自主課題研究 最終レポート

RNN を用いた POMDP 環境での倒立振り子問題

金沢大学 理工学域
電子情報通信学類 3 年

清水 翔仁

令和 2 年 2 月 21 日

1章　はじめに

強化学習は，試行錯誤から学習する点で様々な問題に適用しやすいため，近年盛んに研究されている．DQN(Deep Q-Network) は，Q 学習にディープラーニングを用いた強化学習のモデルで，ゲーム画面を入力した Atari 2600 の様々なゲームに対して人間より良い性能を実現した．しかし，DQN の入力はエージェントが観察した最近の 4 つのフレームのみを用いるため，意味がある行動をするために 4 ステップより昔の情報を必要とするような環境に対しては性能が低下する可能性がある．この欠点を解決するために DRQN(Deep Recurrent Q-Network) が提案されている．DRQN は，DQN に循環神経網の一種である LSTM を組み合わせることで過去の情報を扱う．さらに LSTM はより長期的な情報を考慮することで，現実で起こりうる不完全な情報にも対応できるため，POMDP(Partially Observable Markov Decision Process) の環境でより効果を発揮することができる．

しかし，DRQN は学習の時に LSTM の初期状態をゼロベクトルで初期化するため，学習時に用いた情報のタイムステップより長い期間に対して学習することが難しいという欠点がある．そこで本研究では，LSTM の初期状態を与える方法を複数用意し，それぞれの実行結果についての考察を行う．

2 章　原理

2.1　POMDP(Partially Observable Markov Decision Process)

状態遷移が確率的に生じる動的システムの確率モデルを MDP という．遷移する過程において，将来状態の条件付き確率分布が，現在状態のみに依存し，過去のいかなる状態にも依存しない．(マルコフ性)

各時刻において過程 (process) はある状態 (state) を取り，意思を決定するエージェントはその状態において利用可能な行動 (action) を任意に選択する． その後過程はランダムに新しい状態へと遷移し，その際にエージェントは状態遷移に対応した報酬 (reward) を受けとる．よって MDP は以下の要素で構成される．

$$
\begin{aligned}
&\text{行動集合} \ : \mathcal{A} = \left\{a^{(1)}, a^{(2)}, \cdots, \right\} \\
&\text{状態集合} \ : S = \left\{s^{(1)}, s^{(2)}, \cdots, \right\} \\
&\text{遷移関数} \ : T_{ijk} = \Pr\left(s_{t+1} = s^{(j)} \mid s_t = s^{(i)}, a_t = a^{(k)}\right) \\
&\text{報酬関数} \ : r = g(s, a) \\
&\text{初期状態確率} \ : p_0 = \Pr\left(s_0\right)
\end{aligned}
\tag{2.1}
$$

POMDP はエージェントの状態観測に不確実性を付加させることにより MDP を拡張したものである．

2.2　Q 学習 (Q-learning)

Q 学習は強化学習における手法の一種である．Q 学習は MDP において全ての状態が十分にサンプリングできるようなエピソードを無限回試行した場合，最適な評価値に収束することがわかっている．Q 学習では実行するルールに対しそのルールの有効性を示す Q 値という値を持たせ，エージェントが行動するたびにその値を更新する．ここでいうルールとはある状態とその状態下においてエージェントが可能な行動を対にしたものである．例えばエージェントの現在の状態を s_t とし，この状態で可能な行動が a，b，c，d の 4 通りあるとする．このとき，エージェントは 4 つの Q 値，$Q(s_t, a)$，$Q(s_t, b)$，$Q(s_t, c)$，$Q(s_t, d)$ を元に行う行動を決定する．行動の決定方法は理論上では無限回数試行するならランダムでも Q 値は収束するが，現実には収束を早めるため，なるべく Q 値の大きな行動が高確率で選ばれるように行う．Q 値の計算式を式 (2.2) に示す．

$$
Q\left(s_t, a_t\right) = (1-\alpha) Q\left(s_t, a_t\right) + \alpha \left(r_{t+1} + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}\right)\right) \tag{2.2}
$$

