迷路の強化学習

2017/07/19

大槻 知史

本スライドの目的

- 書籍「最強囲碁AIアルファ碁解体新書」では、強化学習の説明を、かなり端折りました。
- そこで、「最強囲碁AI～」で述べた迷路の事例における強化学習に関して、ソースコードを公開するとともに、少し補足したいと思います。
- ここでは、迷路の事例を考え、Q学習と方策勾配法という2つの強化学習手法を説明します。

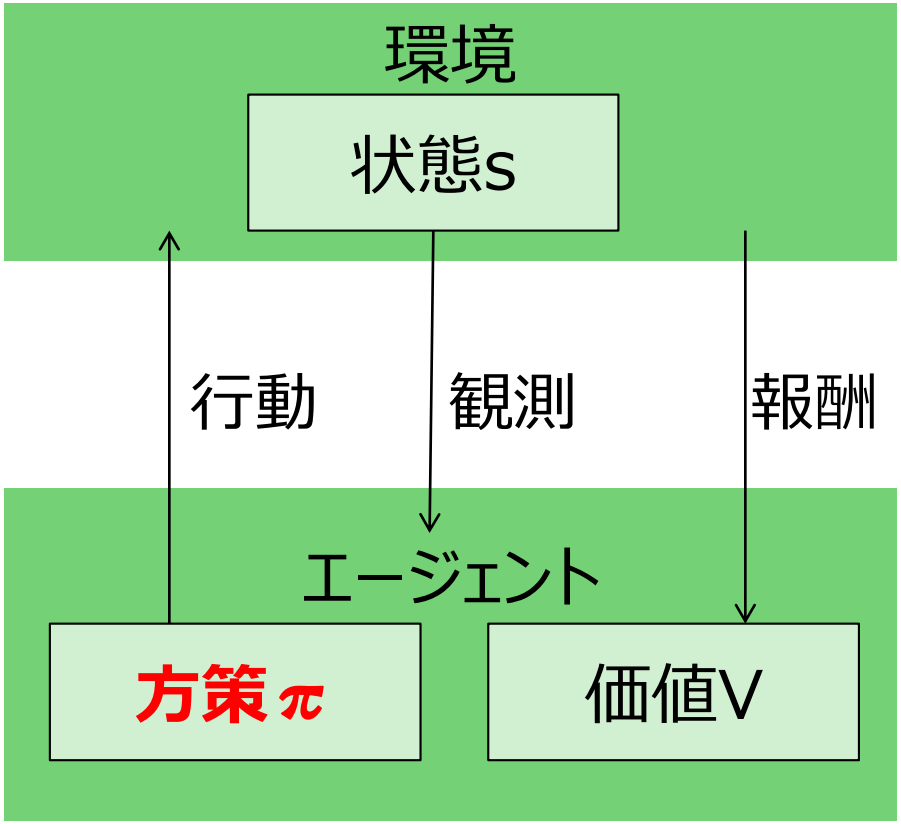
※「最強囲碁AI～」の説明と、ソースコードの内容をつなぐことが主目的なので、言葉の定義とか、詳しい説明は省略します。ご了承ください。

目次

- 迷路の強化学習の概要
- 迷路のQ学習
- 迷路の方策勾配法
- 実行方法
- まとめ

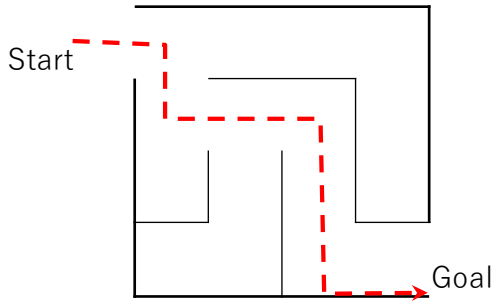
強化学習とは?

- **強化学習:未知の環境の中を探索しながら期待報酬和を最大化するためのエージェントの行動原理**
  - 正解は与えられないが選んだ答えの「良さ」(報酬)を元に、行動原理(方策)を改善

