

## 1 Deep Q Network の擬似コード

以下, Deep Q Network の擬似コード.

---

**Algorithm 1** Deep Q-learning with experience replay

---

**Input:**  $r_{\text{learn}}, n_{\text{rmem}}, \epsilon_{\text{init}}, \epsilon_{\text{final}}, l_{\text{expl}}, n_{\text{rpstart}}, f_{\text{learn}}, f_{\text{update}}, \gamma, n_{\text{batch}}, T$

```
1: Replay Memory  $D$  を初期化
2: Q-Network  $Q$  をランダムな重み  $\theta$  で初期化
3: Target network  $Q^-$  を重み  $\theta^- = \theta$  で初期化
4:  $t = 1$ 
5: while  $t < T$  do
6:   while not done do
7:      $\epsilon$ -greedy に従って行動  $a_t$  を選択
8:      $\epsilon = \max(\epsilon_{\text{final}}, \epsilon - \frac{\epsilon_{\text{init}} - \epsilon_{\text{final}}}{l_{\text{expl}}}) \rightarrow \epsilon$  を線形減少
9:     行動  $a_t$  を実行し, 報酬  $r_t$  と次の画面  $x_{t+1}$  と done を観測
10:    前処理して次の状態  $s_{t+1}$  を生成
11:     $D$  に  $(s_t, a_t, r_t, s_{t+1}, \text{done})$  を追加,  $|D| > n_{\text{mem}}$  なら古いものを削除する.
12:    if  $t > n_{\text{rpstart}}$  then
13:      if  $(t-1)\%f_{\text{learn}} = 0$  then
14:         $D$  からランダムに  $(s_j, a_j, r_j, s_{j+1}, \text{done})$  を  $n_{\text{batch}}$  個の履歴をサンプル
15:        
$$y_j = \begin{cases} r_j & (\text{done}) \\ r_j + \gamma \max_{a'} Q^-(s_{j+1}, a'; \theta^-) & (\text{otherwise}) \end{cases}$$

16:         $\theta$  を  $y_j - Q(s_j, a_j; \theta)$  を最小化する方向に学習率  $r_{\text{learn}}$  で更新 (損失関数には Huber 損失関数を用いる.)
17:      end if
18:      if  $(t-1)\%f_{\text{update}} = 0$  then
19:         $Q^- = Q$ 
20:      end if
21:    end if
22:     $t = t + 1$ 
23:  end while
24: end while
```

---

## 2 Atari Games の環境のラップ処理

Deep Q Network では Atari Games から受け取った状態の前処理などを行うことで学習をしやすいしている. 以下にそれらの処理を示す. なお,  $\text{reset}_{\text{hoge}}()$  は環境をリセットするときに呼ぶメソッドを,  $\text{step}_{\text{hoge}}(a_t)$  は行動  $a_t$  を取って環境を更新するときに呼ぶメソッドを,  $\text{observe}_{\text{hoge}}()$  は環境の状態を返すときに呼ばれるメソッドを,  $\text{reward}_{\text{hoge}}()$  は報酬を返すときに呼ばれるメソッドを表す.

---

**Algorithm 2**  $reset_{noop}()$ 

---

**Input:**  $env, l_{nomax}$ 

- 1: エピソードの開始時に、数フレーム何もしない行動を取り、初期状態を決定する.
- 2:  $T \sim U(1, l_{nomax})$
- 3: **for**  $t' = 1, \dots, T$  **do**
- 4:    $a_{t'} = (\text{do nothing})$  の実行
- 5: **end for**

**Output:** 初期状態が決定した環境  $env$ 

---

---

**Algorithm 3**  $step_{repeat}(a_t)$ 

---

**Input:**  $env, l_{repeat}, a_t$ 

- 1: 1 回行動を取ると、同じ行動を指定フレーム続ける. 指定数分行動を繰り返したら、直前のフレームの観測との最大値を状態として返す.
- 2: ※  $a_t$  は選択したい行動とする.
- 3:  $r_{total} = 0$
- 4: **for**  $t' = 1, \dots, l_{repeat}$  **do**
- 5:    $s_{prev} = s_{t'}$
- 6:    $a_{t'} = a_t$  として行動を選択し、環境  $env$  を更新、 $(s_{t'+1}, r_{t'}, done_{t'})$  を観測する.
- 7:    $r_{total} = r_{total} + r_{t'}$
- 8:    $s_{max} = \max(s_{prev}, s_{t'+1})$
- 9:    $done = done_{t'}$
- 10: **end for**

**Output:**  $s_{max}, r_{total}, done$ 

---

---

**Algorithm 4**  $observe_{gray84}()$ 

---

**Input:**  $s_t$ 

- 1: 観測した画面を (84,84) サイズのグレースケール画像に変換して返す.
- 2:  $s_t$  をグレースケール画像に変換
- 3: 変換後の  $s_t$  をさらに (84,84) に reshape

**Output:** 変換後の  $s_t$ 

---

---

**Algorithm 5**  $step_{stack}(a_t)$ 

---

**Input:**  $env, l_{history}, a_t, S$ 

- 1:  $l_{history}$  ステップ数分の観測の履歴を状態として返す.
- 2:  $S$  は観測の履歴とする.
- 3:  $a_t$  を行動として選択し、環境  $env$  を更新、 $(s_{t+1}, r_t, done_t)$  を観測する.
- 4:  $S$  に  $s_{t+1}$  を追加.  $|S| > l_{history}$  なら、一番古い履歴を  $S$  から削除する.

**Output:**  $S, r_t, done_t$ 

---

---

**Algorithm 6**  $\text{reward}_{\text{clip}}()$ 

---

**Input:**  $r_t$ 1: 報酬  $r_t$  が正なら +1 に, 負なら -1 に, 0 なら 0 として返す.2: **if**  $r_t > 0$  **then**3:    $r_t = 1$ 4: **else if**  $r_t < 0$  **then**5:    $r_t = -1$ 6: **else**7:    $r_t = 0$ 8: **end if****Output:**  $r_t$ 

---