卒 業 研 究 報 告 題 目

# DQNにおける Lookahead-Replicateの有効性の検証

# Verifying the Effectiveness of Lookahead-Replicate in DQN


指 導 教 員　　出口 利憲 教授

岐 阜 工 業 高 等 専 門 学 校　電気情報工学科

2021E29 保浦歩太


令 和 8 年（ 2 0 2 6 年 ）　　2 月 1 3 日 提 出

# Abstract

The purpose of this study is verify the effectiveness of Lookahead-Replicate in Deep Q-Network, a recently proposed method for updating target networks. Unlike previous methods, which constrain matching in parameter space ($\theta = w$), Lookahead-Replicate constrains matching in value function space ($v_\theta = v_w$). This facilitates learning across a wider solution space.

Using the imperfect information game "Penguin Party", we optimized the Lookahead-Replicate parameters $K_L$, $K_R$, batch size, learning rate, and optimizer. Then, we compared the performance of this method with that of previous methods. Experimental results showed that Lookahead-Replicate achieved a win rate of 76.67% against random players, outperforming previous methods such as Fixed Q-Target (62.39%) and Soft Update (69.40%). Furthermore, we demonstrated that learning is possible even when the online network and target network has different network architectures, by leveraging the property of this method that does not require perfect parameter matching.

# 目 次

# 第1章　序論

近年、機械学習は自動運転やロボット制御などの分野で注目されている。特にボードゲーム分野での進歩は目覚ましく、囲碁では「Alpha GO（2014）」、将棋では「Ponanza（2009）」が人間のトッププロ相手に勝利し話題となったことは記憶に新しい。

機械学習の手法の一つにQ学習がある。Q学習は試行錯誤を繰り返すことで、定義された価値を最大化する行動を取るように学習を進める。Deep Q-Network(DQN)は、2015年にDeepMindによって開発された強化学習の手法で、Q学習とディープラーニングを組み合わせたものである。これにより、複雑なタスクでも解くことができる。DQNでは学習を安定させるためにTarget Networkという技法が用いられる。学習を行うメインのネットワークの他にもう一つネットワークを用意し、そこにパラメータをコピーすることで安定化を図っている。

本研究では、DQNのTarget Network更新アルゴリズムの新手法であるLookahead-Replicate[1)]をDouble DQNに適用し、適切なハイパーパラメータの探索と既存技法との比較を行う。

# 第2章　実験で使用した技法

## 2.1　機械学習[2)3)]

機械学習とは、人間の学習に相当する仕組みをコンピュータ等で実現するものである。コンピュータに大量のデータを読み込ませ、様々なアルゴリズムに基づいてデータ内のルールやパターンを学習させる。そのパターンやルールを未知の新たなデータに適用することで識別や予測を可能にするデータ解析技術である。機械学習の手法は「教師あり学習」と「教師なし学習」、「強化学習」の3つに分けることができる。

1. 教師あり学習

   教師あり学習は、読み込んだデータから「入力と出力の関係」を学習することで、データ間の関係を認知させる学習手法である。事前に正解のラベルを付けたデータを使用する。与えられたデータの中で「正解」ラベルの付与されたデータの特徴と、それ以外のデータの特徴をコンピュータが自動で識別し、識別する力を向上させていくことで、入力に対する「正解」のデータを出力できるようになる。

2. 教師なし学習

   教師なし学習は、ラベルの付与されていない学習データを用いて、データセットのパターンからデータの関係を認知させる学習手法である。例として2012年にGoogle社が、Web上の画像や動画を1週間読み取り続けることで、教師なし学習により自律的に「猫」を識別できるAIを開発している。このように、与えられたデータからコンピュータが特徴を見いだし、未知のデータに対しても予測や識別を行うことができるようになる。

3. 強化学習

   強化学習は、正解を与える代わりに将来の価値を最大化することを学習するモデルである。コンピュータが一定の環境で試行錯誤を繰り返すことで学習データを収集し、良い結果に結びつく行動をするように学習する。囲碁のような、人間が必ずしも正解を予測できない場合でも学習可能であるため、人間を超える力を身につけることが期待される。

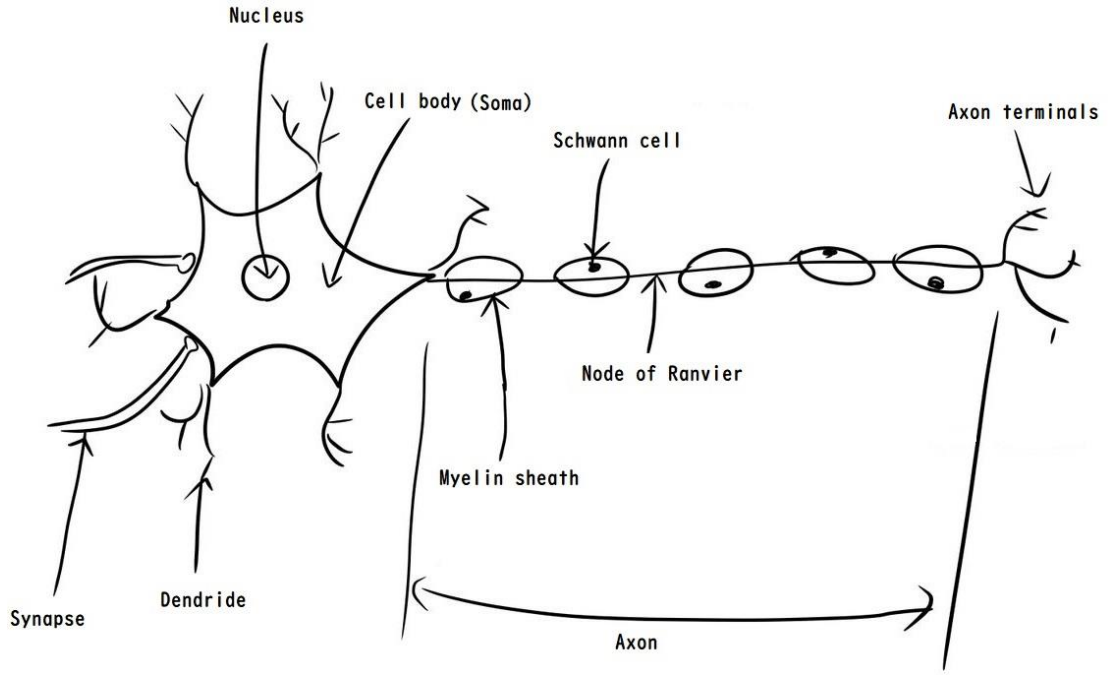


Figure 2.1　Neuron.

## 2.2　ニューラルネットワーク (Neural Network)

### 2.2.1　ニューロン [4)5)]

ニューロン（神経細胞）とは、生物の脳を構成する神経細胞のことである。Figure 2.1 のような構造を持ち、外界からの入力を受け取り、変換・中継を行い、運動指令を送る役割を担っている。人間の脳は千数百億のニューロンからなっている。

ニューロンは主として細胞体・軸索・樹状突起からなっている。軸索は細長い構造体で、活動電位と呼ばれる電気信号を生成して信号を他のニューロンに送る、ニューロンの出力装置と呼べるものである。樹状突起は、他のニューロンからの信号を受け取る受信部である。細胞体の中心には核が存在し、活動に必要なタンパク質などが作られる。

### 2.2.2　ニューラルネットワーク [6)]

ニューラルネットワーク（人工ニューラルネットワーク、Artificial Neural Network とも）は、プログラム上で作成された人間の脳に似た構造内で相互接続された、人間のニューロンの仕組みを模したパーセプトロンを用いて学習を行うシステムである。ニューラルネットワークは非線形関係のモデル化に特に適しており、パターン認識やオブジェクト・音声などの分類に用いられる。

ニューラルネットワークの構造を Figure 2.2 に示す。ニューラルネットワークは入力

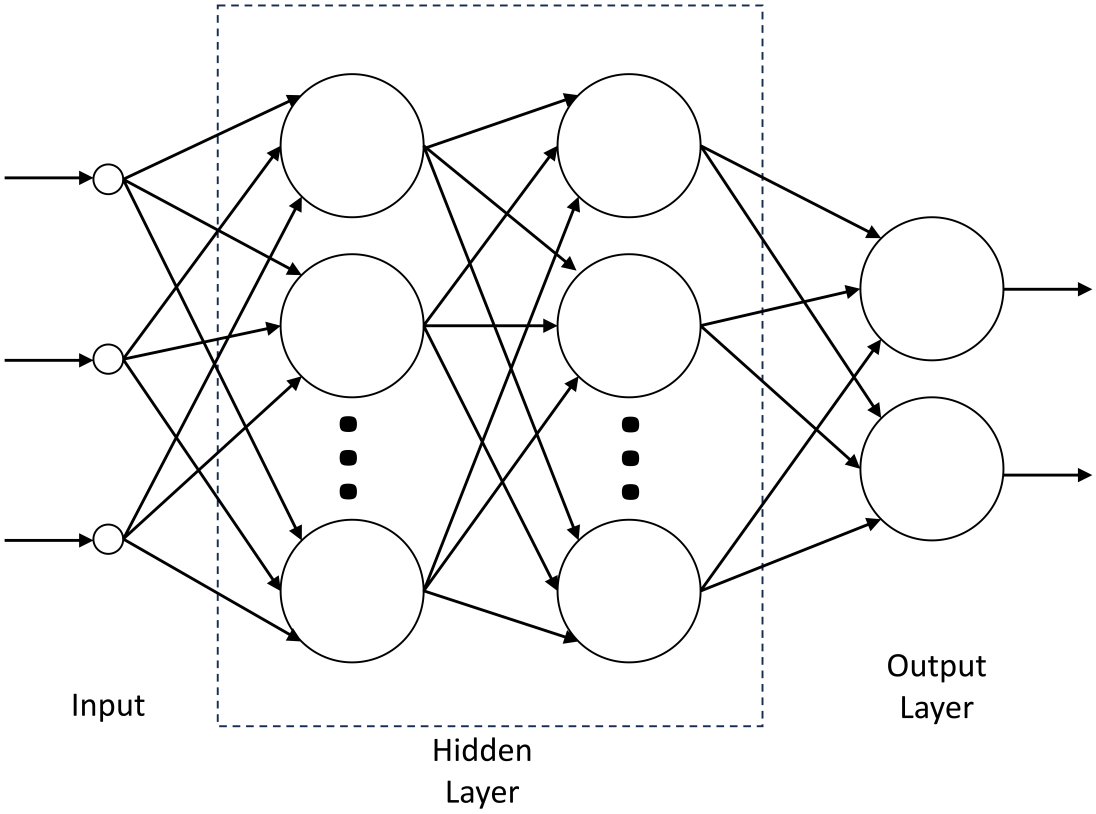


Figure 2.2　Neural network.

