アルゴリズムの解説と実験内容

担当：秋本　洋平
TA：河上　幸樹，齊藤　啓太
連絡先：jikken@bbo.cs.tsukuba.ac.jp

October 10, 2023

Contents

1 本テーマの目的

本テーマでは，深層強化学習法である TD3（Twin Delayed Deep Deterministic Policy Gradient アルゴリズム）[1] を実装し，シミュレータ上で二足歩行ロボットの歩行動作を獲得する．この実験を通して，以下の知識と技能を獲得することを期待する：

1. 方策勾配法や Actor-Critic 法と呼ばれる強化学習法の基本的な設計理念
2. ニューラルネットワークを扱うフレームワークの使い方
3. 確率的な振る舞いをするシミュレーションにおける実験技法

2 強化学習の説明

ここでは，本実験を実施するにあたり最低限必要な強化学習についての知識をまとめる．より網羅的に強化学習を学びたい場合には文献 [5, 7, 8] などを参照されたい．

2.1 強化学習の定式化

強化学習エージェントは，タイムステップ t において，環境から状態 s_t を観測し，自身の持つ行動決定則に応じて行動 a_t を決定する．この行動決定則のことを**方策**（*policy*）と呼ぶ．行動 a_t を実行することで次状態 s_{t+1} へと遷移する．その際，環境から報酬 r_t を受け取る．これを 1 ステップと呼ぶ．本実験で考える強化学習の目的は，初期状態 s_1 によらずに割引報酬和 $\sum_{t=1}^{T} \gamma^{t-1} r_t$（減衰率 $0 < \gamma \leqslant 1$）の期待値を最大にするような方策を獲得することである．このような方策を**最適方策**と呼ぶ．

本実験で考える強化学習エージェントと環境は，以下のように定義される．

環境 状態空間 $\mathcal{S} \subseteq \mathbb{R}^n$ および行動空間 $\mathcal{A} \subseteq \mathbb{R}^m$ はいずれも実ベクトル空間の部分集合とする．状態 $s \in \mathcal{S}$ で行動 $a \in \mathcal{A}$ をとったとき，確率密度関数 $p(s'; s, a)$ に従って次状態 s' に遷移する．これを状態遷移確率と呼ぶ．その際に与えられる即時報酬は $r(s, a) \in \mathbb{R}$ で与えられる．初期状態 s_1 は確率密度関数 $p_1(s)$ に基づいて決定される．状態が終端状態に到達するか最大ステップ数 T に達したとき，終了シグナルをだし，状態を p_1 に従いリセットする．

エージェント エージェントは学習したい方策（*target policy*）π と，環境とのインタラクションの際に実際に利用する行動方策（*behavior policy*）β を持つ．π は状態から行動への決定的な関数（$a = \pi(s)$）とし，β は確率密度関数 $\beta(a; s)$ である．β は多くの場合 π に確率的摂動を加えることで得られる．エージェントは状態や報酬を観測できるが，その背後にある状態遷移確率や報酬関数についての知識は無いものとする．

ここで，β として π と異なる確率的方策を採用する理由は，学習結果として得たい方策は決定的方策であるが，決定的な方策では行動空間の探索が困難なためである．

学習中の各ステップは以下のようになる．

1: 行動を選択 $a_t \sim \beta_t(a; s_t)$
2: 行動 a_t を実行し，次状態 s_{t+1}，報酬 r_t，終了シグナル `done` を観測
3: 方策 π_t を更新

例：Toy Task 環境　概念を理解するために，単純な例を図1に示す．この例では，状態遷移が決定的である（ある状態 s である行動 a をとると，次の状態は一意に定まる）．

- A, B, C：エージェントの状態 s_t．エージェントは A か B か C の状態にある．
- L, R：エージェントの行動 a_t．各状態においてエージェントは L（左に移動）か R（右に移動）の行動を取る．
- $\rightarrow$：エージェントの状態遷移．エージェントがとった行動に従って状態が変化する．
- $0, 1$：エージェントの獲得する報酬 r_t．

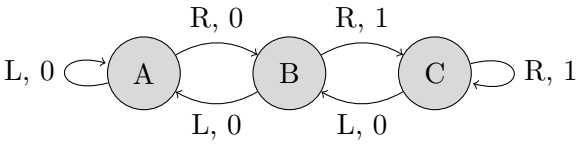

Figure 1: Toy Task 環境

二つの方策を例に取り，方策と割引報酬和の関係を示す．まず，$\pi(A) = R$, $\pi(B) = R$, $\pi(C) = R$ となる方策を考える．すなわち，常に行動 R を取る方策である．このとき，$T = +\infty$ ならば，$s_t = A$ のとき割引報酬和は $R_t = \gamma/(1-\gamma)$，$s_t = B, C$ の場合には割引報酬和は $R_t = 1/(1-\gamma)$ となる．割引報酬和の定義に沿って考えれば容易にわかる．同様にして，$\pi(A) = R$, $\pi(B) = R$, $\pi(C) = L$ となる方策を考える．このとき，$s_t = A, C$ の場合の割引報酬和は $R_t = \gamma/(1-\gamma^2)$，$s_t = B$ の場合の割引報酬和は $R_t = 1/(1-\gamma^2)$ となる．この二つのケースについて見れば，前者の方がどの状態にいても高い割引報酬和が得られる事になり，前者の方が望ましい方策であることがわかる．

実際に強化学習を適用する場面では，状態遷移は未知である．すなわち，状態 B で行動 R をとると状態 C に遷移して報酬 1 を受け取る，というルールは未知である．そのため，状態 B で行動 R を実際に取る事によって，状態遷移や得られる報酬についての情報を集めていき，推測される割引報酬和を最大にするような方策 π を見つけていく必要がある．そのために，様々な状態で様々な行動を取るための方策が行動方策 β である．

2.2　価値関数

各状態の価値を定義する．まず，**最適価値** $V^*(s)$ は，初期状態が s であるときに達成可能な割引報酬和の期待値の最大値によって定義される．これに対し，方策 π のもとでの状態 s の**価値** $V^\pi(s)$ は，初期状態が s であるときに π に従って行動した結果として得られる割引報酬和の期待値 $V^\pi(s) = \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t \mid S_1 = s]$ で定義される．任意の方策について $V^*(s) \geqslant V^\pi(s)$ であり，強化学習の目的は $V^*(s) = V^\pi(s)$ を全ての $s \in \mathcal{S}$ で実現するような最適方策 π を見つけることである．

