






ポケモンで きょうか がくしゅう
を おうようへん

ポケモンを例題として, Q学習を理解する
名古屋工業大学 助教 上村知也

-  1. 問題設定(前回の復習)
-  2. 未知環境に対する学習(モンテカルロ法)
-  3. Q学習とSARSA(TD法)


1. 問題設定(前回の復習)

ピカチュウ対コイキング

- ピカチュウとコイキングの戦い
- 問題はゲーム実機よりも簡単しておく
 - 技は(技に固有の)確率でヒットし, 固定値のダメージを与える



コイキング

HP:  21/21

HP:21

持ち物なし

わざ:

はねる ダメージを与えることができない

- 攻撃し合うとややこしいので, コイキングは一切反撃できないように技を設定する
- PP切れにはならないので, わるあがきは出せない

わざ:

はねる めいちゅう 0 ダメージ 0



ピカチュウ

HP:  100/100

HP:(今回関係ない)

持ち物なし

わざ:

たいあたり 命中率100%, ダメージ5



でんじほう 命中率50%, ダメージ20

- ダメージ期待値はでんじほうの方が高い

わざ:

たいあたり めいちゅう 100 ダメージ 5

でんじほう めいちゅう 50 ダメージ 20

- 状態 s_t は, t ターン目のコイキングの残りHP
- 行動 a_t は, t ターン目のピカチュウの技
- ピカチュウは1ターンに1回**行動**を行い(技を使用する), コイキングの**状態**(残りHP)が確率的に変化する
- ピカチュウの攻撃の結果, コイキングの残りHPが0または負の値になったとき, 残りHPをゼロにする
 - このときを**終端状態**として, 試合が終了する

可能な状態の集合 $S = \{0, 1, 2, \dots, 21\}$

可能な行動の集合 $A = \{T, D\}$

T: たいあたり, D: でんじほう

前回までと比較して,
状態集合が変更されています!

- 方策の良し悪しを評価するために、報酬 r_t を設定する
- コイキングに勝利することが目的なので、 $s_{t+1} = 0$ となった瞬間に $r_t = 10$ を与える
- 長々と戦うことに価値はないので、それ以外の状態では $r_t = -1$ を与える
- 戦いが終わったあとは何をしても変わらないので、 $r_t = 0$ を与える
- 累計の報酬

$$G_t = \sum_{i=1}^t r_i = r_1 + r_2 + \cdots + r_t$$

を求める. これが方策の良し悪しを判断する基準になる



2. 未知環境に対する学習 (モンテカルロ法)

- 動的計画法では、環境を全て知っている必要があった
- 例えば、最も価値の高い行動を選択し続けるアルゴリズム「価値反復」では、ある行動を起こしたとき、**どれだけの確率で、どの次状態に遷移するのか、予め知っていないと計算ができない**

$$v_{k+1}(s) = \max_a \sum_{s', r} \underbrace{p(s', r | s, a)}_{\text{これを予め知っている必要}} \{r + \gamma v_k(s')\}$$

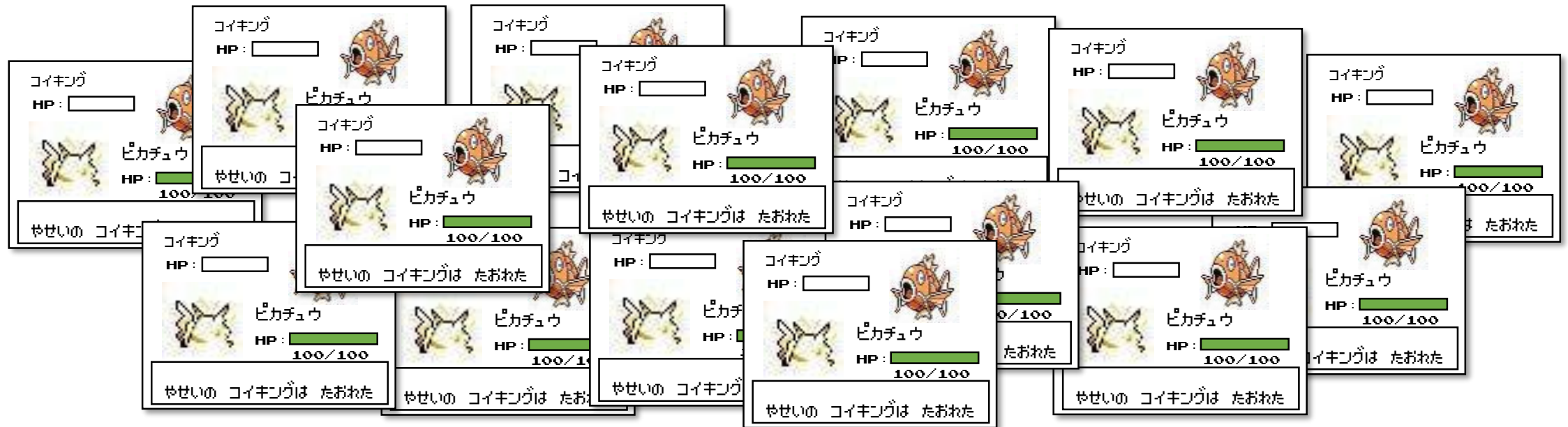
これを予め知っている必要

- 環境が未知の場合は、最適行動を探索するどころか、価値関数の推定すらできない
- 例えば、ピカチュウが命中率と威力が不明な新たな技「ピカボルト」を覚えたとき、どうすれば最適方策を得られるだろう？

モンテカルロ(MC)予測

9

- 再び, ある方策に対して状態価値関数を推定する方法を考えるとところから始める
- **モンテカルロ予測**では, 状態価値関数を今後得られる**累計報酬の平均値**として推定
 - 多数回の実験を行い平均値を取ることで, 未知の環境に対しても適用できる
 - 状態価値関数の本来の定義は, 「今後得られる**累計報酬の期待値**」
 - 理論上, 無限回の実験を行えば, 期待値 = 平均値(大数の法則)



1. 1つのエピソード(初期状態から終端状態にいたるまで)を実験やシミュレーションによって生成
2. 得られたエピソードから, 各ステップに対して累計報酬を計算
3. 累計報酬の平均値を, 価値関数の推定値とする
4. 1~3を, できるだけ多くの回数繰り返す

できるだけ多く繰り返し:

エピソードの生成

$G \leftarrow 0$

エピソードの各ステップについて繰り返し

$G \leftarrow \gamma G + r$

s が, エピソード内で初出現なら

G を $Returns(s)$ に付け足し(足し算するのではない)

$V(s) \leftarrow average(Returns(s))$

今回の設定のポケモン問題であれば,
コイキングのHPは減る一方だから
この条件は必ず満たされる

これまでの実験結果をひたすら貯める箱

MATLABで作成したサンプルプログラムpokemon_MC1.mに, 上記アルゴリズムを実装している.
pokemon_MC2.mでは, メモリ使用量を削減した平均値の算出を行っている.

