計測自動制御学会東北支部 第228回研究集会 (2006.5.17)
資料番号 228-1

1プレーヤーサッカーゲームにおける戦略学習

Strategy Learning in One-Player Soccer Games

○山田 知明*，西山 清*

○Tomoaki Yamada*, Kiyoshi NISHIYAMA*

*岩手大学工学部情報システム工学科

*Dep. of Computer & Infomation Science, Iwate University

キーワード： 強化学習(reinforcement learning), 1プレーヤーサッカーゲーム(one-player soccer game), 戦略学習(strategy learning), Q学習(Q-learning)

連絡先： 〒020-8551 盛岡市上田4-3-5 岩手大学 工学部 情報システム工学科
西山 清, Tel.: 019-621-6475，Fax.: 019-621-6475，E-mail: nisiyama@cis.iwate-u.ac.jp

1. はじめに

強化学習とは、試行錯誤を通じて環境に適応する学習制御の枠組であり、ネットワークセキュリティやネットワークロボットなどマルチエージェント学習に有効であることから近年急速に普及している。本研究では1プレーヤーサッカーゲームの戦略学習に強化学習を実装し、その性能を評価する。

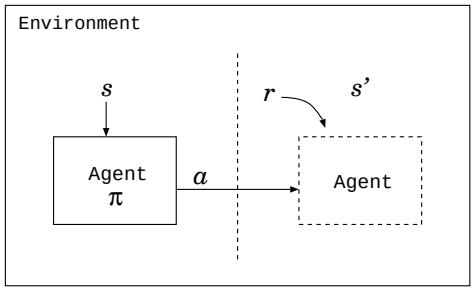

Fig. 1 環境とエージェントの相互作用

2. 強化学習

2.1 概要

強化学習では、まず学習する主体(エージェント)とそれをとりまく環境を定義する。エージェントはまず(1)環境から状態sを観測し、(2)状態sと政策πにより行動aを決定、(3)行動aを実行することで状態s'へ移行し、報酬rを得、(4)学習が終了するまで(1) ～ (3)のサイクルを繰り返す、という一連の環境との相互作用により最も効率の良い行動の仕方(戦略)を環境から学んでいく。(3)によってエージェントが得る報酬は、0や負の数、すなわちペナルティに相当する場合もある。

2.2 環境モデル

強化学習の多くは、環境モデルがマルコフ決定過程(MDP)であることを仮定している。環境がMDPであるということは将来の状態は、現在の状態とそのときとる行動にのみに依存し、過去の状態や行動の系列には依存しないことを意味する。また環境のとりうる状態の集合を$S = \{s_1, s_2, \cdots, s_n\}$、エージェントがとりうる行動の集合を$A = \{a_1, a_2, \cdots, a_m\}$と表す。環境中のある状態$s \in S$において、エージェントがある行動$a$を実行すると、環境は確率的に状態$s' \in S$へ遷移する。その遷移確率を$P^a(s, s')$により表す。このとき環境からエージェントへ報酬$r$が確率的に与えられるが、その期待値を$R^a(s, s')$

により表す。エージェントにおける状態集合から行動集合への確率分布を政策と呼び、πと表す。

2.3 Q-Learning

実環境においては環境モデルが予め与えられるとは限らず、状態遷移確率$P^a(s,s')$や報酬の与えられ方$R^a(s,s')$は通常未知の変数である。そこでなんらかの方法で「価値」を推定しなければならないが、Q-Learningは行動価値関数と呼ばれる$Q(s,a)$を用いて、行動学習を行うものである。ここで、$Q(s,a)$値とはある状態sのときに行動aをとる価値を表し、この値が大きければ価値も大きいということになる。また、式(1)は$Q(s,a)$の更新式である[1]。

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)) \quad (1)$$

ここで、αは学習率、γは割引率であり、s'は状態sで行動aをとったときの遷移先の状態を表す。

このQ-learningには次の収束定理が知られている[2]。

> エージェントの行動選択において、全ての行動を十分な回数選択し、かつ学習率αが$\sum_{t=0}^{\infty} \alpha(t) \to \infty$かつ$\sum_{t=0}^{\infty} \alpha(t)^2 < \infty$を満たす時間$t$の関数となっているとき、Q-learningのアルゴリズムで得るQ値は確率1で最適なQ値に収束する(概収束)。ただし、環境はエルゴート性を有する離散有限マルコフ決定過程であることを仮定する。