層、1 つ以上の中間層、出力層の 3 つの層から構成される。各層には複数のパーセプトロンが存在し、各層のノードは前の層の全てのノードの出力を入力として使用し、全てのパーセプトロンが異なる層を通じて相互接続される。中間層が 2 つ以上あるニューラルネットワークのことを特にディープニューラルネットワーク（Deep Neural Network）と呼ぶ。

パーセプトロンは Figure 2.3 に示す構造を持つ。入力（$x_1, x_2, ...x_n$）が与えられ、入力と同じ数だけ重み $w$ が存在する。$b$ はバイアスと呼ばれる値で、パーセプトロンにつき 1 つ存在する。重み $w$ とバイアス $b$ はパーセプトロンの内部変数である。$f$ は活性化関数と呼ばれる関数である。

### 2.2.3　活性化関数 [7)8)]

活性化関数はニューラルネットワークにおいて入力の値を変形させて出力へつなげる関数である。活性化関数を使用することで値間の非線形関係を表現可能になり、モデル

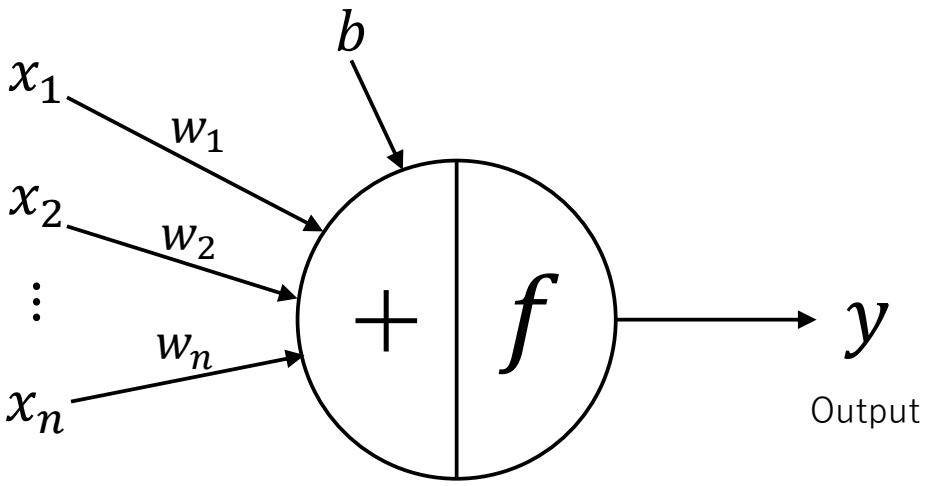


Figure 2.3　Perceptron.

の表現力が向上する。任意の数学関数を活性化関数として利用可能だが、よく使用される活性化関数にはシグモイド、tanh（双曲線正接関数）、ReLU がある。今回はその中でも ReLU を用いた。

正規化線形ユニット（Rectified Linear Unit, ReLU）活性化関数は、入力値 $x$ が 0 未満の時は 0 を返し、0 以上の時は入力値を返すというアルゴリズムである。そのため数学的には max 関数を用いて表すことができ、Figure 2.4 のようなグラフとなる。ReLU はトレーニング中に勾配消失の問題の影響を受けにくいため、シグモイド関数や tanh よりも滑らかな関数となり、また計算も簡単である。

$$F(x) = \max(0, x) \tag{2.1}$$

## 2.3　強化学習 (Reinforcement Learning)[9)]

強化学習は、人間の介入の必要なくエージェントが動的な環境とやりとりすることで学習を行う機械学習の手法である。エージェントは行動、観測値、報酬に基づき、受け取る累計報酬が最大になるように学習アルゴリズムを用いて方策パラメータを更新する。そして方策パラメータを基にして行動を決定し、実行する。一般的に方策は深層ニューラルネットワークなどの関数近似器である。強化学習のアルゴリズムにはモンテカルロ

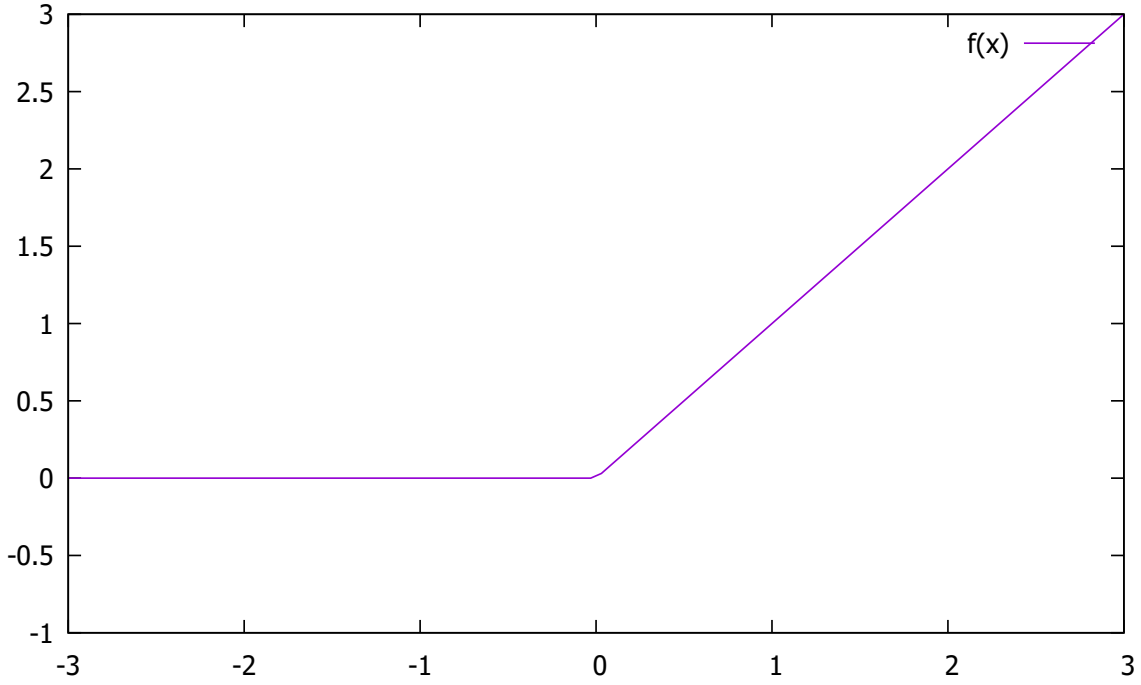


Figure 2.4　ReLU.

法、Q 学習、DQN、PPO、Soft Actor-Critic などがある。

### 2.3.1　Q 学習 [10)11)]

Q 学習とは、モデルフリーの強化学習の一種である。行動の結果を経験することで、マルコフ領域において最適な行動を取る能力を得ることができる。

Q 学習では、エージェントは以下の手順を踏んで経験を貯めていく。

1. 現在の状態 $x_n$ を観測する
2. 行動 $a_n$ を選択し実行する
3. 次の状態 $y_n$ を観測する
4. 即時報酬 $r_n$ を受け取る
5. 学習率 $\alpha_n$ を用いて $Q_{n-1}$ の値を更新する。

Q 値は以下のように定義される。$R$ は報酬、$\gamma$ は割引率、$\pi$ はポリシー（方策）、$S_t = s$ は状態、$A_t = a$ は行動を意味する。

$$Q_\pi(s,a) = \mathbf{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s, A_t = a] \tag{2.2}$$

しかし上式の計算では、将来得られる全ての報酬を合計する必要があり計算コストがかかる。これを効率よく計算するために用いられるのがベルマン方程式である。ある状

態 $s$ における価値関数と、その次の状態 $s'$ における価値関数を関連付ける方程式となっている。ベルマン方程式が成り立つ前提として、学習対象がマルコフ決定過程に従う必要がある。

$$Q(S_t, a) \leftarrow (1-\alpha)Q(S_t, a) + \alpha(R + \gamma \max Q) \tag{2.3}$$

### 2.3.2 $\epsilon$-greedy[12)]

$\epsilon$-greedy 方策は、Q 値から行動を決定する際に最適行動ばかりを取らず、一定の確率 $\epsilon$ でランダムな行動を取る手法のことである。最適行動ばかりを取っていると未知の行動を発見できないため、一定確率でランダムな手を取らせることでより優れた行動を発見することができるようになる。

### 2.3.3 Deep Q-Network[13)]

Deep-Q Network は、強化学習における代表的な手法である Q 学習を拡張し、行動価値関数（Q 関数）の近似にディープニューラルネットワークを用いる手法である。DQN では Q 関数を、ReLU ネットワークに代表されるパラメータを持つ関数として定義しそのパラメータを学習することで、Q 学習よりも高次元の状態空間に対応する。DQN は Neural Fitted Q-Iteration アルゴリズムの一種として解釈可能で、ディープニューラルネットワークを用いた強化学習において高い性能と安定性を実現している。

### 2.3.4 Double DQN[14)]

Q 学習には、特定の条件下で価値関数を過大に見積もってしまうという既知の問題がある。これは Q 学習の更新式に max 演算子が含まれており、推定値に誤差やノイズが含まれている場合に過大評価された値が選択され、学習されることが原因である。過大評価により誤った方策が学習され、性能を低下させる。この過大評価現象は DQN でも同様に発生する。

この問題を解決するため、Double Q-Learning の考え方を DQN に適用したものが Double DQN である。従来の DQN では行動の選択と評価の計算にターゲットネットワークを使用していたが、Double DQN では行動の選択にオンラインネットワークを使用し、行動の評価にターゲットネットワークを使用している。Double DQN の式は以下の通り

である。状態 $S_{t+1}$ における行動 $a$ の Q 値の最大値集合をオンラインネットワーク $\theta_t$ を用いて求め、最大値集合に対してターゲットネットワーク $w_t$ を用いて評価を行う。

$$Y_t^{DoubleDQN} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \theta_t); w_t) \tag{2.4}$$

### 2.3.5 Experience Replay[15)16)]

Experience Replay（経験再生）は、強化学習において収束を早め、試行錯誤のプロセスを効率化するために提案された手法である。強化学習において得られる現在の状態 $x$、行動 $a$、次の状態 $y$、報酬 $r$ からなる経験 $(x, a, y, r)$ を Replay Buffer（または Replay Memory）と呼ばれるメモリに一度保存し、ランダムにサンプリングしてミニバッチとして学習に用いる。この手法によりサンプル効率が向上し、また学習速度も向上する。

### 2.3.6 損失関数[17)18)]

損失関数とは、入力からモデルが算出した値と正解の値の誤差を数値的に示す指標である。機械学習においてはこの誤差を小さくすることを目的に学習が行われる。

Deep Q-Network では、損失関数として Huber 損失関数が用いられる。Huber 損失関数は平均二乗誤差と平均絶対誤差の利点を組み合わせたものであり、外れ値の影響が小さく微調整しやすいという特徴がある。

$$L(x, y) = \begin{cases} \frac{1}{2}(x-y)^2 & (|x-y| \leq 1) \\ |x-y| - \frac{1}{2} & (|x-y| > 1) \end{cases} \tag{2.5}$$

### 2.3.7 One-Hot エンコーディング[19)]

One-Hot エンコーディングとは、基本的にはダミー変数を用いてデータを変換することである。数値として扱えない型のデータを 0 または 1 で表現して数値に変換するために使用される。例として Table 2.1 のように、「ある人の血液型が何であるか」というデータを数値に変換する際に使用することができる。

Table 2.1　Example of One-Hot Encoding.

| 血液型 | A 型 | B 型 | O 型 | AB 型 |
|---|---|---|---|---|
| A さん | 1 | 0 | 0 | 0 |
| B さん | 0 | 1 | 0 | 0 |
| C さん | 0 | 0 | 1 | 0 |
| D さん | 0 | 0 | 1 | 0 |
| E さん | 0 | 0 | 0 | 1 |

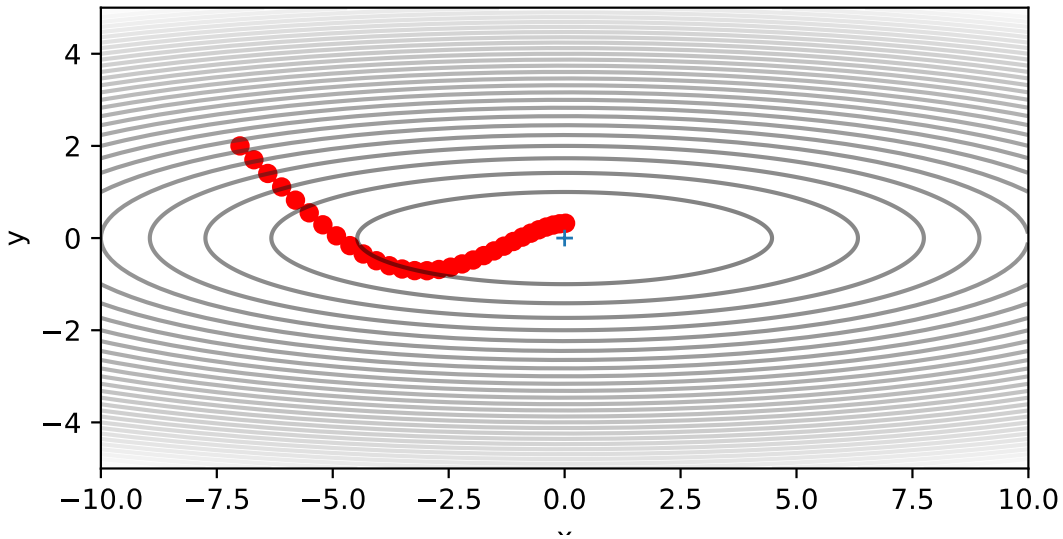


Figure 2.5　Optimization path of Adam.