サンプルプログラムpokemon_MC1.mの一部

```
% 方策を定義する.  
policy = 3;  
  
% 状態価値関数を定義する  
value = zeros(length(states),1); % 初期値は全て0  
  
% 平均値を求めるためにデータを貯めておく箱  
for i_s = 1:length(states)  
    R(i_s).G = [];  
end  
  
tic  
% 繰り返し計算によって、現在の方策に対して状態価値関数を推定する  
for i_v = 1:1e5 % たくさん繰り返す  
  
    s0 = round(rand()*max(states)); % 初期状態の残りHPをランダムに決定する  
    [sset,rset] = episode(s0,policy); % 1回の戦闘が終了するまで実行する  
  
    G = 0;  
    for i_s = length(sset):-1:1  
        s = sset(i_s);  
        G = G + discount * rset(i_s);  
        R(states==s).G = [R(states==s).G;G];  
        value(states==s) = mean([R(states==s).G]);  
    end  
  
    if discount == 1 && policy < 3 % この場合は解析解を求めている  
        error(i_v) = norm(value - value_true(:, policy));  
    end  
  
end  
toc
```

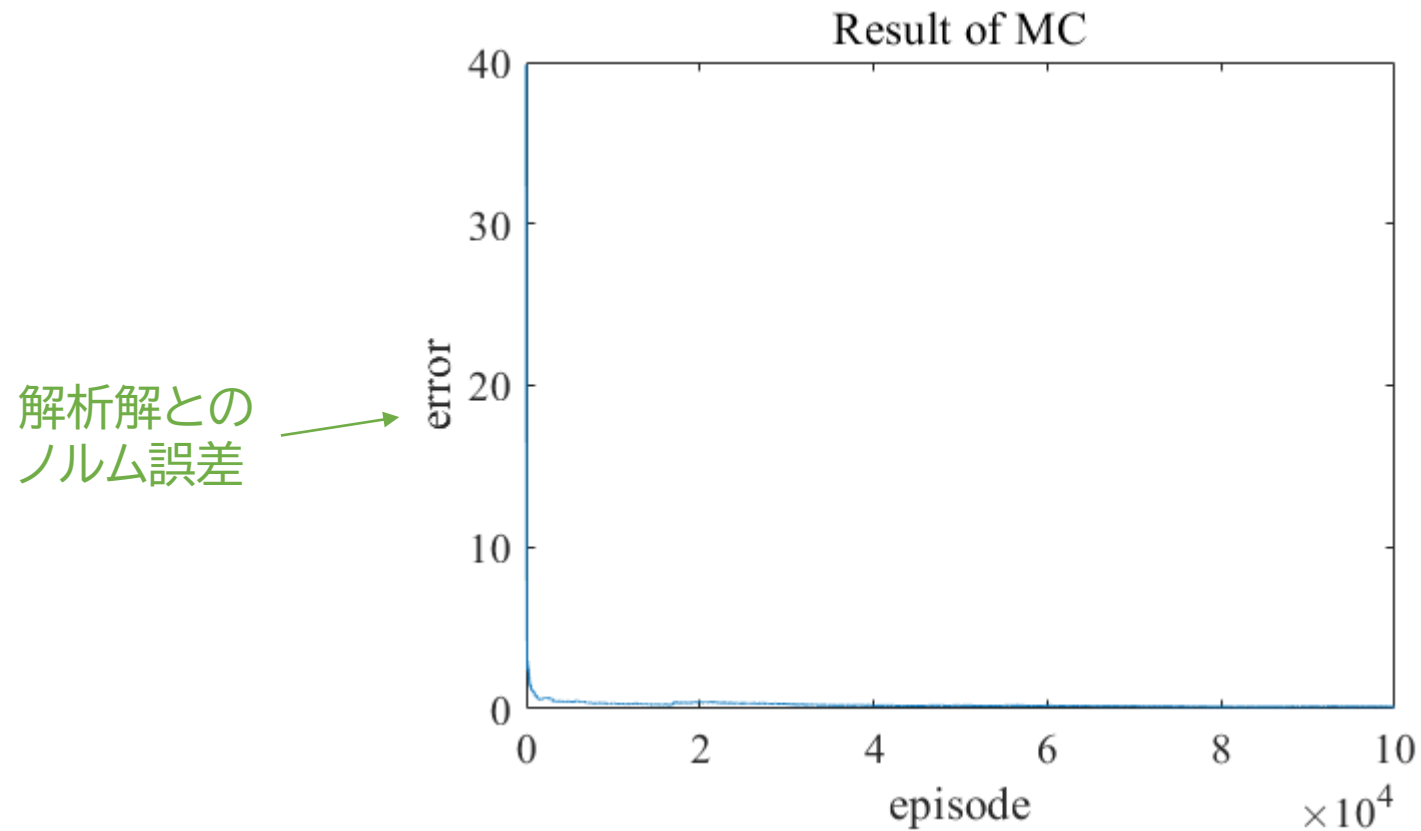
何らかの方法でエピソードを生成する
アルゴリズム内に環境の情報が入っていないので、
未知環境においてもこのアルゴリズムは適用可能

