学習および進化アルゴリズムの実装に関する実験

4 年情報工学科 17 番

澤田 直樹

提出日： 2023 年 2 月 17 日（土）

アブストラクト

本実験では,python を使用して Breakout(Atari) 環境に対して pytorch による強化学習 (DQN) を行った.

まず第 1 章では本章の目的について, 強化学習を行いその学習モデルを評価, 比較することであると定めた.

第 2 章では本実験で必要となった理論について,2.1 節でニューラルネットについて, パーセプトロン, 順伝搬, 逆伝搬について述べ,2.2 節で活性化間数について sigmoid 関数,ReLu 関数,softmax 関数について述べ,2.3 節では畳み込みニューラルネットについて, 畳み込み層, プーリング層について述べ,2.4 節では DQN について述べ,2.5 節では Experience Replay について述べ,2.6 節では Dueling Network について述べ,2.7 節では損失関数について SmoothL1 Loss について述べ,2.8 節では最適化関数について,SGD,Momentum,AdaGrad,RMSProp,Adam について述べた.

第 3 章では, 本実験を行った方法について使用したニューラルネットの構造, 実行, 学習のアルゴリズムについて述べた.

第 4 章では, 本実験を行った結果について, 得られた結果を表, グラフ, 箱ひげ図としてまとめ, ε-greedy 方策で学習したモデルでは, 学習エピソード数と共に合計獲得報酬が増加していることがわかり,boltzman 方策で学習したモデルでは,200000 エピソードをピークに以降合計獲得報酬が減少していることがわかった.

第 5 章では,boltzman 方策と ε-greedy 方策での学習モデルの結果を比較し, 各パラメータのグラフから, その探索と利用のバランスについて考察した. また, 第 4 章で示した箱ひげ図から, 分散が大きくなっている理由について考察した.

第 6 章では, 本実験のまとめや感想などを述べた.

第 7 章では本報告書の参考文献について記した.

第 1 章　実験目的

本章では, 本実験の目的について述べる.

基本課題で使用した Q 学習について, それをニューラルネットによる学習に利用した DQN について学ぶために本実験を行った.

本実験の主目的として python を使用して Gymnasium ライブラリ内の Breakout(Atari) 環境に対して報酬の向上を目指して,pytorch による強化学習 (DQN) を行い, その学習モデルにゲームをプレイさせ, 学習エピソードごとに評価すること, 行動選択方策において, ε -greedy 方策と boltzman 方策を用い, その探索と利用のバランスを検討することを設定した.

他の目的を以下に箇条書きにする.

・ pytorch を用いた強化学習モデル設計を習得する.
・ ニューラルネットについて理解する.
・ CNN,Dueling Network について理解する.
・ CUDA を使用するためのプログラミング, メモリ管理を習得する.

第 2 章　理論

本章では本実験で使用した理論について述べる.

2.1　ニューラルネット

ニューラルネットとは, 脳の神経細胞が持つ回路網を模した数理モデルである. 脳内神経のネットワークで行われている情報処理の仕組みを計算式に落とし込み, パーセプトロンを使って数学的にモデル化したものである.[1]

厳密にはパーセプトロンの構造に, 活性化関数として sigmoid 関数や ReLu 関数を使用する.

また, 数学的な観点からは, 各構成原子の活性化関数における入力での線型結合と, 層を深めるていくときの活性化関数の出力を再度入力とする非線形変換を繰り返す合成関数とも言える. この非線形変換により, 次数を上げていけば, 必ずどの集合対においても線形分離が可能となる.

2.1.1　パーセプトロン

パーセプトロンは複数の信号を受け取ったときに, 一つの信号を出力するアルゴリズムのことである. 図 1 に例を示す.

ここで,x_1, x_2 は入力信号,y は出力信号,w_1, w_2 はそれぞれの信号に固有のパラメータであり, 重みとされる. 入力された信号に重みを乗算した総和が計算され, その総和がある閾値 b を超えたときにのみ 1 が出力され, 閾値を超えなかった場合には 0 が出力される. なお,1 を出力することを発火と呼ぶ.[1]

ニューラルネットではこのパーセプトロンの構造を基本原理として使用するが, ある閾値を堺に 0 か 1 だけを出力するのではなく, 活性化関数によって出力を決定する. なお, ある閾値を堺に 0 か 1 だけを出力する関数をステップ関数と呼び, パーセプトロンは活性化関数としてステップ関数を使用していると言える.

ステップ関数がニューラルネットにおける活性化関数として使用されないのは微分

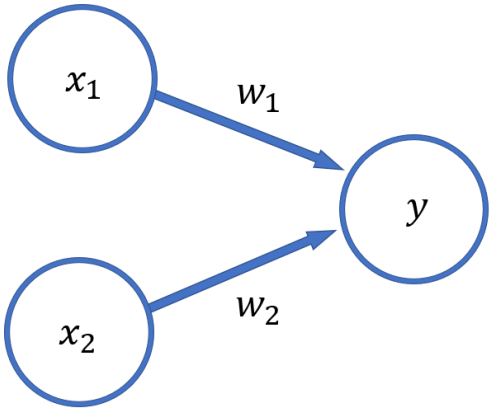

図 1　パーセプトロンの例

値が常に 0 であるため, 誤差逆伝播法に適さないためである.

2.1.2　順伝搬

ニューラルネットワークにおける順伝播とは, 前の層の出力を本層の入力として処理することを指す. 図 1 の例の構造をニューラルネットとみなすと出力 y は式 (1) のようになる.

$$y = f(\sum_{i=1}^{N} x_i w_i + bias) \tag{1}$$

このとき, 前層の出力は 2 であるため N=2 であり,f は活性化関数である.

2.1.3　逆伝搬

ニューラルネットワークにおいては出力値を理想の値に近づけるためにパラメータを調節する. このときに順伝搬と全く逆方向に行われる処理のことを逆伝搬という.

出力層では順伝搬による最終的な値を出力するが, その値と理想値との誤差を算出する. その算出した誤差が最小値を取るようにパラメータを調整していく.

2.2　活性化関数

活性化関数とは, ニューラルネットにおいて各層の出力の前に適用することで, 値を最適化することができる関数である.[2]

代表的な例としてシグモイド関数, ランプ関数などが使用される. これらの関数は非線形であり, ニューラルネットにおける非線形変換の役割を果たす.

2.2.1　sigmoid 関数

sigmoid 関数 (2) は, 値を必ず 1 から 0 の間に変換することのできる関数である.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

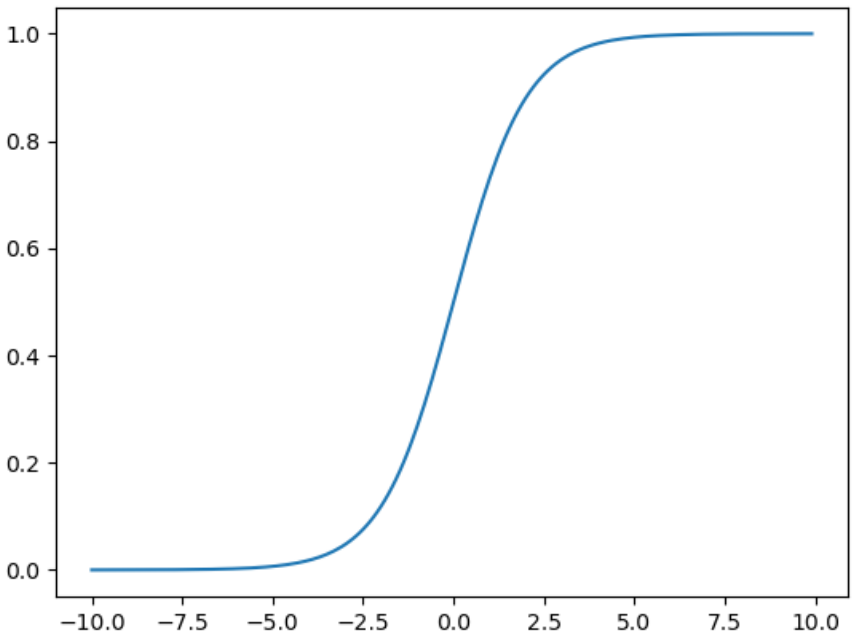

図 2　sigmoid 関数

値が必ず 0〜1 の間になるため, その値を確率としてみなすことができ, この特性を生かして主に二値分類の最終層に使われる.[2]