行動についても価値を導入する．**行動価値** $Q^\pi(s, a)$ は初期状態 s で行動 a を選択した後にその後のステップにおいて方策 π に従った行動選択をした場合に得られる割引報酬和の期待値 $Q(s, a) = \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t \mid S_1 = s, A_1 = a]$ で定義される．方策 π が決定的方策である場合，$V^\pi(s) = Q(s, \pi(s))$ である．状態価値の場合と同様，**最適行動価値** $Q^*(s, a)$ は初期状態 s で行動 a を選択した後に達成し得る割引報酬和の期待値の最大値で定義される．

最適価値関数と最適行動価値関数の間には以下のような関係がある．

1. $V^*(s) = \sup_{a \in \mathcal{A}} Q^*(s, a)$
2. $Q^*(s, a) = r(s, a) + \gamma \int_{\mathcal{S}} V^*(s') p(s'; s, a) \mathrm{d}s'$

１つ目の関係式は，各状態での最適な行動は（存在するならば）各状態 s において $Q^*(s,a)$ を最大にするような a を出力する方策：$\pi(s) = \mathrm{argmax}_{a\in\mathcal{A}} Q^*(s,a)$ によって与えられることを表す．すなわち，最適な方策は決定的方策で与えられる．（これが決定的な方策を π としたい一つの理由である．）そのため，最適行動価値関数が得られれば，最適方策が得られる．２つ目の関係式は，最適行動価値関数が満たすべき条件を表している．最適行動価値関数を推定するためにこの条件式が活用される．代表的な方法として，本実験で実装するテーブル Q 学習（3.1節および3.2節）が挙げられる．

例：Toy Task 環境　先程の Toy Task 環境における二つの方策の行動価値関数は表1のようになる．

Table 1: 方策と Q 値の関係

(a) $\pi(A) = R$, $\pi(B) = R$, $\pi(C) = R$ の場合

State	L	R
A	$\gamma^2/(1-\gamma)$	$\gamma/(1-\gamma)$
B	$\gamma^2/(1-\gamma)$	$1/(1-\gamma)$
C	$\gamma/(1-\gamma)$	$1/(1-\gamma)$

(b) $\pi(A) = R$, $\pi(B) = R$, $\pi(C) = L$ の場合

State	L	R
A	$\gamma^2/(1-\gamma^2)$	$\gamma/(1-\gamma^2)$
B	$\gamma^2/(1-\gamma^2)$	$1/(1-\gamma^2)$
C	$\gamma/(1-\gamma^2)$	$1/(1-\gamma^2)$

2.3　方策勾配

方策がパラメータ θ によって表される関数モデル π_θ によって表現されるとする．方策 π のもとでの行動価値関数 Q^π がわかっているならば，上述の強化学習の目的は任意の $s \in \mathcal{S}$ および $a \in \mathcal{A}$ において $V^*(s) = V^\pi(s)$ を達成する π を求めることであるが，関数モデルを用いた方策表現のもとでは任意の初期値 $s \in \mathcal{S}$ に対して上記の等式を満たす方策 π_θ が存在するとは限らない．そこで，より現実的な目的関数として，π についての状態価値 $V^\pi(s)$ の β のもとでの期待値 $J^\beta(\pi) = \mathbb{E}[V^\pi(S)]$ を考える．（ここで，期待値は β に従って行動選択し続けた場合の状態 S の定常分布について取られている．定常分布は，初期状態から十分なステップ数遷移したあとの状態が従う確率分布と理解しておけばよい．）方策は θ によって表されるため，以後これを $J^\beta(\theta)$ と書くことにすれば，目的は $J^\beta(\theta)$ を最大化するような θ を求めることとなる．

決定的方策についての方策勾配定理 [4] は，$J^\beta(\theta)$ の θ についての勾配を近似する術を与える．決定的方策 π のもとで状態価値は $V^\pi(s) = Q^\pi(s, \pi(s))$ となることに注意すれば，β が θ に依存しない場合に以下を得る．

$$\nabla_\theta J^\beta(\theta) \approx \mathbb{E}[\nabla_a Q^\pi(S,a)|_{a=\pi(S)} \nabla_\theta \pi_\theta(S)]$$

すなわち，定常分布に従う状態のサンプル $s_1, \ldots, s_N$ が得られれば，

$$G = \frac{1}{N} \sum_{i=1}^{N} \nabla_a Q^\pi(s_i,a)|_{a=\pi(s_i)} \nabla_\theta \pi_\theta(s_i)$$

は $\nabla_\theta J^\beta(\theta)$ の不偏な推定値を与える．現実には厳密に定常分布に従う状態のサンプルは得られないが，β に従って行動した過去の系列からランダムに N 個の状態を抽出すれば，これをもって近似する事ができる．[1]

決定的方策を用いた Actor-Critic 法は，方策 π のもとでの行動価値関数や状態価値関数を Q 学習のような方法を用いて推定しつつ，推定された価値関数を用いた方策勾配に

[1] 一般に方策勾配定理と言えば，上述の定理でなく文献 [6] などに示されている確率的方策に対する勾配定理を指すことに注意されたい．

より方策 π を逐次的に改善していく強化学習手法である．Actor-Critic 法では，価値評価器を *critic* と呼び，行動決定器を *actor* と呼ぶ．本実験で実装する TD3 など（3.3節および3.4節）は，Actor と Critic の両者にニューラルネットワークを用いた Actor-Critic 法である．

