AI基礎セミナー

第11回

機械学習（5回目:強化学習）

改訂履歴

日付	担当者	内容
2021/05/30	M.Takeda	Git 公開

目次

(1) はじめに

・ｾﾐﾅｰ11回目では、「強化学習」を取り上げます。
前半では、強化学習の概要と「ﾓﾃﾞﾙﾍﾞｰｽな手法での価値関数と方策改善」までを取り上げます。
後半では、「ﾓﾃﾞﾙﾌﾘｰな手法での価値関数と方策改善」を取り上げます。

・強化学習は、
「行動指針となる方策について、最終的な報酬を最大化するように、学習により最適化を行う」
というものですが、定量的には漸化式を解いたりと、少々ややこしいです。
然しながら、様々な機械学習の中では一番ＡＩらしいという印象があり、魅力的です。

・本資料で主に参考にしたものは以下のものです。
(1)「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto) (※1)
(2)「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井悦司)
(3)「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

(※1)この本を、本ｾﾐﾅｰでは「Sutton本」と呼ぶことにします。
「Sutton本」は強化学習入門者のﾊﾞｲﾌﾞﾙと呼ばれています。
英語の第2版は、以下のｻｲﾄでpdfﾌｧｲﾙをﾀﾞｳﾝﾛｰﾄﾞできます：
ｻｲﾄ⇒ https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

・掲載した Python コードは全て JupyterNotebook で動作確認済ですが、
Google が公開している「Google Colaboratory」で検証するのがお手軽です(※2)。
興味のある方は、添付したソースで検証してみてください。

(※2)「Google Colaboratory」を使用する方は、以下の手順でご利用ください：
(1) 予め以下のサイトでご自分の Google アカウントを用意しておきます：
(Gmail のアカウントをお持ちの場合、Google アカウントは既にお持ちの状態です)
「Google アカウントの作成 (https://support.google.com/accounts/answer/27441)」
(2) 「Google Colaboratory」へ手順(1)で作成したアカウントでログインします：
「Google Colaboratory (https://colab.research.google.com)」

・ご意見・ご感想など有りましたら、お寄せ下さい。

（表記法について）

・記号などの表記方法は、「Sutton本」に準じたものにしています。
以下に本資料に掲載している主な表記法をｱﾙﾌｧﾍﾞｯﾄ順に記します(※1)：

a ：行動値（aは "action"の略称）
A ：とりうる行動aの集合
$A(s)$ ：状態s でとりうる行動aの集合
$A^*(s)$ ：状態s でとりうる行動aの集合のうち価値関数が最大(=$V^*(s)$)となる行動aの集合
A_t ：時刻t における行動
$E_\pi(G_t \mid S_t=s)$ ：方策πの下で、時刻t の状態S_tがs の時に、時刻t での割引報酬和G_t の期待値
$E_\pi(R_{t+1} \mid S_t=s)$ ：方策πの下で、時刻t の状態S_tがs の時に、時刻t+1 での即時報酬R_{t+1} の期待値
G_t ：時刻t から将来にわたる割引報酬和
$I(\chi)$ ：条件χが成立する時1、それ以外の時0 を返す関数
M ：集合$A(s)$の要素数(濃度)
M^* ：集合$A^*(s)$の要素数(濃度)
$N_t(s)$ ：時刻tで、ｴﾋﾟｿｰﾄﾞ上での状態s への訪問回数
$P(G_t \mid S_t=s)$ ：時刻t の状態S_tがs の時、割引報酬和がG_tとなる確率
$P(\alpha \mid \beta)$ ：事象βが発生している下での事象αの発生確率（pは "probability"の略称）
$P(\chi)$ ：事象χの発生確率
$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$P_{ss'}^{ar}$ ：状態がs の時に、行動a を選択して状態がs'に遷移し報酬がr となる確率
P^π ：方策πの下で、ある状態から遷移先の状態への遷移確率ﾏﾄﾘｸｽ
$Q^*(s,a)$ ：状態s で、最適方策π^*の下で行動aをとる時の行動価値関数
$Q_\pi(s,a)$ ：状態s で行動a を取り、状態遷移後は方策πに従った時の行動価値関数
r ：報酬値（rは "rewards"の略称）
$R_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
R_t ：時刻t における報酬
R^π ：方策πの下での報酬期待値ﾍﾞｸﾄﾙ
s ：状態値（sは "status" の略称）
s' ：遷移先の状態値
s^n ：状態値、$s^n \in S$ （n=1～集合Sの要素数）
S ：取りうる状態値の集合
S_t ：時刻t における状態
t ：時刻または時点。手順の順番を表し、順序尺度または間隔尺度として使用(※2)
$V^*(s)$ ：状態がs の時に、最適方策π^* の下での状態価値関数（V は "value" の略称）
$V_t(s)$ ：時刻tで、状態s の状態価値関数
V_π ：方策πの下での価値ﾍﾞｸﾄﾙ
$V_\pi(s)$ ：方策πの下で、状態がs の時の状態価値関数($\{V_\pi\}_s = V_\pi(s)$)
α, β ：事象α、β
$\alpha \cap \beta$ ：事象αとβが同時発生する事象
γ ：割引率
ε ：ε-greedy法のﾊﾟﾗﾒｰﾀε
π ：方策（πは "policy" の略称）
π^* ：最適方策
$\pi(a|s)$ ：確率的方策(状態がs の時に、行動a をとる確率)
$\pi(s)$ ：決定論的方策(状態がs の時に、とる行動a)

（※1）集合や確率変数、およびそれを用いた確率を大文字(S，A，R，P等)で、
実際の要素や状態値、行動・報酬・確率を小文字(s,a,r,p 等)で表現します。
（※2）「尺度」はｾﾐﾅｰ8回目「前処理」を参照のこと。

(2) 強化学習とは

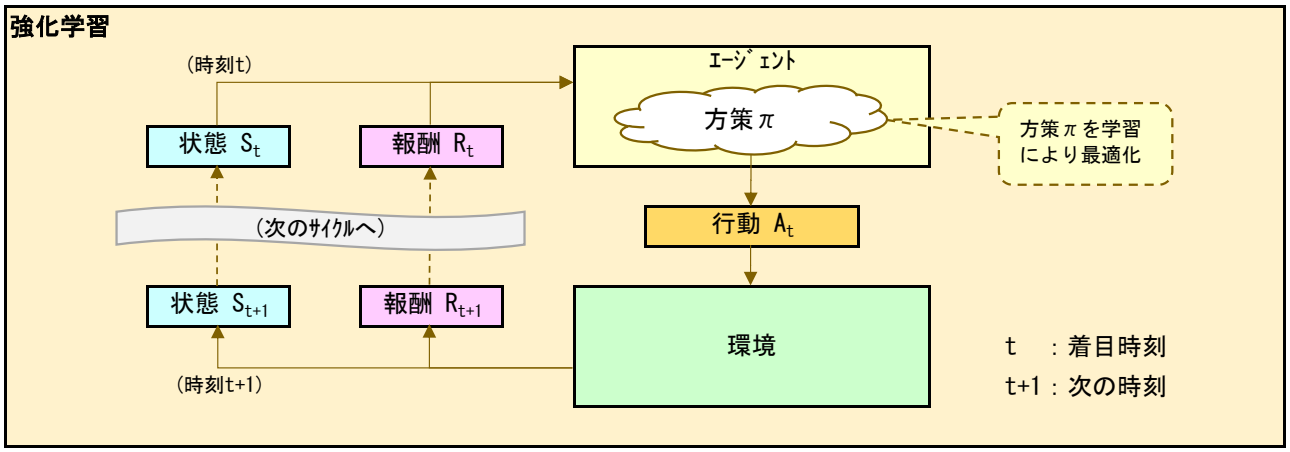
(2.1) 強化学習とは

・ｹﾞｰﾑ、業務改善、自動車運転などでは、
その時の状態を認知し、最終的な報酬を最大化するように、
次にどんな方策を講じるか判断して行動に移す・・・　ということを繰り返します。

・以下では、「環境」は制御対象の系(システム)のことです。
「ｴｰｼﾞｪﾝﾄ」は制御ﾌﾟﾛｸﾞﾗﾑのことで、環境の置かれた状態にもとづいて行動選択をしながら
環境を制御する主体です。ｴｰｼﾞｪﾝﾄの行動選択ﾙｰﾙを「方策」と呼びます。

・「強化学習（ｷｮｳｶｶﾞｸｼｭｳ、Reinforcement learning)」では、「ｴｰｼﾞｪﾝﾄ（Agent、制御器)」が
「環境（ｶﾝｷｮｳ、Environment)」の「状態（ｼﾞｮｳﾀｲ、State)」を認知し、
「報酬（ﾎｳｼｭｳ、Reward)」が最終的に最大化するように、
次にどんな「方策（ﾎｳｻｸ、Policy)」を講じるか判断して
「行動（ｺｳﾄﾞｳ、Action)」に移して「環境」に働きかける・・ということを繰り返しながら、
「環境」を制御する「方策」を学習します(図2.1)。

（図2.1）強化学習の図解



・こうした「ｴｰｼﾞｪﾝﾄ」と「環境」のやりとりを繰り返しつつ、最終的な「報酬」を最大化するように、
「ｴｰｼﾞｪﾝﾄ」の「方策」を最適化することが強化学習の目的であり、
そのｱﾙｺﾞﾘｽﾞﾑは以下のようになります。

強化学習のｱﾙｺﾞﾘｽﾞﾑ概要

(手順1)　「ｴｰｼﾞｪﾝﾄ」が、時刻t において「環境」に影響を及ぼす際に、
時刻t における環境の「状態 S_t」と「報酬 R_t」をみて、
「方策 π」に基づいて「行動 A_t」を選択して実行します。

(手順2)　環境は、ｴｰｼﾞｪﾝﾄが選んだ「行動 A_t」がもたらす影響に応じて
「報酬 R_{t+1}」をｴｰｼﾞｪﾝﾄに返すと同時に、
「状態 S_t」を「S_{t+1}」へ更新します。

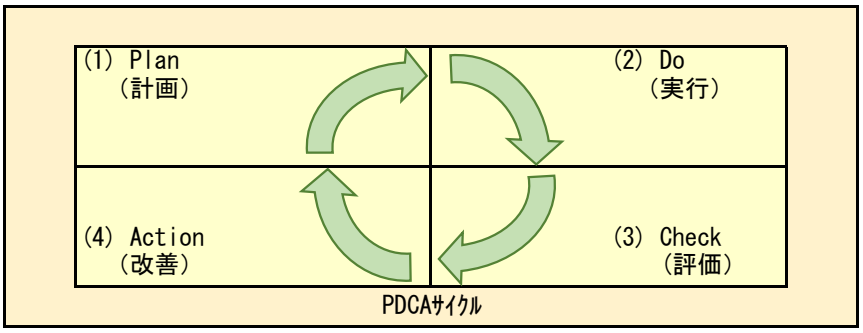
(手順3)　ｴｰｼﾞｪﾝﾄは、その時刻までの観測結果 S_t、A_t、R_{t+1}、S_{t+1} に基づいて
「方策 π」を改善します。

(手順4)　上記(手順1)～(手順3)を、ｴｰｼﾞｪﾝﾄの「方策 π」が最適なものに収束するまで
繰り返します。

（参考）

「強化学習」は、「PDCAサイクル（ﾋﾟｰﾃﾞｨｰｼｰｴｰｻｲｸﾙ、PDCA cycle、plan-do-check-act cycle)」と似ています。これは、「Plan（計画)」「Do（実行)」「Check（評価)」「Action（改善)」の四段階を繰り返すことにより、 業務を継続的に改善する手法です。

(1) Plan（計画）　：従来の実績や将来の予測などをもとにして業務計画を作成する。
(2) Do（実行）　：計画に沿って業務を行う。
(3) Check（評価）　：業務の実施が計画に沿っているかどうかを評価する。
(4) Action（改善）　：実施が計画に沿っていない部分を調べて改善をする。



【出典・参考】

強化学習⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社)
PDCAｻｲｸﾙ⇒ https://flhouse.co.jp/article/178/pdca
PDCAｻｲｸﾙ⇒ https://ja.wikipedia.org/wiki/PDCAサイクル

(3) 強化学習の数理ﾓﾃﾞﾙ

・「ﾏﾙｺﾌ決定過程（ﾏﾙｺﾌｹｯﾃｲｶﾃｲ、Markov decision process、MDP)」とは、
状態遷移が確率的に生じる動的ｼｽﾃﾑの確率ﾓﾃﾞﾙの一つで、ﾏﾙｺﾌ性をもつ確率過程のことをいいます。
これは、ﾛﾎﾞｯﾄ工学や自動制御、経済学、製造業を含む幅広い分野で用いられており、
強化学習では、環境を表す数理ﾓﾃﾞﾙになります。これを以下に解説します。

(3.1) 強化学習とﾏﾙｺﾌ決定過程

・「確率変数（ｶｸﾘﾂﾍﾝｽｳ、random variable)」とは、ある確率で値を取る変数のことです。
（例）　確率変数X：ｻｲｺﾛの目(1～6)、その確率P(X)(=1/6)

・確率変数X が時刻t の経過に伴って起こる変化を X(t) で表し、これを
「確率過程（ｶｸﾘﾂｶﾃｲ、stochastic process)」と呼びます。
（例）　ｻｲｺﾛをt回目に振って出た目の値X(t)（＝1～6)

・時刻t における状態が確率変数Ｓ(t) で与えられた時、次時刻t+1 における状態Ｓ(t+1) が、
時刻t での状態Ｓ(t) だけで決定され、それ以前の状態と無関係である場合、
この性質を「ﾏﾙｺﾌ性（ﾏﾙｺﾌｾｲ、Markov property)」と言います。

・確率ﾓﾃﾞﾙを作成するにあたり、ﾏﾙｺﾌ性を持つﾓﾃﾞﾙとなるように「状態」を定義します。
「状態」は、過去の情報を集約し、過去の関連情報を全て保持していることが理想的です。

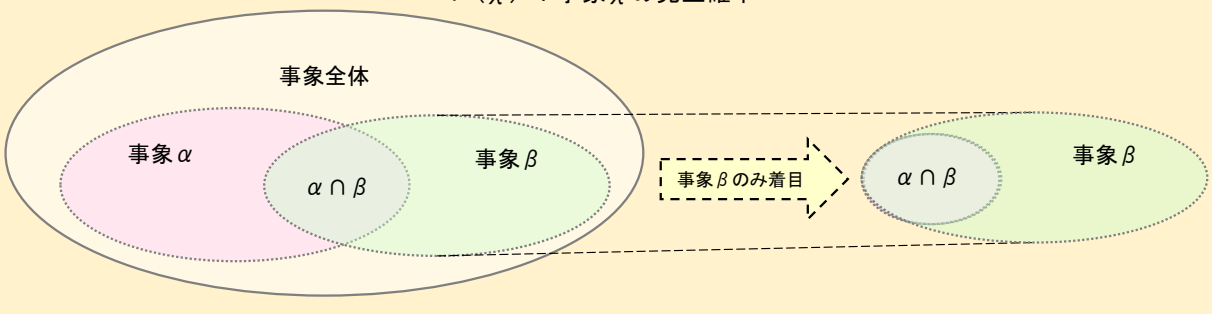
・強化学習で、「ｴｰｼﾞｪﾝﾄ（Agent、制御器)」は、
時刻t における「状態（ｼﾞｮｳﾀｲ、state）S_t」と「報酬（ﾎｳｼｭｳ、reward）R_t」をみて、
何らかの「方策（ﾎｳｻｸ、policy）π」に基づいて「行動(ｺｳﾄﾞｳ、action）A_t」を通して
「環境（ｶﾝｷｮｳ、Environment)」に働きかけます。
一方、環境は、ｴｰｼﾞｪﾝﾄの行動がもたらす影響に応じて新たな「報酬 R_{t+1}」を
ｴｰｼﾞｪﾝﾄに返すと同時に、「状態」を更新($S_t \Rightarrow S_{t+1}$)し、次時刻t+1 に移ります。
各変数を確率変数として扱い、一連の変化を確率過程として捉えると、
これは「ﾏﾙｺﾌ決定過程」に他なりません。

・ある事象βが起きているという条件の下で、別のある事象αが生起する確率を、
「条件βの下でのαの確率（probability of α under the condition β)」、あるいは単に
「条件付き確率（ｼﾞｮｳｹﾝﾂｷｶｸﾘﾂ、conditional probability)」といい、$P(\alpha|\beta)$と表現します。
「条件付き確率」は次式で計算できます。

条件付き確率

$$P(\alpha|\beta) = P(\alpha \cap \beta) \,/\, P(\beta) \qquad \cdots(式3\text{-}1)$$

α, β ：事象
$\alpha \cap \beta$ ：事象αとβが同時発生する事象
$P(\alpha|\beta)$ ：事象βが発生している下での事象αの発生確率
$P(\chi)$ ：事象χの発生確率



・条件付き確率を用いて、ﾏﾙｺﾌ決定過程を以下のように数式表現します(※1)。

ﾏﾙｺﾌ決定過程の数式表現 （その１：指定した(状態、報酬)に至る確率）

時刻t での(状態,行動)＝(s,a)の下で、
時刻t+1 において(状態,報酬)＝(s',r)となる確率 $P_{ss'}^{ar}$

$$P_{ss'}^{ar} = P(S_{t+1}=s', R_{t+1}=r \mid S_t=s, A_t=a) \quad \cdots(式3\text{-}2)$$

$P_{ss'}^{ar}$ ：時刻t での(状態,行動)＝(s,a)の下で、
時刻t+1 において(状態,報酬)＝(s',r)となる確率
S_t ：時刻t における「状態 (state)」
R_t ：時刻t における「報酬 (reward)」
A_t ：方策πに基づき、時刻t にとる「行動(action)」
r ：「報酬」の値
a ：「行動」の値
s, s' ：「状態」の値

(※1) これ以降では、確率変数やそれを用いた確率を大文字(S, A, R, P等)で、
実際の状態値や行動・報酬・確率を小文字(s,a,r,p 等)で表現します。

ﾏﾙｺﾌ決定過程の数式表現 （その２：状態遷移の確率と報酬）

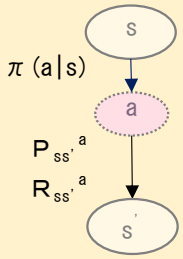
時刻t での(状態,行動)＝(s,a)の下で、
時刻t+1 において(状態)＝(s')となる確率 $P_{ss'}^{a}$ は、
(式3-2)を報酬rについて集計することにより算出され、状態s,行動a のみの関数となります。

$$P_{ss'}^{a} = P(S_{t+1}=s' \mid S_t=s, A_t=a) \quad \cdots(式3\text{-}3)$$
$$= \Sigma_r P_{ss'}^{ar}$$

状態s の下で行動a を取った時に、遷移先の状態がs' となるときの報酬$R_{ss'}^{a}$ は、
報酬rの期待値（＝報酬r にその発生確率をかけて集計したもの）になります(※2)。

$$R_{ss'}^{a} = \Sigma_r r \times \{P_{ss'}^{ar} / P_{ss'}^{a}\} \quad \cdots(式3\text{-}4)$$

状態s（時刻t）…
方策π(a|s) …
行動a …
遷移確率p …
報酬r …
状態s'（時刻t+1）…



(※2)「期待値」の説明は「ｾﾐﾅｰ6回目」を参照のこと。

(※) 参考までに、ﾏﾙｺﾌ性を持たない場合、
時刻t+1 において(状態,報酬)＝(s',r)となる確率 p(s',r)は、
過去の全ての状態・行動・報酬の関数として扱うと、次式になります。

$$p(s', r) = P(S_{t+1}=s', R_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, \cdots, r_1, s_0, a_0)$$

(3.2) ﾏﾙｺﾌ決定過程の様々な表現

・ﾏﾙｺﾌ決定過程は状態遷移であり、数式以外にも状態遷移表、状態遷移図、ﾊﾞｯｸｱｯﾌﾟ線図などで表現します。
尚、状態遷移表と状態遷移図は、UMLで標準化した記法が提示されています。

・ﾏﾙｺﾌ決定過程は、4つの要素（S， A， P， R）の組で表されます：

(表3.1) ﾏﾙｺﾌ決定過程の要素

要素	意味	集合と写像での表現	説明
S	取りうる状態値の集合	$\{s^1, s^2, \cdots, s^N\}$	N個の状態値 (s^n :n=1～N)
A	取りうる行動の集合	$\{a^1, a^2, \cdots, a^K\}$	K種類の行動 (a^k :k=1～K)
P	遷移関数	S×A×S→[0,1]	状態s で行動a を取った時、 状態s' への遷移確率 $P_{ss'}^{\ a} \in [0,1]$
R	報酬関数	S×A×S→実数	状態s で行動a を取った時、 状態s' への遷移時の即時報酬 $R_{ss'}^{\ a}$

(3.2.1) 状態遷移表

・ﾏﾙｺﾌ決定過程は、上記の表現を用いて
「状態遷移表 (ｼﾞｮｳﾀｲｾﾝｲﾋｮｳ、State Transition Table)」で表現できます。

(表3.2) 状態値が2種類$\{s^1, s^2\}$で、2種類の行動$\{a^1, a^2\}$をもつﾏﾙｺﾌ決定過程の状態遷移表例

No.	時刻t			時刻t+1	遷移確率(例) $P_{ss'}^{\ a}$		即時報酬(例) $R_{ss'}^{\ a}$
	状態s	方策π	行動a	状態s'			
①	s^1	$\pi(a^1\|s^1)$	a^1	s^1	0.3	(小計)	+1
②				s^2	0.7	1.0	−2
③		$\pi(a^2\|s^1)$	a^2	s^1	0.5	(小計)	+1
④				s^2	0.5	1.0	+2
⑤	s^2	$\pi(a^1\|s^2)$	a^1	s^1	0.6	(小計)	−1
⑥				s^2	0.4	1.0	+2
⑦		$\pi(a^2\|s^2)$	a^2	s^1	0.7	(小計)	+1
⑧				s^2	0.3	1.0	+1

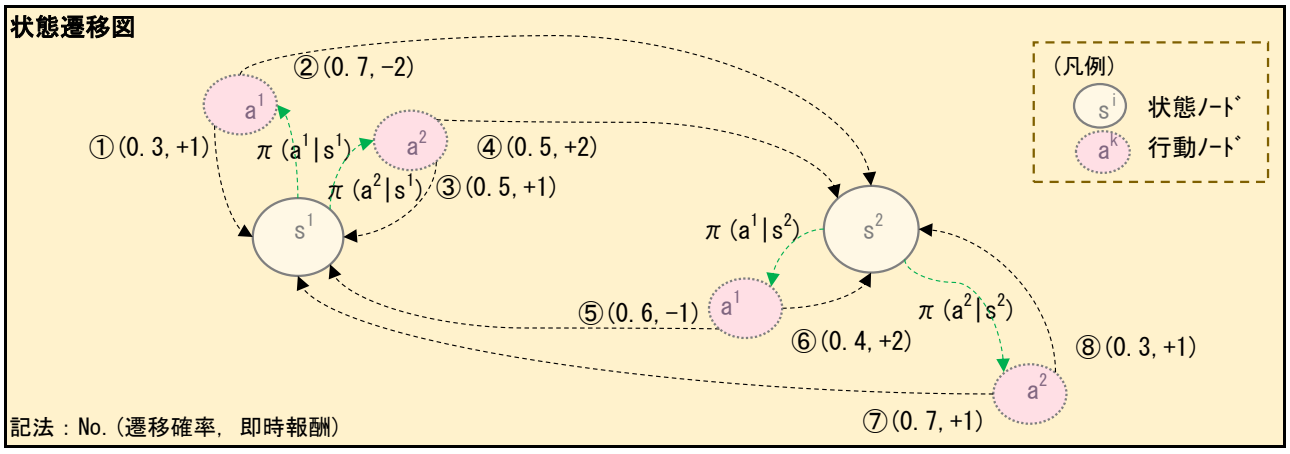
(※) ・上記表には「時刻tでの状態がsの時に行動aを選択する方策(確率)」π(a|s)も含めています。
・上記表で遷移確率 $P_{ss'}^{\ a}$ 、方策確率π(a|s) について、次の関係式があります：

$$\begin{cases} 1 = \sum_{s'} P_{ss'}^{\ a} & (s, s' \in \{s^1, s^2\},\ a \in \{a^1, a^2\}) \\ 1 = \sum_{k} \pi(a^k|s^i) & (k=1\sim2,\ i=1\sim2) \end{cases}$$

(3.2.2) 状態遷移図

・ﾏﾙｺﾌ決定過程の状態遷移表は、「状態ﾉｰﾄﾞ (state node)」と「行動ﾉｰﾄﾞ (action node)」からなる
「状態遷移図 (ｼﾞｮｳﾀｲｾﾝｲｽﾞ、State Transition Diagram)」でも表現できます。

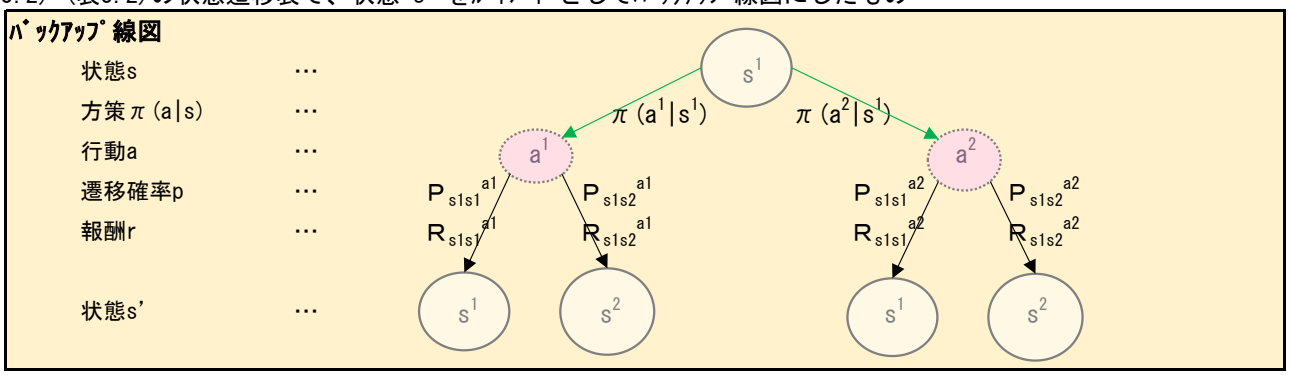
(図3.1) (表3.2)の状態遷移表を状態遷移図にしたもの



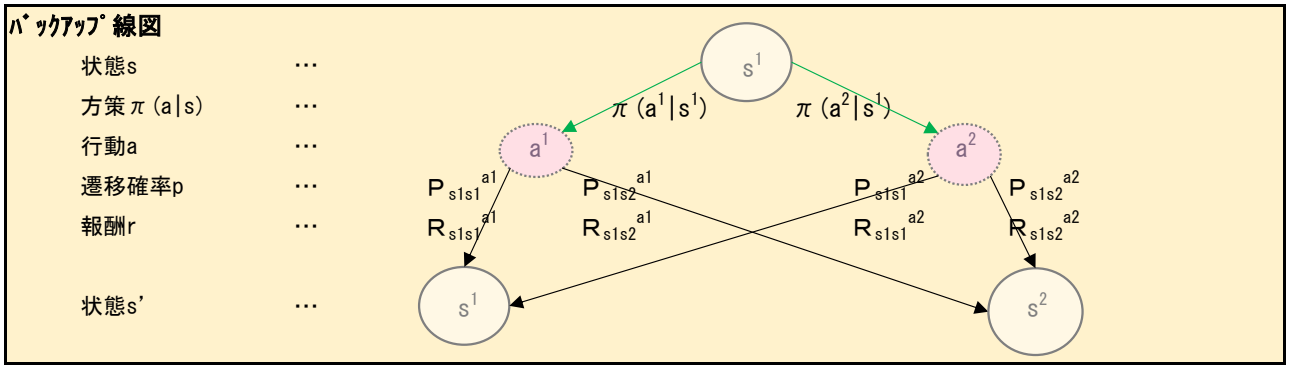
(3.2.3) ﾊﾞｯｸｱｯﾌﾟ線図とﾊﾞｯｸｱｯﾌﾟ木

・ﾏﾙｺﾌ決定過程の状態間の遷移には必ずｱｸｼｮﾝが介在します。
着目する状態を「ﾙｰﾄﾉｰﾄﾞ（根、root node)」、その状態で起こすｱｸｼｮﾝを「節（ｾﾂ、node)」、
ｱｸｼｮﾝの結果の状態を「ﾘｰﾌﾉｰﾄﾞ（葉、leaf node)」とする木構造で表現します。
これを「ﾊﾞｯｸｱｯﾌﾟ線図（ﾊﾞｯｸｱｯﾌﾟｾﾝｽﾞ、backup diagram)」と言います。

(図3.2)（表3.2)の状態遷移表で、状態 s^1 をﾙｰﾄﾉｰﾄﾞとしてﾊﾞｯｸｱｯﾌﾟ線図にしたもの

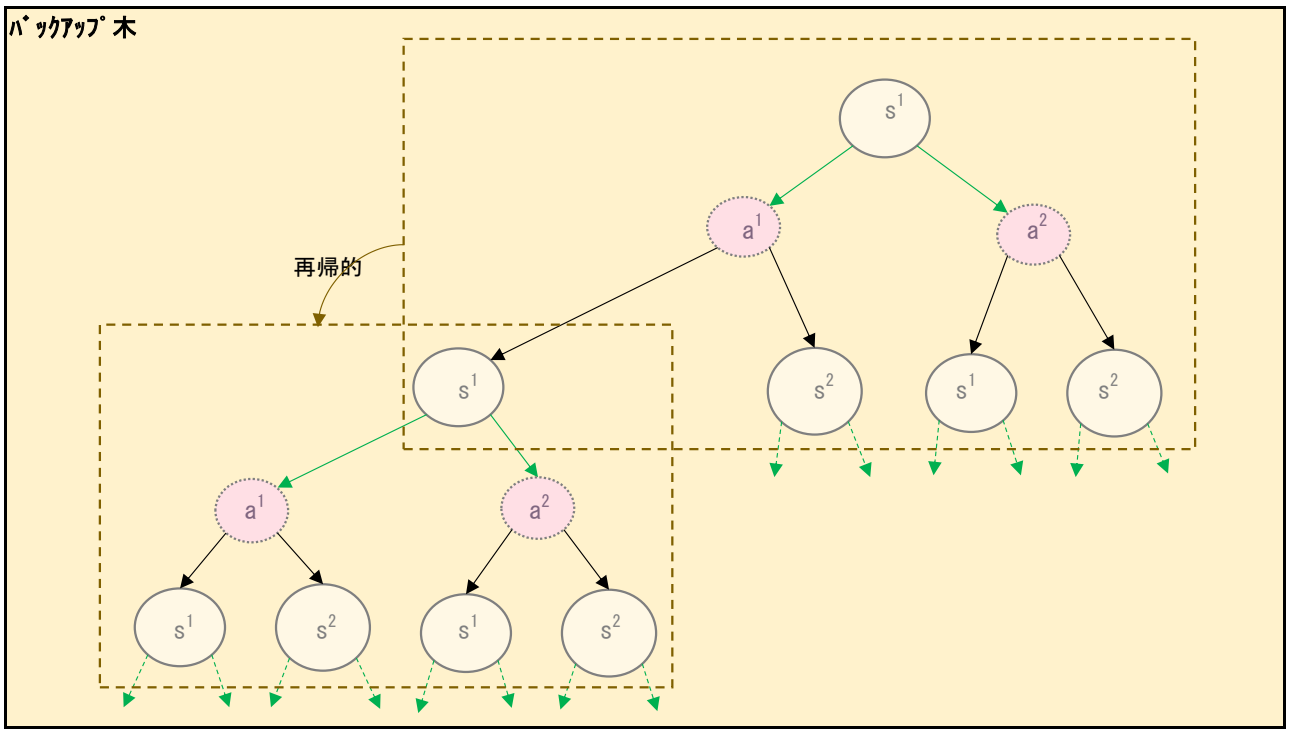


(図3.3)（図3.2)のﾊﾞｯｸｱｯﾌﾟ線図を、状態s から次の状態s' へ至る経路について集約したもの



ﾊﾞｯｸｱｯﾌﾟ線図(図3.2)は状態遷移とともに下方に伸びてゆき、ﾊﾞｯｸｱｯﾌﾟ木(図3.4)になりますが、
状態が同じものについては、再帰的な構造になるので、
有りうる状態を起点とした次の状態に至るまでのﾊﾞｯｸｱｯﾌﾟ線図(図3.3)で、既述は十分です。

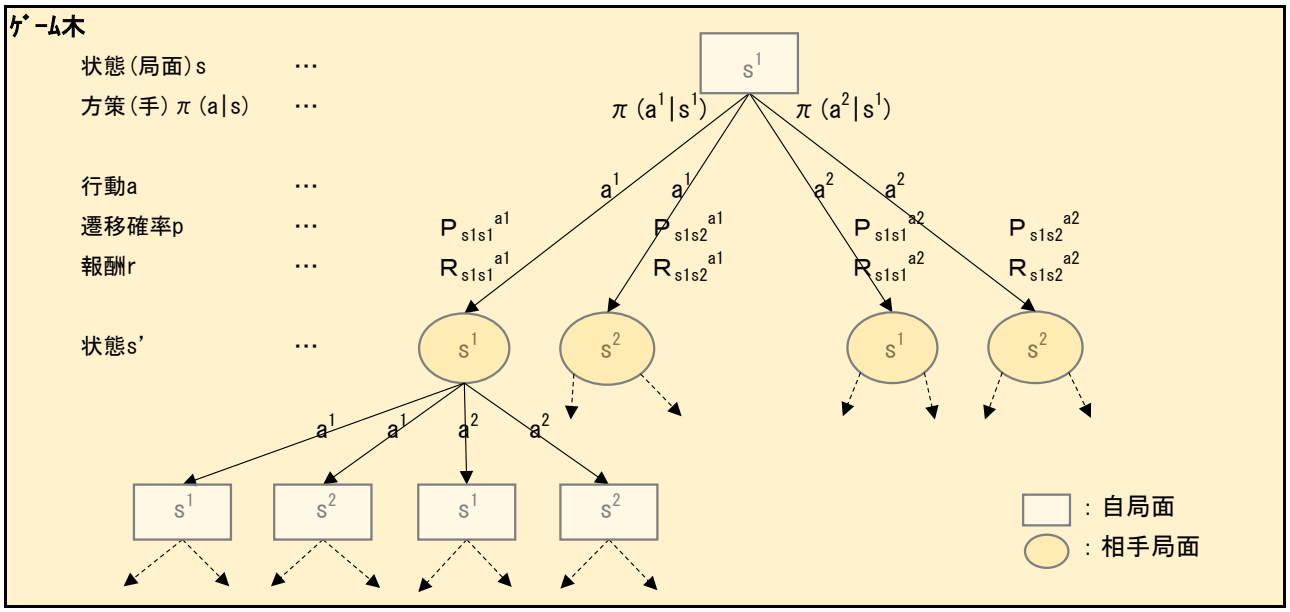
(図3.4)（図3.2)のﾊﾞｯｸｱｯﾌﾟ線図を後続の遷移も含めて木にし、再帰的であることを示したもの



(3.2.4) ｹﾞｰﾑ木

・ｹﾞｰﾑのように自分と対戦相手との2人のﾌﾟﾚｲﾔからなるｼｽﾃﾑの場合、
局面を「ﾉｰﾄﾞ（節、node)」に、ﾌﾟﾚｲﾔの手を「ｱｰｸ（枝、arc)」とし、
最初の局面を「ﾙｰﾄﾉｰﾄﾞ（根、root node)」、
最後の局面を「ﾘｰﾌﾉｰﾄﾞ（葉、leaf node)」とする木構造で表現し、
これを「ｹﾞｰﾑ木（ｹﾞｰﾑｷ、game tree)」と言います。
ﾏﾙｺﾌ決定過程との対応では、局面が状態Ｓに、ｱｰｸがｱｸｼｮﾝＡに相当します。
ｹﾞｰﾑ木も状態が同じものについては、再帰的な構造になります。