わざのダメージ設定によっては、何度繰り返しても
全ての状態 s が得られないことがある
そのため、コイキングの残りHPを
最大値以外でも試すようにしている

例によって終端状態から逆向きに計算している

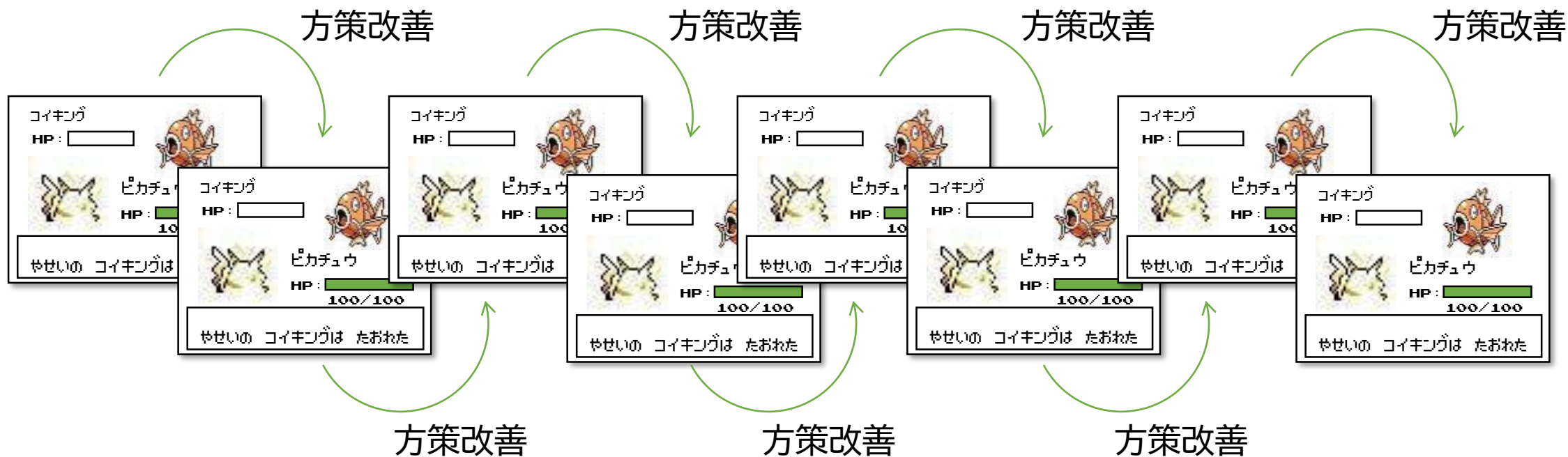
蓄えたデータの平均値から状態価値関数を推定

サンプルプログラムpokemon_MC1.mを実行し、
でんじほうのみを選択する方策に対する状態価値関数を推定した結果



- 理論値に近い価値関数の推定値が得られる
- より高い精度を得るためには、繰り返し回数を増やしていけばよいが、計算時間はその分増える

- 多くのエピソードを使って, 行動価値関数を推定できる
- 推定値をもとにして行動価値関数を作り, 最適な行動を求めていく
- 理論上は, 方策改善のたびに多数のエピソードを用いて方策を評価するのがよいが, それでは時間がかかりすぎてしまう
- 実際の運用では, **エピソードごとに評価と改善を繰り返す**



1. ランダムに状態 s_0 と行動 a_0 を選択(開始点探索, Exploring Starts)
2. 現在の方策 π にそって, s_0 と a_0 から始まるエピソードを生成
3. エピソード内の各 s について, 平均値として行動価値関数 $Q(s, a)$ を推定
4. 行動価値関数を用いて, Greedyな方策 π に更新
5. 1~4をできるだけ多く繰り返す

できるだけ多く繰り返し:

ランダムに状態 s_0 と行動 a_0 を選択

方策 π に基づいて, s_0 と a_0 から始まるエピソードを生成

$G \leftarrow 0, n \leftarrow 0$

エピソードの各ステップについて繰り返し

$G \leftarrow \gamma G + r$

(s, a) が, エピソード内で初出現なら

$Q(s, a) \leftarrow (Q(s, a)n_{s,a} + G)/(n_{s,a} + 1)$

$\pi(s) \leftarrow \arg \max_a Q(s, a)$

$n_{s,a} \leftarrow n_{s,a} + 1$

← メモリを節約しながら
平均値を計算

サンプルプログラムpokemon_MC3.mの一部

```
20 % 繰り返し計算によって、現在の方策に対して状態価値関数を推定する
21 for i_v = 1:1e6 % たくさん繰り返す
22
23     s0 = round(rand()*(max(states)-1))+1; % 初期状態の残りHPをランダムに決定する
24     a0 = round(rand()*(max(actions)-1))+1; % 初期行動をランダムに決定する
25     [sset,aset,rset] = episode(s0,a0, policy,states); % 1回の戦闘が終了するまで実行する
26
27     G = 0;
28     for i_s = length(sset):-1:1
29         s = sset(i_s);
30         a = aset(i_s);
31         G = G + discount * rset(i_s);
32         cnt(states==s,actions==a) = cnt(states==s,actions==a) + 1;
33         q(states==s,actions==a) = (q(states==s,actions==a)*(cnt(states==s,actions==a)-1) + G)/cnt(states==s,actions==a); % 平均値を求める
34         [max_q, max_a] = max(q(states==s,:));
35         policy(states==s,:) = zeros(1,length(actions));
36         policy(states==s,max_a) = 1;
37     end
38 end
```

適当な初期値からスタートして、すべての場合を試す
また、1回目の行動だけは現在の方策に関係なく
ランダムに試してみて、新たな可能性を探る

行動価値関数は平均値として求める

現在推定している行動価値関数が最大になるように、
方策を更新する

上記プログラムを実行すると、理論値に近い価値関数の推定値と、それに基づいた最適方策を得る
より高い精度を得るためには、繰り返し回数を増やしていけばよいが、計算時間はその分増える