行動選択方法(探査戦略) 上記の収束定理は、全ての行動を十分な回数選択しさえすれば行動選択方法には依存せずに成り立つ。よって行動選択はランダムでもよい。しかし、強化学習ではまだQ値が収束していない学習の途中においてもなるべく多くの報酬を得るような行動選択を求められることが多い。学習に応じて序々に挙動を改善していくような行動選択方法として、1)ϵ-greedy 選択: ϵの確率でランダム、それ以外は最大のQ 値を持つ行動を選択、2) ボルツマン選択：$\exp(Q(s,a)/T)$に比例した割合で行動選択がある。ただし、Tは時間とともにゼロに近付く、などの方法が提案されている。本研究ではボルツマン選択(式(2))を用いる。

$$\pi(s,a) = \frac{e^{Q(s,a)/T}}{\sum_{b \in A} e^{Q(s,b)/T}} \quad (2)$$

3. サッカーゲームの戦略学習

3.1 ルール

今回、戦略学習するサッカーゲームはLittman[3]がシミュレーションに用いたサッカーゲームに引き分けのルールを加えたものを用いる。ルールの詳細は以下の通りである。

- 図2のような5 × 4マスのフィールドに2体のプレーヤー(PとC)が存在する。
- ボールは○で表され、プレーヤーのどちらかが必ずボールを所有する。
- ボールを所有した状態でそれぞれのゴール(Pは座標(4,1),(4,2)のどちらか、Cは座標(0,1),(0,2)のどちらか)の前にたてば勝ちである。
- プレーヤーは5つの行動(隣接する上下左右のマスに移動する、その場に立ち止まる)をとることができる。ただし、フィールドの外や他のプレーヤーがいるマスには移動できない。
- ボールを所有しているプレーヤーが他のプレーヤーのいるマスに移動しようとすると、ボールの所有は移る。逆にボールを持っていないプレーヤーが他のプレーヤーのいるマスに移動しようとした場合は何も起こらない。
- 30ステップ行動しても勝負がつかないときは引き分けとする。

次にゲームの流れであるが、まずゲームは初期位置(Pは(1,2), Cは(3,1))から始まり、ボールの所有はランダムに決定される。各時間ステップ毎にそれぞれのプレーヤーは行動を同期して決定する。どちらも行動が決定したら行動を実行するが、このときどちらが先に行動を実行するのかはランダムである。これをどちらかがゴールするまでつづけ、先にゴールした方に1点がはいる。勝負が決まったら再び初期位置に戻され、ボールの所有はランダムに決定される。

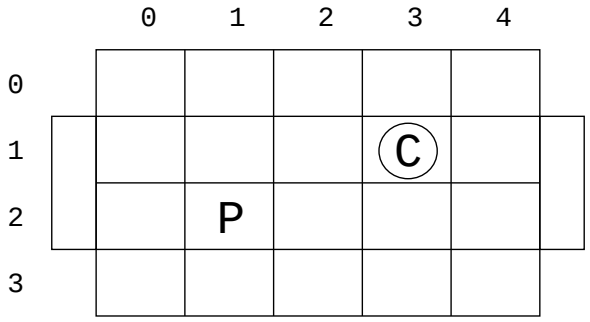

Fig. 2　サッカーゲームのフィールド

3.2　Q-Learningの適用

プレーヤーCをエージェントとする。環境の状態は、プレーヤーCがボール持っているかどうか(2通り)、プレーヤーCの位置(20通り)、プレーヤーPの位置(20通り)によって決定されるため、800($2 \times 20 \times 20$)通りの状態が存在する。ゲームが始まると、まずエージェントは現在の環境の状態を観測し、これをsとする。次にボルツマン選択法により得られる政策πから行動を決定し、これがaとなる。今回Pは学習を行わないため、ある一定の戦略から行動を決定する。こうして2体のプレーヤーの行動が決定された後、ランダムな順番によって行動を実行し、環境の状態に変化を与える。2体とも行動を実行したあとの環境の状態をs'とし、環境の変化によって報酬rを得る。以上のようにして得られたs, a, s', rと現在のQ値を用いて、Q値をより最適な値へと更新していくことでサッカーゲームの戦略を学習する。

