プロジェクト研究 B　活動内容報告

東方花映塚の Gym 環境を用いた強化学習手法の提案

TAKUTO ITOI, Waseda Univ.

近年、囲碁の世界チャンピオンを相手に三連勝した AlphaGo[Silver et al. 2016] や、Atari 社が開発したビデオゲームを人間の平均スコア以上のパフォーマンスでプレイする人工知能 [Mnih et al. 2013] が開発され、ゲーム AI が注目されている。そこで、ゲーム AI の有用性をより複雑なゲームにおいても示す必要があると考えた。東方花映塚というゲームは画面が大量の弾丸で覆われるため、囲碁のように大量のデータを処理する必要がある。さらにターン制ではないゲームであるため、Atari 社のビデオゲームのようにリアルタイムでの処理が必要あるという特徴がある。これらの 2 つの特徴を東方花映塚が兼ね備えることから、囲碁やビデオゲームよりも複雑度の高いゲームであると考え、ゲーム AI の有用性を示すために取り組む題材として最適であると考えた。

本紙では、エージェントが学習するために、春学期に作成した東方花映塚をプレイすることができる Gym 環境を使用し、最適な行動を学習する手法を提案する。

CCS Concepts: • **Theory of computation → Reinforcement learning**.

Additional Key Words and Phrases: OpenAI, gym, PyTorch, Touhou, Kaeiduka, DuelingDDQN, PER, CurriculumLearning

ACM Reference Format:
Takuto Itoi. 2021. プロジェクト研究 B　活動内容報告: 東方花映塚の Gym 環境を用いた強化学習手法の提案. *ACM Trans. Intell. Syst. Technol.* -, -, Article - (February 2021), 4 pages. https://doi.org/10.1000/182

1　はじめに

東方花映塚とは、正式名称「東方花映塚　～ Phantasmagoria of Flower View.」であり、上海アリス幻樂団が東方 Project の第 9 作目として、2005 年に発売された対戦型弾幕シューティングゲームである。弾幕シューティングゲームとは、2D 画面において、敵が画面を埋め尽くすほど撃ってくる低速で大量の弾に当たらないように自機を操作しながら、敵に攻撃するゲームである。また、花映塚では東方 Project の他の作品と異なり、対戦型であり、敵を倒すことで相手フィールドにより多くの弾幕を送りつけることができる仕様になっており、最終的に相手を先に撃墜することが目的となっている。プレイヤーは上下左右への移動と、弾丸を発射することができ、またスペルカードを使用することで相手フィールドへ弾幕を送りつける事ができる。

春学期に当ゲームをプレイすることのできる Gym 環境を作成しており、今学期はこの環境上で最適な行動を取るエージェントの学習を様々な手法を用いて行なった。試した手法を述べるとともに、それぞれでどのような性能を得ることができたかを報告する。

Author's address: Takuto Itoi, Waseda Univ.

2157-6904/2021/2-ART- $15.00
https://doi.org/10.1000/182

2　先行研究

2.1　東方花映塚用 Gym 環境

春学期に東方花映塚をプレイする Gym 環境の作成をした。これは、実行中の東方花映塚が使用するメモリ上の数値を直接読み込み、各オブジェクトの位置などの盤面の状態を取得するツール花 AI 塚 [ide_an 2014] をベースとし、以下の 3 つのメソッドを用いてゲームを操作することができる。なお、実装の詳細については春学期のプロジェクト研究 A レポートで報告したためここでは省略する。また、春学期の実装では 1 ゲームが終了すると通信が終了する実装になっていたが、本ゲームのルールでは 3 度まで復活することが可能であるため、1 ゲームが終了したあと復活をし、ゲームオーバーになるまで通信を行なうように改良した。

2.1.1　reset(self). reset が呼ばれた際にはサーバとの接続を確立する。このときにサーバ側ではワーカースレッドの 1 つが接続を確立させ、DLL インジェクションを行ない、ゲームを起動するといった処理が実行される。