(図3.4)（表3.2)の状態遷移表を、(敢えて)ｹﾞｰﾑ木に表現してみたもの



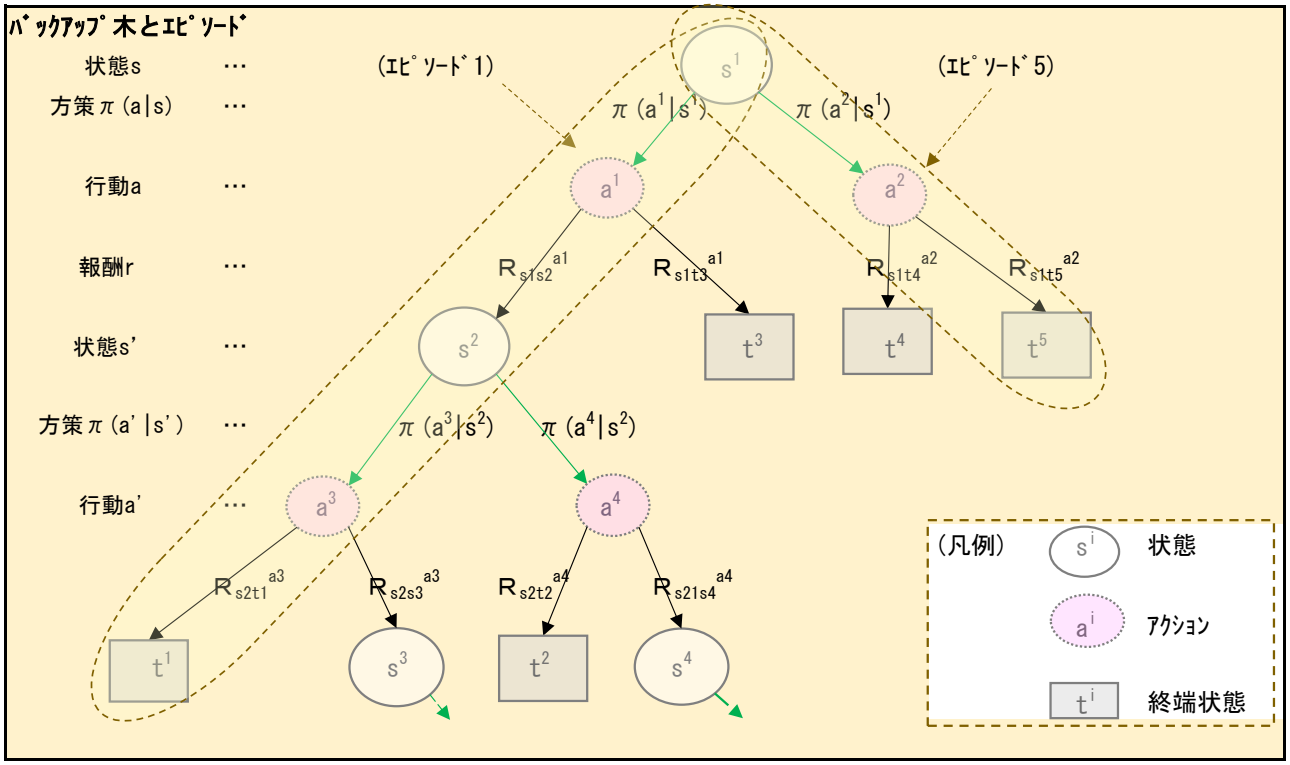
(3.3) 終端状態とｴﾋﾟｿｰﾄﾞ

・ﾊﾞｯｸｱｯﾌﾟ線図は状態遷移とともに下方に伸びてゆき、ﾊﾞｯｸｱｯﾌﾟ木になります。
状態遷移で、ある状態に到達するとそこから他の状態に遷移することのできない状態、
あるいは、そこを最終目標とする状態として他の状態に遷移することのない状態を
「終端状態（ｼｭｳﾀﾝｼﾞｮｳﾀｲ、Terminal state)」と言います。
「目的地に到着した」、「ｹﾞｰﾑｵｰﾊﾞｰとなった」などがこれにあたります。

・状態遷移で終端状態がある場合、状態遷移は有限ｽﾃｯﾌﾟで終了します。
この場合、ﾏﾙｺﾌ決定過程(MDP)は「ｴﾋﾟｿﾃﾞｨｯｸ（episodic)」であると言い、
初期状態から始まって状態遷移を繰り返して終端状態に至るまでの一連の状態遷移を
「ｴﾋﾟｿｰﾄﾞ（Episode)」と言います。

・一般的に、ｴﾋﾟｿﾃﾞｨｯｸなﾏﾙｺﾌ決定過程のﾊﾞｯｸｱｯﾌﾟ木には、複数のｴﾋﾟｿｰﾄﾞが存在します。

（図3.5）ﾊﾞｯｸｱｯﾌﾟ木とｴﾋﾟｿｰﾄﾞ



・ｴﾋﾟｿｰﾄﾞ上の一連の状態遷移の中で、
状態s に遷移したことを「状態s への訪問(visit)」と言います。

(3.4) 強化学習と方策

・方策πには「確率的方策」と「決定論的方策」の２種類あります。

方策πの種類

(1)「確率的方策 (ｶｸﾘｯﾃｷﾎｳｻｸ、Stochastic Policy)」

方策πを「ある状態s の時に行動a を取る確率」として、
確率「π(a|s)」で表現します。これを「確率的方策」と呼びます：

$$0 \leqq \pi(a|s) \leqq 1 \qquad (a \in A(s),\ s \in S)$$

状態s^i の時に行動a^k をとる確率的方策$\pi(a^k|s^i)$ について、次の関係式があります：

$$1 = \Sigma_k\ \pi(a^k|s^i) \qquad (k=1\sim\text{行動の種類},\ i=1\sim\text{状態の種類}) \qquad \cdots(式3\text{-}5)$$

(2)「決定論的方策 (ｹｯﾃｲﾛﾝﾃｷﾎｳｻｸ、Deterministic Policy)」

方策πを「ある状態s の時に必ず取る行動a」として
行動「π(s)」で表現します。これを「決定論的方策」と呼びます：

$$a = \pi(s) \qquad (a \in A(s),\ s \in S)$$

A(s) ：状態s でとりうる行動aの集合
S ：取りうる状態値の集合

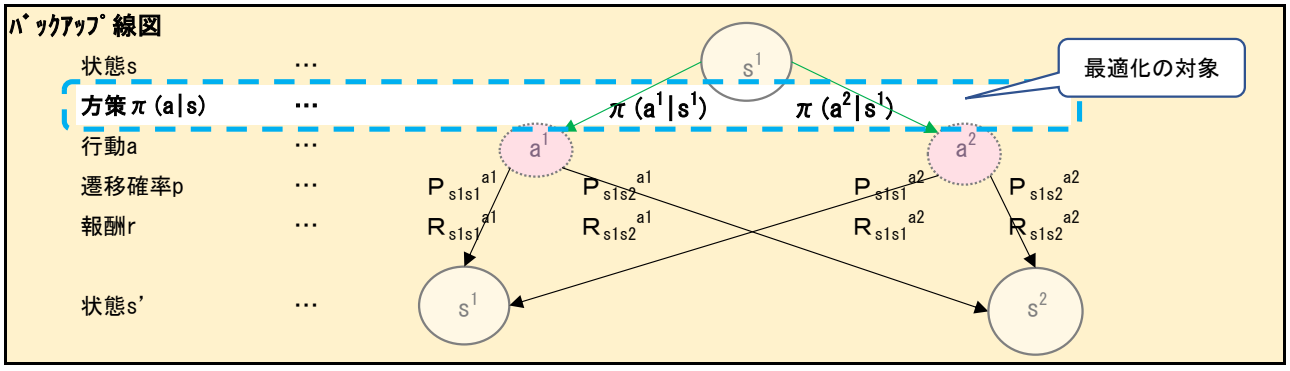
・強化学習では、「方策 (policy) π」を最適化します。
「方策πを最適化する」というのは、環境から受け取る最終的な報酬を最大化するように、
(1) 確率的方策の場合、状態がs の時に、行動a をとる確率π(a|s) を最適化することです。
(2) 決定論的方策の場合、状態がs の時に、とる行動a＝π(s) を最適化することです。

(表3.3) (表3.2)で最適化対象が「方策π」であることを示したもの

No.	時刻t 状態s	時刻t 方策π	時刻t 行動a	時刻t+1 状態s'	遷移確率(例) $P_{ss'}^{a}$		即時報酬(例) $R_{ss'}^{a}$
①	s^1	$\pi(a^1\|s^1)$	a^1	s^1	0.3	(小計)	+1
②				s^2	0.7	1.0	−2
③		$\pi(a^2\|s^1)$	a^2	s^1	0.5	(小計)	+1
④				s^2	0.5	1.0	+2
⑤	s^2	$\pi(a^1\|s^2)$	a^1	s^1	0.6	(小計)	−1
⑥				s^2	0.4	1.0	+2
⑦		$\pi(a^2\|s^2)$	a^2	s^1	0.7	(小計)	+1
⑧				s^2	0.3	1.0	+1

最適化の対象

(図3.6) (図3.3)で最適化対象が「方策π」であることを示したもの



・方策πの最適化をどのように行うのかを、これ以降では見てゆきます。

【出典・参考】

全般⇒ 「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)
強化学習、ﾏﾙｺﾌ決定過程⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社)
ﾏﾙｺﾌ決定過程⇒ https://ja.wikipedia.org/wiki/マルコフ決定過程
確率変数⇒ https://bellcurve.jp/statistics/course/6596.html
確率過程⇒ https://ja.wikipedia.org/wiki/確率過程
確率過程⇒ http://www.math.s.chiba-u.ac.jp/~yasuda/sysKOU/kakuKatei.pdf
条件付き確率⇒ https://bellcurve.jp/statistics/course/6438.html
条件付き確率⇒ https://ja.wikipedia.org/wiki/条件付き確率
ｹﾞｰﾑ木⇒ http://hp.vector.co.jp/authors/VA015468/platina/algo/2_1.html
ｹﾞｰﾑ木⇒「ｱﾙﾌｧ碁解体新書」(大槻知史 2017年07月 翔泳社)
決定論的・確率的方策⇒ https://qiita.com/sconeman/items/23662e7607c947208c2f

(4) 強化学習の目的関数

・ﾏﾙｺﾌ決定過程では、既述のように状態Ｓに応じて行動Ａを選択することで、報酬Ｒを獲得します。
この過程を繰り返しながら獲得した報酬を蓄えてゆきますが、この蓄えた報酬を最大化するように、
行動Ａの選択基準（これを方策πと言います）を最適化するのが強化学習の目的です。
この為に、以下に述べる状態価値関数あるいは行動価値関数を最大化することを、目的とします。

(4.1) 即時報酬R_tと割引報酬和G_t

・ﾏﾙｺﾌ決定過程で、ある行動Ａで環境に働きかけた直後に環境から返される報酬Ｒを
「即時報酬（ｿｸｼﾞﾎｳｼｭｳ、immediate reward)」と言います。
目的関数としては、一度の行動から得られる即時報酬よりも、
一連の行動で累積される最終的な報酬を評価できるものが必要です。

・ﾏﾙｺﾌ決定過程では、各時刻t における即時報酬R_tが得られ、即時報酬の時系列が作成されます。
即時報酬の時系列の累積和を「収益（ｼｭｳｴｷ、return)」と呼びます。

・強化学習の収益予測では、遠い将来程その即時報酬の予測値の寄与率を低くするような
「割引率（ﾜﾘﾋﾞｷﾘﾂ、discounting rate)」γ ($0 \leqq \gamma \leqq 1$) を取り入れた
「割引報酬和（ﾜﾘﾋﾞｷﾎｳｼｭｳﾜ, discounted return)」G_t が用いられます。
ｴｰｼﾞｪﾝﾄは「割引報酬和」G_t を最大化するように行動Ａを選択します。

割引報酬和

$$G_t \equiv R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \quad \cdots(式4\text{-}1)$$

$$= \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \quad \cdots(式4\text{-}2)$$

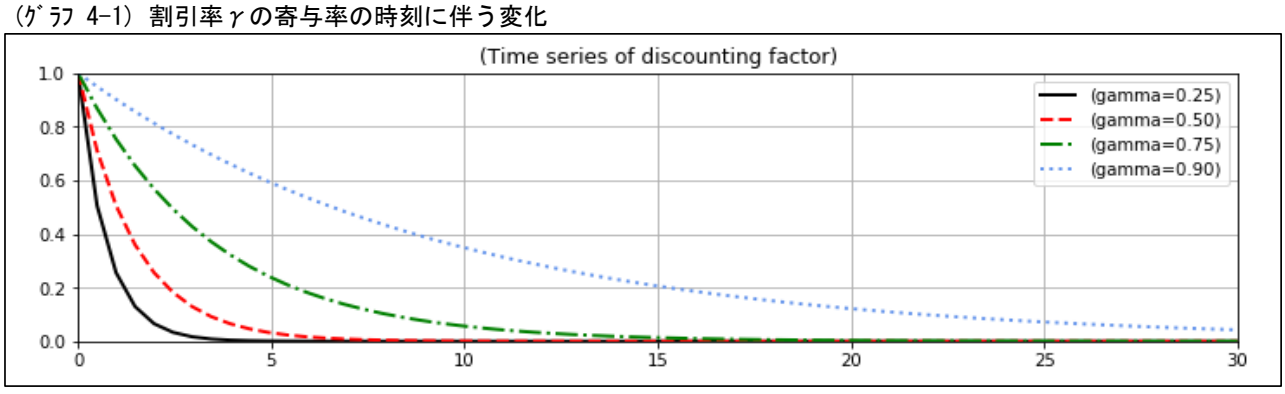
G_t ：時刻t から将来にわたる割引報酬和
k ：時刻t からの相対時刻（$1 \leqq k < \infty$）
γ ：割引率（$0 \leqq \gamma \leqq 1$）
γ^{k-1} ：時刻t+k における即時報酬の割引報酬和に対する寄与率
R_{t+k} ：時刻t+k における即時報酬

・「割引報酬和」G_t の計算式（式4-1）から次の漸化式が成り立ちます。

$$G_t = R_{t+1} + \gamma G_{t+1} \quad \cdots(式4\text{-}3)$$

・「割引率γ」が大きいほど、割引率による減衰が緩やかになり、遠い将来の報酬の予測値が尊重され、
「割引率γ」が小さいほど、割引率による減衰が急になり、近い将来の報酬の予測値が尊重されます。
「割引率γ」＝０ だと、即時報酬のみを報酬和とし、将来の報酬を無視したものになります。
その様子を(ｸﾞﾗﾌ 4-1)に示します。

(ｸﾞﾗﾌ 4-1) 割引率γの寄与率の時刻に伴う変化



(4.2) 状態価値関数$V_\pi(s)$とﾍﾞﾙﾏﾝ方程式

・時刻t の状態がs の時、ある方策πに従って、ﾏﾙｺﾌ決定過程による状態遷移を繰り返す結果、
期待できる収益を「状態価値関数（ｼﾞｮｳﾀｲｶﾁｶﾝｽｳ、State-value Function)」と言い$V_\pi(s)$ で表します。

・状態価値関数$V_\pi(s)$は、時刻t の状態がs の時に方策πに従った結果の割引報酬和G_t の期待値として、
次式で定義します(※1)：

方策πの下で、時刻t の状態がs の時の状態価値関数

$$V_\pi(s) \equiv E_\pi(G_t \mid S_t=s) \quad \cdots(式4\text{-}4)$$

$$= \sum_t G_t \times P(G_t \mid S_t=s) \quad \cdots(式4\text{-}5)$$

π ：方策
s ：状態値
$V_\pi(s)$ ：状態がs の時の状態価値関数
G_t ：時刻t から将来にわたる割引報酬和
S_t ：時刻t における状態
$E_\pi(G_t \mid S_t=s)$ ：時刻t の状態がs の時、時刻t での割引報酬和G_t の期待値
$P(G_t \mid S_t=s)$ ：時刻t の状態がs の時、割引報酬和がG_tとなる確率

(※1)「期待値」についてはｾﾐﾅｰ6回目を参照のこと。

・強化学習の目的は、
「状態価値関数$V_\pi(s)$ を全ての状態s について最大化する方策πを見つけること」
であり、これが強化学習の目的関数になります。

・割引報酬和の漸化式(式4-3)をもとに、状態価値関数$V_\pi(s)$ は以下のように整理されます(※2)。
これを「状態価値関数のﾍﾞﾙﾏﾝ方程式 (Bellman equation)」と言い、
状態s と遷移先の状態s' をつなぐ状態価値関数$V_\pi(s)$ の漸化式を与えます。

状態価値関数のﾍﾞﾙﾏﾝ方程式 (Bellman equation)

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{\ a} \times \{R_{ss'}^{\ a} + \gamma V_\pi(s')\} \quad \cdots(式4\text{-}6)$$

あるいは、(式3-4)から $P_{ss'}^{\ a}$と$R_{ss'}^{\ a}$を、$P_{ss'}^{\ ar}$とr に置き換えて

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} \sum_r P_{ss'}^{\ ar} \times \{r + \gamma V_\pi(s')\} \quad \cdots(式4\text{-}6')$$

$V_\pi(s)$ ：状態がs の時に、方策πの下での状態価値関数
$V_\pi(s')$ ：状態がs'の時に、方策πの下での状態価値関数
$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策(確率)
$P_{ss'}^{\ a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$P_{ss'}^{\ ar}$ ：状態がs の時に、行動a を選択して状態がs'に遷移し報酬がr となる確率
$R_{ss'}^{\ a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
r ：「報酬」の値

・「状態価値関数のﾍﾞﾙﾏﾝ方程式」は、
状態がs の時の状態価値関数$V_\pi(s)$が、
『状態s で行動a を取ったことで次の状態s' へ遷移する時の即時報酬$R_{ss'}^{\ a}$ と、
次の状態s' が持つ状態価値関数$V_\pi(s')$ に割引率γを掛けた値との和 $\{R_{ss'}^{\ a} + \gamma V_\pi(s')\}$
の期待値である』ことを意味しています。

(※2)状態価値関数のﾍﾞﾙﾏﾝ方程式(式4-6)は、

割引報酬和についての漸化式(式4-3)の両辺の期待値を取ることにより、以下のように導出されます：

・割引報酬和についての漸化式(式4-3)

$$G_t = R_{t+1} + \gamma G_{t+1} \quad \cdots(式4\text{-}3)再掲$$

・この式を用いて、割引報酬和についての漸化式(式4-3)の期待値は以下のようになります：

$$E_\pi(G_t \mid S_t = s)$$
$$= E_\pi(R_{t+1} + \gamma G_{t+1} \mid S_t = s)$$
$$= E_\pi(R_{t+1} \mid S_t = s) + \gamma E_\pi(G_{t+1} \mid S_t = s) \quad \cdots(式4\text{-}7)$$

・(式4-7)左辺は、状態価値関数の定義(式4-4)のとおり：

$$E_\pi(G_t \mid S_t = s) \quad = V_\pi(s) \quad \cdots(式4\text{-}4)再掲$$

・(式4-7)右辺第一項は、
時刻tにおける状態s を出発点とした時刻t+1 での報酬R_{t+1}の期待値ですが、
報酬$R_{ss'}^a$を得る確率が、方策確率$\pi(a|s)$と$P_{ss'}^a$との積となることから、
期待値の定義により、

$$E_\pi(R_{t+1} \mid S_t = s)$$
$$= \Sigma_a \pi(a|s) \Sigma_{s'} P_{ss'}^a \times R_{ss'}^a \quad \cdots(式4\text{-}8)$$

・(式4-7)右辺第二項は、
時刻tにおける状態s を出発点とした時刻t+1 での割引報酬和G_{t+1}の期待値に
割引率γをかけたものです。
割引報酬和G_{t+1} は、時刻t+1における状態s' での状態価値関数$V_\pi(s')$を
s' について集計して計算します。
状態価値関数が$V_\pi(s')$となる確率が、方策確率$\pi(a|s)$と$P_{ss'}^a$との積の行動a についての
総和となることから、期待値の定義により、

$$\gamma E_\pi(G_{t+1} \mid S_t = s)$$
$$= \gamma \Sigma_{s'} \{\Sigma_a \pi(a|s) P_{ss'}^a\} \times V_\pi(s')$$
$$= \gamma \Sigma_a \pi(a|s) \Sigma_{s'} P_{ss'}^a \times V_\pi(s') \quad \cdots(式4\text{-}9)$$

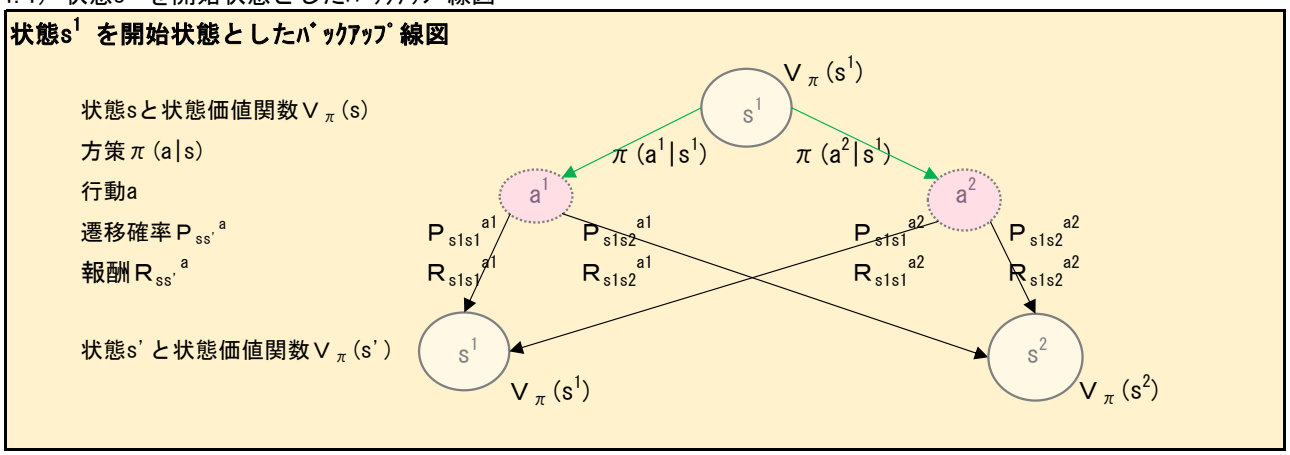
・以上をまとめることにより、「状態価値関数のﾍﾞﾙﾏﾝ方程式」(式4-6)を得ます。

$$V_\pi(s) = \Sigma_a \pi(a|s) \Sigma_{s'} P_{ss'}^a \times \{R_{ss'}^a + \gamma V_\pi(s')\} \quad \cdots(式4\text{-}6)再掲$$

G_t ：時刻t から将来にわたる割引報酬和
R_{t+k} ：時刻t+k における即時報酬
S_t ：時刻t の状態
$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策(確率)
$R_{ss'}^a$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
$P_{ss'}^a$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$V_\pi(s)$ ：状態がs の時に、方策πの下での価値関数
$V_\pi(s')$ ：状態がs'の時に、方策πの下での価値関数
$E_\pi(R_{t+1} \mid S_t = s)$ ：時刻t の状態がs の時に、時刻t+1 での即時報酬R_{t+1} の期待値
$E_\pi(G_t \mid S_t = s)$ ：時刻t の状態がs の時に、時刻t での割引報酬和G_t の期待値
$E_\pi(G_{t+1} \mid S_t = s)$ ：時刻t の状態がs の時に、時刻t+1 での割引報酬和G_{t+1} の期待値

・状態価値関数$V_\pi(s)$を、以下の(図4.1)を見ながら計算してみましょう。

(図4.1) 状態s^1 を開始状態としたﾊﾞｯｸｱｯﾌﾟ線図



上記(図4.1)の例で、状態価値関数を(式4-6)で計算すると、以下のようになります：

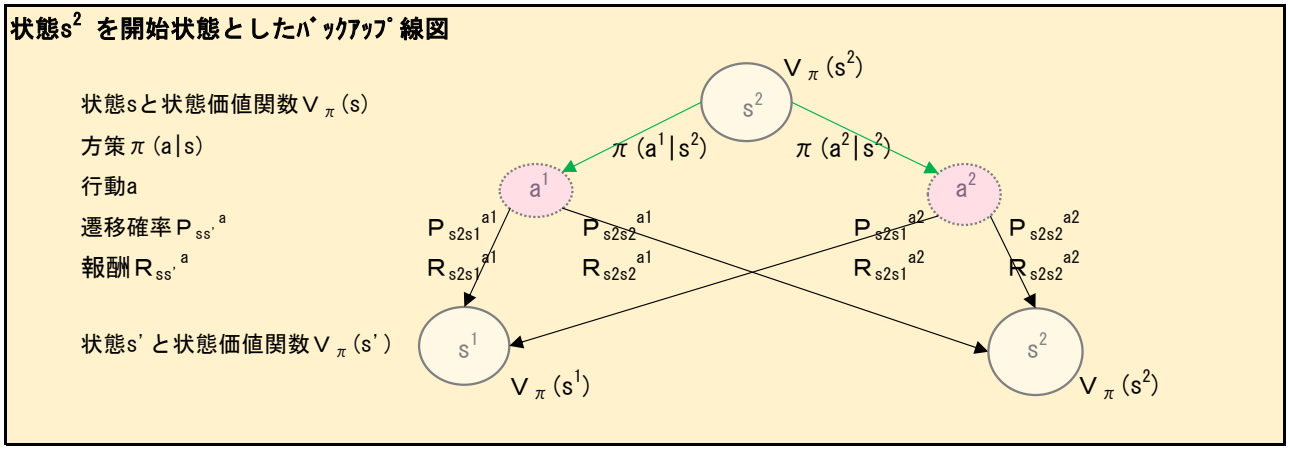
$$V_\pi(s^1) = \Sigma_a \pi(a|s^1) \Sigma_{s'} P_{s1s'}^{a} \times \{R_{s1s'}^{a} + \gamma V_\pi(s')\}$$

$$= \pi(a^1|s^1) \left[P_{s1s1}^{a1} \times \{R_{s1s1}^{a1} + \gamma V_\pi(s^1)\} + P_{s1s2}^{a1} \times \{R_{s1s2}^{a1} + \gamma V_\pi(s^2)\} \right]$$

$$+ \pi(a^2|s^1) \left[P_{s1s1}^{a2} \times \{R_{s1s1}^{a2} + \gamma V_\pi(s^1)\} + P_{s1s2}^{a2} \times \{R_{s1s2}^{a2} + \gamma V_\pi(s^2)\} \right]$$

・上記(図4.1)では、状態s^1 を開始状態とした図になっていますが、
(表3.2)の状態遷移表を見ると、状態s^2 を開始状態としたﾊﾞｯｸｱｯﾌﾟ線図(図4.2)も必要です。

(図4.2) 状態s^2 を開始状態としたﾊﾞｯｸｱｯﾌﾟ線図



上記(図4.2)の例で、状態価値関数を(式4-6)で計算すると、以下のようになります：

$$V_\pi(s^2) = \Sigma_a \pi(a|s^2) \Sigma_{s'} P_{s2s'}^{a} \times \{R_{s2s'}^{a} + \gamma V_\pi(s')\}$$

$$= \pi(a^1|s^2) \left[P_{s2s1}^{a1} \times \{R_{s2s1}^{a1} + \gamma V_\pi(s^1)\} + P_{s2s2}^{a1} \times \{R_{s2s2}^{a1} + \gamma V_\pi(s^2)\} \right]$$

$$+ \pi(a^2|s^2) \left[P_{s2s1}^{a2} \times \{R_{s2s1}^{a2} + \gamma V_\pi(s^1)\} + P_{s2s2}^{a2} \times \{R_{s2s2}^{a2} + \gamma V_\pi(s^2)\} \right]$$

・上記の計算例のように、「状態価値関数のﾍﾞﾙﾏﾝ方程式」を全ての状態$s \in S$について記述すると、
$V_\pi(s)$ についての連立方程式が作成されます。
これを解くことにより状態価値関数$V_\pi(s)$ を解くことができます。

(※ 現実的には動的計画法などで計算します。
具体的には「(7) ﾓﾃﾞﾙﾍﾞｰｽな手法での価値関数と方策改善」を参照のこと)

(4.3) 行動価値関数$Q_\pi(s,a)$とﾍﾞﾙﾏﾝ方程式

・状態がs の時、特定の行動a を取り、状態遷移後は方策πに従って行動を選択する結果、
期待できる収益を「行動価値関数 (ｺｳﾄﾞｳｶﾁｶﾝｽｳ、Action-value function)」と言い$Q_\pi(s,a)$で表します。

・行動価値関数は、「状態s で特定の行動a を取ったことで次の状態s' へ遷移する時の即時報酬$R_{ss'}^{a}$と、
遷移先の状態s' の状態価値関数$V_\pi(s')$に割引率γを掛けた値との和$\{R_{ss'}^{a}+\gamma V_\pi(s')\}$の期待値」
として、次式で定義されます：

状態がs の時、特定の行動a を取り、状態遷移後は方策πに従った時の行動価値関数

$$Q_\pi(s,a) \equiv \Sigma_{s'} P_{ss'}^{a} \times \{R_{ss'}^{a}+\gamma V_\pi(s')\} \quad \cdots(式4\text{-}10)$$

a ：状態がs の時に取る行動
s' ：状態がs で行動a を取って遷移した状態値
$Q_\pi(s,a)$ ：状態がs で行動a を取って、状態遷移後は方策πに従った時の行動価値関数
$P_{ss'}^{a}$ ：状態がs で行動a を取って状態がs'に遷移する確率
$R_{ss'}^{a}$ ：状態がs で行動a を取って状態がs'に遷移した時の報酬
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
$V_\pi(s')$ ：遷移先の状態s' の状態価値関数

・状態価値関数$V_\pi(s)$ の(式4-6)を、行動価値関数$Q_\pi(s,a)$で書き直すと、(式4-11)になります。
これは状態価値関数$V_\pi(s)$が、方策πの下での行動価値関数$Q_\pi(s,a)$の期待値であることを意味します。

状態価値関数$V_\pi(s)$と行動価値関数$Q_\pi(s,a)$の関係

$$V_\pi(s) = \Sigma_a \pi(a|s) \times Q_\pi(s,a) \quad \cdots(式4\text{-}11)$$

a ：状態がs の時に取る行動
$V_\pi(s)$ ：状態がs の時、方策πの下での状態価値関数
$Q_\pi(s,a)$ ：状態がs で行動a を取り、状態遷移後は方策πに従った時の行動価値関数
$\pi(a|s)$ ：状態がs で行動a を選択する(方策)確率

・(式4-10)の状態価値関数$V_\pi(s')$ の項に(式4-11)の関係を適用して、状態価値関数を行動価値関数に
置き換えることにより、次の「行動価値関数のﾍﾞﾙﾏﾝ方程式 (Bellman equation)」が導かれます。
これは、状態s と遷移先の状態s' をつなぐ行動価値関数$Q_\pi(s,a)$の漸化式を与えます。

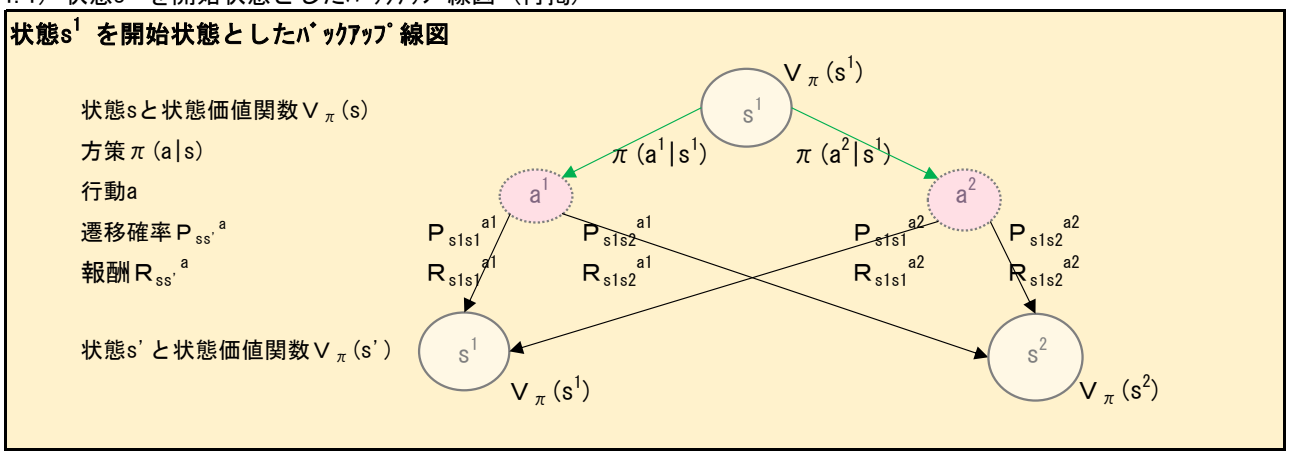
行動価値関数のﾍﾞﾙﾏﾝ方程式

$$Q_\pi(s,a) = \Sigma_{s'} P_{ss'}^{a} \times \left[R_{ss'}^{a}+\gamma \{\Sigma_{a'} \pi(a'|s') \times Q_\pi(s',a')\}\right] \quad \cdots(式4\text{-}12)$$

$Q_\pi(s,a)$ ：状態がs で行動a を取り、状態遷移後は方策πに従った時の行動価値関数
$Q_\pi(s',a')$ ：状態がs'で行動a'を取り、状態遷移後は方策πに従った時の行動価値関数
$P_{ss'}^{a}$ ：状態がs で行動a を取り、状態がs'に遷移する確率
$R_{ss'}^{a}$ ：状態がs で行動a を取り、状態がs'に遷移した時の報酬
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
$\pi(a'|s')$ ：状態がs' の時に、行動a' を選択する(方策)確率

・では、行動価値関数を、既出の(図4.1)を見ながら計算してみましょう。

(図4.1) 状態s^1 を開始状態としたﾊﾞｯｸｱｯﾌﾟ線図 (再掲)



上記(図4.1)の例で、行動価値関数を(式4-10)で計算すると、以下のようになります:

$$Q_\pi(s^1,a^1) = \Sigma_{s'} P_{s1s'}^{a1} \times \{R_{s1s'}^{a1} + \gamma V_\pi(s')\}$$
$$= P_{s1s1}^{a1} \times \{R_{s1s1}^{a1} + \gamma V_\pi(s^1)\} + P_{s1s2}^{a1} \times \{R_{s1s2}^{a1} + \gamma V_\pi(s^2)\}$$
$$Q_\pi(s^1,a^2) = \Sigma_{s'} P_{s1s'}^{a2} \times \{R_{s1s'}^{a2} + \gamma V_\pi(s')\}$$
$$= P_{s1s1}^{a2} \times \{R_{s1s1}^{a2} + \gamma V_\pi(s^1)\} + P_{s1s2}^{a2} \times \{R_{s1s2}^{a2} + \gamma V_\pi(s^2)\}$$

同じ(図4.1)の例での状態価値関数の計算結果:

$$V_\pi(s^1) = \Sigma_a \pi(a|s^1) \ \Sigma_{s'} P_{s1s'}^{a} \times \{R_{s1s'}^{a} + \gamma V_\pi(s')\}$$
$$= \pi(a^1|s^1) \ [\ P_{s1s1}^{a1} \times \{R_{s1s1}^{a1} + \gamma V_\pi(s^1)\} + P_{s1s2}^{a1} \times \{R_{s1s2}^{a1} + \gamma V_\pi(s^2)\}\ ]$$
$$+ \pi(a^2|s^1) \ [\ P_{s1s1}^{a2} \times \{R_{s1s1}^{a2} + \gamma V_\pi(s^1)\} + P_{s1s2}^{a2} \times \{R_{s1s2}^{a2} + \gamma V_\pi(s^2)\}\ ]$$

と比較すると、(式4-11)の関係式が成り立っていることがわかります:

$$V_\pi(s^1) = \Sigma_a \pi(a|s^1) \times Q_\pi(s^1,a)$$
$$= \pi(a^1|s^1) \times Q_\pi(s^1,a^1) + \pi(a^2|s^1) \times Q_\pi(s^1,a^2)$$

(4.4) 最適価値関数$V^*(s)$、$Q^*(s,a)$

・任意の状態s(∈Ｓ)に対して状態価値関数の最大値$V^*(s)$を与える方策πを
「最適方策 (ｻｲﾃｷﾎｳｻｸ、optimal policy)」と言います。
「最適方策」πは一般的には複数あり、これを最適方策群π^* で表します。
また、最適方策群π^* の下で状態s で取る行動a^* の集合を$A^*(s)$で表します。

・最適方策群π^*は、
「最適状態価値関数 (ｻｲﾃｷｼﾞｮｳﾀｲｶﾝｽｳ、optimal state-value function)」と呼ばれる
状態s に対して最大の値をとる状態価値関数$V^*(s)$を持ちます。

$$V^*(s) = \max_{\pi} \{V_\pi(s)\} \quad \text{for } \forall s \in S \qquad \cdots(式4\text{-}13)$$
$$= V_{\pi^*}(s)$$

・最適方策群π^*は、
「最適行動価値関数 (ｻｲﾃｷｺｳﾄﾞｳｶﾝｽｳ、optimal action-value function)」と呼ばれる
状態s に対して最大の値をとる行動価値関数$Q^*(s,a)$を持ちます。

$$Q^*(s,a) = \max_{\pi} \{Q_\pi(s,a)\} \quad \text{for } \forall s \in S、\forall a \in A(s) \qquad \cdots(式4\text{-}14)$$
$$= Q_{\pi^*}(s,a^*) \quad \text{for } \forall s \in S、\exists a^* \in A^*(s)$$

$$a^* = \mathrm{argmax}_{a'} Q_\pi(s,a') \quad \text{for } \forall s \in S、\forall a' \in A(s)、\exists a^* \in A^*(s)$$

(※) 一般に「$\mathrm{argmax}_z F(z)$」は「関数$F(z)$を最大にするz」の意味

(4.5) 最適ﾍﾞﾙﾏﾝ方程式

・「状態価値関数のﾍﾞﾙﾏﾝ方程式」(式4-6)

$$V_\pi(s) = \Sigma_a \pi(a|s) \Sigma_{s'} P_{ss'}^a \times \{R_{ss'}^a + \gamma V_\pi(s')\} \quad \cdots(式4\text{-}6)再掲$$

において、最適方策群π^* を取る場合、
価値関数が最大$V^*(s)$となる特定の行動a のみを取る為、

$$\pi(a|s) = 1 \qquad (a \in A^*(s))$$

となり、行動a についての集計Σ_a は不要となります。
また、状態s' の時の状態価値関数$V_\pi(s')$も、最適状態価値関数$V^*(s')$で置き換わります。
これにより、「状態価値関数のﾍﾞﾙﾏﾝ最適方程式 (Bellman optimality equation)」が得られます：