### 2.3.8　Adam[20)]

Adaptive Moment Estimation（Adam）は、機械学習における最適化アルゴリズムの一種である。勾配降下法を改良したもので、パラメータごとに学習率を自動的に調整する。過去の勾配の平均と勾配の分散（二乗平均）を求め、平均を分散の平方根で割った値で学習率を調整する。勾配の大きさに応じた加減速を行うため、安定した学習を行うことができる。手動で細かいチューニングをする必要が無いため、多くの機械学習フレームワークで実装されている。Adam は Figure 2.5 のように、スムーズな経路で解へと向かう。

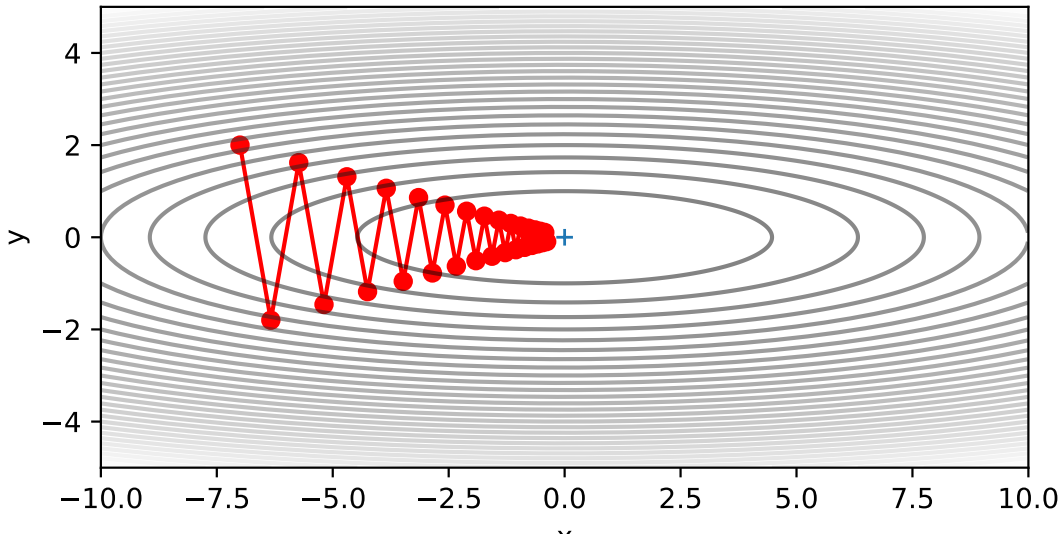


Figure 2.6　Optimization path of SGD.

### 2.3.9　SGD[21)]

Stochastic Gradient Descent（SGD、確率的勾配降下法）は、機械学習における最適化アルゴリズムの一種である。学習データの中から一つを取り出して勾配を計算し、誤差を減らす方向へパラメータを更新する手法である。全データを用いて勾配を計算する最急降下法と比べて計算量が小さく、そのため大規模なデータに対して効率的に学習を進められる。勾配に大きなノイズが含まれるが、その揺らぎによって探索範囲が広がり、局所的な解から抜け出しやすくなる場合がある。機械学習では振動を抑えて収束を早めるために慣性（Momentum）を追加することが多い。素の SGD は Figure 2.6 のようにジグザグとした経路でかなり非効率的であるが、慣性（Momentum）を適用すると Figure 2.7 のように、よりなだらかな経路で解へと向かう。

### 2.3.10　Target Network[22)23)24)]

DQN では、オンラインネットワークを更新することで方策を学び、Q 値を予測する。しかし、学習中に同じネットワークから現在と将来の Q 値を計算するため、パラメータが頻繁に変わると発散して学習が不安定になる。これを解決するために、Target Network という固定された別のネットワークを用意し、そのネットワークから将来の Q 値を予測することで、安定して学習を行うことができる。

ターゲットネットワークの更新手法として、DQN では Fixed Target Q-Network（Hard Update とも呼ばれる）が用いられる。これは一定間隔でオンラインネットワークのパラ

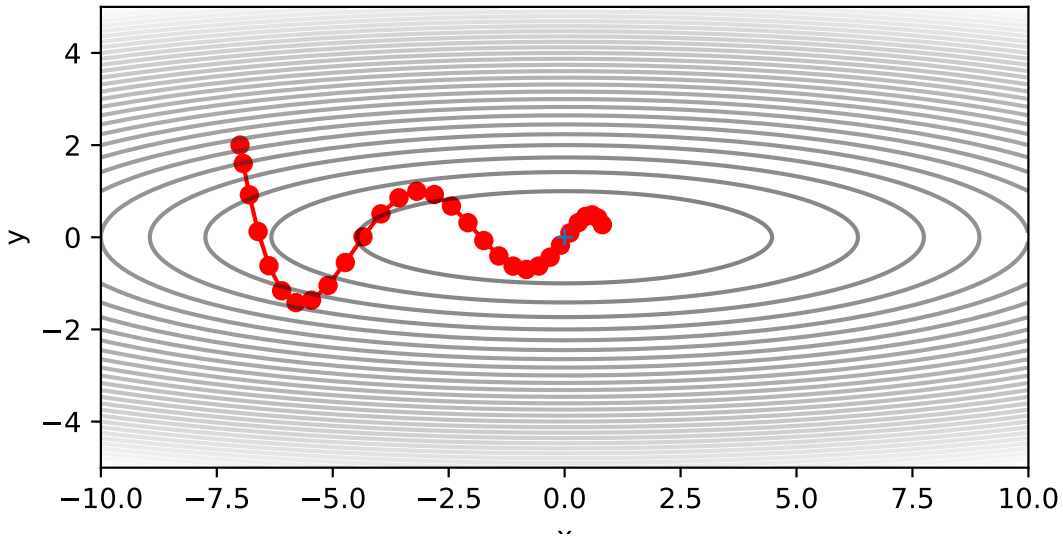


Figure 2.7　Optimization path of SGD with momentum.

メータをターゲットネットワークにコピーする手法である。一方、学習毎にターゲットネットワークを徐々に更新していく手法を Soft Update と呼ぶ。

### 2.3.11　Reward Clipping[23)]

Reward Clipping は、強化学習における報酬について、報酬を [-1,1] の範囲にクリップする手法のことである。Q 値の急激な増大を抑制し、勾配が安定する効果がある一方、報酬の大小を区別できなくなるという弱点がある。

### 2.3.12　Lookahead-Replicate[1)]

Lookahead-Replicate（LR、先読み再現）は Kavosh Asadi らが 2024 年に提案したターゲットネットワークの更新アルゴリズムである。

強化学習において、正確な価値関数を学習することは、そのアルゴリズムに関わらず中核となる課題である。価値関数 $v$ の重要な特性は、ベルマン演算子 $\mathcal{T}$ の不動点であること、すなわち $v = \mathcal{T}v$ と満たすことである。大規模な状態空間を持つ場合、ニューラルネットワークを用いた関数近似により、パラメータ $\theta$ を持つ価値関数 $v_\theta$ を学習する手法が一般的である。DQN などの既存のアルゴリズムにおいては、オンラインネットワーク（パラメータ $w$）とターゲットネットワーク（パラメータ $\theta$）の 2 つを維持し、ターゲットネットワークを定期的（Fixed Q-Target）または徐々に（Soft Update）更新する。これらは、ベルマン方程式 $v_w = \mathcal{T}v_\theta$ を満たすという「関数空間の制約」と、両方のネット

ワークが同じパラメータを持つ（$\theta = w$）という「パラメータ空間の制約」の双方を満たすようになっている。これに対し Asadi らは、単一の価値関数を学習するという目的においてパラメータ空間での一致（$\theta = w$）は過剰な制約であると考え、代わりに $v_\theta = v_w$ という「価値関数空間での一致」を制約とすることを提案した。この概念を実現するために設計されたのが Lookahead-Replicate である。

Lookahead-Replicate は以下の 2 ステップを交互に繰り返す。

1. Lookahead（先読み）

   ベルマン方程式の解決を目的とするステップである。オンラインネットワーク $w$ を更新し、価値関数 $v_w$ を現在の推定値に基づくターゲット値 $\mathcal{T}v_\theta$ に近づける。損失関数としては $||v_w - \mathcal{T}v_\theta||^2$ を最小化する。この部分は既存の時間差分学習の手法と変わらない部分である。

2. Replicate

   価値関数空間が等価になる（$v_\theta = v_w$）ように、二つの価値関数の二乗誤差 ($||v_\theta - v_w||^2$）が最小になる勾配降下法を用いてターゲットネットワーク $\theta$ を更新する。この間、オンラインネットワーク $w$ は固定される。既存手法がパラメータの複製（Duplicate）を目指していたのに対し、価値関数の再現（Replicate）を目指している。

Lookahead-Replicate において重要なのが、解の空間 $\mathcal{F}$ が従来手法よりも広いという点である。ターゲットネットワークを持たない基本的な時間差分学習は下式のような解の集合を持ち、自身の価値関数の出力がベルマン方程式を満たすパラメータ $\theta$ の集合である。$\Theta$ はパラメータ空間である。

$$\mathcal{F}_{single} = \{\theta \in \Theta \mid v_\theta = \mathcal{T}^\pi v_\theta\} \tag{2.6}$$

ターゲットネットワークを持つ DQN のような学習では下式のような解の集合を持ち、ベルマン方程式を満たしつつパラメータが完全に一致するペアに限定される。

$$\mathcal{F}_{pair} = \{(\theta, w) \in \Theta \times \Theta \mid v_w = \mathcal{T}^\pi v_\theta \text{ and } \theta = w\} \tag{2.7}$$

これに対し、Lookahead-Replicate では下式のような解の集合を持ち、ベルマン方程式を満たし $v_\theta = v_w$ を満たすパラメータのペア $(\theta, w)$ である。

$$\mathcal{F}_{value} = \{(\theta, w) \in \Theta \times \Theta \mid v_w = \mathcal{T}^\pi v_\theta \text{ and } v_\theta = v_w\} \tag{2.8}$$

ニューラルネットワークのような過剰パラメータ化されたモデルでは、異なるパラメータであっても同一の関数を表すことが可能であるため、これらの集合には $\mathcal{F}_{pair} \subset \mathcal{F}_{value}$ と