ここで γ は割引率といい，将来の価値をどれだけ割り引いて考えるかのパラメータである．

2.3 DQN(Deep Q-Network)

Q 値を最大化させる関数である最適行動価値関数を，ニューラルネットワーク (NN) を使った近似関数で求める手法．状態を NN の入力にし，出力層の各ノードが，各行動の行動価値を出力するようにする．

強化学習において与えられるデータは時系列的に連続したものになっており，データ間に相関が出てしまうためバラバラにする必要がある．これを Experience Replay という．手法としては，まず一旦経験した状態/行動/報酬/遷移先をメモリーに蓄積し，学習を行う際はそこからランダムサンプリングして利用する．

2.4 RNN(Recurrent Neural Network)

RNN とは，Deep Learning の手法の一種であり，時系列データの分析に特化している．図 2.1 に，RNN の模式図を示す．ここで，x_t を時刻 t における RNN の入力，h_t を時刻 t における RNN の出力とする．

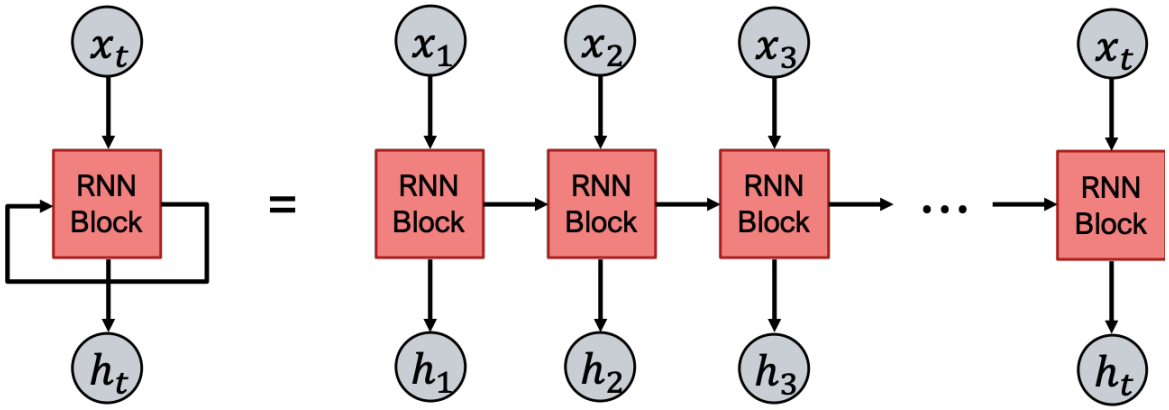

図 2.1: 展開された RNN

図 2.1 のように，RNN は内部にループ構造を持っている．これにより，前のステップでの分析結果を記憶し，データの時系列を理解することができる．しかし，RNN には長期依存性問題という欠点がある．長期依存性問題とは，記憶するステップ数が膨大になると計算が爆発するという問題である．そのため，現在単純な RNN はあまり使用されていない．

2.5 LSTM(Long Short Term Memory)

LSTM とは，RNN の長期依存性問題を解決した手法である．図 2.2 に RNN の内部を，図 2.3 に LSTM の内部を示す．このとき，σ は 0〜1 を出力する関数，tanh は-1〜1 を出力する関数とする．