2.1.2　step(self, action). 東方花映塚では、自機の上下左右への移動と z キーを押したときに弾を打つ、シフトキーを押したときにバリアを張ることができる。これらの行動と引数 action を以下の表のように対応させた。

Table 1. action と行動の対応 (二進数)

攻撃\移動	上	下	左	右
なし	0000	0001	0010	0011
z	0100	0101	0110	0111
shift	1000	1001	1010	1011
z+shift	1100	1101	1110	1111

この表に基づいた数値をゲームを実行しているサーバに送信し、反対にゲームの現在の状況を表すデータを受け取る。メソッドの戻り値として盤面データと盤面の評価値を設定する。なお、盤面の評価値の計算方法は後述する。

2.1.3　close(self). また、close が呼ばれた際にはサーバにゲームを終了する命令を送信し、websocket の接続を終了する。

2.2　DoubleDQN(DDQN)

DDQN とは、環境におけるある行動の評価値を予想する Q 関数をニューラルネットワークによって求める DQN という手法の応用である。はじめに、Q 関数について考える。学習中のある時間 t における環境のパラメータ (今回の条件では上記した 9 次元のベクトル) を s_t とし、この環境においてある行動 a_t を行なったあとの状態で得られる報酬の期待値を計算する関数 Q_{main} を定義する。なお、時間 t において得られた報酬は r_t と表現する。

$$Q_{\text{main}}(s_t, a_t; \theta_t) = E(r_t + \gamma r_{t-1} + \gamma^2 r_{t-2} + \cdots) \tag{1}$$

なお、ここでの θ_t は Q 関数の定数パラメータの集合、γ は過去に得られた報酬が現在の学習に影響を与えないようにするための減衰率を決めるパラメータとする。ここで、式1における報酬をすべてそれぞれの時間における Q 関数の値に置き換えて変形すると

$$Q_{\text{main}}(s_t, a_t; \theta_t) = r_{t+1} + \gamma \max_a Q_{\text{main}}(s_{t+1}, a; \theta_{t+1}) \quad (2)$$

となるため、現在の Q 値 (Q 関数の出力値) が計算できる。しかし、この式は現在の Q 値を計算するために、未来の状態 (s_{t+1}) を用いているため、実際には計算不可能である。そこで、この出力値が何になるかをとりあえず予想して行動してみてから、あとで予測値との誤差から学習することでより正確な Q 値が計算できるようにする。ここで、Q 値の計算に用いられるのがパラメータ θ_t である。つまり、Q 学習における目標は、とりあえずやってみた行動で得られた報酬をなるべく近似できるように θ_{t+1} を設定することである。

また、この Q 関数はどのような関数でも良いが、反対にどのような関数が適しているかを導くことは非常に困難である。そこで、多層パーセプトロンのニューラルネットワークは十分なノード数があればどんな多項式でも近似することが可能である [Csáji et al. 2001] という性質を用いて、この Q 関数をニューラルネットで実装したものが DQN(Deep Q Network) である。

一方で、DQN を用いた Q 関数の近似では、学習時に直近の学習結果を過大に評価してしまうため、直近の結果に大きく依存してしまい学習効率がよくないと指摘されている [Han and Yang 2020]。そこで、学習時に Q 値を計算する Q 関数 (Q_{target}) を新しく用意し、そちらで Q 値を計算することで、直近の系の影響を強く受けないように改良したものとして DoubleDQN(DDQN) が提案された [Han and Yang 2020]。DDQN では、ある時刻 t での Q_{target} のパラメータを θ'_t として、出力値 y_t を

$$y_t = r_{t+1} + \gamma Q_{\text{target}}\left(s_{t+1}, \arg\max_a Q_{\text{main}}(s_{t+1}, a; \theta_t); \theta'_t\right) \quad (3)$$

と計算する。この値と Q_{main} が出力した Q 値との誤差からパラメータを調節し、学習を行なっている。

2.3 DuelingDDQN

DDQN の Q 関数として使用されるニューラルネットワークにおいて、出力層付近で一旦状態価値関数 $V(s)$ と Advantage 関数 $A(s,a)$ に分けてから再度結合する工夫をしたものが DuelingDDQN[Wang et al. 2016] である。これは、$V(s)$ のパラメータが行動に依存せずに学習される点が特徴として挙げられる。これにより、例えば東方花映塚では、自機がすでに弾に囲まれており、どのような行動をとったとしても得られる報酬が確定している場合などをより表現することができる。

