたことで学習の精度が高くなったものと推定される．

強化学習のデータについては DQN の設計のままでは学習が進まなかった．これはニューラルネットワークが初期状態からでは報酬が得られるケースが少なすぎてスパースな報酬の状態となり学習が進まないものと思われる．DQN に優先度付き経験再生を付けた場合では 40 イタレーションで Ex_train の合法手出力割合が 40 % 程度まで上昇し，エピソードの成功割合もわずかにゼロから上昇している．これは DQN だけでは学習が進んでいないものが，優先度付き経験再生による加速効果による上昇と思われる．

本章の実験では 式 (5.5) の強化関数を用いて学習が成功しているが，常にこれが最善の手法であると主張するわけではない．この実験では制限ターン数の設定をエピソードを成功させるために充分なレベルで設定しているため，冗長な成功経路が得られることはまれであり，また駒同士が近い配置を高い確率で与えていることにより，もし仮に冗長経路が発生しても十分に上書きされるために打ち消されることが，学習が成功した理由の一部と思われる．減衰型の強化関数を採用した場合や制限ターン数を大幅に緩めた場合にどのようになるかは今後の課題である．

5.8　おわりに

複雑なデータ構造を持つゲームに深層学習を適用する上でのニューラルネットワークの出力段についての新しい提案を行った．同時に少ないデータ量でスパースな報酬における強化学習ができる手法について検討しすばやい立ち上がりができることを確認した．

第6章　棋譜からの学習によるポリシーネットワークの設計

この章では，前章で説明したRNNを使用した深層学習と強化学習をターン制戦略ゲームに適用した手法をさらに応用して，既存のプログラムを用いて大量のゲーム記録（棋譜データ）を作成して学習することでpolicy networkを設計する手法について述べる．

前章で取り組んだRNNを使用した行動を表現する出力段設計を生かしてさらに取り扱うユニットの数を増加させ探索木の幅を広げ「幅広探索木問題」について検討するための基礎的な検討を行う．

この章は16th Advances in Computer Games Conference (ACG 2019) [74] において発表された内容を骨子としている．

6.1　ゲームにおける棋譜学習

ゲーム記録すなわち棋譜データを学習して評価関数の学習やプログラムの最適化に使用することは過去のゲーム研究でしばしば行われてきた．

将棋では棋譜データを学習して実現確率探索 [15] の遷移確率を求めたり，評価関数の機械学習 [21] に使用されたりしてきた．

囲碁は評価関数を作るのが難しいとされていたがデータベースへの一致率で2015年にDNNを使用して44% [75] まで進歩していたところへ，2016年のAlphaGoのSL policy network [1] が優れたニューラルネットワーク技術によって57%にまで性能を向上させた．画像処理に強いDNNを使用することで囲碁の盤面を画像として認識し特徴量を抜き出して，設計が難しいとされていた囲碁の評価関数の性能を高めることに成功したといえる．

6.2　棋譜学習

本章では，棋譜学習によってニューラルネットワークを訓練し，性能を確認する．本章の研究は，次の3つの手順で構成される．最初は，試合記録データベース，つまり棋譜の作成で，2つ目はニューラルネットワークの学習，3つ目は作成されたニューラルネットワークの性能の検証である（図 6.1）．

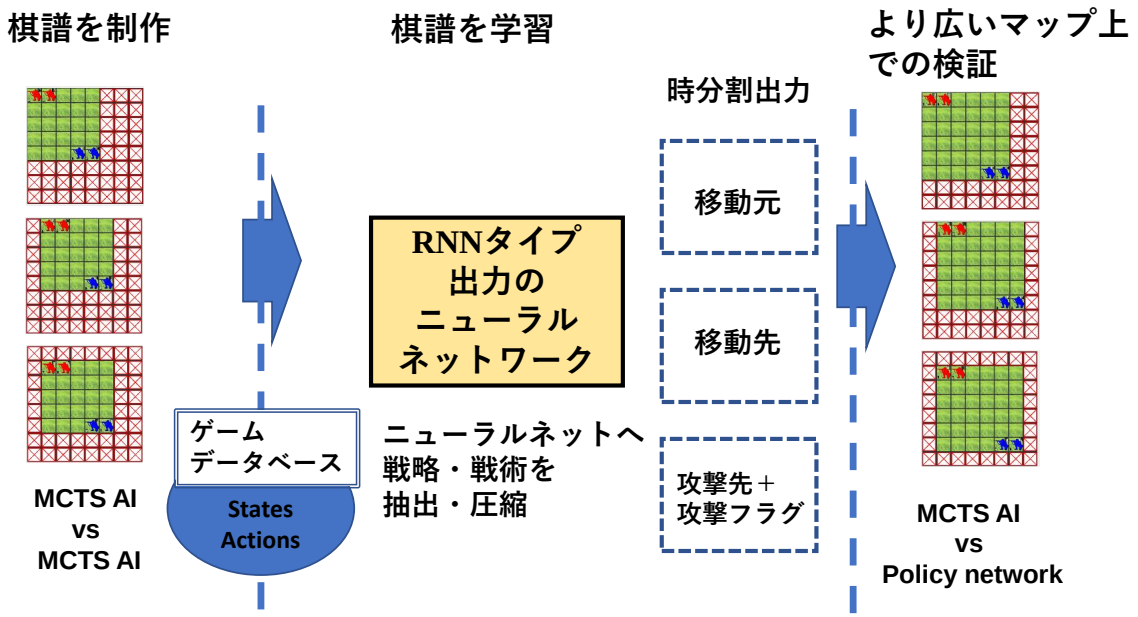


図 6.1: 実験手順

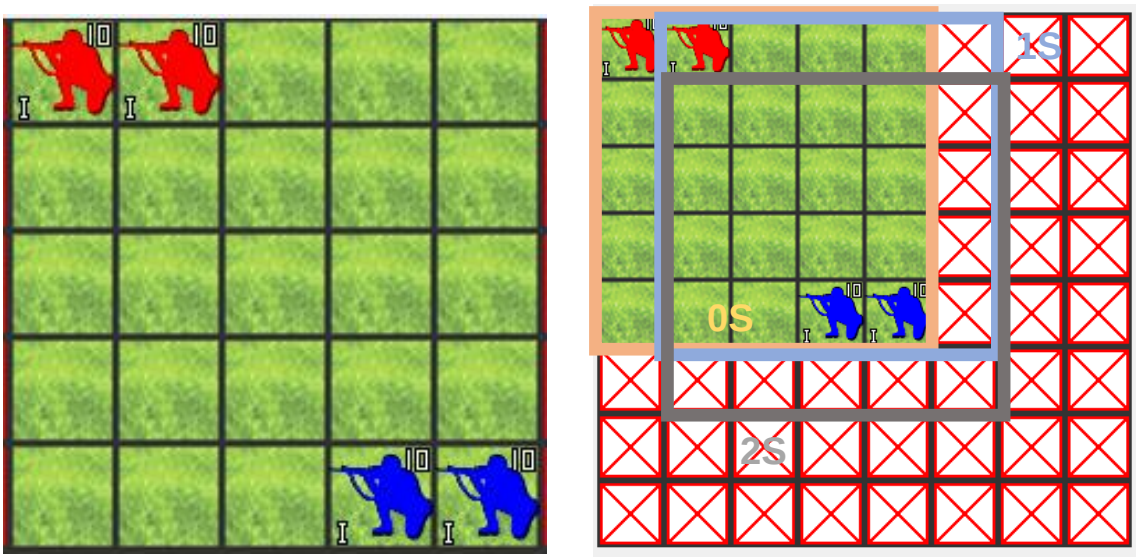
深層学習によって，データベースからゲームプレイに必要な戦術と戦略を抽出して圧縮し，ニューラルネットワークに貯蔵することを目的としている．ここでは戦術・戦略とはゲーム中における一定の手順を指している．深層学習の学習汎化性能を確認するために，学習用のマップだけではなく，より広いマップでの性能の検証も試みる．

試合が行われるのは．障害物のない平野での歩兵間の戦いであり，歩兵は RED と BLUE に各 2 個である．前章に比較してユニットの数が 1 個から 2 個へと増加した．ニューラルネットワークの標準化を意識して，マップ全体のサイズは 8×8 マスに固定した．ただし，戦闘を容易にするために戦場は 5×5 マスに縮小した．

6.2.1 棋譜学習用データベースの作成

棋譜学習用のデータベースの作成手法は，次の 2 つである，一つ目は TUBSTAP で用意されている AI エージェントを使用して，学習マップの初期状態から対戦を記録して試合データを作成する．二つ目は同じ AI エージェントを使用して，多数の後述する手筋マップから出発して追加の試合データを作成する．これは戦術的な分析を行い，これらの局面を優先的にデータを与え効率的にニューラルネットに学習させるためである．

試合データには各局面におけるマップデータと RED/BLUE ユニットの情報と手番データ，そしてユニットの行動データ（移動元，移動先，攻撃先）が格納されている．ゲームが終了すると，勝者側の局面情報と行動のみがデータベースに格納され，優れた経験のみを残すようにしている．試合に使用された M-UCT は，2016 年のゲーム AI トーナメント（GAT) [58] で優勝した AI プログラムであり，M3Lee は同じ大会で



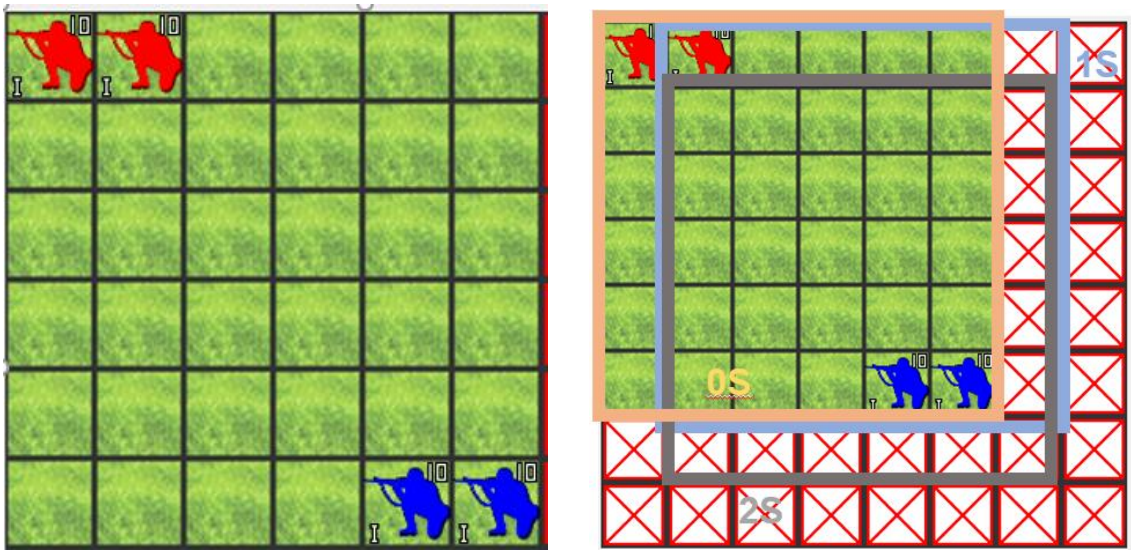
(a) sq5x5 基本設定　　(b) 基本設定の配置位置

図 6.2: 学習用マップ

2 位になったものである．また，自作のアルファベータ検索に基づくツリー検索アルゴリズムも使用された．M-UCT はゲームの終局まで 5000 回のシミュレーションを実行しまた，M3Lee は 3000 回のシミュレーションを実行する．実際に作成されたデータベースのサイズは，試合局面の数で約 800,000，試合数で約 100,000 である．これは手筋マップによるものを含んでいる．これは今回用意したマップ上での試合では，おおよそ 6 局面から 10 局面程度で決着がつく場合が多く，結果として試合あたりの平均局面数がおよそ 8 局面程度であることを意味している．

学習マップ

学習用マップは，図 6.2 に示すように，5 × 5 マスの正方形のセルで囲まれた平野となっており，図 6.2(a) は学習マップの初期設定を示している．さらにニューラルネットワークのバイアスを軽減させ多様化した配置を学習させるために，0S オプションは左上に配置し，1S オプションは 1 つマスを右にシフトした配置で，2S オプションは 1 マス右下にシフトした配置（図 6.2(b)）となっている．単純な地形上に歩兵ユニットが 2 個しかないため，一見簡単に見えるマップではあるが，後述するような勝つために少し難しい戦術や手筋が必要になる場合があったり，移動先の数が多いために MCTS のシミュレーションがうまくいかない場合があり，常に正しい行動を取るのが困難のあるマップである．障害物があってユニットの移動先が少ないマップの方が MCTS アルゴリズムにとっては読む量が少なくなるためミスが出にくいマップである．つまり画像としては簡単で地形の効果が入らないので囲碁に比較的近く，アルゴリズム的には難しいレベルとして，これらのマップを最初のステップとして選択した．