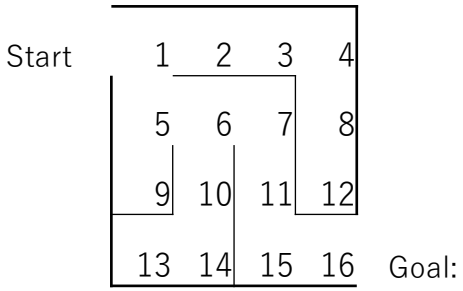


迷路の強化学習

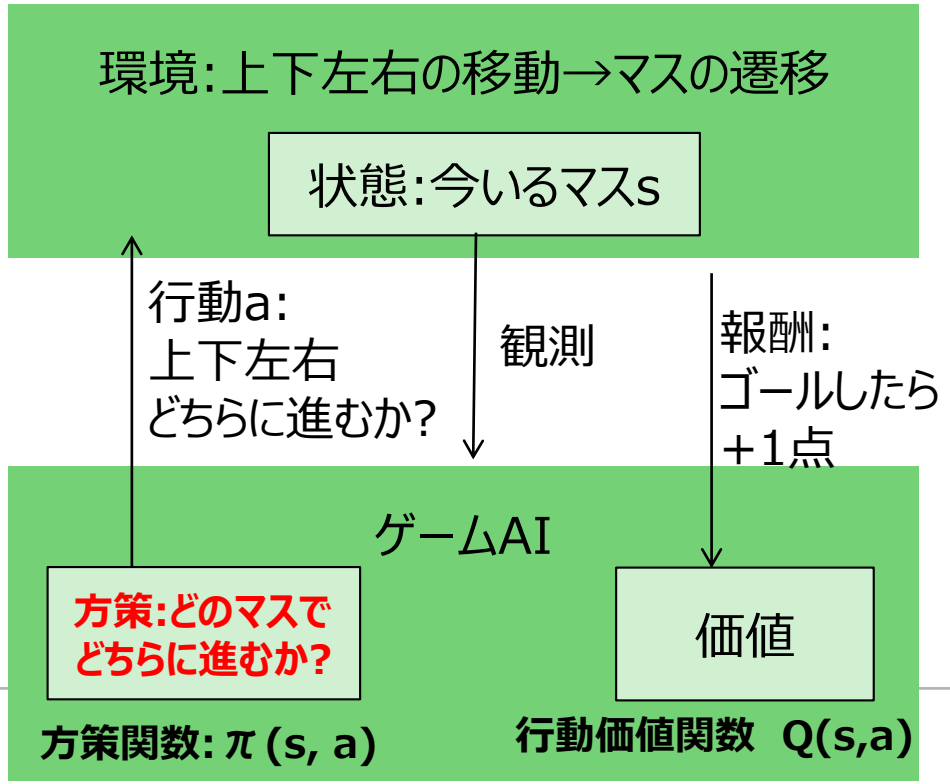
(a)迷路の例



(b)マスの座標



(c)迷路の強化学習のフレームワーク



迷路の強化学習の補足

- **状態　→　どのマスにいるか?**
- **行動　→　上下左右のどちらに進むか?**
- **方策　→　あるマスにいるときに、どちらに進むかの戦略**
- **環境　→　あるマスで、上下左右のいずれかに進むとき、どのマスに行くかを決定する迷路の構造そのもの**
- **報酬　→　ゴールすると1点、それ以外の行動は0点**
- **行動価値**
  - あるマスで、上下左右のそれぞれに進んだ時に得られる、報酬の長期的な期待値。マスと、行動に対する関数(または、テーブル)として表すことができる。
- **方策関数**
  - あるマスで、上下左右のそれぞれに進む確率を与える関数。マスと、行動に対する関数(またはテーブル)として表すことができる。
- **強化関数の目的**
  - スタートからゴールに最短経路で至る(⇔得られる報酬の期待値を最大化する)方策関数を見つけること

迷路の強化学習の骨格(Q学習の場合)

// EPISODE を繰り返して、パラメータを更新し、迷路を早く抜けられるようになる

//　in　main() 関数

```
for (i = 0; i < EPISODE_MAX; i++){   // 各エピソードに関するループ
    s = rl->start;                   // 状態を初期化(状態sをスタートとする)
    while(1){                        // goal に到達するまで、迷路の中を移動し続ける
      if (s == rl->goal){            // ゴールしたら、このエピソードは終了
        break;
      }
      a = get_next_action(rl, s, i);  // 次のアクションを決定
      next_s = NEXT[s][a];            // 環境がアクションaに基づき次の状態を決定する
      update_q(rl, s, a, next_s);     // Q学習の場合、行動ごとにパラメータを更新
      s = next_s;                     // 状態sを次の状態に更新する
    }
 }
```

迷路の強化学習の骨格(方策勾配法の場合)