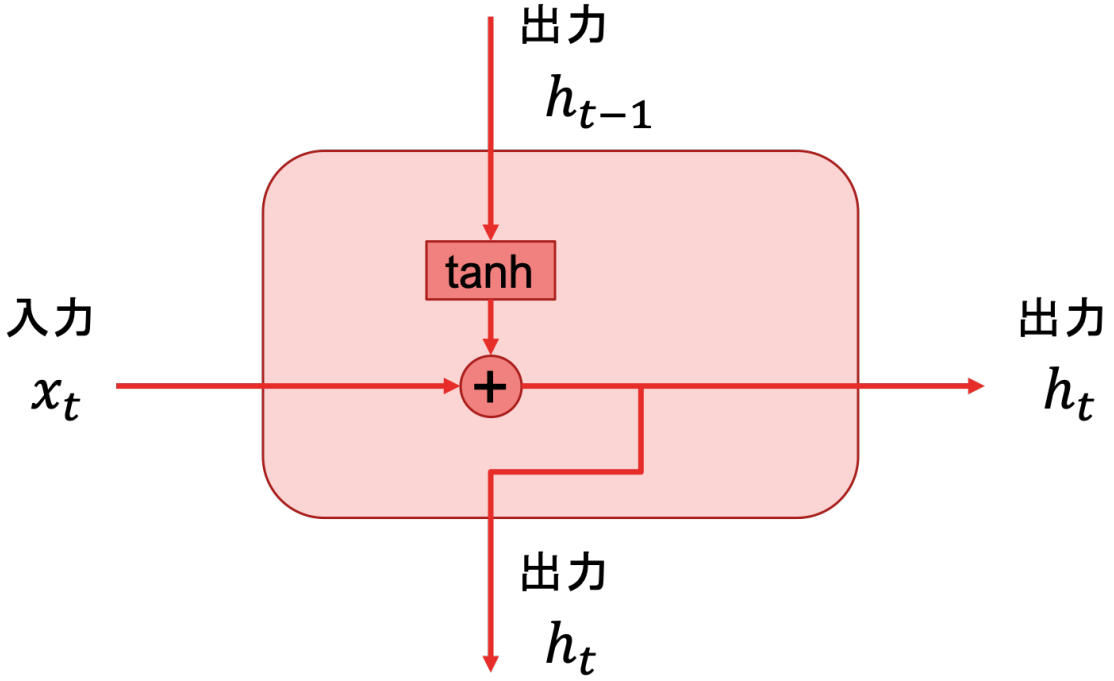

図 2.2: RNN の内部

図 2.2，図 2.3 より，RNN は単一の tanh 関数という非常に単純な構造に比べ，LSTM は 4 つの関数を含む複雑な構造をしている．

ここで，図 2.3 の LSTM 内にデータが流れる手順について説明する．

1. 前のステップからの出力 h_{t-1} と入力 x_t が合流する．合流した信号はコピーされて 4 つのラインに分岐する．
2. 一番上のラインの忘却ゲートでは，前のステップからの記憶一つ一つに対して，σ 関数からの 0〜1 の値によって情報の取捨選択を行う．このとき 1 は情報を全て残し，0 は全て捨てる．これにより，不要と思われる情報を捨てることで計算の爆発を防ぐ．
3. 入力ゲートにおいて，前のステップからの出力 h_{t-1} と入力 x_t の合算を長期保存用に変換した上で，どの信号をどのくらいの重みで記憶に保存するか制御する．これは 2 つの手順で処理する．