4.　サッカーゲームの実装

3.節で説明したサッカーゲームの具体的な実装について説明する。開発はLinux上でC言語を用い、GUIの部分はXlibを用いている。図3にサッカーゲームのフローチャートを示す。

まず環境は図4で示されるようにそれぞれのプレーヤーの位置座標とボールの所有情報から構成される。

以下、図3のフローチャートに沿って具体的に説明する。

1) Q,πの初期化

Qとπを2次元配列とする。πは状態sのときのそれぞれの行動をとる確率を表すが、$\pi(s, a_0) \sim \pi(s, a_m)$の値の総和は1.0としている(すなわち、いずれか

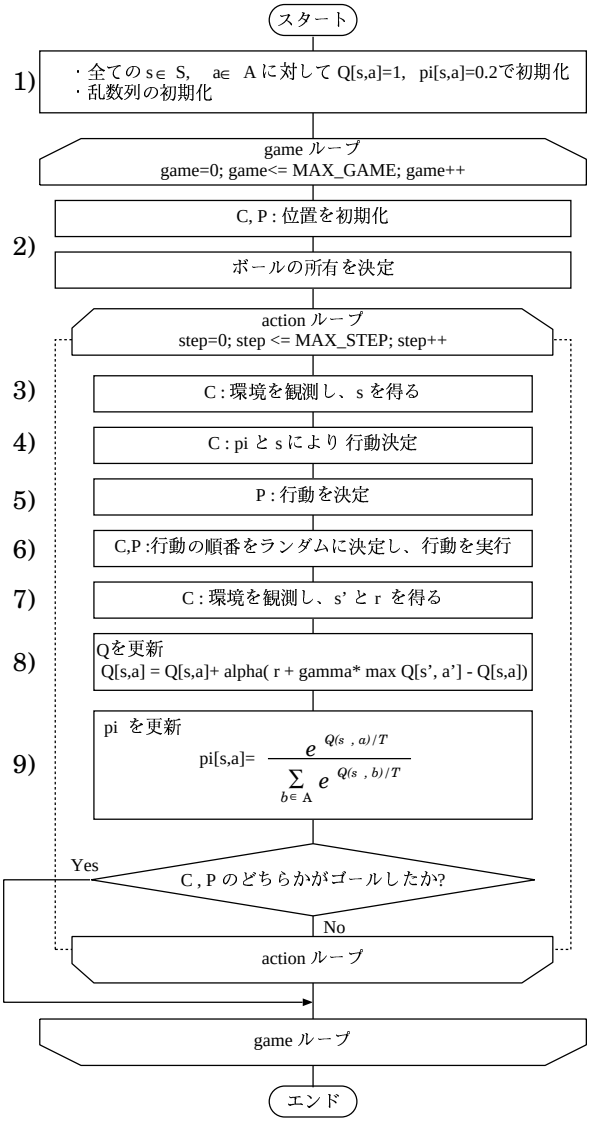

Fig. 3　サッカーゲームプログラムのフローチャート

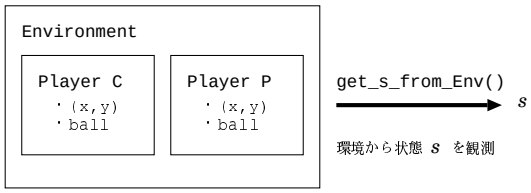

Fig. 4　環境から状態sを観測

の行動は確率1で必ず実行される)。

2) 環境を初期化

プレーヤーの位置を初期位置(Cは(3,1)、Pは(1,2))に設定し、ボールの所有も設定する。

3) 環境の観測と状態sの取得

プレーヤーCは環境を観測することで状態sを得るが、本研究では環境の状態に応じて表1のようにしてsを定義した。これを式で表すと、式(3)となる。

ボールの所有(C)	Pの座標	Cの座標	s
していない	(0,0)	(0,0)	0
していない	(0,0)	(1,0)	1
していない	(0,0)	(2,0)	2
していない	(0,0)	(3,0)	3
していない	(0,0)	(4,0)	4
していない	(0,0)	(0,1)	5
していない	(0,0)	(1,1)	6
していない	(0,0)	(2,1)	7
⋮	⋮	⋮	⋮
していない	(1,0)	(0,0)	20
していない	(1,0)	(1,0)	21
していない	(1,0)	(2,0)	21
⋮	⋮	⋮	⋮
している	(0,0)	(0,0)	400
している	(0,0)	(1,0)	401
している	(0,0)	(2,0)	402
⋮	⋮	⋮	⋮