3 実装する強化学習アルゴリズム

3.1 方法１：テーブル Q 学習

テーブル Q 学習は，最適行動価値関数の推定値をテーブル形式で保持・学習していく代表的な強化学習法である．テーブル Q 学習では状態空間と行動空間をそれぞれ K 個と L 個の互いに素な部分集合 $\mathcal{S} = \cup_{i=1}^{K} \mathcal{S}_i$，$\mathcal{A} = \cup_{j=1}^{L} \mathcal{A}_j$ に分割する．行動空間については，各部分集合に対応する代表点 $a_j \in \mathcal{A}_j$（$j = 1, \ldots, L$）を決めておく．行動価値関数 Q は $K \times L$ のテーブル $Q_{[i,j]}$ で近似される．状態や行動からそれぞれの対応するインデックスへの写像を $\mathtt{idx}_s(s)$，$\mathtt{idx}_a(a)$ と表せば，$Q(s,a) = Q_{[\mathtt{idx}_s(s),\mathtt{idx}_a(a)]}$ となる．各学習ステップにおいて，Q テーブルは以下のようにして更新される．

$$\delta \leftarrow r_t + \gamma \max_{0 \leqslant j \leqslant L} Q_{[\mathtt{idx}_s(s_{t+1}),j]} - Q_{[\mathtt{idx}_s(s_t),\mathtt{idx}_a(a_t)]} \tag{1}$$

$$Q_{[\mathtt{idx}_s(s_t),\mathtt{idx}_a(a_t)]} \leftarrow Q_{[\mathtt{idx}_s(s_t),\mathtt{idx}_a(a_t)]} + \alpha\delta \tag{2}$$

ここで，α は学習率であり，$\alpha = 1$ であれば，最適行動価値関数の満たすべき条件式が成り立つように Q 関数を更新しようとしていることがわかる．実際には，観測された次状態は確率的に決まることや，Q 関数の推定値はあくまで推定値であるためこれを最大にする行動は必ずしも最適ではないことなどから，学習率は $\alpha < 1$ とする．Q 学習エージェントは方策 π を陽に持つ必要がなく，$\pi(s) = a_{\mathrm{argmax}_{1 \leqslant j \leqslant L} Q_{[\mathtt{idx}_s(s),j]}}$ で計算される．

上記の Q 関数の更新式と最適行動価値関数の関係について，補足する．最適価値関数と最適行動価値関数の性質から，

$$Q^*(s,a) = r(s,a) + \gamma \int_{s' \in \mathcal{S}} p(s'; s, a) \sup_{a' \in \mathcal{A}} Q^*(s', a') \mathrm{d}s' \tag{3}$$

が満たされることがわかる．いま，状態が $s_t = s$，選択した行動が $a_t = a$ であったとすれば，得られた報酬は $r_t = r(s_t, a_t)$ である．s_{t+1} は確率分布 $p(s'; s_t, a_t)$ からのサンプルであるから，$\sup_{a' \in \mathcal{A}} Q^*(s_{t+1}, a')$ は右辺の期待値計算を（標本数１の）モンテカルロ推定であると考えられる．真の Q^* は未知であるが，現時点での推定値である Q 関数で近似できるとすれば，右辺は，

$$r_t + \gamma \max_{0 \leqslant j \leqslant L} Q_{[\mathtt{idx}_s(s_{t+1}),j]} \tag{4}$$

と近似される．これは，$\delta + Q_{[\mathtt{idx}_s(s_t),\mathtt{idx}_a(a_t)]}$ に他ならない．すなわち，Q 関数が最適行動価値関数を近似できているならば，δ は 0 に近い値を取るはず（上記の等式を満たすはず）である．そうでない場合には，現在の状態行動対 (s_t, a_t) についての Q 関数の値は，Q 関数を用いて近似される最適行動価値関数の推定値から δ だけ小さな（大きな）値となっているため，その方向に値を修正する，ということが行われている．あくまで推定値であるため，小さな学習率 α が導入されている，というわけである．

3.2 方法２：経験再生を利用したテーブル Q 学習

上述のテーブル Q 学習では，一度のインタラクションの結果をその場で学習に利用し，再度利用することは無い．**経験再生**（*experience replay*）[3] は，環境とのインタラクションの履歴を保存しておき，これを再利用する枠組みであり，サンプル効率（環境とのインタラクション数に対する学習効率）を改善することを目的としている．経験再生では，履歴

を保存するキュー $\mathcal{H}$ を用意し，毎ステップ経験 (s_t, a_t, s_{t+1}, r_t) を追加していく．学習時は決められた数（**バッチサイズ**）だけ $\mathcal{H}$ からランダムに経験を選択し，これらを用いて Q テーブルを学習する．

以下は，各ステップにおける学習を擬似コードで表したものである．擬似コードにおいて，$\delta^{\texttt{done}}$ は遷移後の状態 s' が終端状態ならば 1，そうでなければ 0 となる．

---

procedure TABLE Q-LEARNING WITH EXPERIENCE REPLAY($\mathcal{H}$)
  if $|\mathcal{H}| \geqslant$ `batch_size` **then**
    for $i = 1, \ldots,$ `batch_size` **do**
      Sample $h = (s, a, s', r, \delta^{\texttt{done}})$ uniform-randomly from $\mathcal{H}$
      Compute $\delta \leftarrow r + (1 - \delta^{\texttt{done}})\gamma \max_{0 \leqslant j \leqslant L} Q_{[\texttt{idx}_s(s'), j]} - Q_{[\texttt{idx}_s(s), \texttt{idx}_a(a)]}$
      Update $Q_{[\texttt{idx}_s(s), \texttt{idx}_a(a)]} \leftarrow Q_{[\texttt{idx}_s(s), \texttt{idx}_a(a)]} + \alpha\delta$
    end for
  end if
end procedure

補足：終了フラグについて　まず,強化学習では多くの場合に無限タイムステップ($T = \infty$, infinite horizon）を想定していることに注意されたい．$T = \infty$ の場合，現状態での割引報酬和が R_t，そこでの状態行動対が (s_t, a_t)，その時の即時報酬が $r_t = r(s_t, a_t)$，次状態での割引報酬和が R_{t+1} であれば，定義より $R_t = r_t + \gamma R_{t+1}$ が成り立つ．この再帰的な関係が，最適行動価値関数の再帰的な関係や Q 関数の更新式のベースとなっている．