(a) sq6x6 基本設定　　(b) 基本設定の配置位置

図 6.3: 検証用マップ

検証マップ

ニューラルネットワークを学習した後に，ニューラルネットワークの汎化性能を検証するためのマップが検証マップであり，図 6.3 に示すように学習マップよりサイズが大きくなっておりその分行動決定が困難になっている．検証マップは学習には使用されない．学習マップと同様に基本配置（図 6.3(a)）があり，これを 0S,1S,2S とオプションによって配置をシフト （図 6.3(b)）させている．

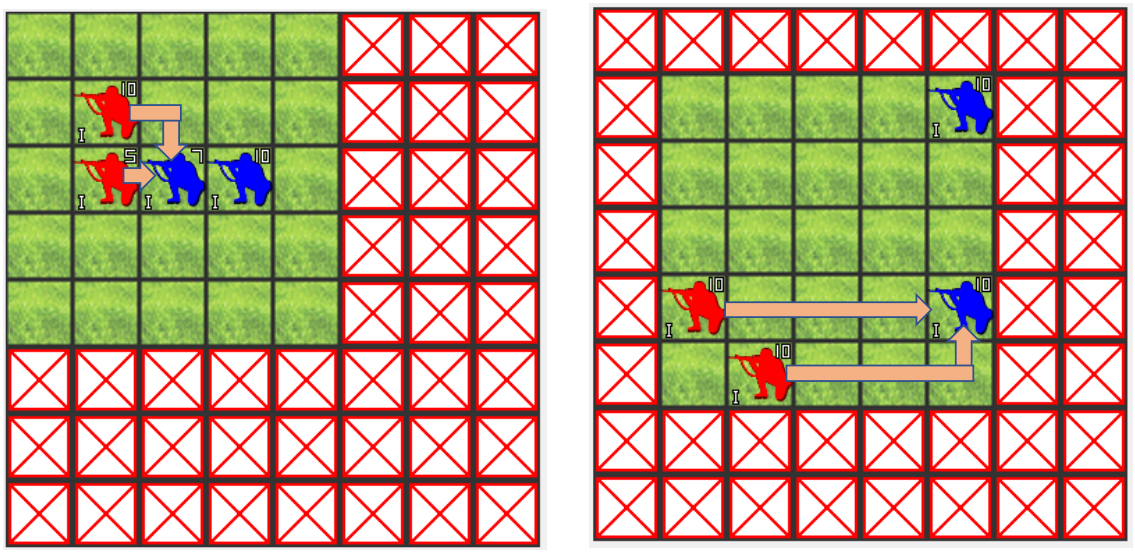
手筋マップ

図 6.4 は，手筋マップの戦術の手順の一部を示している．矢印で示されている攻撃は，相手の HP を効果的に低下させ，ゲームに勝つ可能性を高める．このような戦術手順の 30 以上のパターンを検討し，40 万局面を超えるデータを手作業で準備した．

図 6.4(a) においては RED の歩兵が 2 個とも BLUE の HP=7 の歩兵を攻撃するのが効率的である．図 6.4(b) では RED の歩兵は 2 個とも BLUE の歩兵を攻撃するのが効率的に HP を奪える手順である．

6.2.2　ニューラルネットワークの設計

ターン制戦略ゲーム用のニューラルネットワークの設計は，地形やユニットの特徴などの多くのパラメータがあるために複雑で困難なことが多い．ニューラルネットワークを policy network として使用するには，出力をユニットへの行動指示として



(a) tesuji1　　(b) tesuji3

図 6.4: 学習に使用した手筋例

表現する必要がある．ここでは，複雑なデータ出力が不可欠なターン制戦略ゲームに適した高性能な設計を提案する．

ニューラルネットワークの入力データ構造

本章で設計されるニューラルネットワークへのすべての入力チャネルは，7 層の地形データ，14 層のユニットデータ（HP と配置を入れたユニットのタイプ 6 層に加え動作済みフラグ 1 層の計 7 層を各 RED/BLUE にそれぞれ割当て），および 1 層の手番データにエンコードされる．

ニューラルネットワークの出力データ構造

TUBSTAP でのユニットの動作を説明するための必要な最小の情報は，ユニットの移動する前の元の位置，移動する先の目的地，攻撃ユニットの位置と攻撃をするか移動のみかを表現するフラグになる．ニューラルネットワークはこれらの信号を出力する必要があるが，従来からあるワンホット出力表現を行うには膨大な数のニューロンが必要になる．8×8 の正方形のマップの場合，$8 \times 8 = 64$ のニューロンに移動元，移動先，攻撃先位置が乗算されるため，少なくとも $64 \times\ 64 \times\ 64 = 262144$ のニューロンが必要となる．これだけの数のニューロンを含む設計は非現実的である．したがって，出力ニューロンの数を減らし，リカレントニューラルネットワーク（RNN）を使用して出力を時分割することにより，設計を簡素化することを目指す．設計されたニューラルネットワークの出力構造を，時間方向に展開したものを図 6.5 に示す．移動元を示す $t = 1$ の出力は，入力 $t = 2$ に接続され，$t = 2$ の出力は，移動先を示

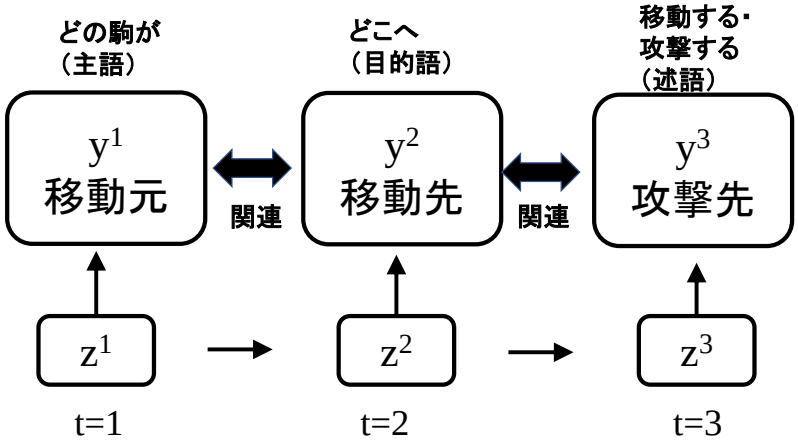


図 6.5: 出力段の RNN による構成

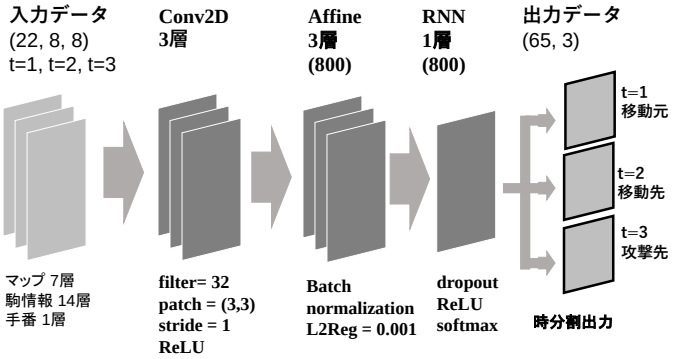


図 6.6: 使用したニューラルネットワーク構成

し，攻撃先を示す $t = 3$ の入力に接続される．このように接続することで各出力間の相関を高め，推論エラーを低減する．

実行速度を考慮して，自然言語処理でよく使用される LSTM ユニットの代わりに GRU ユニット [71] をここでは採用した．GRU ユニットは，$t = 3$ の終わりにリセットされる．ニューラルネットワークの出力は，プログラムによって TUBSTAP の合法手のみの出力となるように出力をマスクしている．

ニューラルネットワークは，1 回につき 1 個のユニットの動作を出力する．もし，2 個のユニットが存在する場合は，2 回の推論を実行する必要がある．

ニューラルネットワークの構成

ニューラルネットワークの構成は図 6.6 に示すようになっており，入力層は 3 層の Conv2D で，次の段が Batch Nomaraization (BN) と L2 正則化を持つ 3 層の全結合層であるアフィン層である．次に，時分割出力のための RNN がドロップアウトと ReLU とともに接続されている．アフィン層と RNN 層では，各層に 800 個のニューロンを使用している．

6.3 数値実験

作成した policy network の性能を評価するために学習に使用した元のデータベースと policy network の出力との一致率を測定し，さらに実際の対戦によって性能を確認する．

6.3.1 学習手順

棋譜を記録したデータベースのファイルサイズは非常に大きく，ファイル全体を PC メモリに一度でロードできないため，バッファメモリを設定し，ここからランダムにデータをサンプリングし，ニューラルネットワークの学習に使用している．この場合，バッファメモリは DQN で使用される Experience Replay の機能 [6] とほぼ同等で，バッファメモリサイズは 12000 試合局面相当分確保されており，イテレーションごとに 1 つのファイルのデータが更新される．この学習手順では，イテレーションごとに 1 エポック，バッチサイズは 2048，最大 8000 イテレーションまで実行される．オプティマイザーは Adam を使用し，損失関数はクロスエントロピータイプに設定した．ニューラルネットワークは，ランダムな重みの初期状態から学習を開始する．したがって，使用するデータベース以外の他のデータや設定は必要としない．

学習率のスケジューリングを実行して，学習プロセスをイタレーション 1000 回まで 10^{-3}，5000 回まで 10^{-4}，および 8000 回まで 10^{-5} と設定することで学習を高速化している．

Data Augmentation

Data Augmentation(データ拡張) [28] は，少ないデータ量を補い，ニューラルネットワークの特定方向へのバイアスを防止するために，データに回転と鏡映を実行するものである．ここでは，回転で 4 倍（0/90/180/270 度）と鏡映の 2 倍（垂直/水平）で合計 8 倍のデータに拡張している．

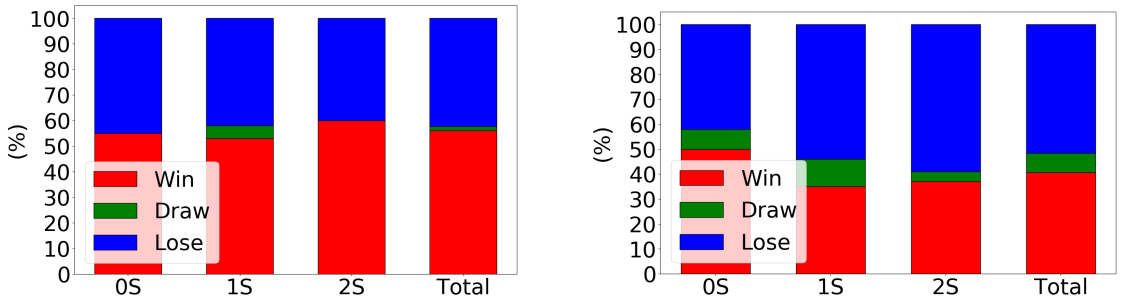
データベースとの一致率

学習の進行程度の評価のために，作成されたデータベースから抽出された 10000 局面と学習されたニューラルネットワークの出力との間の一致率のデータを表 6.1 に示す．この一致率はあくまで一つの局面とその行動出力をデータベースのデータと比較するものである．

なお，以下の対戦実験では 7000 イタレーションのバージョンを最も性能の良いものとして対戦に使用し，8000 イタレーションのものは過学習状態にまで至ったとして採用しなかった．

表 6.1: policy network 出力と訓練データの一部との一致率（一万局面）

Iteration	Accuracy
1000	29.8%
5000	37.6%
7000	59.6%
8000	54.3%



(a) 学習マップ sq5x5　　(b) 検証マップ sq6x6

図 6.7: M-UCT との対戦結果

現実のゲームの試合においては，正解手順は一つとは限らず，多くの正解手順がある場合が多い．したがって，通常の教師付き学習で使用されるような精度や正解ラベルの手法では訓練された policy network の性能を評価するには適切ではない．本研究では 7000 回のイタレーション後に一致率が 59.6% に達したが，これらのデータを適切と言える比較ができないため，ここでは MCTS プログラムとの実際の対戦によって性能を評価することにする．

6.3.2　対戦実験の結果

学習を完了したニューラルネットワークの性能を評価するために，学習マップ（図 6.2）と検証マップ（図 6.3）でそれぞれ 300 試合（各マップにごとに 100 試合）を実行した．対戦相手として，MCTS で動作する M-UCT を使用し，先攻と後攻は前半と後半で交代し先攻と後攻による有利や不利がないように設定した．テストする AI エージェントは常に RED に割り当てられており，先読み探索を行わずに policy network の出力のみによって行動を決定する．学習マップでは，各マップの試合結果は Win/ Lose/ Draw = 55/ 45/ 0 (0S)，53/ 42/ 5 (1S)，60/ 40/ 0 (2S) であり，トータルの勝率は 56.0% である．検証マップでは，各マップの試合結果は Win/ Lose/ Draw = 50/ 42/ 8 (0S)，35/ 54/ 11 (1S)，37/ 59/ 4 (2S) であり，トータルの勝率は 40.7% である．検証マップでの勝率は，学習マップと比較して約 16% 低下しており，引分けが増加している．結果をまとめたものを図 6.7 に示した．