3. Q学習とSARSA (TD法)

【DP法】

- ベルマン方程式を活用して、各状態における状態価値関数を推定
- 環境が全て分かっていないと実行できない

【モンテカルロ法】

- 未知環境でも、たくさんのサンプルデータを用いて、状態価値関数を推定
- ベルマン方程式に示される、状態と状態の関係を全く活用していない

Q 両方の良いところを取ってきたような強化学習は可能だろうか？

A 時間的差分(**Temporal Differential, TD**)学習なら可能！

- 環境モデルを必要とせず、経験から学ぶ
- ベルマン方程式を活用して、推定値を他の推定に用いる(ブートストラップ)ことで、エピソードが終了する前に計算可能

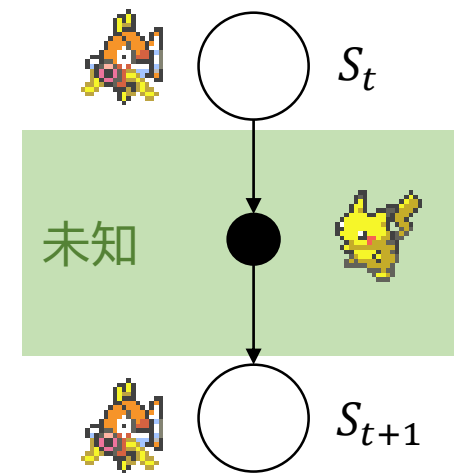
- ここでも, ある方策に対して状態価値関数を推定することから始める
- ベルマン方程式 $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s]$ を参考に, 実験(シミュレーション)中に報酬を受け取り, 次状態が確定した段階で,

$$V(S_t) \leftarrow V(S_t) + \alpha [R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

によって状態価値関数の平均値を取ることなく,
推定値を逐次的に更新していく

TD誤差

現在の推定値 $V(S_t)$ と, より良い推定値 $R_{t+1} + \gamma V(S_{t+1})$ との誤差



特徴

- モンテカルロ法と同様に, 環境モデルが不明でも使える
- モンテカルロ法と違って, エピソードが終了するのを待つ必要がない

pokemon_TD1.mが, TD(0)法とも呼ばれるTD予測のサンプルプログラムである

サンプルプログラムpokemon_TD1.mの一部

```
% ステップサイズパラメータ
```

```
alpha = 0.1;
```

```
% 状態価値関数
```

```
v = zeros(length(states),1); % 初期値は全て0
```

```
tic
```

```
% 繰り返し計算によって、現在の方策に対して状態価値関数を推定する
```

```
for i_v = 1:1e5 % たくさん繰り返す
```

```
    s = round(rand()*(max(states)-1))+1; % 初期状態の残りHPをランダムに決定する
```

```
    while s > 0
```

```
        a = find(policy(states==s,:),1);
```

```
        next_s = battle(s,a);
```

```
        r = reward(next_s);
```

```
        v(states==s) = v(states==s) + alpha*(r + discount*v(states==next_s) - v(states==s));
```

```
        s = next_s;
```

```
    end
```

```
    if discount == 1 && i_action < 3 % この場合は解析解を求めている
```

```
        error(i_v) = norm(v - v_true(:, i_action));
```

```
    end
```

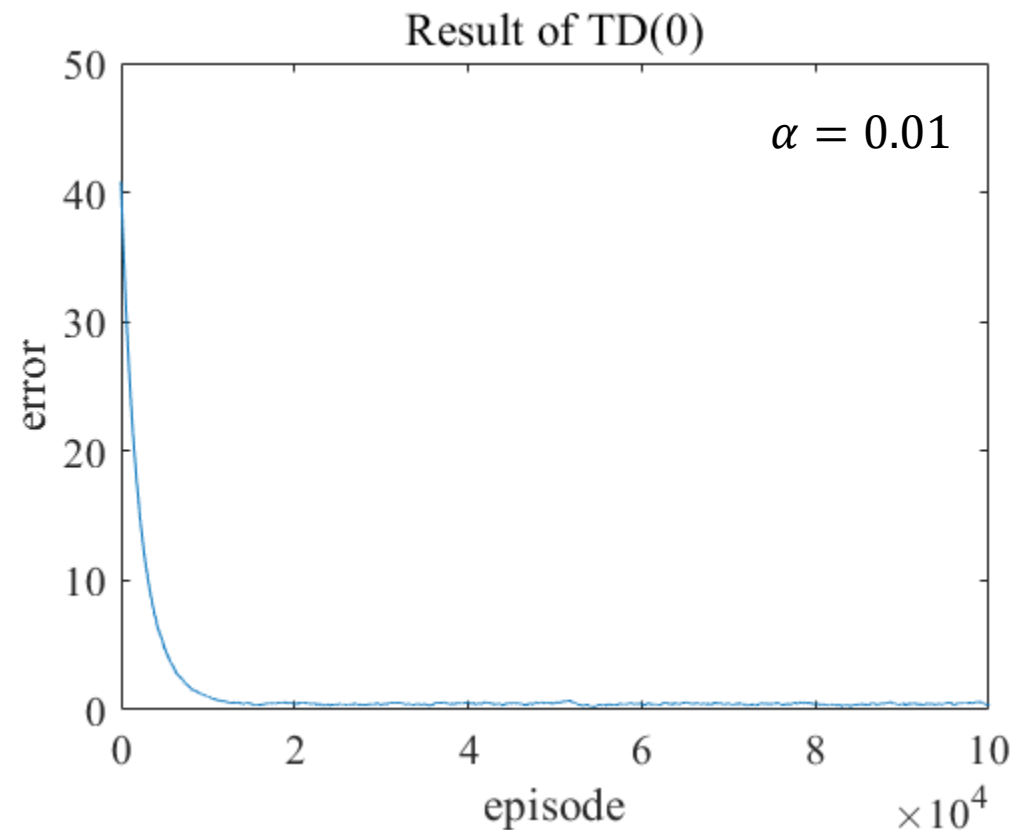
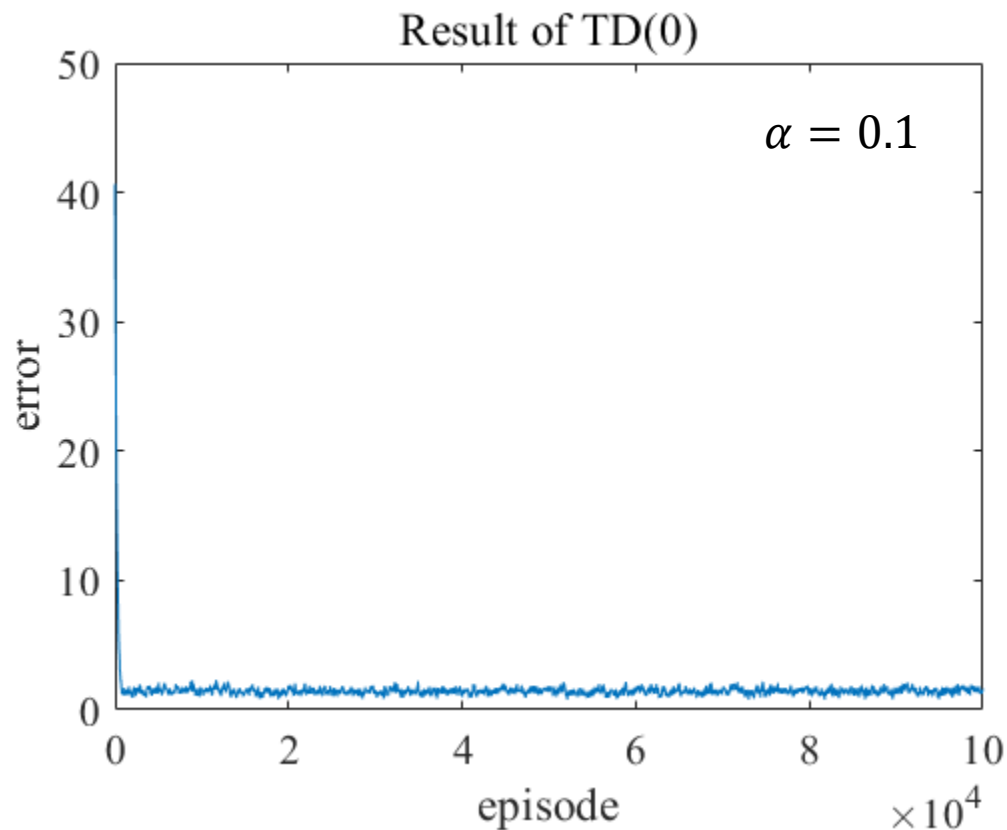
```
end
```

```
toc
```