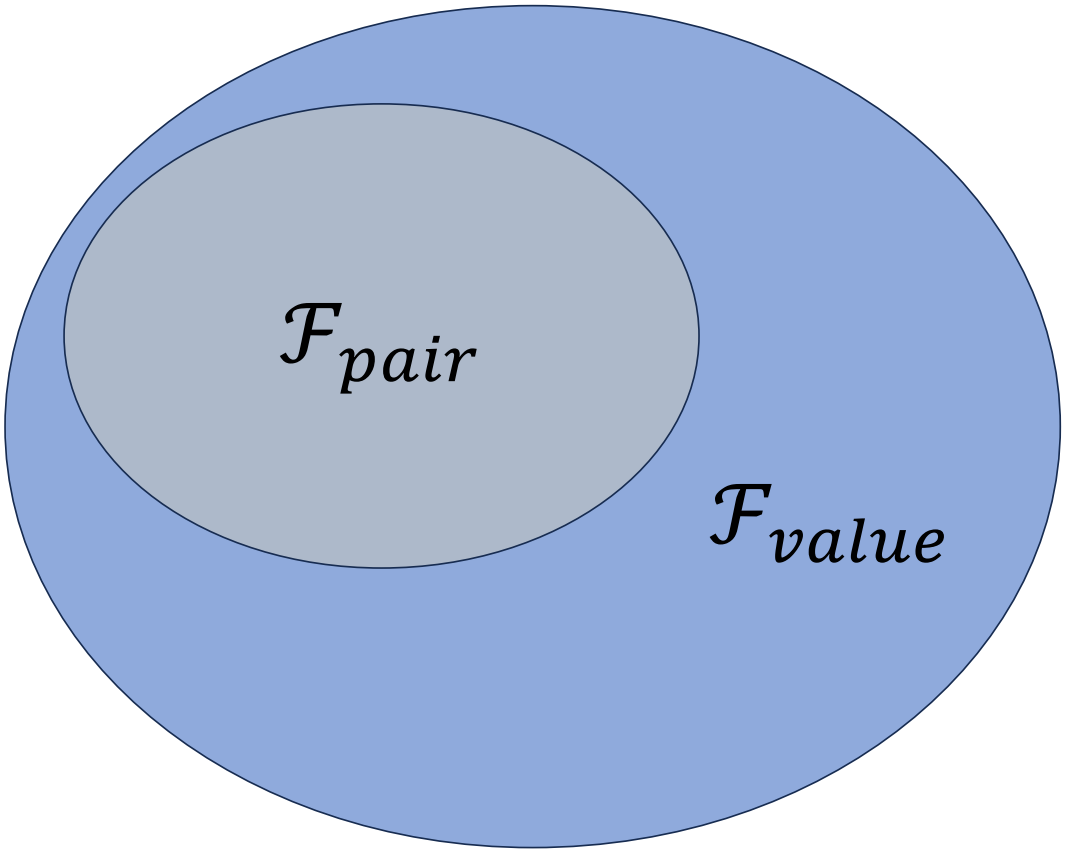


Figure 2.8　Schematic diagram of the solution space.

いう包含関係が成立する。また、その解空間の大きさについては $|\mathcal{F}_{pair}| \leq |\mathcal{F}_{value}|$ であることが示されている。つまり、Lookahead-Replicate が得ることのできる解は Figure 2.8 のように従来手法で得られる解全てが含まれており、より広い探索範囲で解を発見できるアルゴリズムであるといえる。

Asadi らの論文では、パラメータが一致しない（$\theta \neq w$）状態であっても、$v_\theta = v_w$ となるペアに収束することが確認されている。また、Atari ゲームを使用した実際の学習においても、DQN から発展した Rainbow というアルゴリズムよりも優れたパフォーマンスを示している。

### 2.3.13　TensorFlow[25)26)]

TensorFlow は、Google Brain により開発された機械学習向けのオープンソースプラットフォームである。Tensor（テンソル）と呼ばれる多次元配列としてデータを入力し、学習を行う。自然言語、画像、強化学習など様々なモデルを学習するためのライブラリが用意されている。学習を行うためのデバイスとして CPU、GPU、TPU がサポートさ

れており、様々なプラットフォームで実行可能である。主に Python で動作し、C++や Java 等で実行するための API も用意されている。自然言語処理や画像認識、手書き認識など、様々な機械学習モデルを作成することができる。

強化学習向けのライブラリとして、TF-Agents が用意されている。学習エージェントとして DQN、PPO、SAC、QT-Opt などが用意されており、環境として ATARI ゲームが用意されている。また様々なネットワークやリプレイバッファ等、強化学習を行うために必要なものが一通り揃っている。

# 第3章　実験

## 3.1　目的

この研究の目的は、Lookahead-Replicate のアルゴリズムを Deep Q-Network に実装し、Penguin Party における適切なパラメータの探索を行い、既存手法との比較を行うことである。

## 3.2　ゲームの概要

Penguin Party は、2 人のプレイヤーが戦う対戦ゲームである。プレイヤーは手札を使ってピラミッドを築き、できるだけ早く手札を全て使い切ることを目指す。

カードには赤・青・緑・黄・紫の 5 色あり、緑が 8 枚、そのほかは 7 枚の計 36 枚が用意されている。ゲーム開始時に 36 枚の中から無作為に 8 枚を抜き、残った 28 枚をプレイヤーに 14 枚ずつ均等に配布する。

盤面を Figure 3.1 に示す。盤面となるピラミッドは 7 段あり、プレイヤーは 1 段目から順番にカードを積み上げる。最初のカードは 1 段目の中央に置かれ、それ以降土台となるカードは既に置かれているカードの左右に配置する。1 段目は最大 7 枚まで配置することが可能である。2 段目以降にカードを配置する場合、その場所の 1 段下の左右にカードが配置されている必要があり、さらに配置するカードの色が土台となる 2 枚のカードのうち少なくともどちらか一方の色と同じでなければならない。

プレイヤーは交互に手番を行い、自分の手札の中に配置ルールを満たすものがあれば配置する。もし手札に配置可能なカードがなければパスし、相手に手番を回す。双方がパスするか、手持ちのカードがなくなればラウンド終了となる。得点の計算はペナルティ方式で行われ、ラウンド終了時に手持ちに残ったカードの枚数が加算される。もし全て使い切ることができたら得点から 2 点減点される。最終的に、累計得点の少なかったものが勝者となる。

片方のプレイヤーからは自身の手札と盤面の情報しか観測することができない。プレイヤーは自身の手札にある各色の枚数と、盤面に出ているカードの情報のみを基に思考判断してプレイする。

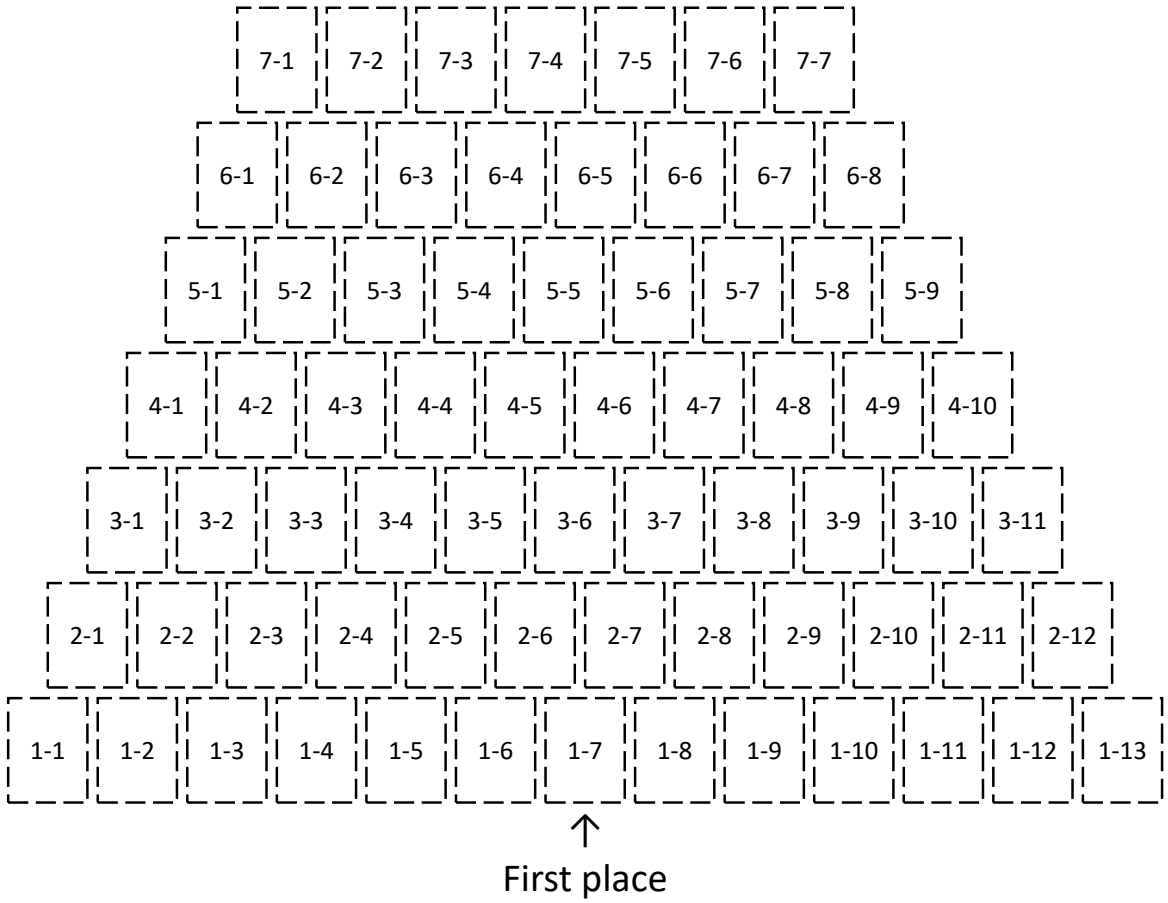


Figure 3.1　Penguin Party board.