// EPISODE を繰り返して、パラメータを更新し、迷路を早く抜けられるようになる

//　in　main() 関数

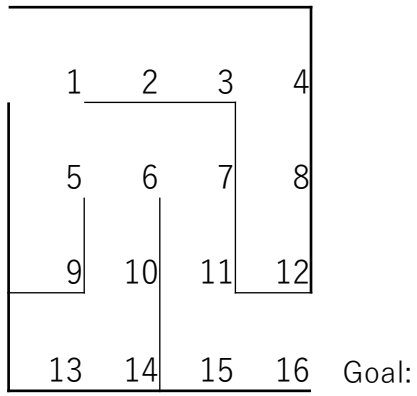
```
for (i = 0; i < EPISODE_MAX; i++){   // 各エピソードに関するループ
    s = rl->start;                   // 状態を初期化(状態sをスタートとする)
    clear_n(rl);                     //各(s,a)の頻度rl->N[s][a]を0で初期化する
    while(1){                        // goal に到達するまで、迷路の中を移動し続ける
      if (s == rl->goal){            // ゴールしたら、このエピソードは終了
        break;
      }
       rl->N[s][a]++;                // 各(s,a)の頻度を記録する
      a = get_next_action(rl, s, i); // 次のアクションを決定
      s = NEXT[s][a];                // 環境がアクションaに基づき次の状態を決定する
    }
    update_pi(rl, cnt);              // 方策勾配法では、エピソードごとにパラメータを更新
 }
```

迷路の環境を定義するのは、NEXT配列

```
// 環境を定義する配列、
// マスs で行動a を採ると、次にどのマスに移動するか決定
// 壁の向きに進もうとした場合は、同じマスに留まるとする
/* 0: 上, 1:右, 2:下, 3:左 */
int NEXT[STATE_MAX][ACTION_MAX] ={
   { 0, 0, 0, 0},
   { 1, 2, 5, 1},
   { 2, 3, 2, 1},
   { 3, 4, 3, 2},
   { 4, 4, 8, 3},
   { 1, 6, 9, 5},
   { 6, 7,10, 5},
   { 7, 7,11, 6},
   { 4, 8,12, 8},
   { 5, 9, 9, 9},
   { 6,10,14,10},
   { 7,11,15,11},
   { 8,12,12,12},
   {13,14,13,13},
   {10,14,14,13},
   {11,16,15,15},
   {16,17,16,15},     };
```

右図(b)と見比べながら
この配列を見てください
・マス1では
　上に行く(=0)と壁なので状態は1のまま
　右に行く(=1)とマス2に進む
　下に行く(=2)とマス5に進む
　左に行く(=3)と壁なので状態は1のまま
・マス2では
　上に行く(=0)と壁なので状態は2のまま
　　・・・

(b)マスの座標



目次

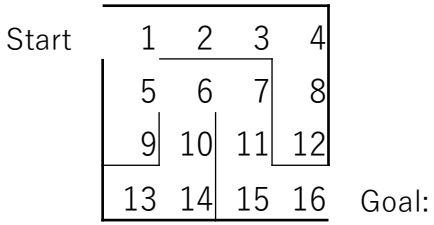
- 迷路の強化学習の概要
- 迷路のQ学習
- 迷路の方策勾配法
- 実行方法
- まとめ

Q学習とは

- Q学習は、ここでは、行動価値関数を得る手法と考えます。
- 16の各マスに対しそれぞれ4つの行動選択肢があるため、行動価値関数は16×4のテーブルで表せます(次頁図(c))
- Q学習は、ある行動を採るたびに、次に行くマスの価値と今いるマスの価値の差分を計算します。そしてその差分だけ、今いるマスの価値を増やすような手法です。
- イメージとしては、最初はゴール地点にだけコインが積まれているとします。これに対し、AからBへ動く行動を採る場合に、もし次の行先Bにコインが積んであったら、コインを少しもらってきて、今いるAにも積んでおく、ということをひたすら繰り返します。
- ゴールにあるコイン（報酬）を少しずつ分散していくことで、いつかスタート地点までコインの経路をつなげる、という方針です。

迷路のQ学習のパラメータ(テーブル)

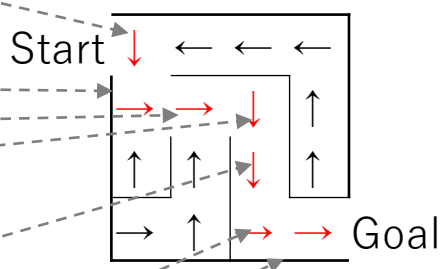
(b)マスの座標



(c)行動価値関数Q(s,a)の学習結果(エピソード = 100)

状態s(マス)	行動a			
	↑	→	↓	←
1	4.11	2.33	33.32	7.36
2	0.77	0.14	0.46	8.43
3	0.09	0.01	0.05	0.61
4	0.00	0.01	0.00	0.07
5	4.87	41.31	7.02	6.33
6	16.44	50.64	4.50	11.32
7	18.15	15.63	61.49	10.69
8	0.00	0.00	0.00	0.00
9	13.95	3.13	2.29	2.56
10	12.74	0.93	0.08	1.95
11	16.57	18.27	73.52	28.19
12	0.00	0.00	0.00	0.00
13	0.01	0.09	0.01	0.01
14	0.51	0.11	0.07	0.02
15	23.48	86.19	27.11	21.11
16	37.42	100.00	29.24	35.92