2.4 PointNet

PointNet[Qi et al. 2017] は、点群データを対象としてクラスタリングや分類問題を解く手法である。主に自動運転や、建設事業での三次元計測データなどに応用されている。PointNet は順不変性、移動不変性、局所性によらず学習が可能という特徴がある。

PointNet のアーキテクチャを図1に示す。n 個のデータが与えられたときに、最後の層において n-Max Pooling を行なうことでデータ数に関係なく任意の次元数の出力を得ることができ、この出力が与えられた点群データ全体の特徴量として考えることができる。

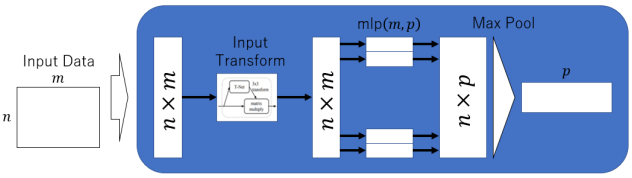

Fig. 1. PointNet のアーキテクチャ

3 提案手法

3.1 Q 関数

冒頭でも述べたように、東方花映塚というゲームは弾幕ゲームであり盤面上に大量の敵と弾が生成される。入力データが取りうる最大次元数を表2に示す。加えて、弾などのデータは順番を入れ替えても同じ盤面を表すことから、順不同な性質を持つ。そこで、Bullets, Enemies, Items, ExAttacks を PointNet へ入力し、PointNet からの出力をそれらの特徴量とし、この特徴量を結合して全結合ニューラルネットワークに渡している。なお、PyTorch において Q 関数の計算中の行列を concatenate した際にも計算グラフは消失しないため、逆誤差伝播が可能である。

Table 2. 入力データの次元数

データ名	データの個数	1 データあたりの次元数
Player	1	34
Enemies	128	16
Bullets	1000	13
Items	4	13
ExAttacks	256	14

また、東方花映塚では自機がすでに弾に囲まれており、どのような行動をとったとしても得られる報酬が確定している場合など、行動によらず報酬が決定してしまう場面が存在する。Q 関数に全結合ニューラルネットワークを使用すると盤面と行動の両方を参照して報酬を計算するため、上のようなケースは学習を遅くさせてしまう可能性がある。そこで、DuelingDQN を応用し、行動に依らない状態価値関数 $V(s)$ を用いて盤面評価を行なった。

以上のことから、本研究では図2のようなモデルを Q 関数としてエージェントの行動決定に使用した。

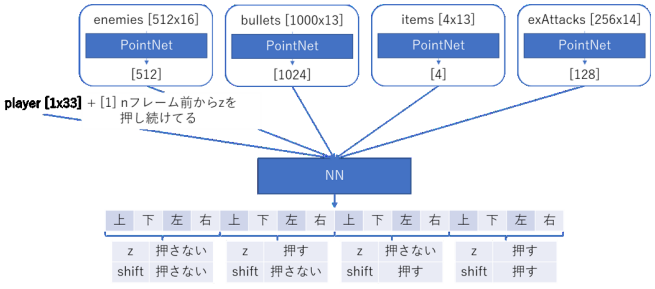

Fig. 2. Q 関数のモデル

このようなモデルを用いることで、敵や弾の情報を順不同的に扱えるのに加えて、点群データに関しては出力される特徴量を使用して計算することができるため、膨大なデータ数になっても計算量が増えないという利点がある。東方花映塚はリアルタイムで敵の位置を把握しながら行動することが求められるゲームであるため、計算に時間がかからないという性質は非常に重要である。