状態価値関数のﾍﾞﾙﾏﾝ最適方程式 (Bellman optimality equation)

$$V^*(s) = \max_{a \in A^*(s)} \Sigma_{s'} P_{ss'}^a \times \{R_{ss'}^a + \gamma V^*(s')\} \quad \cdots(式4\text{-}15)$$

$V^*(s)$ ：状態がs の時に、最適方策π^* の下での状態価値関数
$V^*(s')$ ：状態がs'の時に、最適方策π^* の下での状態価値関数
$A^*(s)$ ：状態がs の時に、状態価値関数が最大$V^*(s)$となる行動aの集合
$R_{ss'}^a$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
$P_{ss'}^a$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
s' ：状態s からの遷移先の状態
$\max_{a \in A^*(s)}$ ：状態がs の時に、$a \in A^*(s)$ となる特定の行動a を選択するという意味

・「行動価値関数のﾍﾞﾙﾏﾝ方程式」(式4-12)

$$Q_\pi(s,a) = \Sigma_{s'} P_{ss'}^a \times [R_{ss'}^a + \gamma \{\Sigma_{a'} \pi(a'|s') \times Q_\pi(s',a')\}]$$

$\cdots(式4\text{-}12)再掲$

において、最適方策群π^* を取る場合、
状態s から遷移後の状態s' 以降では、行動関数が最大$Q^*(s',a')$となる特定の行動a' のみを取る為、

$$\pi(a'|s') = 1 \qquad (a' \in A^*(s'))$$

となり、行動a' についての集計$\Sigma_{a'}$ は不要となります。
また、状態s' の時の行動価値関数$Q_\pi(s',a')$も、最適行動価値関数$Q^*(s',a')$で置き換わります。
これにより、「行動価値関数のﾍﾞﾙﾏﾝ最適方程式 (Bellman optimality equation)」が得られます：

行動価値関数のﾍﾞﾙﾏﾝ最適方程式 (Bellman optimality equation)

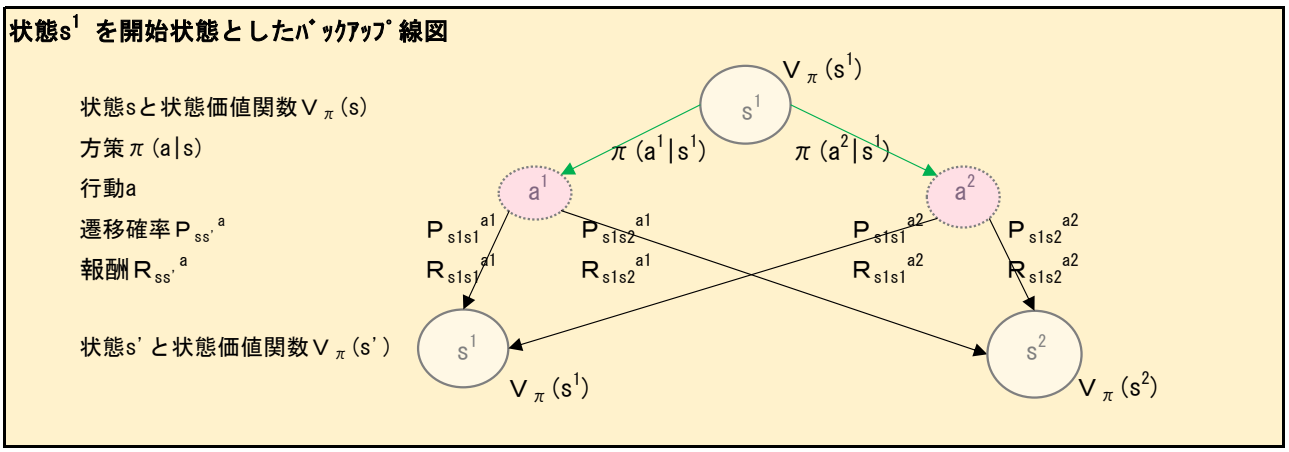
$$Q^*(s,a) = \Sigma_{s'} P_{ss'}^a \times [R_{ss'}^a + \gamma \max_{a' \in A^*(s')} Q^*(s',a')] \quad \cdots(式4\text{-}16)$$

$Q^*(s,a)$ ：状態がs の時に、最適方策π^*の下で行動aをとる時の行動価値関数
$Q^*(s',a')$ ：状態がs'の時に、最適方策π^*の下で行動a'をとる時の行動価値関数
$A^*(s')$ ：状態がs'の時に、価値関数が最大$V^*(s')$となる行動a'の集合
$R_{ss'}^a$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
$P_{ss'}^a$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
s' ：状態s からの遷移先の状態
$\max_{a' \in A^*(s')}$ ：状態がs' の時に、$a' \in A^*(s')$ となる特定の行動a' を選択するという意味

(4.6) 報酬の設計

・強化学習では、「報酬を最大化するように方策πを最適化する」というのが目標ですが、
ここで出てくる「報酬」はどのように設定するのが相応しいのでしょうか？

(図4.3) (図4.1)のﾊﾞｯｸｱｯﾌﾟ線図の再掲でﾊﾟﾗﾒｰﾀの説明付き



・強化学習では、望ましくない状態遷移より、望ましい状態遷移に大きい報酬を付与するように、
報酬の値を設計します。
目標地点へ到達することが目的の場合、中間の状態遷移への報酬の付与の仕方によっては、
途中で報酬稼ぎをしてしまい、目標地点へ到達しないままになってしまう場合もあります。
この場合は、「目標地点に到達する状態遷移にのみ正の報酬を置く」、といった報酬設計が必要です。

(例) 2次元の格子点上での、開始点から目的地への最短経路の探索では、以下のようにﾓﾃﾞﾙを設定し、
方策改善を通して任意の状態s で状態価値関数Vπ(s)を最大にします。

状態s	：格子点上の座標値(x, y)
行動a	：格子点上の移動方向のﾍﾞｸﾄﾙ(Δx, Δy)。但し右か下へのみ移動可能とする。
方策π(a|s)	：状態s(＝座標)での行動a(＝移動方向の選択)の選択確率
遷移確率Pss'a	：状態がs の時に、行動a を選択して状態がs'に遷移する遷移確率Pss'a は、 遷移先s'が範囲外の場合 0%、遷移先s'が範囲内の場合 100% 但し、現状態s が目標地点の場合は遷移しないため 0%
報酬Rss'a	：状態s で行動a を取った時の状態s'への遷移に伴う報酬Rss'a は、 目的地への遷移を+0、それ以外への遷移を −1

・着目している状態s(＝開始点座標)から、目的地へ着くための最短の経路は、
方策π(a|s)に従い、状態sで最大の確率π(a|s)の行動a∈A(s)をとれば得られます。
また、隣接格子点を見渡して、状態価値関数Vπ(s)が最大のものを辿ることでも得られます。

目的地を(3, 3) とした時の
状態価値関数Vπ(s)

	0	1	2	3
0	−6.0	−5.0	−4.0	−3.0
1	−5.0	−4.0	−3.0	−2.0
2	−4.0	−3.0	−2.0	−1.0
3	−3.0	−2.0	−1.0	0.0

開始点を(2, 1)とした時の目的地(3, 3)までの最短経路

(その1)

	0	1	2	3
0	−6.0	−5.0	−4.0	−3.0
1	−5.0	−4.0	−3.0	−2.0
2	−4.0	−3.0	−2.0	−1.0
3	−3.0	−2.0	−1.0	0.0

(その2)

	0	1	2	3
0	−6.0	−5.0	−4.0	−3.0
1	−5.0	−4.0	−3.0	−2.0
2	−4.0	−3.0	−2.0	−1.0
3	−3.0	−2.0	−1.0	0.0

(その3)

	0	1	2	3
0	−6.0	−5.0	−4.0	−3.0
1	−5.0	−4.0	−3.0	−2.0
2	−4.0	−3.0	−2.0	−1.0
3	−3.0	−2.0	−1.0	0.0

(凡例)
：Vπ(s)が開始点以下
：開始点
：目的地
：移動方向

（「(ｸﾞﾗﾌ 5-1) 割引率γの寄与率の変化」の実装例）

```
#**************************************************************************/
# ﾘｽﾄ11-(04)-1_割引率γ の寄与率の変化
#**************************************************************************/
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 割引率γ**n
def fGammaN(gamma, n):
    return gamma**n

# n データ列
nlist = np.linspace(0, 30, 62)
ylist025 = fGammaN(0.25, nlist)
ylist050 = fGammaN(0.50, nlist)
ylist075 = fGammaN(0.75, nlist)
ylist090 = fGammaN(0.90, nlist)

# グラフ表示：割引率γ**n
plt.figure(figsize=(12,3))
plt.title("(Time series of discounting factor)")
plt.plot(nlist, ylist025, 'black', linewidth=2, label='(gamma=0.25)', linestyle='solid')
plt.plot(nlist, ylist050, 'red', linewidth=2, label='(gamma=0.50)', linestyle='dashed')
plt.plot(nlist, ylist075, 'green', linewidth=2, label='(gamma=0.75)', linestyle='dashdot')
plt.plot(nlist, ylist090, 'cornflowerblue', linewidth=2, label='(gamma=0.90)', linestyle='dotted')
plt.ylim(0, 1)
plt.xlim(0, 30)
plt.grid(True)
plt.legend(loc='upper right')

plt.show()
```

【出典・参考】

全般⇒「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)

全般⇒「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井悦司)

全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

ﾍﾞﾙﾏﾝ方程式⇒ https://note.com/npaka/n/n7fca1d4d5ce8

行動価値関数のﾍﾞﾙﾏﾝ方程式⇒ https://qiita.com/triwave33/items/8966890701169f8cad47

(5) 価値関数の解析解

・前節の(式4-6) で状態価値関数のﾍﾞﾙﾏﾝ方程式を、
(式4-12) で行動価値関数のﾍﾞﾙﾏﾝ方程式を各々紹介しました。
価値関数を最大化する方策πを求めるのが強化学習の目的ですが、
この方程式をどのようにして解くのでしょうか？

状態価値関数のﾍﾞﾙﾏﾝ方程式 (Bellman equation)

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{a} \times \{R_{ss'}^{a} + \gamma V_\pi(s')\}$$ …(式4-6)再掲

$V_\pi(s)$ ：状態がs の時に、方策πの下での状態価値関数
$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策(確率)
$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$R_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
$V_\pi(s')$ ：状態s からの遷移先の状態がs' の時に、方策πの下での状態価値関数

行動価値関数のﾍﾞﾙﾏﾝ方程式

$$Q_\pi(s,a) = \sum_{s'} P_{ss'}^{a} \times \left[R_{ss'}^{a} + \gamma \left\{ \sum_{a'} \pi(a'|s') \times Q_\pi(s',a') \right\} \right]$$ …(式4-12)再掲

$Q_\pi(s,a)$ ：状態がs で行動a を取り、状態遷移後は方策πに従った時の行動価値関数
$Q_\pi(s',a')$ ：状態がs'で行動a'を取り、状態遷移後は方策πに従った時の行動価値関数
$P_{ss'}^{a}$ ：状態がs で行動a を取り、状態がs'に遷移する確率
$R_{ss'}^{a}$ ：状態がs で行動a を取り、状態がs'に遷移した時の報酬
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
$\pi(a'|s')$ ：状態がs' の時に、行動a' を選択する(方策)確率

・次の諸量を導入して、状態価値関数のﾍﾞﾙﾏﾝ方程式を解析的に解く方法を、以下に紹介します。
- 遷移確率ﾏﾄﾘｸｽ P^{π}
- 報酬期待値ﾍﾞｸﾄﾙ R^{π}
- 価値ﾍﾞｸﾄﾙ V_π

【出典・参考】
・全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

(5.1) 遷移確率ﾏﾄﾘｸｽ P^{π}

・方策(確率) $\pi(a|s)$ の下で、状態s から遷移先の状態s' に至る遷移確率 $\{P^{\pi}\}_{ss'}$ は、
行動a を選択して状態がs' に遷移する確率 $P_{ss'}^{a}$ に、行動a の方策(確率) $\pi(a|s)$ を掛けて、
行動a について集計したものになります。

方策(確率) $\pi(a|s)$ の下で、状態sから遷移先の状態s' に至る遷移確率

$$\{P^{\pi}\}_{ss'} = \sum_a \pi(a|s) \times P_{ss'}^{a} \qquad \cdots(式5\text{-}1)$$

$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策(確率)
$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率

・遷移確率 $\{P^{\pi}\}_{ss'}$ を、(状態値集合Sの要素数×Sの要素数)のｻｲｽﾞのﾏﾄﾘｸｽの(s,s')成分とする行列が、
「遷移確率ﾏﾄﾘｸｽ (ｾﾝｲｶｸﾘｯﾂﾏﾄﾘｸｽ、Transition Probability Matrix)」 P^{π} です。

(例) 既述の状態遷移表(表3.2)について、「遷移確率ﾏﾄﾘｸｽ」 P^{π} は以下のようになります：

$$P^{\pi} = \begin{bmatrix} \{P^{\pi}\}_{s1s1} & \{P^{\pi}\}_{s1s2} \\ \{P^{\pi}\}_{s2s1} & \{P^{\pi}\}_{s2s2} \end{bmatrix}$$

$$\{P^{\pi}\}_{s1s1} = \pi(a^1|s^1) \times P_{s1s1}^{a1} + \pi(a^2|s^1) \times P_{s1s1}^{a2}$$
$$\{P^{\pi}\}_{s1s2} = \pi(a^1|s^1) \times P_{s2s1}^{a1} + \pi(a^2|s^1) \times P_{s2s1}^{a2}$$
$$\{P^{\pi}\}_{s2s1} = \pi(a^1|s^2) \times P_{s1s2}^{a1} + \pi(a^2|s^2) \times P_{s1s2}^{a2}$$
$$\{P^{\pi}\}_{s2s2} = \pi(a^1|s^2) \times P_{s2s2}^{a1} + \pi(a^2|s^2) \times P_{s2s2}^{a2}$$

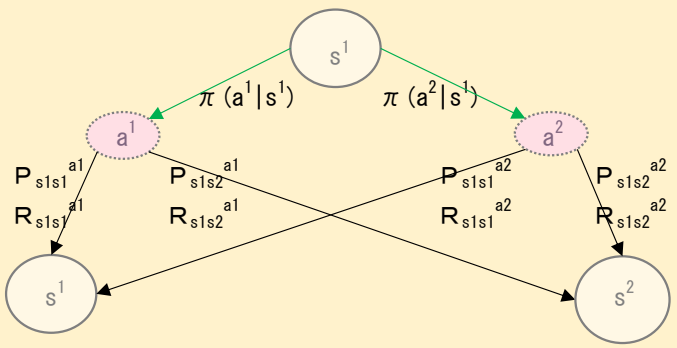
(表3.2) 状態値が2種類 $\{s^1, s^2\}$ で、2種類の行動 $\{a^1, a^2\}$ をもつﾏﾙｺﾌ決定過程の状態遷移表例 (再掲)

No.	時刻t			時刻t+1	遷移確率(例) $P_{ss'}^{a}$		即時報酬(例) $R_{ss'}^{a}$
	状態s	方策π	行動a	状態s'			
①	s^1	$\pi(a^1\|s^1)$	a^1	s^1	0.3	(小計)	+1
②				s^2	0.7	1.0	−2
③		$\pi(a^2\|s^1)$	a^2	s^1	0.5	(小計)	+1
④				s^2	0.5	1.0	+2
⑤	s^2	$\pi(a^1\|s^2)$	a^1	s^1	0.6	(小計)	−1
⑥				s^2	0.4	1.0	+2
⑦		$\pi(a^2\|s^2)$	a^2	s^1	0.7	(小計)	+1
⑧				s^2	0.3	1.0	+1

(表3.2)で、状態 s^1 を出発点として2種類の行動 $\{a^1, a^2\}$ を取り、2種類 $\{s^1, s^2\}$ の状態へ遷移する様子

ﾊﾞｯｸｱｯﾌﾟ線図

状態s …
方策 $\pi(a|s)$ …
行動a …
遷移確率p …
報酬r …
状態s' …



※ これは(図3.3)の再掲です。

(5.2) 報酬期待値ﾍﾞｸﾄﾙR^{π}

・状態s で行動a を取った時の報酬期待値$E_r(s,a)$ は、
即時報酬$R_{ss'}^{a}$ にその発生確率$P_{ss'}^{a}$ を掛けたものを s' について集計したものになります。

$$E_r(s,a) = \Sigma_{s'} \; P_{ss'}^{a} \times R_{ss'}^{a}$$

・方策(確率)$\pi(a|s)$ の下で、状態sでの報酬期待値$\{R^{\pi}\}_s$は、
状態s で行動a をとった時の報酬期待値$E_r(s,a)$に、
行動a の方策確率$\pi(a|s)$を掛けて、行動a について集計したものになります。

方策(確率)$\pi(a|s)$ の下で、状態sでの報酬期待値

$$\{R^{\pi}\}_s = \Sigma_a \; \pi(a|s) \times E_r(s,a)$$
$$= \Sigma_a \; \pi(a|s) \times \Sigma_{s'} \; P_{ss'}^{a} \times R_{ss'}^{a} \qquad \cdots(式5\text{-}2)$$

$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策(確率)
$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$R_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬

・報酬期待値$\{R^{\pi}\}_s$を、(状態値集合Sの要素数)のｻｲｽﾞのﾍﾞｸﾄﾙの(s)成分とするﾍﾞｸﾄﾙが、
「報酬期待値ﾍﾞｸﾄﾙ (ﾎｳｼｭｳｷﾀｲﾁﾍﾞｸﾄﾙ、Reward expected value vector)」R^{π}です。

(例) 既述の状態遷移表(表3.2)について、「報酬期待値ﾍﾞｸﾄﾙ」R^{π}は以下のようになります：

$$R^{\pi} = \begin{bmatrix} \{R^{\pi}\}_{s1} & \{R^{\pi}\}_{s2} \end{bmatrix}$$

$$\{R^{\pi}\}_{s1} = \pi(a^1|s^1) \times \{ P_{s1s1}^{a1} \times R_{s1s1}^{a1} + P_{s1s2}^{a1} \times R_{s1s2}^{a1} \}$$
$$+ \pi(a^2|s^1) \times \{ P_{s1s1}^{a2} \times R_{s1s1}^{a2} + P_{s1s2}^{a2} \times R_{s1s2}^{a2} \}$$

$$\{R^{\pi}\}_{s2} = \pi(a^1|s^2) \times \{ P_{s2s1}^{a1} \times R_{s2s1}^{a1} + P_{s2s2}^{a1} \times R_{s2s2}^{a1} \}$$
$$+ \pi(a^2|s^2) \times \{ P_{s2s1}^{a2} \times R_{s2s1}^{a2} + P_{s2s2}^{a2} \times R_{s2s2}^{a2} \}$$

(表3.2) 状態値が2種類$\{s^1, s^2\}$で、2種類の行動$\{a^1, a^2\}$をもつﾏﾙｺﾌ決定過程の状態遷移表例 (再掲)

No.	時刻t			時刻t+1	遷移確率(例) $P_{ss'}^{a}$		即時報酬(例) $R_{ss'}^{a}$
	状態s	方策π	行動a	状態s'			
①	s^1	$\pi(a^1\|s^1)$	a^1	s^1	0.3	(小計)	+1
②				s^2	0.7	1.0	−2
③		$\pi(a^2\|s^1)$	a^2	s^1	0.5	(小計)	+1
④				s^2	0.5	1.0	+2
⑤	s^2	$\pi(a^1\|s^2)$	a^1	s^1	0.6	(小計)	−1
⑥				s^2	0.4	1.0	+2
⑦		$\pi(a^2\|s^2)$	a^2	s^1	0.7	(小計)	+1
⑧				s^2	0.3	1.0	+1

(5.3) 価値ﾍﾞｸﾄﾙV_πとその解析解

・方策(確率)$\pi(a|s)$ の下で、状態$s \in S$ での状態価値関数 $V_\pi(s)$ を
ﾍﾞｸﾄﾙの(s)成分とするﾍﾞｸﾄﾙが、「価値ﾍﾞｸﾄﾙ (ｶﾁﾍﾞｸﾄﾙ、Value vector)」V_πです。
価値ﾍﾞｸﾄﾙV_π のｻｲｽﾞは、状態値集合Sの要素数になります。

方策(確率)$\pi(a|s)$ の下での価値ﾍﾞｸﾄﾙ

$$V_\pi = (V_\pi(s^1),\ V_\pi(s^2),\ \cdots,\ V_\pi(s^N)) \qquad \cdots(式5\text{-}3)$$

s^n ：状態値、$s^n \in S$ (n=1～N)
S ：取りうる状態値の集合

・状態価値関数のﾍﾞﾙﾏﾝ方程式(式4-6)は、上記の関係式 (式5-1),(式5-2),(式5-3) を用いて
価値ﾍﾞｸﾄﾙV_πが従うﾍﾞｸﾄﾙ方程式(式5-4)として表され、(式5-5)で解析的に解くことができます(※1)：

価値ﾍﾞｸﾄﾙv_πの解析解

$$V_\pi = R^\pi + \gamma P^\pi V_\pi \qquad \cdots(式5\text{-}4)$$
$$\therefore \quad V_\pi = (E - \gamma P^\pi)^{-1} R_\pi \qquad \cdots(式5\text{-}5)$$

γ ：割引率 ($0 \leqq \gamma \leqq 1$)
V_π ：方策(確率)$\pi(a|s)$ の下での価値ﾍﾞｸﾄﾙ
R^π ：方策(確率)$\pi(a|s)$ の下での報酬期待値ﾍﾞｸﾄﾙ
P^π ：方策(確率)$\pi(a|s)$ の下での状態から遷移先の状態への遷移確率ﾏﾄﾘｸｽ
E ：N行N列の単位行列 (N：状態値集合Sの要素数)

(※1) 価値ﾍﾞｸﾄﾙV_πが従うﾍﾞｸﾄﾙ方程式(式5-4)は、
状態価値関数のﾍﾞﾙﾏﾝ方程式(式4-6)において、(式5-1),(式5-2),(式5-3) を用いることにより、
以下のように導出されます：

状態価値関数のﾍﾞﾙﾏﾝ方程式 (Bellman equation)

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{\ a} \times \{ R_{ss'}^{\ a} + \gamma V_\pi(s') \} \qquad \cdots(式4\text{-}6)再掲$$

$V_\pi(s)$ ：状態がs の時に、方策πの下での状態価値関数
$V_\pi(s')$ ：状態がs' の時に、方策πの下での状態価値関数
$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策確率
$P_{ss'}^{\ a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$R_{ss'}^{\ a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
γ ：割引率 ($0 \leqq \gamma \leqq 1$)

状態価値関数のﾍﾞﾙﾏﾝ方程式(式4-6)において、(式5-1),(式5-2),(式5-3) を用いることにより、

$$\begin{aligned}
(式4\text{-}6)\quad V_\pi(s) &= \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{\ a} \times \{ R_{ss'}^{\ a} + \gamma V_\pi(s') \} \\
&= \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{\ a} \times R_{ss'}^{\ a} + \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{\ a} \times \gamma V_\pi(s') \\
(式5\text{-}2)\rightarrow &= \{R^\pi\}_s + \gamma \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{\ a} \times V_\pi(s') \\
&= \{R^\pi\}_s + \gamma \sum_{s'} \sum_a \pi(a|s) P_{ss'}^{\ a} \times V_\pi(s') \\
(式5\text{-}1)\rightarrow &= \{R^\pi\}_s + \gamma \sum_{s'} \{P^\pi\}_{ss'} \times V_\pi(s') \\
行列の積\rightarrow &= \{R^\pi\}_s + \gamma \{P^\pi V_\pi\}_s \\
\therefore \quad V_\pi(s) &= \{R^\pi\}_s + \gamma \{P^\pi V_\pi\}_s \\
(式5\text{-}3)\rightarrow \quad V_\pi &= R^\pi + \gamma P^\pi V_\pi
\end{aligned}$$

(6) 強化学習ﾓﾃﾞﾙの学習

・前節で、遷移確率ﾏﾄﾘｸｽP_{π}、報酬期待値ﾍﾞｸﾄﾙR_{π}が与えられていれば、状態価値ﾍﾞｸﾄﾙV_{π}が
(式5-5)で解析的に計算できることが分かりました。
では、状態価値関数$V_{\pi}(s)$を評価して方策πの改善にどのように反映してゆくのでしょうか？

(6.1) 方策改善定理

・強化学習において、ある状態s において選択可能な行動の集合Ａ(s) の中から、
行動価値Ｑ(s,a)を推定する為に、任意の行動a(∈Ａ(s)) を取って必要な情報を収集することを
「探索 (ﾀﾝｻｸ、exploration)」と言います。
「探索」の結果得られた行動価値を次に取るべき方策に反映することを
「活用 (ｶﾂﾖｳ、exploitation)」と言います。

・この探索と活用による方策改善にあたり、その指針となるのが
「方策改善定理 (ﾎｳｻｸｶｲｾﾞﾝﾃｲﾘ、policy improvement theorem)」です。

方策改善定理

状態がs の時、既存の方策πに従うことによる状態価値関数$V_{\pi}(s)$と、
状態がs の時、新しい方策π'で行動a を取り、
その後既存の方策πに従った時の行動価値関数$Q_{\pi}(s, \pi'(s))$について、
以下の方策改善の関係が成り立ちます(※1)：

$$V_{\pi}(s) \leqq Q_{\pi}(s, \pi'(s)) \quad \Rightarrow \quad V_{\pi}(s) \leqq V_{\pi'}(s) \qquad \cdots(式6\text{-}1)$$

これを「方策改善定理 (ﾎｳｻｸｶｲｾﾞﾝﾃｲﾘ、policy improvement theorem)」と言い、
新しい行動方策π'は、既存の行動方策πの改善となります。

$V_{\pi}(s)$ ：状態がs の時、既存の方策πの下での状態価値関数
$V_{\pi'}(s)$ ：状態がs の時、新しい方策π'の下での状態価値関数
$Q_{\pi}(s, \pi'(s))$ ：状態がs の時、新しい方策π'で行動a(＝$\pi'(s)$) を取り、
その後既存の方策πに従った時の行動価値関数(※1)

(※1) 新しい方策π'の下での行動であることを意識し、a の代わりに$\pi'(s)$で表現します。

(6.2) greedy法／ε-greedy法

・方策改善定理に裏付けられた改善方法の一つが
「greedy法（ｸﾞﾘｰﾃﾞｨﾎｳ、貪欲法、greedy algorithm)」です。
これは、状態がs(∈Ｓ）の時に、
方策πの下で行動価値関数$Q_\pi(s,a)$が最大となる行動a（$a \in A^*(s) \subset A(s)$）のみを選ぶように、
方策(確率)$\pi(a|s)$を更新するものです。

greedy法による方策改善

状態s の時に取る行動a　＝ a　($a \in A^*(s)$)　($A^*(s) \subset A(s)$)

改善後の方策(確率)$\pi(a|s)$ ＝
- $1/M^*$　for $a \in A^*(s)$
- 0　for $a \notin A^*(s)$

集合Ａ(s) ：状態s でとりうる行動aの集合
集合$A^*(s)$ ：状態s での行動価値関数$Q_\pi(s,a)$が最大となる行動aの集合($A^*(s) \subset A(s)$)
M^* ：集合$A^*(s)$の要素数(濃度)
$1/M^*$ ：「$a \in A^*(s)$ となる行動a について均等な割合とする」という意味

・「greedy法」では、行動の選択肢を集合$A^*(s)$に属す行動a（$a \in A^*(s) \subset A(s)$）に限定し、
他の選択肢について可能性を探るということをしないために、
最適な方策を見逃す場合があります。

・最適な方策の見逃し改善の為には、探索と活用にﾄﾚｰﾄﾞｵﾌを取ることが必要です。
その方法の一つが、「ε-greedy法（ｲﾌﾟｼﾛﾝｸﾞﾘｰﾃﾞｨﾎｳ、ε-貪欲法、ε-greedy algorithm)」です。
これは、「確率εで無作為に行動を選択し(探索)、確率(1－ε)でgreedy法により行動を選択する(活用)」
という方策です。

ε-greedy法による方策改善

状態s の時に取る行動a　＝ a∈Ａ(s)　($A^*(s) \subset A(s)$)

改善後の方策(確率)$\pi(a|s)$ ＝
- $(1-\varepsilon)/M^* + \varepsilon/M$　for $a \in A^*(s)$
- ε/M　for $a \notin A^*(s)$

集合Ａ(s) ：状態s でとりうる行動aの集合
集合$A^*(s)$ ：状態s での行動価値関数$Q_\pi(s,a)$が最大となる行動aの集合($A^*(s) \subset A(s)$)
M ：集合Ａ(s)の要素数(濃度)
M^* ：集合$A^*(s)$の要素数(濃度)
ε/M ：「a∈集合Ａ(s) となる任意の行動a について割合(ε)を等分割する」という意味
$(1-\varepsilon)/M^*$ ：「a∈集合$A^*(s)$ となる任意の行動a について割合(1-ε)を等分割する」という意味

・「greedy法」は、「ε-greedy法」で「ε＝０」とした場合に相当します。
・「ε-greedy法」においてどのようなεを選択するかについては、
「ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀﾁｭｰﾆﾝｸﾞ」という作業になり、「ｸﾞﾘｯﾄﾞｻｰﾁ」といった手法などがあります。
「ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀ (Hyperparameter)」とは、機械学習ﾓﾃﾞﾙを構成する調整可能なﾊﾟﾗﾒｰﾀのうち、
学習で自動的にﾁｭｰﾆﾝｸﾞできず人手で調整する必要のあるﾊﾟﾗﾒｰﾀのことです。

【出典・参考】
全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)
ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀ⇒「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井 悦司)
ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀ⇒ https://www.codexa.net/hyperparameter-tuning-python/

(7) ﾓﾃﾞﾙﾍﾞｰｽな手法での価値関数と方策改善

・環境ﾓﾃﾞﾙの情報が既知の場合、
ﾍﾞﾙﾏﾝ方程式を動的計画法により厳密に解くことができます。
これを「ﾓﾃﾞﾙﾍﾞｰｽな手法 (Model-Based Reinforcement Learning)」と言います。
ﾓﾃﾞﾙﾍﾞｰｽな手法に、以下のようなものがあります。

(1) 「方策反復法 (ﾎｳｻｸﾊﾝﾌﾟｸﾎｳ、Policy iteration)」
(2) 「価値反復法 (ｶﾁﾊﾝﾌﾟｸﾎｳ、Value iteration)」

・「環境ﾓﾃﾞﾙの情報が既知」というのは、状態遷移確率 $P_{ss'}^{ar}$ が既知ということです。

ﾓﾃﾞﾙﾍﾞｰｽな手法での既知関数

$P_{ss'}^{ar}$ ：状態がs の時に、行動a を取って、
遷移先の状態がs'、遷移時の報酬がr となる確率

$$P_{ss'}^{ar} = P(S_{t+1}=s', R_{t+1}=r \mid S_t=s, A_t=a) \quad \cdots(式3\text{-}2)再掲$$

・状態遷移確率 $P_{ss'}^{ar}$ が既知の場合、以下の値も計算式により導出できます：

ﾓﾃﾞﾙﾍﾞｰｽな手法での既知関数(導出)

$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率
$R_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬

導出式は以下の通り：

$$P_{ss'}^{a} = P(S_{t+1}=s' \mid S_t=s, A_t=a) = \sum_r P_{ss'}^{ar} \quad \cdots(式3\text{-}3)再掲$$

$$R_{ss'}^{a} = \sum_r r \times \{P_{ss'}^{ar} / P_{ss'}^{a}\} \quad \cdots(式3\text{-}4)再掲$$

(例) 既述の(表3.2)で示されるﾓﾃﾞﾙは、「環境ﾓﾃﾞﾙが既知」となっている例です。
この例では、$P_{ss'}^{ar}$ のr は明記されていませんが、
時刻t での(状態,行動)＝(s,a)の時に、
時刻t+1 において(状態,報酬)＝(s',r)となる確率 $P_{ss'}^{ar}$ は、
遷移確率 $P_{ss'}^{a}$ として与えられており、
その時の報酬r は必ず1種類の即時報酬 $R_{ss'}^{a}$ を得るものとしています。

(表3.2) 状態値が2種類{s^1, s^2}で、2種類の行動{a^1, a^2}をもつﾏﾙｺﾌ決定過程の状態遷移表例 (再掲)

No.	時刻t			時刻t+1	遷移確率(例) $P_{ss'}^{a}$		即時報酬(例) $r=R_{ss'}^{a}$
	状態s	方策π	行動a	状態s'			
①	s^1	$\pi(a^1 \mid s^1)$	a^1	s^1	0.3	(小計)	+1
②				s^2	0.7	1.0	−2
③		$\pi(a^2 \mid s^1)$	a^2	s^1	0.5	(小計)	+1
④				s^2	0.5	1.0	+2
⑤	s^2	$\pi(a^1 \mid s^2)$	a^1	s^1	0.6	(小計)	−1
⑥				s^2	0.4	1.0	+2
⑦		$\pi(a^2 \mid s^2)$	a^2	s^1	0.7	(小計)	+1
⑧				s^2	0.3	1.0	+1

【出典・参考】
ﾓﾃﾞﾙﾍﾞｰｽ、ﾓﾃﾞﾙﾌﾘｰ⇒ https://note.com/npaka/n/n5a6bc4825555
ﾓﾃﾞﾙの分類⇒ https://qiita.com/sconeman/items/cb6047540a39b3bf15d7

(7.1) 方策反復法

・ﾓﾃﾞﾙﾍﾞｰｽな手法の方策改善手法の一つに「方策反復法」があります。

(7.1.1) 方策反復法のｱﾙｺﾞﾘｽﾞﾑ

・「方策πの下での状態価値関数V_πを評価して方策πを改善し、より優れた方策π'を得る」という
操作を繰り返してゆくことにより、方策の改善を行い、最適価値関数に収束することができます。
最適方策を発見するこのような手法は「方策反復（ﾎｳｻｸﾊﾝﾌﾟｸ、policy iteration)」と呼ばれています。

・「優れた方策」というのは、
「状態価値関数$V_\pi(s)$ を全ての状態s について最大化する方策」のことです。

・以下にε-greedy法による方策反復法のｱﾙｺﾞﾘｽﾞﾑを記します：

方策反復（ﾎｳｻｸﾊﾝﾌﾟｸ、policy iteration）のｱﾙｺﾞﾘｽﾞﾑ（ε-greedy法）

(0) 以下のﾊﾟﾗﾒｰﾀは既知とします

$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率

$R_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬

γ ：割引率 $(0 \leqq \gamma \leqq 1)$

ε ：ε-greedy法のﾊﾟﾗﾒｰﾀε

ΔV_{thresh} ：状態価値関数$V_\pi(s)$ の更新量のしきい値

（これより最適状態価値関数$V^*(s)$の最大値の更新量が小さければ、
本ｱﾙｺﾞﾘｽﾞﾑを終了する）

(1) 初期化

全ての状態s∈S、行動a∈A(s)に対して、以下の初期化を行う：

(1.1) 状態価値関数の値$V_\pi(s)$ を任意に初期化する。

(1.2) 方策確率$\pi(a|s)$ を任意に初期化する。

(2) 方策評価

現方策πの下での状態価値関数$V_\pi(s)$を以下のようにして確定する：

(2.1) $\Delta V_{\pi max} \leftarrow 0$

(2.2) 全ての状態s∈Sについて以下を繰り返す：

$V_{\pi old}(s) \leftarrow$ 現方策πの下での状態価値関数$V_\pi(s)$

$V_\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} P_{ss'}^{a} \times \{R_{ss'}^{a} + \gamma V_\pi(s')\}$ …(式4-6)

$\Delta V_{\pi max} \leftarrow \max\{\Delta V_{\pi max}, abs(V_{\pi old}(s) - V_\pi(s))\}$

(2.3) $\Delta V_{\pi max}$ をみて状態価値関数$V_\pi(s)$ の確定判定を行う

(1) $\Delta V_{\pi max} < \Delta V_{thresh}$ の場合、
全ての状態s∈S に対して、方策πの下での状態価値関数$V_\pi(s)$
が確定したとして、(3)方策改善へ。

(2) $\Delta V_{\pi max} \geqq \Delta V_{thresh}$ の場合、
状態価値関数$V\pi(s)$が未確定として、(2.1)から繰り返す。

(3) 方策改善

現方策$\pi(a|s)$ を以下のようにして改善する：

(3.1) 方策改善なしフラグ← true

(3.2) 全ての状態$s \in S$ について以下を繰り返す：

(3.2.1) $Q_{\pi max}(s)$ ← (式4-10)で計算される$Q_{\pi}(s,a)$では
ありえない小さい値

(3.2.2) $A^*(s)$ ← Φ (空集合)

(3.2.3) 全ての行動$a \in A(s)$について以下の手順(1)～(2)を繰り返し、
状態s での最適行動価値関数$Q_{\pi max}(s)$を特定する：

(1) 状態s で行動a を取る時の行動価値関数$Q_{\pi}(s,a)$を計算する：

$$Q_{\pi}(s,a) \leftarrow \sum_{s'} P_{ss'}^{a} \times \{R_{ss'}^{a} + \gamma V_{\pi}(s')\} \quad \cdots(式4\text{-}10)$$

(2) 計算した$Q_{\pi}(s,a)$ に応じて、
最適行動価値関数$Q_{\pi max}(s)$の更新を行う：

(a) $Q_{\pi max}(s) > Q_{\pi}(s,a)$ の時、
何もしない

(b) $Q_{\pi max}(s) = Q_{\pi}(s,a)$ の時
集合$A^*(s)$ の要素として行動a を追加する

(c) $Q_{\pi max}(s) < Q_{\pi}(s,a)$ の時
$Q_{\pi max}(s) \leftarrow Q_{\pi}(s,a)$
集合$A^*(s)$ の要素を行動a のみとする