各エピソードが有限タイムステップ（$T < \infty$）の場合であっても，最適方策を無限系列についての割引報酬和を最大にする方策と考えるならば，Q 関数が推定の対象とするものは上記と同じ最適行動価値関数となる．そのため，T ステップ後のエピソードの初期化（状態の初期化）を Q 関数の更新後に行っている限り（s' はあくまで本来状態遷移されるはずであった次状態を表し，次のステップの開始時に s を初期化する），上のテーブル Q 学習の擬似コードにあるような終了フラグを導入する必要はない．

ただし，終端状態が存在するようなタスクの場合には，注意が必要である．終端状態とは，その状態に到達したら状態遷移を終了するような状態（エピソードが強制的に終わるような状態）を表す．例えば，ビデオゲームにおいて，ゴールに辿り着いた場合（ゴール状態）やクラッシュした場合などがこれに当たる．終端状態での状態価値や行動価値は，原理上定義されない．そのため，終端状態に遷移するような場合，最適行動価値関数の再帰式は満たされない．価値が割引報酬和の期待値であり，終端状態以降は報酬が得られないことから割引報酬和が 0 になることに注意すれば，終端状態に遷移するような場合には，行動価値は即時報酬 $r(s, a)$ そのものとなることがわかる．すなわち，終端状態での状態行動価値関数を 0 と定めている場合には，再帰式が満たされることになる．実装上は，終端状態に遷移した場合には，Q 関数の更新の目標値を即時報酬 r にする（$\delta = r - Q(s, a)$ とする）する．この場合，終端状態での Q 値は初期化以降更新されず，無意味な値が入っていることに注意されたい．（同時に，これを利用することが不適切であることも理解されたい．）

3.3　方法３：経験再生を利用した Actor-Critic

Actor と Critic はともにパラメータ θ および ω によって表されるニューラルネットワークであるとする．Critic $Q^{\omega}(s, a)$ は現在の方策 π_θ のもとでの行動価値関数 $Q^{\pi}(s, a)$ を推定することを目的とし，actor π_θ は critic のもとでの方策勾配に従って価値の期待値 $J^{\beta}(\theta)$ を最大化することを目的とする．

経験再生を利用しない場合，各ステップでの Critic の更新は，更新後の critic の値 $Q^{\omega'}(s_t, a_t)$ が $\Delta_t = r_t + \gamma Q^{\omega}(s_{t+1}, \pi(s_{t+1}))$ に近づくように行われる．これは，行動価値

関数について成り立つ式 $Q^\pi(s,a) = r(s,a) + \gamma \int_{\mathcal{S}} Q^\pi(s', \pi(s'))p(s';s,a)\mathrm{d}s'$ から来ている．（テーブル Q 学習における方策が $\pi(s) = a_{\mathrm{argmax}_{1 \leqslant j \leqslant \ell} Q_{[\mathrm{idx}_S(s),j]}}$ であったことに注意すれば，critic の更新式が Q テーブルの更新式と対応していることがわかる．）具体的には，critic の誤差関数を

$$L_Q(\omega) = (\Delta_t - Q^\omega(s_t, a_t))^2 \tag{5}$$

と定義し，勾配法に基づき ω を $L_Q(\omega)$ の勾配の逆方向に少し動かす．経験再生を用いる場合には，上の誤差関数がバッチサイズ個分の平均二乗誤差となる．

Actor の更新は，2.3節にて紹介した決定的方策のもとでの方策勾配を用いて行われる．このとき，方策 π のもとでの行動価値関数 $Q^\pi(s,a)$ は現在の critic$Q^\omega(s,a)$ をもって近似する．行動方策 β のもとでの定常分布からのサンプル $s_1, \ldots, s_{\texttt{batch\_size}}$ は経験再生バッファーからランダムに取り出したもので近似する．

以下は，各ステップにおける学習を擬似コードで表したものである．

---

procedure DETERMINISTIC ACTOR-CRITIC WITH EXPERIENCE REPLAY($\mathcal{H}$)
  if $|\mathcal{H}| \geqslant \texttt{batch\_size}$ **then**
    Sample $h_i = (s_i, a_i, s'_i, r_i, \delta_i^{\texttt{done}})$ uniformly from $\mathcal{H}$ for $i = 1, \ldots, \texttt{batch\_size}$
    Compute $\Delta_i = r_i + (1 - \delta_i^{\texttt{done}})\gamma Q^\omega(s'_i, \pi_\theta(s'_i))$ for $i = 1, \ldots, \texttt{batch\_size}$
    Compute $G_\omega = \nabla_\omega \sum_{i=1}^{\texttt{batch\_size}} (\Delta_i - Q^\omega(s_i, a_i))^2 / \texttt{batch\_size}$
    Update $\omega \leftarrow \omega - \alpha_\omega G_\omega$
    Compute $G_\theta = \nabla_\theta \sum_{i=1}^{\texttt{batch\_size}} Q^\omega(s_i, \pi_\theta(s_i)) / \texttt{batch\_size}$
    Update $\theta \leftarrow \theta + \alpha_\theta G_\theta$
  end if
end procedure

3.4 方法４：TD3

TD3 は上述の Actor-Critic の学習を安定化させるために，複数の工夫を施している．

Target Actor & Target Critic Critic の学習では，行動価値 $Q^\omega(s_t, a_t)$ をターゲット Δ_t に近づけるように ω を更新するが，ターゲット自体が ω に依存するため critic の更新が不安定になることが知られている．これを解消するために，target actor $\pi_{\bar{\theta}}$ と target critic $Q^{\bar{\omega}}$ を用意し，Δ_t の計算の際に π_θ および Q^ω に代わりこれらを用いる．Target actor と target critic のパラメータは，通常の actor と critic よりも緩やかに更新されるよう，各ステップにおいて $\bar{\theta} \leftarrow (1-\tau)\bar{\theta} + \tau\theta$, $\bar{\omega} \leftarrow (1-\tau)\bar{\omega} + \tau\omega$（$\tau \ll 1$）のように更新される．なお，これらのパラメータは $\bar{\theta} = \theta$，$\bar{\omega} = \omega$ と初期化される．（方法３にこれを加えたものが DDPG [2] と呼ばれる Actor-Critic 法である．）