6.4　考察

学習の結果は，ニューラルネットワークがランダムな重みの初期状態から出発して，学習マップの MCTS アルゴリズムに対して勝率が 50 %を超えるレベルに達した．検証マップでは，全く学習に使用していないマップであるにもかかわらず勝率が 40 %を超えた．

本章で作成された AI プレイヤーは MCTS によって作成された棋譜を基にしており，基本的に元の棋譜を作成した MCTS の強さを越えないものと思われる．したがって，学習の進行が良好であったとしても，元の MCTS アルゴリズムと対戦させた場合は五分五分の成績程度になるのが妥当であると考えられるが，上記の結果はその通りに近く，勝率的には学習マップ上では 50 %を超えた勝率となった．元の MCTS に対してより強い成績を修めたのは，ニューラルネットワーク上でのアンサンブル効果によって棋譜からの戦略の抽出に成功し，元の MCTS アルゴリズムの成績をわずかに超えた性能になったためと思われる．

AI エージェントの行動の決定には policy network の出力のみが使用されており，検証マップの形状は学習マップに似ているがサイズが大きく同一のものではない．したがって，この研究で作成されたニューラルネットワークの学習能力は，高い性能であるといえ，本章の目的の戦術と戦略のニューラルネットワークへの抽出に成功したといえる．

試合では，AI エージェントは推論のみで実行されるためディープラーニング側の思考時間は非常に短くなるが，MCTS 側の思考時間は，1 ターンにつき約 5〜10 秒である．思考時間に関しては，この章で採用した方式であれば MCTS 方式に比較して高速で行動出力を行うことができる．

6.5　おわりに

ここでは，深層学習の適用が簡単ではないターン制戦略ゲームで，リカレントネットワークを使用して棋譜学習からの policy network の設計について紹介した．

しかしながら，学習に使用するための棋譜のデータベースを作成する必要があり，また作成されたアルゴリズムは元の棋譜の作成に使用したアルゴリズムのレベルを学習マップ上では越えはしたものの，未知のマップ上では勝率は 50% を越えていない．棋譜による学習はアンサンブル効果によってある程度は元の棋譜より強くなるかもしれないが，大幅に能力の向上が見込めるともいえないことから，さらに能力の向上を目指すには他のアルゴリズムと組み合わせた形で，実現するのが望ましいであろう．

さらに AI プレイヤーの性能の向上を求めるためには，次章にて検討する poliy network と value network の双方を利用して効率的に探索を進めるアルゴリズムが必要である．

第7章　ターン制戦略ゲームへの AlphaZeroの適用と工夫

この章では，AlphaZero によって紹介された手法である policy network と value network を深層学習によって統合し MCTS と組み合わせて探索するアルゴリズムを，ターン制戦略ゲームへと適用する手法について述べ，各種の数値実験によって性能を測定し，その有効性を確認する．

この章において「幅広探索木問題」と「行動データの表現問題」の課題を前章までの研究を参考にしつつ，複雑な行動出力を簡素化し探索木の幅が広がるのを防止する工夫をこの章で導入する．この工夫により AlphaZero アルゴリズムをターン制戦略ゲームへと適用し AI プレイヤーの性能を向上させる．

この章は 2019 年ゲームプログラミングワークショップ (GPW-19) [76] および，12th International Conference on Agents and Artificial Intelligence（ICAART 2020）[77] において発表された内容を骨子としている．

7.1　提案システム

本章では AlphaZero によって示された手法である，深層学習を使用して policy network と value network を統合し MCTS と組み合わせて探索と強化学習を行うアルゴリズム，をターン制戦略ゲームへと適用する提案を行う．

基本的なアルゴルズムは AlphaZero に準拠するが，ターン制戦略ゲームに適用する上で変更や追加する部分についても説明する．

7.1.1　これまでの手法の問題点と AlphaZero の導入

第 5 章で強化学習アルゴリズムとして検討した DQN は，得られる報酬が疎である場合，学習が進まないという問題があった．第 6 章において採用した RNN タイプ出力はニューラルネットワークの出力を複数にしようとすると，RNN 型出力と他の単出力が混在することになり時系列の管理など複雑な設計になってしまうことに加え，後述するような大型のニューラルネットワークで RNN を導入すると大量の演算量が必要となることが難点である．

このような設計上の課題の解決に加え,「幅広探索木問題」や「行動データの表現問題」についての研究を進めるため AlphaZero アルゴリズムの適用を検討する.

AlphaZero のアルゴリズムを採用することで DQN では必要だった報酬を得られるようにするためのマップの変更や報酬設計の労力を削減できる．また，DQN では対戦相手を用意しなければならなかったが，用意する対戦相手のクセや特徴に学習が左右されやすかった．AlphaZero 方式では自己対戦によって学習を進めるため，対戦相手を用意する必要がなくなる．囲碁や将棋であれば過去の研究から大量の棋譜データの蓄積があるがターン制戦略ゲームではそのように活用できるデータがないことも研究上の課題のひとつであったが，自己対戦で学習するシステムであれば棋譜は自動生成されるのでこの点で大きな改善と言える．

しかしながら，AlphaZero アルゴリズムを単純に適用するだけでは性能が向上するとは保証されない．元々の AlphaZero システムが題材としていたのがチェス，将棋と囲碁といった研究が進んでいる分野であったが，ターン制戦略ゲームでは研究の蓄積といえるものが少なく，参考データも豊富とは言えない．このようななかで，ターン制戦略ゲームに AlphaZero に適用するための変更を盛り込みつつ，過去の深層学習をターン制戦略ゲームへ適用した事例を考慮しながら性能向上を図る．

具体的な適用上の検討項目としては，巨大な行動空間における探索を効率的に行い高速化するためのバイアスの導入や，ニューラルネットワークの設計における，大きすぎる出力段設計の負担を軽減するためのユニット選択手順の変更，マップの局面の多様性を効率よく学習するためのデータの導入などがある．

自己対戦で自ら強くなっていく思考アルゴリズムがより正確で高度な判断ができるようになることで，旧来のアルゴリズムでは正しく動作して正解にたどり着くことが困難だったマップで適切な動作ができるようになったり，強いプレイが行えるようになること，結果として，ターン制戦略ゲーム研究において選択できるアルゴリズムの範囲が広がることを示す.

各種パラメータの比較で新しく変更したことによる性能向上や旧来の MCTS アルゴリズムとの対戦で良い成績を修めることを確認する.

7.1.2 基本システム

今回提案する学習システムのシステム構成について説明する．システムは大きく分けてニューラルネットワーク部，TUBSTAP のゲームを実現する部分，自己対戦や探索などのアルゴリズムを実現する部分，対戦プログラム等に分けられる．

AlphaZero のシステムと比較して同一の部分は，ロールアウトを使用しない MCTS 探索部分，着手決定にニューラルネットワークの確率出力を利用して自己対戦のみで学習を進める部分，policy network と value network を統合した部分が主要な部分である．

異なる部分は TUBSTAP に特有の部分として，駒の行動出力を表現する policy network の出力表現にかかわる部分，探索の高速化のための PUCT 式にバイアスを加え

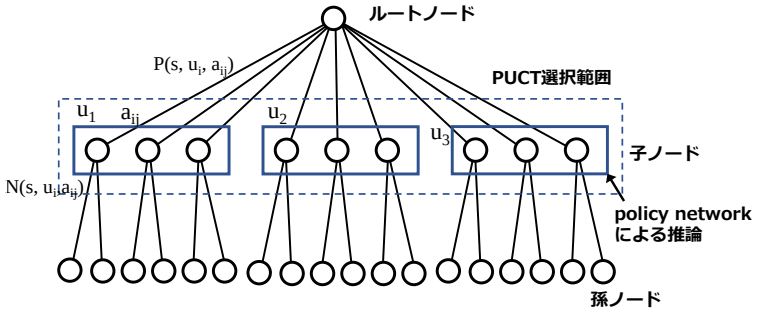


図 7.1: ノード展開の基本概念

た部分等である．なお，AlphaGo のアルゴリズムを彼らは APV-MCTS と呼んでいたが，ここでは A（非同期）の要素がないので policy と value の要素がある AlphaZero タイプのアルゴリズムの呼称として PV-MCTS と呼ぶことにする．

7.1.3 探索アルゴリズム部

ここでは TUBSTAP の一つの駒 u_i ごとの指し手を行動 $a_{i,j}$ とし各局面を表現するノードは $a_{i,j}$ を表現するエッジから次の局面のノードに遷移するものとする．本研究のノード展開においては図 7.1 のように選択されたユニット u_i ごとに子ノードが展開されユニットの選択情報はノードの縦の接続で表現される．

オリジナルの AlphaZero では，読もうとする局面のデータをニューラルネットワークへの入力とすることでその局面の合法手すべての選択確率を一度に求めていた．しかし，この方式をターン制戦略ゲームにそのまま用いようとすると，出力空間が大きくなりすぎることになることは前章までで述べた通りである．そこで本研究では，ユニットの指し手行動を表現するのに必要な行動する駒の（移動元）×（移動先）×（攻撃先）の三つのマスの位置の情報のうち，移動元の情報をノードの入力側に移動することで，ニューラルネットワーク部の出力ニューロン数を削減する．この適用によりどの駒を移動させるかの情報をエッジ部分に含めることになる．ニューラルネットワークでの推論は各ユニット u_i ごとに自軍のユニット数の回数行い，選択確率 $P(s, u, a_{i,j})$ は各エッジに対応する．つまり自軍ユニットが n 個ある場合は，ニューラルネットワークによる推論は n 回分行われることになる．例えば，図 7.1 では policy network によって u_1 を選択した際の選択確率 $P(s, u_1, a_{1j})$ が推論され，しかる後に u_2 を選択した際の選択確率 $P(s, u_2, a_{2j})$ が続いて推論される．エッジにはユニット情報である u_i を含む形になっている．

ユニット数が 2 個の例

例として，ユニット数が 2 個の場合のユニットの行動選択の例について図 7.2 で説明する．ユニット数が 2 個の場合は，第 1 回目の PUCT 選択によってユニットの行

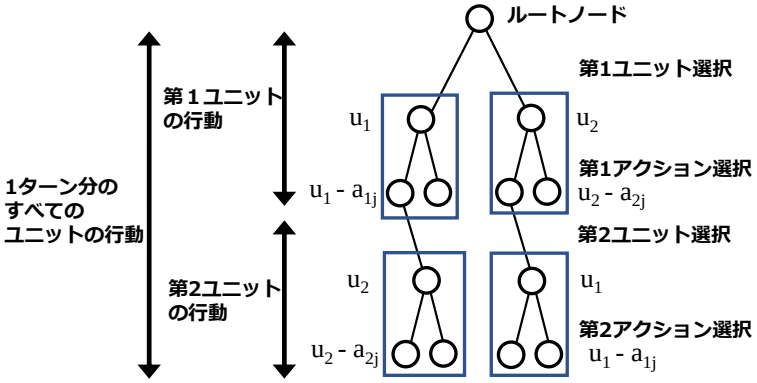


図 7.2: ユニットが 2 個の場合のノード展開の例

動としてユニット選択とユニットのアクションが決定される．この決定がなされた後に第 2 回目の PUCT 選択がおこなわれ第 2 のユニットが決定しかつアクションも決定される．ここではもし，第 1 ユニット選択に u_2 であった時，第 2 ユニット選択では u_1 がアクションと共に選択されることになる．

ユニット選択手順

このように順番まで含めてユニットの選択が行われるため，このアルゴリズムにおいてもユニットのすべての順列についての探索が行われることから，アルゴリズムの一般性は損なわれない．この本研究におけるノード展開に移動元の情報を集約することで，ユニット選択上の順番の問題について一般性を失わずに，行動表現の簡素化を行うことができる．この手法はニューラルネットワークの出力部分の設計の負荷を減少させるという点では優れているが，ユニット選択手順が探索木の PUCT 選択とともに行われるために，ユニット数分のニューラルネットワークによる推論を行わなければならず，ユニットの個数が増加するにともなって実行速度が低下する傾向がある．

基本アルゴリズムにおいては多くの点において AlphaZero と同一とした．基本的な点にとしては，探索は深層学習による policy と value の二つを使用した MCTS 型探索を行う．MCTS 探索ではニューラルネットワークの policy network の出力である確率分布ベクトル $\mathbf{p}$ に基づいた確率探索を行い，最終的に最も探索回数の多いノードを着手とする．MCTS でよく使われるロールアウトは使用していない．自己対戦時にはあらかじめ展開されたノードの再利用を行っている．

PUCT 値の選択

PUCT 値の選択にユニットの選択も含まれる形となっており，着手可能ユニット数が一個の場合は通常の PUCT 探索と同一のアルゴリズムとなる．探索中のノードの選択はすべての着手可能ユニットの着手可能手のなかから次式で計算される PUCT