## 3.3　Double DQN

### 3.3.1　状態

Double DQN などの強化学習においては、状態を 0/1 に離散化することでニューラルネットワークが非線形な特徴を学習しやすくなり、性能が向上する。Double DQN では自身の手札情報、盤面情報と相手の手札情報を状態として用いる。以下のようにベクトル化した各状態を連結し、合計 273 次元のベクトルを入力する。

1. プレイヤーの手札

   手札の枚数を Table 3.1 のようにベクトル化する。手札に来る可能性のある 1 色あたりの最大の枚数は 8 枚であるため、0 枚〜8 枚の状態を One-Hot ベクトルで表現する。9 × 5 色で 45 次元のベクトルとなる。

2. 盤面情報

   盤面情報を Table 3.2 のようにベクトル化する。各マスごとに「空き、緑、赤、青、黄、紫」の有無を 0/1 で表現する。これを全 28 マス分連結し、6 × 28 マスで合計

Table 3.1　Vectorization of the number of cards.

| Number of cards | Vector |
|---|---|
| 0 | [1,0,0,0,0,0,0,0,0] |
| 1 | [0,1,0,0,0,0,0,0,0] |
| 2 | [0,0,1,0,0,0,0,0,0] |
| 3 | [0,0,0,1,0,0,0,0,0] |
| 4 | [0,0,0,0,1,0,0,0,0] |
| ... | ... |

Table 3.2　Vectorization of board position.

| Status | Vector |
|---|---|
| Blank | [1,0,0,0,0,0] |
| Green | [0,1,0,0,0,0] |
| Red | [0,0,1,0,0,0] |
| Blue | [0,0,0,1,0,0] |
| Yellow | [0,0,0,0,1,0] |
| Purple | [0,0,0,0,0,1] |

168 次元のベクトルとなる。

3. 相手の手札情報

相手の手札情報は本来プレイヤー側から観測することはできない。そのため、相手の残り手札枚数と未観測の各色の枚数を情報として与える。Table 3.1 と同様の手法を用いてベクトル化する。ルール上の各色の初期枚数である緑 8 枚とその他 7 枚から、プレイヤーが観測可能な自身の手札情報と盤面の情報から数えた各色の枚数を引く。残り手札枚数は、初期枚数の 14 枚から相手の手番が終わるごとに 1 を引く。残り手札枚数は 15 次元、未観測の各色の枚数は 45 次元で合計 60 次元の One-Hot ベクトルで表現する。

### 3.3.2　報酬

報酬として、ラウンド勝利時に+1、敗北時に-1 を与える。中間報酬は与えない。

### 3.3.3　ハイパーパラメータ

学習に使用するパラメータを以下に示す。

- 学習率 $1.0 \times 10^{-4}$
- 割引率 0.99
- バッチサイズ 64
- 初期 $\epsilon$ 1.0
- $\varepsilon$ の減少率 0.9995
- 最小 $\epsilon$ 0.01

学習率は大きすぎると学習が不安定になり、小さすぎると遅くなることで知られている。今回採用した学習率 $1.0 \times 10^{-4}$ という値は機械学習で一般的な値の範囲であり、安定的な学習が期待される。

割引率には 0.99 という値を採用した。この値は、将来の報酬を現在とほとんど同じ価値で評価する意味がある。Penguin Party は長期的な思考が必要なゲームであることから、高い割引率を採用した。

初期 $\epsilon$ は 1 とし、最小 $\epsilon$ は 0.05、減少率は 0.995 とした。$\epsilon$ は Figure 3.2 のように減少する。Penguin Party は毎ターンお互いに異なる手札が配られるため、早期に探索をやめてしまうと十分な経験が集まらない可能性があると考え、緩やかに減少するよう設定した。

### 3.3.4　ネットワーク構造

今回学習するモデルの構造を Figure 3.3 に示す。入力層には 273 次元の One-Hot ベクトルを入力する。入力層の状態数はベクトルのサイズと同じ 273 とし、出力層からは各行動の Q 値が出力される。中間層のニューロンの数は 512 とし、活性化関数には ReLU を使用する。中間層は 2 層用意する。

出力層からは全 28 マス 5 色についての Q 値が出力される。このとき、ルールに則さない行動についてマスク処理を行う。具体的には、自身の手札と盤面から合法手を判定し、出力された Q 値のうち合法手以外の部分の Q 値をマイナス無限大（-inf）で埋めている。

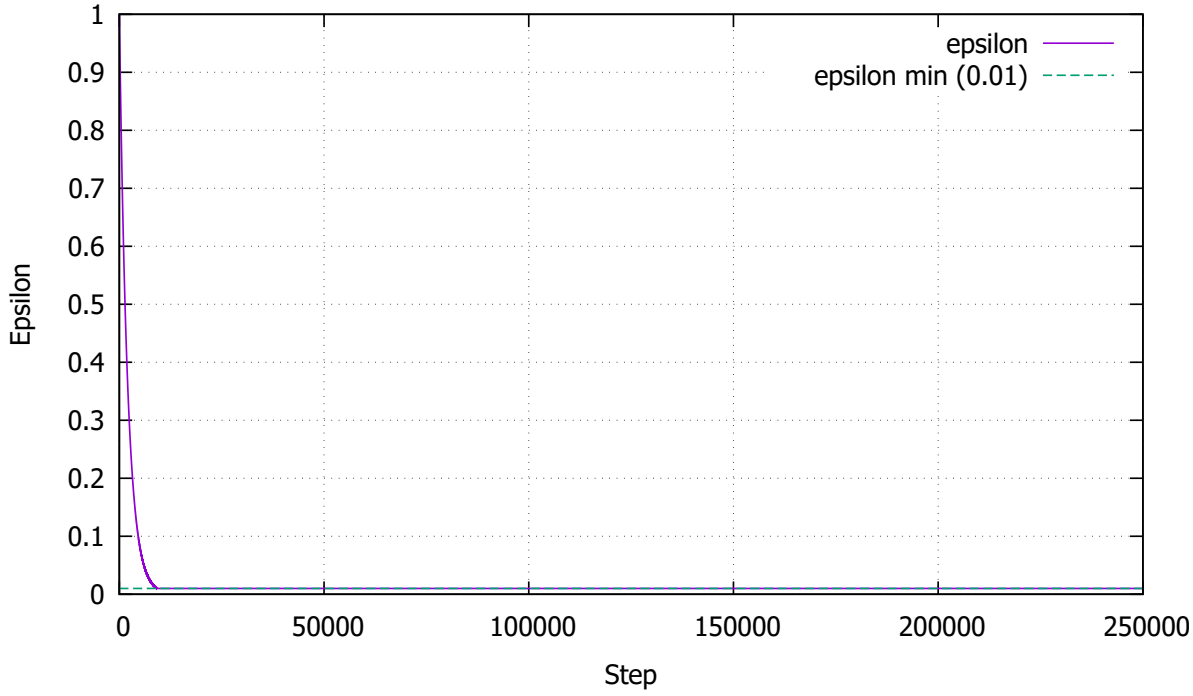


Figure 3.2　Epsilon decay.

ネットワークの最適化アルゴリズムには Adam を、損失関数には Huber 損失関数を使用する。

### 3.3.5　Lookahead-Replicate の実装

Lookahead-Replicate の実装は、Double DQN の既存の方法を置き換える形で行われる。ハイパーパラメータは以下の通りである。

- $K_L$ 1,000
- $K_R$ 500
- Batch Size 64
- Optimizer(Learning Rate) Adam($1.0 \times 10^{-4}$)

Lookahead ステップでは、ベルマン方程式を満たすようにオンラインネットワークのパラメータ $w$ を更新する。Lookahead ステップの後、Replicate ステップでターゲットネットワークをオンラインネットワークに追従させる更新を行う。Replicate ステップ中、オンラインネットワークは固定され、教師データとして扱われる。Lookahead ステップの頻度と Replicate ステップの回数はそれぞれ $K_L, K_R$ という独立したハイパーパラメータで定められる。ターゲットネットワークの最適化にはオンラインネットワークと同様に Adam を使用し、学習率も同様の $1.0 \times 10^{-4}$ とした。バッチサイズも同様に 64 とした。

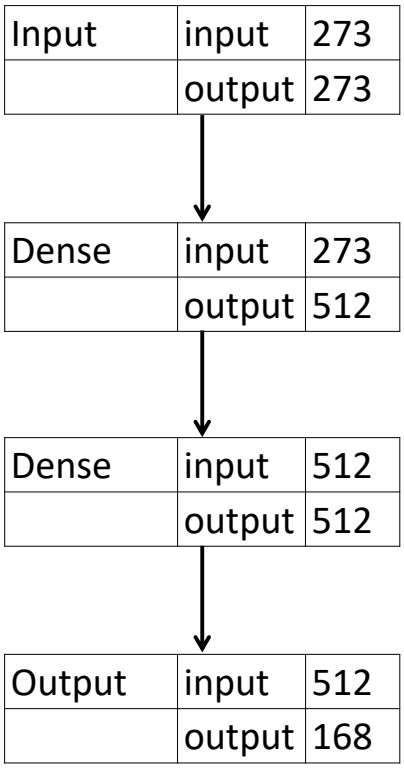


Figure 3.3　Learning network structure.

Penguin Party は Asadi らの論文で使用された Atari ゲームと比較して状態数が小さいため、論文のハイパーパラメータをそのまま使用することは最適ではないと考えられる。そのため、本研究ではこれらのハイパーパラメータについて独自に調整を行う。

### 3.3.6　学習

学習はランダムに手を打つプレイヤーを相手として行う。学習中の勝率は、1,000 ラウンドごとに累計の勝利数を Iterations 数で割った値を勝率とする。学習後の評価では、学習と同様にランダムに手を打つプレイヤーを相手として、1 万ラウンド対戦を行った際の累計の勝率を計算する。

## 3.4　$K_L$ の決定

ここでは、$K_L$ に注目して調整を行う。$K_L$ と $K_R$ の比は 0.25 に固定し、$K_L$ を 200、500、700、1,000、2,000 に変化させる。$K_R$ はそれぞれ 50、125、175、250、500 となる。

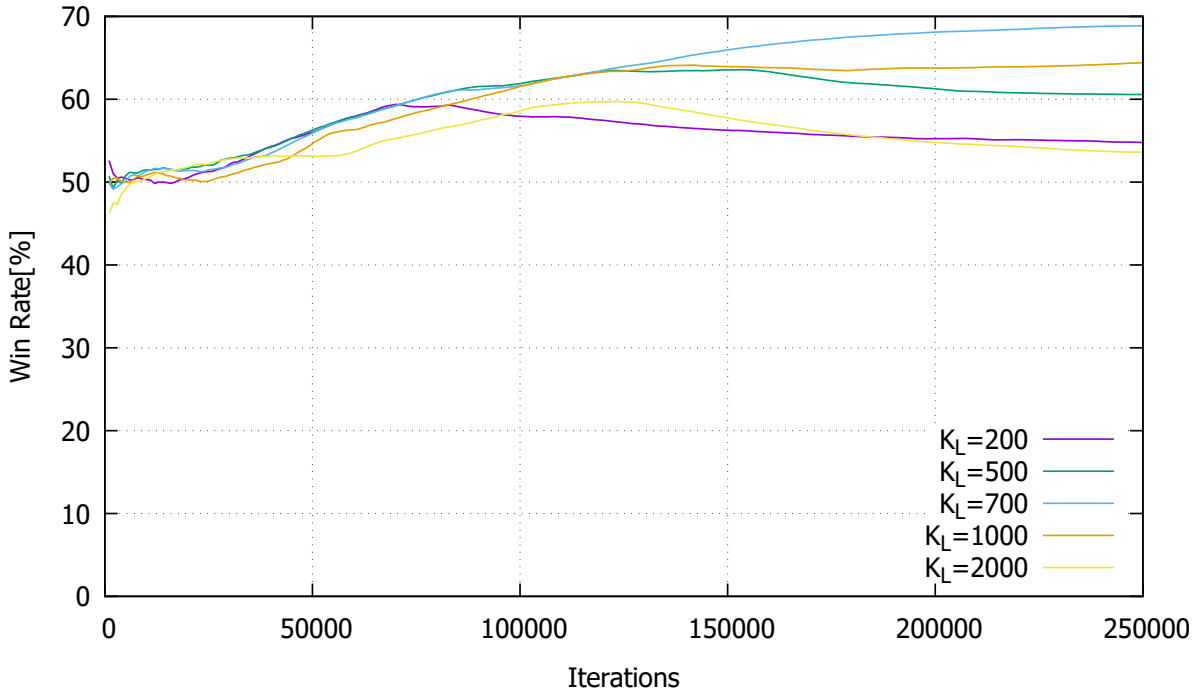


Figure 3.4　Learning curve for each Lookahead interval ($K_L$).