(3.2.4) 状態s での最適行動価値関数$Q^*(s,a)$ を$Q_{\pi max}(s)$で更新する。

(3.2.5) 状態s での全ての行動$a \in A(s)$について、
ε-greedy法を用いて方策$\pi(a|s)$を更新する：

$\pi_{old}(a|s) \leftarrow \pi(a|s)$

$$\pi(a|s) \leftarrow \begin{cases} \varepsilon/M + (1-\varepsilon)/M^* & \text{for } a \in A^*(s) \\ \varepsilon/M & \text{for } a \notin A^*(s) \end{cases}$$

もし、$\pi_{old}(a|s) \neq \pi(a|s)$ ならば、
方策改善なしフラグ←false で更新する(つまり方策改善された)。

(3.3) もし、方策改善なしフラグ＝true ならば、本「方策反復」ｱﾙｺﾞﾘｽﾞﾑを終了。
それ以外ならば、改善された方策で評価し直す為に「(2)方策評価」へ戻る。

$V_{\pi}(s)$ ：状態がs の時に、方策πの下での状態価値関数
$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策確率
$Q_{\pi}(s,a)$ ：状態がs で行動a を取り、状態遷移後は方策πに従った時の行動価値関数
集合$A(s)$ ：状態s でとりうる行動aの集合
集合$A^*(s)$ ：状態s での行動価値関数$Q_{\pi}(s,a)$が最大となる行動aの集合($A^*(s) \subset A(s)$)
M ：集合$A(s)$の要素数
M^* ：集合$A^*(s)$の要素数
ε/M ：「a∈集合$A(s)$ となる任意の行動a について割合(ε)を等分割する」という意味
$(1-\varepsilon)/M^*$ ：「a∈集合$A^*(s)$ となる任意の行動a について割合$(1-\varepsilon)$を等分割する」という意味

(7.1.2) 方策反復法の実装例

・以下に、解析解による方策評価と、ε-greedy法により方策改善を行う方策反復法の実装例を示します。
この例は、環境ﾓﾃﾞﾙが既知の(表3.2) の状態遷移表について、方策πの最適解を得ようとする実装です。

(1) 初期値を以下のように与えます：
(1-1) ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの設定を以下のようにします；
・割引率γ＝0.8
・ε-greedy法のﾊﾟﾗﾒｰﾀε＝0.2
・価値関数の変化分⊿$V_{\pi max}$ が十分小さくなったという判定の値(delta_thresh)＝0.001
(1-2) 状態価値関数$V_{\pi}(s)$の初期値として、0を設定します。
(1-3) 方策πの初期値として、各状態s(∈Ｓ) の時に、行動a(∈Ａ) を行う割合が平等になるように均等な遷移確率を与えます：
・π(a|s) ＝ 1/ 集合Ａの要素数 ＝ 1/2 ＝ 0.5
(2) そのうえで、価値ﾍﾞｸﾄﾙ$V_{\pi}(s)$、更に行動価値関数$Q_{\pi}(s,a)$を計算して、ε-greedy法により、方策(確率) π(a|s) を更新する・・・ということを繰り返して、方策の最適化を行います。

・実行結果を以下に掲載します。

```
Hyper parameters :
  (1) ε=0.200
  (2) γ=0.800
  (3) delta_thresh=0.001

【 iteratePolicy : No.=1 】
 Policy : π(a|s) ----------------------------------
   π('a1'|'s1') = 0.500
   π('a2'|'s1') = 0.500
   π('a1'|'s2') = 0.500
   π('a2'|'s2') = 0.500
 evaluatePolicy: repeated 22 times
 State-value function : Vπ(s) ---------------------
   Vπ('s1') = 1.798
   Vπ('s2') = 2.132
 Action-value function : Qπ(s,a) ------------------
   Qπ('s1','a1') = 0.525
   Qπ('s1','a2') = 3.072
   Qπ('s2','a1') = 1.745
   Qπ('s2','a2') = 2.518

【 iteratePolicy : No.=2 】
 Policy : π(a|s) ----------------------------------
   π('a1'|'s1') = 0.100
   π('a2'|'s1') = 0.900
   π('a1'|'s2') = 0.100
   π('a2'|'s2') = 0.900
 evaluatePolicy: repeated 24 times
 State-value function : Vπ(s) ---------------------
   Vπ('s1') = 5.628
   Vπ('s2') = 5.354
 Action-value function : Qπ(s,a) ------------------
   Qπ('s1','a1') = 3.249
   Qπ('s1','a2') = 5.893
   Qπ('s2','a1') = 4.615
   Qπ('s2','a2') = 5.437
```

方策改善の反復は2回で収束しましたが、
各方策πの下での、状態価値関数の計算には、各々22回、24回の逐次近似をしていることがわかります。

結果を見ると、

$$\begin{cases} \pi(a^1|s^1)=0.1 \quad ,\pi(a^2|s^1)=0.9 & : 状態=s^1 \text{ の場合、行動}(a^1,\ a^2)\text{を}(0.1:0.9)\text{で選択} \\ \pi(a^1|s^2)=0.1 \quad ,\pi(a^2|s^2)=0.9 & : 状態=s^2 \text{ の場合、行動}(a^1,\ a^2)\text{を}(0.1:0.9)\text{で選択} \end{cases}$$

という方策にすると、状態価値$V_{\pi}(s)$ が最大になることがわかります。

(表3.2) の即時報酬を見ると、行動a^1を見ると、ﾏｲﾅｽの報酬が目立っていて、
状態s^1, s^2 のいずれの場合も、行動a^2を選んだ方がよさそうだということが、(表3.2)からも推察できます。

(表3.2) 状態値が2種類$\{s^1, s^2\}$で、2種類の行動$\{a^1, a^2\}$をもつﾏﾙｺﾌ決定過程の状態遷移表例 (再掲)

No.	時刻t			時刻t+1	遷移確率(例) $P_{ss'}^{a}$		即時報酬(例) $R_{ss'}^{a}$
	状態s	方策π	行動a	状態s'			
①	s^1	$\pi(a^1\|s^1)$ ⇒ 0.1	a^1	s^1	0.3	(小計)	+1
②				s^2	0.7	1.0	−2
③		$\pi(a^2\|s^1)$ ⇒ 0.9	a^2	s^1	0.5	(小計)	+1
④				s^2	0.5	1.0	+2
⑤	s^2	$\pi(a^1\|s^2)$ ⇒ 0.1	a^1	s^1	0.6	(小計)	−1
⑥				s^2	0.4	1.0	+2
⑦		$\pi(a^2\|s^2)$ ⇒ 0.9	a^2	s^1	0.7	(小計)	+1
⑧				s^2	0.3	1.0	+1

参考までに、ε=0.0 とした greedy法では、以下のような結果になります：

$$\begin{cases} \pi(a^1|s^1)=0.0 \quad ,\pi(a^2|s^1)=1.0 & : 状態=s^1 \text{ の場合、必ず行動 } a^2 \text{を選択する。} \\ \pi(a^1|s^2)=0.0 \quad ,\pi(a^2|s^2)=1.0 & : 状態=s^2 \text{ の場合、必ず行動 } a^2 \text{を選択する。} \end{cases}$$

状態価値$V_\pi(s)$ は、ε=0.2 とした ε-greedy法よりも高くなっています。

```
Hyper parameters :
  (1) ε=0.000
  (2) γ=0.800
  (3) delta_thresh=0.001

【 iteratePolicy : No.=1 】
 Policy : π(a|s) ----------------------------------
   π('a1'|'s1') = 0.500
   π('a2'|'s1') = 0.500
   π('a1'|'s2') = 0.500
   π('a2'|'s2') = 0.500
 evaluatePolicy: repeated 22 times
 State-value function : Vπ(s) ---------------------
   Vπ('s1') = 1.798
   Vπ('s2') = 2.132
 Action-value function : Qπ(s,a) ------------------
   Qπ('s1','a1') = 0.525
   Qπ('s1','a2') = 3.072
   Qπ('s2','a1') = 1.745
   Qπ('s2','a2') = 2.518

【 iteratePolicy : No.=2 】
 Policy : π(a|s) ----------------------------------
   π('a1'|'s1') = 0.000
   π('a2'|'s1') = 1.000
   π('a1'|'s2') = 0.000
   π('a2'|'s2') = 1.000
 evaluatePolicy: repeated 25 times
 State-value function : Vπ(s) ---------------------
   Vπ('s1') = 6.636
   Vπ('s2') = 6.205
 Action-value function : Qπ(s,a) ------------------
   Qπ('s1','a1') = 3.968
   Qπ('s1','a2') = 6.636
   Qπ('s2','a1') = 5.371
   Qπ('s2','a2') = 6.205
```

(ﾘｽﾄ11-(07)-1_方策反復法 (方策評価と、ε-greedy法による方策改善))

```
#**************************************************************************
# ﾘｽﾄ11-(07)-1_方策反復法 (方策評価と、ε-greedy法による方策改善)
#**************************************************************************
import sys
import numpy as np

#**************************************************************************
# ｸﾗｽ     Environment
#--------------------------------------------------------------------------
# 環境のｸﾗｽ
#**************************************************************************
class Environment:

    #**********************************************************************
    # Environment ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
    #----------------------------------------------------------------------
    # 以下のｲﾝｽﾀﾝｽ変数を定義する：
    #    states        :状態値s（＝現在の位置座標(x,y)）の集合Ｓ
    #**********************************************************************
    def __init__(self):

        # 状態s の集合Ｓを、状態名を表す文字列のリストで作成
        self.states = ["s1", "s2"]

    #**********************************************************************
    # Environment ｸﾗｽ : transit
    # 状態s で行動a を取った時の状態遷移に伴う
    # （遷移確率Ｐss'a、報酬Ｒss'a、次の状態s'）のタプル型オブジェクトを
    # ﾘｽﾄで返す（これは(表3.2)を実装したもの）。
    #----------------------------------------------------------------------
    # 引数：
    #   self   :Agent のｲﾝｽﾀﾝｽ
    #   s      :現在の状態s
    #   a      :状態s でとる行動a
    #----------------------------------------------------------------------
    # 戻り値：
    # (第１返値) 以下の値の組のタプル型ｵﾌﾞｼﾞｪｸﾄ(p,r,s) の配列
    #      p   : 状態s で行動a を取った時の状態s'への遷移確率Ｐss'a
    #      r   : 状態s で行動a を取った時の状態s'への遷移に伴う報酬Ｒss'a
    #      s'  : 状態s で行動a を取った時の遷移先の状態s'
    #**********************************************************************
    def transit(self, s, a):

        if( s == "s1" ):
            if( a == "a1" ):
                return [ (0.3,  1.0, "s1"), (0.7, -2.0, "s2") ]
            elif( a == "a2" ):
                return [ (0.5,  1.0, "s1"), (0.5,  2.0, "s2") ]

        elif( s == "s2" ):
            if( a == "a1" ):
                return [ (0.6, -1.0, "s1"), (0.4,  2.0, "s2") ]
            elif( a == "a2" ):
                return [ (0.7,  1.0, "s1"), (0.3,  1.0, "s2") ]

        return [ (1.0,  0.0,   s ) ]
```

```
#*****************************************************************************
# ｸﾗｽ     HyperParam
#-----------------------------------------------------------------------------
# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀのｸﾗｽ
#*****************************************************************************
class HyperParam:

    #*************************************************************************
    # HyperParam ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
    #-------------------------------------------------------------------------
    # 以下のｲﾝｽﾀﾝｽ変数を定義し、指定値で初期化する：
    #
    #    epsilon       : ε-greedy法のパラメータ ε
    #    gamma         : 割引率γ (0≦γ＜1)
    #    delta_thresh  : 状態価値関数Ｖπ(s) の更新量のしきい値
    #                    (これより更新量が小さければ、状態価値関数の更新を終了する)
    #
    #*************************************************************************
    def __init__(self, epsilon, gamma, delta_thresh ):

        # epsilon : ε-greedy法のパラメータ ε
        self.epsilon = epsilon

        # gamma : 割引率γ
        self.gamma = gamma

        # delta_thresh  : 状態価値関数Ｖπ(s) の更新量のしきい値
        #                 (これより更新量が小さければ更新を終了する)
        self.delta_thresh = delta_thresh

    #*************************************************************************
    # HyperParam ｸﾗｽ : showHyperParameters
    # ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを出力する。
    #-------------------------------------------------------------------------
    # 引数：なし
    #*************************************************************************
    def showHyperParameters(self):
        print( "Hyper parameters :")
        print( "  (1) ε={0:.3f}".format(self.epsilon) )
        print( "  (2) γ={0:.3f}".format(self.gamma) )
        print( "  (3) delta_thresh={0}".format(self.delta_thresh) )
```

```
#*****************************************************************************
# クラス     Agent
#-----------------------------------------------------------------------------
# ｴｰｼﾞｪﾝﾄのｸﾗｽ
# ε-greedy法を用いた方策反復法による方策改善の（表3.2）への適用
#*****************************************************************************
class Agent:

    #*************************************************************************
    # Agent ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
    #-------------------------------------------------------------------------
    # 以下のｲﾝｽﾀﾝｽ変数を定義し初期化する：
    #
    #   actions  ：行動aの集合Ａを、行動名を表す文字列のリストで作成
    #   pai      ：方策πを辞書型で作成
    #               ｷｰ：(s,a)、ﾊﾞﾘｭｰ：π(a|s)
    #               方策の初期値：行動 "a1"を50%、"a2"を50%の割合で選択
    #   v_pais   ：状態s の状態価値関数Ｖπ(s)を辞書型で作成し、0で初期化
    #               ｷｰ：s(状態名を表す文字列型)、ﾊﾞﾘｭｰ：Ｖπ(s)
    #   q_paisa  ：状態s での行動a の行動価値関数Ｑπ(s,a)を辞書型で作成し
    #              0で初期化
    #               ｷｰ：(s,a) のタプル型、ﾊﾞﾘｭｰ：Ｑπ(s,a)
    #-------------------------------------------------------------------------
    # 引数：
    #   env      ：Environment のｲﾝｽﾀﾝｽ
    #*************************************************************************
    def __init__(self, env):

        # 行動aの集合Ａを、行動名を表す文字列のリストで作成
        self.actions = ["a1", "a2"]

        # 方策πを辞書型で作成
        #     ｷｰ：(s,a)、ﾊﾞﾘｭｰ：π(a|s)
        #     方策の初期値：行動 "a1" と "a2" を平等の割合で選択する
        self.pai = {}
        for s in env.states:
            for a in self.actions:
                self.pai[(s, a)] = 1.0/len(self.actions)

        # 状態(=位置)の状態価値関数Ｖπ(s)を辞書型で作成し、0で初期化
        # ｷｰ：s(位置：2次元座標を表すタプル型)、ﾊﾞﾘｭｰ：Ｖπ(s)
        self.v_pais = {}
        for s in env.states:
            self.v_pais[s] = 0

        # 状態s での行動a 行動価値関数Ｑπ(s,a)を辞書型で作成し、0で初期化
        # ｷｰ：(s,a) のタプル型、ﾊﾞﾘｭｰ：Ｑπ(s,a)
        self.q_paisa = {}
        for s in env.states:
            for a in self.actions:
                self.q_paisa[(s,a)] = 0
```

```
    #**************************************************************************
    # Agent ｸﾗｽ : evaluatePolicy
    # Agent のｲﾝｽﾀﾝｽに登録済の方策πの下での、
    # 状態価値関数Ｖπ(s)を算出する。
    #--------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #   param    : HyperParam のｲﾝｽﾀﾝｽ
    #--------------------------------------------------------------------------
    # 戻り値：    なし
    #**************************************************************************
    def evaluatePolicy(self, env, param):

        # 状態価値関数Ｖπ(s) の値が収束するまで繰り返す
        repeatNo = 0
        while True:
            repeatNo += 1

            # 状態価値関数Ｖπ(s) の更新量の最大値を初期化
            delta_max = 0

            # 全ての状態s について繰り返す
            for s in env.states:

                # 状態s の状態価値関数Ｖπ(s)を初期化する
                # v_pais_dash＝Ｖπ(s)＝Σa π(a|s) Σs' Pss'a× {Rss'a＋γＶπ(s')}
                v_pais_dash = 0

                # 全ての行動a について繰り返す
                for a in self.actions:

                    # 状態s で行動a を取った時の状態遷移に伴う (p,r,s_dash) をタプル型オブジェクトで持ち、
                    # その組をリスト(results) として取得する
                    #     p＝Pss'a：状態がs の時に、行動a を選択して状態がs'に遷移する確率
                    #     r＝Rss'a：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
                    #     s_dash＝s'：遷移先の状態s'
                    results = env.transit(s, a)

                    # ﾍﾞﾙﾏﾝ方程式で、状態価値関数Ｖπ(s) の行動a による寄与分を計算する
                    # （行動a による寄与分＝π(a|s) Σs' Pss'a× {Rss'a＋γＶπ(s')} ）
                    # （※ 次式で参照する agent.v_pais[s_dash] の設定値は、
                    #      状態価値関数Ｖπ(s)の計算を繰り返し行うことで収束する）
                    for p, r, s_dash in results:
                        v_pais_dash += self.pai[(s, a)] * p * (r + param.gamma * self.v_pais[s_dash])

                # 状態価値関数Ｖπ(s) の更新量の最大値を保存
                delta_max = max(delta_max, abs(self.v_pais[s] - v_pais_dash))

                # 状態s の状態価値関数Ｖπ(s)を更新
                self.v_pais[s] = v_pais_dash

            # 状態価値関数Ｖπ(s) の更新量の最大値(delta_max)がしきい値(delta_thresh)より小さければ
            # 状態価値関数Ｖπ(s) の値が収束したものとして、繰り返し計算を終了
            if delta_max < param.delta_thresh:
                print("  evaluatePolicy: repeated {0} times".format(repeatNo))
                break
```

```
    #**************************************************************************
    # Agent ｸﾗｽ : updatePolicy
    # 方策改善を ε-greedy法で行う。
    #--------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #   param    : HyperParam のｲﾝｽﾀﾝｽ
    #--------------------------------------------------------------------------
    # 戻り値：
    # (第１返値) updateFlag :方策確率πの更新有無フラグ
    #                        (True：更新がある、False：更新がない)
    #**************************************************************************
    def updatePolicy(self, env, param):
        # 方策確率πの更新有無フラグ
        updateFlag = False

        # 全ての状態s について繰り返す
        for s in env.states:

            # 最適行動価値関数Q*(s,a) を初期化
            q_max = -sys.maxsize -1

            # 最適行動価値関数Q*(s,a)を与える行動a* の集合Ａ*(s)をリストで作成
            a_asters = []

            # 全ての行動a について繰り返す
            for a in self.actions:

                # 状態s で行動a を取った時の状態遷移に伴う (p,r,s_dash) をタプル型オブジェクトで持ち、
                # その組をリスト(results) として取得する
                #     p＝Ｐss'a：状態がs の時に、行動a を選択して状態がs'に遷移する確率
                #     r＝Ｒss'a：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
                #     s_dash＝s'：遷移先の状態s'
                results = env.transit(s, a)

                # 状態s で行動a を取った時の行動価値関数Ｑπ(s,a) を計算する
                # q_paisa＝Ｑπ(s,a)＝Σs'Ｐss'a× {Ｒss'a＋γＶπ(s')}
                q_paisa = 0
                for p, r, s_dash in results:
                    q_paisa += p * (r + param.gamma * self.v_pais[s_dash])
                self.q_paisa[(s,a)] = q_paisa

                # 状態s での最適行動価値関数Q*(s,a)を更新し、
                # その時の行動a* も記録する。
                if q_paisa > q_max:
                    q_max = q_paisa
                    a_asters = [a]
                elif q_paisa == q_max:
                    a_asters.append(a)

            # M ：集合Ａ(s)の要素数
            # M*：集合Ａ*(s)の要素数
            M = len(self.actions)
            M_aster = len(a_asters)
            pai_aster = (1-param.epsilon)/M_aster + (param.epsilon / M)
            pai_a     = param.epsilon / M

            # (ε-greedy法による方策改善)
            # 着目している状態s での
            # (1) 行動a*           の方策確率π(a*|s)を (1-ε)/M*＋ε/M
            # (2) 他の行動a (a≠a*) の方策確率π(a |s)を          ε/M
            # に各々更新する。
```

```
        for a in self.actions:
            if( a in a_asters ):
                if( self.pai[(s, a)] != pai_aster ):
                    updateFlag = True
                    self.pai[(s, a)] = pai_aster
            else:
                if( self.pai[(s, a)] != pai_a ):
                    updateFlag = True
                    self.pai[(s, a)] = pai_a

    # 方策確率πの更新有無フラグを返す
    return updateFlag

#**************************************************************************
# Agent ｸﾗｽ : iteratePolicy
# 方策反復法による方策改善を行う。
#--------------------------------------------------------------------------
# 引数：
#   env      : Environment のｲﾝｽﾀﾝｽ
#   param    : HyperParam のｲﾝｽﾀﾝｽ
#--------------------------------------------------------------------------
# 戻り値：    なし
#**************************************************************************
def iteratePolicy(self, env, param):
    iterateNo = 0
    while True:
        iterateNo += 1
        print( "¥n【 iteratePolicy : No.={0} 】".format(iterateNo) )

        # 方策π(a|s) を出力
        self.showPolicy(env)

        # 方策πの下での、状態価値関数Vπ(s)を算出
        self.evaluatePolicy(env, param)

        # 状態価値関数Vπ(s) を出力
        self.showStateValues(env)

        # 方策改善を行う
        updateFlag = self.updatePolicy(env, param)

        # 行動価値関数Qπ(s,a) を出力する。
        self.showActionValues(env)

        # 方策に更新が無ければ繰り返し終了
        if not updateFlag:
            break

#**************************************************************************
# Agent ｸﾗｽ : showPolicy
# 方策π(a|s) を出力する。
#--------------------------------------------------------------------------
# 引数：
#   env      : Environment のｲﾝｽﾀﾝｽ
#**************************************************************************
def showPolicy(self, env):
    print( "  Policy : π(a|s) ---------------------------------")
    for s in env.states:
        for a in self.actions:
            print( "    π('{0}'|'{1}') = {2:.3f}".format(a, s, self.pai[(s, a)]))
```

```
    #************************************************************************
    # Agent ｸﾗｽ : showStateValues
    # 状態価値関数Vπ(s) を出力する。
    #------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #************************************************************************
    def showStateValues(self, env):
        print( "  State-value function : Vπ(s) ----------------------")
        for s in env.states:
            print( "    Vπ('{0}') = {1:.3f}".format(s, self.v_pais[s]))

    #************************************************************************
    # Agent ｸﾗｽ : showActionValues
    # 行動価値関数Qπ(s,a) を出力する。
    #------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #************************************************************************
    def showActionValues(self, env):
        print( "  Action-value function : Qπ(s,a) ------------------")
        for s in env.states:
            for a in self.actions:
                print( "    Qπ('{0}','{1}') = {2:.3f}".format(s, a, self.q_paisa[(s,a)]))


#****************************************************************************
# 主制御
#****************************************************************************

# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの指定
epsilon = 0.2
gamma = 0.8
delta_thresh = 0.001
param = HyperParam(epsilon, gamma, delta_thresh)

# 環境のｲﾝｽﾀﾝｽを作成
env = Environment()

# ｴｰｼﾞｪﾝﾄのｲﾝｽﾀﾝｽを作成
agent = Agent(env)

# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを出力
param.showHyperParameters()

# 方策反復法による方策改善
agent.iteratePolicy(env, param)
```

【出典・参考】

全般⇒「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)

全般⇒「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井 悦司)

全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

方策改善定理⇒
http://yagami12.hatenablog.com/entry/2019/02/22/210608#%E6%96%B9%E7%AD%96%E6%94%B9%E5%96%84%E5%AE%9A%E7%90%86

(7.2) 価値反復法

・ﾓﾃﾞﾙﾍﾞｰｽな手法の方策改善手法の一つに「価値反復法」があります。

(7.2.1) 価値反復法のｱﾙｺﾞﾘｽﾞﾑ

・「方策反復法」では方策評価と方策改善を交互に繰り返します。
方策評価ではﾍﾞﾙﾏﾝ方程式を解く必要があり、計算ｺｽﾄが高くなります。
これに対し、ﾍﾞﾙﾏﾝ方程式を解く代わりに、最適ﾍﾞﾙﾏﾝ方程式を解くことにすれば、その解の下で
greedyな方策は最適方策となります。
この方法を「価値反復（ｶﾁﾊﾝﾌﾟｸ、value iteration)」法と言います。

・以下に価値反復法のｱﾙｺﾞﾘｽﾞﾑを記します：

価値反復（ｶﾁﾊﾝﾌﾟｸ、value iteration）のｱﾙｺﾞﾘｽﾞﾑ

(0) 以下のﾊﾟﾗﾒｰﾀは既知とします

$P_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移する確率

$R_{ss'}^{a}$ ：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬

γ ：割引率（$0 \leqq \gamma \leqq 1$）

ΔV_{thresh} ：状態価値関数$V_{\pi}(s)$ の更新量のしきい値

（これより最適状態価値関数$V^{*}(s)$の最大値の更新量が小さければ、
本ｱﾙｺﾞﾘｽﾞﾑを終了する）

(1) 初期化

(1.1) 全ての状態$s \in S$に対して、状態価値関数の値$V_{\pi}(s)$ を任意に初期化する。

(2) 方策改善

現方策$\pi(a|s)$ を以下のようにして改善する：

(2.1) $\Delta V_{\pi max} \leftarrow 0$

(2.2) 全ての状態$s \in S$ について、

以下の手順で、最適状態価値関数$V^{*}(s)$の計算と方策改善を繰り返す：

(2.2.1) $Q_{\pi max}(s)$ ← (式4-10)で計算される$Q_{\pi}(s,a)$ではあり得ない小さい値

(2.2.2) $A^{*}(s)$ ← Φ（空集合）

(2.2.3) 全ての行動$a \in A(s)$について以下を繰り返し、

状態s での最適行動価値関数$Q_{\pi max}(s)$を特定する：

(1) 状態s で行動a を取る時の行動価値関数$Q_{\pi}(s,a)$を計算する：

$$Q_{\pi}(s,a) \leftarrow \Sigma_{s'} P_{ss'}^{a} \times \{R_{ss'}^{a} + \gamma V_{\pi}(s')\} \quad \cdots(式4\text{-}10)$$

(2) 計算した$Q_{\pi}(s,a)$により、最適行動価値関数$Q_{\pi max}(s)$の更新を行う：

(a) $Q_{\pi max}(s) > Q_{\pi}(s,a)$ の時、

何もしない

(b) $Q_{\pi max}(s) = Q_{\pi}(s,a)$ の時

集合$A^{*}(s)$ の要素として行動a を追加する

(c) $Q_{\pi max}(s) < Q_{\pi}(s,a)$ の時

$Q_{\pi max}(s) \leftarrow Q_{\pi}(s,a)$

集合$A^{*}(s)$ の要素を行動a のみとする

(2.2.4) 状態s での最適行動価値関数$Q^*(s,a)$ を$Q_{\pi max}(s)$で更新する。

$$Q^*(s,a) \leftarrow Q_{\pi max}(s)$$

(2.2.5) 状態s での全ての行動$a \in A(s)$について、
方策確率$\pi(a|s)$を greedy法で更新する：

$$\pi(a|s) \leftarrow \begin{cases} 1/M^* & \text{for } a \in A^*(s) \\ 0 & \text{for } a \notin A^*(s) \end{cases}$$

(2.2.6) 上記手順で算出された最適行動価値関数$Q^*(s,a)$を
状態s での最適状態価値関数$V^*(s)$とする(※1)。

$$V^*(s) \leftarrow Q^*(s,a)$$

(2.2.7) 更新された最適状態価値関数$V^*(s)$と状態s での更新前の状態価値関数$V_\pi(s)$
との差分(更新量)で、$\Delta V_{\pi max}$ を更新する：

$$\Delta V_{\pi max} \leftarrow \max\{\Delta V_{\pi max}, abs(V_\pi(s) - V^*(s))\}$$

(2.2.8) 状態s での最適状態価値関数$V^*(s)$を状態価値関数$V_\pi(s)$として保存する

$$V_\pi(s) \leftarrow V^*(s)$$

(2.3) 上記手順(2.1)～(2.2)で得た、全ての状態$s \in S$を通しての
最適状態価値関数$V^*(s)$の変化量の最大値$\Delta V_{\pi max}$ をみて反復継続を判断する：
$\Delta V_{\pi max} < \Delta V_{thresh}$ の場合、本「価値反復」ｱﾙｺﾞﾘｽﾞﾑを終了。
$\Delta V_{\pi max} \geqq \Delta V_{thresh}$ の場合、本「価値反復」ｱﾙｺﾞﾘｽﾞﾑを継続。手順(2.1)へ。

$V_\pi(s)$ ：状態がs の時に、方策πの下での状態価値関数
$\pi(a|s)$ ：状態がs の時に、行動a を選択する方策確率
$Q_\pi(s,a)$ ：状態がs で行動a を取り、状態遷移後は方策πに従った時の行動価値関数
集合$A(s)$ ：状態s でとりうる行動aの集合
集合$A^*(s)$ ：状態s での行動価値関数$Q_\pi(s,a)$が最大となる行動aの集合($A^*(s) \subset A(s)$)
M^* ：集合$A^*(s)$の要素数

(※1) この置換

$$V^*(s) \leftarrow Q^*(s,a)$$

は、状態価値関数$V_\pi(s)$と行動価値関数$Q_\pi(s,a)$の関係式

$$V_\pi(s) = \Sigma_a \pi(a|s) \times Q_\pi(s,a) \qquad \cdots(式4\text{-}11)再掲$$

で、上記手順(2.2.5)の「確率的方策」$\pi(a|s)$の設定と、
(2.2.4)の最適行動価値関数$Q^*(s,a)$を用いて得られます：

$$\begin{aligned} V_\pi(s) &= \Sigma_a \pi(a|s) \times Q_\pi(s,a) \\ &= \Sigma_{a \in A^*(s)} \pi(a|s) \times Q_\pi(s,a) + \Sigma_{a \notin A^*(s)} \pi(a|s) \times Q_\pi(s,a) \\ (2.2.5)\rightarrow &= \Sigma_{a \in A^*(s)} (1/M^*) \times Q_\pi(s,a) + \Sigma_{a \notin A^*(s)} 0 \times Q_\pi(s,a) \\ &= (1/M^*) \times \Sigma_{a \in A^*(s)} Q_\pi(s,a) \\ (2.2.4)\rightarrow &= (1/M^*) \times (M^* \times Q^*(s,a)) \\ &= Q^*(s,a) \qquad a \in A^*(s) \end{aligned}$$

特に$M^*=1$の時は、状態s の時に特定の行動a のみを選択することになり、
「確率的方策」$\pi(a|s)$が1.0 で、「決定論的方策」$\pi(s)=a$ に等しくなります。

(7.2.2) 価値反復法の実装例

・以下に、価値反復法の実装例を示します。
この例は、環境ﾓﾃﾞﾙが既知の(表3.2) の状態遷移表について、方策πの最適解を得ようとする実装です。

(1) 初期値を以下のように与えます：
(1-1) ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの設定を以下のようにします；
・割引率γ＝0.8
・価値関数の変化分⊿$V_{\pi max}$ が十分小さくなったという判定の値⊿V_{thresh}＝0.001
(1-2) 状態価値関数$V_{\pi}(s)$の初期値として、0を設定します。
(1-3) 方策πの初期値として、各状態s(∈Ｓ) の時に、行動a(∈Ａ) を行う割合が平等になるように均等な遷移確率を与えます：
・$\pi(a|s)$ ＝ 1/ 集合Ａの要素数 ＝ 1/2 ＝ 0.5
(2) 価値反復法により、最適状態価値関数$V^*(s)$を算出し、方策(確率) $\pi(a|s)$ を更新する・・・
ということを繰り返して、方策の最適化を行います。

・実行結果を以下に掲載します。
最終結果は、方策反復法でε=0.0 としたgreedy法のものとほぼ同じ結果になっています。

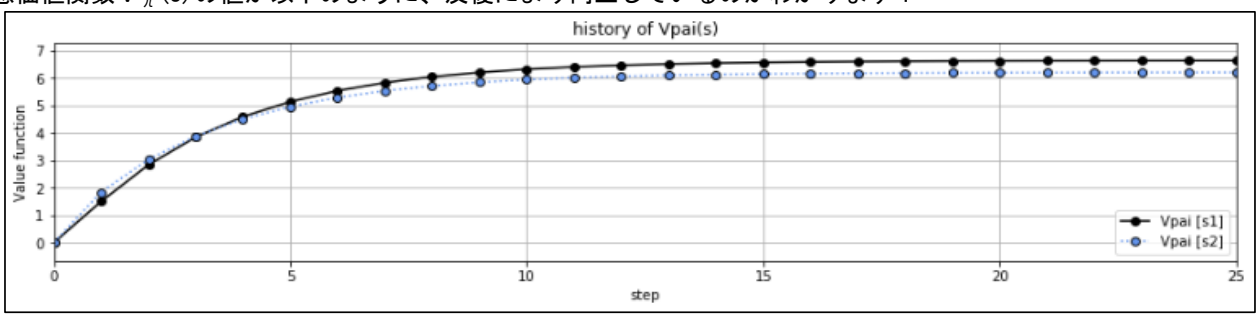
```
Hyper parameters :
  (1) γ=0.800
  (2) delta_thresh=0.001
  (3) iterate_max=100

【 iterateValue : No.=1 】
 Policy : π(a|s) -----------------------------------
   π('a1'|'s1') = 0.500
   π('a2'|'s1') = 0.500
   π('a1'|'s2') = 0.500
   π('a2'|'s2') = 0.500
 State-value function : Vπ(s) ---------------------
   Vπ('s1') = 0.000
   Vπ('s2') = 0.000
 Action-value function : Qπ(s,a) ------------------
   Qπ('s1','a1') = 0.000
   Qπ('s1','a2') = 0.000
   Qπ('s2','a1') = 0.000
   Qπ('s2','a2') = 0.000
```

：(途中、省略)

```
【 iterateValue : No.=26 】
 Policy : π(a|s) -----------------------------------
   π('a1'|'s1') = 0.000
   π('a2'|'s1') = 1.000
   π('a1'|'s2') = 0.000
   π('a2'|'s2') = 1.000
 State-value function : Vπ(s) ---------------------
   Vπ('s1') = 6.635
   Vπ('s2') = 6.204
 Action-value function : Qπ(s,a) ------------------
   Qπ('s1','a1') = 3.966
   Qπ('s1','a2') = 6.635
   Qπ('s2','a1') = 5.370
   Qπ('s2','a2') = 6.204
```