エピソードが終了するのを待つことなく、
1ターンの行動が終了した時点で
状態価値関数を更新していく

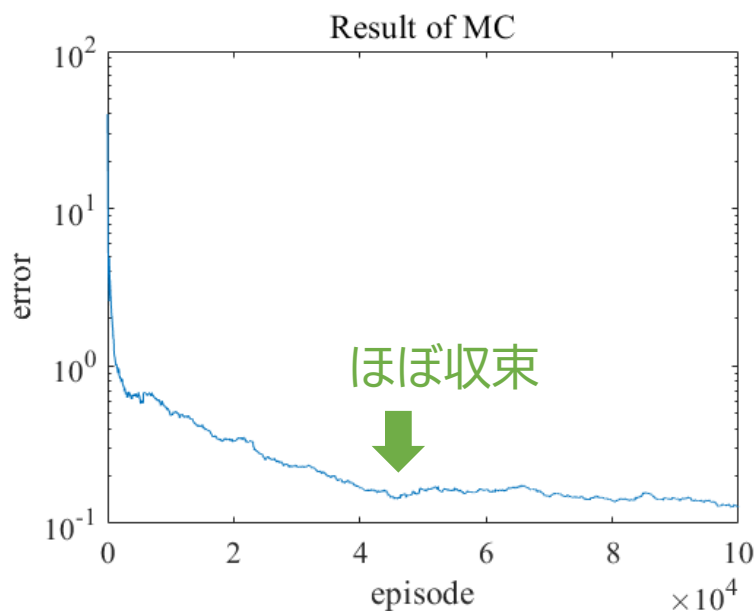


サンプルプログラムpokemon_TD1.mを実行し、
でんじほうのみを選択する方策に対する状態価値関数を推定した結果

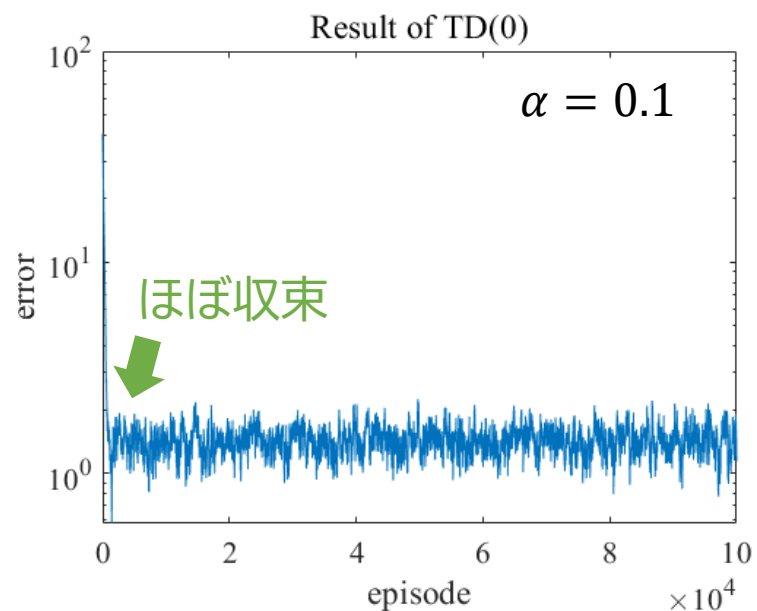


- 理論値に近い価値関数の推定値が得られる
- 誤差はある程度まで下がるが、ステップサイズパラメータ α が一定であれば、最新の結果に応じていつまでも上下し続ける
- α が小さければ、収束は遅くなるが、誤差はより小さな値に収束する
- 計算が進むほど α を小さくするように調整すれば、誤差はやがて0に収束する