Table 3.3　Evaluate win rate for each Lookahead interval ($K_L$).

| $K_L$ | $K_R$ | Win Rate |
|---|---|---|
| 200 | 50 | 53.81 % |
| 500 | 125 | 57.01 % |
| 700 | 175 | 69.48 % |
| 1,000 | 250 | 70.07 % |
| 2,000 | 500 | 56.58 % |

学習中の勝率を Figure 3.4 に、学習後の評価勝率を Table 3.3 に示す。

### 3.4.1　結果・考察

学習中の勝率を見ると、$K_L = 750$、1000 では安定した学習が行われている。一方 $K_L = 200$、500 や $K_L = 2,000$ という $K_L$ が小さい場合や大きい場合には、学習途中から勝率が低下しており、学習が不安定になっていることがわかる。また、$K_L = 200$、2,000 では他の場合と比較して頭打ちになる勝率が低く、また $K_L = 2000$ は学習が今回のパラメータの中で最も遅い。評価時の勝率を見ても、安定した学習が行われている

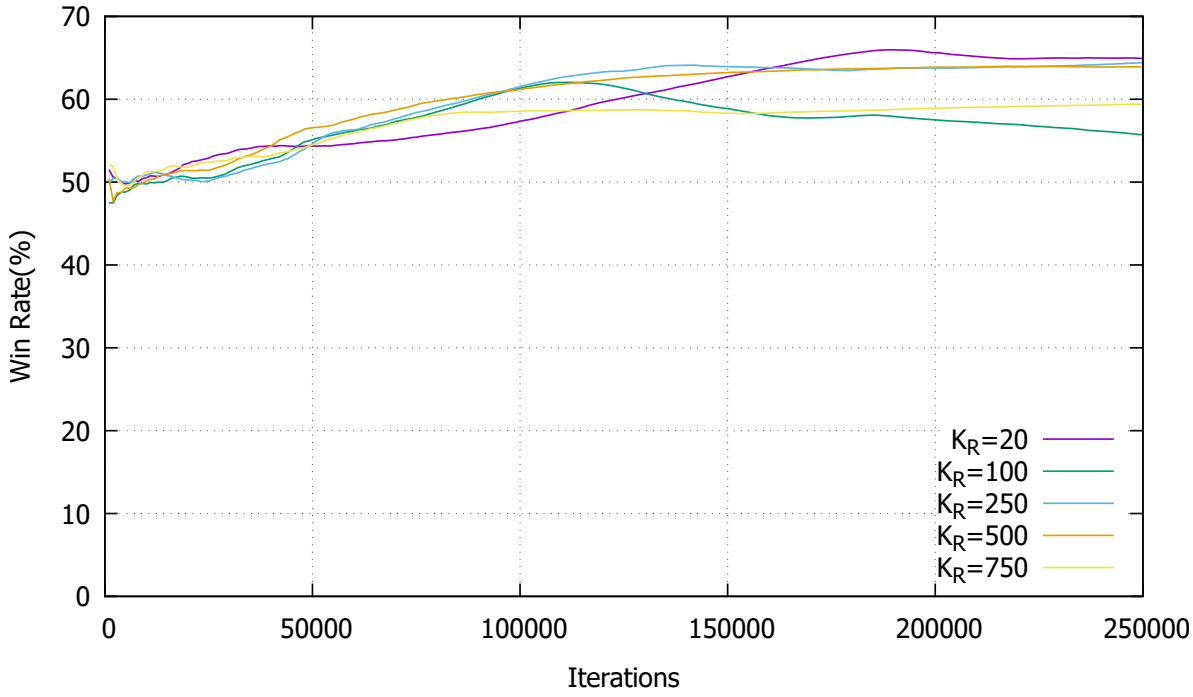


Figure 3.5　Learning curve for each Replicate updates ($K_R$).

$K_L = 750$、$1,000$ はどちらも 70%程度の勝率を上げているのに対し、学習が不安定になった $K_L = 200$、500、$2,000$ ではいずれも 50%台に止まっている。

$K_L = 250$、500 といった $K_L$ が小さい場合には、ターゲットネットワークが頻繁に更新されるため Q 値が短期間のうちに変動し、学習が不安定になったと考えられる。$K_L = 2,000$ では、ターゲットネットワークの更新頻度が遅く、新しい方策を反映するまでに遅延が生じたため、学習が遅くなったと考えられる。また、ターゲットネットワークが古い方策を基に Q 値を計算することにより新しい方策に対して最適な Q 値を計算することができず、学習が不安定になったと考えられる。Penguin Party の環境では、$K_L = 750$ から $1,000$ が安定性や追従性の面から最適な更新間隔であることがわかる。

## 3.5　$K_R$ の決定

次に、$K_L$ を最も性能の良かった 1000 に固定し、$K_R$ を 20、100、250、500、750 に変化させる。学習中の勝率を Figure 3.5 に、学習後の評価勝率を Table 3.4 に示す。

### 3.5.1　結果・考察

学習中の勝率を見ると、$K_R = 250$、500、750 は安定した学習が行われており、$K_R = 100$ は勝率が低下しており学習が不安定であることがわかる。$K_R = 20$ は学習曲線がいびつ

Table 3.4　Evaluate win rate for each Replicate updates ($K_R$).

| $K_R$ | Win Rate |
|---|---|
| 20 | 53.09 % |
| 100 | 50.97 % |
| 250 | 70.07 % |
| 500 | 63.74 % |
| 750 | 62.12 % |

ながら勝率は上昇しているが、終盤になって低下しており、不安定気味な学習であることがわかる。評価時の勝率を見ると、$K_R = 250$ の勝率が 70%と最も高く、安定して学習を行っていた $K_R = 500$、750 が次いで 62 から 63%程度の勝率となっている。学習が不安定になった $K_R = 20$、100 は勝率が 50%台と低い状態である。

$K_R = 20$、100 といった $K_R$ が小さい場合、Replicate ステップにおける学習回数が不足することによりパラメータを十分に近づけることができないため $v_\theta = v_w$ を十分に達成することができず、出力される Q 値が不正確なものになったと考えられる。$K_R = 20$ の学習中の勝率に上昇傾向が見られた理由としては、得られた不正確な Q 値を使用して行動する中で局所的に適した行動を取ることができ、累計の勝利数が増えたものと思われ、見かけ上の勝率は高いが学習は適切でない状態になったと考えられる。

$K_R = 500$、750 といった $K_R$ が大きい場合、学習回数を増やしたことにより $v_\theta = v_w$ を完全に満たすように学習が進んだと考えられる。結果として全く同じ出力を目指す傾向となり、$\theta \neq w$ でも行動可能という Lookahead-Replicate の利点を潰すような方向に進んだと考えられる。

これらの結果から、$K_R$ には適切な中間の値があり、そこから外れると性能が低下することがわかる。

## 3.6　バッチサイズ

続いて、$K_L, K_R$ を最も性能の良かった 1,000、250 に固定し、バッチサイズを 32、64、128、256 に変化させる。学習中の勝率を Figure 3.6 に、学習後の評価勝率を Table 3.5 に示す。

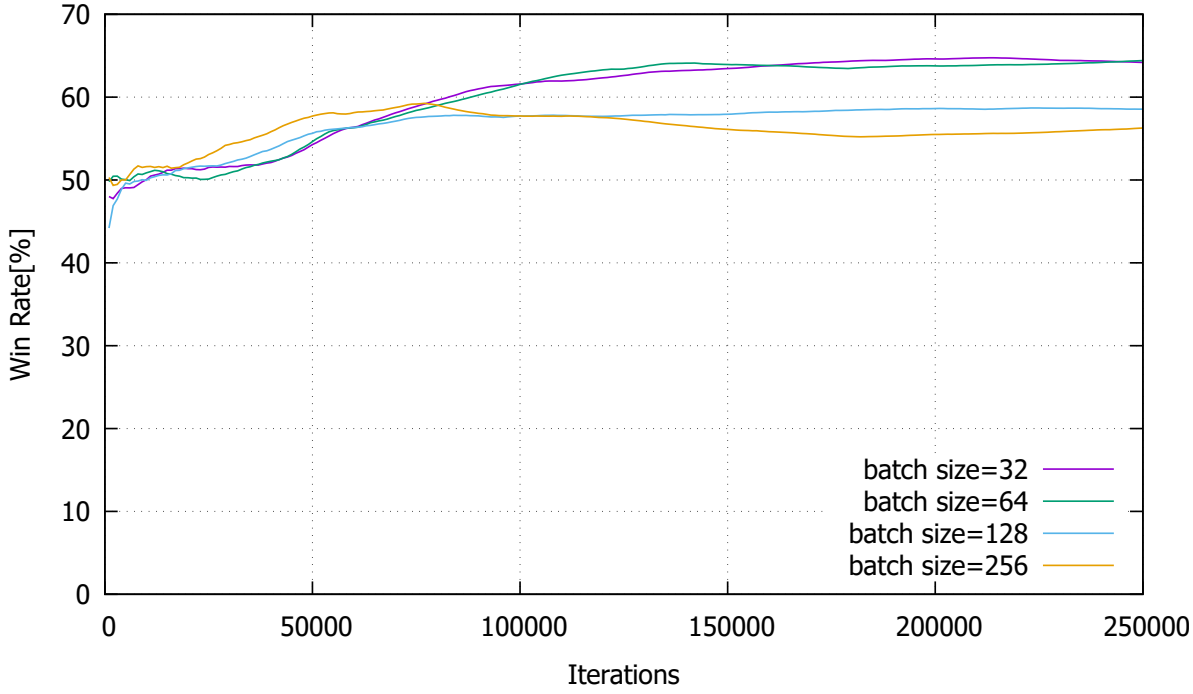


Figure 3.6　Learning curve for each batch size.

Table 3.5　Evaluate win rate for each batch size.

| batch size | Win Rate |
|---|---|
| 32 | 60.47 % |
| 64 | 70.07 % |
| 128 | 58.41 % |
| 256 | 63.78 % |

### 3.6.1　結果・考察

学習中の勝率を見ると、バッチサイズ 32、64 では安定した学習が行われている。バッチサイズ 128 でも比較的安定しているが、頭打ちになる勝率は 32、64 の場合と比べて低い。バッチサイズ 256 では勝率が低下している様子が見られ、学習が不安定であることがわかる。評価時の勝率では、バッチサイズ 64 が最も高く、他は 60%前後に止まっている。

バッチサイズ 32 では得られるサンプルの数が少なく、学習を十分に進めることができず勝率が低下したと考えられる。バッチサイズ 128 や 256 といった大きなバッチサイズの場合、多くのサンプルを集めたことによって過学習や局所解に陥ったことが考えられる。

$K_R$ のパラメータ調整の結果とバッチサイズ調整の結果を踏まえると、Lookahead-Replicate において $K_R$ やバッチサイズを大きくして $v_\theta = v_w$ をできるだけ満たすよう

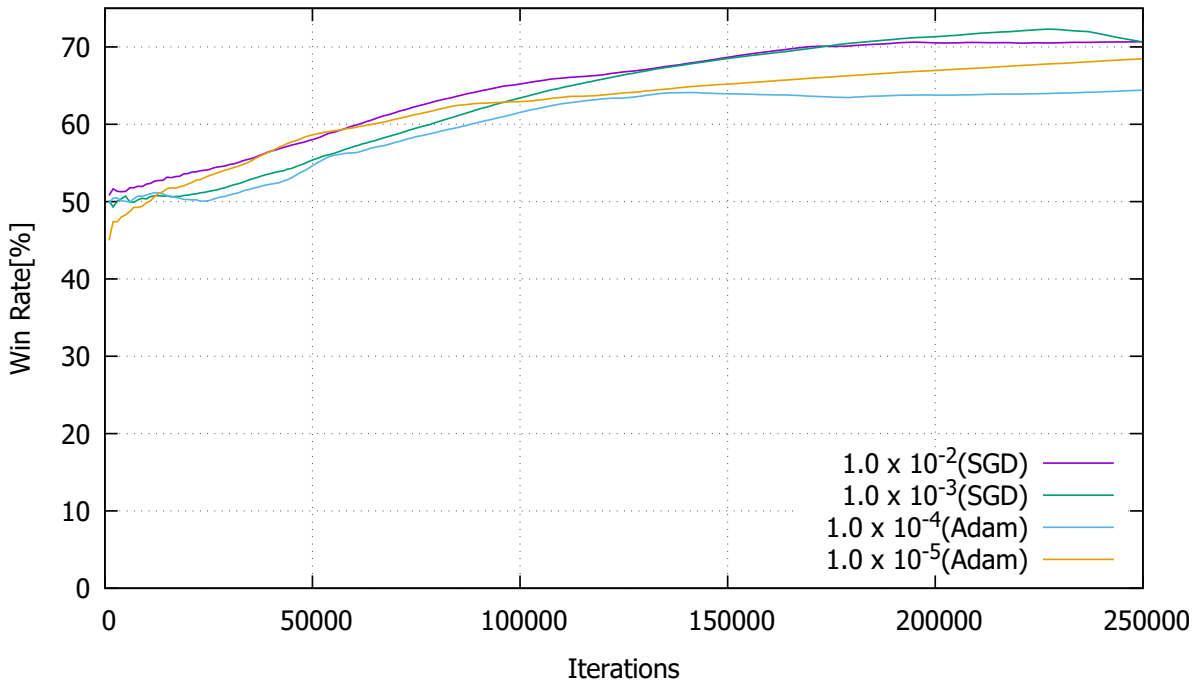


Figure 3.7　Learning curve for each optimizer and learning rate.