値の最も高いノードを探索する．

$$PUCT(s,u_i,a_{ij}) = Q(s,u_i,a_{ij}) + Cpuct \cdot P(s,u_i,a_{ij}) \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,u_i,a_{ij})} + B(s,u_i,a_{ij}) \tag{7.1}$$

$$a_{i,j}, u_{i,j} = \mathrm{argmax}_{a,u}(PUCT(s,u_i,a_{ij})) \tag{7.2}$$

ここで$Q(s,u_i,a_{ij})$はノードの勝利確率，$Cpuct$は探索補正係数でここでは0.8の固定値，$P(s,u_i,a_{ij})$は各ノードの選択確率，$N(s,u_i,a_{ij})$はノードの訪問回数，$B(s,u_i,a_{ij})$は探索の高速化のための本研究独自のバイアス項であり次のようになっている．

$$B(s,u_i,a_{ij}) = \frac{b_{attack}}{1+N(s,u_i,a_{ij})} \tag{7.3}$$

b_{attack}はノードが攻撃ノードの場合，3.7として攻撃ノードが優先的に選択されやすい設定としており，学習時とプレイ時の両方で常に有効である．このようにアルゴリズムがPUCT値による選択を行うと行動と同時にユニットも選択されるようになっている．

ノード設計はニューラルネットの出力段の負担を軽減するために，各駒ごとに推論を行いすべての行動を統合する形にまとめた．各ノードに格納される情報は，訪問回数N，選択確率P，総value値W，平均value値Q, そのノードの子ノードへのリンク，等になる．

探索が探索木の葉ノードに到達してのち，訪問回数とvalue値を$N(s_t,a_t) = N(s_t,a_t)+1$, $W(s_t,a_t) = W(s_t,a_t)+v$, $Q(s_t,a_t) = \frac{W(s_t,a_t)}{N(s_t,a_t)}$ によって更新していく．探索の最終段において最も訪問回数の多かったノードを最終的な着手とする．

MCTSシミュレーションは試合が終了するまで実行され，試合が終了すると各局面ごとに局面s, 分布$\boldsymbol{\pi}$，試合結果z，を保存し，定期的にニューラルネットワークのパラメータを損失関数を最小化するように勾配法にて更新して学習を進める．

広い範囲を探索するためのノイズ

PV-MCTSでは自己対戦で棋譜データを作成していくため，特定の手ばかり探索してしまうなどの偏りを防止するためのノイズをAlphaGo Zeroと同様な手法で付加している．ひとつは自己対戦時の着手選択時に温度減衰付きボルツマン確率選択が次式に基づいて行われる．

$$\pi(a|s_0) = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}} \tag{7.4}$$

ここで，τ は温度パラメータで探索の範囲を決定し，s_0 は根ノード，a は指し手を，$\pi(a|s_0)$ は s_0 での a の訪問回数を表している．$\tau = 0$ の時，通常のMCTSと同様に訪問回数の最も多い指し手が選択され，$\tau = 1$ の時，訪問回数に比例した指し手が選

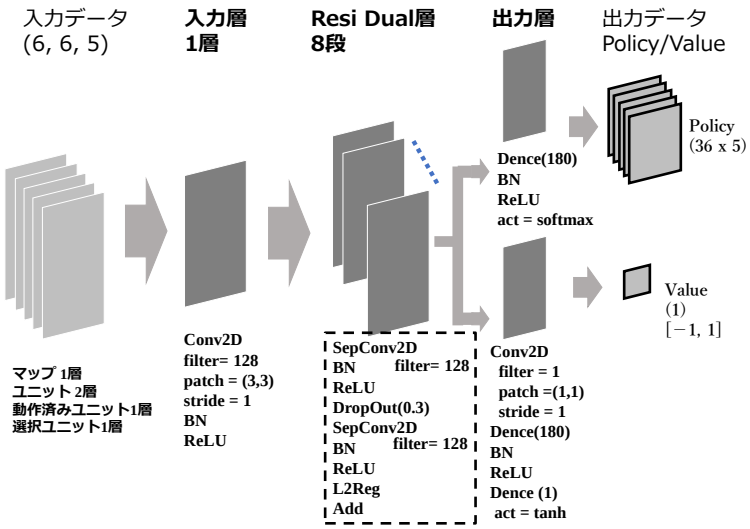


図 7.3: 設計したニューラルネットワークのブロック図

択される．τ はゲームの開始から最大ターンのおよそ 30% までに減衰され，それ以降は影響を与えない．また，MCTS シミュレーション時には探索木の根ノードにのみ Dirichlet ノイズ Dir(0.15)，$\varepsilon = 0.5$ [1] を付加することで特定の手に探索が偏ることを防止している．

7.1.4 ニューラルネットワーク部

ニューラルネットワーク部の設計は AlphaZero で使用された Residual [78] タイプの設計を導入しているが，今回のゲームに適用するために細かい部分で多くの変更を行った（図 7.3）．独自の変更として Dropout 部を追加して収束効率を高めている．

ターン制戦略ゲームに深層学習を適用しやすくするため policy network の出力段のニューロン削減策として，駒の移動先（36 か所）と攻撃先（四方向）をデコードした形の出力（36×5）とした（図 7.4）．この手法によりユニットの行動を表現する攻撃先のマスの位置データの量を削減し，さらには，前節で説明したユニット情報をニューラルネットワークの出力から入力に移動したことで出力ニューロン数を大幅に減らした．これにより（移動元）×（移動先）×（攻撃先）のマスの位置データを本来なら $36 \times 36 \times 5 = 6480$ の出力ニューロンが必要になる所を $36 \times 5 = 180$ に削減した．この手法は歩兵等に適用できるが自走砲に適用する際は攻撃先の追加が必要になる．第 5 章と第 6 章ではリカレントネットワークを利用して出力ニューロンの削減を図る手法 [62, 74] をとっていた．この手法は単一出力を時分割で出力には適用することができるが，この章においてはさらに value network の統合を考えているため 2 出力のニューラルネットワークを設計しなければならない．この場合，時分割で動作する部分と単出力である value network の統合が面倒になるという欠点がある．

ニューラルネットワークへの入力データは 0 から 1 までにそれぞれ正規化された形

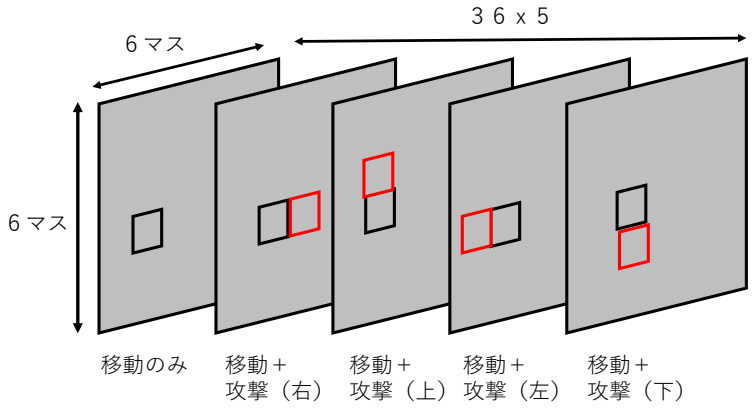


図 7.4: policy network 出力のデータ表現

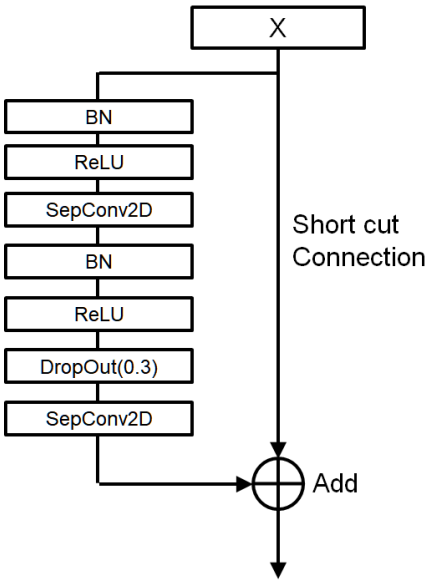


図 7.5: Residual ブロック

で，マップ状の進入禁止と草原かをマス目状にした情報が 1 層，ユニットの位置情報と HP の値をマス目状にデコードしたものを敵味方各 1 層で計 2 層，動作済みユニットの位置をマス目にした情報が 1 層，動作ユニットの位置をマス上においた情報が 1 層の計 5 層を 6 マス × 6 マス × 5 層としてまとめられて入力される．本章では地形情報は平野と進入禁止の二つに限定され，ユニットは歩兵のみに限定されている．この手法で TUBSTAP のすべての機能を表現するためにはチャンネル数を駒の種類に応じて増加させる必要がある．手番情報はユニット層の順番で表現されている．ニューラルネットワークの入力層は Conv2D が 1 層あり，続いて ResiDual Network が 8 段積層され，ここからポリシーヘッドとバリューヘッドに分岐して二つの出力へと接続される．policy 出力は行動予測の確率分布の形であり，value 出力は $[-1, 1]$ の範囲の試合の勝敗予測を表すスカラー出力である．

Residual Network ブロック

本研究で使用された Residual Network [78] のブロック図を図 7.5 に示す．Residual 層の 8 ブロックが入力レイヤーの後に配置され，学習の逆伝播は，ショートカット接続によって簡単に伝播されることにより，Residual Network の性能を大幅に向上させている．さらには学習の収束を改善するために，AlphaZero では使用されていなかった Wide Residual Network [79] を使用して，ドロップアウト部が中央部に挿入されている．メモリの節約と速度の向上のために，一般的に使用される Conv2D の代わりに SeparableConv2D が使用されている．これは，DepthwiseConv2D [80] と Conv2D の組み合わせであり，グラフでは SepConv2D として記述されている．

7.1.5　ニューラルネットワークの学習

自己対戦によって生成されたゲームデータを Experience replay [6] のリプレイメモリに蓄積して決められたタイミングでニューラルネットワークの学習を行う．これらの点は AlphaGo の方式と同じである．リプレイメモリに格納する際にはデータの鏡映と回転を行いデータ量を 8 倍にする Data Augmentation [28] を行っている．学習の際には学習率のスケジューリングを行い，オプティマイザーは Nesterov オプション付き SGD+momentum を使用しモーメンタム係数は 0.9 としている．バッチサイズは 128 とした．損失関数は次式のように設定した．

$$loss = (z - v)^2 + D_{KL}(\boldsymbol{\pi} \mid \mathbf{p}) + L2Reg \tag{7.5}$$

ここで，z は葉ノードにおける報酬値，v はバリューの出力，$D_{KL}(\pi \mid p)$ はカルバック・ライブラー情報量，$L2Reg$ は L2 正則化項である．第一項と第二項の重みは同一とし L2 正則化定数は 1×10^{-4} とした．自己対戦で学習するニューラルネットワークのトーナメントによる交代は行っていない．

7.2　対戦相手の設計とベンチマーク問題による性能比較

対戦実験に使用する対戦相手の設計と性能の評価を，ベンチマーク問題を使用して行う．

7.2.1　対戦相手の設計

ここでは対戦相手として次のアルゴリズムを用意した．

- 原始モンテカルロ法（PMC）．木探索のない単純なモンテカルロ法で配分しゲームの終了まで各枝に等分に 100 回のロールアウトを行う．

表 7.1: ベンチマーク問題　pinch01 での検証結果

検証アルゴリズム	成功	失敗
PMC	2	8
MCTS	10	0
PV-MCTS	10	0

- モンテカルロ木探索法（MCTS）．従来からあるモンテカルロ木探索として $UCB1 =$ 勝利数/訪問数 $+ \sqrt{\log(\text{総訪問数})/\text{訪問数}}$ の値に基づいて枝の選択を行い，訪問回数がしきい値をこえたノードはノードの展開を行う．シミュレーション回数はゲームの終局まで 2000 回行う．

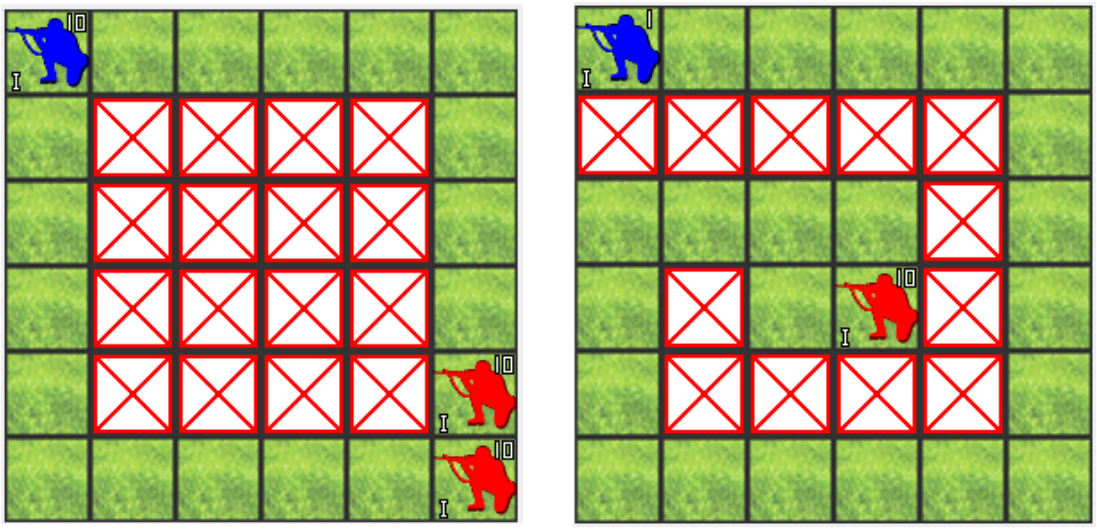
PMC と MCTS はここで設定したマップ上でできる限り強くするため一手あたりのシミュレーション回数を多めに設定し，結果として動作時間は標準的な思考時間が 30 秒から 1 分程度と長めになっている．これは以下の評価でも共通の設定である．ここで新しいプログラムを使用し，過去の研究で使用された M-UCT や M3Lee を使用しないのは，これらのプログラムは一手あたりに使用する思考時間が 10 秒以下と短い設定になっているため，より移動先が多く行動選択肢が広がるタイプのマップという条件においてはランダムな動きが多くなり，正しく動作できないケースがあるためである．M-UCT や M3Lee といったマップが想定しているのはある程度複雑なマップであって，そのようなマップでは枝刈り機能等によって効率的に探索ができるものの，本章での実験のような特殊な条件を想定しておらず，性能が上がらないものと思われる．