(d)行動価値関数を最大化する経路



Q学習のパラメータの更新式と方策

- **Q学習**
  - **1行動ごとに、行動価値関数Q(s,a)を更新**

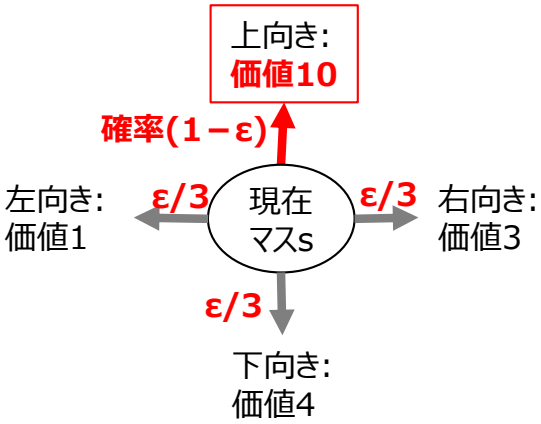
$$Q(s,a) \leftarrow Q(s,a) + \alpha \cdot \Delta Q$$

$$\Delta Q = r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a)$$

TD誤差*1　隣のマスの価値　現在マスの価値

(r: 次に得られる報酬，a': 次のマス)

  - **方策: ε-greedy**
    - **確率1-εでQ(s,a)最大の向き**
    - **確率εで、ランダム選択**



*1: 厳密に云うとΔQをTD誤差というのは誤りのようです

Q学習のパラメータ更新の実装

```
// Q学習では、行動するごとに下記の関数が呼ばれる
// 更新式は、前頁のもの
//  GAMMA(割引率) = 0.95
//  ALPHA(学習率)   = 0.10

  int update_q(rl_t *rl, const int s, const int a, const int next_s)
  {
     if (next_s == rl->goal){
        rl->Q[s][a] += 1.0;
     } else {
        int max_a = get_max_a(rl, next_s);
        rl->Q[s][a] += ALPHA * (GAMMA * rl->Q[next_s][max_a] - rl->Q[s][a]);
     }
     return 0;
  }
```

Q学習の方策(次のアクション決定)の実装: εグリーディ法

```
// Q学習の場合はεグリーディ法
int get_next_action(rl_t *rl, const int s, const int ite)
{
   double p = rand()/(double)RAND_MAX;
   double epsilon = (EPISODE_MAX - ite) /(double)EPISODE_MAX;  //εはエピソードの進行にともないだんだん小さく
   if (p <= epsilon){
      return rand()%ACTION_MAX;     // ランダムに行動を決定する処理
   } else {
      return get_max_a(rl, s);     //  Q[s][a] が最大となるa を採る処理
   }
}
```

目次

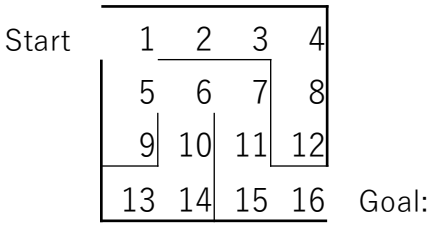
- 迷路の強化学習の概要
- 迷路のQ学習
- 迷路の方策勾配法
- 実行方法
- まとめ

方策勾配法とは

- 方策勾配法は、ここでは、各マスにおいて、各行動を採る確率を付ける方策関数を得るための手法とします。
- 方策関数も、価値関数と同様に16×4のテーブルとして表されます（次頁)
- 方策勾配法は、1エピソード終わるごとに、そのエピソードで採用した行動の確率を少し高め、それ以外の行動の確率を少し下げる、ということを繰り返す手法です。
- ゴールした経路に含まれる行動は、「良い行動であることが多い」という経験則に基づく手法です。

迷路の方策勾配法のパラメータ(テーブル)

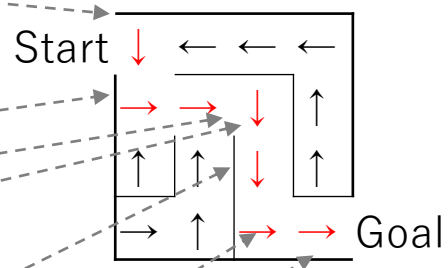
(b)マスの座標



(c)方策テーブル$\pi(s,a)$の学習結果(エピソード = 100)