2.2.2　ReLu 関数

ReLu(レクティファイドリニアユニット) 関数 (3) は値が負の場合は 0 を出力し, 正の場合はそのまま出力する関数である. ランプ関数とも呼ばれる.

$$f(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \tag{3}$$

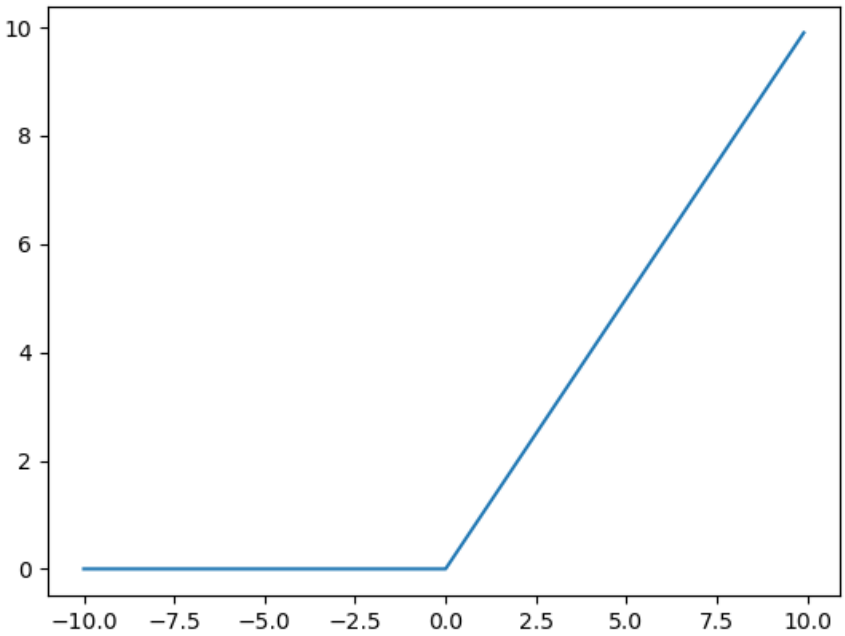

図 3　ReLu 関数

負の値があると不都合となる, 画像に関わる問題などに使用されることが多い.[2]

2.2.3　softmax 関数

softmax 関数は, 入力された項目毎に出力される値が変わり、それらすべての値を足し合わせると必ず 1 になるようにな関数である.(4)

$$f(x) = \frac{e^{a_x}}{\sum_{i=1}^{k} e^{a_i}} \tag{4}$$

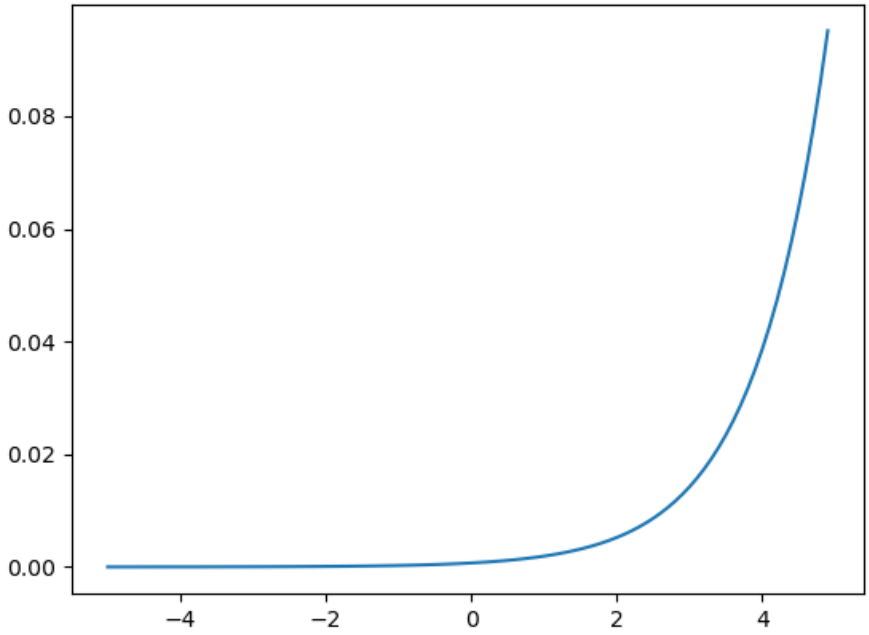

図 4　softmax 関数

この関数は他の関数とは違い, 全入力 ((3) では k 個の入力) によってグラフの形が変化する. 全ての値を足し合わせると 1 になるため, 確率として使用できる.[2]

2.3　畳み込みニューラルネット (CNN)

畳み込みニューラルネット (CNN) は主に画像認識の分野で用いられるディープラーニングアルゴリズムである. データを学習することで入力画像から特徴量を抽出しそれらを区別することができるようになるため、ディープラーニングを画像認識の分野に

応用する上で大変重要なものとなっている. CNN は入力層, 畳み込み層, プーリング層からなっている.

2.3.1 畳み込み層

畳み込み層では, 入力画像とは別にカーネルというフィルタのような重み行列を用意する. カーネルを図 5 のように左斜め上から入力行列 (画像) の全ての位置に移動させ, 加重和を計算することで出力を得ます。このカーネルの移動間隔をストライドと呼ぶ. また, この出力は特徴マップと呼ばれる.

図 5 の例は入力チャネル数を 6, 出力チャネル数を 32, カーネルサイズを 8, ストライドを 4 としている.

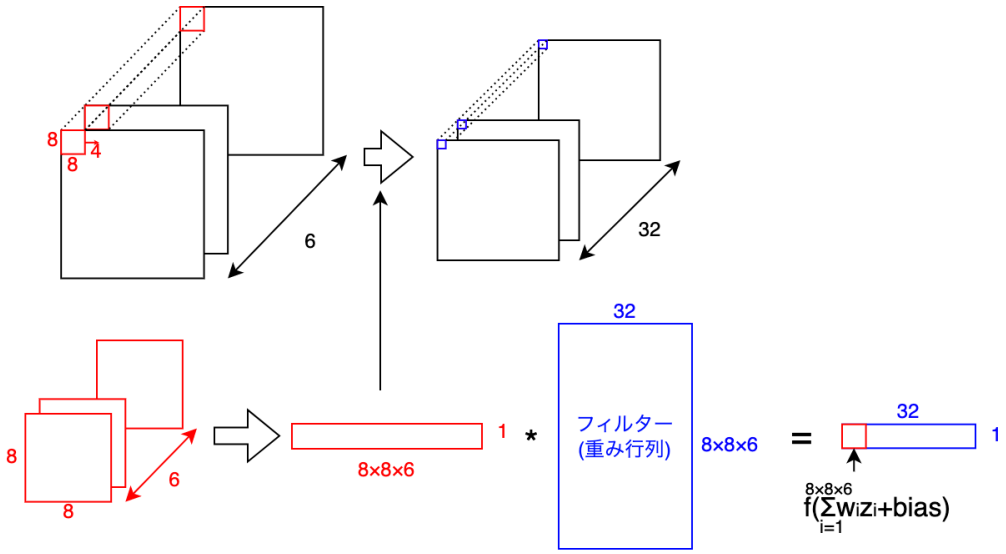

図 5　畳み込み層の例

また, 重み行列はニューラルネットの逆伝搬によって最適化されていく.

2.3.2 プーリング層

プーリング層では画像内のある領域毎 (カーネル毎) に画像のサンプリング操作を行う. 代表的な例としてカーネル内の最大値を取る Max Pooling や平均値をとる Average Pooling がある.

畳み込み層との明確な違いは逆伝搬によって変化するパラメータを持たず, 同じ入力

に対しては常に同じ出力を返す.

2.4 DQN(Deep Q-Network)

Q 学習にニューラルネットワークの考え方を含めたものを DQN という.

Q 学習では,Q テーブルを更新していく仕組みのため, 連続的な状態を表現しようとすると,Q テーブルが膨大な量となるため, 現実的に計算ができなくなるデメリットがある.