3.2 Prioritized Experience Replay (PER)

PER[Schaul et al. 2015] は、Q 関数のパラメータを最適化するための手法の 1 つである。学習を行なう際に、行動をした瞬間にパラメータを更新する場合、最近学習したことの影響が強くでてしまう傾向がある。さらに、単位時間あたりの環境の変化量はさほど大きくないことが多いため、最近の学習結果は似ている可能性が高く、似ている学習結果が多い場合、その場面における最適解を出力することに固執してしまい学習しづらくなる。また、学習する環境の情報のエントロピーが低い場合、学習曲線における勾配が小さくなってしまい、経験から得られる学習量が減ってしまい、過学習に陥りやすい [Hawkins 2004]。そこで、以前に経験したものをすべてメモリに保存し、学習時には過去の経験から無作為に抽出し、それをもとに学習する手法として PER が提案された。本研究の題材である東方花映塚も時間あたりの盤面の変化量が比較的小さいため、PER を用いて学習を行なった。

3.3 カリキュラム学習

加えて、学習がより早く進むために本研究ではカリキュラム学習を用いた。カリキュラム学習では、エージェントが学習する環境を、はじめは非常に簡単な課題にしておき、その環境において十分に良い結果が出せるようになったら、少しづつ課題の難易度を上げていくという手法である。

Q 学習は Q 値を学習するために最初はとりあえず行動をしてみるという手法であることから、学習初期ではランダムに近い挙動をするという性質がある。そのため与えられた環境において報酬を得る手段が非常に達成困難である場合にはどのような行動をとってもなかなか報酬を得ることができず、行動たちに優劣をつけることができなくなってしまい、正確に学習が行えない状況に陥ってしまう。そこで、学習初期には極小点を広く定義してある問題を人間が提供してあげることで楽に解に近づけるようにしてあげる。カリキュラム学習を用いることで、非常に達成困難な課題においても十分に成果を出すことが可能になることがある。また、報酬が複数の連続した行動によってのみ得られる課題などについてもカリキュラム学習を用いる (最初は 1 つの行動だけで報酬が得られるように設定するなど) ことで劇的に学習時間を削減することが可能になる。

本研究では表3に示すように 5 つのカリキュラムを用意し、それぞれの報酬の重みを示す。

Table 3. 設定したカリキュラム

報酬\カリキュラム	1	2	3	4	5
中央からの自機の距離	1	0.5	0.2		
一番近い弾・一番近い敵との距離		0.5	0.5	0.2	
自機と同じ x 軸上に敵がいるかどうか			0.3	0.5	0.2
倒した敵の数				0.3	0.5
得た得点					0.3

3.4 ϵ-Greedy 法

学習中にモデルが局所解に陥ってしまわないように、ある一定の確率 ϵ でランダムな行動を選択する手法である。この手法は、特にカリキュラム学習では必須の手法であり、カリキュラム (ゴール) が度々変化してしまうため、今までのエージェントの選択した行動のみに従ってしまうと条件が変わったときにうまく学習することができなくなってしまう。しかし、一定確率でランダムな行動を選択するため、運良くランダムな行動で良い報酬が得られた場合に学習を進めることができる。加えて、単純な Q 学習では基本的に Q 関数が選択した最良の選択肢を選択して行動してしまうため、逆に評価値の低い行動について学習することができない。この問題も、ランダムな行動をすることで、「よくない行動」についての学習もすることができ、理想の Q 関数 (すべての環境において最善の行動を導く関数) をより正確に近似することが可能になる。

ここで、ϵ-Greedy 法において、ϵ の値を大きく設定すると学習が早く収束するが、学習の精度が下がってしまう。一方で値を小さくすると学習が収束するまでに非常に長い時間がかかることが知られている。そこで、本研究では ϵ を学習時間に応じて変化させた。学習初期では精度よりも学習速度が重視され、またどの行動が最適かを確定することが難しいため、ランダムな行動が選択されやすく、学習が進むにつれて精度を上げるため、エージェント自信が選択した行動が取られるようにした。具体的には本研究では、1 イテレーション目では $\epsilon = 1$(完全ランダム) とし、毎イテレーション ϵ を 0.999 倍した。