Target Policy Smoothing Regularization 深層ネットワークを critic に用いることから，過学習によって望ましくないピークが critic に現れることがある．方策は critic によって近似される行動価値を最大にするような方策を学習するため，望ましくないピークを出力してしまうような方策が獲得される恐れがある．これを解消するために，“近い行動の価値は近いはず” という仮定のもと，行動価値関数がなめらかになるよう，Δ_t 計算の際に target actor の出力 $\pi_{\bar{\theta}}$ にノイズ $\epsilon \sim \mathrm{clip}(\mathcal{N}(0, \sigma_{\mathrm{target}}), -c, c)$ を加える．ただし，行動が取りうる範囲をはみ出した場合には，境界上に射影する．

Delayed Policy Update Actor の学習は critic の精度に依存するため，critic の学習は actor よりも早く進むことが望ましい．そこで，actor の更新を d ステップに一度だけ行うようにする．

Clipped Double Q-Learning Q 学習において，推定された行動価値 $Q^{\omega}(s,\pi(s))$ が真の行動価値 $Q^{\pi}(s,\pi(s))$ よりも大きく見積もられる問題が知られている．行動価値の過大評価を防ぎ，価値の推定精度を改善するために，critic を二つ（Q^{ω_1}，Q^{ω_2}）用意し，これらをそれぞれ学習させる．二つの critic の更新におけるターゲット Δ_i を計算する際は，$Q^{\omega_1}(s,a)$ と $Q^{\omega_2}(s,a)$ のうち小さい値を採用する．

各ステップにおける TD3 の学習を擬似コードで表すと以下のようになる．

---

procedure TD3($\mathcal{H}$)
  $t \leftarrow t+1$
  if $|\mathcal{H}| \geqslant$ `batch_size` **then**
    Sample $h_i = (s_i, a_i, s_i', r_i, \delta_i^{\texttt{done}})$ uniformly from $\mathcal{H}$ for $i = 1, \ldots,$ `batch_size`
    Compute $\bar{a}_i = \pi_{\bar{\theta}}(s_i') + \epsilon_i$ where $\epsilon_i \sim \mathrm{clip}(\mathcal{N}(0,\sigma), -c, c)$ for $i = 1, \ldots,$ `batch_size`
    Compute $\Delta_i = r_i + (1-\delta_i^{\texttt{done}})\gamma \min_{j=1,2} Q^{\bar{\omega}_j}(s_i', \bar{a}_i)$ for $i = 1, \ldots,$ `batch_size`
    Compute $G_{\omega_j} = \nabla_{\omega} \sum_{i=1}^{\texttt{batch\_size}} (\Delta_i - Q^{\omega_j}(s_i, a_i))^2 / \texttt{batch\_size}$ for $j = 1, 2$
    Update $\omega_j \leftarrow \omega_j - \alpha_{\omega} G_{\omega_j}$ for $j = 1, 2$
    if t = d **then**
      Compute $G_{\theta} = \nabla_{\theta} \sum_{i=1}^{\texttt{batch\_size}} Q^{\omega_1}(s_i, \pi_{\theta}(s_i)) / \texttt{batch\_size}$
      Update $\theta \leftarrow \theta + \alpha_{\theta} G_{\theta}$
      Update $\bar{\theta} \leftarrow (1-\tau)\bar{\theta} + \tau\theta$
      Update $\bar{\omega}_j \leftarrow (1-\tau)\bar{\omega}_j + \tau\omega_j$ for $j = 1, 2$
      $t \leftarrow 0$
    end if
  end if
end procedure

---

4 実験の内容

上述の方法 1 から方法 4 を `python` で実装し，OpenAI の提供する強化学習環境 `gym` において，これらの方法を比較・評価する．

4.1 実験環境の構築

4.1.1 `gym` のインストール

以下，Anaconda がインストールされていることを前提とする．まず，仮想環境 gym（python の version は 3.6）を作成し，そこに gym などのパッケージをインストールする．以下で必要なパッケージは全てインストールされる．

環境：Intel Mac, MacOS ver.12.6 Monteley で確認済み．

```
conda create -n gym python=3.6 tensorflow jupyterlab
conda activate gym
conda install matplotlib ffmpeg tqdm
conda install pytorch torchvision -c pytorch
pip install 'gym<0.21, >=0.16'
pip install pyglet < 2.0.0
pip install box2d-py
pip install pybullet
```

環境: windows 10 Home バージョン 21H2 19044.1766, Anaconda Navigator 2.3.0 で確認済み．注意事項：`box2d-py` は c++ が入っていない場合は，以下のインス

トーラからインストールする必要がある．https://visualstudio.microsoft.com/visual-cpp-build-tools/

```
conda create -n gym python=3.6 tensorflow
conda activate gym
conda install matplotlib ffmpeg tqdm
conda install gym -c conda-forge
conda install pytorch torchvision -c pytorch
conda install swig
pip install box2d-py
pip install pybullet
```

以下のコードを実行し，2足歩行ロボットのアニメーションが流れれば問題なくインストールできている．なお，pybullet は余力のある場合にのみ利用する可能性があるパッケージであるため，無理にインストールする必要はない．

```
import gym
env = gym.make('BipedalWalker-v3')
state = env.reset()
for t in range(1000):
    env.render()
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)
    if done:
        break
env.close()
```

```
import gym
import pybullet_envs
env = gym.make('HumanoidBulletEnv-v0')
env.render(mode="human")
state = env.reset()
for t in range(1000):
    env.render()
    action = env.action_space.sample()
    state, reward, done, info = env.step(action)
    if done:
        break
env.close()
```

注意: 以降，コマンドプロンプトを開き直すたびに，conda activate gym によって仮想環境 gym をアクティベートする必要がある．

OpenAI gym の提供する強化学習環境のインターフェースは以下のリンク先の，Environments，Observations，Spaces にて説明されているので，よく理解しておくこと．https://gym.openai.com/docs/