それに対し,DQN は Q 値そのものを推定するのにニューラルネットワークを使うことで,Q 値の近似関数を得てしまおうという考え方で連続的な行動にも対応できるようになっている.[3]

2.5 Experience Replay

Experience Replay は, 遷移を一時的に Replay Memory と呼ばれる領域に保存しておき, 後からその遷移をランダムに取り出して 1 つのバッチとして学習に用いる手法である. 時間的に離れたデータを使用することができ, 偏りを抑えた安定した学習ができる.[4]

2.6 Dueling Network

Q 学習では, 状態と行動のペアに対して Q 値を推定するが,Dueling Network はこの Q 値の推定を 2 つの部分に分離して考える.[5]

- Value Function(V)

  これは特定の状態の価値を評価するもので, 行動を考慮せずに環境の状態のみに基づいて評価する.
- Advantage Function(A)

  これは各行動の価値を評価するもので, 特定の状態での各行動が平均よりもどれだけ優れているか, または劣っているかを評価する.

この V(s) と A(s,a)(s: 状態,a: 行動) の出力の和を Q 値として考える. 状態価値関数 V(s) が行動 a に寄らずに学習できることが通常の DQN にない利点である.

2.7 損失関数

損失関数とは目標と実際の出力の誤差を表すものであり, 最適なパラメータに近づくための指標となる.

2.7.1 SmoothL1Loss

SmoothL1 Loss は, ディープラーニングにおいて主に物体検出や回帰問題において用いられる誤差関数の一つ.

式 (5) に示す式で定義され, 二つの値の差の絶対値を用いる通常の L1 Loss に比べ, 値の差が小さい場合には損失をより滑らかに計算することができる.[6]

$$smoothL1(x) = \begin{cases} 0.5x^2 & (|x| < 1) \\ |x| - 0.5 & (|x| \geq 1) \end{cases} \tag{5}$$

2.8 最適化関数

損失関数から得られた正解値と予測値の差を微分し, 得られた勾配を利用してどのくらいの強度でパラメータを更新するかを決めているのが, 最適化関数である.

2.8.1 SGD(確率的勾配降下法)

ランダムに取り出したデータを用いて, 重み W を更新していく勾配降下法である.[7]

$$W_{t+1} = W_t - lr\frac{\partial\ L}{\partial\ W_t} \tag{6}$$

W_{t+1} は更新後の重みで,W_t は学習前の重みであり,L は損失関数で,lr は学習率である.

2.8.2 Momentum

SGD では, 学習の各段階において, 勾配が最も急な方向に重みを更新していくため, 学習の初期には, 最も早く損失関数の谷底へ向かうことができるが, 学習が進むと, 勾

配の小さい地点にもかかわらず, 重みの更新量が大きすぎることにより, 最小値付近で振動が起こり, 最小値になかなかたどり着くことができないという問題が発生してしまう.[7]

Momentum では, ボールを転がしたような挙動をさせることができ, このような振動を抑えることができる.

$$
\begin{aligned}
v_{t+1} &= av_t - lr\frac{\partial\ L}{\partial\ W_t} \\
W_{t+1} &= W_t + v_{t+1}
\end{aligned}
\tag{7}
$$

v というベクトルは, 勾配由来の $-\ lr\ \partial\ L\ \partial\ W_t$ という力を受けて, 時間を経るごとに加速していくので, 物理学での速度に対応する. また,a は 0 以上 1 未満のパラメータであり, 更新前の速度をどれだけ保持するかを決定し, これは物理学での慣性に対応する.

2.8.3 AdaGrad

AdaGrad では,Momentum とは異なり, 学習が進むにつれて, 学習係数を小さくしていくことで, 最小値付近での振動に対応する.[7]

$$
\begin{aligned}
h_{t+1} &= h_t + (\frac{\partial\ L}{\partial\ W_t})^2 \\
W_{t+1} &= W_t + lr\frac{1}{\varepsilon\ +\ \sqrt{h_{t+1}}}\frac{\partial\ L}{\partial\ W_t}
\end{aligned}
\tag{8}
$$

ε は数値安定性のためで一定の値なので, 本質的には $\frac{1}{\sqrt{h_{t+1}}}$ によって学習率 lr を調整しているのが AdaGrad であると言える.

2.8.4 RMSProp

RMSProp は AdaGrad の改良版であり, 移動平均を導入している.[8]

$$
\begin{aligned}
h_{t+1} &= \beta\ h_t + (1 - \beta)(\frac{\partial\ L}{\partial\ W_t})^2 \\
W_{t+1} &= W_t + lr\frac{1}{\varepsilon\ + \sqrt{h_{t+1}}}\frac{\partial\ L}{\partial\ W_t}
\end{aligned}
\tag{9}
$$

β という 0 以上 1 未満のハイパーパラメータを新たに設定することで, 過去の勾配情報をどの程度用いて更新するかを設定することができ, それによって AdaGrad では実現しないジグザグの動きを可能にする.

2.8.5 Adam

Adam は Momentum と RMSProp のアイデアを組み合わせた収束が早く、さまざまな問題でうまく機能し、現在最もよく使われている最適化関数の 1 つである.[9]

$$
\begin{aligned}
v_{t+1} &= \beta_1 v_t - (1 - \beta_1)\frac{\partial\ L}{\partial\ W_t} \\
h_{t+1} &= \beta_2 h_t + (1 - \beta_2)(\frac{\partial\ L}{\partial\ W_t})^2 \\
W_{t+1} &= W_t + lr\frac{v_t}{\varepsilon\ + \sqrt{h_{t+1}}}\frac{\partial\ L}{\partial\ W_t}
\end{aligned}
\tag{10}
$$

第 3 章　方法

今回の実験では,python,Jupyter Notebook を使用して,pytorch による Gymnasium ライブラリ内の Breakout(Atari) 環境に対して強化学習 (DQN) を行うプログラムを作成, 実行し, 保存したモデルを使用して Breakout をプレイさせ, 獲得報酬推移と共に評価した.

また, 行動選択方策において, ε -greedy 方策と boltzman 方策を比較するために, それぞれの方策をとるプログラムを作成し, 同一の環境で実行することで比較を行った.

以下に作成したプログラムでのニューラルネットの構造, アルゴリズムを示す.

3.1　Replay Memory

遷移を状態, アクション, 次状態, 報酬の tuple として宣言し, それを収集する Experience Replay Memory を実装し, 最大サイズ 100000 で宣言した.

3.2　入力画像のリサイズ

ゲーム画像を状態として使用するため, 使用メモリの削減のために, 元画像 (図 6) から, 使用するためにプレイに関係ない部分を削除し,84 × 84 にリサイズ, グレースケール化を行った. また, ブロックの状態に依存しすぎないように, ブロックがあるエリアのみ RandomErasing を実施した.(図 7)