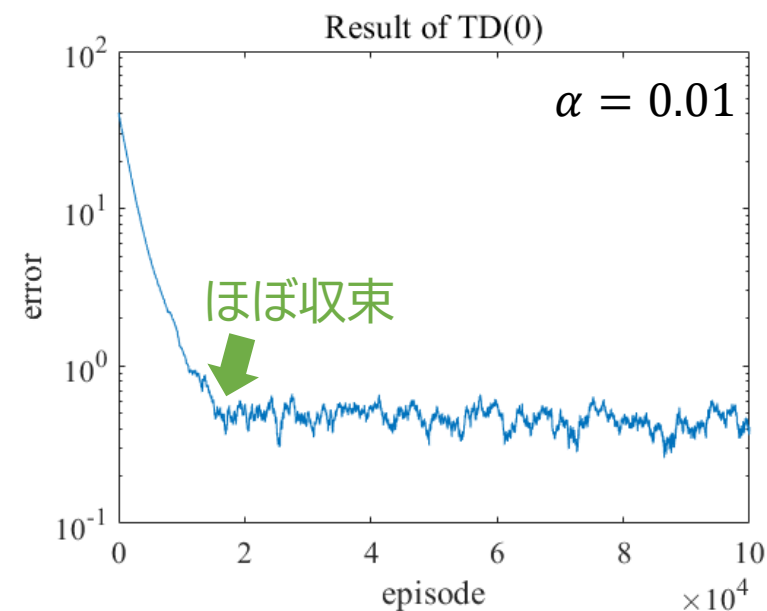
- TD法はエピソードの途中で状態価値関数を更新し続けるので、長い時間のかかるエピソードに対しても適用できる
- 収束速度の違いについて数学的な証明はないが、一般的にはTD法のほうが速い
- ただし、 α 一定の条件下では、TD法では誤差0に収束できない



モンテカルロ法の計算結果



TD法の計算結果

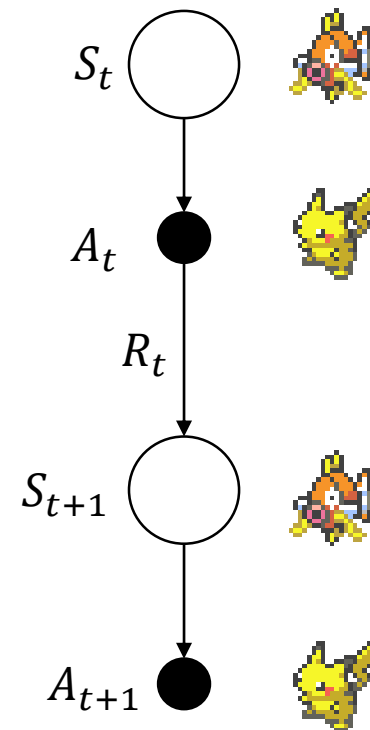


- TD予測にもとづいて, 最適方策を得る方法を考える
- 基本的には方策 π に従い(On-policy), たまに異なる行動を試してみる
- 行動価値関数に次の更新則を適用する:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$$

TD誤差から行動価値関数の更新量を決定

- エピソードの完了まで待たず, $Q(S_t, A_t)$ が更新されるたびに最大の $Q(S_t, A_t)$ を取るようの方策を改善していく
- このアルゴリズムは5つの情報($S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}$)を用いるので**SARSA法**と呼ぶ



1. 現在の行動価値関数が最大になるように(小さい確率でランダムに,) s で取る行動 A を選択
2. 次の状態 s' と, 得られる報酬 R を観測
3. 更新則 $Q(s_t, A_t) \leftarrow Q(s_t, A_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, A_{t+1}) - Q(s_t, A_t)]$
4. 1~3をエピソードが終わるまで繰り返し
5. 1~4を何エピソードか繰り返す

初期状態を設定

Q に基づいて, (小さい確率でランダムに,) s で取る行動 A を選択

エピソード終了まで各ステップについて繰り返し

行動 A を取り, 報酬 R と次状態 s' を観測

Q を最大にするように(小さい確率でランダムに), 次の行動 A を選択

$Q(s, A) \leftarrow Q(s, A) + \alpha[R + \gamma Q(s', A') - Q(s, A)]$

$s \leftarrow s', A \leftarrow A'$

SARSA法をpokemon_TD2.mに実装している. どのような実装になるか確認しよう.

サンプルプログラムpokemon_TD2.mの一部

% ステップサイズパラメータ

alpha = 0.2;

% イブシロン

e = 0.2;

% 状態価値関数

q = zeros(length(states),length(actions)); % 初期値は全て0

for i_q = 1:num % たくさん繰り返す

 s = max(states); % 初期状態の残りHP

 if rand() < e

 % 小さい確率eでランダムに行動を選択する

 a = randi(length(actions));

 else

 % 最大の価値を持つ行動を選択する

 [~,a] = max(q(states==s,:));

 end

 while s > 0

 next_s = battle(s,a);

 if rand() < e

 % 小さい確率eでランダムに行動を選択する

 next_a = randi(length(actions));

 else

 % 最大の価値を持つ行動を選択する

 [~,next_a] = max(q(states==s,:));

 end

 r = reward(next_s);

 q(states==s,actions==a) = q(states==s,actions==a) + alpha*(r + discount*q(states==next_s,actions==next_a) - q(states==s,actions==a));

 total_rewards(i_q) = total_rewards(i_q) + discount * r;

 s = next_s;

 a = next_a;

 steps(i_q) = steps(i_q) + 1;