7.2.2　ベンチマーク問題による性能比較

以前作成されたベンチマーク問題集 [54] をベースにする図 7.6 のような挟み撃ち問題（図 7.6(a)） と経路探索問題（図 7.6(b)） を用意し，成功するかどうか検証した．成功の基準は挟み撃ちマップであれば規定のターン数内で BLUE を挟み撃ちして相手を全滅させられたら成功，経路探索マップは BLUE までにたどり着いて相手を全滅させられたら成功である．

PV-MCTS の場合は二つのマップを合わせて 50 回程度の試合で学習させた．二つのマップとも BLUE 側のアルゴリズムは MCTS を使用した．これらのマップに限って学習した場合は学習時間は一時間弱であった．以上の結果をまとめたものが表 7.1 と表 7.2 である．

結果としては PMC は原始モンテカルロ法であり木探索が入っていないせいか失敗があったが，経路探索問題では 8 回も成功していて性能は高い．また MCTS はすべての試行で成功したので，これらの問題に対しては性能が高いと言える．これはどちらのアルゴリズムに対してもそれなりに長い計算時間を与えて動作させているためと思われる．PV-MCTS は学習する手間があり学習したマップ専用の AI ということ



(a) pinch01　　(b) pathfind01

図 7.6: アルゴリズム検証用マップ

表 7.2: ベンチマーク問題　pathfind01 での検証結果

検証アルゴリズム	成功	失敗
PMC	8	2
MCTS	10	0
PV-MCTS	10	0

になってしまうものの，探索自体は高速で行うことができすべて成功することができているので基本的な動きは問題ないと言える．

7.3　平原マップにおける学習

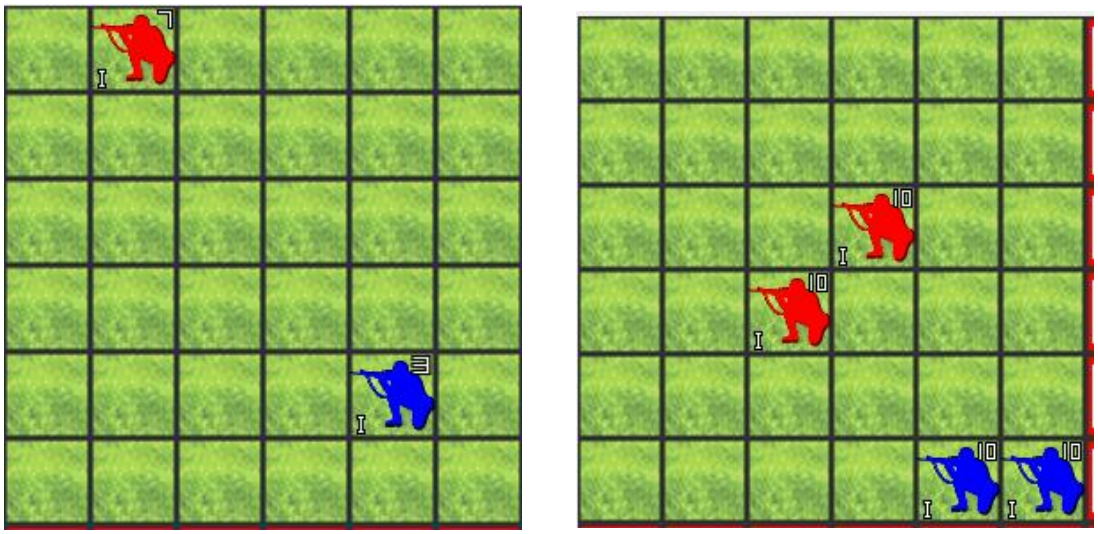
幅広探索木問題における広い探索条件になるマップとしてここでは障害物がなく移動先が多い平原マップを考える．自己対戦に使用するマップは学習を効率的に進めるために用意した学習マップを使用してニューラルネットワークを訓練する．

まず，ニューラルネットワークの学習の進展について各種パラメータとの関係を損失データから確認する．

さらに用意していた対戦用 AI との対戦実験により性能を確認する．

7.3.1　学習マップの設定

自己対戦で使用した学習用のマップの初期盤面設定は 6×6 マスに固定した図 7.7 のようなもので以下のように設定している．



(a) ランダムマップ　　(b) 手筋マップ

図 7.7: 学習マップの例

- ランダムマップ．RED/BLUE の歩兵は個数と HP と配置はすべてランダムに設定されているマップ．歩兵の個数は 1 個か 2 個．
- 手筋マップ．2019 年ゲーム情報学研究会 [62] において使用された 5×5 マスのマップを 6×6 マスに変更したもの 50 種類程度．

ここでいう手筋マップとは図 7.7(b) のような特定の重要な局面で優先して学習すべき手筋のある場合を特に取り出したマップである．図 7.7(b) は一見ではわかりにくいけれども RED に必勝手順が存在する局面であるがこのたびの学習で PV-MCTS がこの必勝局面を徐々に学習していったものである．この必勝局面で勝てるかどうかによって学習の進み具合を確認できる．学習局面に 1 個対 1 個などの設定があるのは，PV-MCTS の探索手法は特定の局面について少しずつ方策と価値を覚えていって有利な局面を見出していくような方式であるため，探索木の下の局面から覚えていくのがより早く学習を進めるからである．この場合は最終的な目的の局面が 2 個対 2 個の局面であったとしても，終局までには必ず 1 個対 2 個や 1 個対 1 個の局面を通過するためこのような局面の学習も必ず必要になるので，あらかじめ同時に学習を進めることで学習を加速させることが狙いである．

なお，PV-MCTS での 1 回あたりのシミュレーション回数はユニットあたり 500 回であり，この設定は以下すべての実験で共通である．

自己対戦に要した時間

ニューラルネットワークの初期状態から学習を始めて学習と対戦など，のデータ取得で全体で 3 週間程度，学習だけの時間的には一週間程度を要した．

使用したハードウェアとソフトウェア

この章における数値実験では二組のハードウェアとソフトウェアを使用した．その構成は以下のようである．

- ソフトウェア : Python 3.6, Keras 2.2 ハードウェア : CPU Intel i7 3.4GHz, GPU NVIDIA GTX1050Ti.
- ソフトウェア : Python 3.6, Keras 2.4 ハードウェア : CPU Intel Xeon E5-2620 V4 2.14GHz, GPU NVIDIA RTX2080Ti．(計算サーバー)

7.3.2 損失曲線と各種パラメータの関係

ここでは実験に使用するニューラルネットワークの基本性能を確認し，学習する能力について検証する．

損失についてここではグラフで観察するが，強化学習において損失が下がっていることが必ずしも学習が目的とする方向に進んでいることを保証するわけではない．なぜなら，一見損失が低下して学習が進んでいるようにみえても実際のニューラルネットワークは袋小路のように同じ局面ばかり学習していたり，鞍点や局所解に向かっているだけの場合が少なからず発生するからである．

したがって，ここでの損失曲線の確認はあくまで学習の進み具合についての検証にすぎず，プログラムが本当の意味で強いプレイができるようになっているかどうかの確認は対戦実験で行う．

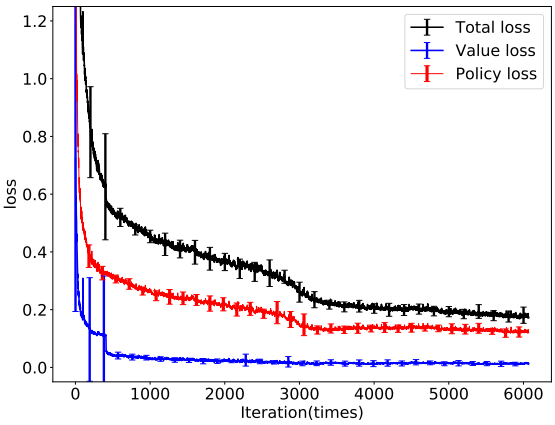


図 7.8: 損失の推移 ($B_{attack} = 3.7$, dropout あり)

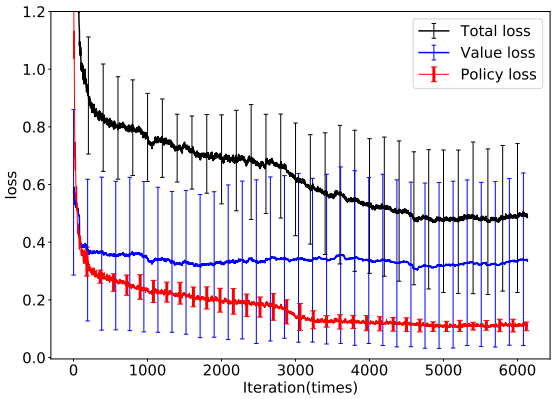


図 7.9: 損失の推移 ($B_{attack} = 0$)

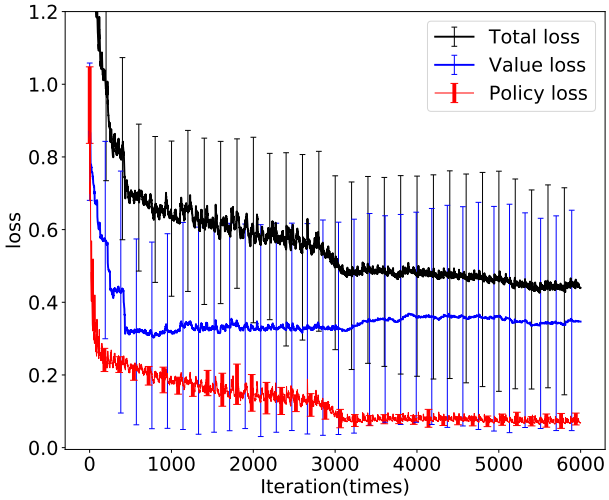


図 7.10: 損失の推移 (dropout なしの場合)

ニューラルネットワークの学習と損失曲線

ニューラルネットワークの学習の推移を損失データで確認する．通常の学習に使用した設定（$B_{attack} = 3.7$，dropout あり，dropout rate $= 0.3$）学習を進め，5 時間から 10 時間程度で学習は終了し，その損失の推移は図 7.8 のようになった．

比較のためにバイアス係数の B_{attack} を使用しなかった場合（図 7.9）と dropout を使用しなかった場合の損失（図 7.10）のグラフを示す．図 7.8 および図 7.9）と図 7.10 は 500 マップを学習させた損失を各イタレーションごとに記録した．これを各条件で 10 回繰り返して各データを平均して描画している．エラーバーは各ポイントの 10 個のデータの標準偏差を表している．いずれも初期状態からスタートし学習マップ 500 個で学習している．学習率は学習マップの 1 から 250 個までは 1×10^{-1} それ以降は 1×10^{-2} としている．この影響でイタレーションがおよそ 3000 回のあたりから損失傾向が変動している．

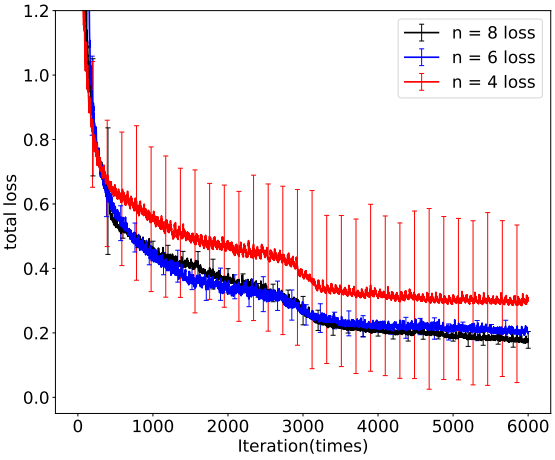


図 7.11: 全体の損失 (n = 8/6/4).