・状態価値関数$V_{\pi}(s)$の値が以下のように、反復により向上しているのがわかります：



(ﾘｽﾄ 11-(7)-2) 価値反復法の実装例

```
#******************************************************************************
# ﾘｽﾄ11-(07)-2_価値反復法による方策改善
#******************************************************************************
import sys
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


#******************************************************************************
# ｸﾗｽ     Environment
#------------------------------------------------------------------------------
# 環境のｸﾗｽ
#******************************************************************************
class Environment:

    #**************************************************************************
    # Environment ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
    #--------------------------------------------------------------------------
    # 以下のｲﾝｽﾀﾝｽ変数を定義する：
    #    states        ：状態値s（＝現在の位置座標(x,y)）の集合Ｓ
    #**************************************************************************
    def __init__(self):

        # 状態s の集合Ｓを、状態名を表す文字列のリストで作成
        self.states = ["s1", "s2"]

    #**************************************************************************
    # Environment ｸﾗｽ : transit
    # 状態s で行動a を取った時の状態遷移に伴う
    # （遷移確率Ｐss'a、報酬Ｒss'a、次の状態s'）のタプル型オブジェクトを
    # ﾘｽﾄで返す（これは(表3.2)を実装したもの）。
    #--------------------------------------------------------------------------
    # 引数：
    #   self   ：Agent のｲﾝｽﾀﾝｽ
    #   s      ：現在の状態s
    #   a      ：状態s でとる行動a
    #--------------------------------------------------------------------------
    # 戻り値：
    # （第１返値）以下の値の組のタプル型ｵﾌﾞｼﾞｪｸﾄ(p,r,s) の配列
    #      p    ：状態s で行動a を取った時の状態s'への遷移確率Ｐss'a
    #      r    ：状態s で行動a を取った時の状態s'への遷移に伴う報酬Ｒss'a
    #      s'   ：状態s で行動a を取った時の遷移先の状態s'
    #**************************************************************************
    def transit(self, s, a):

        if( s == "s1" ):
            if( a == "a1" ):
                return [ (0.3,  1.0, "s1"), (0.7, -2.0, "s2") ]
            elif( a == "a2" ):
                return [ (0.5,  1.0, "s1"), (0.5,  2.0, "s2") ]

        elif( s == "s2" ):
            if( a == "a1" ):
                return [ (0.6, -1.0, "s1"), (0.4,  2.0, "s2") ]
            elif( a == "a2" ):
                return [ (0.7,  1.0, "s1"), (0.3,  1.0, "s2") ]

        return [ (1.0,  0.0,   s ) ]
```

```
#**************************************************************************
# ｸﾗｽ     HyperParam
#--------------------------------------------------------------------------
# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀのｸﾗｽ
#**************************************************************************
class HyperParam:

    #**********************************************************************
    # HyperParam ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
    #----------------------------------------------------------------------
    # 以下のｲﾝｽﾀﾝｽ変数を定義し、指定値で初期化する：
    #
    #    gamma         ：割引率γ (0≦γ＜1)
    #    delta_thresh  ：状態価値関数Vπ(s) の更新量のしきい値
    #                （これより更新量が小さければ、状態価値関数の更新を終了する）
    #    iterate_max   ：価値反復の最大繰り返し回数
    #
    #   ------------------------------------------------------------------
    #
    #**********************************************************************
    def __init__(self, gamma, delta_thresh, iterate_max ):

        # gamma：割引率γ
        self.gamma = gamma

        # delta_thresh  ：状態価値関数Vπ(s) の更新量のしきい値
        #                （これより更新量が小さければ更新を終了する）
        self.delta_thresh = delta_thresh

        # iterate_max ： 価値反復の最大繰り返し回数
        self.iterate_max = iterate_max



    #**********************************************************************
    # HyperParam ｸﾗｽ : showHyperParameters
    # ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを出力する。
    #----------------------------------------------------------------------
    # 引数：無し
    #**********************************************************************
    def showHyperParameters(self):
        print( "Hyper parameters :")
        print( "  (1) γ={0:.3f}".format(self.gamma) )
        print( "  (2) delta_thresh={0}".format(self.delta_thresh) )
        print( "  (3) iterate_max={0}".format(self.iterate_max) )
```

```
#*************************************************************************
# クラス     Agent
#-------------------------------------------------------------------------
# ｴｰｼﾞｪﾝﾄのｸﾗｽ
# 価値反復法による方策改善：(表3.2) への適用
#*************************************************************************
class Agent:

    #*********************************************************************
    # Agent ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
    #---------------------------------------------------------------------
    # 以下のｲﾝｽﾀﾝｽ変数を定義し初期化する：
    #
    #   actions  :行動aの集合Ａを、行動名を表す文字列のリストで作成
    #   pai      :方策πを辞書型で作成
    #               ｷｰ：(s,a)、ﾊﾞﾘｭｰ：π(a|s)
    #               方策の初期値：行動 "a1"を50%、"a2"を50%の割合で選択
    #   v_pais   :状態s の状態価値関数Ｖπ(s)を辞書型で作成し、0で初期化
    #               ｷｰ：s(状態名を表す文字列型)、ﾊﾞﾘｭｰ：Ｖπ(s)
    #   q_paisa  :状態s での行動a の行動価値関数Ｑπ(s,a)を辞書型で作成し
    #             0で初期化
    #               ｷｰ：(s,a) のタプル型、ﾊﾞﾘｭｰ：Ｑπ(s,a)
    #---------------------------------------------------------------------
    # 引数：
    #   env      :Environment のｲﾝｽﾀﾝｽ
    #*********************************************************************
    def __init__(self, env):

        # 行動aの集合Ａを、行動名を表す文字列のリストで作成
        self.actions = ["a1", "a2"]

        # 方策πを辞書型で作成
        #     ｷｰ：(s,a)、ﾊﾞﾘｭｰ：π(a|s)
        #     (実は初期化は不要だが、履歴表示の為に初期化しておく)
        self.pai = {}
        for s in env.states:
            for a in self.actions:
                self.pai[(s, a)] = 1.0/len(self.actions)

        # 状態(=位置)の状態価値関数Ｖπ(s)を辞書型で作成し、0で初期化
        # ｷｰ：s(位置：2次元座標を表すタプル型)、ﾊﾞﾘｭｰ：Ｖπ(s)
        self.v_pais = {}
        for s in env.states:
            self.v_pais[s] = 0

        # 状態s での行動a 行動価値関数Ｑπ(s,a)を辞書型で作成し、0で初期化
        # ｷｰ：(s,a) のタプル型、ﾊﾞﾘｭｰ：Ｑπ(s,a)
        #     (実は初期化は不要だが、履歴表示の為に初期化しておく)
        self.q_paisa = {}
        for s in env.states:
            for a in self.actions:
                self.q_paisa[(s,a)] = 0


    #*********************************************************************
    # Agent ｸﾗｽ : updatePolicy
    # 状態s での最適行動価値関数Ｑ*(s,a)を取得し、
    # 状態s での方策確率π(a |s) の改善を greedy法で行う。
    #---------------------------------------------------------------------
    # 引数：
    #   env      :Environment のｲﾝｽﾀﾝｽ
    #   param    :HyperParam のｲﾝｽﾀﾝｽ
```

```
#   s       ：状態s
#-----------------------------------------------------------------
# 戻り値：
# （第１返値）q_aster ： 状態s での最適行動価値関数Ｑ*(s,a)
#*****************************************************************
def updatePolicy(self, env, param, s):

    # 最適行動価値関数Ｑ*(s,a) を初期化
    q_aster = -sys.maxsize -1

    # 最適行動価値関数Ｑ*(s,a)を与える行動a* の集合Ａ*(s)をリストで作成
    a_asters = []

    # 全ての行動a について繰り返す
    for a in self.actions:

        # 状態s で行動a を取った時の状態遷移に伴う (p,r,s_dash) をタプル型オブジェクトで持ち、
        # その組をリスト(results) として取得する
        #     p＝Ｐss'a：状態がs の時に、行動a を選択して状態がs'に遷移する確率
        #     r＝Ｒss'a：状態がs の時に、行動a を選択して状態がs'に遷移した時の報酬
        #     s_dash＝s'：遷移先の状態s'
        results = env.transit(s, a)

        # 状態s で行動a を取った時の行動価値関数Ｑπ(s,a) を計算する
        # q_paisa＝Ｑπ(s,a)＝Σs'Ｐss'a× {Ｒss'a＋γＶπ(s')}
        q_paisa = 0
        for p, r, s_dash in results:
            q_paisa += p * (r + param.gamma * self.v_pais[s_dash])
        self.q_paisa[(s,a)] = q_paisa

        # 状態s での最適行動価値関数Ｑ*(s,a)を更新し、
        # その時の行動a* も記録する。
        if q_paisa > q_aster:
            q_aster = q_paisa
            a_asters = [a]
        elif q_paisa == q_aster:
            a_asters.append(a)

    # M*：集合Ａ*(s)の要素数
    M_aster = len(a_asters)
    pai_aster = 1/M_aster
    pai_a     = 0

    # (greedy法による方策改善)
    # 着目している状態s での
    # (1) 行動a*           の方策確率π(a*|s)を 1/M*
    # (2) 他の行動a (a≠a*) の方策確率π(a |s)を 0
    # に各々更新する。
    for a in self.actions:
        if( a in a_asters ):
            self.pai[(s, a)] = pai_aster
        else:
            self.pai[(s, a)] = pai_a

    # 状態s での最適行動価値関数Ｑ*(s,a)を返す
    return q_aster

#*****************************************************************
# Agent ｸﾗｽ ： iterateValue
# 価値反復法による方策改善を行う。
```

```
    #---------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #   param    : HyperParam のｲﾝｽﾀﾝｽ
    #---------------------------------------------------------------------
    # 戻り値：    なし
    #*********************************************************************
    def iterateValue(self, env, param):
        # 状態価値関数の更新履歴
        vpiHistory = np.zeros((len(env.states), param.iterate_max))

        # 終了条件に達するまで繰り返し
        iterateNo = 0
        while True:
            # 繰り返し最大回数に達したか？
            iterateNo += 1
            if( iterateNo > param.iterate_max ):
                break

            print( "¥n【 iterateValue : No.={0} 】".format(iterateNo) )

            # 方策π(a|s) を出力
            self.showPolicy(env)

            # 状態価値関数Vπ(s) を出力
            self.showStateValues(env)

            # 行動価値関数Qπ(s,a) を出力
            self.showActionValues(env)

            # 状態価値関数Vπ(s) を履歴に保存
            sii = 0
            for s in env.states:
                vpiHistory[sii, iterateNo-1] = self.v_pais[s]
                sii += 1

            # 状態s の全体に渡る最適状態価値関数V*(s)の変化の最大値
            delta_max = 0

            # 状態s についての繰り返し
            for s in env.states:

                # 状態s での最適行動価値関数Q*(s,a)を得て、
                # 状態s について方策改善を行う。
                # 価値反復法では、Q*(s,a)が、状態s での最適状態価値関数V*(s)になる
                q_aster = self.updatePolicy(env, param, s)
                v_aster = q_aster

                # 状態s での最適状態価値関数V*(s)の変化分を計算する
                delta_max = max(delta_max, abs(self.v_pais[s] - v_aster))

                # 状態s での最適状態価値関数V*(s)を状態価値関数V(s)として保存する
                self.v_pais[s] = v_aster

            # 状態s の全体に渡る最適状態価値関数V*(s)の変化の最大値が
            # しきい値より小さい場合、反復を終了する
            if delta_max < param.delta_thresh:
                break

        # 最適状態価値関数V*(s)の履歴のｸﾞﾗﾌ表示
        self.showVpiHistory(env, iterateNo, vpiHistory)
```

```
    #**************************************************************************
    # Agent ｸﾗｽ : showPolicy
    # 方策π(a|s) を出力する。
    #--------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #**************************************************************************
    def showPolicy(self, env):
        print( "  Policy : π(a|s) ----------------------------------")
        for s in env.states:
            for a in self.actions:
                print( "    π('{0}'|'{1}') = {2:.3f}".format(a, s, self.pai[(s, a)]))

    #**************************************************************************
    # Agent ｸﾗｽ : showStateValues
    # 状態価値関数Vπ(s) を出力する。
    #--------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #**************************************************************************
    def showStateValues(self, env):
        print( "  State-value function : Vπ(s) ---------------------")
        for s in env.states:
            print( "    Vπ('{0}') = {1:.3f}".format(s, self.v_pais[s]))

    #**************************************************************************
    # Agent ｸﾗｽ : showActionValues
    # 行動価値関数Qπ(s,a) を出力する。
    #--------------------------------------------------------------------------
    # 引数：
    #   env      : Environment のｲﾝｽﾀﾝｽ
    #**************************************************************************
    def showActionValues(self, env):
        print( "  Action-value function : Qπ(s,a) ------------------")
        for s in env.states:
            for a in self.actions:
                print( "    Qπ('{0}','{1}') = {2:.3f}".format(s, a, self.q_paisa[(s,a)]))

    #**************************************************************************
    # Agent ｸﾗｽ : showVpiHistory
    # 最適状態価値関数V*(s) (＝状態価値関数Vπ(s)) の
    # 価値反復に伴う履歴のｸﾞﾗﾌ表示
    #--------------------------------------------------------------------------
    # 引数：
    #   env          : Environment のｲﾝｽﾀﾝｽ
    #   lastStep     : 繰り返しの最終ｽﾃｯﾌﾟ番号
    #   vpiHistory[] : 状態価値関数Vπ(s)のｽﾃｯﾌﾟ毎の履歴
    #**************************************************************************
    def showVpiHistory(self, env, lastStep, vpiHistory):
        # 表示範囲 (X)
        xlist = np.zeros((lastStep))
        for i in range(lastStep):
            xlist[i] = i

        # 表示範囲 (Y)
        yrange = np.max(vpiHistory[:,:]) - np.min(vpiHistory[:,:])
        ymin = np.min(vpiHistory[:,:]) - yrange * 0.1
        ymax = np.max(vpiHistory[:,:]) + yrange * 0.1

        # ｸﾞﾗﾌ表示
        plt.figure(figsize=(15, 3))
        plt.title("history of Vpai(s)")
```

```
        # 状態s についての繰り返し
        dispColors = ['black', 'cornflowerblue']
        dispStyles = ['-', ':']
        sii = 0
        for s in env.states:
            plt.plot(xlist, vpiHistory[sii,:lastStep], marker='o', linestyle=dispStyles[sii%2],
                markeredgecolor='black', color=dispColors[sii%2], label='Vpai [{0}]'.format(s))
            sii += 1

        plt.xlim(0, lastStep-1)
        plt.ylim(ymin, ymax)
        plt.xlabel("step")
        plt.ylabel("Value function")
        plt.legend(loc='lower right')
        plt.grid(True)
        plt.show()


#****************************************************************************
# 主制御
#****************************************************************************

# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの指定
gamma = 0.8
delta_thresh = 0.001
iterate_max = 100
param = HyperParam(gamma, delta_thresh, iterate_max)

# 環境のｲﾝｽﾀﾝｽを作成
env = Environment()

# ｴｰｼﾞｪﾝﾄのｲﾝｽﾀﾝｽを作成
agent = Agent(env)

# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを出力
param.showHyperParameters()

# 価値反復法による方策改善
agent.iterateValue(env, param)
```

【出典・参考】

全般⇒「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)

全般⇒「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井 悦司)

全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

(8) ﾓﾃﾞﾙﾌﾘｰな手法での価値関数と方策改善

・環境ﾓﾃﾞﾙの情報が未知、あるいは既知であっても大規模・複雑な場合、
ﾍﾞﾙﾏﾝ方程式を探索的な方法により近似的に解きます。
これを「ﾓﾃﾞﾙﾌﾘｰな手法 (Model-Free Reinforcement Learning)」と言います。

・方策改善するには、方策改善定理にあるように状態価値関数$V_{\pi}(s)$ と行動価値関数$Q_{\pi}(s,a)$ の推定値
が必要ですが、ﾓﾃﾞﾙﾌﾘｰな状況では、これらの計算に必要な状態遷移確率$P_{ss'}^{a}$ が与えられておらず、
ﾓﾃﾞﾙﾍﾞｰｽな手法とは別のｱﾌﾟﾛｰﾁが必要です。

・ﾓﾃﾞﾙﾌﾘｰな手法により方策πを改善する方法として、以下のような分類があります。

ﾓﾃﾞﾙﾌﾘｰな手法の分類その1 (価値ﾍﾞｰｽ／方策ﾍﾞｰｽ)

(1) 価値ﾍﾞｰｽの手法 (ｶﾁﾍﾞｰｽﾉｼｭﾎｳ、Value-Based Method)
・行動価値関数$Q_{\pi}(s,a)$により方策πを間接的に最適化する方法を「価値ﾍﾞｰｽの手法」と言います。
(例)「ﾓﾝﾃｶﾙﾛ法」「SARSA」「Q学習」

(2) 方策ﾍﾞｰｽの手法 (ﾎｳｻｸﾍﾞｰｽﾉｼｭﾎｳ、Plicy-Based Method)
・価値関数を用いることなく、方策πを直接的に最適化する方法を「方策ﾍﾞｰｽの手法」と言います。
(例)「方策勾配法」「Actor-Critic法」

・評価対象の方策πに従う探索をするかどうかで、以下のような学習方法の分類があります。

ﾓﾃﾞﾙﾌﾘｰな手法の分類その2 (方策ｵﾝ型／方策ｵﾌ型)

(1) 方策ｵﾝ型学習 (ﾎｳｻｸｵﾝｶﾞﾀｶﾞｸｼｭｳ、On-Policy Learning)
・方策改善にあたり、評価対象の方策πで、
探索しながら学習に必要な情報を収集する手法を「方策ｵﾝ型ﾃﾞｰﾀ収集」と言います。
「方策ｵﾝ型ﾃﾞｰﾀ収集」で収集したﾃﾞｰﾀによる学習を、「方策ｵﾝ型学習」と言います。
(例)「SARSA」

(2) 方策ｵﾌ型学習 (ﾎｳｻｸｵﾌｶﾞﾀｶﾞｸｼｭｳ、Off-Policy Learning)
・方策改善にあたり、評価対象の方策πにとらわれない方策π'で、
探索しながら学習に必要な情報を収集する手法を「方策ｵﾌ型ﾃﾞｰﾀ収集」と言います。
「方策ｵﾌ型ﾃﾞｰﾀ収集」で収集したﾃﾞｰﾀによる学習を、「方策ｵﾌ型学習」と言います。
(例)「Q学習」

・ﾓﾃﾞﾙﾌﾘｰな手法の場合、所与の環境の下で、探索的に状態・行動・報酬の系列をｻﾝﾌﾟﾘﾝｸﾞして
価値関数を推定により求めます。価値関数の推定方法に、以下のようなものがあります。

ﾓﾃﾞﾙﾌﾘｰな手法の分類その3 (ﾓﾝﾃｶﾙﾛ法／TD学習)

(1)「ﾓﾝﾃｶﾙﾛ法 (ﾓﾝﾃｶﾙﾛﾎｳ、Monte-Carlo Search)」
(2)「TD学習 (ﾃｨｰﾃﾞｨｰｶﾞｸｼｭｳ、Temporal Difference Search)」
(例)「SARSA」「Q学習」

【出典・参考】
全般⇒「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)
全般⇒「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井 悦司)
全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)
方策ｵﾝ型・ｵﾌ型⇒ https://qiita.com/ABmushi/items/83a639506fcbc4050ce8
価値ﾍﾞｰｽ・方策ﾍﾞｰｽ⇒ http://arduinopid.web.fc2.com/N66.html
ﾓﾝﾃｶﾙﾛ法⇒ https://qiita.com/sconeman/items/cb6047540a39b3bf15d7
ＴＤ学習⇒ https://qiita.com/sconeman/items/cb6047540a39b3bf15d7

(8.1) ﾓﾃﾞﾙﾌﾘｰな手法での例題

・ﾓﾃﾞﾙﾌﾘｰな手法の説明にあたり、手法間の相違点を見る為、
「Sutton本」にある「崖歩き」を取り上げて、各手法で実装してみます。

(例題) 崖歩き (Cliff Walking)

歩行者が縦4×横12の格子点上を、崖を避けながら上下左右に歩いて、
開始点(S 座標(0,3))から目標点(G 座標(11,3))までに至る最短経路を探します。

Y ↓	0	1	2	3	4	5	6	7	8	9	10	11	X→
0													
1	→	→	→	→	↓		→	→	→	→	→	↓	
2	↑				→	→	↑					↓	
3	S	X	X	X	X	X	X	X	X	X	X	G	

(1) 格子点の配置についての説明
- S と書かれた点は開始点(Start、座標(0,3))です。
- G と書かれた点は目標点(Goal、座標(11,3))です。
- X と書かれた領域は崖領域(座標(1,3)(2,3)…(10,3))です。
- 何も記載されていない点は、歩行者が移動可能な格子点です。

(2) 移動のﾙｰﾙと取りうる行動a の集合Ａ
- 一回の移動で、上下左右に1格子点のみ移動可能で、
  斜め方向や、2格子点以上飛躍する移動はできないものとします。
  移動ﾍﾞｸﾄﾙ(⊿X,⊿Y)で行動a を表現します。
- 崖領域に踏み込んだ場合、歩行者は強制的に開始点S に戻されるものとします。
- 移動によって縦4×横12の格子点以外の領域に踏み込む場合、移動はしません。

(3) 状態値s の集合Ｓ
- 格子点の座標値(X,Y) を状態値sとします(0≦X≦11、0≦Y≦3)。

(4) 状態遷移に伴う報酬$R_{ss'}^{a}$
- 目標点G への移動を報酬$R_{ss'}^{a}=0$ とします。
- 崖領域X への移動を報酬$R_{ss'}^{a}=-100$ とします。
- 上記以外の格子点への移動を報酬$R_{ss'}^{a}=-1$ とします。

この例題を解くにあたり、以下のようなﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを使用します：

(a) ﾓﾝﾃｶﾙﾛ法でのみ使用
- ε_{first} ：ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のﾊﾟﾗﾒｰﾀε ($0 \leqq \varepsilon \leqq 1$)の初期値
- ε_{last} ：ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のﾊﾟﾗﾒｰﾀε ($0 \leqq \varepsilon \leqq 1$)の最終値(終了判断値)
- $\Delta\varepsilon_{rate}$ ： εの1ｴﾋﾟｿｰﾄﾞ毎の減少率($0 < \Delta\varepsilon_{rate} < 1$、$\varepsilon \leftarrow \Delta\varepsilon_{rate} \times \varepsilon$)

(b) Q学習でのみ使用
- ε_{train} ：ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のﾊﾟﾗﾒｰﾀ ($0 \leqq \varepsilon \leqq 1$)

(c) ﾓﾝﾃｶﾙﾛ法とQ学習の共通で使用
- ε_{run} ：実行時のε-greedy法のﾊﾟﾗﾒｰﾀε ($0 \leqq \varepsilon \leqq 1$) ・・・ﾓﾝﾃｶﾙﾛ法、Q学習で使用
- γ ：割引率 ($0 \leqq \gamma \leqq 1$)・・・・・・・・・・・・・・・・・SARSA、Q学習で使用
- α ：重み係数 ($0 \leqq \alpha \leqq 1$)・・・・・・・・・・・・・・・・・SARSA、Q学習で使用

【出典・参考】
(1)「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)

・「(例題) 崖歩き (Cliff Walking)」問題について、
その環境と計算情報表示用ｸﾗｽの実装を以下に示します。

(ﾘｽﾄ11-(08)-1_崖歩き問題の環境と表示のｸﾗｽ)

```
#********************************************************************************
# ﾘｽﾄ11-(08)-1_崖歩き問題の環境と表示のｸﾗｽ
#********************************************************************************
import numpy as np
import matplotlib.pyplot as plt

#********************************************************************************
# ｸﾗｽ     Environment
#--------------------------------------------------------------------------------
# 崖歩きの環境のｸﾗｽ
#********************************************************************************
class Environment:

  #********************************************************************************
  # Environment ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #--------------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義する：
  #    cliff_img     : 崖歩きの格子点情報 (縦4×横12)
  #                      ・'S'：開始点
  #                      ・'G'：目標点
  #                      ・'X'：崖領域の格子点
  #                      ・上記以外：歩行者が移動可能な格子点
  #--------------------------------------------------------------------------------
  #    cliff         : 崖歩きの格子点情報を配列で表現
  #    size_y,size_x : 崖歩きの格子点ﾃﾞｰﾀのｻｲｽﾞ (X方向,Y方向)
  #    states        : 状態値s (＝現在の位置座標(x,y)) の集合Ｓ
  #    state_ini     : 初期状態値 (＝開始点の位置座標(x,y))
  #
  #********************************************************************************
  def __init__(self):
    self.cliff_img = '''
++++++++++++
++++++++++++
++++++++++++
SXXXXXXXXXXG
'''

    # 崖歩きの格子点ﾃﾞｰﾀを、行上で配置情報を表す文字の配列を1行とし、行の配列で格納する
    #      'S':開始点、'G':目標点、'X':崖領域、'':左記以外の通行可能領域
    self.cliff = []
    for line in self.cliff_img.split('¥n'):
      if line == '':
        continue
      self.cliff.append(list(line))

    # 崖歩きの格子点ﾃﾞｰﾀのｻｲｽﾞ (X方向,Y方向)
    self.size_y, self.size_x = len(self.cliff), len(self.cliff[0])

    # 状態値s (＝現在の位置座標(x,y)) の集合Ｓをﾀﾌﾟﾙ型(x,y)の配列として保持する
    self.states = [(x, y) for x in range(self.size_x) for y in range(self.size_y)]

    # 初期状態値s0 (＝開始点の位置座標(x,y)) ﾀﾌﾟﾙ型(x,y)として保持する
    for y in range(self.size_y):
      for x in range(self.size_x):
        if self.cliff[y][x] == 'S':
          self.state_ini = (x, y)
          break
```

```
#*****************************************************************************
# Environment ｸﾗｽ : getCliff
# 崖歩きの格子点情報を配列で取得するﾒｿｯﾄﾞ
#-----------------------------------------------------------------------------
# 引数：なし
#-----------------------------------------------------------------------------
# 戻り値：
# （第１返値）cliff ： 崖歩きの格子点情報
#*****************************************************************************
def getCliff(self):
  return self.cliff

#*****************************************************************************
# Environment ｸﾗｽ : transit
# 状態s で行動a を取った時の状態遷移に伴う
# 報酬Rss'a、次の状態s'を返す。
#-----------------------------------------------------------------------------
# 引数：
#   s     ： 現在の状態s （＝現在の位置座標(x,y)）
#   a     ： 状態s でとる行動a （＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#-----------------------------------------------------------------------------
# 戻り値：
# （第１返値）r  ： 状態s で行動a を取った時の状態s'への遷移に伴う報酬Rss'a
#                     0：目標点'G'への移動
#                  -100：崖領域'X' への移動
#                    -1：上記以外
# （第２返値）s' ： 状態s で行動a を取った時の遷移先の状態s'
#                （＝状態遷移後の位置座標(x',y')）)
#*****************************************************************************
def transit(self, s, a):

  # 現在の状態s （＝位置座標(x,y)）
  x, y = s
  # 現在の状態s でとる行動a （＝移動量(Δx,Δy)））
  dx, dy = a

  # 現在位置が出口の場合、状態・報酬共に不変
  if self.cliff[y][x] == 'G':
    return 0, s

  # 次状態へ遷移した場合、範囲外なら遷移せずに、報酬は -1
  if (y+dy < 0) or (y+dy >= self.size_y) :
      return -1, (x, y)
  if (x+dx < 0) or (x+dx >= self.size_x) :
      return -1, (x, y)

  # 次状態へ遷移した場合、崖領域'X' なら開始点'S'に戻る、報酬は -100
  if self.cliff[y+dy][x+dx] == 'X':
      x, y = self.state_ini
      return -100, (x, y)

  # 上記以外の場合、行動a のとおりに遷移を行う
  x += dx
  y += dy

  # 次状態へ遷移した場合、出口'G' なら、報酬は 0
  if self.cliff[y][x] == 'G':
      return 0, (x, y)

  # 上記以外なら、報酬は -1
  return -1, (x, y)
```

```
#******************************************************************************
# ｸﾗｽ     ShowFunc
#------------------------------------------------------------------------------
# 計算情報表示用ｸﾗｽ
#******************************************************************************
class ShowFunc:

  #******************************************************************************
  # ShowFunc ｸﾗｽ : showLearningCurve
  #------------------------------------------------------------------------------
  # 方策πの学習で、1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数の履歴をｸﾞﾗﾌ表示する
  #------------------------------------------------------------------------------
  # 引数：
  #   episodeLengthHistory ：1ｴﾋﾟｿｰﾄﾞ分の状態遷移数の履歴
  #******************************************************************************
  def showLearningCurve(self, episodeLengthHistory):
    xlist = range(len(episodeLengthHistory))
    plt.figure(figsize=(12,3))
    plt.plot( xlist, episodeLengthHistory, 'cornflowerblue', linewidth=3 )
    plt.title("Transition count every episode")
    plt.xlabel("Episode no")
    plt.ylabel("count")
    plt.grid(True)
    plt.show()

  #******************************************************************************
  # ShowFunc ｸﾗｽ : showQ
  #------------------------------------------------------------------------------
  # 行動価値関数Qπ(s,a)を表示する
  #------------------------------------------------------------------------------
  # 引数：
  #   agent    : Agent のｲﾝｽﾀﾝｽ
  #   env      : Environment のｲﾝｽﾀﾝｽ
  #******************************************************************************
  def showQ(self, agent, env):
    # 出力ﾏｯﾌﾟ
    maptbl = [['  ' for x in range(env.size_x)] for y in range(env.size_y)]

    # 出力ﾏｯﾌﾟに写しこみ
    for s in env.states:
      x, y = s
      aq_xy = ""
      for a in agent.actions:
        dx, dy = a
        q = agent.q[(s, a)]

        footprint = '  '
        if( dx == 0 and dy == -1 ):
          footprint = '↑'
        elif( dx == 0 and dy == 1 ):
          footprint = '↓'
        elif( dx == 1 and dy == 0 ):
          footprint = '→'
        elif( dx == -1 and dy == 0 ):
          footprint = '←'
        aq_xy = "{0}  {1}:{2:+7.2f}".format(aq_xy, footprint, q)
      maptbl[y][x] = aq_xy
```

```
  # 出力ﾏｯﾌﾟを表示
  print("行動価値関数Qπ(s,a)")
  for y in range(env.size_y):
    print("Y={0:3}: ".format(y))
    for x in range(env.size_x):
       print("({0:3},{1:3}):{2}".format(x, y, maptbl[y][x]))
  print("")

#*************************************************************************
# ShowFunc ｸﾗｽ : showFootprint
#-------------------------------------------------------------------------
# 1ｴﾋﾟｿｰﾄﾞ分の足跡を出力する
#-------------------------------------------------------------------------
# 引数:
#   agent    :Agent のｲﾝｽﾀﾝｽ
#   env      :Environment のｲﾝｽﾀﾝｽ
#   episode  :1ｴﾋﾟｿｰﾄﾞ分の状態遷移情報
#*************************************************************************
def showFootprint(self, agent, env, episode):

  # 状態遷移数
  print("ｴﾋﾟｿｰﾄﾞの状態遷移数: {0}".format(len(episode)))

  # 結果表示内容を初期化
  result = np.copy(env.cliff)
  for y in range(env.size_y):
    for x in range(env.size_x):
      if(result[y][x] == 'X' ):
        result[y][x] = '■'
      elif(result[y][x] == 'S' ):
        result[y][x] = 'Ｓ'
      elif(result[y][x] == 'G' ):
        result[y][x] = 'Ｇ'
      else:
        result[y][x] = '　'

  # 1ｴﾋﾟｿｰﾄﾞ分を取得する
  for (s, a, r) in episode:
    # 位置
    x, y = s

    # 方向
    dx, dy = a
    footprint = '＋'
    if( dx == 0 and dy == -1 ):
      footprint = '↑'
    elif( dx == 0 and dy == 1 ):
      footprint = '↓'
    elif( dx == 1 and dy == 0 ):
      footprint = '→'
    elif( dx == -1 and dy == 0 ):
      footprint = '←'

    # 格子上に足跡を残す
    result[y][x] = footprint

  #  1ｴﾋﾟｿｰﾄﾞ分の足跡を出力
  print("  ＋", end="")
  for x in range(env.size_x):
    print("－", end="")
  print("＋")
```

```
    ycoord = 0
    for y in range(env.size_y):
      print("{0:2d} |".format(ycoord), end="")
      for x in range(env.size_x):
        print(result[y][x], end="")
      print("|")
      ycoord += 1

    print("  ＋", end="")
    for x in range(env.size_x):
      print("－", end="")
    print("＋")
    print("    ", end="")
    for x in range(env.size_x):
      print("{0:2d}".format(x), end="")
    print()
```

【出典・参考】

(1)「強化学習」(2018年11月 第1版第11刷 森北出版 Richard S. Sutton and Andrew G.Barto)

(2)「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井悦司)

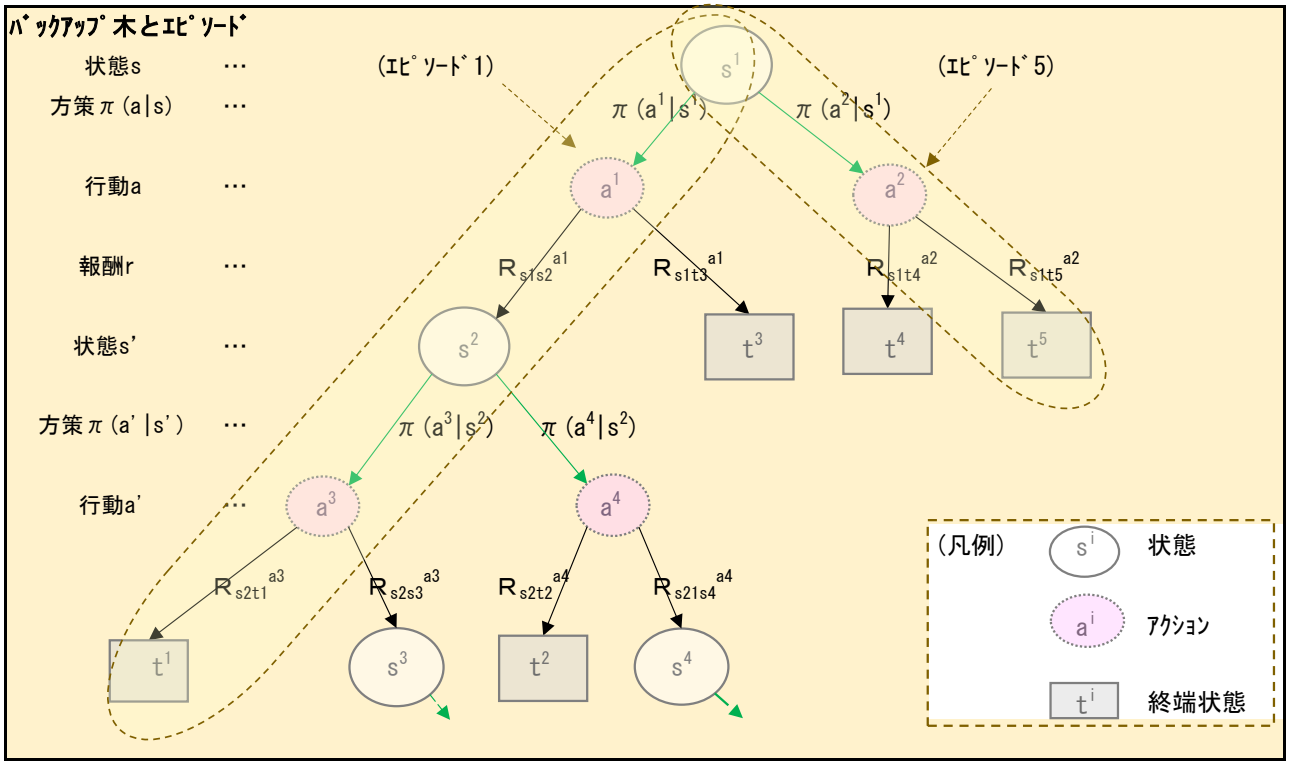
(8.2) ﾓﾝﾃｶﾙﾛ法

(8.2.1) ﾓﾝﾃｶﾙﾛ法とは

・一般に「ﾓﾝﾃｶﾙﾛ法 (ﾓﾝﾃｶﾙﾛﾎｳ、Monte Carlo method、Monte-Carlo Search)」は、
ｼﾐｭﾚｰｼｮﾝや数値計算を、乱数によりﾗﾝﾀﾞﾑに事象を発生させて行う手法のことです。

・強化学習のﾓﾝﾃｶﾙﾛ法では、以下のように乱数を用いて状態価値関数を推定します：

(1) 各ｴﾋﾟｿｰﾄﾞ毎に、初期状態から終端状態に至るまで乱数を用いて状態遷移をｼﾐｭﾚｰｼｮﾝし、
状態遷移情報（遷移元の状態s、取った行動a、遷移先の状態s'、遷移時の報酬$R_{ss'}^{a}$）を
ｻﾝﾌﾟﾘﾝｸﾞします(下図参照：(図3.5)再掲)。



(2) ｴﾋﾟｿｰﾄﾞ上での状態s への訪問時の割引報酬和G_k の総和を
訪問回数N_t で割って算出した平均値$V_t(s)$を、状態価値関数$V_\pi(s)$の推定値とします。

ﾓﾝﾃｶﾙﾛ法での状態価値関数$V_\pi(s)$の推定値

ﾓﾝﾃｶﾙﾛ法での状態価値関数$V_\pi(s)$の推定値は、以下の式で与えられます：

$$V_{t+1}(s) = \{ 1/N_{t+1}(s) \} \sum_{k=0}^{t} G_k \times I(S_k=s) \quad \cdots(式8\text{-}1)$$
$$N_{t+1}(s) = \sum_{k=0}^{t} I(S_k=s)$$

上記関係式は、以下の漸化式の形に書き換えられます(※1)：

$$V_{t+1}(s) = V_t(s) + \{ 1/N_{t+1}(s) \} \{ G_t - V_t(S_t) \} \times I(S_t=s) \quad \cdots(式8\text{-}2)$$
$$N_{t+1}(s) = N_t(s) + I(S_t=s)$$

ｻﾝﾌﾟﾘﾝｸﾞ平均の重み $1/N_{t+1}(s)$ を係数αとすると、以下のようになります：

$$V_{t+1}(s) = V_t(s) + \alpha \{ G_t - V_t(S_t) \} \times I(S_t=s) \quad \cdots(式8\text{-}3)$$

$I(S_k=s)$ ：時刻k(k=1～t)での状態S_k がs の時1、それ以外の時0 を返す関数
$N_t(s)$ ：時刻tで、全ｴﾋﾟｿｰﾄﾞ上で状態s への訪問回数
$V_t(s)$ ：時刻tで、状態価値関数の推定値
G_t ：時刻t から将来にわたる割引報酬和
α ：ｻﾝﾌﾟﾘﾝｸﾞ平均の重み $1/N_{t+1}(s)$ を係数として一般化した値

(※1) (式8-2)の導出について

・一個観測値を加えた時の平均値は、加える前の平均値を用いて、以下のように計算できます。

$$x_{mean\_n+1} = x_{mean\_n} + (x_{n+1} - x_{mean\_n}) / (n+1)$$ (ｾﾐﾅｰ6回目参照)

x_{mean_n+1} ：n+1個の観測値の平均値
x_{mean_n} ：n個の観測値の平均値
n+1 ：観測値の個数
x_i ：i番目の観測値 (i=1～n+1)

・この関係式を用いて、(式8-1)から(式8-2)が導かれます：

$$V_{t+1}(s) = \{ 1/N_{t+1}(s) \} \Sigma_{k=0}^{t} G_k \times I(S_k=s)$$ …(式8-1)再掲

$$= V_t(s) + \{ 1/N_{t+1}(s) \} \{ G_t - V_t(S_t) \} \times I(S_t=s)$$ …(式8-2)再掲

・ﾓﾝﾃｶﾙﾛ法では、ｴﾋﾟｿｰﾄﾞの終端状態に達してから開始状態に遡って、各々の状態遷移に伴う価値関数を特定します。このため、ｴﾋﾟｿｰﾄﾞ内の状態遷移時に随時に価値関数の計算を進めるような SARSA や Q学習よりも計算ｺｽﾄが高くなります。