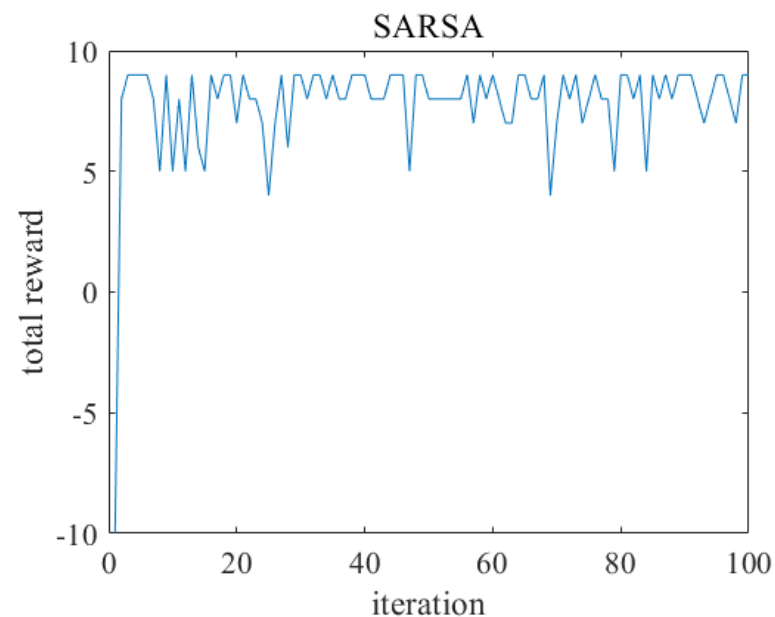
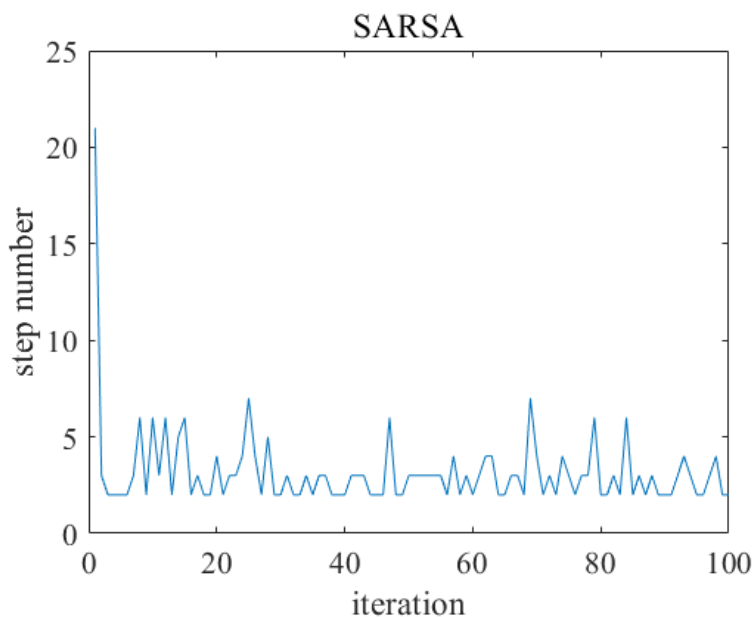
 end

end

行動価値関数にもとづき、
現時点で最適な行動を取る
小さい確率で
ランダムな行動を取る
(ϵ -Greedy)

エピソードが終了するのを待つことなく、
1ターンの行動が終了した時点で
現在と次状態の情報をもとに
行動価値関数を更新していく

- たいあたりのダメージを1に変更して計算した結果
- $\alpha = 0.2, \varepsilon = 0.2$ と設定している
- 100回しかエピソードを回していないものの, 方策が改善された結果, すでに良い報酬が得られるようになっている
- α, ε を一定値としているので, 最終的に方策が収束することはない



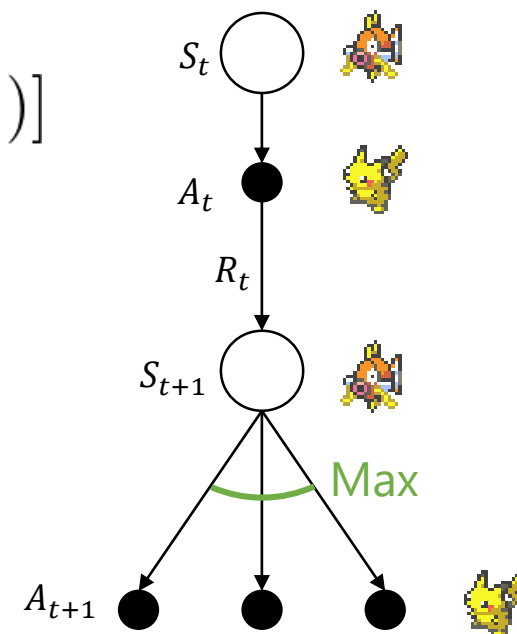
- SARSAより優れた方法として, **Q学習**が提案されている
- 現在の行動 A_t は方策に従うか, 小さい確率で新しい行動を試す(ϵ -Greedy)
- 行動 A_{t+1} は**方策に従わず(Off-policy)**, **最も行動価値関数が大きくなるものを選択**するものとする. これでSARSAよりも効率よく真の値に近づいていけるはず

- 行動価値関数の更新則は

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_{t+1} + \underbrace{\gamma \max_a Q(S_{t+1}, a)}_{\text{TD誤差を考えると、今後の価値が最大になるものを選ぶ}} - Q(S_t, A_t)]$$

TD誤差を考えると、今後の価値が最大になるものを選ぶ

- エピソードの完了まで待たず, $Q(S_t, A_t)$ が更新されるたびに最大の $Q(S_{t+1}, A_{t+1})$ を取るようの方策を改善していく



1. 現在の行動価値関数が最大になるように(小さい確率でランダムに), s で取る行動 A を選択
2. 次の状態 s' と, 得られる報酬 R を観測
3. 更新則 $Q(s_t, A_t) \leftarrow Q(s_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, A_t)]$
4. 1~3をエピソードが終わるまで繰り返し
5. 1~4を何エピソードか繰り返す

初期状態を設定

エピソード終了まで各ステップについて繰り返し

Q を最大にするように(小さい確率でランダムに), 状態 s における行動 A を選択

行動 A を取り, 報酬 R と次状態 s' を観測

$Q(s, A) \leftarrow Q(s, A) + \alpha[R + \gamma \max_a Q(s', a) - Q(s, A)]$

$s \leftarrow s'$

Q学習をpokemon_TD3.mlに実装している. どのような実装になるか確認しよう.