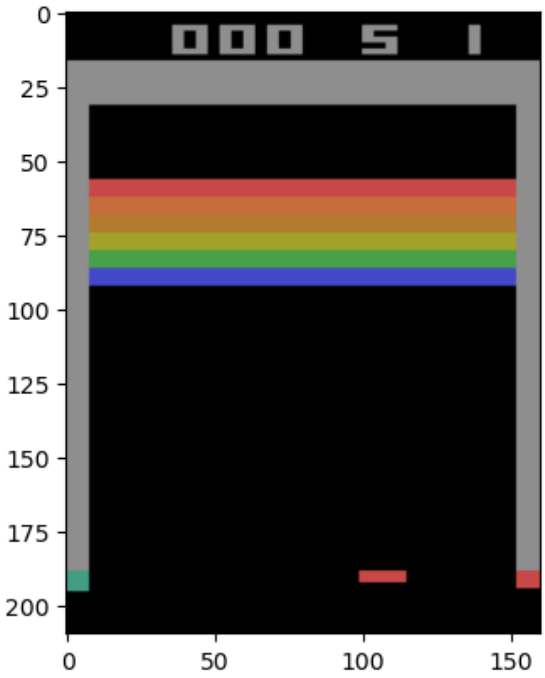

図 6　元画像

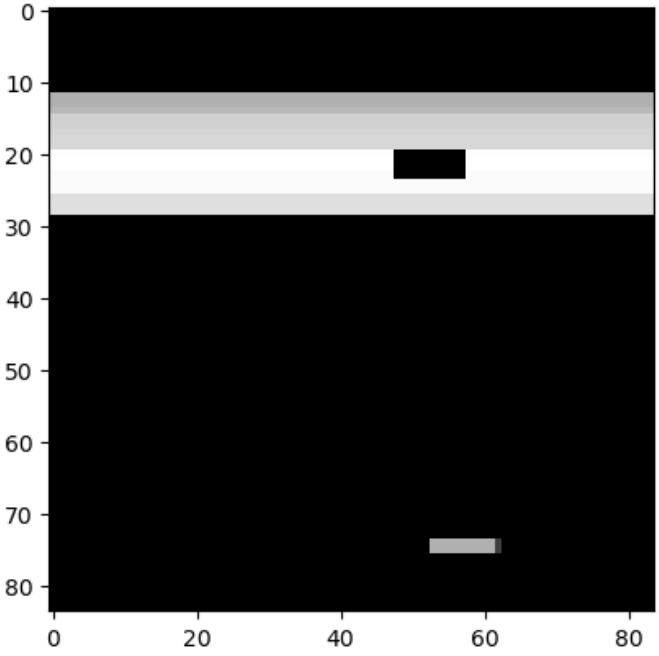

図 7　リサイズ, グレースケール化,RandomErasing 実施後

3.3　ニューラルネットについて

まず, 今回のニューラルネットでは活性化関数として ReLu 関数を使用した. また, 入力画像の畳み込みとして, 入力チャネル数: フレーム数 (今回は 6), 出力チャネル数:32, カーネル:8 × 8, ストライド:4 の第 1 畳み込み層, 入力チャネル数:32, 出力チャネル数:64, カーネル:4 × 4, ストライド:2 の第 2 畳み込み層, 入力チャネル数:64, 出力チャネル数:64, カーネル:3 × 3, ストライド:1 の第 3 畳み込み層, 但し, ここでの入力チャネル数と出力チャネル数の値は画像の縦横とは別次元の軸方向への大きさである.

この CNN の各カーネルとストライドの値より, 今回の入力サイズである 84 × 84 の画像 6 フレーム分を入力した時に図 8 のように 7 × 7 × 64 の出力が返されることがわかる.

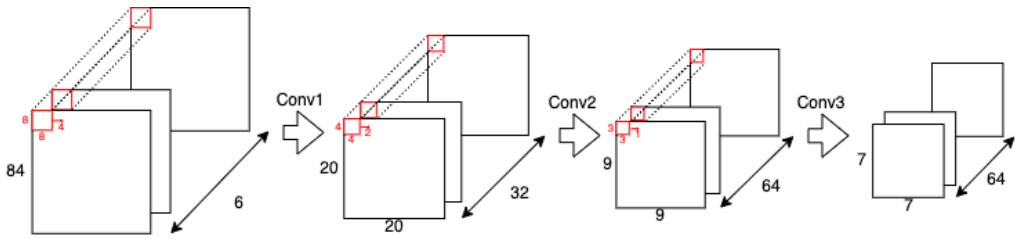

図 8　使用した CNN

この CNN により畳み込みがされた状態を再び入力とした act_fc,value_fc を定義し,Dueling Network を構成している.act_fc は入力チャネル数:3136(7 × 7 × 64), 出力チャネル数:512 の第 1 線型結合, 入力チャネル数:512, 出力チャネル数: 全行動数 (今回は 4) の第 2 線型結合で構成され,value_fc は入力チャネル数:3136(7 × 7 × 64), 出力チャネル数:512 の第 1 線型結合, 入力チャネル数:512, 出力チャネル数:1 の第 2 線型結合で構成している.

ここで act_fc は状態に対する各行動価値であり,value_fc は状態価値である.

また, このニューラルネットでは,CNN の畳み込み後に 1 次元変換を行い, それを act_fc,value_fc の入力とし, 出力の Q 値を式 (11) で出力する順伝搬を定義している.

$$Q(s,a) = value\_fc(s) + act\_fc(s,a) - act\_fc\_ave(s)(\text{各行動価値の平均}) \qquad (11)$$

3.4 実行部

使用した各パラメータの説明やその設定値を表 1 にまとめる.

表 1: 学習を行うためのパラメータ

型	名前	説明
int	num_episodes	学習させるエピソード数. 今回は 500000.
int	n_frame	過去フレームの利用数であり, 遷移はこの単位で管理する. 今回は 6.
int	resize_image	リサイズ後のピクセル数. 今回は 84.
int	num_episodes_save	何エピソードでモデルを保存するか. 今回は 25000.
Dueling_Network	policy_net	行動選択に使用する関数. 最適化モデルにより 4 遷移ごとに更新されていく.
Dueling_Network	target_net	次状態の Q 値を出力する時に使用する関数.400 遷移ごとに policy_net の値を反映し, 更新していく.
float	TAU	重みを近づける割合であり,target_net 更新時の policy_net の反映率. 今回は 0.005
int	total_steps	環境に行動を入力した総回数. 初期値は 0.
int	count_update	6 フレーム単位の遷移を Experience Replay Memory に push した回数.total_steps の 1/6. 初期値は 0.

bool	fire_ball	true のとき, 必ずボールを出すアクションにする. エピソードの最初の行動を固定するためのパラメータ.
tensor	reward_frame	遷移ごとの報酬の合計値. 要素数 1 の float 型
tensor	state_frame	現状態を格納する. フレーム数 (6) ×画像サイズ (84 × 84) の要素数 1 の float 型で 1 つ目の要素に現在のフレームを格納し, 以降順番に過去 5 フレームを格納する. 行動入力後に更新しない.5 エピソード (1 ゲーム) ごとに過去 5 フレームは 0 で初期化される.
tensor	next_state_frame	次状態を格納する.state_frame を同様に 6 フレーム分を格納するが, 行動入力後に最新フレームを更新する.5 エピソード (1 ゲーム) ごとに過去 5 フレームは 0 で初期化される.
bool	frame1	遷移の 6 フレームのうちの最終フレーム (最新フレーム) であるかどうか. 初期値は true
tensor	state_frame1	state_frame の情報が格納され,6 フレームごとに更新される.
tensor	next_state_frame1	next_state_frame の情報が格納され,6 フレームごとに更新される.
tensor	action_frame1	選択した行動の番号が格納され,6 フレームごとに更新される.
int	old_life	現在の残りライフ数. 環境への行動入力時の返り値からの情報で更新する.

今回は Gymnasium の Breakout-v5 の環境に対して実行した. この環境に行動を入力したときの返り値を表 2 に示す.[10]

表 2: ALE/Breakout-v5 の行動入力に対する返り値

型	名前	説明
Box	observation	210 × 160 のピクセルの 0 から 255 までの RGB 値がそれぞれ格納された Box(各ピクセルにアクセスできる dictionary).
int	reward	そのフレームでの獲得報酬. 破壊したブロックの色によって値が異なる.
bool	terminated	ライフが 0 になり, ゲームが終了したかどうか. 初期値は true を代入しておく.
bool	truncated	ライフが 0 になるか, すべてのブロックを破壊すると True が返ってくる. 今回はライフ 1 つでエピソードを区切るためにライフが減ったときにも手動で true に変更し, このパラメータを利用している.
dictionary	info	現合計スコアや残りライフ数などの情報をまとめた dictionary.

次に, 実行のアルゴリズムについて示す. 以下を num_episodes 回繰り返す.

1. fire_ball を true にし,reward_frame を 0 で初期化する.
2. terminated が true のとき,state_frame,next_state_frame の全ての要素を 0 で初期化, 環境を初期化し, 初期状態の observation,info を得て observation をリサイズ (3.2 節) し,state_frame,next_state_frame の 1 つ目の要素に代入する. また,old_life を info['lives'] で初期化する.
3. 以下の 3.1 から 3.9 を terminated もしくは truncated が true になるまで繰り返す.

    3.1. total_steps をインクリメントし, 任意の行動方策で得た行動を環境に与え, 表 2 のパラメータを得る.

3.2. 現在のライフ数 (info['lives']) と old_life を比較し, 現在のライフ数の方が小さい時,old_life を現在のライフ数で更新し,truncated を true にする.

3.3. terminated もしくは truncated が true であるとき,reward を-1 で更新しマイナスの報酬を与える.

3.4. reward を要素数 1 の tensor に変換し最小値-1, 最大値 1 にクリッピングする. また,next_state_frame を 1 要素分 roll(要素をインデックスの増加方向にシフト) し,1 つ目の要素に observation をリサイズして代入する.

3.5. frame1 が true のとき,state_frame1 に state_frame を,next_state_frame1 に next_state_frame を,action_frame1 に選択した行動を代入する. また,frame1 を false に変更する. さらに terminated もしくは truncated が true であるときは next_state_frame1 を None に変更する.

3.6. reward_frame に reward を追加する.