(8.2.2) ﾓﾝﾃｶﾙﾛ法でのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)

・ﾓﾝﾃｶﾙﾛ法でのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)を以下に記します：

ﾓﾝﾃｶﾙﾛ法でのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)

(1) 方策を学習する

・ε-greedy法のﾊﾟﾗﾒｰﾀεを徐々に減少させながら、εが十分に小さくなるまで
ｴﾋﾟｿｰﾄﾞの取得による状態遷移情報のｻﾝﾌﾟﾘﾝｸﾞを行い、それに基づく方策改善を繰り返す。

(1.1) 初期設定

(1.1.1) あり得る状態値s の集合Ｓを作成する

(1.1.2) 取りうる行動a の集合Ａを作成する

(1.1.3) 決定論的方策π を初期化する
(乱数でπ(s)に対応する行動aを決定、s∈Ｓ, a∈Ａ)

(1.1.4) 状態s で行動a を取った時の行動価値関数$Q_\pi(s,a)$ を初期化する

(1.1.5) 状態s で行動a を取るｴﾋﾟｿｰﾄﾞ上での訪問回数Ｎ(s,a) をＯで初期化する

(1.1.6) ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のﾊﾟﾗﾒｰﾀεを設定する (0≦ε≦1)
最初は探索範囲を大きくとる為、εを大きめの値で初期化する。

(1.2) 1回分のｴﾋﾟｿｰﾄﾞを取得

以下の処理を初期状態から終端状態まで繰り返し、1ｴﾋﾟｿｰﾄﾞ分の遷移情報
(状態s、行動a、遷移先状態s'、遷移時報酬$R_{ss'}^{a}$ の系列) を蓄積する

(1st) 状態s を初期状態で設定

(2nd) 状態s でとる行動a をε-greedy法で選択する
・εの割合で、ﾗﾝﾀﾞﾑに行動a を選択する　　：a ←Ａから乱数で抽出
・1-εの割合で、方策πに従って行動a を選択する：a ←π(s)

(3rd) 状態s で上記で選択した行動a を取り、状態遷移に伴う次の量を計算する。
・遷移による次状態 s'
・遷移時報酬 $R_{ss'}^{a}$

(4th) 遷移先状態s' が終端状態の場合、手順(1.2)を終了する。

(5th) 状態s を遷移先状態s' で更新し、手順(2nd)へ
s←s'

(1.3) 取得した1回分のｴﾋﾟｿｰﾄﾞについて、

終端状態から初期状態に向かって状態遷移情報を遡って参照し、方策改善を行う。

(1st) 総報酬Ｒを０で初期化：Ｒ←０
(Ｒは最終的には総報酬になるが、本ｴﾋﾟｿｰﾄﾞを終端状態から初期状態に遡って
状態遷移情報を参照してゆく場合、各状態s での総報酬になる)

(2nd) ｴﾋﾟｿｰﾄﾞ上の「着目先」状態を終端状態とする。

(3rd) 「着目先」状態遷移情報から、次の量を取得する：
・s ：遷移元の状態
・a ：状態s で取った行動
・$R_{ss'}^{a}$ ：状態遷移時報酬

(4th) 総報酬Ｒに、$R_{ss'}^{a}$を追加：
Ｒ←Ｒ＋$R_{ss'}^{a}$

(5th) 状態s で行動a を取るｴﾋﾟｿｰﾄﾞ上での訪問回数Ｎ(s, a) を更新：
Ｎ(s, a) ←Ｎ(s, a)＋１

(6th) 状態s で行動a を取る行動価値関数Q_π(s, a) を
報酬$R_{ss'}^{a}$ の総和の平均値として算出する：
Q_π(s, a) ←Q_π(s, a)＋｛Ｒ－Q_π(s, a)｝／Ｎ(s, a)

(7th) 状態s での方策π(s)(決定論的方策)を
最大の行動価値関数Q_π(s, a) を与える行動a で更新することで改善する：
π(s)← argmax_a Q_π(s, a)

(8th) 手順(3rd)で取得した状態遷移情報について、
行動a が状態s での方策π(s)でなかったら
本ｴﾋﾟｿｰﾄﾞによる方策改善処理で、残りの状態遷移を遡って参照することは
しないので(※)、手順(1.3)を終了する。

(9th) 「着目先」状態遷移情報を、本ｴﾋﾟｿｰﾄﾞを一つ遡ったものへ移して、
手順(3rd)へ。

(1.4) 探索範囲を狭める為、ε-greedy法のﾊﾟﾗﾒｰﾀεを減少させ、その結果、
εが終了判断基準値未満になった場合、方策学習手順(1)を終了する。
εが終了判断基準値以上の場合、手順(1.2)へ遷移し別のｴﾋﾟｿｰﾄﾞ取得をおこなう。
ε←ε－⊿ε

(2) 学習結果の検証と利用
手順(1)で学習した方策πで、着目する初期状態から終端状態まで状態遷移させ、
最適化された状態遷移の1ｴﾋﾟｿｰﾄﾞを取得し、学習が成功したかどうかを確認する。

(※) 行動価値関数 Q_π(s, a)は、
「状態がs の時、特定の行動a を取り、状態s からの状態遷移後は方策πに従って行動を
選択する結果、期待できる収益」であったことを思い出してください。

(8.2.3) ﾓﾝﾃｶﾙﾛ法でのｻﾝﾌﾟﾘﾝｸﾞと方策改善の実装例

・ﾓﾝﾃｶﾙﾛ法で、「(例題) 崖歩き (Cliff Walking)」の方策πを作成する実装例とその実行結果を
以下に示します。

(実行結果)

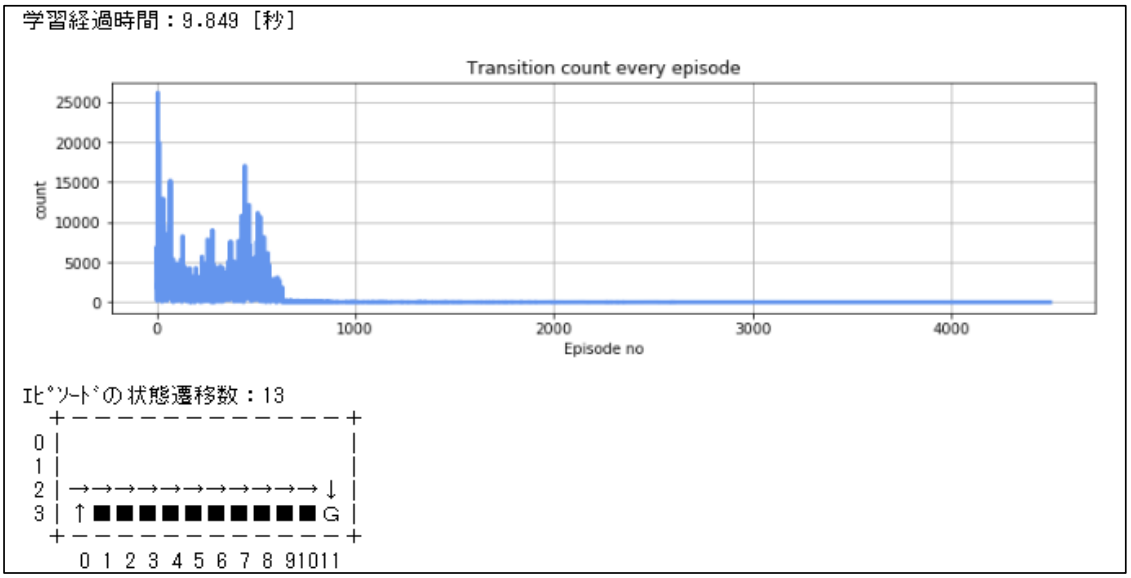
・100回ほど、学習⇒実行を繰り返してみたところ、開始点から終点までの状態遷移数は、
以下のような結果となりました。
実行時のεを0に近い小さい値にしているため、学習した方策πそのものではなく
任意に行動選択する余地を僅かに残していますが、６割程度は最適な経路を取っています。
参考の為、実行時のεを0で実行した結果も併記しますが、傾向は同じでした。

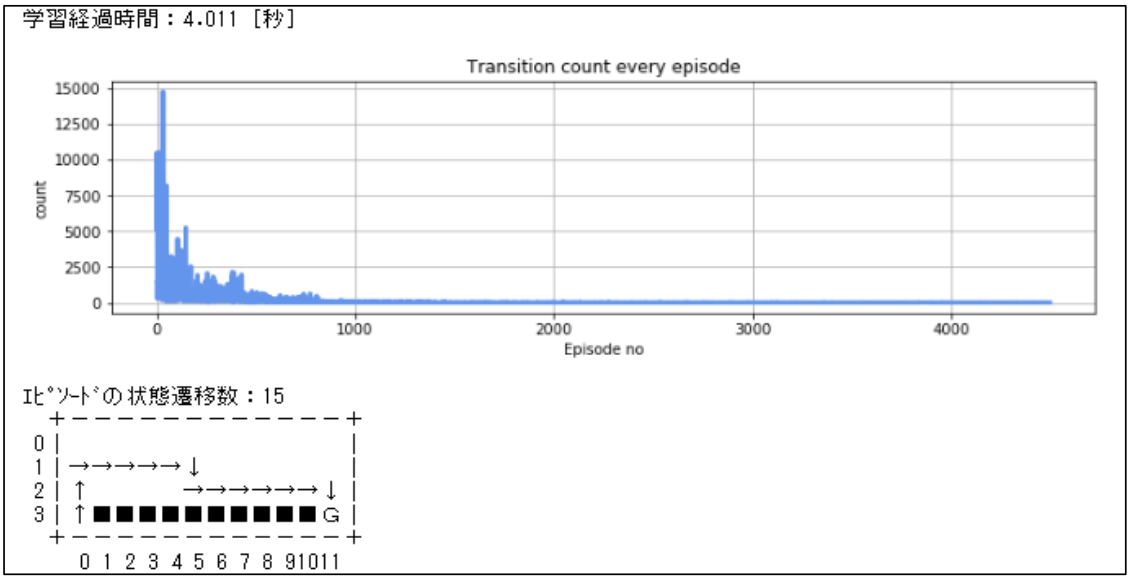
(ε=0.001)

1ｴﾋﾟｿｰﾄﾞの状態遷移数	頻度	備考
13	63	最短経路
15	29	2番目に短い経路
17	8	3番目に短い経路

(ε=0.000)

1ｴﾋﾟｿｰﾄﾞの状態遷移数	頻度	備考
13	58	最短経路
15	32	2番目に短い経路
17	10	3番目に短い経路





(ﾘｽﾄ11-(09)-1_崖歩き問題の解法その1：ﾓﾝﾃｶﾙﾛ法)

```
#********************************************************************************
#  ﾘｽﾄ11-(09)-1_崖歩き問題の解法その1：ﾓﾝﾃｶﾙﾛ法
#  ﾓﾝﾃｶﾙﾛ法で、
#  「(8.1) ﾓﾃﾞﾙﾌﾘｰな手法での例題」の「(例題) 崖歩き」の決定論的方策πを作成する。
#  ※ 事前に「ﾘｽﾄ11-(08)-1_崖歩き問題の環境と表示のｸﾗｽ」を実行しておくこと。
#********************************************************************************
#import numpy as np
import time
#import matplotlib.pyplot as plt


#********************************************************************************
# ｸﾗｽ     HyperParam
#--------------------------------------------------------------------------------
# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀのｸﾗｽ
#********************************************************************************
class HyperParam:

  #********************************************************************************
  # HyperParam ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #--------------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義し、指定値で初期化する：
  #
  # eps_First :  ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のｰﾊﾟﾗﾒｰﾀ (0≦ε≦1) εの初期値
  # eps_Last  :  学習時に、ε-greedy法のﾊﾟﾗﾒｰﾀεを徐々に減少させながら、
  #              ｴﾋﾟｿｰﾄﾞを繰り返して行動価値関数Qπ(s,a) の精度を上げてゆく。
  #              eps_Last は学習のためのｴﾋﾟｿｰﾄﾞ繰り返しの終了判断のεの最終値
  # eps_Rate  : εの1ｴﾋﾟｿｰﾄﾞ毎の減少率(0＜ε＜1、ε←◿εrate×ε)
  # eps_Run   : εの実行時の値
  #           (ε=0 で実行すると、方策πが完成していなくて終端に到達しない場合がある。
  #            εを 0 より大きいができるだけ小さい値で実行することにより、
  #            学習済の方策πを最大限に生かす方針の下で開始点から終了点まで移動する。)
  # trace     :True:追跡出力する、False:しない
  #
  #********************************************************************************
  def __init__(self, eps_First, eps_Last, eps_Rate, eps_Run, trace):
    self.eps_First = eps_First
    self.eps_Last = eps_Last
    self.eps_Rate = eps_Rate
    self.eps_Run = eps_Run
    self.trace = trace


#********************************************************************************
# ｸﾗｽ     Agent
#--------------------------------------------------------------------------------
# 崖歩きのｴｰｼﾞｪﾝﾄのｸﾗｽ
#********************************************************************************
class Agent:

  #********************************************************************************
  # Agent ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #--------------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義し初期化する：
  #
  #    env           :Environment のｲﾝｽﾀﾝｽ
  #--------------------------------------------------------------------------------
  #    actions       :取りうる行動a (＝移動ﾍﾞｸﾄﾙ(Δx,Δy)) の集合Ａ
  #    policy        :決定論的方策π(s) (＝現在の状態s でとる行動a)
  #    q             :状態s で行動a を取った時の行動価値関数Qπ(s,a)
  #    n             :状態s で行動a を取った訪問回数
```

```
#
#***************************************************************************
def __init__(self, env):

  # 取りうる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）の集合Ａをﾀﾌﾟﾙ型(Δx,Δy)の配列として保持する
  # (0,-1)：上, (-1,0)：左, (1,0)：右, (0,1)：下
  self.actions = [(0, -1), (-1, 0), (1, 0), (0, 1)]

  # 決定論的方策π を辞書型で保持し、乱数で初期化する
  #  キー：s、値：a
  #     s： 状態s（＝現在の位置座標(x,y)）
  #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
  self.policy = {}
  for s in env.states:
    self.policy[s] = self.actions[np.random.randint(len(self.actions))]

  # 状態s で行動a を取った時の行動価値関数Ｑπ(s,a) "q" を
  # ディクショナリ型で保持し、計算上あり得ない小さい値で初期化する
  #  キー：(s,a)、値：q
  #     s： 状態s（＝現在の位置座標(x,y)）
  #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
  #     q： 状態s で行動a を取った時の行動価値関数Ｑπ(s,a)
  self.q = {}
  for s in env.states:
    for a in self.actions:
      self.q[(s, a)] = -10**10

  # 状態s で行動a を取った訪問回数 "n" をディクショナリ型で保持し、0で初期化する
  #  キー：(s,a)、値：n
  #     s： 状態s（＝現在の位置座標(x,y)）
  #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
  #     n： 状態s で行動a を取った訪問回数Ｎ(s,a)
  self.n = {}
  for s in env.states:
    for a in self.actions:
      self.n[(s, a)] = 0

#***************************************************************************
# Agent ｸﾗｽ : getEpisode
# ｼﾐｭﾚｰｼｮﾝにより、1回分のｴﾋﾟｿｰﾄﾞを取得する
#---------------------------------------------------------------------------
# 引数：
#   env      : Environment のｲﾝｽﾀﾝｽ
#   epsilon  : ε-greedy法のﾊﾟﾗﾒｰﾀε
#---------------------------------------------------------------------------
# 戻り値：
# (第１返値) episode:
#             開始点から出口に至るまでの１回分のｴﾋﾟｿｰﾄﾞの
#             状態遷移の流れを、ﾀﾌﾟﾙ型オブジェクト (s,a,r)  の
#             ﾘｽﾄとして保存したもの
#
#          s：状態s（＝位置座標(x,y)）
#          a：状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#          r：状態s で行動a を取った時の状態s'への遷移に伴う報酬Ｒss'a
#***************************************************************************
def getEpisode(self, env, epsilon):
  # １回分のｴﾋﾟｿｰﾄﾞの状態遷移の流れをﾘｽﾄで持つ
  episode = []

  # 状態s の初期値＝開始点
  s = env.state_ini
```

```
  # 「終端状態」となるまで繰り返す
  while True:
    if np.random.random() < epsilon:
      # εの割合で、ランダムに行動a を選択する
      a = self.actions[np.random.randint(len(agent.actions))]
    else:
      # 1-εの割合で、決定論的方策πに従って行動a を選択する
      a = self.policy[s]

    # 現在の状態s で、上記で選択した行動a を取り、状態遷移を行う
    r, s_dash = env.transit(s, a)

    # ｴﾋﾟｿｰﾄﾞへ状態遷移情報を追加する
    episode.append((s, a, r))

    # 状態遷移で終点に着いたら「終端状態」となるので、本ｴﾋﾟｿｰﾄﾞは終了する
    x, y = s_dash
    if env.cliff[y][x] == 'G':
      break

    # 状態s を更新
    s = s_dash

  return episode

#*************************************************************************
# Agent ｸﾗｽ : updatePolicy
# 方策改善（最適のものを１つだけ採用する）
#-------------------------------------------------------------------------
# 指定した状態s における最適行動価値関数Q*(s,a) を求め(=q_max)、
# その時の行動a* (=a_best)を、状態sにおける最適方策π*(s) として
# 行動方策π(s) "policy[s]"を更新する。
#          s：状態s （＝位置座標(x,y)）
#          a：状態s でとる行動a （＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#-------------------------------------------------------------------------
# 引数：
#   s     : 現在の状態s （＝位置座標(x,y)）
#*************************************************************************
def updatePolicy(self, s):
  q_max = -10**10
  a_best = None
  for a in self.actions:
    if self.q[(s, a)] > q_max:
      q_max = self.q[(s, a)]
      a_best = a

  self.policy[s] = a_best

#*************************************************************************
# Agent ｸﾗｽ : train
# ｼﾐｭﾚｰｼｮﾝにより、
# ε-greedy法のﾊﾟﾗﾒｰﾀεを、ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの現象率で徐々に減少させ、
# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの指定値より小さくなったら学習を終了するまで、
# ｴﾋﾟｿｰﾄﾞを取得し、状態価値関数Vπ(s) を見積もる。
#-------------------------------------------------------------------------
# 引数：
#   env    : Environment のｲﾝｽﾀﾝｽ
#   param  : HyperParam のｲﾝｽﾀﾝｽ
#-------------------------------------------------------------------------
# 戻り値：
# (第１返値) episode_lengths:
#      1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数を要素とするﾘｽﾄ型で保持し、その履歴とする
```

```
#*****************************************************************************
def train(self, env, param):
  # 経過時間測定開始
  t1 = time.time()

  # 1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数の履歴をﾘｽﾄ型で保持する
  episode_lengths = []

  # ε-greedy法のﾊﾟﾗﾒｰﾀεの初期化
  epsilon = param.eps_First

  # ε-greedy法のﾊﾟﾗﾒｰﾀεを徐々に減少させながら、
  # ｴﾋﾟｿｰﾄﾞの取得とそれによる方策改善を繰り返す
  seqNo = 0
  while True:
    seqNo += 1

    # 1回分のｴﾋﾟｿｰﾄﾞを取得
    episode = self.getEpisode(env, epsilon)

    # 1回分のｴﾋﾟｿｰﾄﾞでの状態遷移数を追加
    episode_lengths.append(len(episode))

    # 各状態s での決定論的方策π(s)のｵﾝﾎﾟﾘｼｰの範囲を確認するために
    # 終端から開始点に向かって参照できるように逆順にする
    episode.reverse()

    # 終端から開始点に向かって状態遷移情報を参照し、
    # 各状態s での決定論的方策π(s)のｵﾝﾎﾟﾘｼｰの範囲内の状態遷移情報で、
    # 以下の一連の処理で方策改善を行う

    # total_r は最終値は総報酬であるが、各状態s での総報酬にもなる
    total_r = 0

    last = False
    for (s, a, r) in episode:

      # 行動a が状態s での方策π(s)でなかったら
      # 本ｴﾋﾟｿｰﾄﾞによる方策改善で、残りの状態遷移を遡って参照することはしない。
      # 最後に参照する状態遷移情報は、
      # ε-greedy法によりεの割合で既存の方策πでない方策π'での行動a'
      # を取ったことによる状態遷移情報
      if a != self.policy[s]:
        last = True

      # 各状態(s, a)での総報酬を更新
      total_r += r

      # 状態(s, a)への訪問回数を更新する
      self.n[(s, a)] += 1

      # 状態s の行動価値関数Qπ(s,a) を報酬Rss'a の総和の平均値として算出する
      self.q[(s, a)] += (total_r - self.q[(s, a)]) / self.n[(s, a)]

      # 状態s での方策を改善する
      self.updatePolicy(s)

      # 残りの状態遷移がｵﾝﾎﾟﾘｼｰの範囲内でなくなったら終了
      if last:
        break

    # ε-greedy法のﾊﾟﾗﾒｰﾀεを、ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの現象率で徐々に減少させ、
```

```
      # ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀの指定値より小さくなったら学習を終了する
      epsilon *= param.eps_Rate
      if epsilon < param.eps_Last:
        # 方策改善最後の1ｴﾋﾟｿｰﾄﾞ分の足跡を、表示する
        if(param.trace == True):
          print("¥ntrain: last episode seqNo={0}".format(seqNo))
          showFootprint(self, env, episode)
        break

      # 1ｴﾋﾟｿｰﾄﾞ分の足跡を、100回ごとに表示する
      if(param.trace == True) and (seqNo % 100 == 0):
        print("¥ntrain: episode seqNo={0}".format(seqNo))
        showFootprint(self, env, episode)

    # 経過時間を表示する
    t2 = time.time()
    print("学習経過時間：{0:.3f} [秒]".format(t2-t1))

    return episode_lengths

  #***************************************************************************
  # Agent ｸﾗｽ : run
  # 学習済の方策で1ｴﾋﾟｿｰﾄﾞ分(開始点から出口に至る経路探索)を取得する
  #---------------------------------------------------------------------------
  # 引数：
  #   env    : Environment のｲﾝｽﾀﾝｽ
  #   param  : HyperParam のｲﾝｽﾀﾝｽ
  #---------------------------------------------------------------------------
  # 戻り値：
  # (第１返値) episode: 1ｴﾋﾟｿｰﾄﾞ分の状態遷移
  #***************************************************************************
  def run(self, env, param):

    # getEpisode 関数を、実行時のεで実行
    episode = self.getEpisode(env, param.eps_Run)
    return episode


#*******************************************************************************
# 主制御
#*******************************************************************************

#-------------------------------------------------------------------------------
# (1) ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを設定
#-------------------------------------------------------------------------------
eps_First = 0.9
eps_Last = 0.01
eps_Rate = 0.999
eps_Run = 0.001
param = HyperParam(eps_First, eps_Last, eps_Rate, eps_Run, False)

#-------------------------------------------------------------------------------
# (2) 環境のｲﾝｽﾀﾝｽを作成
#-------------------------------------------------------------------------------
env = Environment()

#-------------------------------------------------------------------------------
# (3) 方策πの学習と利用を、100回繰り返す
#-------------------------------------------------------------------------------
showObj = ShowFunc()
for rii in range(0,100):
```

```
# Agent ｸﾗｽのｲﾝｽﾀﾝｽを作成する
agent = Agent(env)

# ﾓﾝﾃｶﾙﾛ法で方策πを学習する
param.trace = False
episode_lengths = agent.train(env, param)

# 方策πの学習で、1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数の履歴をｸﾞﾗﾌ表示する
showObj.showLearningCurve(episode_lengths)

# 方策πの学習終了時の、行動価値関数Qπ(s,a)を表示する
showObj.showQ(agent, env)

# 学習済の方策で1ｴﾋﾟｿｰﾄﾞ分(開始点から出口に至る経路)を取得する
param.trace = True
episode = agent.run(env, param)

# 1ｴﾋﾟｿｰﾄﾞ分の足跡を表示する
showObj.showFootprint(agent, env, episode)
# 1ｴﾋﾟｿｰﾄﾞ分の足跡を表示する
showObj.showFootprint(agent, env, episode)
```

【出典・参考】

ﾓﾝﾃｶﾙﾛ法⇒ https://qiita.com/sconeman/items/cb6047540a39b3bf15d7

ﾓﾝﾃｶﾙﾛ法⇒
https://ja.wikipedia.org/wiki/%E3%83%A2%E3%83%B3%E3%83%86%E3%82%AB%E3%83%AB%E3%83%AD%E6%B3%95

ﾓﾝﾃｶﾙﾛ法⇒「ITエンジニアのための強化学習理論入門」(2020年07月 技術評論社 中井悦司)

ﾓﾝﾃｶﾙﾛ法⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

(8.3) TD学習

・「TD学習」は、「ﾓﾝﾃｶﾙﾛ法」と動的計画法の考え方を組み合わせた、ﾓﾃﾞﾙﾌﾘｰな手法の一つです。

(8.3.1) TD(0)法

・逐一訪問ﾓﾝﾃｶﾙﾛ法での、状態価値関数の推定値についての漸化式(式8-3)

逐一訪問ﾓﾝﾃｶﾙﾛ法での状態価値関数$V_\pi(s)$の推定値（再掲）

$$V_{t+1}(s) = V_t(s) + \alpha \{G_t - V_t(S_t)\} \times I(S_t=s) \quad \cdots(式8\text{-}3)再掲$$

$I(S_k=s)$ ：時刻k(k=1～t)での状態S_kがs の時1、それ以外の時0 を返す関数
$V_t(s)$ ：時刻tで、状態がs の時の状態価値関数の推定値
G_t ：時刻t から将来にわたる割引報酬和
S_t ：時刻t での状態値
α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

において、時刻t から将来にわたる割引報酬和G_t を
一時刻先の推定値「$R_{t+1} + \gamma V_t(S_{t+1})$」で置き換えると、以下の漸化式(式8-4)が得られます。

TD学習での状態価値関数$V_\pi(s)$の推定値

$$V_{t+1}(s) = V_t(s) + \alpha \times \delta_{t+1} \times I(S_t=s) \quad \cdots(式8\text{-}4)$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \quad \cdots(式8\text{-}5)$$

δ_{t+1} ：TD誤差　（時刻t での状態価値関数の推定値$V_t(S_t)$と
時刻t+1 での状態価値関数の推定値との差分）
$I(S_k=s)$ ：時刻k(k=1～t)での状態S_kがs の時1、それ以外の時0 を返す関数
$V_t(s)$ ：時刻t における状態s での状態価値関数の推定値
R_{t+1} ：時刻t における状態S_t から時刻t+1 での状態S_{t+1} への遷移時の報酬
S_t ：時刻t での状態値
α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

このうち、δ_{t+1} は、
時刻t での状態価値関数の推定値$V_t(S_t)$と、
時刻t+1 での状態価値関数の推定値「$R_{t+1} + \gamma V_t(S_{t+1})$」との差分であり、
「TD誤差 (ﾃｨｰﾃﾞｨｰｺﾞｻ、Temporal Difference error)」と言います。
このTD誤差を小さくするように状態価値関数$V_t(s)$ を推定する学習法を、
「TD学習 (ﾃｨｰﾃﾞｨｰｶﾞｸｼｭｳ、Temporal Difference Learning、時間的差分学習)」と言います。

・上記の手法では、一時刻先の状態価値関数の推定値を用いており、
「TD誤差」は「1-ｽﾃｯﾌﾟTD誤差」、「TD学習」の手法を「TD(0)法」と言います。

・「ﾓﾝﾃｶﾙﾛ法」では各ｴﾋﾟｿｰﾄﾞの終端状態に至るまでｻﾝﾌﾟﾘﾝｸﾞして状態価値関数を計算していたのに対し、
「TD(0)法」では一時刻先迄の状態価値関数の推定値を用いる為、計算ｺｽﾄが低くなります。

・より具体的なTD学習での価値関数の推定とそれに基づく方策学習については、
「(8.4) SARSA」、「(8.5) Q学習」を参照のこと。

(8.3.2) TD(λ)法

・「TD(0)法」で推定に用いる未来時刻の範囲を拡張したのが「TD(λ)法」です。

・状態価値関数$V_\pi(s)$ の推定値の計算(式8-3)で、時刻t から将来にわたる割引報酬和G_t について、
「TD(0)法」では、一時刻先での状態価値関数の推定値を用いて計算するのに対し、
「TD(λ)法」では、ｎ時刻先までの状態価値関数の推定値を用いて計算します。

・時刻tにおける「n-ｽﾃｯﾌﾟ収益」$G_t^{(n)}$を以下のように定義します：

時刻tにおける「n-ｽﾃｯﾌﾟ収益」$G_t^{(n)}$

$$G_t^{(n)} \equiv \underbrace{R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n}} + \gamma^n V_t(S_{t+n}) \quad \cdots(式8\text{-}6)$$

$$= \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n V_t(S_{t+n}) \quad \cdots(式8\text{-}7)$$

$G_t^{(n)}$ ：時刻t における「n-ｽﾃｯﾌﾟ収益」
R_{t+k} ：時刻t+k における即時報酬 (k=1～n)
γ ：割引率 ($0 \leqq \gamma \leqq 1$)

・上記の「n-ｽﾃｯﾌﾟ収益」$G_t^{(n)}$をﾊﾟﾗﾒｰﾀλのべき乗で加重平均を取った「λ-収益」G_t^{λ}を
以下のように定義します(※1)：

時刻tにおける「λ-収益」G_t^{λ}

$$G_t^{\lambda} \equiv (1-\lambda) \sum_{n=1}^{T-t} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t} G_t \quad \cdots(式8\text{-}8)$$

G_t^{λ} ：時刻t における「λ-収益」
$G_t^{(n)}$ ：時刻t における「n-ｽﾃｯﾌﾟ収益」
G_t ：時刻t から将来にわたる割引報酬和
γ ：割引率 ($0 \leqq \gamma \leqq 1$)
λ ：ﾊﾟﾗﾒｰﾀ

(※1) (式8-8) 右辺各項にかかる係数の和は１となり、「加重平均」となっています：

$$\text{右辺各項にかかる係数の和} = (1-\lambda) \Sigma_{n=1}^{T-t} \lambda^{n-1} + \lambda^{T-t}$$
$$= \{ \Sigma_{n=1}^{T-t} \lambda^{n-1} + \lambda^{T-t} \} - \lambda \Sigma_{n=1}^{T-t} \lambda^{n-1}$$
$$= \lambda^0 + \Sigma_{n=1}^{T-t} \lambda^n - \Sigma_{n=1}^{T-t} \lambda^n = 1$$

・時刻t での状態価値関数の推定値$V_t(S_t)$と
時刻t での「λ-収益」G_t^{λ} との差分δ_{t+1} をTD誤差とし、
このTD誤差を小さくするように状態価値関数$V_t(s)$ を推定する学習法を、「TD(λ)法」と言います。

TD学習 (TD(λ)法)

$$V_{t+1}(s) = V_t(s) + \alpha \times \delta_{t+1}^{\lambda} \times I(S_t=s) \quad \cdots(式8\text{-}9)$$

$$\delta_{t+1}^{\lambda} \equiv G_t^{\lambda} - V_t(S_t) \quad \cdots(式8\text{-}10)$$

δ_{t+1}^{λ} ：TD誤差　(時刻t での状態価値関数の推定値$V_t(S_t)$と
時刻t での「λ-収益」G_t^{λ}との差分)
$I(S_k=s)$ ：時刻k(k=1～t)での状態S_k がs の時1、それ以外の時0 を返す関数
G_t^{λ} ：時刻t における「λ-収益」
$V_t(s)$ ：時刻t で、状態s の状態価値関数の推定値
S_t ：時刻t での状態値
α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

・「TD(λ)法」は、以下のように他の手法を包含しています：

λ＝１ ：ﾓﾝﾃｶﾙﾛ法と同じです(ﾓﾝﾃｶﾙﾛ法＝TD(1)法)。

…(式8-8)でλ=1と置くと $G_t^{\lambda=1}=G_t$ となり、(式8-9)は(式8-3)と同じになります。

・次に、「適格度ﾄﾚｰｽ (ﾃｷｶｸﾄﾞﾄﾚｰｽ、eligibility trace)」 $E_{k+1}(s)$ を以下のように定義します：

「適格度ﾄﾚｰｽ」 $E_{k+1}(s)$

$$E_{k+1}(s) \equiv \sum_{t=0}^{k}(\lambda\gamma)^{k-t}\times I(S_t=s) \quad \cdots(式8\text{-}11)$$
$$= I(S_k=s) + (\lambda\gamma)I(S_{k-1}=s) + \cdots + (\lambda\gamma)^k\times I(S_0=s)$$

$E_{k+1}(s)$ ：適格度ﾄﾚｰｽ

$I(S_k=s)$ ：時刻k(k=1～t)での状態S_k がs の時1、それ以外の時0 を返す関数

S_t ：時刻t での状態値

γ ：割引率 (0≦γ≦1)

λ ：ﾊﾟﾗﾒｰﾀ

・TD(λ)法の更新式は「1-ｽﾃｯﾌﾟTD誤差」と「適格度ﾄﾚｰｽ」を用いて、以下のようになります：

TD(λ)法の更新式

$$V_{t+1}(s) = V_t(s) + \alpha\times\delta_{t+1}\times E_{t+1}(s) \quad \cdots(式8\text{-}12)$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \quad \cdots(式8\text{-}5)再掲$$

：1-ｽﾃｯﾌﾟTD誤差

(時刻t での状態価値関数の推定値$V_t(S_t)$と

時刻t+1 での状態価値関数の推定値との差分)

$$E_{t+1}(s) \equiv \lambda\gamma E_t(s) + I(S_t=s)$$

：適格度ﾄﾚｰｽ

$I(S_k=s)$ ：時刻k(k=1～t)での状態S_k がs の時1、それ以外の時0 を返す関数

$V_{t+1}(s)$ ：時刻t+1で、状態価値関数の推定値

R_{t+1} ：時刻t における状態S_t から時刻t+1 での状態S_{t+1} への遷移時の報酬

S_t ：時刻t での状態値

α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

(8.3.3) TD学習と方策ｵﾝ型・方策ｵﾌ型

・TD学習におけるTD誤差δ_{t+1}の計算について、

TD学習での状態価値関数$V_{\pi}(s)$ の推定値

$$V_{t+1}(s) = V_t(s) + \alpha \times \delta_{t+1} \times I(S_t=s) \quad \cdots(式8\text{-}4)再掲$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma V_t(S'_{t+1}) - V_t(S_t) \quad \cdots(式8\text{-}5')$$

δ_{t+1} ：TD誤差　（時刻t での状態価値関数の推定値$V_t(S_t)$と
時刻t+1 での状態価値関数の推定値との差分）

$I(S_k=s)$ ：時刻k(k=1～t)での状態S_k がs の時1、それ以外の時0 を返す関数

$V_t(s)$ ：時刻t における状態s での状態価値関数の推定値

R_{t+1} ：時刻t における状態S_t から時刻t+1 での状態S_{t+1} への遷移時の報酬

S_t ：時刻t での状態値

S'_{t+1} ：時刻t+1 での状態値

α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

で(式8-5')にある時刻t+1 での状態値S'_{t+1} として、

(1) 方策ｵﾝ型学習では、
状態S_tで評価対象の方策$\pi(S_t)$に従い行動A_tを取った結果
遷移する状態S_{t+1}を採用します。
方策ｵﾝ型学習の例として「SARSA」があります。

(2) 方策ｵﾌ型学習では、
状態S_tで評価対象の方策$\pi(S_t)$に従う行動A_tとは無関係の行動A_t'をとった結果
遷移する状態を採用します。
方策ｵﾌ型学習の例として「Q学習」があります。

(8.4) SARSA

(8.4.1) SARSAとは

・TD学習（TD(0)法）の(式8-4)、(式8-5)は、状態価値関数$V_t(s)$についての更新式になっています。

TD学習での状態価値関数$V_\pi(s)$の推定値（再掲）

$$V_{t+1}(s) = V_t(s) + \alpha \times \delta_{t+1} \times I(S_t=s) \quad \cdots(式8\text{-}4)再掲$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \quad \cdots(式8\text{-}5)再掲$$

これを行動価値関数$Q_t(s,a)$についての式に焼き直すと、以下の更新式を得ます。

SARSA

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \times \delta_{t+1} \times I(S_t=s、A_t=a) \quad \cdots(式8\text{-}13)$$