状態s (マス)	行動a			
	↑	→	↓	←
1	9.0%	15.0%	**74.0%**	1.0%
2	3.0%	41.0%	2.0%	54.0%
3	17.0%	3.0%	5.0%	75.0%
4	15.0%	29.0%	17.0%	39.0%
5	1.0%	**98.0%**	0.0%	1.0%
6	1.0%	**96.0%**	1.0%	3.0%
7	1.0%	3.0%	**95.0%**	1.0%
8	21.0%	21.0%	41.0%	17.0%
9	16.0%	16.0%	15.0%	53.0%
10	28.0%	9.0%	17.0%	46.0%
11	2.0%	1.0%	**95.0%**	2.0%
12	30.0%	21.0%	30.0%	19.0%
13	26.0%	31.0%	22.0%	22.0%
14	50.0%	27.0%	12.0%	11.0%
15	2.0%	**96.0%**	1.0%	2.0%
16	0.0%	**98.0%**	1.0%	1.0%

(d)方策関数を最大化する経路



方策勾配法のパラメータ更新式と方策関数

- **方策勾配法**
  - **1エピソードごとに、方策のパラメータテーブル$\pi(s,a)$を更新**[*1]

$$\pi(s,a) \leftarrow \pi(s,a) + \Delta\pi(s,a)$$

$$\Delta\pi(s,a) \sim \{(1-p(s,a))\cdot N_1 - p(s,a)\cdot N_2\}/T$$

$$= \{N_1 - p(s,a)\cdot N_0\}/T$$

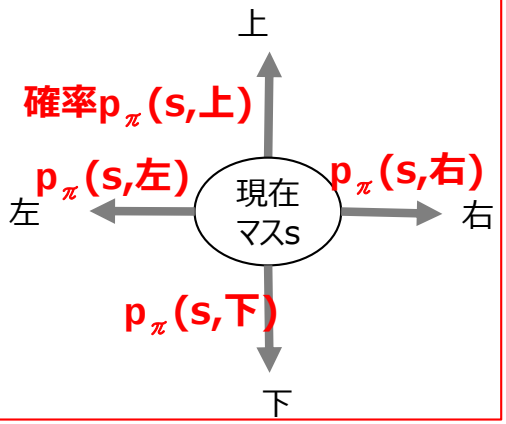
N_1：状態sで行動aを採った回数

N_2：状態sで行動a以外を採った回数

T：当該エピソードでゴールにたどり着くまでに要したステップ数

  - **方策関数は$\pi(s,a)$を用いたソフトマックス関数**[*2]**:**

$$p_\pi(s,a) = \frac{e^{\beta\cdot\pi(s,a)}}{\Sigma_b e^{\beta\cdot\pi(s,b)}}$$



*1:「最強囲碁AI～」の付録A1.2.2節を参照

*2:「最強囲碁AI～」では煩雑な議論を避けるため、 $p_\pi(s,a)$ と$\pi(s,a)$の区別を曖昧にしたまま議論しているので要注意

方策勾配法のパラメータ更新の実装

```
// 方策勾配法では、1エピソード終了する(ゴールする)ごとに下記が呼ばれる
// 更新式は、前頁のもの
//  ALPHA(学習率)   = 0.10
//  BETA(ソフトマックス関数の温度) = 100.0
int update_pi(rl_t *rl, const int ite)
{
   int s, a, n1, n0;
   double delta;

   for (s = 0; s < STATE_MAX; s++){
      for (a = 0; a < ACTION_MAX; a++){
         n1 = rl->N[s][a];
         n0 = get_freq(rl, s, a);
         // 方策勾配法の更新式(前頁)
         delta = (n1 - n0 * softmax(rl, s, a))/(double)(ite);
         rl->PI[s][a] += ALPHA * delta;
      }
   }
   return 0;
}
```

方策勾配法の方策(次のアクション決定): ソフトマックス関数

```
// 方策勾配法の場合は、ソフトマックス関数により行動を決定
//  BETA  (ソフトマックス関数の温度) = 100.0


  int get_next_action(rl_t *rl, const int s, const int ite)
  {
    double p = rand()/(double)RAND_MAX;
    // 方策勾配法の場合は、ソフトマックス関数により次の行動を決定
    int a;
    double sum = 0.0;
    for (a = 0; a < ACTION_MAX; a++){
      sum += softmax(rl, s, a);
      if (sum >= p){
        return a;
      }
    }
  }
```

目次

- 迷路の強化学習の概要
- 迷路のQ学習
- 迷路の方策勾配法
- 実行方法
- まとめ

ソースコードからの実行方法

- **Q学習を実行したい場合**
  - maze.c の14行目を**有効**にして(つまり下記のようにして)
    - #define QLEARNING
  - コンパイル(ビルド)方法
    - (eg:) gcc環境ならば、 gcc maze.c を実行
  - 実行方法
    - ./a.out
    - すると、それっぽい出力が出ます。
- **方策勾配法を実行したい場合**
  - maze.c の14行目を**無効**にして(つまり下記のようにして)
    - //#define QLEARNING
  - コンパイル、実行方法は、Q学習と同じです。