3.7. total_steps が n_frame(6) の倍数のときと,terminated もしくは truncated が true のとき,Experience Replay Memory に遷移として state_frame1,action_frame1,next_state_frame1, reward_frame を push し,reward_frame を 0 で初期化,frame1 を true に変更,count_update をインクリメントする. 最後に,state_frame を next_state_frame で更新する.

3.8. count_update が 4 の倍数のとき, 最適化モデルを実行し,policy_net を更新する. しかし,Experience Replay Memory が最大サイズに達していない時には最適化モデルを実行しない.

3.9. count_update が 400 の倍数のとき,target_net の全てのパラメータを式 (12) で更新する. 但し, θ_{policy_net}, θ_{target_net} は各パラメータを示している.

$$\theta_{target\_net} = \theta_{policy\_net} * TAU + \theta_{target\_net} * (1 - TAU) \tag{12}$$

4. 現在のエピソード数が num_episode_save(25000) の倍数であれば,target_net のパラメータを PyTorch ファイルとして拡張子.pth で保存する.

3.5 最適化モデル

使用した各パラメータの説明やその設定値を表 3 にまとめる.

表 3: 最適化モデルのパラメータ

型	名前	説明
int	BATCH_SIZE	この値分の遷移を Experience Replay Memory からランダムに読み込み学習する. 今回は 256.
namedtuple	batch	BATCH_SIZE 分取り出した遷移を格納する.
tensor	non_final_mask	batch と同じ大きさの bool 型の tensor.batch 内の各遷移に次状態が存在するかどうかを示す.
tensor	non_final_next_state	batch 内の各遷移で次状態が存在するものだけを格納する.
tensor	state_action_values	$Q(s, a)$ が格納された tensor.
tensor	next_state_values	$Q(s', a_{max})$ が格納された tensor.
float	LR	学習率. 今回は 0.0001
float	GAMMA	期待値を算出する上で target_net の値を使用する割合. 今回は 0.99
tensor	expected_state_action_values	期待値が格納された float 型の tensor.

今回は最適化関数に Adam を使用し, 学習率を LR(0.0001) とした. また, 誤差関数に SmoothL1Loss を使用した.

以下に, 実行のアルゴリズムについて示す. 但し,Experience Replay Memory の長さが BATCH_SIZE 以下のときは実行しない.

1. Experience Replay Memory から BATCH_SIZE 分の遷移をランダムに取り出し, そのリスト内の遷移の tuple をそれぞれ要素で分割し, 各要素ごとをまとめたリストにして, それぞれのリストを要素として新たに遷移の namedtuple を作成し,batch に代入する.

2. non_final_mask の各要素に batch の次状態が None でないときは true, そうでないときは false を代入する.
3. batch の次状態全体から,None であるものを除外したものを non_final_next_state に代入する.
4. policy_net に各遷移の状態を入力したときの Q 値の中から, その遷移の行動についての Q 値を state_action_values に代入する.
5. next_state_values の各要素を 0 で初期化し,non_final_mask で true の箇所にだけ target_net に各遷移の次状態を入力したときの Q 値の中から, 最大値を代入する.non_final_mask で false の箇所の batch 内の各遷移には次状態が存在しないため,next_state_values は 0 のままにする.
6. 期待値 E を式 (13) で計算し,expected_state_action_values に代入する.

$$E = Q(s', a_{max}) * GAMMA + reward(s, a) \tag{13}$$

7. 最適化関数で計算される勾配を初期化したうえで, 誤差関数に state_action_values と expected_state_action_values を引数として与え, それを逆伝搬し, 最適化関数による policy_net のパラメータの更新を行う. このときに学習を安定させるために policy_net のパラメータの勾配は最大値 1000 でクリッピングしている.

3.6 行動選択

3.6.1 ε -greedy 方策

使用した各パラメータの説明やその設定値を表 4 にまとめる.

表 4: ε -greedy 方策で行動選択を行うためのパラメータ

型	名前	説明
float	eps_threshold	式 (8) によって算出される閾値.0 以上 1 未満で生成した乱数がこの値より大きければモデルによって行動を選択し, そうでなければランダムに行動を選択する.
float	EPS_START	total_steps(表 1) が小さいときにより反映されるパラメータ. 今回は 1.0.
float	EPS_END	total_steps が大きいときにより反映されるパラメータ. 今回は 0.01.
int	EPS_DECAY	eps_threshold の変化量を減衰させるパラメータ.num_episode の 50 倍の値に設定した.

以下に, 行動選択のアルゴリズムについて示す. 但し, 引数で現状態が与えられるものとする.

1. fire_ball(表 1) が true のとき,fire_ball を false にし, ボールを出す行動を返して終了する.
2. eps_threshold を式 (14) で計算して更新する.

$$eps\_threshold = EPS\_END + \frac{(EPS\_START - EPS\_END)}{e^{\frac{total\_steps}{EPS\_DECAY}}} \tag{14}$$

3. 0 以上 1 未満で乱数を生成し,eps_threshold より大きければ policy_net に引数の現状態を入力し,Q 値が最大の行動を返す. そうでなければランダムに選択した行動を返す.

3.6.2 boltzman 方策

使用した各パラメータの説明やその設定値を表 5 にまとめる.

表 5: boltzman 方策で行動選択を行うためのパラメータ

型	名前	説明
tensor	ratio	式 (9) によって算出される各行動の閾値. 各行動が選択される確率を表す. 要素数 4(行動数) の float 型.
float	temperature	ratio を求める時に使用する温度パラメータ. 式 (10) で定義した.
float	BOLTZMAN_TEMPERATURE_DECAY	temperature の変化量を減衰させるパラメータ.num_episode の $\frac{1}{1000}$ の値に設定した.

以下に, 行動選択のアルゴリズムについて示す. 但し, 引数で現状態と現エピソード数が与えられるものとする.

1. fire_ball(表 1) が true のとき,fire_ball を false にし, ボールを出す行動を返して終了する.
2. ratio の各要素を式 (15) で計算して更新する. このときの Q(s,a) は policy_net に現状態 s を入力して得ている. また,temperature は式 (16) で定義している. 但し,i_episode は現エピソード数である.

$$ratio\_a_i = \frac{e^{\frac{Q(s,a_i)}{temperature}}}{\sum_j e^{\frac{Q(s,a_j)}{temperature}}} \tag{15}$$

$$temperature = \frac{1}{log(\frac{i\_episode}{BOLTZMAN\_TEMPERATURE\_DECAY} + 1.1)} \tag{16}$$

3. 0 以上 1 未満で乱数を生成し, その値が ratio の第一要素以下のとき, 行動 1 を返し,ratio の第一要素より大きく, 第二要素までの合計値 (第一要素 + 第二要素) 以下のとき, 行動 2 を返し,ratio の第二要素までの合計値より大きく, 第三要素までの合

計値以下のとき, 行動 3 を返し, 第三要素までの合計値より大きく, 第四要素までの合計値 (1 になる) 以下のとき, 行動 4 を返す.

3.7 学習させたモデルでの実行

3.4 節と基本動作は同じであるが, 学習のためのパラメータは全て削除し, 行動選択は学習させたモデルを読み込んだニューラルネットの出力から最大の Q 値を必ず選ぶようにした.

また, 行動ごとに observation(表 2) の画像を表示し, 動作過程を表示した上で, ゲーム終了時の画像を結果として保存した.

第 4 章　結果

100000 エピソードから 500000 エピソード学習済みモデルでの実行結果を 100000 エピソードごとに 8 ゲーム分収集した. その合計獲得報酬を表 6 に示す.

この平均値をグラフにしたものを図 9 に示す.