   (a) tanh 関数を用いて，入ってきた情報の情報量を削減し，必要な情報だけに変換された

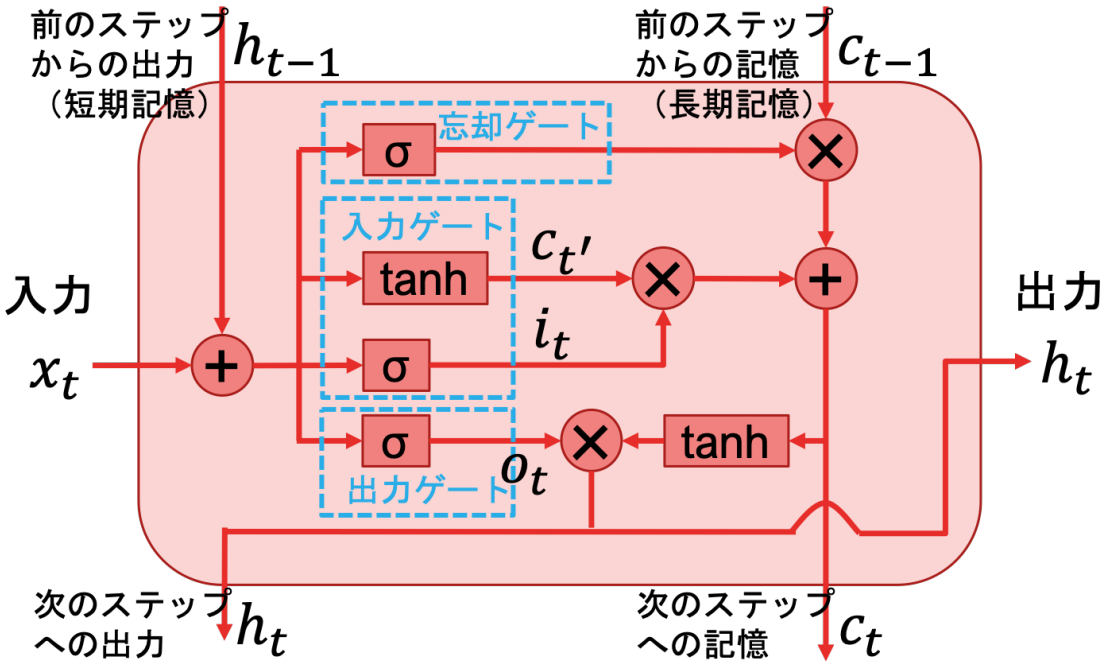

図 2.3: LSTM の内部

$c_{t'}$ が出力する．

(b) σ 関数の出力 i_t によって，h_{t-1} を考慮して入力 x_t の重みを調整する．

4. 出力ゲートにおいて，上記の処理で取捨選択された長期記憶 c_t の中で，短期記憶 h_t に関する部分のみを出力する．これも 2 つの手順で処理する．

(a) 前のステップからの記憶 c_{t-1} と，入力 x_t を変換した短期記憶 $c_{t'}$ を合算し，長期記憶 c_t として出力する．これは，それぞれ既に忘却ゲートおよび入力ゲートで取捨選択が行われている．

(b) tanh 関数に c_t を入力したものに対し，σ 関数からの 0〜1 の値 o_t によって情報の取捨選択を行う．

3 章　実験方法

本章では，開発環境や実験方法について説明する．

3.1　開発環境

本研究では Google Colaboratory によって提供された開発環境を用いて実験を行なった．また Colab 内部で動作している GPU は Tesla T4 である．

3.2　実験方法

今回，LSTM の初期状態を与える方法を検証するため，DQN,DRQN,DRQN-Stack,DRQN-Store-State,R2D2 の 5 つの手法を用いて実装されたプログラムをそれぞれ用意した．それぞれのプログラムの概要について説明を行う．

3.2.1　DQN

第 2.3 章において説明した DQN を用いたプログラム．このプログラムでは LSTM は使用していない．

3.2.2　DRQN

DRQN を用いたプログラム．DRQN とは DQN に LSTM を組み込んだ手法．時系列表現を獲得することで POMDP に対応でき，過去の情報を使って解くタスクに強い．

このプログラムではエピソードからランダムに軌跡を取り出し，LSTM の初期状態を zero 入力して学習を行う．これを Bootstrapped Random Updates という．シンプルかつランダムにサンプリングできるメリットはあるが，初期状態を zero 入力するため LSTM が適切な表現を獲得できない可能性がある．Bootstrapped Random Updates の模式図を図 3.1 に示す．