Table 1　環境と状態sの対応

$$\begin{aligned} s &= C_{ball}F_W^2F_H^2 + (F_WP_y + P_x)F_WF_H \\ &\quad + F_WC_y + C_x \end{aligned} \tag{3}$$

ここでF_W、F_Hはそれぞれフィールドの幅と高さを表し(ここでは$F_W = 5$、$F_H = 4$)、C_{ball}はプレーヤーCがボールを所有しているときは1、所有していないときは0となる変数である。プログラムでは環境からsを求める関数get_s_from_Env()を以下のように定義することで実装した。

```
int get_s_from_Env( void )
{
    return Env[C].ball
     * FIELD_W*FIELD_W*FIELD_H*FIELD_H
     +(FIELD_W*Env[P].y+Env[P].x)
     *FIELD_W*FIELD_H
     + FIELD_W*Env[C].y+Env[C].x ;
}
```

4) πとsによる行動決定

πは図5のようなテーブルで表現することができる。ここでは例としてπの値が図5のようであったとする。これは行動a_0、a_1、a_2、a_3、a_4がそれぞれ20%、10%、10%、40%、20%の確率で選択されることを意味する。ここではまずπの値にそれぞれ1000を掛け図6のようにその値に比例した幅を持つ領域として考える。ここで1 ~ 1000の間の値をもつ整数rndを乱数により生成し、rndがどの領域に含まれているかでπの確率分布にしたがった行動選択を行う。

	a_0	a_1	a_2	a_3	a_4
⋮			⋮		
s	0.2	0.1	0.1	0.4	0.2
⋮			⋮		

Fig. 5　πテーブル

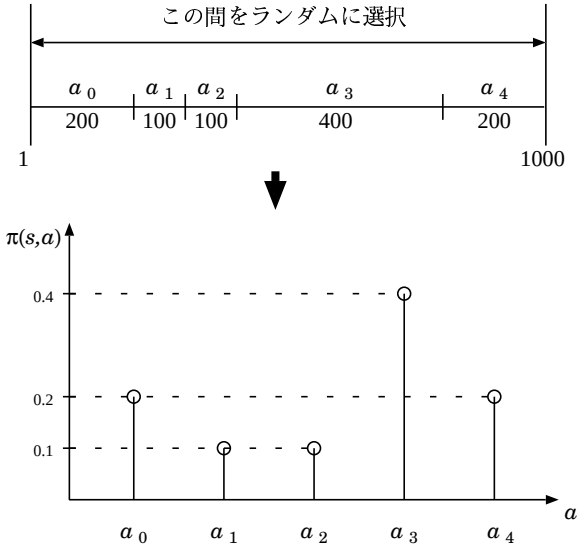

Fig. 6　πに従った行動選択の実現

5) 行動順番のランダム決定と行動実行

ここでは、まず乱数によって行動の順番をランダムに決定する。次にその順番に従って行動を実行し、環境も更新する。

6) 環境の観測とs'とrの取得
プレーヤーCの遷移後の状態s'は3)で用いた関数によって得る。報酬rは関数get_rew_from_Env()によって勝ったときは$+1.0$、負けたときは-1.0、その他の場合は-0.1とした。

```
double get_rew_from_Env( void )
{
    /* 勝った */
    if( is_goal( C ) )
    {
        return 1;
    }
    /* 負けた */
    else if( is_goal( P ) )
    {
        return -1;
    }
    else
    {
        return -0.1;
    }
}
```

ここで関数is_goal(int player)はプレーヤーがゴールしているときは1、それ以外のときは0を返す関数である。
7) Qの更新
状態s、行動a、遷移後の状態s'、報酬rと式(4)によってQを更新する。

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha(r + \gamma \max_{a' \in A} Q(s_{t+1},a') - Q(s_t,a_t)) \quad (4)$$

```
Q[s][a[C]] = (1.0-alpha)*Q[s][a[C]]
    + alpha*(rew + GAMMA*max_Q);
```