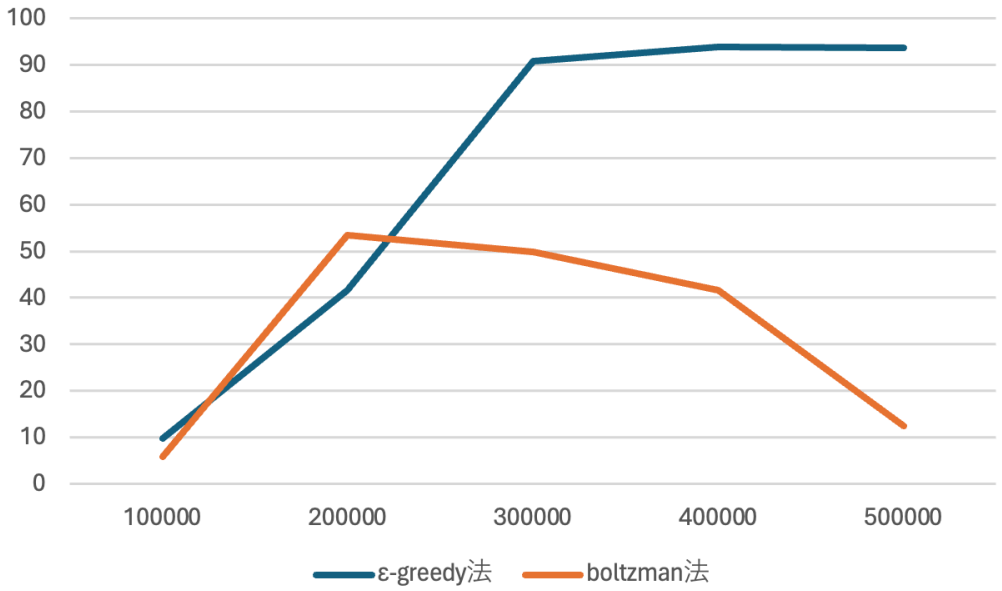

図 9　モデルの学習エピソード数による獲得報酬の変化 (平均値)

図 9 から, ε -greedy 方策で学習したモデルでは, 学習エピソード数と共に合計獲得報酬が増加していることがわかる. しかし,boltzman 方策で学習したモデルでは,200000 エピソードをピークに以降合計獲得報酬が減少し,500000 エピソードでは 100000 エピソードと同程度までになっていることがわかる.

次に, 表 6 の結果を箱ひげ図にまとめたものを図 10,11 に示す.

図 10,11 より, ε -greedy 方策で学習したモデルでは,500000 エピソードで分散が大きくなり,boltzman 方策で学習したモデルでは,200000 エピソードで分散が大きくなっていることがわかる.

表 6　学習させたモデルでの実行結果

モデルの学習エピソード数	100000	200000	300000	400000	500000
ε -greedy 方策による学習モデル	9	45	74	58	87
	10	51	326	43	30
	8	30	68	26	48
	7	39	56	31	22
	6	30	36	46	34
	11	48	41	71	110
	7	73	58	382	239
	20	17	67	94	180
平均値	9.75	41.625	90.75	93.875	93.75
中央値	8.5	42	62.5	52	67.5
boltzman 方策による学習モデル	4	90	51	25	4
	3	25	69	38	5
	9	20	47	25	5
	8	36	39	29	5
	8	51	48	56	56
	6	69	52	55	10
	4	51	26	56	7
	5	86	66	49	8
平均値	5.875	53.5	49.75	41.625	12.5
中央値	5.5	51	49.5	43.5	6

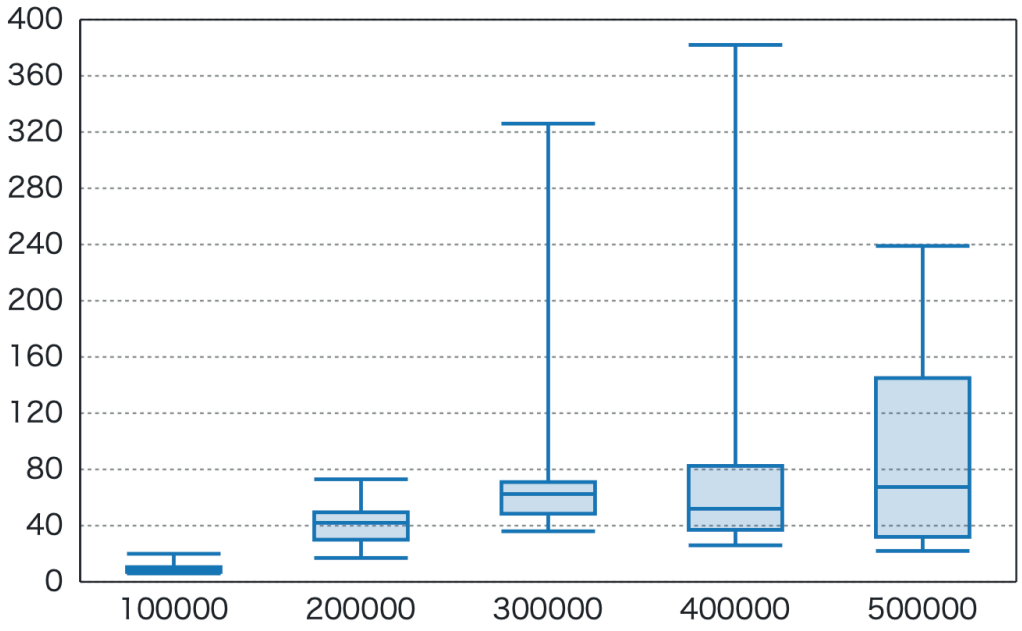

図 10　モデルの学習エピソード数による獲得報酬 (ε -greedy 方策)

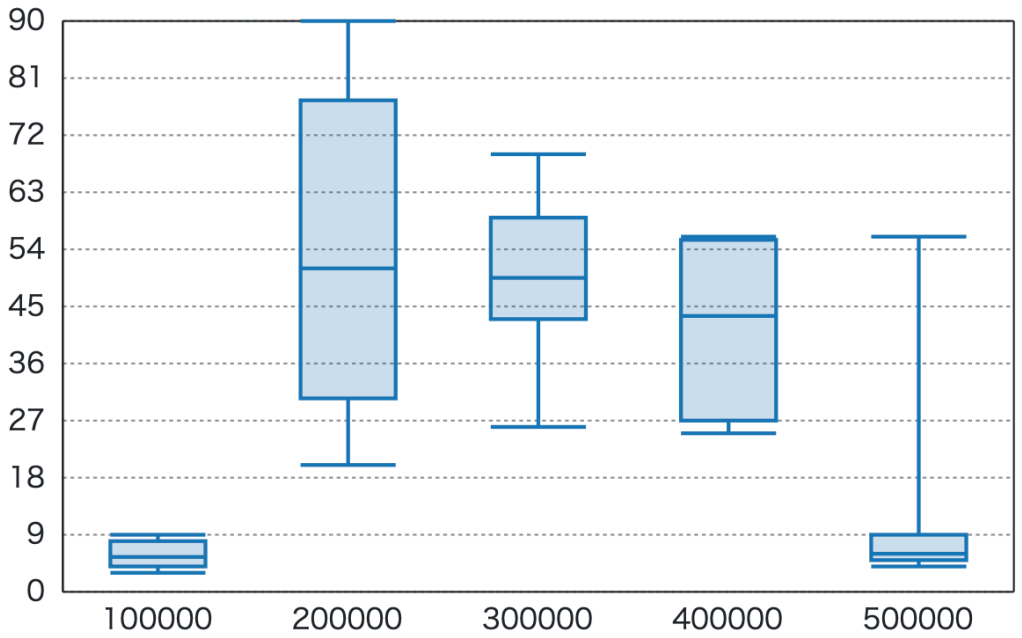

図 11　モデルの学習エピソード数による獲得報酬 (boltzman 方策)

第 5 章　考察

図 9 で boltzman 方策による学習モデルでは 200000 エピソードをピークに以降合計獲得報酬が減少していた. この理由については,200000 エピソードで比較すると,boltzman 方策の方が獲得報酬の平均が大きいことからも, 温度パラメータによって設定した探索と利用のバランスが 200000 エピソード付近で利用に傾いていたためであると考えられる. 以下に探索と利用のバランスに影響を与える ε -greedy 方策の eps_threshold と boltzman 方策の temperature と, それぞれ total_steps,i_episode との関係を示したグラフを図 12,13 に示す. また,eps_threshold と temperature の定義式を式 (17),(18) に再掲する.