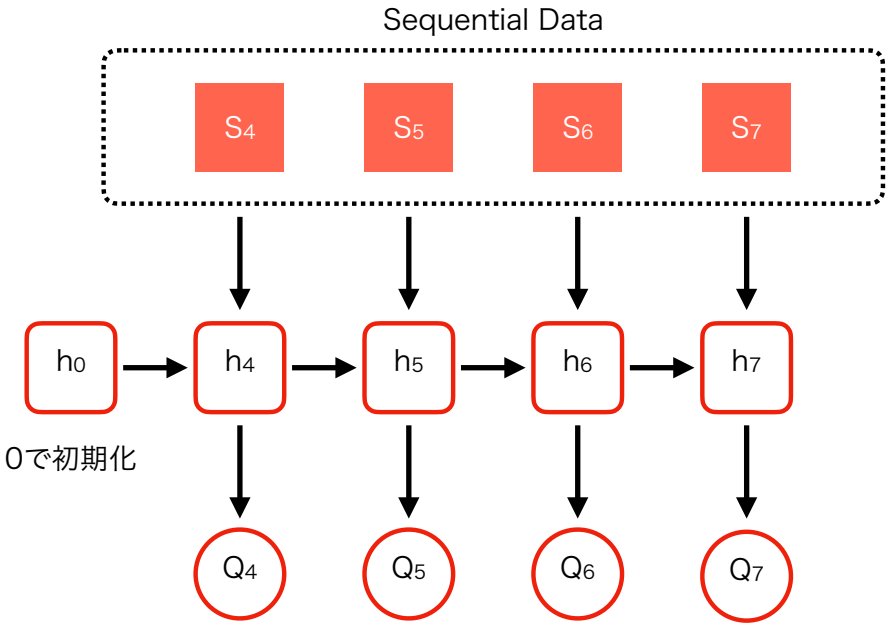

図 3.1: Bootstrapped Random Updates

3.2.3 DRQN-Stack

DRQN を用いたプログラム．このプログラムでは 1 エピソード全部のデータを使って学習を行う．これを Bootstrapped Sequential Updates という．LSTM 初期化で困ることはないが，バリアンスが大きい，エピソード長が可変などの問題がある．Bootstrapped Sequential Updates の模式図を図 3.2 に示す．

3.2.4 DRQN-Store-State

DRQN を用いたプログラム．このプログラムでは Bootstrapped Random Updates に加えて，Replay Buffer に経験 $e!$ とともにその時の LSTM の hidden state を貯めておく．これにより LSTM の初期状態問題を改善している．模式図を図 3.3 に示す．

3.2.5 R2D2

DRQN を用いたプログラム．このプログラムでは第 3.2.4 節で述べた Bootstrapped Random Updates と Replay Buffer の組み合わせに加えて，Burn-in という手法を採用している．これは最初は学習を行わずデータだけを流し今のネットワークの重みに慣れさせるという手法である．これにより正確な hidden state を復元することができる．模式図を図 3.4 に示す．