8) πの更新
πの値はQからボルツマン選択法(式(5))を用いて計算されるため、Qが更新されればπも更新できる。ここで、Qは$Q(s,a)$の値だけが更新されているが、式(5)からわかるように$Q(s,a)$が更新されると分母の値が変化するため、sに関わる全ての行動確率$\pi(s,a_0),\pi(s,a_1),\cdots,\pi(s,s_m)$も更新されることになる。プログラムでは以下のようになる。ここではステップごとに温度パラメータTを0に近づけるため、TEMPARA_DECAY(0<TEMPARA_DECAY<1)をかけている。

$$\pi(s,a) = \frac{e^{Q(s,a)/T}}{\sum_{b \in A} e^{Q(s,b)/T}} \quad (5)$$

```
/* Qに従って pi を更新 */
void boltzmann( const double Q_s[],
                    int pi_s[]      )
{
    double bunbo;
    static double temparature
                    = TEMPARATURE_PARA;
    int i;

    bunbo = 0.0;
    for( i = 0; i < NUM_OF_A; i++ )
    {
        bunbo = bunbo
        + exp( Q_s[i]/temparature );
    }

    for( i = 0; i < NUM_OF_A; i++ )
    {
        pi_s[i] = ((exp( Q_s[i]/
        temparature )) / bunbo);
    }
    temparature *= TEMPARA_DECAY;

    return;
}

boltzmann( Q[s], pi[s] );
```

以上を実装したシステムの画面を図7に示す。

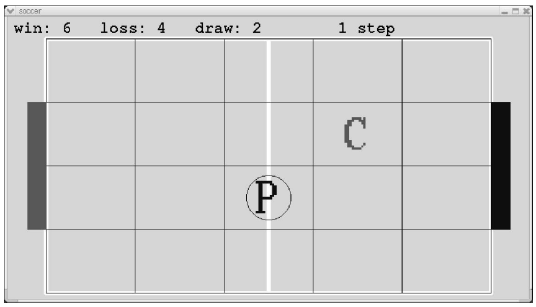

Fig. 7 サッカーゲームのフィールド

5. シミュレーション

学習step数1,000stepと1,000,000stepで事前学習を行った後、被検者P_1、P_2、P_3、P_4にそれぞれ10試合ずつ対戦を行わせた。ここで学習に用いたパ

ラメータは学習率αの初期値: 1.0、学習率αの減衰率: 0.9999954、割引率γ: 0.9、温度パラメータTの初期値: 0.10、温度パラメータTの減衰率: 0.999999、乱数の種: 1234とした。このときのプレーヤーCの対戦成績を表2にまとめた。

Table 2　プレーヤーCの対戦成績; 1セット10ゲーム

	ランダム戦略との事前学習step数	
	1,000	1,000,000
被験者P_1	0勝10敗0分	1勝4敗5分
被験者P_2	1勝 9敗0分	4勝5敗1分
被験者P_3	0勝 8敗2分	5勝5敗0分
被験者P_4	0勝10敗0分	0勝5敗5分
勝率	2.5%	25.0%
負け率	92.5%	47.5%

まず、事前学習step数が1,000stepの時の結果を見るとプレーヤーCはほとんど負けていることがわかる。これは事前の学習step数が1,000stepでは不充分であるためと考えられる。

一方、事前学習step数を1,000,000stepに増やした場合、プレーヤーCの勝率はあがり、ある程度戦略を学習できていることがわかる。

6.　結論

事前学習の際に学習step数を変えることで強化学習のエージェントも強さが変化することがわかった。ただし、現在のままでは学習が遅く、人間が戦略を変えてきた場合などに順応するまでに時間がかかりすぎるという問題がある。

今後はこのゲームの対称性を用いることで状態の数を減らしたり、対戦中の相手の行動を自分の手として学習するなどして、学習時間の短縮を目指す。

参考文献

1) 長行 康男, 伊藤 実:“2体エージェント確率ゲームにおける他エージェントの政策推定を利用した強化学習法”, 電子情報通信学会論文誌, vol.J86-D-I, No.11, pp.821-829, 2003

2) 木村 元、宮崎 和光、小林 重信: “強化学習システムの設計指針”,計測と制御, Vol.38, No.10, pp.618-623, 1999, 計測自動制御学会.

3) Michael L. Littman: “Markov games as a framework for multi-agent reinforcement learning”, Proc. 11th International Conference on Machine Learning, pp.157-163, New Brunswick, New Jersey, USA, July 1994.