Q学習の実行例

100エピソード後には、最短経路の向きと、袋小路から抜け出す向きを学習できている

(a)1エピソード経過後

状態s（マス）	行動a			
	↑	→	↓	←
1	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00
6	0.00	0.00	0.00	0.00
7	0.00	0.00	0.00	0.00
8	0.00	0.00	0.00	0.00
9	0.00	0.00	0.00	0.00
10	0.00	0.00	0.00	0.00
11	0.00	0.00	0.00	0.00
12	0.00	0.00	0.00	0.00
13	0.00	0.00	0.00	0.00
14	0.00	0.00	0.00	0.00
15	0.00	0.00	0.00	0.00
16	0.00	1.00	0.00	0.00

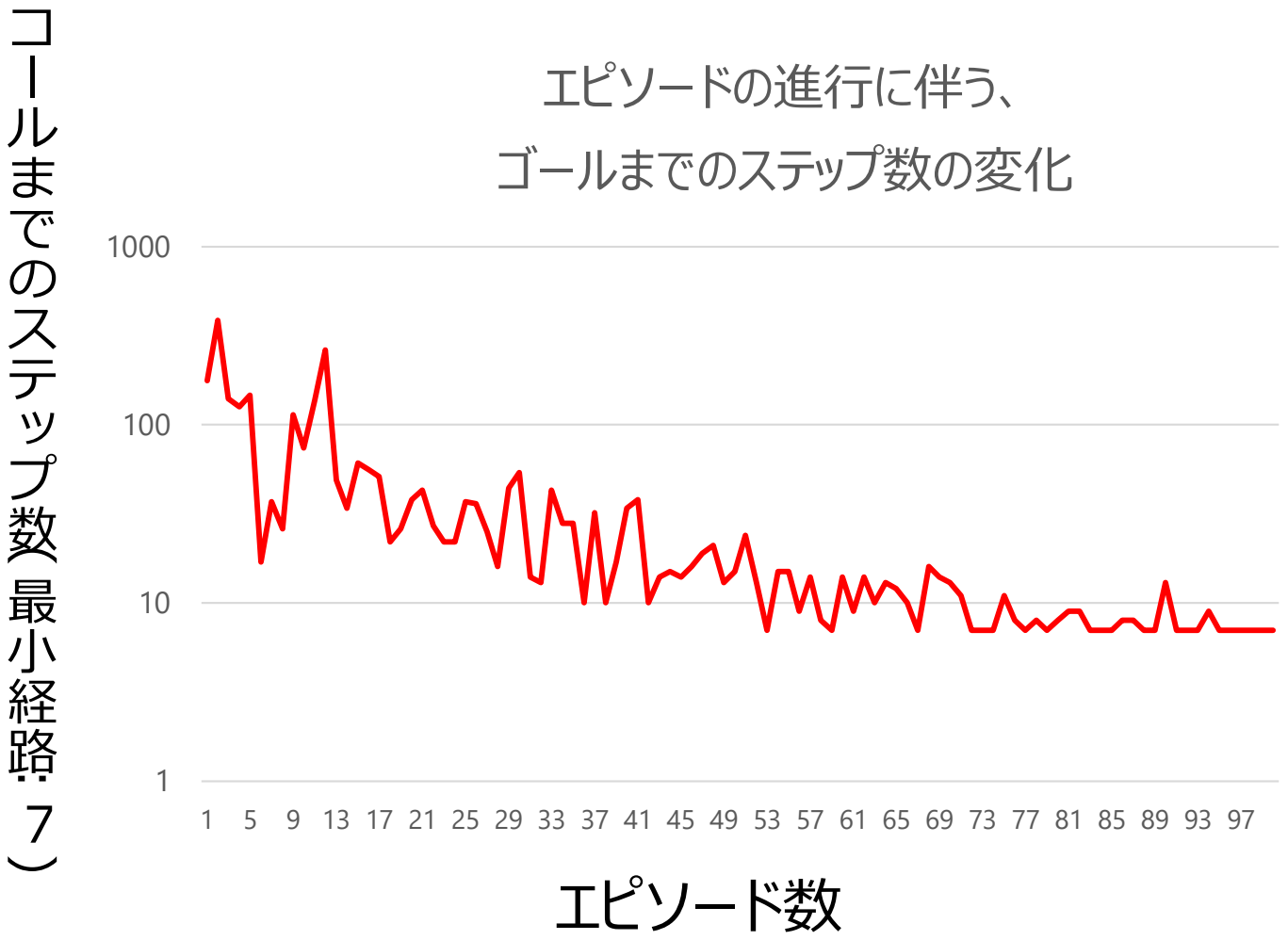
(b)50エピソード経過後

状態s（マス）	行動a			
	↑	→	↓	←
1	2.73	1.01	8.24	2.35
2	0.17	0.14	0.46	2.86
3	0.09	0.01	0.05	0.61
4	0.00	0.01	0.00	0.07
5	2.84	11.84	3.67	5.12
6	8.81	17.45	1.96	5.70
7	10.83	12.67	23.96	8.21
8	0.00	0.00	0.00	0.00
9	6.34	2.15	1.17	1.57
10	5.39	0.93	0.08	0.86
11	11.06	13.25	31.84	17.00
12	0.00	0.00	0.00	0.00
13	0.01	0.09	0.01	0.01
14	0.51	0.11	0.07	0.02
15	14.98	39.78	19.49	16.66
16	23.65	50.00	18.15	17.18

(c)100エピソード経過後

状態s（マス）	行動a			
	↑	→	↓	←