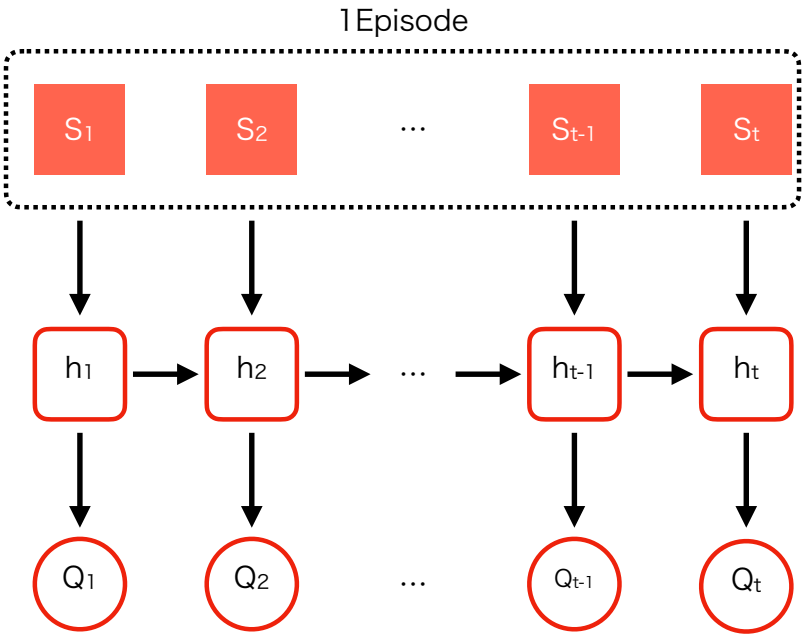

図 3.2: Bootstrapped Sequential Updates

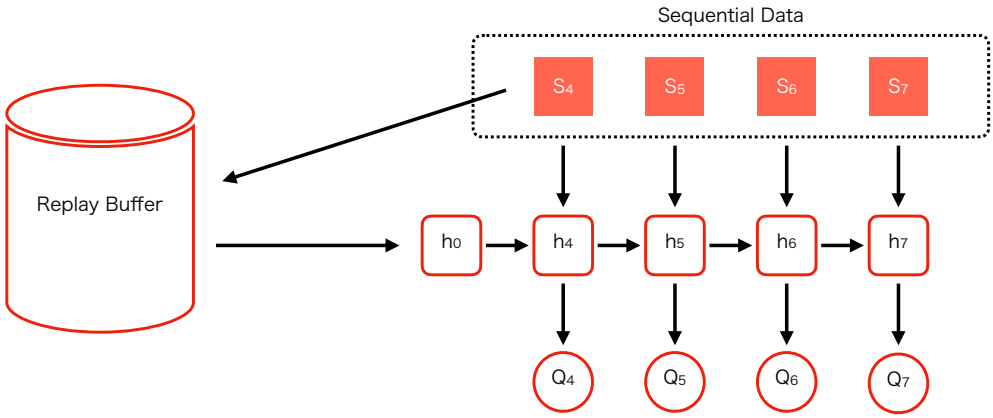

図 3.3: Bootstrapped Random Updates + Replay Buffer

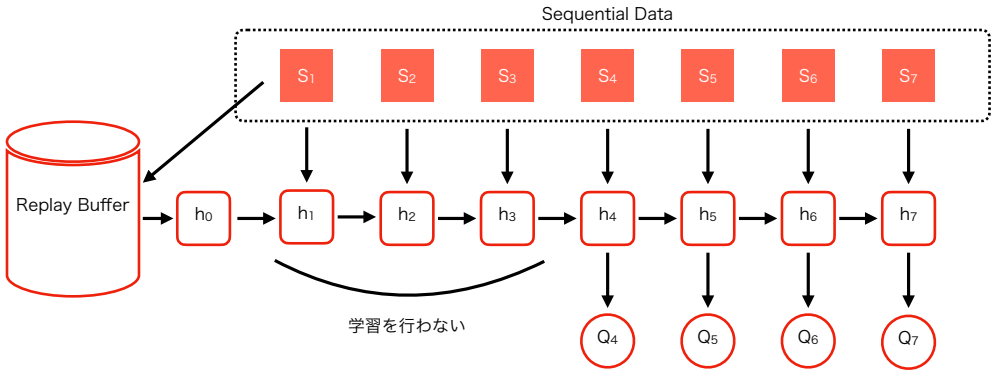

図 3.4: Bootstrapped Random Updates + Replay Buffer + Burn-in

4 章　結果

第 3.2 章で述べた 5 つのプログラムの実行結果を示す．

DQN の実行結果を以下に示す．これより，score は伸びず収束しないまま終了していることがわかる．

DQN の実行結果

```
0 episode | score: 30.00 | epsilon: 1.00
10 episode | score: 29.53 | epsilon: 1.00
20 episode | score: 28.56 | epsilon: 1.00
30 episode | score: 28.49 | epsilon: 1.00
40 episode | score: 27.54 | epsilon: 1.00
                    ...
29950 episode | score: 20.64 | epsilon: 0.10
29960 episode | score: 20.01 | epsilon: 0.10
29970 episode | score: 19.50 | epsilon: 0.10
29980 episode | score: 19.06 | epsilon: 0.10
29990 episode | score: 19.01 | epsilon: 0.10
```