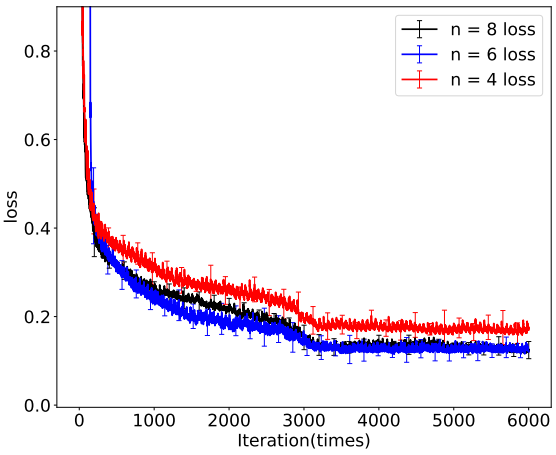


図 7.12: Policy の損失 (n = 8/6/4).

図 7.9 の $B_{attack} = 0$ の設定では一見，収束が早く行われているようにみえるが，実際の駒の動きは攻撃が適切に行うことができない場合が多く，学習が進まなくなってしまうことから，早いうちから過学習が発生していると推定される．また，この設定では全体波形が乱れる傾向が一試行を通じてまたは部分的にもみられた．

図 7.10 の dropout なしの設定でも比較的損失が低下するのが早いものの，学習が停滞することが多く dropout を設定したほうが安定して損失が収束する可能性が高く波形も安定しない場合があった．

Residual ブロックの 4, 6, 8 段の効果を比較するため，学習マップ 500 個と少数のマップ上でおよそ 6000 イタレーションを実行した結果が図 7.11 である．ここでマップの数を固定し一つのマップ上での局面数はゲームの終局によって変動し，トータルのイタレーションの数は固定されず変動することになるためグラフの横軸は固定値になっていない．

グラフでは，n は Residual ブロックの段数を表し，n = 6 と n = 8 ではほぼ同等の損失であるが，n = 4 ではわずかに劣化している．また n = 4 では波形が安定しない場合があるためエラーバーで示した標準偏差が大きくなっている．

図 7.12 は n = 8, 6, 4 に対する同一の学習における policy 損失を示しているが，段数による違いは顕著ではない．

図 7.13 は n= 8, 6, 4 に対する value 損失を示しているが n=4 は収束性能について全体のボトルネックとなっている．

Residual ブロックにおける Conv2D 部と SepConv2D 部の性能を比較するため，Conv2D で n=6，SepConv2D で n=8 とした全体の損失曲線が図 7.14 である．曲線は両者がほぼ同様の傾向を示しているが，Conv2D はより大きなメモリを必要としておりおおよそ 30% 程度多くメモリを消費し，推論に要する時間も SepConv2D に比較して長くなる．また波形が乱れる傾向が Conv2D に見られたためエラーバーが広くなっている．

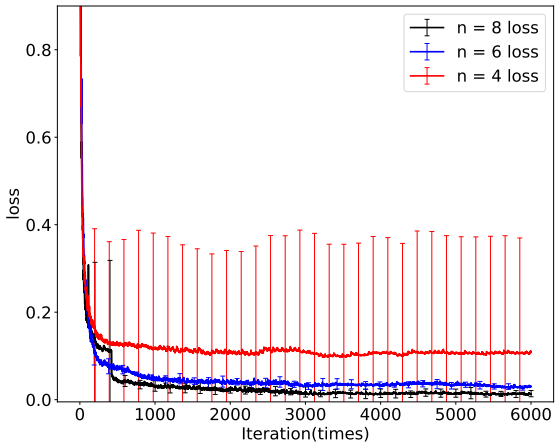


図 7.13: Value の損失 (n = 8/6/4).

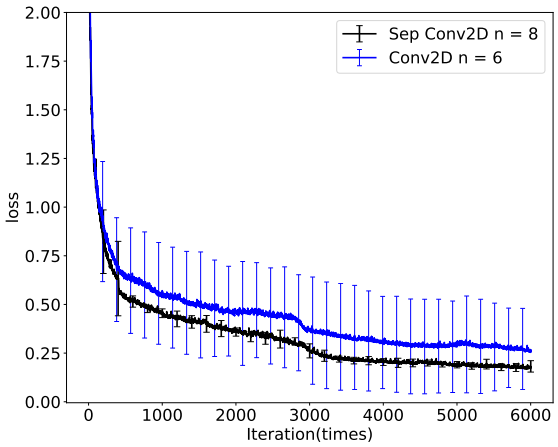


図 7.14: Sep Conv2D (n = 8) と Conv2D (n = 6) の全体の損失.

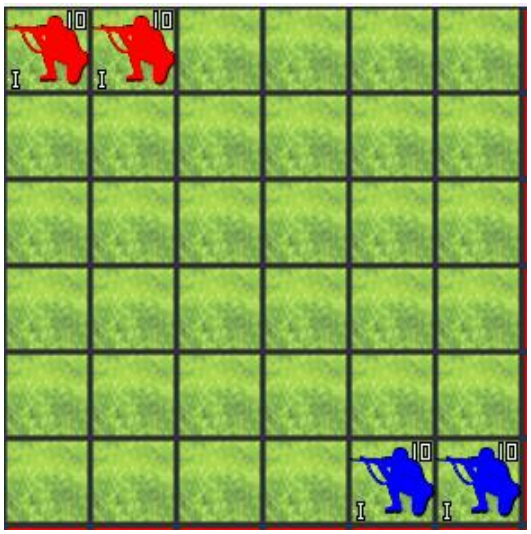


図 7.15: 対戦用マップ sq6x6_01

7.4 平原マップにおける対戦実験

自己対戦によって学習した PV-MCTS の性能評価のため，対戦実験を行った．対戦性能の比較のため，Conv2D と SepConv2D の比較，Redidual ブロックの段数，dropout の有無，B_{attak} の攻撃バイアスの効果，policy network のみでの動作，expand の効果などを比較した．

7.4.1 対戦による性能評価

対戦に使用したマップは図 7.15 である．このマップでは歩兵ユニットが各二個の配置となっており，最大ターン数は 16 ターンまでと設定し勝利条件は相手を全滅し

表 7.3: PV-MCTS のマップ sq6x6_01 での対戦結果．S は Short，L は Long，Sep は SepConv2D，2D は Conv2D をそれぞれ表す．

番号	対戦相手	Time	n	Sep/2D	B_{attack}	Dropout	勝ち	引分け	負け	勝率(%)
(1)	PMC	S	4	Sep	3.7	Yes	7	3	40	14.0
	MCTS	S	4	Sep	3.7	Yes	3	5	42	6.0
(2)	PMC	S	6	Sep	3.7	Yes	18	5	27	36.0
	MCTS	S	6	Sep	3.7	Yes	13	9	28	26.0
(3)	PMC	S	8	Sep	3.7	Yes	16	1	33	32.0
	MCTS	S	8	Sep	3.7	Yes	11	4	35	22.0
(4)	PMC	S	6	2D	3.7	Yes	17	6	27	34.0
	MCTS	S	6	2D	3.7	Yes	20	4	26	40.0
(5)	PMC	S	8	Sep	3.7	No	7	3	40	14.0
	MCTS	S	8	Sep	3.7	No	6	1	43	12.0
(6)	PMC	L	8	Sep	0.0	Yes	20	3	27	40.0
	MCTS	L	8	Sep	0.0	Yes	24	2	26	48.0
(7)	PMC	L	8	Sep	3.7	Yes	130	5	15	86.7
	MCTS	L	8	Sep	3.7	Yes	128	6	16	85.3

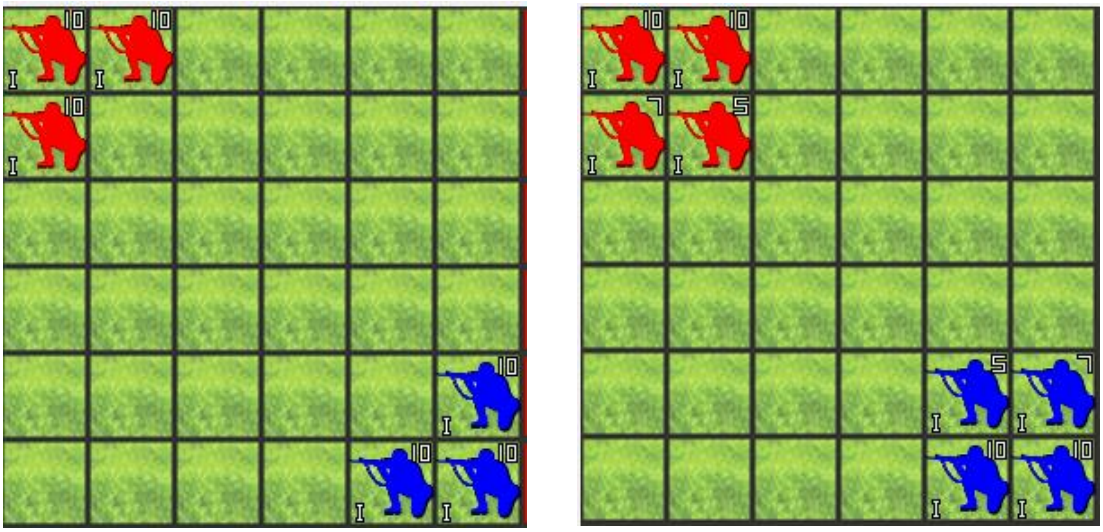
た時のみとし，全滅がおきずに最大ターンに到達した場合はすべて引分けとした．先攻の設定は試合の前半を PV-MCTS とし，後半を後攻として先手後手の有利不利条件がなくなるようにした．対戦マップが単純で障害物がない設計になっており，プログラムが簡単化し評価しやすくなっている．

対戦結果を設定パラメータとともにまとめたものが表 7.3 である．

ここで n は Residual ブロックの段数を表し，Time は Long が 1 週間の長期学習バージョンで，Short が 500 ゲームのイタレーションバージョンで実行時間は 5 時間から 15 時間程度である．Time が Short の設定では必ずしも最終的な性能ではなく簡易的な評価となっている．

対戦結果については，Residual ブロックの段数の差について見てみると (1) の n=4 と (2) の n=6,(3) の n=8 の比較では n=4 は性能が悪いが n=6 と n=8 ではさほど差はない．(4) の Conv2D と (2) の SepConv2D の比較では大きな性能の差はみうけられない．dropout 部を除いた Residual ブロックで学習した場合の (5) では dropout を使用した (3) よりも明らかに勝率が低下することから dropout の効果も大きいと言える．$B_{attack} = 0.0$ の設定にすることで攻撃バイアスの影響をみた場合の (6) では，$B_{attack} = 3.7$ の設定の (7) に比較して勝率はおよそ半分に減ってしまったので明らかに攻撃バイアスには効果がある．

最終的には (7) において，PMC と MCTS に対して 86.7% と 85.3% と高い勝率となり高い性能をこの設定で示すことがわかる．



(a) 対戦マップ sq6x6_02　　(b) 対戦マップ sq6x6_03

図 7.16: 学習に使用していない対戦用マップ

表 7.4: PV-MCTS の未知の対戦マップでの対戦結果

対戦相手	マップ	勝ち	引分け	負け	勝率 (%)
PMC	sq6x6_02	37	0	13	74.0
MCTS		36	7	7	72.0
PMC	sq6x6_03	28	5	17	56.0
MCTS		24	5	21	48.0

汎化性能の検証

また，学習したニューラルネットワークの汎化性能を検証し，未知局面に対する対応能力を測定するため，学習していない設定のマップでの対戦を行った．対戦に使用したのは図 7.16 のマップでありどちらも学習の際にはこの設定を使用していない．

図 7.16(a) のマップ sq6x6_02 は歩兵が各 3 個の設定で，図 7.16(b) のマップ sq6x6_03 は歩兵が各 4 個である．先手後手は試合の半分で入れ替えている．バージョンは Time = Long, SepConv2D, $B_{attack} = 3.7$, dropout ありを使用し，対戦結果が表 7.4 である．

学習を行っていない設定の対戦マップにもかかわらず，マップ sq6x6_02 で勝率が 74.0% と 72.0%，マップ sq6x6_03 で 56.0% と 48.0% になっており PV-MCTS は良い性能を示している．このような動作ができているのは初期条件が 3 駒同士や 4 駒同士でも戦闘が進むにつれて 2 駒同士や 1 駒同士の状態になる瞬間があり，このような時に 2 駒マップで学習した経験が探索を助けているものと思われる．PV-MCTS の探索に要する時間は一手あたり 10 秒から 30 秒程度であったが，3 個のユニットのマップでは 1 分程度にのびるようになった．PMC と MCTS は 30 秒から一分程度である．

表 7.5: ポリシーのみでの対戦マップ sq6x6_01 での対戦結果

対戦相手	展開	勝ち	引分け	負け	勝率 (%)
PMC	ゲーム終了まで	86	4	10	86
MCTS		85	7	8	85
PMC	ルート局面のみ	19	29	52	19
MCTS		17	34	49	17

Policy network の性能

AlphaGo の当初の論文では，policy network や value network と ロールアウトの組合せや単体での性能評価が行われていたが，今回の提案ではロールアウトは使用していないので value network を使用しないで policy network 単体での評価を試みる．