以下の実装では，環境として Pendulum-v0 を想定する．これは，状態空間も行動空間も矩形領域 $\mathcal{S} = [s^1_{\min}, s^1_{\max}] \times \cdots \times [s^n_{\min}, s^n_{\max}]$，$\mathcal{A} = [a^1_{\min}, a^1_{\max}] \times \cdots \times [a^m_{\min}, a^m_{\max}]$ で表される環境である．この点を仮定して実装してよい．その他，課題の進捗状況によっては BipedalWalker-v3 や HumanoidBulletEnv-v0 を利用する場合もある．後者を利用する場合には，pybullet_envs をインポートする必要があることに注意．

4.1.2 学習エージェントのインターフェース

学習エージェントのインターフェースは以下のようにする．以下に実装するエージェントはいずれもこの Agent クラスを継承するものとする．

```python
class Agent:
    def save_models(self, path):
        pass
    def load_models(self, path):
        pass
    def select_action(self, state):
        pass
    def select_exploratory_action(self, state):
        pass
    def train(self, state, action, next_state, reward, done):
        pass
```

それぞれのメソッドは以下の通り：

- `save_models`, `load_models`：エージェントの内部パラメータ（テーブル Q 学習であれば Q テーブル，Actor-Critic 法であれば Actor ネットワークと Critic ネットワークのパラメータなど）を `path` に保存，`path` から読み込み．学習後に学習途中の振る舞いを見るため．
- `select_action`：$\pi(\texttt{state})$ を返す．
- `select_exploratory_action`：$a \sim \beta(a; \texttt{state})$ を返す．
- `train`：学習する．

このインターフェースを使った最小限の学習スクリプトは以下の通り．学習過程における振る舞いを見たければループの頭に `env.render()` を追記すればよい．ただし，実行時間が非常に長くなるため，実際に学習する際にはつけないことが推奨される．`tqdm` はプログレスバーを表示する機能．

Listing 1: 実行スクリプト

```python
from tqdm import tqdm
import gym
env = gym.make('Pendulum-v0')
agent = Agent()
state = env.reset()
for t in tqdm(range(1000)):
    action = agent.select_exploratory_action(state)
    next_state, reward, done, info = env.step(action)
    agent.train(state, action, next_state, reward, done)
    state = next_state
    if done:
        state = env.reset()
env.close()
```

4.1.3 ランダム行動エージェントの実装

上のスクリプトが動くように，ランダムな行動を取り続けるエージェントを `Agent` クラスを継承して実装する．ターゲット方策 π は行動の取りうる範囲の中央値，行動方策 β は行動の取りうる範囲の一様分布とする．`save_models`, `load_models`, `train` の実装は不要．必要に応じて `__init__` を実装すること．実装が終われば，Listing 1は 4 行目 (`Agent` の初期化) を修正するだけで実行できるはずである．

確認事項　`env.render()` をつけて複数回実行し，実行毎に異なる振る舞いをしていることを確認する．

4.1.4 再現性のある実行スクリプトの実装

前節で実装したランダム行動エージェントの実行結果は，実行の度に異なる．原因として，

1. エージェントの取る行動がランダムである
2. 環境の初期状態や状態遷移が確率的である

ことが挙げられる．実行毎に毎回異なる結果になるのは自然であり，本来そうならなくては困る．しかし，ある典型的な結果を示したシミュレーション結果を再現することができない，不自然な振る舞いをしたためバグが疑われるがそれを再現できないため確認が困難である，実験結果の証拠が残せない，などといろいろ都合が悪い．

そこで，疑似乱数の系列を再現できるようにしておくことが望ましい．Python パッケージである `random` や `numpy.random` を利用している場合には，これらの乱数の種をセットできるようにしておく．これにより，種を変えれば異なる結果がえられ，変えなければ同じ結果を再現できる．`gym` は環境の内部で独自の疑似乱数を利用しているので，これも制御する必要がある．また，ニューラルネットワークを実装する際に，`pytorch` などのパッケージを利用する場合には，それらが利用する乱数の種も制御できるようにしておく必要がある．

上に出てきた疑似乱数の種は，以下のようにして設定できる．

Listing 2: 実行スクリプト

```
import random
import numpy as np
import torch
import gym

seed = 2
env = gym.make('Pendulum-v0')

env.seed(seed)
torch.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)
```

確認事項　ランダム行動エージェントを実装する際に利用した疑似乱数生成器や `gym` の乱数生成器の `seed` を適当な値に固定し，`env.render()` をつけて複数回実行し，毎回同じ振る舞いをしていることを確認する．

4.1.5 エージェント評価用関数の実装

エージェントのターゲット方策 π と行動方策 β は異なるため，学習時のエージェントの振る舞いを見てもそのエージェントが π をうまく学習できているかどうか評価できない．そこで，エージェントの方策を評価するための関数を実装しておく．

エージェントの方策の評価は，次のように実施する．

- 環境は学習に利用しているものとは独立に作成し，異なる乱数の種を利用する．一方で，評価の度に異なる乱数系列であると評価が確率的に揺らぐので，毎回同じ乱数系列を利用する．
- 方策 π を用いて環境と 1 エピソード分（T ステップ）インタラクションし，累積報酬 $R = \sum_{t=1}^{T} r_t$ を計算する．
- 方策は決定的であるが，環境の初期値や遷移は確率的であるため，累積報酬を複数エピソード分計算し，返す．

確認事項　ランダム行動エージェントを上で実装した評価関数に渡し，10 エピソード分の累積報酬を出力する．これらにばらつきがあることを確認する．また，これを複数回呼び出し，毎回同じ結果が返ってくることを確認する．

4.1.6　ログと可視化

学習過程におけるエージェントの方策を評価するために，学習途中でのエージェントの状態を `save_models` を用いて数ステップに一度保存しておき，あとから `load_models` を用いて読み込めるようにしておく．ファイル名の末尾に使用した乱数の種とステップ数を加えるなど，あとから処理しやすいようにしておく．

学習過程を学習中および学習後に確認するために,保存されたデータを読み込み,評価関数を用いて累積報酬を計算し, 結果を箱ひげ図として作図するスクリプトを作成する. 例えば, 学習を 5 回繰り返し, それぞれについて 10 エピソード分の累積報酬を計算すれば, 50 個のデータが得られるので,これについて箱ひげ図を作図する. 図の縦軸は累積報酬, 横軸は学習ステップ数とする.箱ひげ図の作図には,`maplotlib` の `boxplot` を用いれば良い.(https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.boxplot.html) 箱ひげ図の読み方については理解しておくこと.