DRQN の実行結果を以下に示す．これより，DQN と同様に score は伸びず収束しないまま終了していることがわかる．これは初期状態が zero 入力されたことによるものだと考えられる．

DRQN の実行結果

```
0 episode | score: 26.00 | epsilon: 1.00
10 episode | score: 25.24 | epsilon: 1.00
20 episode | score: 24.43 | epsilon: 1.00
30 episode | score: 23.89 | epsilon: 1.00
40 episode | score: 23.61 | epsilon: 1.00
                    ...
```

```
29950 episode | score: 18.53 | epsilon: 0.10
29960 episode | score: 19.06 | epsilon: 0.10
29970 episode | score: 18.99 | epsilon: 0.10
29980 episode | score: 19.30 | epsilon: 0.10
29990 episode | score: 19.10 | epsilon: 0.10
```

DRQN-Stack の実行結果を以下に示す．これより，DQN や DRQN とは違い score が 200 近くで収束していることがわかる．これは Bootstrapped Sequential Updates によって初期状態が与えられたことによるものだと考えられる．

DRQN-Stack の実行結果

```
0 episode | score: 26.00 | epsilon: 1.00
10 episode | score: 25.50 | epsilon: 1.00
20 episode | score: 26.68 | epsilon: 1.00
30 episode | score: 26.13 | epsilon: 1.00
40 episode | score: 25.58 | epsilon: 1.00
                    ...
700 episode | score: 162.81 | epsilon: 0.10
710 episode | score: 170.72 | epsilon: 0.10
720 episode | score: 180.14 | epsilon: 0.10
730 episode | score: 191.17 | epsilon: 0.10
740 episode | score: 199.45 | epsilon: 0.10
```

DRQN-Store-State の実行結果を以下に示す．これより，DRQN-Stack に比べ，収束までのステップ数が長いことがわかる．

DRQN-Stack の実行結果

```
0 episode | score: 23.00 | epsilon: 1.00
10 episode | score: 22.77 | epsilon: 1.00
20 episode | score: 22.79 | epsilon: 1.00
30 episode | score: 22.82 | epsilon: 1.00
40 episode | score: 22.67 | epsilon: 1.00
                    ...
820 episode | score: 148.15 | epsilon: 0.10
830 episode | score: 147.86 | epsilon: 0.10
```

```
840 episode | score: 158.67 | epsilon: 0.10
850 episode | score: 169.52 | epsilon: 0.10
860 episode | score: 192.27 | epsilon: 0.10
```

R2D2 の実行結果を以下に示す．これより，他のプログラムに比べ最も短いステップ数で収束していることがわかる．これは Burn-in によって hidden state の復元期間を設けることでより正確な学習ができていると考えられる．

DRQN-Stack の実行結果

```
0 episode | score: 10.00 | epsilon: 1.00
10 episode | score: 11.90 | epsilon: 1.00
20 episode | score: 12.74 | epsilon: 1.00
30 episode | score: 14.22 | epsilon: 1.00
40 episode | score: 15.35 | epsilon: 1.00
                    ...
490 episode | score: 140.98 | epsilon: 0.10
500 episode | score: 159.77 | epsilon: 0.10
510 episode | score: 179.90 | epsilon: 0.10
520 episode | score: 192.32 | epsilon: 0.10
530 episode | score: 200.57 | epsilon: 0.10
```

5 章　まとめ

本研究の目標である，LSTM の初期状態問題の検証を行うことができた．また今回使用した 5 つのプログラムの内 3 つのプログラムでは，結果が収束したため初期状態問題は改善されていると考えられる．特に R2-D2 においては他のプログラムに比べ 200 ステップ数ほど早い段階で収束しており，Bootstrapped RandomUpdates と Replay Buffer，Burn-in の組み合わせがうまく機能していると考えられる．