4 学習結果

上述したモデルを実際に学習させ、1 ゲームごとの平均ロスと取得した報酬の合計を図3に示す。なお、ロスは Q_{target} が出力した値と実際の盤面の報酬との差の自乗とした。

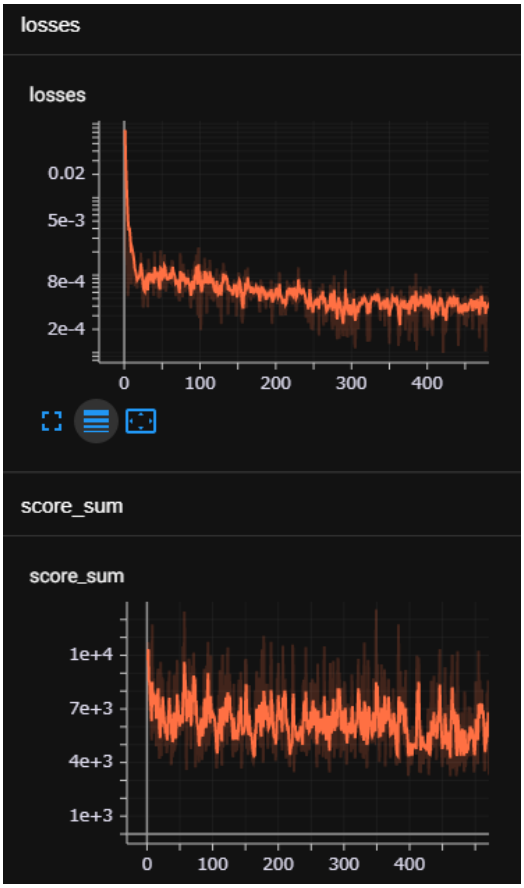

Fig. 3. 学習結果

5 考察

学習結果より、ロスはイテレーションを重ねるごとに減少していることが確認できる。一方でスコアの合計を示すグラフの値はほぼ変化がないことが確認できる。このことから、本研究において設定したモデルは盤面のスコアを正しく予測することができていたが、スコアが高いと思われる行動をとっても実際にスコアが上昇しないことがわかった。

これは、図3において、ロスは平均を表示しているが、スコアは 1 ゲームで取得したものの合計を表示している。このスコアの合計は、長く生き残るほど大きな値になるのに対し、エージェントは各盤面における最大報酬を得る学習を行なっており、必ずしも相関関係があるわけではない。

以上のことから、スコアが増加しなかった原因として、報酬設計がふさわしいものではなく、エージェントが長生きするための学習を行わなかった可能性が考えられる。これを解消するためには、死亡した際の報酬をそのゲームで得た報酬の総和などにすることで、回避できた可能性がある。一方でゲーム中には 1 フレームで得られた報酬を与えているため、大きな差が生じ、ロスが高くなり学習がうまく行かなくなってしまう可能性も考えられる。

6 今後の方針

本研究を通じて、報酬設計が大切であることがわかった。また、エージェントが長生きするように学習する報酬設計を考える必要があると考える。また、本研究では最適化手法に DuelingDDQN を使用したが、他にも強化学習には A3C などの手法も存在するため、それらを使用して学習をすれば、また違った研究成果が得られたと考える。

REFERENCES

Balázs Csanád Csáji et al. 2001. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary* 24, 48 (2001), 7.

Bao-An Han and Jian-Jun Yang. 2020. Research on Adaptive Job Shop Scheduling Problems Based on Dueling Double DQN. *IEEE Access* 8 (2020), 186474–186495.

Douglas M Hawkins. 2004. The problem of overfitting. *Journal of chemical information and computer sciences* 44, 1 (2004), 1–12.

ide_an. 2014. 東方花映塚 AI 自作ツール『花 AI 塚』の開発とその前後. http://www.usamimi.info/~ide/programe/touhouai/report-20140705.pdf.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

Unnat Singh. 2019. Deep Q-Network with Pytorch. https://medium.com/@unnatsingh/deep-q-network-with-pytorch-d1ca6f40bfda.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1995–2003.