データ数が多い場合（学習ステップ数が多い場合や保存頻度が高い場合など）や複数の学習方法を比較する場合には箱ひげ図は適さない．その場合，箱ひげ図に表されている統計量のうち，25%（第一四分位），50%（第二四分位，中央値），75%（第三四分位）に着目し，25%-75% 領域を半透明の帯，50% 点を線でプロットするなどの工夫が必要である．

確認事項　ランダム行動エージェントを学習させ（実際には何も学習しないのだが），途中経過を上述のスクリプトで可視化する．縦軸が累積報酬，横軸が学習ステップ数になっていること，学習曲線がほとんど横ばい（学習していないので）になっていることを確認する．

4.2　テーブル Q 学習の実装

前節にて実装したランダム行動エージェントをもとに，テーブル Q 学習エージェント（方法 1（3.1節）および方法 2（3.2節））を実装する．テーブル Q 学習では，方策 $\pi(s)$ は状態 s のもとで行動価値が最大になる行動 $a = \mathrm{argmax}_a Q(s,a)$ を取る決定的方策となる．一方，行動方策 β には自由度がある．本実験では，行動方策として ϵ-greedy 方策を利用する：

$$\beta(s) = \begin{cases} \pi(s) & \text{with probability } 1-\epsilon \\ a_i & \text{with probability } \epsilon/L \text{ for } i = 1, \ldots, L \end{cases}$$

状態空間と行動空間の分割方法にも設計の自由度がある．本実験では，状態の各次元も行動の各次元も閉区間で表される問題を扱うため，状態の各次元を k 等分割（$K = k^n$），行動の各次元を ℓ 等分割（$L = \ell^m$）する．Q テーブルのサイズは $K \times L = k^n \times \ell^m$ となる．また，行動空間の各部分集合の代表点 a_i は，領域の中心点とする．

4.2.1　方法 1 の実装

状態および行動の分割数をコンストラクタで設定できるようにしたテーブル Q 学習エージェントを実装する．ただし，Q テーブルの初期値は $10^{-8} \times \mathcal{N}(0,1)$ などとすること．

4.2.2　経験再生バッファーの実装

方法 2，3，4 では共通して経験再生バッファーを利用するため，いずれの方法でも利用できる再生バッファーを実装する．経験再生バッファーは FIFO とし，保存する最大サイズを `buffer_size` とする．最低限必要な機能は以下の通り．

```
class ReplayBuffer:
    def add(self, state, action, next_state, reward, done):
        pass
    def sample(self, batch_size):
        pass
```

`sample` は保存された履歴からランダムに選択された `batch_size` 個の履歴を取り出し，`state` の配列，`action` の配列，`next_state` の配列，`reward` の配列，`done` の配列を返すものとする．

4.2.3 方法２の実装

上で実装した経験再生バッファーを用いて方法２（3.2節）を実装する．

4.3 Actor-Critic の実装

本実験では，actor と critic の両者にニューラルネットワークを用いた Actor-Critic 法を実装する．

行動方策としては，以下のように $\pi(s)$ を中心に正規分布に従って行動を生成する方策を採用する：

$$\beta(s) = \pi(s) + \mathcal{N}(0, D^2) \tag{6}$$

ここで，D は対角行列であり，各対角要素は次元毎のノイズの標準偏差を表し，各行動次元の区間の長さ $a^i_{\max} - a^i_{\min}$ を $\sigma_\beta/2$ 倍したものである．ただし，行動の取りうる区間を違反しないように，はみ出した値は境界上に射影する．また，探索初期に多様な行動を取りうるように，T_{expl} ステップ経過するまでは，取りうる行動から一様ランダムな行動を取るものとする．

4.3.1 Actor ネットワークと Critic ネットワークの実装

ニューラルネットワークは以下のような処理を繰り返すことで非線形の入出力関係を表現する関数モデルである．本実験で扱うニューラルネットワークは，$h_0 = x$ をネットワークへの入力とし，$i = 1, \ldots, H$ について中間出力を

$$h_i = g_i(b_i + W_i \cdot h_{i-1}) \tag{7}$$

とし，$y = h_H$ を出力として返す．ここで，W_i は実行列，b_i は実ベクトル，g_i は何らかの非線形関数であり**活性化関数**と呼ばれる．本実験では，g_i（$i = 1, \ldots, H-1$）は **ReLU** と呼ばれる関数とする．ReLU は，各次元において ReLU への入力と 0 の最大値を取る関数である．最後の活性化関数 g_H は，g_H への入力の各次元に対して

$$x \mapsto \frac{a^i_{\max} + a^i_{\min}}{2} + \frac{a^i_{\max} - a^i_{\min}}{2} \tanh(x) \tag{8}$$

とし，行動の取りうる範囲に収まるようにしたものである．中間出力 h_i（$i = 1, \ldots, H-1$）の次元や H は自由に決めることができるが，本実験では $H = 2, 3$ の場合のみ扱う．Actor については入力は状態と同じ n 次元，出力は行動と同じ m 次元である．Critic の入力は $n + m$ 次元，出力は 1 次元である．

ニューラルネットワークの実装においては，原則 `pytorch` を用いること．それ以外のパッケージを利用した場合，バグなどで躓いている場合に補助することが困難であることを理解されたい．これらは，Actor-Critic の学習において必要な，ニューラルネットワークのパラメータ（W_i や b_i）での関数の微分を自動的に計算してくれる．また，勾配を利用した最適化法についても実装されている．

4.3.2 方法３の実装

実装した Actor ネットワークと Critic ネットワークを用いて，方法３を実装する．勾配法のステップにおいては，厳密に3.3節や3.4節にかかれている ω や θ の更新式に従わなくても良い．利用したパッケージに実装されている勾配ベースの最適化法（adam など）を利用すればよい．