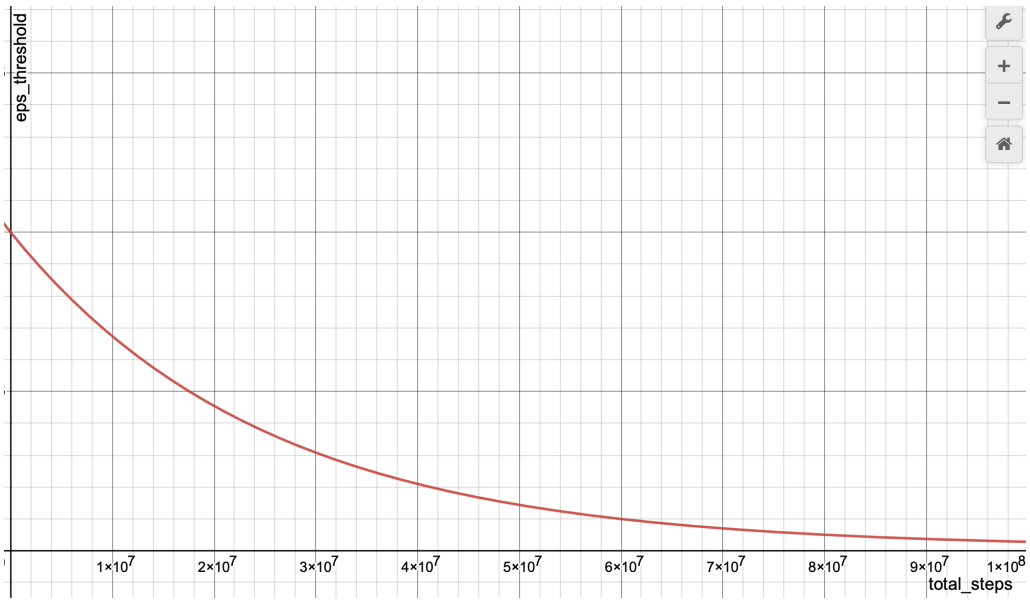

図 12　eps_threshold

$$
\begin{aligned}
eps\_threshold &= EPS\_END + \frac{(EPS\_START - EPS\_END)}{e^{\frac{total\_steps}{EPS\_DECAY}}} \\
&= EPS\_END + \frac{(EPS\_START - EPS\_END)}{e^{\frac{total\_steps}{50 \times 500000_{num\_episosde}}}}
\end{aligned}
\tag{17}
$$

図 13　temperature

$$
\begin{aligned}
temperature &= \frac{1}{log(\frac{i\_episode}{BOLTZMAN\_TEMPERATURE\_DECAY} + 1.1)} \\
&= \frac{1}{log(\frac{i\_episode}{\frac{num\_episode}{1000}} + 1.1)} \\
&= \frac{1}{log(\frac{i\_episode \times 1000}{(num\_episode)} + 1.1)}
\end{aligned}
\tag{18}
$$

また,boltzman 方策において各行動が選択される確率を表す ratio の定義式を式 (19) に再掲する.

$$
ratio\_a_i = \frac{e^{\frac{Q(s,a_i)}{temperature}}}{\sum_j e^{\frac{Q(s,a_j)}{temperature}}} \tag{19}
$$

式 (13) から,temperature が 0 に近づくほど各行動の ratio の差は大きくなり, 探索と利用のバランスが利用に傾くことがわかる. また, 図 13 から,200000 エピソードの時,temperature=0.3841 であるため,200000 エピソード学習モ

デルのニューラルネットにある程度の深さの状態を入力したときの Q 値 [3.6737,3.7578,3.8048,5.0534] をサンプルとして収集し, このときの ratio を計算すると [0.0250254011415,0.0311509503082,0.0352057253009,0.908617923249] となり,9 割近くの確率で行動 3 が選ばれることがわかる. よってこの状態からの最大値以外の行動を行う確率は 1 割以下であり, 探索と利用のバランスが利用に傾いていると考えられる.

これと ε -greedy 方策での探索と利用のバランスについて比較するために,total_steps と i_episode の単位を一致させなければならないため,200000 エピソード学習時点での total_steps を計測すると 20778123 であった.

図 12 の total_steps=20778123 の点を確認すると,eps_threshold=0.441203451138 であることがわかるため, 約 4.4 割の確率でランダムな行動を取ることがわかり, その中で Q 値が最大値の行動をとる確率は $\frac{1}{4}$ であるため, 最大値以外の行動を取る確率は 4.4-2.5 で 1.9 割の確率であることがわかる.

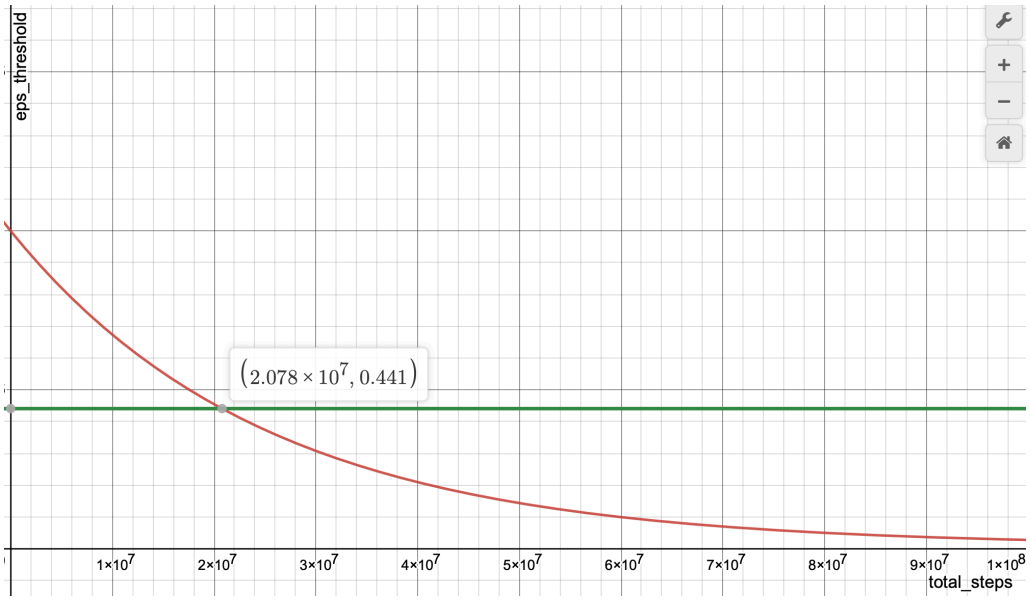

図 14　total_steps=20778123 のときの eps_threshold

これは boltzman 方策で学習したニューラルネットのサンプル状態における Q 値 [3.6737,3.7578,3.8048,5.0534] での最大値以外の行動を行う確率の約 2 倍の確率であるため, 探索と利用のバランスは, 今回のパラメータ設定においては boltzman 方策の

ほうが探索を重視できていたとわかる. 500000 エピソード学習において最終的な獲得報酬の値が ε -greedy 方策の方が大きくなったことから,boltzman 方策における学習モデルで獲得報酬が増加しなかった理由は探索と利用のバランスが利用に傾いていたことであると考えられる.

また, ε -greedy 方策での学習で 200000 エピソード学習時点での total_steps を計測するために,25000 エピソードごとに total_steps の値を記録したが, この測定結果から 200000 エピソードまでの total_steps の変化をラグランジュの補完法で多項式近似を行った. 計測結果を表 7 に, 近似した式を式 (20) に, そのグラフを図 15 に示す. 但し, 式 (20) において x はエピソード数である.

表 7　ε -greedy 方策での学習における各エピソード数での total_steps

モデルの学習エピソード数	total_steps
0	25
25000	892506
50000	1805256
75000	2819576
100000	4445224
125000	7105421
150000	10789130
175000	15513606
200000	20778123

$$total\_steps = -\frac{615523x^8}{15380859375 \times 10^{29}} + \frac{57877x^7}{1922607421875 \times 10^{20}} - \frac{283x^6}{3125 \times 10^{21}} + \frac{96649111x^5}{703125 \times 10^{19}} - \frac{204765737x^4}{1875 \times 10^{17}} + \frac{102676961x^3}{225 \times 10^{13}} - \frac{291782371x^2}{315 \times 10^9} + \frac{149950263x}{35 \times 10^5} + 25 \tag{20}$$

この式 (20) を微分した式がエピソードによる行動回数の変化を表すため, この分析は同環境におけるパラメータの調整に有効であると考える.