1	4.11	2.33	33.32	7.36
2	0.77	0.14	0.46	8.43
3	0.09	0.01	0.05	0.61
4	0.00	0.01	0.00	0.07
5	4.87	41.31	7.02	6.33
6	16.44	50.64	4.50	11.32
7	18.15	15.63	61.49	10.69
8	0.00	0.00	0.00	0.00
9	13.95	3.13	2.29	2.56
10	12.74	0.93	0.08	1.95
11	16.57	18.27	73.52	28.19
12	0.00	0.00	0.00	0.00
13	0.01	0.09	0.01	0.01
14	0.51	0.11	0.07	0.02
15	23.48	86.19	27.11	21.11
16	37.42	100.00	29.24	35.92

Q学習の学習の進行



※Q学習は、比較的うまく収束することが多いようだ

方策勾配法の実行例

(a)1エピソード経過後

状態s（マス）	行動a			
	↑	→	↓	←
1	23.0%	17.0%	29.0%	30.0%
2	25.0%	25.0%	25.0%	25.0%
3	25.0%	25.0%	25.0%	25.0%
4	25.0%	25.0%	25.0%	25.0%
5	18.0%	39.0%	25.0%	19.0%
6	17.0%	22.0%	22.0%	38.0%
7	23.0%	23.0%	31.0%	23.0%
8	25.0%	25.0%	25.0%	25.0%
9	24.0%	32.0%	25.0%	19.0%
10	27.0%	20.0%	26.0%	27.0%
11	22.0%	22.0%	28.0%	28.0%
12	25.0%	25.0%	25.0%	25.0%
13	25.0%	25.0%	25.0%	25.0%
14	28.0%	28.0%	22.0%	22.0%
15	23.0%	31.0%	23.0%	23.0%
16	23.0%	31.0%	23.0%	23.0%