プログラムの設定を value network の効果を停止して，policy network のみでの動作となるようにし，かつ局面の展開をゲーム終了まで行う場合と，ルート局面のみの場合の二つのケースで対戦マップ sq6x6_01 において対戦を行った．ゲーム終了まで展開を行う場合は終了局面における v をターミナル値 z（勝利:1, 引分け:0，敗北:−1）におきかえて value network は使用しないようにしている．ルート局面のみの展開の場合は，policy network の確率出力のうち最も確率の高い行動を選択するのみであり，探索による行動の改善は行っていない．この対戦成績が表 7.5 である．

展開をゲーム終了まで行う場合は勝率的には value network を使用する場合 (表 7.3 の (7)) と比較して差があまりないが終盤には MCTS 特有のゆるみの挙動がみうけられた．ルート局面のみ policy network を使用し探索を行わない場合では，大きく勝率が低下した．このことにより探索を行うことで PV-MCTS の性能が大幅に向上していることがわかる．しかしながらルート局面のみの動作でも勝率は 17% から 19% 程度あり，また引分けが 29% と 34% になっており，勝利の割合と合わせると全体の約半分になることから，policy network の学習はある程度できていると思われる．

Expand の影響

MCTS の基本動作として Expand があるが，ここまでは AlphaZero の Expand 回数がはっきりしなかったために，Expand をゲームの終了まで行いながらシミュレーションをしていた．Expand の回数の影響を調べるため，Expand が一回の場合のデータを比較する．

プログラムを Expand が一回のみとなるように変更し，ニューラルネットワークが初期状態から学習をスタートして 500 マップごとに対戦を対戦マップ sq6x6_01 の上で行った．PV-MCTS のシミュレーション回数は Expand の回数が減少したことに対応して回数を増加させ 1000 回としたが，それにもかかわらず一手あたりの実行時間は速くなり，およそ 15 秒から 20 秒程度となった．この対戦成績が表 7.6 である．

表 7.6: Expand が 1 回の場合の対戦マップ sq6x6_01 での対戦結果

対戦相手	学習マップ数	勝ち	引分け	負け	勝率 (%)
PMC	500	64	6	30	64.0
MCTS		59	11	30	59.0
PMC	1000	80	5	15	80.0
MCTS		80	8	12	80.0

表 7.7: ランダムマップのみで学習した場合の対戦マップ sq6x6_01 での対戦結果

対戦相手	学習マップ数	勝ち	引分け	負け	勝率 (%)
PMC	500	15	17	68	15.0
MCTS		15	15	70	15.0

結果としては Expnad の回数が減少したことで探索シミュレーション回数を 1000 回に増やすことができ，学習が高速化して速い立ち上がりで学習が進んでいる．

手筋マップの効果

今回使用した手筋マップの効果について確認するために，学習用のデータのマップ群から手筋マップを取り除き，ランダムマップのみで学習を行って比較する．他のパラメータについては同一として，学習用マップのみを変更して 500 マップまで対戦マップ sq6x6_01 上で対戦したデータで学習した．設定されたパラメータは n = 8, Sep, $B_{attack} = 3.7$, Dropout ありであり，対戦成績が表 7.7 となった．

ここでの結果の比較対象は表 7.3 の (3) のケースであり，学習マップに手筋マップを含むか含まないの部分であり，他のパラメータは同一である．手筋マップがある場合で対戦相手が PMC の時に 32.0% に対し，手筋マップがない場合で 15.0% と勝率が 17% 低下した．また，対戦相手が MCTS の場合で，手筋マップがある場合で勝率が 22.0% から手筋マップがない場合で 15.0% に 7% 低下した．PMC と MCTS の両方で引分けが増加している．

以上より学習マップに手筋マップを入れることで適切な学習がより早く進むことになり，勝率の向上に寄与することが確認できた．

7.5 おわりに

ニューラルネットワークの性能検証するため，学習の進展を損失の減少を見ている本章の設計で導入された，dropout や B_{attack} 項により損失が安定して減少する様子が見て取れた．また，Rsidual ブロックの段数についての比較から採用した段数において最適化されていることがわかる．

今回用意した TUBSTAP のベンチマーク問題として提起された挟み撃ち問題や経路探索問題においては，広い範囲を探索しなければならないため，旧来のアルゴリズムでは深い探索が必要であるが PV-MCTS では学習をしている状態であれば短時間の探索で正しい答えを出すことができている．また，ここで用意した対戦用のマップのように広い探索が必要な場合，旧来のアルゴリズムでは短いターンですらなかなか正しい探索を行えなくなることがあるが，PV-MCTS では行動データの表現を変更して，出力にあったユニット情報を入力へと移動させることで，ニューラルネットワークの設計負担を減少させ，広い行動範囲から深層学習による効率的な確率選択を行って正しい行動選択を行うことができるようになり，結果として対戦成績は良好となる．

そして，２駒同士の対戦結果が３駒以上での対戦に役立ったように学習した結果はある程度推論がおよぶ範囲であれば応用が可能であるということができる．また，通常のニューラルネットワークによる行動決定では policy network のみでも行うことができるがさらに value network を組み合わせることで他の利点も得ることができるようになる．

旧来の単純な設計では policy network 単独または value network 単独で使用されていたニューラルネットワークであるが，AlphaZero 方式アルゴリズムの導入によって policy と value が統合され一つのニューラルネットワーク上で扱うことができるようになり，MCTS タイプの探索をすることで，探索能力も向上し，結果としてより強力な行動出力が行えるようになり，AI プレイヤーとしての強さが大幅に向上した．

学習に使用したマップは二つの駒のみを使用したものであったが，過去の棋譜を使用することなく学習を進め，最終的には MCTS アルゴリズムをこえる強さへと到達した．

また，学習に使用していない条件のマップ上での対戦においても作成された AI プレイヤーはニューラルネットワークの汎化性能によって未知の条件に対して対応し高い性能を修めた．

本章で行った実験ではユニット数，ユニットの種類，マップ地形に制限があったが，これらのバリエーションについても学習が適切に行われれば正しい行動出力ができると思われる．しかしながら、ユニット数やユニットの種類が増えるごとにデータの複雑さやマップの複雑さが増していくため学習するためのデータ量は大量となっていくことが予想される．

第8章　おわりに

ゲームを数理的に研究するアプローチは当初は人工知能の研究の一分野として開始され，長年にわたって行われてきた．人工知能研究は何度も活発な時期を繰り返してきたが，深層学習の研究の活性化とともに今また激しい勢いで進展しつつある．その勢いは人工知能研究の一分野のみならず，さまざまな分野の研究にも大きな影響を及ぼしている．

本論文では，深層学習をターン制戦略ゲームに適用する上での諸課題について分析を行い，改善する提案を行い，その性能についてデータを提示した．

DeepMind 社が提案した深層学習を使用した新しいアルゴリズムである DQN や AlphaGo/ AlphaGo Zero/ AlphaZero によって深層学習と強化学習の研究を大変活発となった．しかしながら，ターン制戦略ゲームについては研究が活発とは言えないなか，本研究はターン戦略ゲームについて深層学習を適用する手法に関する研究を行った．まずは本研究に関連する既存研究について，AlphaZero に連なる形で歴史的にも概説し，AlphaZero に至る研究の流れを見えるようにした．さらには，ターン制戦略ゲームの歴史や社会とのかかわり，本研究で使用したフレームワーク TUBSTAP についてまとめ本研究におけるベースを作った．そして，まずは用意されているマップの数の少なさを補い，開発したアルゴリズムの性能評価や比較をしやすくするためのベンチマーク問題をまとめて開発し，発表した．

深層学習をターン制戦略ゲームに適用する上で，最初にぶつかる課題としてターン制戦略ゲームの持つ複雑なデータ構造をいかにニューラルネットワークが扱えるようにするかというデータ表現の問題があるが，TUBSTAP においてはユニットの行動表現をリカレントネットワークを用いて出力させることで簡略化し，実際に実装を行い，他の手法のいくつかについての比較をした．そしてゲームにおける探索に必要な policy network の設計を大量の棋譜データを用意することで棋譜学習の形で実現し，データを取得し性能評価した．それに続き，深層学習の新しい技術である Residual Network を導入し多段連結することでニューラルネットワークの性能を大幅に向上させ，policy network と value network を統合することで探索性能の向上を図った．また，深層学習を TUBSTAP に適用するためにユニット選択をニューラルネットワークの出力から入力に移動させるなど，必要な変更を行った．設計された PV-MCTS は学習したマップ上だけではなく，未知のマップ上の条件でも深層学習の汎化性能によって優れた対戦成績を示した．設計パラメータは，PV-MCTS の性能を高めるために最適化した値を用いた．

本研究によって明らかになった課題として複雑なデータをニューラルネットワーク

で表現するデータ表現の問題がある．これはゲーム研究だけではなく，他の分野における研究に共通する課題でもあろう．ターン制戦略ゲームのように過去の研究による棋譜の蓄積がないゲームで，ニューラルネットワークを学習させるための手法としての自己対戦による学習アルゴリズムは非常に有効であるといえる．また，旧来のアルゴリズムでは困難だったり面倒な設定が必要だったりした複雑なマップや深い探索が必要なマップなどを，深層学習の能力である映像的な処理によるアルゴリズムで行うことで簡単で高速に処理することができるようになったことは大きな前進である．

本研究における課題としては，まずは深層学習にかかわるハイパーパラメータの設定や最適化問題がある．学習などに設定するパラメータの最適な値を決定する根拠に乏しく設定が難しいことが多い．これは本研究のみではなく，深層学習全般に共通の課題である．そして，研究で使用された検証しやすく簡略化され制限されたマップは，ターン制戦略ゲームとしてはかなり単純なものであって，本来はマップのサイズは 8×8 サイズよりも大きく，16×16 以上のものも多く，時に 64×64 サイズや 128×128，256×256 などといったものがよくある．地形的にも多様なものがあるのが普通であるので，より広いマップサイズでさまざまな地形を含んだマップ上での研究が次の目標となる．

また，PV-MCTS の動作時間も駒の数に比例して長くなる設定となっており，より動作時間の高速化と高効率化のアルゴリズムの開発も今後の課題である．

参考文献

[1] Silver, D., Huang, A., Maddison, C. J. et al.: Mastering the game of Go with deep neural networks and tree search, *Nature*, Vol. 529, pp. 484–503 (2016).

[2] Silver, D., Schrittwieser, J., Simonyan, K. et al.: Mastering the game of Go without human knowledge, *Nature*, Vol. 550, pp. 354–359 (2017).

[3] Silver, D., Hubert, T., Schrittwieser, J. et al.: A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play, *Science*, Vol. 362, pp. 1140–1144 (2018).

[4] Mnih, V., Kavukcuoglu, K., Silver, D. et al.: Playing Atari with Deep Reinforcement Learning, *Computing Research Repository (CoRR)*, Vol. abs/1312.5602 (2013).

[5] Bellemare, M. G., Naddaf, Y., Veness, J. et al.: The Arcade Learning Environment: An Evaluation Platform for General Agents, *Computing Research Repository (CoRR)*, Vol. abs/1207.4708 (2012).

[6] Mnih, V., Kavukcuoglu, K., Silver, D. et al.: Human-level control through deep reinforcement learning, *Nature*, Vol. 518, pp. 529–33 (2015).

[7] 佐藤直之，藤木　翼，池田　心：戦術的ターン制ストラテジゲームにおける AI 構成のための諸課題とそのアプローチ，情報処理学会論文誌，Vol. 57, No. 11, pp. 2337–2353 (2016).

[8] Shannon, C. E.: *Programming a Computer for Playing Chess*, pp. 2–13, Computer Chess Compendium, Springer New York (1988).

[9] 松原　仁：ゲーム情報学：コンピューター将棋を超えて，情報管理， Vol. 59, No. 2, pp. 89–95 (2016).

[10] Knuth, D. E. and Moore, R. W.: An analysis of alpha-beta pruning, *Artificial Intelligence*, Vol. 6, No. 4, pp. 293 – 326 (1975).

[11] NAGAI, A. and IMAI, H.: Proof for the Equivalence between Some Best-First Algorithms and Depth-First Algorithms for AND/ OR Trees, *IEICE TRANSACTIONS on Information and Systems*, Vol. E85-D, No. 10, pp. 1645–1653 (2002).

[12] Allis, L., van der Meulen, M. and van den Herik, H.: Proof-number search, *Artificial Intelligence*, Vol. 66, No. 1, pp. 91 – 124 (1994).

[13] 長井　歩：df-pn アルゴリズムと詰将棋を解くプログラムへの応用，pp. 96–114, アマ 4 段を超える―コンピュータ将棋の進歩〈4〉，共立出版 (2003).

[14] Schaeffer, J., Burch, N., Björnsson, Y. et al.: Checkers Is Solved, *Science*, Vol. 317, No. 5844, pp. 1518–1522 (2007).

[15] Tsuruoka, Y., Yokoyama, D. and Chikayama, T.: Game-Tree Search Algorithm Based On Realization Probability, *ICGA Journal*, Vol. 25, pp. 145–152 (2002).