サンプルプログラムpokemon_TD3.mの一部

% ステップサイズパラメータ

alpha = 0.2;

% イプシロン

e = 0.2;

% 状態価値関数

q = zeros(length(states),length(actions)); % 初期値は全て0

for i_q = 1:num % たくさん繰り返す

s = max(states); % 初期状態の残りHP

while s > 0

if rand() < e

% 小さい確率eでランダムに行動を選択する

a = randi(length(actions));

else

% 最大の価値を持つ行動を選択する

[~,a] = max(q(states==s,:));

end

next_s = battle(s,a);

r = reward(next_s);

q(states==s,actions==a) = q(states==s,actions==a) + alpha*(r + discount*max(q(states==next_s,:)) - q(states==s,actions==a));

total_rewards(i_q) = total_rewards(i_q) + discount * r;

s = next_s;

steps(i_q) = steps(i_q) + 1;

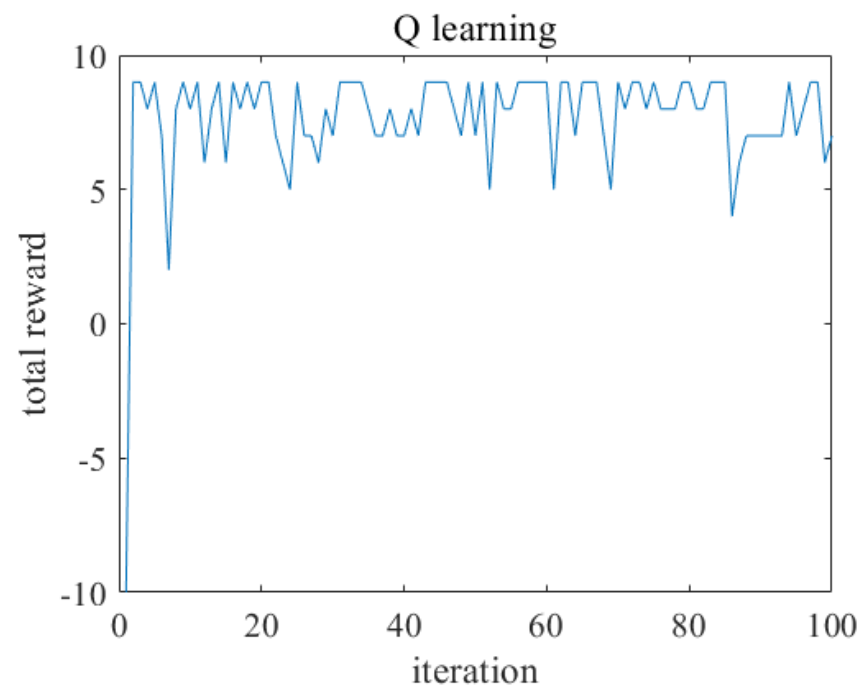
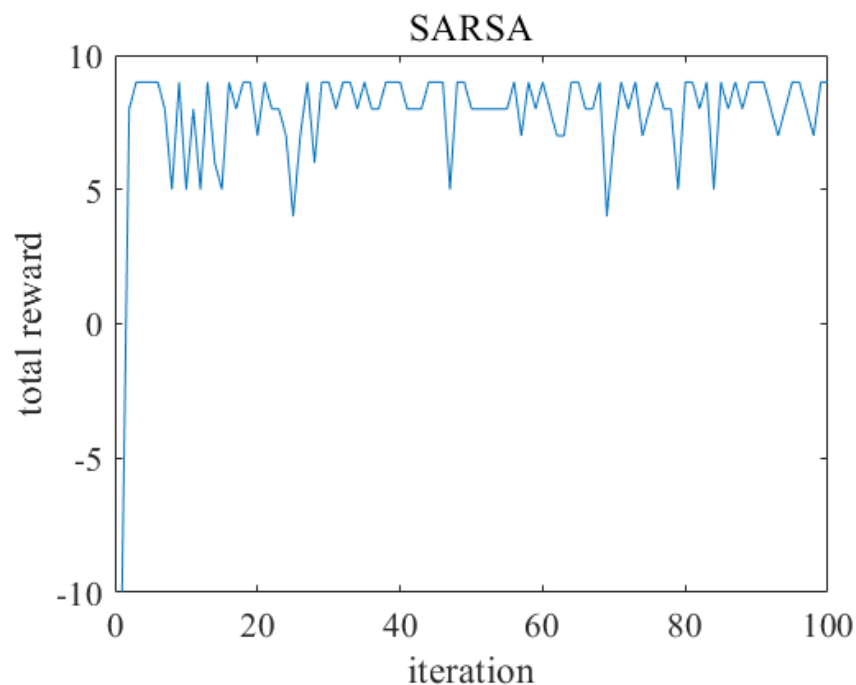
end

end

行動価値関数にもとづき、
現時点で最適な行動を取る
小さい確率で
ランダムな行動を取る
(ϵ -Greedy)

エピソードが終了するのを待つことなく、
1ターンの行動が終了した時点で
行動価値関数を更新していく

- いずれもたいあたりのダメージを1に変更して, $\alpha = 0.2, \varepsilon = 0.2$ と設定した結果
- 今回の問題設定では, いずれも遜色ない結果が得られている
 - 今回の実装ではSARSAもランダム要素を含み, 完全にオンポリシーになっていないことが影響
 - Q学習のほうが結果が悪くなることもある



- 環境モデルが未知でも使える強化学習のアルゴリズムとして、モンテカルロ法とTD法を紹介した
 - いずれも、環境モデルが完全に与えられていない状況で、価値関数を推定し最適方策を導ける
- モンテカルロ法では、多くのエピソードを収集し、平均値を取ることで価値関数を推定するが、エピソードが完了するまで推定値の更新は行われない
- TD法では、エピソードの完了を待つことなく、各ステップにおいてTD誤差を求め、価値関数を更新していく
 - **SARSA法**では、現在の推定方策にもとづいて(オンポリシー), ステップごとに行動価値関数を改善
 - **Q学習**では、現在の推定方策に依存することなく(オフポリシー), 効率よく行動価値関数を改善