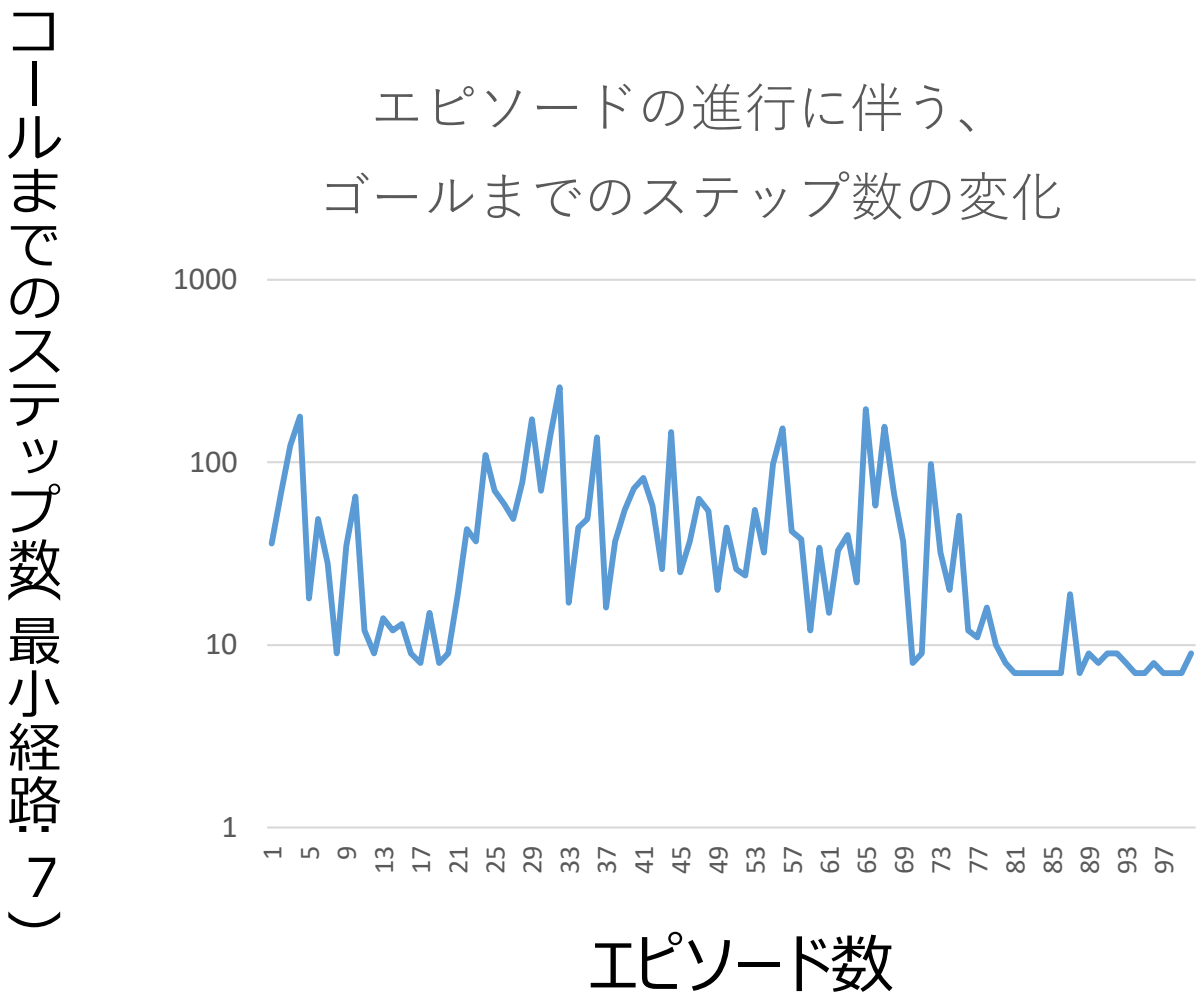
(b)50エピソード経過後

状態s（マス）	行動a			
	↑	→	↓	←
1	78.0%	4.0%	5.0%	13.0%
2	29.0%	37.0%	9.0%	25.0%
3	11.0%	10.0%	12.0%	67.0%
4	18.0%	34.0%	17.0%	31.0%
5	3.0%	92.0%	1.0%	4.0%
6	7.0%	73.0%	5.0%	14.0%
7	2.0%	1.0%	93.0%	3.0%
8	25.0%	30.0%	27.0%	19.0%
9	16.0%	16.0%	15.0%	53.0%
10	30.0%	10.0%	19.0%	41.0%
11	3.0%	3.0%	73.0%	22.0%
12	30.0%	20.0%	29.0%	22.0%
13	26.0%	31.0%	22.0%	22.0%
14	49.0%	29.0%	11.0%	11.0%
15	3.0%	40.0%	7.0%	50.0%
16	2.0%	92.0%	3.0%	3.0%

(c)100エピソード経過後

状態s（マス）	行動a			
	↑	→	↓	←
1	9.0%	15.0%	74.0%	1.0%
2	3.0%	41.0%	2.0%	54.0%
3	17.0%	3.0%	5.0%	75.0%
4	15.0%	29.0%	17.0%	39.0%
5	1.0%	98.0%	0.0%	1.0%
6	1.0%	96.0%	1.0%	3.0%
7	1.0%	3.0%	95.0%	1.0%
8	21.0%	21.0%	41.0%	17.0%
9	16.0%	16.0%	15.0%	53.0%
10	28.0%	9.0%	17.0%	46.0%
11	2.0%	1.0%	95.0%	2.0%
12	30.0%	21.0%	30.0%	19.0%
13	26.0%	31.0%	22.0%	22.0%
14	50.0%	27.0%	12.0%	11.0%
15	2.0%	96.0%	1.0%	2.0%
16	0.0%	98.0%	1.0%	1.0%

方策勾配法の学習の進行



※方策勾配法はあまり安定しないようだ。パラメータ要チューニング(βが大きすぎる?)

目次

- 迷路の強化学習の概要
- 迷路のQ学習
- 迷路の方策勾配法
- 実行方法
- まとめ

まとめ

- 簡単な迷路の事例により、強化学習のフレームワークを説明しました
- Q学習、方策勾配法の2つの手法の実装と実行例を説明しました
- 2手法共に、100エピソード程度で収束することを確認しました