[16] 瀧澤武信，松原　仁，小谷善行ほか：人間に勝つコンピュータ将棋の作り方，技術評論社 (2012).

[17] 築地　毅，柴原一友，但馬康宏ほか：TD(λ) と Bonanza の学習法との性能比較，ゲームプログラミングワークショップ 2007 論文集，Vol. 2007, No. 12, pp. 140–143 (2007).

[18] 金子知適：コンピュータ将棋の評価関数と棋譜を教師とした機械学習，人工知能学会誌，Vol. 27, No. 1, pp. 75–82 (2012).

[19] Buro, M.: Improving heuristic mini-max search by supervised learning, *Artificial Intelligence*, Vol. 134, No. 1, pp. 85–99 (2002).

[20] Tesauro, G.: Programming backgammon using self-teaching neural nets, *Artificial Intelligence*, Vol. 134, No. 1, pp. 181–199 (2002).

[21] 保木邦仁：局面評価の学習を目指した探索結果の最適制御，ゲームプログラミングワークショップ 2006 論文集，Vol. 2006, pp. 78–83 (2006).

[22] 松原仁編ほか：コンピュータ囲碁，共立出版 (2012).

[23] Kocsis, L. and Szepesvári, C.: Bandit Based Monte-Carlo Planning, *Machine Learning: ECML 2006* (Fürnkranz, J., Scheffer, T. and Spiliopoulou, M., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 282–293 (2006).

[24] Hinton, G., Osindero, S. and Teh, Y.-W.: A Fast Learning Algorithm for Deep Belief Nets, *Neural computation*, Vol. 18, pp. 1527–54 (2006).

[25] 岡谷貴之：深層学習，講談社 (2015).

[26] Glorot, X., Bordes, A. and Bengio, Y.: Deep Sparse Rectifier Neural Networks, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Vol. 15, pp. 315–323 (2011).

[27] Srivastava, N., Hinton, G., Krizhevsky, A. and other: Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research*, Vol. 15, pp. 1929–1958 (2014).

[28] Shorten, C. and Khoshgoftaar, T. M.: A survey on Image Data Augmentation for Deep Learning, *Journal of Big Data*, Vol. 6, pp. 1–48 (2019).

[29] Jouppi, N. P., Young, C., Patil, N. et al.: In-Datacenter Performance Analysis of a Tensor Processing Unit, *Computing Research Repository (CoRR)*, Vol. abs/1704.04760 (2017).

[30] Berner, C., Brockman, G., Chan, B. et al.: Dota 2 with Large Scale Deep Reinforcement Learning, *Computing Research Repository (CoRR)*, Vol. abs/1912.06680 (2019).

[31] Vinyals, O., Babuschkin, I., Czarnecki, W. et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning, *Nature*, Vol. 575, p. 350–354 (2019).

[32] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K.: Asynchronous Methods for Deep Reinforcement Learning, *Proceedings of The 33rd International Conference on Machine Learning* (Balcan, M. F. and Weinberger, K. Q., eds.), Proceedings of Machine Learning Research, Vol. 48, New York, New York, USA, PMLR, pp. 1928–1937 (2016).

[33] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O.: Proximal Policy Optimization Algorithms, *Computing Research Repository (CoRR)*, Vol. abs/1707.06347 (2017).

[34] Schulman, J., Levine, S., Moritz, P., Jordan, M. I. and Abbeel, P.: Trust Region Policy Optimization, *Proceedings of the 32nd International Conference on Machine Learning (ICML2015)*, Vol. 37, pp. 1889–1897 (2015).

[35] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H. and Silver, D.: Distributed Prioritized Experience Replay, *International Conference on Learning Representations (ICLR2018)* (2018).

[36] Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J. and Munos, R.: Recurrent Experience Replay in Distributed Reinforcement Learning, *International Conference on Learning Representations* (2019).

[37] 小野憲史:ゲームクリエイターが知るべき97のこと，オライリージャパン (2013).

[38] 高橋浩徳：ボードゲームの近現代史，大阪商業大学アミューズメント産業研究所紀要，No. 20, pp. 119–181 (2018).

[39] boardgamegeek: Tactics(1954), https://www.boardgamegeek.com/boardgame/31766/tactics.

[40] Take-Two: Take-Two-NewsRelease, https://www.businesswire.com/news/home/20160623005149/en/.

[41] 村山公志朗，藤木　翼，池田　心：学術研究用プラットフォームとしての大戦略系ゲームのルール提案，ゲームプログラミングワークショップ 2013 論文集，Vol. 2013, pp. 146–153 (2013).

[42] IkedaLab: TUBSTAP ホームページ，http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs/.

[43] 松原仁，半田剣一：ゲームとしての将棋のいくつかの性質について，情処学会 AI 研究会，pp. AI96–3 (1994).

[44] Arimaa: Arimaa, https://en.wikipedia.org/wiki/Arimaa.

[45] Brian, H.: A Look at the Arimaa Branching Factor, http://arimaa.janzert.com/bf_study/.

[46] 川上裕生，鶴岡慶雅：多様な特徴量を用いた Arimaa 評価関数の比較学習，ゲームプログラミングワークショップ 2014 論文集， Vol. 2014, pp. 151–156 (2014).

[47] Wu, D. J.: Designing a Winning Arimaa Program, *J. Int. Comput. Games Assoc.*, Vol. 38, pp. 19–40 (2015).

[48] Civilization: Civilization, https://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%B4%E3%82%A3%E3%83%A9%E3%82%A4%E3%82%BC%E3%83%BC%E3%82%B7%E3%83%A7%E3%83%B3.

[49] 加藤千裕，三輪　誠，鶴岡慶雅ほか：ターン制ストラテジーゲームにおける戦術決定のための UCT 探索とその効率化，ゲームプログラミングワークショップ 2013 論文集，Vol. 2013, pp. 138–145 (2013).

[50] 武藤孝輔，西野順二：ターン制戦略ゲームにおけるファジィ評価を用いた探索木の枝刈り，ゲームプログラミングワークショップ 2015 論文集，Vol. 2015, pp. 54–60 (2015).

[51] 武藤孝輔，西野順二：ターン制戦略ゲームにおけるファジィ大局的評価，日本知能情報ファジィ学会ファジィ システム シンポジウム 講演論文集， Vol. 32, pp. 503–508 (2016).

[52] Fujiki, T., Ikeda, K. and Viennot, S.: A platform for turn-based strategy games, with a comparison of Monte-Carlo algorithms, *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 407–414 (2015).

[53] 提橋　凜，坪倉弘治，西野順二：ターン制戦略ゲームに対する多段化探索木とその工夫の効果，ゲームプログラミングワークショップ 2018 論文集，Vol. 2018, pp. 187–191 (2018).

[54] 木村富宏，池田　心：ターン制戦略ゲームにおけるベンチマークマップの提案，ゲームプログラミングワークショップ 2016 論文集，Vol. 2016, pp. 36–43 (2016).

[55] 巡回セールスマン問題研究用ベンチマークデータセット：TSPLIB, `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[56] 松原　仁：次の一手形式によるコンピュータ将棋の評価，pp. 68–95, アマ 4 段を超える―コンピュータ将棋の進歩〈4〉，共立出版 (2003).

[57] Huang, S.-C. and Müller, M.: Investigating the Limits of Monte-Carlo Tree Search Methods in Computer Go, *Computers and Games*, pp. 39–48 (2014).

[58] GAT (Game AI Tournaments @UEC): 第 1 回 GAT2016，`http://minerva.cs.uec.ac.jp/cgi-bin/gat_uec/wiki.cgi?page=%C2%E81%B2%F3GAT2016`.

[59] 武藤孝輔，西野順二：ターン制戦略ゲームにおける UCT とファジィ評価の適用，日本知能情報ファジィ学会ファジィ システム シンポジウム 講演論文集， Vol. 31, pp. 226–229 (2015).

[60] 藤木　翼，村山公志朗，池田　心：ターン制ストラテジーのための状態評価型深さ限定モンテカルロ法における消極的行動の抑制，ゲームプログラミングワークショップ 2014 論文集，Vol. 2014, pp. 32–39 (2014).

[61] Bourg, D. M. and Seemann, G.: ゲーム開発者のための AI 入門，共立出版 (2005).

[62] 木村富宏：ターン制戦略ゲームへの深層学習の適用，情報処理学会研究報告ゲーム情報学（GI），Vol. 2019-GI-41, No. 5, pp. 1–8 (2019).

[63] 堀内　匡，藤野昭典，片井　修，椹木哲夫：経験強化を考慮した Q-Learning の提案とその応用，計測自動制御学会論文集，Vol. 35, No. 5, pp. 645–653 (1999).

[64] 木村　元，宮崎和光，小林重信：強化学習システムの設計指針，計測と制御，Vol. 38, No. 10, pp. 618–623 (1999).

[65] 荒井幸代，宮崎和光，小林重信：マルチエージェント強化学習の方法論 : Q-Learning と Profit Sharing による接近，人工知能学会誌 = Journal of Japanese Society for Artificial Intelligence，Vol. 13, No. 4, pp. 609–618 (1998).

[66] 宮崎和光，木村　元，小林重信：Profit Sharing に基づく強化学習の理論と応用，人工知能学会誌，Vol. 14, No. 5, pp. 800–807 (1999).

[67] 植村　渉，上野敦志，辰巳昭治：経験に固執しない Profit Sharing 法，人工知能学会論文誌，Vol. 21, No. 1, pp. 81–93 (2006).

[68] 宮崎和光，山村雅幸，小林重信：強化学習における報酬割当ての理論的考察，人工知能学会誌，Vol. 9, No. 4, pp. 580–587 (1994).

[69] Miyazaki, K.: Exploitation-Oriented Learning with Deep Learning – Introducing Profit Sharing to a Deep Q-Network –, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, Vol. 21, No. 5, pp. 849–855 (2017).

[70] Herrera, F., Charte, F. and Rivera, A. J.: *Multilabel Classification Problem Analysis, Metrics and Techniques*, Springer (2016).

[71] Cho, K., van Merrienboer, B., Gülçehre, Ç. et al.: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, *Computing Research Repository (CoRR)*, Vol. abs/1406.1078 (2014).

[72] OpenAI: OpenAI Gym, `https://gym.openai.com/`.

[73] Schaul, T., Quan, J., Antonoglou, I. and Silver, D.: Prioritized Experience Replay, *Computing Research Repository (CoRR)*, Vol. abs/1511.05952 (2016).

[74] Kimura, T. and Ikeda, K.: Designing Policy Network with Deep Learning in Turn-Based Strategy Games, *16th Advances in Computer Games Conference* (2019).

[75] Clark, C. and Storkey, A.: Training deep convolutional neural networks to play go, *International Conference on Machine Learning (ICML15)*, Vol. 37, p. 1766–1774 (2015).

[76] 木村富宏：ポリシーネットワークとバリューネットワークを併用した強化学習アルゴリズムのターン制戦略ゲームへの適用，ゲームプログラミングワークショップ 2019 論文集，Vol. 2019, pp. 73–79 (2019).

[77] Kimura, T. and Ikeda, K.: High-Performance Algorithms Using Deep Learning in Turn-Based Strategy Games, *12th International Conference on Agents and Artificial Intelligence* (2020).

[78] He, K., Zhang, X., Ren, S. and Sun, J.: Deep Residual Learning for Image Recognition, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778 (2015).

[79] Zagoruyko, S. and Komodakis, N.: Wide Residual Networks, *Computing Research Repository (CoRR)*, Vol. abs/1605.07146 (2016).

[80] Howard, A. G., Zhu, M., Chen, B. et al.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, *Computing Research Repository (CoRR)*, Vol. abs/1704.04861 (2017).

業績リスト

国内学会発表

- 木村富宏 : ポリシーネットワークとバリューネットワークを併用した強化学習アルゴリズムのターン制戦略ゲームへの適用，ゲームプログラミングワークショップ，査読なし，(2019).
- 木村富宏 : ターン制戦略ゲームへの深層学習の適用，情報処理学会研究報告ゲーム情報学 (GI)，査読なし，(2019).
- 木村富宏，池田心 : ターン制戦略ゲームにおけるベンチマークマップの提案，ゲームプログラミングワークショップ，査読なし，(2016).
- 木村富宏，安達雅春 : インスタンスに応じた領域分割による TSP の近似解法 ～ P 型フーリエ記述子を利用した解法 ～，電子情報通信学会 非線形問題研究会，査読なし，(2012).

国際会議発表

- Kimura, T. and Ikeda, K.: High-Performance Algorithms Using Deep Learning in Turn-Based Strategy Games, 12th International Conference on Agents and Artificial Intelligence, 査読あり（ポスター発表）(2020).
- Kimura, T. and Ikeda, K.: Designing Policy Network with Deep Learning in Turn-Based Strategy Games, 16th Advances in Computer Games Conference, 査読あり (2019).
- Kimura, T. and Adachi, M.: A new heuristic algorithm for Traveling Salesman Problem using P type Fourier descriptor, International Conference of Information Science and Computer Applications ICISCA, 査読あり (2012)