に学習を進めることは逆効果であることが考えられる。バッチサイズ 64 では勾配の精度が低く、ノイズが多く含まれている。また $K_R = 250$ ではターゲットネットワークを完全にオンラインネットワークに近づけることはできない。このようなノイズが正規化効果をもたらし、より広い領域で解を探すことができるのではないかと予想される。

## 3.7　学習率及びオプティマイザ

Lookahead-Replicate ではターゲットネットワークも勾配降下法によって更新される。そのため、オプティマイザの種類と学習率が性能に寄与する可能性がある。本実験では、SGD（$1.0 \times 10^{-2}$、$1.0 \times 10^{-3}$）と Adam（$1.0 \times 10^{-4}$、$1.0 \times 10^{-5}$）を比較する。SGD の慣性 (Momentum) は 0.9 とした。学習中の勝率を Figure 3.7 に、学習後の評価勝率を Table 3.6 に示す。

### 3.7.1　結果・考察

学習中の勝率を見ると、いずれの場合でも安定的に学習が行われている。評価時の勝率を見ると、SGD（$1.0 \times 10^{-2}$）の勝率が 76.67% と最も高く、続いて Adam（$1.0 \times 10^{-5}$）が 74.46% と、これまでの実験の勝率より高い勝率が得られた。一方、SGD（$1.0 \times 10^{-3}$）は学習中は勝率が伸びたものの評価時の勝率は 46.55% と、実験を通して最も低い勝率と

Table 3.6　Evaluate win rate for each optimizer and learning rate.

| Learning rate(Optimization Methods) | Win Rate |
|---|---|
| $1.0 \times 10^{-2}$(SGD) | 76.67 % |
| $1.0 \times 10^{-3}$(SGD) | 46.55 % |
| $1.0 \times 10^{-4}$(Adam) | 70.07 % |
| $1.0 \times 10^{-5}$(Adam) | 74.46 % |

Table 3.7　Evaluate win rate for each update algorithms.

| Update Algorithm | Win Rate |
|---|---|
| Fixed Q-Target | 62.39 % |
| Soft Update | 69.40 % |
| Lookahead-Replicate | 76.67 % |

なった。Adam が $1.0 \times 10^{-4}$、$1.0 \times 10^{-5}$ の双方で安定的に学習を行い、70%以上という高い勝率を得ている一方で、SGD は $1.0 \times 10^{-2}$ と $1.0 \times 10^{-3}$ で勝率に大きな差が出た。Lookahead-Replicate においては、SGD は学習率を適切に調整することで高い勝率を得ることができる一方、Adam にはある学習率調整機能がないため学習率を誤れば全く学習しないことがわかる。学習を行う際はまず Adam から始めることが適している。

## 3.8　既存手法との比較

以上の結果から最も成績の良かったパラメータで学習した Lookahead-Replicate のモデルと、既存手法である Fixed Q-Target と Soft Update を用いて学習したモデルの勝率を比較する。Fixed Q-Target のターゲットネットワーク更新頻度は 750、Soft Update の減衰係数は 0.001 とした。学習中の勝率を Figure 3.8 に、学習後の評価勝率を Table 3.7 に示す。

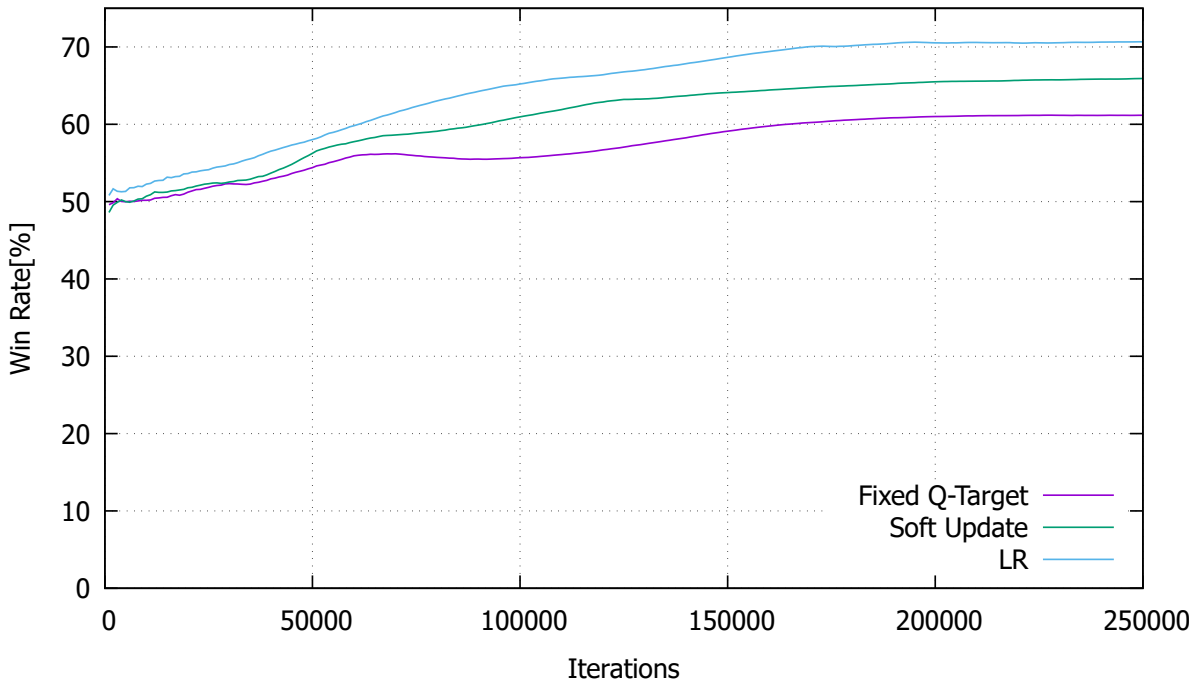


Figure 3.8　Learning curve for each update algorithms.

### 3.8.1　結果・考察

Lookahead-Replicate は、学習中の勝率と評価時の勝率の双方で既存手法を上回った。評価時の勝率では Fixed Q-Target より 14%、Soft Update より 7%高い勝率を得ることができた。

Lookahead-Replicate が価値関数空間での一致（$v_\theta = v_w$）を制約とすることにより、より広い解の空間を持つことから、従来手法より優れた性能を発揮したと考えられる。また、Soft Update が Fixed Q-Target を上回ったことから、緩やかにターゲットネットワークを変更した方が性能が高くなることが分かる。

## 3.9　ネットワーク構造の比較

従来手法ではオンラインネットワークとターゲットネットワークのパラメータが一致する必要があったため、双方で全く同一のネットワーク構造を用いる必要があった。しかし Lookahead-Replicate ではその必要が無いため、両ネットワークが異なるネットワーク構造を持っていても学習が可能である。

新たに Figure 3.9 の構造を持つニューラルネットワークを作成する。このネットワークでは 273 次元の入力を自身の手札・盤面・相手の手札に分割し、それぞれ全結合層で処理し、その後結合する。パラメータ数と層の深さはこれまでの実験で使用したネット

Table 3.8　Experimental setup with different network structures.

| Case | Online Network | Target Network |
|---|---|---|
| Case0 | small | small |
| Case1 | small | large |
| Case2 | large | small |

Table 3.9　Evaluate win rate for different neural network structures.

| Case | Win Rate |
|---|---|
| Case0 | 76.67 % |
| Case1 | 73.54 % |
| Case2 | 58.97 % |

ワークよりも大きい。ここでは、これまでの実験で使用したネットワークを小規模ネットワーク（small)、新たなネットワークを大規模ネットワーク（large）と呼ぶ。これらのネットワークの組み合わせを Table 3.8 に示す。

Case0 はこれまでの実験と同様の構造である。Case1 はターゲットネットワークのみ大規模ネットワークに変更し、オンラインネットワークは小規模ネットワークのままとした設定である。Case2 はオンラインネットワークを大規模ネットワークに変更し、ターゲットネットワークを小規模ネットワークのままとした設定である。学習中の勝率を Figure 3.10 に、学習後の評価勝率を Table 3.9 に示す。

### 3.9.1　結果・考察

Case2 ではターゲットネットワークに小規模ネットワークを適用し、オンラインネットワークに大規模ネットワークを適用した。最終的な勝率は悪いが、学習中の勝率を見ると上昇している様子が見られることから、学習自体は正常に実行されていたと考えられる。性能が低下した原因としては、Replicate 時に大規模なオンラインネットワークの出力を小規模なターゲットネットワークが模倣し切ることができず、ターゲットネットワークの規模がボトルネックになったことが考えられる。