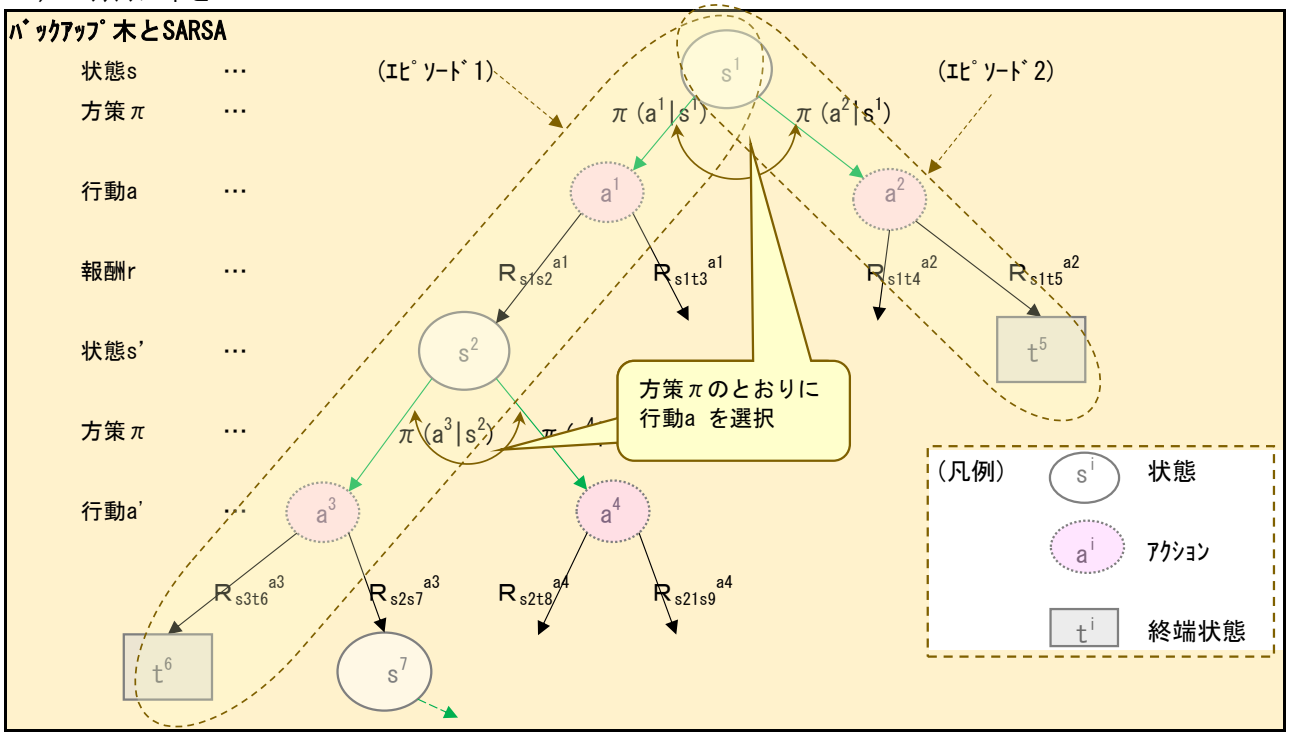
$$\delta_{t+1} \equiv R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) \quad \cdots(式8\text{-}14)$$

：1-ｽﾃｯﾌﾟTD誤差
（時刻t での行動価値関数の推定値$Q_t(S_t, A_t)$と
時刻t+1 での行動価値関数の推定値との差分）

$I(\chi)$ ：条件χが成立する時1、それ以外の時0 を返す関数
$Q_t(s,a)$ ：時刻tで、状態s で行動a をとる時の行動価値関数の推定値
R_{t+1} ：時刻t における状態S_t から時刻t+1 での状態S_{t+1} への遷移時の報酬
S_t ：時刻t での状態値
A_t ：時刻t でとる行動
α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

・この更新式は、対応するﾊﾞｯｸｱｯﾌﾟ線図に沿って状態・行動・報酬の系列が(s, a, r, s', a') の順に並ぶので「SARSA（ｻﾙｻ）」と言います(図9.1)。

(図9.1) ﾊﾞｯｸｱｯﾌﾟ木とSARSA



・SARSAは、「方策ｵﾝ型」の「価値ﾍﾞｰｽ」の学習手法であり、
状態S_tにおいて、評価対象の方策$\pi(S_t)$が示す行動A_tを取る結果遷移する状態をS_{t+1}とし、
遷移先の状態S_{t+1}において、評価対象の方策$\pi(S_{t+1})$が指示する行動をA_{t+1}とした時、
TD誤差の計算式(式8-14)におけるA_{t+1}として、
方策πが指示している行動A_{t+1}をそのまま採用して学習します。

・次節で紹介するｱﾙｺﾞﾘｽﾞﾑでもわかるように、
ﾓﾝﾃｶﾙﾛ法では、1ｴﾋﾟｿｰﾄﾞ上で終端状態に達してから価値関数を評価するのに対し、
SARSAでは、1ｴﾋﾟｿｰﾄﾞ上で状態遷移を行う毎に価値関数を評価する為、
SARSAはﾓﾝﾃｶﾙﾛ法より計算ｺｽﾄが低く抑えられています。

(8.4.2) SARSAでのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)

・SARSAでのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)を以下に記します：

SARSAでのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)

(1) 方策を学習する

・繰り返しｴﾋﾟｿｰﾄﾞを取得しながら
状態遷移情報のｻﾝﾌﾟﾘﾝｸﾞを行い、それに基づく方策改善を繰り返す。

(1.1) 初期設定

(1.1.1) あり得る状態値s の集合Ｓを作成する
(1.1.2) 取りうる行動a の集合Ａを作成する
(1.1.3) 方策π を初期化する (乱数で$\pi(s)$に対応する行動aを決定、s∈Ｓ, a∈Ａ)
(1.1.4) 状態s で行動a を取った時の行動価値関数$Q_\pi(s,a)$ を初期化する
(1.1.5) ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のﾊﾟﾗﾒｰﾀε を設定する (0≦ε≦1)
(1.1.6) 割引率γを設定する (0≦γ≦1)
(1.1.7) 重み係数αを設定する (0≦α≦1)

(1.2) 1回分のｴﾋﾟｿｰﾄﾞを取得しながら、
ｴﾋﾟｿｰﾄﾞ内の遷移情報(状態s、行動a、遷移先状態s'、遷移時報酬$R_{ss'}^{a}$の系列) をみて
行動価値関数$Q_\pi(s,a)$を推定し、方策改善を繰り返す。

(1st) 初期設定
・状態s を初期状態で設定
・初期状態s でとる行動a を方策πに従い選択する　　: $a \leftarrow \pi(s)$

(2nd) 状態s で行動a を取る場合の状態遷移に伴う次の量を計算する。
・遷移による次状態 s'、遷移時報酬 $R_{ss'}^{a}$

(3rd) 遷移先の状態s' でとる行動a' を方策πに従い選択する　　: $a' \leftarrow \pi(s')$

(4th) 行動価値関数$Q_\pi(s,a)$をTD(0)法により推定する。

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha \times \delta_\pi(s,s',a)$$

$$\delta_\pi(s,s',a) \equiv R_{ss'}^{a} + \gamma \times Q_\pi(s',a') - Q_\pi(s,a)$$

(5th) 状態s での方策$\pi(s)$を最大の行動価値関数$Q_\pi(s,a)$ を与える行動a で
更新することにより改善する。

$$\pi(s) \leftarrow \mathrm{argmax}_a\ Q_\pi(s,a)$$

(6th) 遷移先状態s' が終端状態の場合、手順(1.2)を終了する。
終端状態でない場合、状態s を遷移先状態s' で、
行動a を手順(4th)で選択した行動a' で更新し、手順(2nd)へ
s←s'、a←a'

(1.3) 別のｴﾋﾟｿｰﾄﾞで学習する為、手順(1.2)へ。
ｴﾋﾟｿｰﾄﾞ取得回数が目標値に達したら、方策学習手順(1)を終了する。

(2) 学習結果の検証と利用
手順(1)で学習した方策πで、着目する初期状態から終端状態まで状態遷移させ、
最適化された状態遷移の1ｴﾋﾟｿｰﾄﾞを取得し、学習が成功したかどうかを確認する。

(8.4.3) SARSAでのｻﾝﾌﾟﾘﾝｸﾞと方策改善の実装例

・SARSAで、「(例題) 崖歩き (Cliff Walking)」の方策πを作成する実装例とその実行結果を
以下に示します。

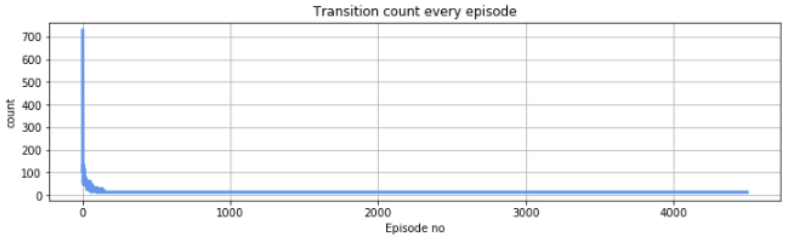
(実行結果)

・ﾓﾝﾃｶﾙﾛ法より学習時間は一桁少ないです。

・100回ほど、学習⇒実行を繰り返してみたところ、開始点から終点までの状態遷移数は、
以下のような結果となりました。
全て最短経路を取っています。

1ｴﾋﾟｿｰﾄﾞの状態遷移数	頻度	備考
13	100	最短経路

```
学習経過時間：0.189 [秒]
```




```
Iﾋﾟｿｰﾄﾞの状態遷移数：13
  +－－－－－－－－－－－－+
 0|                        |
 1|                        |
 2|→→→→→→→→→→→→↓|
 3|↑■■■■■■■■■■■G|
  +－－－－－－－－－－－－+
    0 1 2 3 4 5 6 7 8 91011
```

```
行動価値関数Qπ(s,a)
Y=  0:
(  0,  0):  ↑:  -9.92  ←: -10.02  →:  -9.91  ↓:  -9.88
(  1,  0):  ↑:  -9.61  ←:  -9.52  →:  -9.42  ↓:  -9.42
(  2,  0):  ↑:  -8.81  ←:  -8.91  →:  -8.78  ↓:  -8.75
(  3,  0):  ↑:  -8.11  ←:  -8.23  →:  -8.10  ↓:  -8.14
(  4,  0):  ↑:  -7.39  ←:  -7.61  →:  -7.36  ↓:  -7.35
(  5,  0):  ↑:  -6.65  ←:  -6.85  →:  -6.57  ↓:  -6.57
(  6,  0):  ↑:  -5.89  ←:  -5.76  →:  -5.79  ↓:  -5.89
(  7,  0):  ↑:  -5.18  ←:  -5.04  →:  -5.02  ↓:  -5.08
(  8,  0):  ↑:  -4.40  ←:  -4.28  →:  -4.25  ↓:  -4.34
(  9,  0):  ↑:  -3.66  ←:  -3.79  →:  -3.44  ↓:  -3.44
( 10,  0):  ↑:  -2.94  ←:  -2.94  →:  -2.62  ↓:  -2.68
( 11,  0):  ↑:  -2.17  ←:  -1.99  →:  -2.19  ↓:  -1.89
Y=  1:
(  0,  1):  ↑: -10.28  ←: -10.32  →: -10.26  ↓: -10.59
(  1,  1):  ↑:  -9.63  ←:  -9.68  →:  -9.60  ↓:  -9.74
(  2,  1):  ↑:  -8.91  ←:  -8.96  →:  -8.82  ↓:  -9.01
(  3,  1):  ↑:  -8.12  ←:  -8.13  →:  -8.06  ↓:  -8.05
(  4,  1):  ↑:  -7.33  ←:  -7.43  →:  -7.25  ↓:  -7.35
(  5,  1):  ↑:  -6.50  ←:  -6.66  →:  -6.41  ↓:  -6.42
(  6,  1):  ↑:  -5.78  ←:  -5.59  →:  -5.54  ↓:  -5.63
(  7,  1):  ↑:  -4.93  ←:  -4.80  →:  -4.68  ↓:  -4.68
(  8,  1):  ↑:  -4.04  ←:  -4.01  →:  -3.80  ↓:  -3.83
(  9,  1):  ↑:  -3.07  ←:  -3.25  →:  -2.89  ↓:  -2.89
( 10,  1):  ↑:  -1.98  ←:  -2.10  →:  -1.96  ↓:  -1.96
( 11,  1):  ↑:  -1.21  ←:  -1.20  →:  -1.08  ↓:  -1.00
Y=  2:
(  0,  2):  ↑: -11.00  ←: -11.33  →: -11.00  ↓: -11.22
(  1,  2):  ↑: -10.04  ←: -10.02  →: -10.00  ↓: -20.00
(  2,  2):  ↑:  -9.00  ←:  -9.29  →:  -9.00  ↓: -20.14
(  3,  2):  ↑:  -8.03  ←:  -8.22  →:  -8.00  ↓: -20.22
(  4,  2):  ↑:  -7.23  ←:  -7.34  →:  -7.00  ↓: -20.15
(  5,  2):  ↑:  -6.19  ←:  -6.19  →:  -6.00  ↓: -20.29
(  6,  2):  ↑:  -5.16  ←:  -5.19  →:  -5.00  ↓: -20.27
(  7,  2):  ↑:  -4.08  ←:  -4.23  →:  -4.00  ↓: -20.34
(  8,  2):  ↑:  -3.11  ←:  -3.32  →:  -3.00  ↓: -20.43
(  9,  2):  ↑:  -2.24  ←:  -2.21  →:  -2.00  ↓: -20.37
( 10,  2):  ↑:  -1.20  ←:  -1.07  →:  -1.00  ↓: -20.44
( 11,  2):  ↑:  -0.20  ←:  -0.20  →:  -0.36  ↓:  +0.00
Y=  3:
(  0,  3):  ↑: -12.00  ←: -12.10  →: -36.00  ↓: -12.19
(  1,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  2,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  3,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  4,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  5,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  6,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  7,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  8,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
(  9,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
( 10,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
( 11,  3):  ↑:  +0.00  ←:  +0.00  →:  +0.00  ↓:  +0.00
```

(ﾘｽﾄ11-(09)-2_崖歩き問題の解法その2：SARSA)

```
#********************************************************************************
# ﾘｽﾄ11-(09)-2_崖歩き問題の解法その2：SARSA
#  SARSAで、
#  「(8.1) ﾓﾃﾞﾙﾌﾘｰな手法での例題」の「(例題) 崖歩き」の決定論的方策πを作成する。
#  ※ 事前に「ﾘｽﾄ11-(08)-1_崖歩き問題の環境と表示のｸﾗｽ」を実行しておくこと。
#********************************************************************************
#import numpy as np
import time
#import matplotlib.pyplot as plt


#********************************************************************************
# ｸﾗｽ     HyperParam
#--------------------------------------------------------------------------------
# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀのｸﾗｽ
#--------------------------------------------------------------------------------
# (※) 実装にﾓﾝﾃｶﾙﾛ法と相違あり：ﾊﾟﾗﾒｰﾀの構成に違いあり
#********************************************************************************
class HyperParam:

  #********************************************************************************
  # HyperParam ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #--------------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義し、指定値で初期化する：
  #
  # alpha       :ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀα：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)
  # gamma       :ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀγ：割引率
  # eps_Train   :ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀε：学習時のε-greedy法のﾊﾟﾗﾒｰﾀε
  # eps_Run     :εの実行時の値
  #              (ε=0 で実行すると、方策πが完成していなくて終端に到達しない場合がある。
  #               εを 0 より大きいができるだけ小さい値で実行することにより、
  #               学習済の方策πを最大限に生かす方針の下で開始点から終了点まで移動する。)
  # episodeNum  :方策改善時に学習の為に取得するｴﾋﾟｿｰﾄﾞの最大回数
  # trace       :True:追跡出力する、False:しない
  #
  #********************************************************************************
  def __init__(self, alpha, gamma, eps_Train, eps_Run, episodeNum, trace):
    # ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀα、γ、ε
    self.alpha = alpha
    self.gamma = gamma
    self.eps_Train = eps_Train
    self.eps_Run = eps_Run
    self.episodeNum = episodeNum
    self.trace = trace
```

```
#*****************************************************************************
# ｸﾗｽ     Agent
#-----------------------------------------------------------------------------
# 崖歩きのｴｰｼﾞｪﾝﾄのｸﾗｽ
#*****************************************************************************
class Agent:

  #*****************************************************************************
  # Agent ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #-----------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義し初期化する：
  #
  #    env            : Environment のｲﾝｽﾀﾝｽ
  #-----------------------------------------------------------------------------
  #    actions        : 取りうる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）の集合Ａ
  #    policy         : 決定論的方策π(s)（＝現在の状態s でとる行動a）
  #    q              : 状態s で行動a を取った時の行動価値関数Ｑπ(s,a)
  #
  #-----------------------------------------------------------------------------
  # (※) 実装にﾓﾝﾃｶﾙﾛ法と相違あり：
  #         ・訪問回数 "n" を持たない
  #*****************************************************************************
  def __init__(self, env):

    # 取りうる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）の集合Ａをﾀﾌﾟﾙ型(Δx,Δy)の配列として保持する
    # (0,-1)：上, (-1,0)：左, (1,0)：右, (0,1)：下
    self.actions = [(0, -1), (-1, 0), (1, 0), (0, 1)]

    # 決定論的方策π を辞書型で保持し、乱数で初期化する
    #  キー：s、値：a
    #     s： 状態s（＝現在の位置座標(x,y)）
    #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
    self.policy = {}
    for s in env.states:
      self.policy[s] = self.actions[np.random.randint(len(self.actions))]

    # 状態s で行動a を取った時の行動価値関数Ｑπ(s,a) "q" を
    # ディクショナリ型で保持し、0で初期化する
    #  キー：(s,a)、値：q
    #     s： 状態s（＝現在の位置座標(x,y)）
    #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
    #     q： 状態s で行動a を取った時の行動価値関数Ｑπ(s,a)
    self.q = {}
    for s in env.states:
      for a in self.actions:
        self.q[(s, a)] = 0

  #*****************************************************************************
  # Agent ｸﾗｽ : getEpisodeAndLearn
  # ｼﾐｭﾚｰｼｮﾝにより、1回分のｴﾋﾟｿｰﾄﾞを取得しながら、方策改善を行う
  #-----------------------------------------------------------------------------
  # 引数：
  #   env      : Environment のｲﾝｽﾀﾝｽ
  #   param    : HyperParam のｲﾝｽﾀﾝｽ
  #   epsilon  : ε-greedy法のﾊﾟﾗﾒｰﾀε （不使用）
  #   inTrain  : 方策改善の為の学習中の場合:True、学習中でない場合:False
  #-----------------------------------------------------------------------------
  # 戻り値：
  # (第１返値) episode:
  #              開始点から出口に至るまでの１回分のｴﾋﾟｿｰﾄﾞの
  #              状態遷移の流れを、ﾀﾌﾟﾙ型ｵﾌﾞｼﾞｪｸﾄ (s,a,r)  の
  #              ﾘｽﾄとして保存したもの
```

```
#
#          s：状態s （＝位置座標(x,y)）
#          a：状態s でとる行動a （＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#          r：状態s で行動a を取った時の状態s'への遷移に伴う報酬Rss'a
#**************************************************************************
def getEpisodeAndLearn(self, env, param, epsilon, inTrain):
  # １回分のｴﾋﾟｿｰﾄﾞの状態遷移の流れをﾘｽﾄで持つ
  episode = []

  # 状態s の初期値＝開始点
  s = env.state_ini

  # 初期状態での行動a を方策πに従って選択する
  a = self.policy[s]

  # 「終端状態」となるまで繰り返す
  while True:

    # 現在の状態s で、選択した行動a を取り、状態遷移を行う
    r, s_dash = env.transit(s, a)

    # ｴﾋﾟｿｰﾄﾞへ状態遷移情報を追加する
    episode.append((s, a, r))

    # 遷移先の状態s' (s_dash) でとる行動a' (a_dash) を決定する
    a_dash = self.policy[s_dash]

    # 学習中の場合、
    # 行動価値関数Qπ(s,a)をＴＤ(0)法により推定して、
    # 方策πの改善を行う
    if inTrain:
      self.q[(s, a)] += param.alpha * (r + (param.gamma * self.q[(s_dash, a_dash)]) - self.q[(s, a)])
      self.updatePolicy(s)

    # 状態遷移で終点に着いたら「終端状態」となるので、本ｴﾋﾟｿｰﾄﾞは終了する
    x, y = s_dash
    if env.cliff[y][x] == 'G':
      break

    # 状態s と取る行動a を更新
    a = a_dash
    s = s_dash

  return episode

#**************************************************************************
# Agent ｸﾗｽ : updatePolicy
# 方策改善 (最適のものを１つだけ採用する)
#--------------------------------------------------------------------------
# 指定した状態s における最適行動価値関数Q*(s,a) を求め(=q_max)、
# その時の行動a* (=a_best)を、状態sにおける最適方策π*(s) として
# 行動方策π(s) "policy[s]"を更新する。
#          s：状態s （＝位置座標(x,y)）
#          a：状態s でとる行動a （＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#--------------------------------------------------------------------------
# 引数：
#   s     ： 現在の状態s （＝位置座標(x,y)）
#--------------------------------------------------------------------------
# (※) 実装はﾓﾝﾃｶﾙﾛ法とおなじ
#**************************************************************************
```

```
def updatePolicy(self, s):
  q_max = -10**10
  a_best = None
  for a in self.actions:
    if self.q[(s, a)] > q_max:
      q_max = self.q[(s, a)]
      a_best = a

  self.policy[s] = a_best

#***************************************************************************
# Agent ｸﾗｽ : train
# ｼﾐｭﾚｰｼｮﾝにより、
# ｴﾋﾟｿｰﾄﾞ取得回数が指定回数 "episodeNum" に達するまで、
# ｴﾋﾟｿｰﾄﾞを取得し、状態価値関数Vπ(s) を見積もる。
#---------------------------------------------------------------------------
# 引数：
#   env    : Environment のｲﾝｽﾀﾝｽ
#   param  : HyperParam のｲﾝｽﾀﾝｽ
#---------------------------------------------------------------------------
# 戻り値：
# (第１返値) episode_lengths:
#      1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数を要素とするﾘｽﾄ型で保持し、その履歴とする
#---------------------------------------------------------------------------
# (※) 実装はﾓﾝﾃｶﾙﾛ法と全く異なる
#***************************************************************************
def train(self, env, param):
  # 経過時間測定開始
  t1 = time.time()

  # 指定回数分、ｴﾋﾟｿｰﾄﾞ取得と学習を繰り返す
  episode_lengths = []
  seqNo = 0
  for _ in range(param.episodeNum):
    seqNo += 1
    episode = self.getEpisodeAndLearn(env, param, param.eps_Train, True)
    episode_lengths.append(len(episode))

    # 1ｴﾋﾟｿｰﾄﾞ分の足跡を表示する
    if(param.trace == True):
      print("train: episode seqNo={0}".format(seqNo))
      showFootprint(self, env, episode)

  # 経過時間を表示する
  t2 = time.time()
  print("学習経過時間：{0:.3f} [秒]".format(t2-t1))

  return episode_lengths

#***************************************************************************
# Agent ｸﾗｽ : run
# 学習済の方策で1ｴﾋﾟｿｰﾄﾞ分(開始点から出口に至る経路探索)を取得する
#---------------------------------------------------------------------------
# 引数：
#   env    : Environment のｲﾝｽﾀﾝｽ
#   param  : HyperParam のｲﾝｽﾀﾝｽ
#---------------------------------------------------------------------------
# 戻り値：
# (第１返値) episode: 1ｴﾋﾟｿｰﾄﾞ分の状態遷移
#---------------------------------------------------------------------------
# (※) 実装にﾓﾝﾃｶﾙﾛ法と相違あり
#***************************************************************************
```

```
  def run(self, env, param):
    # getEpisodeAndLearn 関数を、ε=実行時の値、inTrain=False で実行
    episode = self.getEpisodeAndLearn(env, param, param.eps_Run, False)
    return episode


#*****************************************************************************
# 主制御
#*****************************************************************************

#-----------------------------------------------------------------------------
# (1) ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを設定
#-----------------------------------------------------------------------------
alpha = 0.2
gamma = 1
eps_Train = 0.1
eps_Run = 0.001
episodeNum = 4500
param = HyperParam(alpha, gamma, eps_Train, eps_Run, episodeNum, False)

#-----------------------------------------------------------------------------
# (2) 環境のｲﾝｽﾀﾝｽを作成
#-----------------------------------------------------------------------------
env = Environment()

#-----------------------------------------------------------------------------
# (3) 方策πの学習と利用を、100回繰り返す
#-----------------------------------------------------------------------------
showObj = ShowFunc()
for rii in range(0,100):

  # Agent ｸﾗｽのｲﾝｽﾀﾝｽを作成する
  agent = Agent(env)

  # SARSA で方策πを学習する
  param.trace = False
  episode_lengths = agent.train(env, param)

  # 方策πの学習で、1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数の履歴をｸﾞﾗﾌ表示する
  showObj.showLearningCurve(episode_lengths)

  # 方策πの学習終了時の、行動価値関数Qπ(s,a)を表示する
  showObj.showQ(agent, env)

  # 学習済の方策で1ｴﾋﾟｿｰﾄﾞ分(開始点から出口に至る経路)を取得する
  param.trace = True
  episode = agent.run(env, param)

  # 1ｴﾋﾟｿｰﾄﾞ分の足跡を表示する
  showObj.showFootprint(agent, env, episode)
```

(8.4.4) SARSAのTD(λ)法への適用

・TD(λ)法にSARSAを適用すると以下のようになります：

・時刻tにおける「n-ｽﾃｯﾌﾟ収益」$G_t^{(n)}$は以下のように焼き直しされます：

時刻tにおける「n-ｽﾃｯﾌﾟ収益」$G_t^{(n)}$

$$G_t^{(n)} \equiv R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q_t(S_{t+n}, A_{t+n}) \quad \cdots(式8\text{-}15)$$

$$= \sum_{k=1}^{n} \gamma^{k-1} R_{t+k} + \gamma^n Q_t(S_{t+n}, A_{t+n}) \quad \cdots(式8\text{-}16)$$

$G_t^{(n)}$ ：時刻tにおける「n-ｽﾃｯﾌﾟ収益」
γ ：割引率 (0≦γ≦1)
R_{t+k} ：時刻t+k における即時報酬 (k=1～n)
$Q_t(s,a)$ ：時刻tで、状態s で行動a をとる時の行動価値関数の推定値
S_t ：時刻t での状態値
A_t ：時刻t でとった行動

・「TD(λ)法」は以下のように焼き直しされます：

TD学習 (TD(λ)法)

$$Q_{t+1}(s) = Q_t(s) + \alpha \times \delta_{t+1}^{\lambda} \times I(S_t=s) \quad \cdots(式8\text{-}17)$$

$$\delta_{t+1}^{\lambda} \equiv G_t^{\lambda} - Q_t(S_t, A_t) \quad \cdots(式8\text{-}18)$$

：TD誤差
（時刻t での状態価値関数の推定値$V_t(S_t)$と
時刻t での「λ-収益」G_t^{λ}との差分）

$I(\chi)$ ：条件χが成立する時1、それ以外の時0 を返す関数
G_t^{λ} ：時刻tにおける「λ-収益」
$Q_t(s,a)$ ：時刻tで、状態s で行動a をとる時の行動価値関数の推定値
S_t ：時刻t での状態値
A_t ：時刻t でとった行動
α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

・拡張された「適格度ﾄﾚｰｽ (ﾃｷｶｸﾄﾞﾄﾚｰｽ、eligibility trace)」$E_{k+1}(s,a)$を以下のように定義します：

拡張された「適格度ﾄﾚｰｽ」$E_{k+1}(s,a)$

$$E_{k+1}(s) \equiv \sum_{t=0}^{k} (\lambda\gamma)^{k-t} \times I(S_t=s, A_t=a) \quad \cdots(式8\text{-}19)$$

$$= I(S_k=s, A_k=a) + (\lambda\gamma) I(S_{k-1}=s, A_{k-1}=a) + \cdots$$

$$\cdots + (\lambda\gamma)^k \times I(S_0=s, A_0=a)$$

$E_{k+1}(s,a)$ ：拡張版の適格度ﾄﾚｰｽ
$I(\chi)$ ：条件χが成立する時1、それ以外の時0 を返す関数
S_t ：時刻t での状態値
A_t ：時刻t でとった行動
γ ：割引率 (0≦γ≦1)
λ ：ﾊﾟﾗﾒｰﾀ

・TD(λ)法の更新式は適格度ﾄﾚｰｽを用いて、以下のようになります：

TD(λ)法の更新式

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \times \delta_{t+1} \times E_{t+1}(s,a) \quad \cdots(式8\text{-}20)$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t)$$

：1-ｽﾃｯﾌﾟTD誤差

（時刻t での行動価値関数の推定値$Q_t(S_t, A_t)$と

時刻t+1 での行動価値関数の推定値との差分）

$$E_{t+1}(s,a) \equiv \lambda \gamma E_t(s,a) + I(S_t=s, A_t=a)$$

：適格度ﾄﾚｰｽ

$I(\chi)$ ：条件χが成立する時1、それ以外の時0 を返す関数

$Q_t(s,a)$ ：時刻tで、状態s で行動a をとる時の行動価値関数の推定値

S_t ：時刻t での状態値

A_t ：時刻t でとった行動

α ：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

【出典・参考】

SARSAとQ学習⇒ https://qiita.com/triwave33/items/cae48e492769852aa9f1

SARSAとQ学習⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)

(8.5) Q学習

(8.5.1) Q学習とは

・SARSAの(式8-13)、(式8-14)のTD誤差の定義:

SARSA (再掲)

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \times \delta_{t+1} \times I(S_t=s,\ A_t=a) \quad \cdots(式8\text{-}13)再掲$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma \times Q_t(S_{t+1}, A_{t+1}) - Q_t(S_t, A_t) \quad \cdots(式8\text{-}14)再掲$$

: 1-ｽﾃｯﾌﾟTD誤差
(時刻t での行動価値関数の推定値$Q_t(S_t, A_t)$と
時刻t+1 での行動価値関数の推定値との差分)

で、方策πに従う次時点の状態S_{t+1}における行動A_{t+1} を
時点t における行動価値関数の最適値 $\max_{a'} Q_t(S_{t+1}, a')$ を与える行動で置き換えます:

$$A_{t+1} \quad \Rightarrow \quad \mathrm{argmax}_{a'}\ Q_t(S_{t+1}, a')$$

この操作は、TD誤差に現れる次時点t+1 の行動価値関数Q_{t+1} の最大値を選ぶことと同じです:

$$Q_t(S_{t+1}, A_{t+1}) \quad \Rightarrow \quad \max_{a'} Q_t(S_{t+1}, a')$$

この更新式による行動価値関数Qの学習を「Q学習 (ｷｭｰｶﾞｸｼｭｳ、Q-Learning)」呼びます。

Q学習

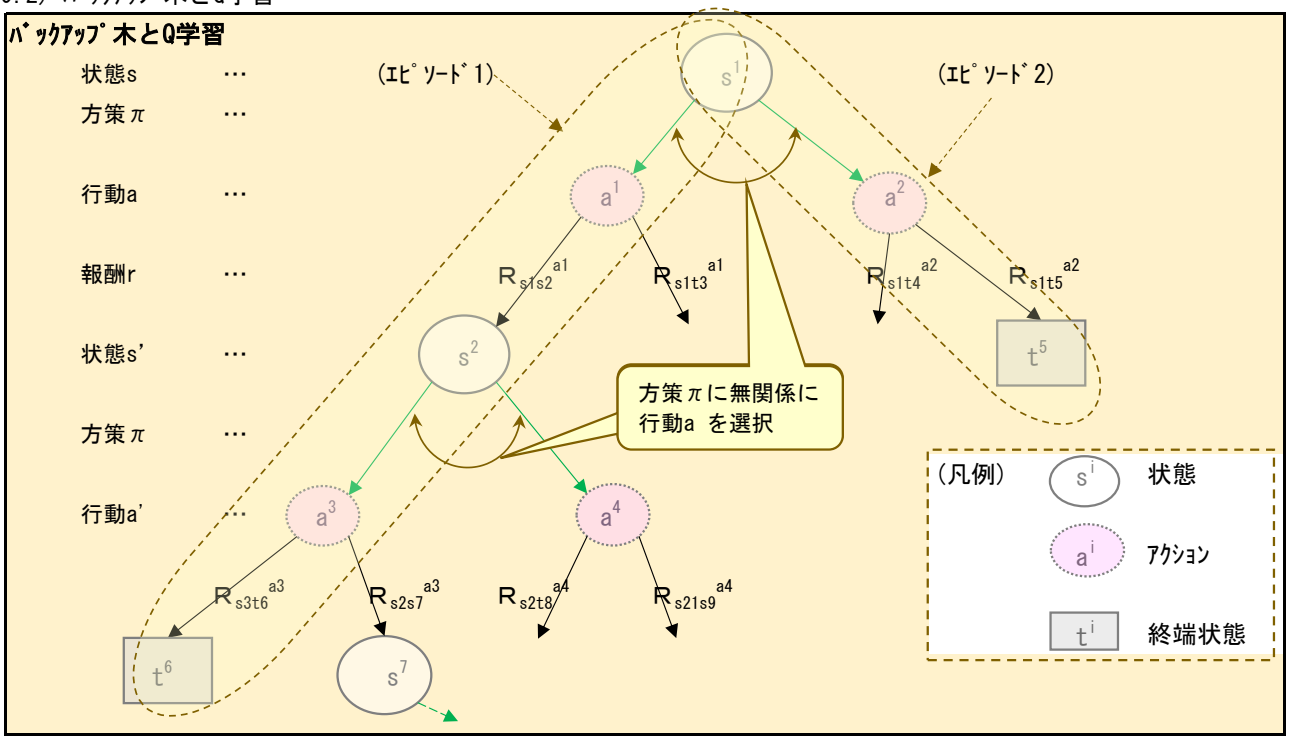
$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \times \delta_{t+1} \times I(S_t=s,\ A_t=a) \quad \cdots(式8\text{-}21)$$

$$\delta_{t+1} \equiv R_{t+1} + \gamma \times \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t) \quad \cdots(式8\text{-}22)$$

: 1-ｽﾃｯﾌﾟTD誤差
(時刻t での行動価値関数の推定値$Q_t(S_t, A_t)$と
時刻t+1 での行動価値関数の推定値との差分)

$I(\chi)$: 条件χが成立する時1、それ以外の時0 を返す関数
$Q_t(s,a)$: 時刻tで、状態s で行動a をとる時の行動価値関数の推定値
R_{t+1} : 時刻t における状態S_t から時刻t+1 での状態S_{t+1} への遷移時の報酬
S_t : 時刻t での状態値
A_t : 時刻t でとった行動
α : 係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)

(図9.2) ﾊﾞｯｸｱｯﾌﾟ木とQ学習



・Q学習は「方策ｵﾌ型」の「価値ﾍﾞｰｽ」の学習手法であり、
状態S_tにおいて、評価対象の方策π(S_t)が示す行動A_tを取る結果遷移する状態をS_{t+1}とし、
遷移先の状態S_{t+1} において、評価対象の方策π(S_{t+1})が指示する行動をA_{t+1}とした時、
TD誤差の計算式(式8-22)における (S_{t+1}, a') の行動a' として、
方策π(S_{t+1})が指示する行動A_{t+1}に無関係の行動 $\mathrm{argmax}_{a'} Q_t(S_{t+1}, a')$ を選択して学習します。

(8.5.2) Q学習でのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)

・Q学習でのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)を以下に記します：

Q学習でのｻﾝﾌﾟﾘﾝｸﾞと方策改善のｱﾙｺﾞﾘｽﾞﾑ(例)

(1) 方策を学習する

・繰り返しｴﾋﾟｿｰﾄﾞを取得しながら

状態遷移情報のｻﾝﾌﾟﾘﾝｸﾞを行い、それに基づく方策改善を繰り返す。

(1.1) 初期設定

(1.1.1) あり得る状態値s の集合Ｓを作成する

(1.1.2) 取りうる行動a の集合Ａを作成する

(1.1.3) 方策π を初期化する (乱数で$\pi(s)$に対応する行動aを決定、$s \in S$, $a \in A$)

(1.1.4) 状態s で行動a を取った時の行動価値関数$Q_\pi(s,a)$ を初期化する

(1.1.5) ｻﾝﾌﾟﾘﾝｸﾞ時のε-greedy法のﾊﾟﾗﾒｰﾀε を設定する ($0 \leqq \varepsilon \leqq 1$)

(1.1.6) 割引率γを設定する ($0 \leqq \gamma \leqq 1$)

(1.1.7) 重み係数αを設定する ($0 \leqq \alpha \leqq 1$)

(1.2) 1回分のｴﾋﾟｿｰﾄﾞを取得しながら、

ｴﾋﾟｿｰﾄﾞ内の遷移情報(状態s、行動a、遷移先状態s'、遷移時報酬$R_{ss'}^a$の系列) をみて

行動価値関数$Q_\pi(s,a)$を推定し、方策改善を繰り返す。

(1st) 状態s を初期状態で設定

(2nd) 状態s でとる行動a をε-greedy法で選択する

・εの割合で、ﾗﾝﾀﾞﾑに行動a を選択する　　：a ←Ａから任意抽出

・1-εの割合で、方策πに従って行動a を選択する：a ←$\pi(s)$

(3rd) 状態s で行動a を取る場合の状態遷移に伴う次の量を計算する。

・遷移による次状態 s'、遷移時報酬 $R_{ss'}^a$

(4th) 行動価値関数$Q_\pi(s,a)$をTD(0)法により推定する。

$$Q_\pi(s,a) \leftarrow Q_\pi(s,a) + \alpha \times \delta_\pi(s,s',a)$$

$$\delta_\pi(s,s',a) \equiv R_{ss'}^a + \gamma \times Q_\pi(s', \pi(s')) - Q_\pi(s,a)$$