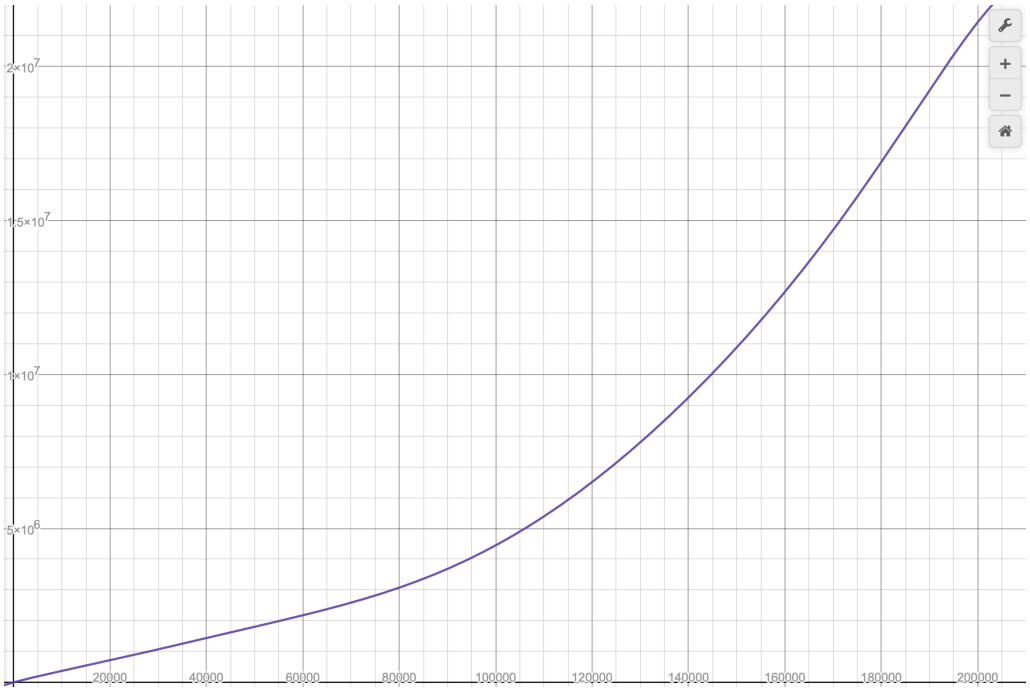

図 15　ε -greedy 方策での学習におけるエピソード数と total_steps の関係

次に，ε -greedy 方策で学習したモデルでは,500000 エピソードで分散が大きくなり,boltzman 方策で学習したモデルでは,200000 エピソードで分散が大きくなったことについて考える.

モデルによる実行では, 常に使用するニューラルネットでの出力の中から最適な行動を選択するため, 最終状態を変化させる要素は各ライフごとのボールを出す行動によるボールが放たれる方向のランダム性のみである. ランダム性を持った行動による遷移で十分な量, 深さで探索がされていれば, どの方向にボールが放たれたとしても最終的には獲得報酬が一様に上昇すると考えられるが, ランダム性を持った行動による遷移で十分に学習できていない時点で探索と利用のバランスが利用に傾くと, 特定の状態からは探索の深さが増し, 特定の状態からは増さなくなると考えられる. そうして学習されたモデルを使用したために獲得報酬のばらつきが大きくなると考えられる.

そのため, 探索と利用のバランスが利用に傾いていった点であると考えられる ε -greedy 方策で学習したモデルの 500000 エピソードと boltzman 方策で学習したモデルの 200000 エピソードで分散が大きくなったと考えられる.

また, このことから, ε -greedy 方策においても探索の期間はこれよりも伸ばすようにパラメータを設定すべきであったと考えられる.

第 6 章　まとめ

今回の実験ではモデルによる学習に膨大な時間を要したため, パラメータなどの調整に関して試行錯誤をすることができなかった.

このことに関してパラメータを結果論的に評価して調整するだけではなく最適であると考えられるパラメータを予測することに挑戦したいと思った.

本実験を通じて機械学習に関する様々な理解を深めることができたと感じたため, このことを卒業研究などにも生かしていきたい.

第 7 章　参考文献

[1] @sakigakeman,”初心者の初心者による初心者のためのニューラルネットワーク#1〜理論：順伝播編〜”,https://qiita.com/sakigakeman/items/55f4cc1c50f8236ac7a6,2023 年 12 月 18 日月曜日参照

[2] Lapinna 技術部,”活性化関数って何？”,https://qiita.com/Lapinna/items/de0fb85121186a83e1f3,2023 年 12 月 18 日月曜日参照

[3] 三谷 大暁,”DQN(Deep Q-Network) とは？ DQN で強化学習する方法を解説”,https://ai-kenkyujo.com/artificial-intelligence/algorithm/deep-q-network-toha/#:~:text=%E5%BE%93%E6%9D%A5%E3%81%AEQ%E5%AD%A6%E7%BF%92%E3%81%AE,%E3%81%AB%E5%AD%A6%E7%BF%92%E3%81%95%E3%81%9B%E3%82%8B%E6%96%B9%E6%B3%95%E3%81%A7%E3%81%99%E3%80%82,2024 年月日月曜日参照

[4] 山田 (ymd),”【強化学習】Experience Replay の理論”,https://zenn.dev/ymd_h/articles/c3ba23033a6442,2023 年 12 月 18 日月曜日参照

[5] Reinforz Insight 編集部,”Dueling Network AI 完全ガイド: 基本概念から先端研究までの深掘り解説”,https://reinforz.co.jp/bizmedia/10218/,2023 年 12 月 18 日月曜日参照

[6] 財満 誠,”Fast R-CNN を使った物体検出でより高速かつ高精度なモデルの仕組みと活用方法を解説”,https://zero2one.jp/learningblog/object-detection-fast-r-cnn/#:~:text=Smooth%20L1%20Loss は、ディープ, することができます%E3%80%82,2023 年 12 月 18 日月曜日参照

[7] そろくま,”【最適化手法】SGD・Momentum・AdaGrad・RMSProp・Adam を図と数式で理解しよう。”,https://kunassy.com/author/kunassy/,2023 年 12 月 20 日水曜日参照

[8] 小久保亘佑,”【機械学習】Optimizer(最適化関数) – SGD、Momentum、AdaGrad、RMSProp、Adam とは何か”,https://chefyushima.com,2023 年 12 月 20 日水曜日参照

[9] eisan fumio,”ニューラルネットワークにおける最適化手法（ＳＧＤからＡＤＡＭまで）を丁寧に理解しようとした”,https://qiita.com/Fumio-eisan/items/798351e4915e4ba396c2,2023 年 12 月 20 日水曜日参照

[10] Gymnasium Documentation,”Breakout”,https://gymnasium.farama.org/environments/atari/breakout/,2023 年 12 月 16 日土曜日参照

[11] Volodymyr Mnih,Koray Kavukcuoglu,David Silver,Alex Graves,Ioannis Antonoglou,Daan Wierstra,Martin Riedmiller,”Playing Atari with Deep Reinforcement Learning”,https://arxiv.org/abs/1312.5602,2023 年 12 月 16 日土曜日参照

[12] Takeru@Software Engineer,”Gymnasium で Atari Breakout を動かす【Google Colab】”,https://take-tech-engineer.com/gymnasium-atari-breakout/,2023 年 12 月 15 日金曜日参照

[13] Takeru@Software Engineer,”【全クリア】強化学習（DQN）で Atari Breakout を動かす【PyTorch】”,https://take-tech-engineer.com/dqn-atari-breakout-pytorch/,2023 年 12 月 15 日金曜日参照

[14] @forusufia,”線形分離可能のイメージをつかむ”,https://qiita.com/forusufia/items/53de18be0e07b258fc76,2023 年 12 月 18 日月曜日参照

[15] miyamoto keiichiro,”pytorch 超入門”,https://qiita.com/miyamotok0105/items/1fd1d5c3532b174720cd,2023 年 12 月 15 日金曜日参照

[16] The PyTorch Foundation,”Welcome to PyTorch Tutorials”,https://pytorch.org/tutorials/index.html,2023 年 12 月 15 日金曜日参照

[17] Suraj Subramanian,Seth Juarez,Cassie Breviu,Dmitry Soshnikov,Ari Bornstein”TENSORS”,https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html,2023 年 12 月 18 日月曜日参照

[18] @Seny,”namedtuple で美しい python を書く！（翻訳）”,https://qiita.com/Seny/items/add4d03876f505442136,2023 年 12 月 15 日金曜日参照

[19] Motoki Omura,”【深層強化学習】Dueling Network 実装・解説”,https://qiita.com/omuram/items/c03a2b2266c958564a9a,2023 年 12 月 18 日月曜日参照

[20] Ziyu Wang,Tom Schaul,Matteo Hessel,Hado van Hasselt,Marc Lanctot Nando de Freitas,”Dueling Network Architectures for Deep Reinforcement Learning”,https://arxiv.org/pdf/1511.06581.pdf,2023 年 12 月 18 日月曜日参照