Case1 ではターゲットネットワークに大規模ネットワークを適用し、オンラインネッ

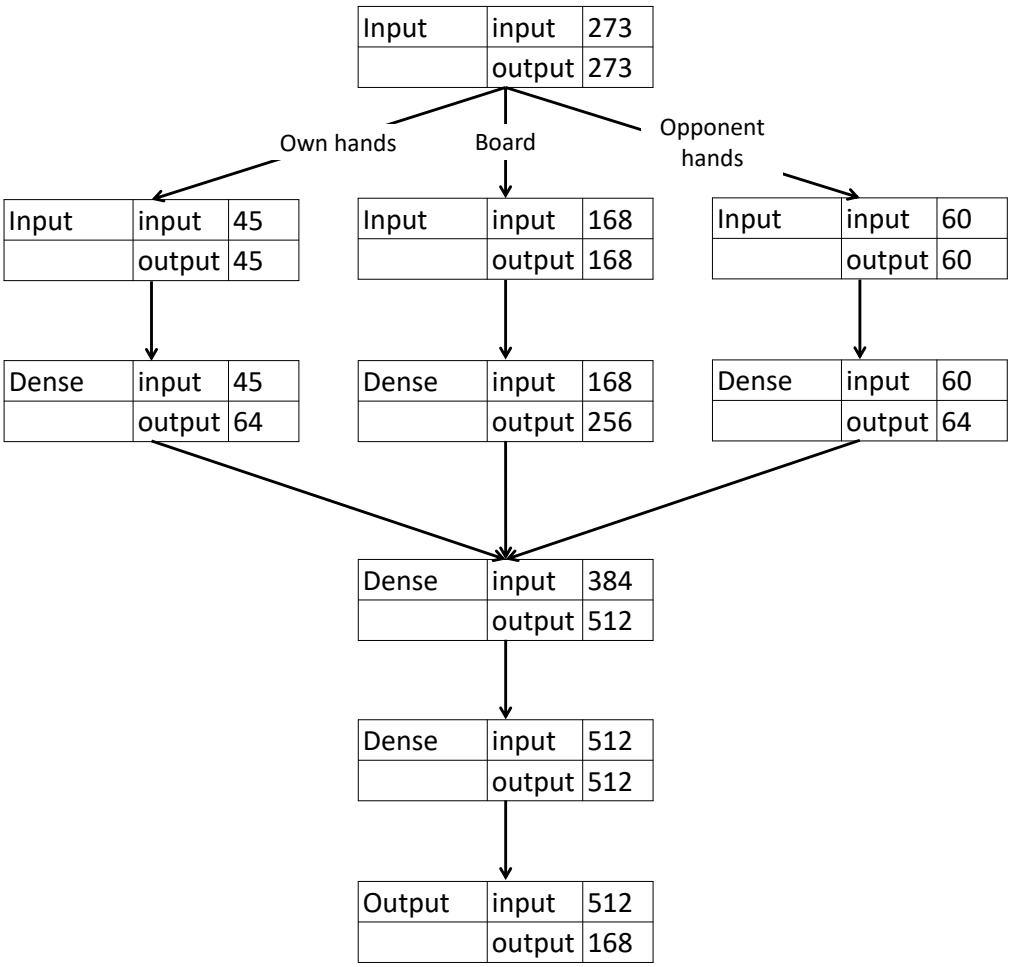


Figure 3.9　Learning network structure.

トワークは小規模ネットワークのままとした。学習中の勝率はCase0とそれほど差はなかったが、評価時の勝率ではCase0よりも3%低い勝率となっている。Case2と異なり学習が不安定にならなかった原因として、ターゲットネットワークを大規模にしたことにより、Replicate時にボトルネックとならなかったことが考えられる。Case0と比較して勝率が3%低下した原因としては、オンラインネットワークとターゲットネットワークの規模の差によりReplicate時に誤差が生じたことにより性能が上がりきらなかったことが考えられる。

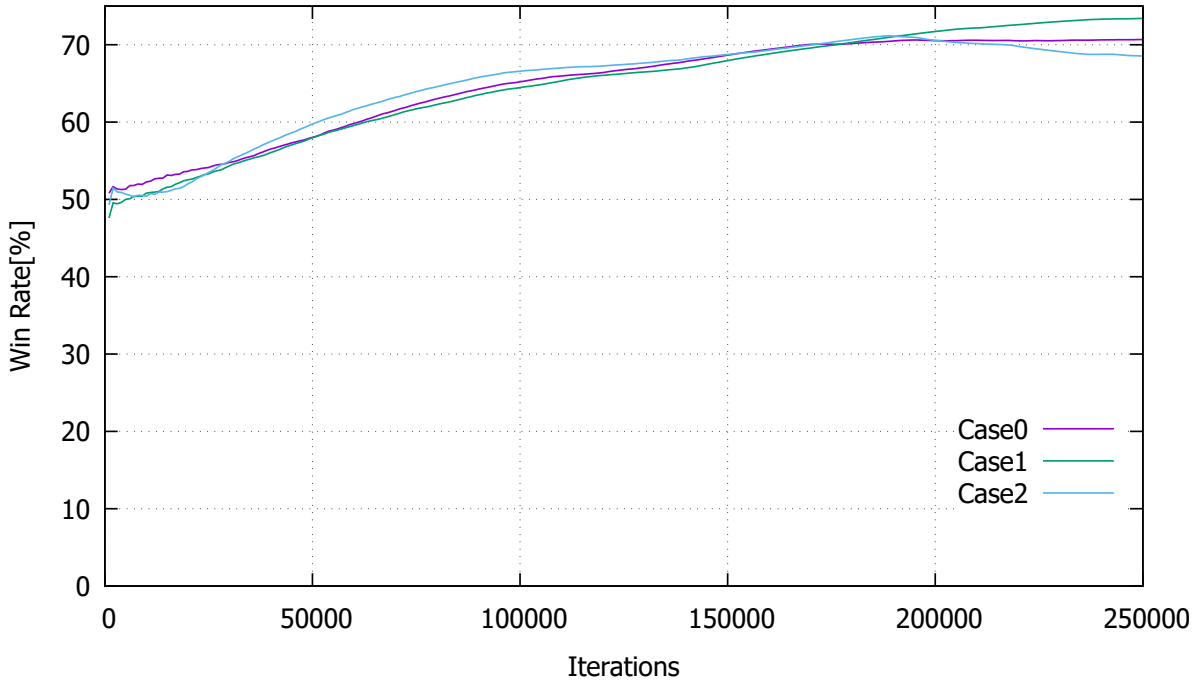


Figure 3.10 Learning curve for different neural network structures. In Case 0, a small network structure is applied to both the online and target networks. Case 1 uses a small network structure for the online network and a large one for the target network. Case 2 uses a large network structure for the online network and a small one for the target network.

# 第4章　結論

本研究の目的はLookahead-Replicateアルゴリズムを Deep Q-Network に実装し、既存手法との比較を行うことであった。

実験では自身の手札・盤面・相手の手札の情報を One-Hot ベクトルとして渡し、行動の Q 値を返す Deep Q-Network を作成した。Lookahead-Replicate アルゴリズムをこれに対して実装し、$K_L, K_R$、バッチサイズ、学習率、オプティマイザを変更することによって Penguin Party に最適なパラメータの探索を実施した。

結果として既存手法に対して 7%以上の勝率の上昇が見られ、また追加で行った、オンラインネットワークとターゲットネットワークで異なる構造を用いる実験では、同じ構造を用いる場合より高い勝率を得ることはできなかったものの、学習を行うことができると判明した。Lookahead-Replicate はパラメータを適切に調整する必要があるが、既存手法よりも高い性能を得ることができ、またネットワーク構造についても、オンラインネットワークとターゲットネットワークを異なる構造にしても学習を行うことができると示された。

今後の展望としては、今回の実験では性能の高かったパラメータのみをピックアップしながら実験を行ったが、性能の低かったパラメータについても実験を行うことでパラメータに関する知見を深めることが考えられる。また、オンラインネットワークとターゲットネットワークで異なるネットワーク構造を用いる場合のパラメータ調整について実験を行うことで、異なる構造を用いることによる効果を調査できると考えられる。

# 参考文献


1) Kavosh Asadi, Yao Liu, Shoham Sabach, Ming Yin, Rasool Fakoor, ”Learning the Target Network in Function Space”, arXiv, https://arxiv.org/html/2406.01838v2, (参照:2026-1-7)

2) 総務省, ”AI に関する基本的な仕組み”, 総務省, https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r01/html/nd113210.html, (参照:2025-10-14).

3) 株式会社 NTT データ グローバルソリューションズ, ”機械学習とは?”, 株式会社 NTT データ グローバルソリューションズ, https://www.nttdatagsl.co.jp/related/column/what-is-machine-learning.html, (参照 2025-10-14).

4) 玉川学園, ”ニューロンの構造”, 玉川学園, https://www.tamagawa.ac.jp/teachers/aihara/kouzou.html, (参照:2025-12-25).

5) The University Of Queensland, ”What is a neuron?”, The University Of Queensland, https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron, (参照:2025-12-25).

6) Mathworks, ”ニューラルネットワークとは”, Mathworks, https://jp.mathworks.com/discovery/neural-network.html, (参照:2026-1-15)

7) アルゴ式, ”活性化関数とは”, アルゴ式, https://algo-method.com/descriptions/5NMPhdlr3EB1PUma, (参照:2025-12-27).

8) Google for Developers, ”ニューラル ネットワーク: 活性化関数”, Google, https://developers.google.com/machine-learning/crash-course/neural-networks/activation-functions?hl=ja, (参照:2025-12-27).

9) Mathworks, ”強化学習とは”, Mathworks, https://jp.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html, (参照:2026-1-15)

10) CHRISTOPHER J.C.H. WATKINS, ”Technical Note Q-Learning”, UCL, https://www.gatsby.ucl.ac.uk/~dayan/papers/cjch.pdf, (参照:2025-12-24)

11) 制御工学の基礎あれこれ, ”価値関数　ベルマン方程式”, 制御工学の基礎あれこれ, https://taketake2.com/N83.html, (参照:2025-12-26).

12) IT 用語辞典 e-words, ” ε -greedy 方策とは”, IT 用語辞典 e-words, https://e-words.jp/w/%CE%B5-greedy%E6%96%B9%E7%AD%96.html, (参照:2026-1-4).

13) Jianqing Fan, Zhaoran Wang, Yuchen Xie, Zhuoran Yang, "A Theoretical Analysis of Deep Q-Learning", (参照:2026-1-2).

14) Hado van Hasselt, Arthur Guez, David Silver, "Deep Reinforcement Learning with Double Q-learning", (参照:2026-1-2).

15) Long-Ji Lin, "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching", (参照:2025-12-30).

16) cpprb, "Understanding of Experience Replay", https://ymd_h.gitlab.io/cpprb/understanding/, (参照:2025-12-30).

17) IBM, "損失関数とは", IBM, https://www.ibm.com/jp-ja/think/topics/loss-function, (参照:2026-1-3).

18) PyTorch, "Huberloss", PyTorch, https://docs.pytorch.org/docs/stable/generated/torch.nn.HuberLoss.html, (参照:2026-1-3).

19) codexa, "ダミー変数（One-Hot エンコーディング）とは?", codexa, https://www.codexa.net/getdummies/, (参照:2026-1-4).

20) IT 用語辞典 e-words, "Adam とは", IT 用語辞典 e-words, https://e-words.jp/w/Adam.html, (参照:2026-1-6).

21) IT 用語辞典 e-words, "確率的勾配降下法 (SGD) とは", IT 用語辞典 e-words, https://e-words.jp/w/%E7%A2%BA%E7%8E%87%E7%9A%84%E5%8B%BE%E9%85%8D%E9%99%8D%E4%B8%8B%E6%B3%95.html, (参照:2026-1-6).

22) milvus.io, "What are target networks in DQN?", milvus.io, https://milvus.io/ai-quick-reference/what-are-target-networks-in-dqn, (参照:2026-1-7).

23) 中部大学 機械知覚&ロボティクスグループ, "深層強化学習と活用するためのコツ", 電子情報通信学会, https://www.ieice.org/~prmu/jpn/ieice/2018/dt_01_004s.pdf, (参照:2026-1-7).

24) Kaustab Pal, "Experiments with DQN", github, https://kaustabpal.github.io/dqn#experiment-4-soft-update-target-network3, (参照:2026-1-7)

25) NVIDIA, "What Is TensorFlow?", NVIDIA, https://www.nvidia.com/en-us/glossary/tensorflow/, (参照:2026-1-3).

26) tensorflow, "TF-Agents: A reliable, scalable and easy to use TensorFlow library for Contextual Bandits and Reinforcement Learning.", github, https://github.com/

tensorflow/agents, (参照:2026:1-3).

## 謝辞

最後に、本研究を進めるにあたり、ご多忙中にも関わらず多大なご指導をしていただきました出口利憲先生、また共に勉学に励んだ同研究室のメンバーに厚く御礼申し上げます。