(5th) 状態s での方策$\pi(s)$を最大の行動価値関数$Q_\pi(s,a)$ を与える行動a で

更新することにより改善する。

$$\pi(s) \leftarrow \mathrm{argmax}_a \, Q_\pi(s,a)$$

(6th) 遷移先状態s' が終端状態の場合、手順(1.2)を終了する。

終端状態でない場合、状態s を遷移先状態s' で更新し、手順(2nd)へ

$s \leftarrow s'$

(1.3) 別のｴﾋﾟｿｰﾄﾞで学習する為、手順(1.2)へ。

ｴﾋﾟｿｰﾄﾞ取得回数が目標値に達したら、方策学習手順(1)を終了する。

(2) 学習結果の検証と利用

手順(1)で学習した方策πで、着目する初期状態から終端状態まで状態遷移させ、

最適化された状態遷移の1ｴﾋﾟｿｰﾄﾞを取得し、学習が成功したかどうかを確認する。

(8.5.3) Q学習でのｻﾝﾌﾟﾘﾝｸﾞと方策改善の実装例

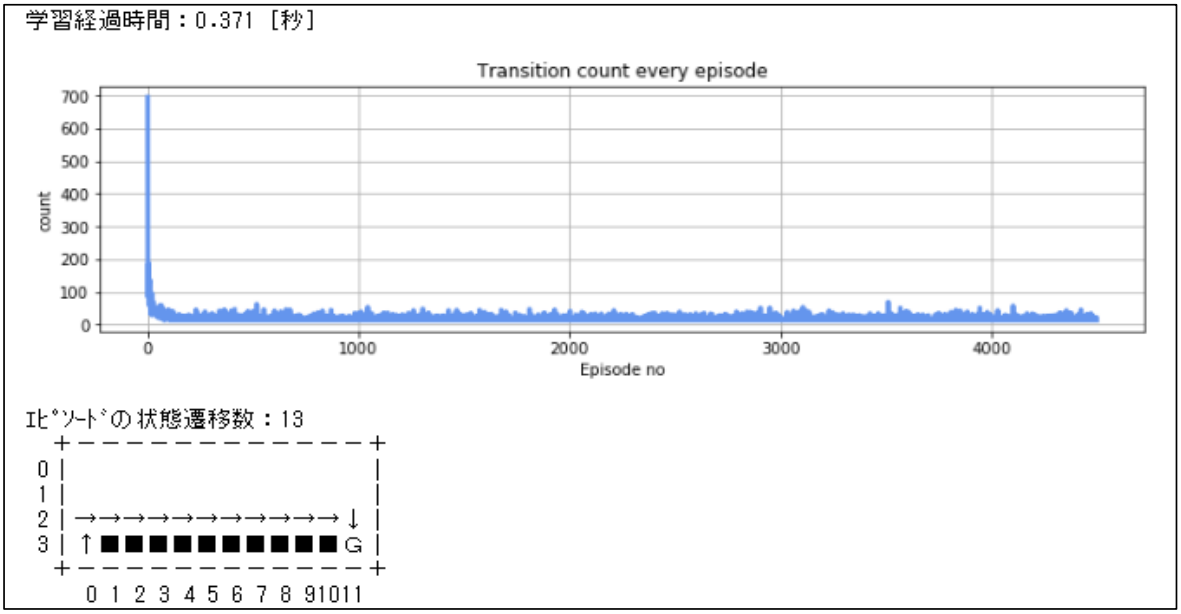
・Q学習で、「(8.1) ﾓﾃﾞﾙﾌﾘｰな手法での例題」の「(例題) 崖歩き」の方策πを作成する実装例と
その実行結果を以下に示します。

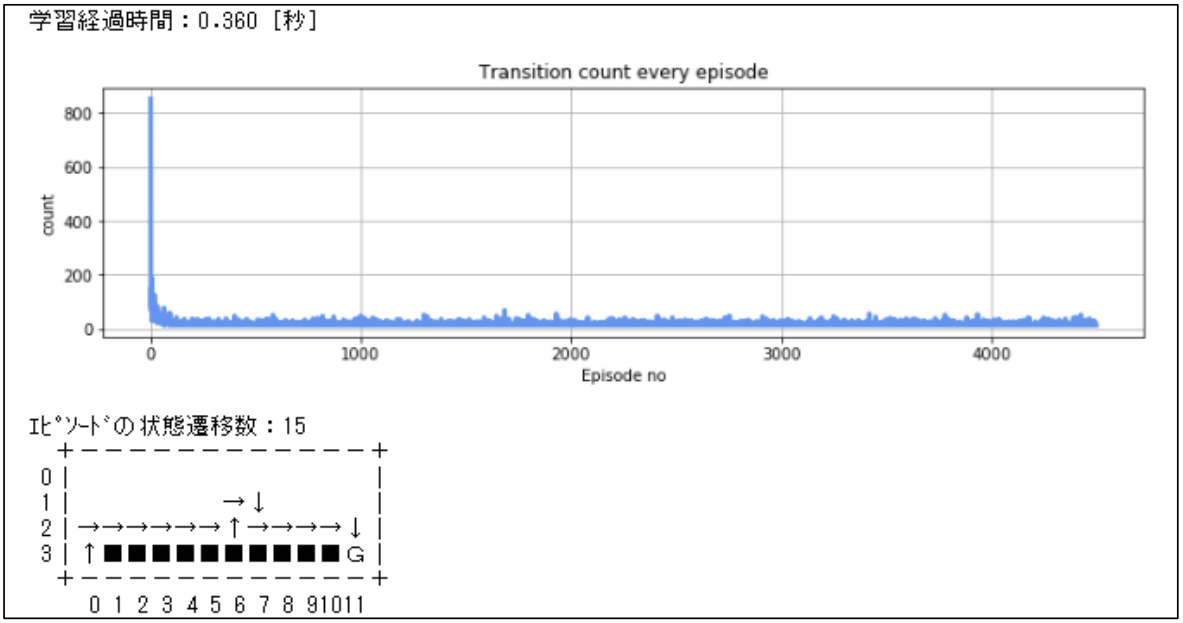
(実行結果)

・ﾊﾟﾗﾒｰﾀの設定にもよりますが、ﾓﾝﾃｶﾙﾛ法より学習時間は一桁少ないです。
・100回ほど、学習⇒実行を繰り返してみたところ、開始点から終点までの状態遷移数は、
以下のような結果となりました。
・実行時のεを0に近い小さい値にしているため、学習した方策πそのものではなく
任意に行動選択する余地を僅かに残していますが、殆ど最短経路を取っています。

(ε=0.001)

1ｴﾋﾟｿｰﾄﾞの状態遷移数	頻度	備考
13	98	最短経路
15	1	2番目に短い経路
23	1	





・Q学習 (Q-Learning)の実装例

・Q学習はSARSA と「行動価値関数$Q_\pi(s,a)$」の推定の仕方が異なり、
関数「getEpisodeAndLearn」がQ学習固有のものとなります。
それ以外はSARSAと同じ実装になっています。

(ﾘｽﾄ11-(09)-3_崖歩き問題の解法その3：Q_Learning)

```
#********************************************************************************
# ﾘｽﾄ11-(09)-3_崖歩き問題の解法その3：Q_Learning
#  Q学習 (Q-Learning)で、
# 「(8.1) ﾓﾃﾞﾙﾌﾘｰな手法での例題」の「(例題) 崖歩き」の決定論的方策πを作成する。
#  ※ 事前に「ﾘｽﾄ11-(09)-0_ﾓﾃﾞﾙﾌﾘｰな手法での共通実装」を実行しておくこと。
#********************************************************************************
#import numpy as np
import time
#import matplotlib.pyplot as plt


#********************************************************************************
# ｸﾗｽ     HyperParam
#--------------------------------------------------------------------------------
# ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀのｸﾗｽ
#--------------------------------------------------------------------------------
# (※) 実装はSARSAと同じ
#********************************************************************************
class HyperParam:

  #********************************************************************************
  # HyperParam ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #--------------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義し、指定値で初期化する：
  #
  # alpha       :ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀα：係数(ｻﾝﾌﾟﾘﾝｸﾞ時の平均用の重み係数に相当)
  # gamma       :ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀγ：割引率
  # eps_Train   :ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀε：学習時のε-greedy法のﾊﾟﾗﾒｰﾀε
  # eps_Run     :εの実行時の値
  #              (ε=0 で実行すると、方策πが完成していなくて終端に到達しない場合がある。
  #               εを 0 より大きいができるだけ小さい値で実行することにより、
  #               学習済の方策πを最大限に生かす方針の下で開始点から終了点まで移動する。)
  # episodeNum  :方策改善時に学習の為に取得するｴﾋﾟｿｰﾄﾞの最大回数
  # trace       :True:追跡出力する、False:しない
  #
  #********************************************************************************
  def __init__(self, alpha, gamma, eps_Train, eps_Run, episodeNum, trace):
    # ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀα、γ、ε
    self.alpha = alpha
    self.gamma = gamma
    self.eps_Train = eps_Train
    self.eps_Run = eps_Run
    self.episodeNum = episodeNum
    self.trace = trace
```

```
#******************************************************************************
# ｸﾗｽ     Agent
#------------------------------------------------------------------------------
# 崖歩きのｴｰｼﾞｪﾝﾄのｸﾗｽ
#******************************************************************************
class Agent:

  #******************************************************************************
  # Agent ｸﾗｽ : ｺﾝｽﾄﾗｸﾀ
  #------------------------------------------------------------------------------
  # 以下のｲﾝｽﾀﾝｽ変数を定義し初期化する：
  #
  #    env            : Environment のｲﾝｽﾀﾝｽ
  #------------------------------------------------------------------------------
  #    actions        : 取りうる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）の集合Ａ
  #    policy         : 決定論的方策π(s)（＝現在の状態s でとる行動a）
  #    q              : 状態s で行動a を取った時の行動価値関数Ｑπ(s,a)
  #
  #------------------------------------------------------------------------------
  # (※) 実装はSARSAと同じ
  #******************************************************************************
  def __init__(self, env):

    # 取りうる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）の集合Ａをﾀﾌﾟﾙ型(Δx,Δy)の配列として保持する
    # (0,-1)：上, (-1,0)：左, (1,0)：右, (0,1)：下
    self.actions = [(0, -1), (-1, 0), (1, 0), (0, 1)]

    # 決定論的方策π を辞書型で保持し、乱数で初期化する
    #  ｷｰ：s、値：a
    #     s： 状態s（＝現在の位置座標(x,y)）
    #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
    self.policy = {}
    for s in env.states:
      self.policy[s] = self.actions[np.random.randint(len(self.actions))]

    # 状態s で行動a を取った時の行動価値関数Ｑπ(s,a) "q" を
    # ディクショナリ型で保持し、0で初期化する
    #  ｷｰ：(s,a)、値：q
    #     s： 状態s（＝現在の位置座標(x,y)）
    #     a： 状態s でとる行動a（＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
    #     q： 状態s で行動a を取った時の行動価値関数Ｑπ(s,a)
    self.q = {}
    for s in env.states:
      for a in self.actions:
        self.q[(s, a)] = 0

  #******************************************************************************
  # Agent ｸﾗｽ : getEpisodeAndLearn
  # ｼﾐｭﾚｰｼｮﾝにより、1回分のｴﾋﾟｿｰﾄﾞを取得しながら、方策改善を行う
  #------------------------------------------------------------------------------
  # 引数：
  #   env      : Environment のｲﾝｽﾀﾝｽ
  #   param    : HyperParam のｲﾝｽﾀﾝｽ
  #   epsilon  : ε-greedy法のﾊﾟﾗﾒｰﾀε
  #   inTrain  : 方策改善の為の学習中の場合:True、学習中でない場合:False
  #------------------------------------------------------------------------------
  # 戻り値：
  # (第１返値) episode:
  #             開始点から出口に至るまでの１回分のｴﾋﾟｿｰﾄﾞの
  #             状態遷移の流れを、ﾀﾌﾟﾙ型ｵﾌﾞｼﾞｪｸﾄ (s,a,r)  の
  #             ﾘｽﾄとして保存したもの
  #
```

```
#          s：状態s （＝位置座標(x,y)）
#          a：状態s でとる行動a （＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#          r：状態s で行動a を取った時の状態s'への遷移に伴う報酬Rss'a
#-------------------------------------------------------------------------
# (※) 実装は Q-Learnig 固有のもの
#*************************************************************************
def getEpisodeAndLearn(self, env, param, epsilon, inTrain):
  # １回分のｴﾋﾟｿｰﾄﾞの状態遷移の流れをﾘｽﾄで持つ
  episode = []

  # 状態s の初期値＝開始点
  s = env.state_ini

  # 「終端状態」となるまで繰り返す
  while True:
    # 状態s でとる行動a
    if np.random.random() < epsilon:
      # εの割合で、ランダムに行動a を選択する
      a = self.actions[np.random.randint(len(self.actions))]
    else:
      # 1-εの割合で、方策πに従って行動a を選択する
      a = self.policy[s]

    # 現在の状態s で、上記で選択した行動a を取り、状態遷移を行う
    r, s_dash = env.transit(s, a)

    # エピソードへ状態遷移情報を追加する
    episode.append((s, a, r))

    # 学習中の場合、
    # 行動価値関数Qπ(s,a)をＴＤ(0)法により推定して、
    # 方策πの改善を行う
    if inTrain:
      self.q[(s, a)] += param.alpha * (r + (param.gamma * self.q[(s_dash, self.policy[s_dash])]) - self.q[(s, a)])
      self.updatePolicy(s)

    # 状態遷移で終点に着いたら「終端状態」となるので、本ｴﾋﾟｿｰﾄﾞは終了する
    x, y = s_dash
    if env.cliff[y][x] == 'G':
      break

    # 状態s を更新
    s = s_dash

  return episode

#*************************************************************************
# Agent ｸﾗｽ : updatePolicy
# 方策改善 （最適のものを１つだけ採用する）
#-------------------------------------------------------------------------
# 指定した状態s における最適行動価値関数Q*(s,a) を求め(=q_max)、
# その時の行動a* (=a_best)を、状態sにおける最適方策π*(s) として
# 行動方策π(s) "policy[s]"を更新する。
#          s：状態s （＝位置座標(x,y)）
#          a：状態s でとる行動a （＝移動ﾍﾞｸﾄﾙ(Δx,Δy)）
#-------------------------------------------------------------------------
# 引数：
#   s     ： 現在の状態s （＝位置座標(x,y)）
#-------------------------------------------------------------------------
# (※) 実装はSARSAと同じ
#*************************************************************************
```

```
def updatePolicy(self, s):
  q_max = -10**10
  a_best = None
  for a in self.actions:
    if self.q[(s, a)] > q_max:
      q_max = self.q[(s, a)]
      a_best = a

  self.policy[s] = a_best

#***************************************************************************
# Agent ｸﾗｽ : train
# ｼﾐｭﾚｰｼｮﾝにより、
# ｴﾋﾟｿｰﾄﾞ取得回数が指定回数 "episodeNum" に達するまで、
# ｴﾋﾟｿｰﾄﾞを取得し、状態価値関数Vπ(s) を見積もる。
#---------------------------------------------------------------------------
# 引数：
#   env    : Environment のｲﾝｽﾀﾝｽ
#   param  : HyperParam のｲﾝｽﾀﾝｽ
#---------------------------------------------------------------------------
# 戻り値：
# (第１返値) episode_lengths:
#      1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数を要素とするﾘｽﾄ型で保持し、その履歴とする
#---------------------------------------------------------------------------
# (※) 実装はSARSAと同じ
#***************************************************************************
def train(self, env, param):
  # 経過時間測定開始
  t1 = time.time()

  # 指定回数分、ｴﾋﾟｿｰﾄﾞ取得と学習を繰り返す
  episode_lengths = []
  seqNo = 0
  for _ in range(param.episodeNum):
    seqNo += 1
    episode = self.getEpisodeAndLearn(env, param, param.eps_Train, True)
    episode_lengths.append(len(episode))

    # 1ｴﾋﾟｿｰﾄﾞ分の足跡を表示する
    if(param.trace == True):
      print("train: episode seqNo={0}".format(seqNo))
      showFootprint(self, env, episode)

  # 経過時間を表示する
  t2 = time.time()
  print("学習経過時間：{0:.3f} [秒]".format(t2-t1))

  return episode_lengths

#***************************************************************************
# Agent ｸﾗｽ : run
# 学習済の方策で1ｴﾋﾟｿｰﾄﾞ分(開始点から出口に至る経路探索)を取得する
#---------------------------------------------------------------------------
# 引数：
#   env    : Environment のｲﾝｽﾀﾝｽ
#   param  : HyperParam のｲﾝｽﾀﾝｽ
#---------------------------------------------------------------------------
# 戻り値：
# (第１返値) episode: 1ｴﾋﾟｿｰﾄﾞ分の状態遷移
#---------------------------------------------------------------------------
# (※) 実装はSARSAと同じ
#***************************************************************************
```

```
  def run(self, env, param):
    # getEpisodeAndLearn 関数を、ε=実行時の値、inTrain=False で実行
    episode = self.getEpisodeAndLearn(env, param, param.eps_Run, False)
    return episode


#*****************************************************************************
# 主制御
#*****************************************************************************

#-----------------------------------------------------------------------------
# (1) ﾊｲﾊﾟｰﾊﾟﾗﾒｰﾀを設定
#-----------------------------------------------------------------------------
alpha = 0.2
gamma = 1
eps_Train = 0.1
eps_Run = 0.001
episodeNum = 4500
param = HyperParam(alpha, gamma, eps_Train, eps_Run, episodeNum, False)

#-----------------------------------------------------------------------------
# (2) 環境のｲﾝｽﾀﾝｽを作成
#-----------------------------------------------------------------------------
env = Environment()

#-----------------------------------------------------------------------------
# (3) 方策πの学習と利用を、100回繰り返す
#-----------------------------------------------------------------------------
showObj = ShowFunc()
for rii in range(0,100):

  # Agent ｸﾗｽのｲﾝｽﾀﾝｽを作成する
  agent = Agent(env)

  # Q学習で方策πを学習する
  param.trace = False
  episode_lengths = agent.train(env, param)

  # 方策πの学習で、1ｴﾋﾟｿｰﾄﾞ毎の状態遷移数の履歴をｸﾞﾗﾌ表示する
  showObj.showLearningCurve(episode_lengths)

  # 方策πの学習終了時の、行動価値関数Qπ(s,a)を表示する
  showObj.showQ(agent, env)

  # 学習済の方策で1ｴﾋﾟｿｰﾄﾞ分(開始点から出口に至る経路)を取得する
  param.trace = True
  episode = agent.run(env, param)

  # 1ｴﾋﾟｿｰﾄﾞ分の足跡を表示する
  showObj.showFootprint(agent, env, episode)
```

(9) その他の方策学習

・以下の節では、未紹介の手法を幾つか取り上げて概観します。

(10.1) 方策勾配法

・方策ﾍﾞｰｽ手法は、方策πをﾓﾃﾞﾙ化して行動価値関数Qへ反映しながら最適な方策πを学習する方法です。
その一つとして「方策勾配法 (ﾎｳｻｸｺｳﾊﾞｲﾎｳ、policy gradient methods)」があります。

・方策のﾓﾃﾞﾙ化の一つに「ｷﾞﾌﾞｽ方策 (ｷﾞﾌﾞｽﾎｳｻｸ、Gibbs policy)」があります。
これは特徴量$\xi(s,a)$と方策ﾊﾟﾗﾒｰﾀθを用いて以下の式で定義されるものです。
なお、「特徴量 (ﾄｸﾁｮｳﾘｮｳ)」は、学習の入力となる測定可能な量のことです。

ｷﾞﾌﾞｽ方策 (方策ﾓﾃﾞﾙ)

$$\pi(a_i|s,\theta) \equiv \{\exp(\theta \times \xi(s,a_i))\} \Big/ \Sigma_{j=1}^{K}\{\exp(\theta \times \xi(s,a_j))\} \quad \cdots(式9\text{-}1)$$
$$= \{\exp(\theta \times \xi(s,a_i))\} \Big/ \{\exp(\theta \times \xi(s,a_1)) + \exp(\theta \times \xi(s,a_2)) + \cdots + \exp(\theta \times \xi(s,a_K))\}$$

$\pi(a|s,\theta)$ ：状態がs の時に、方策ﾊﾟﾗﾒｰﾀθの下で行動a を選択する方策(確率)
θ ：方策ﾊﾟﾗﾒｰﾀ
$\xi(s,a)$ ：特徴量(ξは「ｸｻｲ」と読みます)
a_i, a_j ：状態s で取りうる行動 (i,j=1～K、$a_i, a_j \in A(s)$)

・この関数(式9-1)の形は何かに似ていますね・・・
それは、ﾆｭｰﾗﾙﾈｯﾄﾜｰｸの活性化関数として紹介した「ｿﾌﾄﾏｯｸｽ関数 (softmax function)」です。

$$Y_i = \exp(X_i) \Big/ \Sigma_{j=1}^{k} \exp(X_j) \qquad (i=1\sim k)$$

ｿﾌﾄﾏｯｸｽ関数は、出力が k 個ある場合に、k個の入力値 Xi (i=1～k)の順序関係を保ちながら、
確率としての値 Yi (i=1～k、各値Yiは0～1の範囲で、総和が1になる) に変換する関数です。
k個のｸﾗｽへ分類する際の所属の確率を示すのによく用いられます。

・ｷﾞﾌﾞｽ方策は、ｿﾌﾄﾏｯｸｽ関数との類推でもわかるように、
離散的な行動選択(行動$a_i \in A(s)$, i=1～K)をする場合のﾓﾃﾞﾙです。

・方策ﾊﾟﾗﾒｰﾀθの更新は、目的関数を学習開始時の状態s_0 での期待収益として、
次の勾配法の式で行い、方策関数$\pi(a|s,\theta)$を改善します：

方策ﾊﾟﾗﾒｰﾀθの更新：方策勾配法

$$\theta_{t+1} = \theta_t + \alpha \times \nabla_\theta J(\theta_t) \quad \cdots(式9\text{-}2)$$

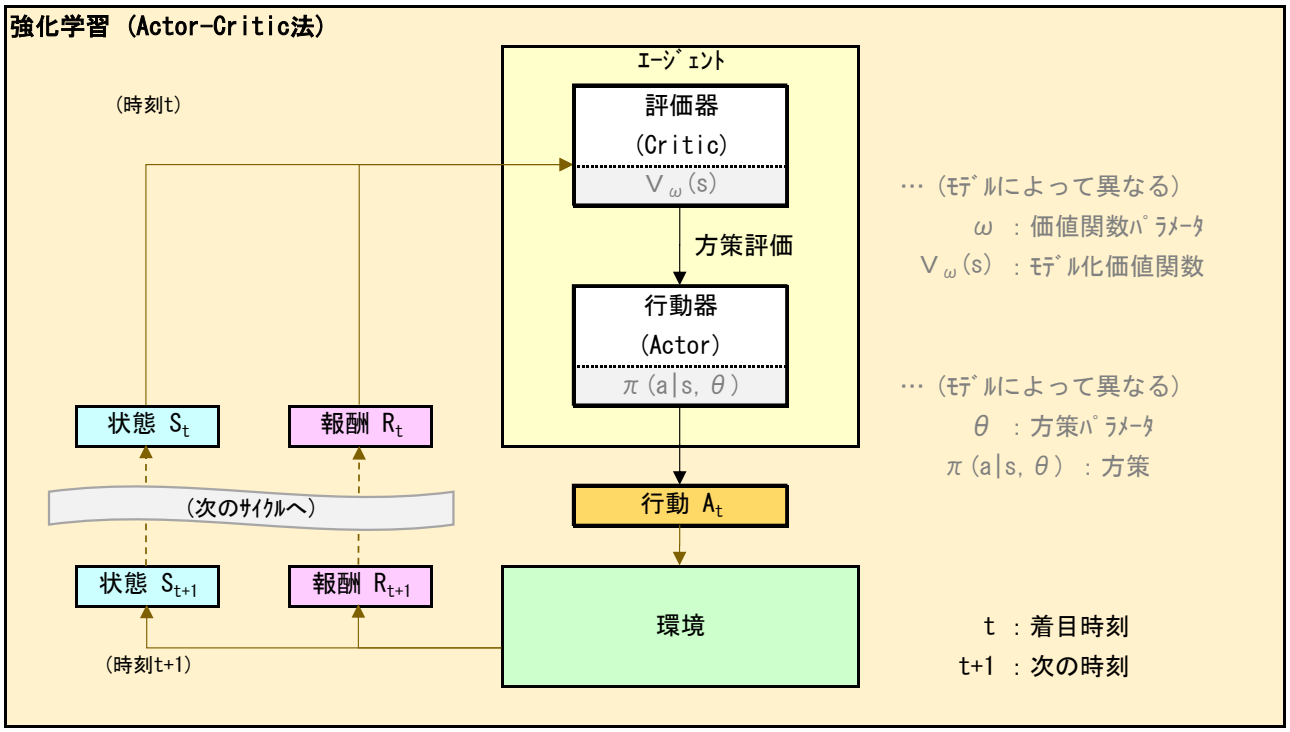
θ_t ：時刻t での方策ﾊﾟﾗﾒｰﾀ
α ：学習率
∇_θ ：偏微分演算子 ($\partial/\partial\theta_1$、$\partial/\partial\theta_2$、…、$\partial/\partial\theta_M$)
$J(\theta_t)$ ：時刻t での方策$\pi(a|s_0,\theta)$の下で計算された価値関数$V_\pi(s_0)$
$V_\pi(s_0)$ ：方策$\pi(a|s_0,\theta)$の下で計算された状態s_0での価値関数
s_0 ：学習開始時の状態

(10.2) Actor-Critic法

・「Actor-Critic法 (ｱｸﾀｰｸﾘﾃｨｯｸﾎｳ)」は方策ﾍﾞｰｽ手法の一つで、TD学習を用います。

・Actor-Critic法は、ｴｰｼﾞｪﾝﾄの機能のうち、方策改善を担う部分を「行動器 (ｺｳﾄﾞｳｷ、Actor)」、
方策評価を担う部分を「評価器 (ﾋｮｳｶｷ、Critic)」として役割分担をします(図10.1)。
そして各々の部分をﾓﾃﾞﾙ化して、交互に学習しながら最適方策を学習します。

(図10.1) Actor-Critic法



・「行動器 (Actor)」のﾓﾃﾞﾙとしては以下のようなものがあります：
(1) 「ｷﾞﾌﾞｽ方策 (ｷﾞﾌﾞｽﾎｳｻｸ、Gibbs policy)」(離散的な行動空間の場合)
(2) 「ｶﾞｳｽ方策 (ｶﾞｳｽﾎｳｻｸ、Gaussian policy)」(連続的な行動空間の場合)
(3) 「再帰型ﾆｭｰﾗﾙﾈｯﾄﾜｰｸ (ｻｲｷｶﾞﾀﾆｭｰﾗﾙﾈｯﾄﾜｰｸ、Recurrent Neural Network、RNN)」

・「評価器 (Critic)」のﾓﾃﾞﾙは、以下のような手順で用意します(例)：
(1) 「価値関数$V_\pi(s)$」を「価値関数ﾊﾟﾗﾒｰﾀω」でﾓﾃﾞﾙ化して「ﾓﾃﾞﾙ化価値関数$V_\omega(s)$」とします。
(2) 「ﾓﾃﾞﾙ化価値関数$V_\omega(s)$」から「損失関数L(ω)」を用意して(TD誤差の最小二乗和など)、
これが最小になるようにωを最適化します。
(3) 最適化されたωで「ﾓﾃﾞﾙ化価値関数$V_\omega(s)$」の値を算出します。

・「評価器 (Critic)」で計算した「ﾓﾃﾞﾙ化価値関数$V_\omega(s)$」を元に
TD誤差が算出されるので、これを元に「行動器 (Actor)」は方策改善を行います。

【出典・参考】
全般⇒「現場で使える！Python 深層強化学習入門」(2019年07月 翔泳社 伊藤多一、他)
Actor-Critic法⇒ http://www.st.nanzan-u.ac.jp/info/gr-thesis/ms/2008/05mm029.pdf

(10.3) DQN (deep Q-network)

・「Q学習」では、行動価値関数Q_tを「状態s」と「行動a」の組合せでの値として管理していました。
然しながら、「状態s」と「行動a」の組合せは、その種類が多くなったり連続値になったりすると、
「状態行動空間の爆発（ｼﾞｮｳﾀｲｺｳﾄﾞｳｸｳｶﾝﾉﾊﾞｸﾊﾂ、The state and action space explosion problem)」
という問題を引き起こし、解を得ることが困難になります。

これの解決方法の一つとして、Q値を関数で近似する方法が考えられました。
更に、この関数近似にﾆｭｰﾗﾙﾈｯﾄﾜｰｸを適用したものが「DQN (deep Q-network)」です。

・「DQN」は、2015年に ｸﾞｰｸﾞﾙ・ﾃﾞｨﾌﾟﾏｲﾝﾄﾞ社により開発された、
ATARI2600ｹﾞｰﾑ(ﾃﾚﾋﾞｹﾞｰﾑ)の操作ｽｷﾙを学習するｱﾙｺﾞﾘｽﾞﾑです。

・ATARI2600上で動作するｹﾞｰﾑのうち、49本(Space Invadors, Video Pinball, Boxing, Breakout, Demon Attack 等々)についてDQNで学習し、そのｴｰｼﾞｪﾝﾄの得点が人のﾚﾍﾞﾙ以上と評価されたものが、29本ありました。

(ATARI2600)

(出典⇒ https://ja.wikipedia.org/wiki/Atari_2600)

・ATARI2600ｹﾞｰﾑのﾓﾃﾞﾙは以下のようなものです：

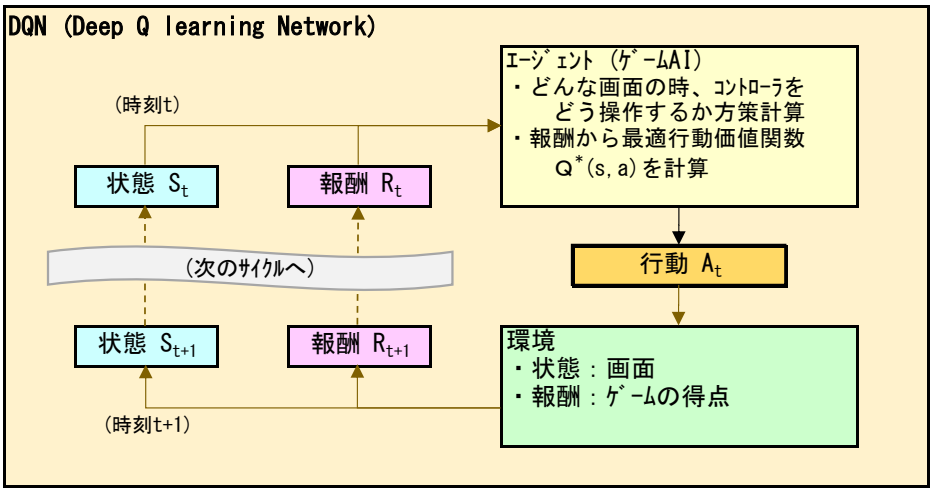
(1) 取りうる行動a の集合Ａ
- ｺﾝﾄﾛｰﾗの操作 (8方向＋赤ﾎﾞﾀﾝ)

(2) 状態値s の集合Ｓ
- 画面(84ﾋﾟｸｾﾙ×84ﾋﾟｸｾﾙ×4ﾁｬﾈﾙ)

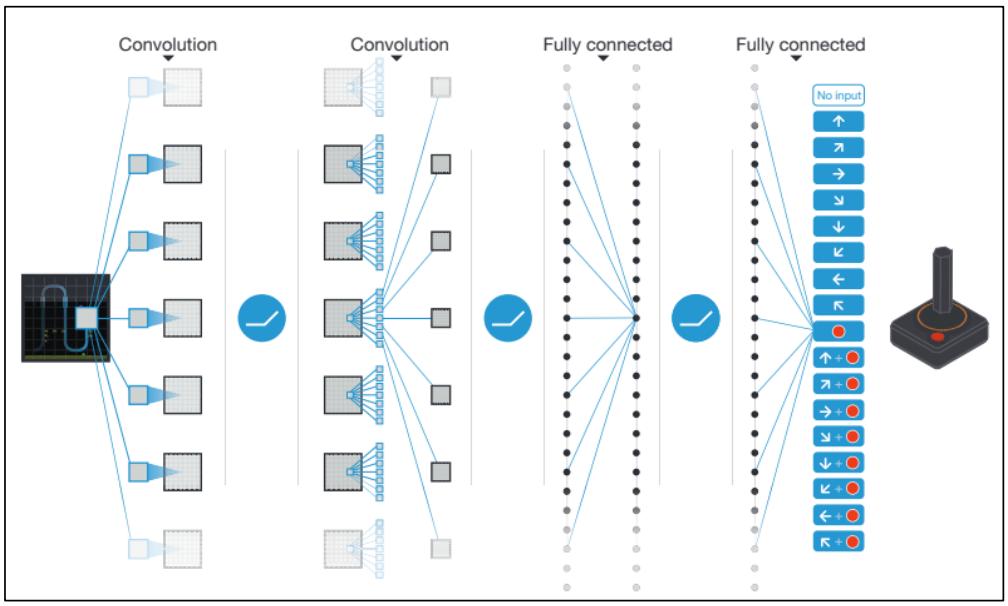
(3) 状態遷移に伴う報酬$R_{ss'}^{a}$
- ｹﾞｰﾑの得点

(Space Invadors)

(出典⇒「Human-level control through deep reinforcement learning」)



・DQNでは、どんな画面s の時、ｺﾝﾄﾛｰﾗをどう操作するかという行動a の方策π(a|s)を
下図のようなﾆｭｰﾗﾙﾈｯﾄﾜｰｸ(CNN)で関数近似しています。入力は画面s で出力は方策π(a|s)です。
近似計算された方策π(a|s)と、報酬$R_{ss'}^{a}$ を元に、最適行動価値関数$Q^{*}(s,a)$を計算して学習します。



【出典・参考】

DQN⇒「ｱﾙﾌｧ碁解体新書」(大槻知史 2017年07月 翔泳社)
「Human-level control through deep reinforcement learning」 ⇒
https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf

(10) まとめ

・強化学習での方策改善手法は、これまで紹介してきたものも含めていくつもあり、
現在も進歩し続け、多くの手法が出てきています。

(1) ﾓﾃﾞﾙﾍﾞｰｽな手法
　(1.1) 方策反復法
　(1.2) 価値反復法

(2) ﾓﾃﾞﾙﾌﾘｰな手法
　(2.1) 価値ﾍﾞｰｽの手法
　　(2.1.1) ﾓﾝﾃｶﾙﾛ法
　　(2.1.2) SARSA
　　(2.1.3) Q学習
　(2.2) 方策ﾍﾞｰｽの手法
　　(2.2.1) 方策勾配法
　　(2.2.2) Actor-Critic法

・ﾃﾞｨｰﾌﾟﾗｰﾆﾝｸﾞの強化学習への適用といったことにはほんのさわりしか言及しませんでしたが、
行動価値関数をﾆｭｰﾗﾙﾈｯﾄﾜｰｸで近似するとか、状態の認識の為にﾃﾞｨｰﾌﾟﾗｰﾆﾝｸﾞを活用するといった
ような研究・応用は盛んに行われています。
強化学習のﾓﾃﾞﾙを構成する要素として、教師あり学習が使われていたりして、興味深いです。

・強化学習のことを調べ始めるまでは、これほど盛んに研究・応用が行われているとは想像して
いませんでした。また、調べるほど、非常に奥が深く範囲の広がりもあることに気づかされます。

・ＡＩによる最適化技術を身に着けて、様々な社会問題に取り組みたいところです。

以上です。

(11) 確認問題

(1) 「崖歩き (Cliff Walking)」の拡張版のプログラミング

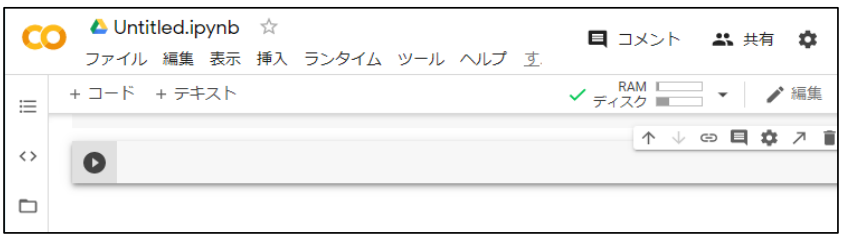
・「崖歩き (Cliff Walking)」の問題で、
掲載している「Q学習」のプログラム「ﾘｽﾄ11-(09)-3_崖歩き問題の解法その3：Q_Learning」を改造して、
(1) 格子点のｻｲｽﾞと、開始点座標'S'、終了点座標'G'、それに崖領域'X' の配置を変更して下さい。
(2) 取りうる行動を上下左右だけでなく、斜め方向にも移動できるようにして下さい。
(3) 上記の改造を加えた上で、最短経路を辿れるプログラムになっているか動作確認してください。

※「ﾘｽﾄ11-(09)-3」は別ファイルで配布します。
動作環境は、Anaconda で Jupyter Notebook を起動して実行する方法などもありますが、
環境を特に用意しなくても、ネット上で無料で使用できる環境が公開されています。

(1) 予め以下のサイトでご自分の Google アカウントを用意しておきます：
「Google アカウントの作成 (https://support.google.com/accounts/answer/27441)」

(2) 「Google Colaboratory」へ手順(1)で作成したアカウントでログインします：
「Google Colaboratory (https://colab.research.google.com)」

(3) ログインすると、以下のような画面が表示されるので、「ノートブックを新規作成」を
選択すると、別のウィンドウで Jupyter Notebook が開きます。
あるいは「Colaboratory へようこそ」画面で、「ファイル」メニューから
「ノートブックを新規作成」でも同じことができます：



(4) これより後は、Anaconda 環境で Jupyter Notebook を開いた時と同様な操作になります。
ライブラリのインストールなどの手間は不要です。

(5) ご自分で作成した Anaconda 環境で実行した場合、機械学習のＰＣリソース占拠で
ＰＣ環境が使用できなくなることがありますので、上記環境での試行をお勧めします。
尚、ファイルの保存や、接続時間などの制限がありますのでご注意下さい。

以上