4.3.3 方法４の実装

方法３をベースに，3.4節に紹介した４つの工夫を実装し，方法４を実装する．

5 レポート課題

本実験では，中間レポートと最終レポートを提出することになる．各レポートにおいて，4.2節と4.3節の内容を実装し，これを用いた数値実験を行い，結果を考察する．

重要 数値実験を行う際は，まず，実験の目的を明確にすること．そのうえで，実験の目的を実現するために必要な実験方法を計画すること．実験方法を計画する際には，実験後にどのようなデータを比較・考察するのかについても熟考しておくこと．とりわけ，乱数を用いた数値実験を行う場合には，得られたデータが偶然得られた特異なデータでないことを主張するのに足る統計的議論が必要になる．（複数回同様の実験をする必要がある．）4.1.5節や4.1.6節で実装した評価および可視化ツールを上手く活用すること．

5.1 中間レポート課題：テーブル Q 学習

以下の課題について，レポートをまとめよ．本資料の繰り返しになったとしても，実験目的と実験方法を必ず記載すること．実験結果については，作図したものを貼り付けるとともに，実験の目的を照らし合わせながら考察せよ．実験の結論は，仮説が受容／棄却されたか，もしくは実験設定が不適切であるため結論が出せないか，である．結論が出せない場合，どのように実験設定を変更すべきか考察せよ．必要に応じて追加のデータを示すこと．

課題１ 4.2節の内容（方法 1 と方法 2）を実装せよ．実装した方法 1 と方法 2 および `Pendulum-v0` 環境での実行スクリプトを提出せよ．コードの該当行数を参照しつつ，アルゴリズムについて解説すること．

課題２ 方法２は方法１のサンプル効率を改善するために経験再生バッファーを導入している．その効果を数値実験により確認せよ．

実験目的 今回の数値実験の目的は，`Pendulum-v0` 環境において経験再生バッファーの利用によりテーブル Q 学習のサンプル効率が改善するという**仮説を検証**することである．

実験設定 状態空間と行動空間の各次元の分割数を $k = 10$，$\ell = 9$ などとし，それぞれの設定において異なる乱数の種を用いて 5 回独立に学習する．減衰率は $\gamma = 0.99$ とする．設定の推奨値は，$\alpha = 3 \times 10^{-4}$，$\epsilon = 0.05$，`batch_size` $= 256$，バッファーサイズは最大ステップ数と同じ，最大ステップ数は 5×10^5，とするが実験の趣旨に合う限り変更してもよい．それ以外の設定については，実験の趣旨に合わせて設計せよ．

注意 5×10^5 ステップの実行には，実装方法にもよるが数時間かかる可能性がある．実験スクリプトやアルゴリズムにバグが無いことを十分に確認したうえで，まずは短いステップ数で実験してみるのが良い．問題なく動いている（問題なく学習できると

は限らない）ようであれば，最大ステップ数を増加し，実験せよ．可能ならば，乱数を変えて複数試行並列実行するのが効率良い．

課題３（余力があれば） 方法１や方法２について，自由に仮説を設定し，数値実験により確認せよ．実験の目的（仮説）を立て，これを検証するような実験設定を考え，実験を実施せよ．

例えば，方法２における K や L の学習結果や学習速度に与える影響などに着目する．テーブル Q 学習において，状態や行動の分割数は表現できる方策の自由度や学習しなければならないパラメータ数に直結するため，学習結果や学習速度に影響を与えることが予想される．これについて，仮説を設けて数字実験を実施する．

5.2 最終レポート課題：Actor-Critic

以下の課題について，レポートをまとめよ．実験目的と実験方法は必ず記載すること．実験結果については，作図したものを貼り付けるとともに，実験の目的を照らし合わせながら考察せよ．実験の結論は，仮説が受容／棄却されたか，もしくは実験設定が不適切であるため結論が出せないか，である．結論が出せない場合，どのように実験設定を変更すべきか考察せよ．必要に応じて追加のデータを示すこと．

課題４ 4.3節の内容（方法３と方法４）を実装せよ．実装した方法３と方法４および `Pendulum-v0` 環境での実行スクリプトを提出せよ．コードの該当行数を参照しつつ，アルゴリズムについて解説すること．

課題５ 方法４は方法３に４つの工夫を加えている．それぞれの効果を検証したいが，複数の工夫の相乗効果を考慮すると全部で $2^4 = 16$ 通りの方法を検証することになり，時間がかかる．そのような場合，各コンポーネントだけを取り除いた方法（４つ）と全ての構成要素を加えた方法（１つ）を比較することで，いずれの要素も重要であることを示すアブレーションテスト（ablation test, ablation study）がしばしば用いられる．方法３と方法４の間の差異について，アブレーションテストを実施し，考察せよ．

実験目的 `Pendulum-v0` 環境において，方法４で導入された４つの工夫がいずれも学習の安定性に寄与するという仮説を検証すること

実験設定 Actor および Critic ネットワークの層数は $H = 2$ および $H = 3$ とし，h_1 および h_2（$H = 3$ の場合）の次元数は 256 とし，それぞれの設定において異なる乱数の種を用いて 5 回独立に学習する．減衰率は $\gamma = 0.99$ とする．その他のパラメータの推奨値は，以下の通りとするが，実験の趣旨に合う限り変更して良い．行動方策のノイズ $\sigma_\beta = 0.1$，学習率 $\alpha = 3 \times 10^{-4}$，ランダム行動ステップ数 $T_{\text{expl}} = 10^4$，`batch_size` $= 256$，バッファーサイズは最大ステップ数，最大ステップ数は 10^5，方法４については $\tau = 0.005$，$\sigma_{\text{target}} = 0.2$，$c = 0.5$，$d = 2$ とする．それ以外の設定については，実験の趣旨に合わせて設計せよ．

課題６（余力があれば） `BipedalWalker-v3`（2D）もしくは `HumanoidBulletEnv-v0`（3D）環境において，方法３や方法４に関して自由に仮説を立て，検証せよ．

References

[1] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1582–1591, 2018.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[4] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395, 2014.

[5] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[6] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[7] 久保隆宏. *Python* で学ぶ強化学習. 機械学習スタートアップシリーズ. 講談社, 2019.

[8] 森. 哲郎. 強化学習. 機械学習プロフェッショナルシリーズ. 講談社, 2019.