これからの強化学習

第2章 強化学習の発展的理論

Yuma Yamakura

目次

- 2.1 統計学習の観点から見たTD学習
- 2.2 強化学習アルゴリズムの理論性能解析と
  ベイズ統計による強化学習のモデル化
- 2.3 逆強化学習(Inverse Reinfocement Learning)
- 2.4 試行錯誤回数の低減を指向した手法
  : 経験強化型学習 XoL
- 2.5 群強化学習法
- 2.6 リスク考慮型強化学習
- 2.7 複利型強化学習

2.4 試行錯誤回数の低減を指向した手法 : 経験強化型学習 XoL

2.4節概要

強化学習は環境とエージェントの相互作用
⇒試行錯誤回数が膨大

試行錯誤回数が限定される場合はどうするの？

2.4節概要

学習の高速化にはいくつかの観点
①計算回数/計算の単純化で計算時間を減らす
②試行錯誤(for文のイメージ)を減らす
③計算の並列化

今回は②の話

2.4節概要

試行錯誤を減らす

⇒教示(teaching)

　これは, 学習器に環境の情報(事前知識)を与えること

環境の情報 : 遷移確率, 報酬期待値, etc...

⇒事前知識がない場合は使用できない！

2.4節概要

そもそもこれは, 環境の情報が必要な動的計画法(DP)の
Bellman方程式から強化学習を構成したために起こる
⇒別のアプローチで強化学習を再構成！

2.4節概要

試行錯誤回数を低減

⇒得られた経験を貴重なものとする

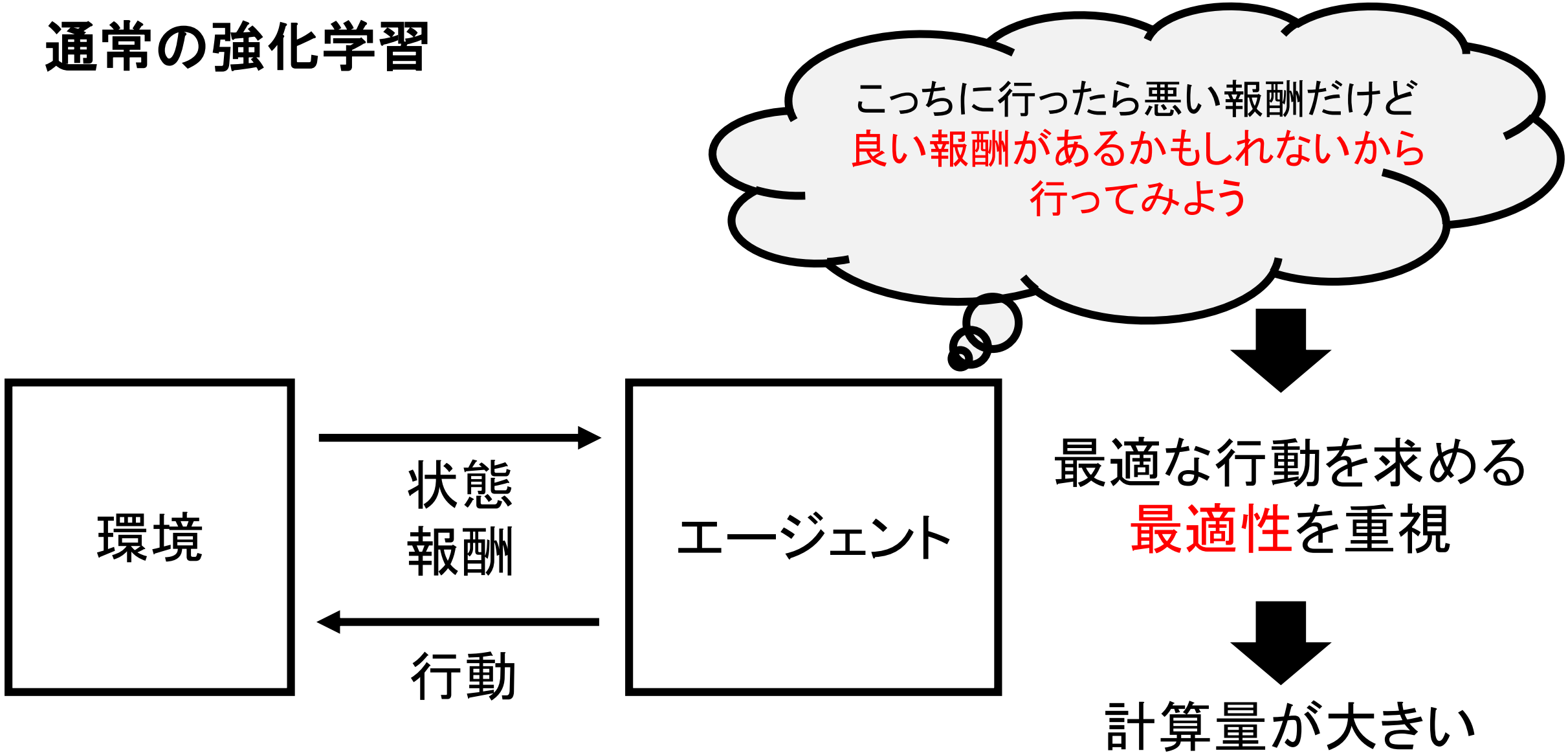
⇒経験型強化学習

(Exploitation-oriented Learning: XoL)

2.4.1 経験強化型学習 XoL

2.4.1 強化学習と XoLの違い

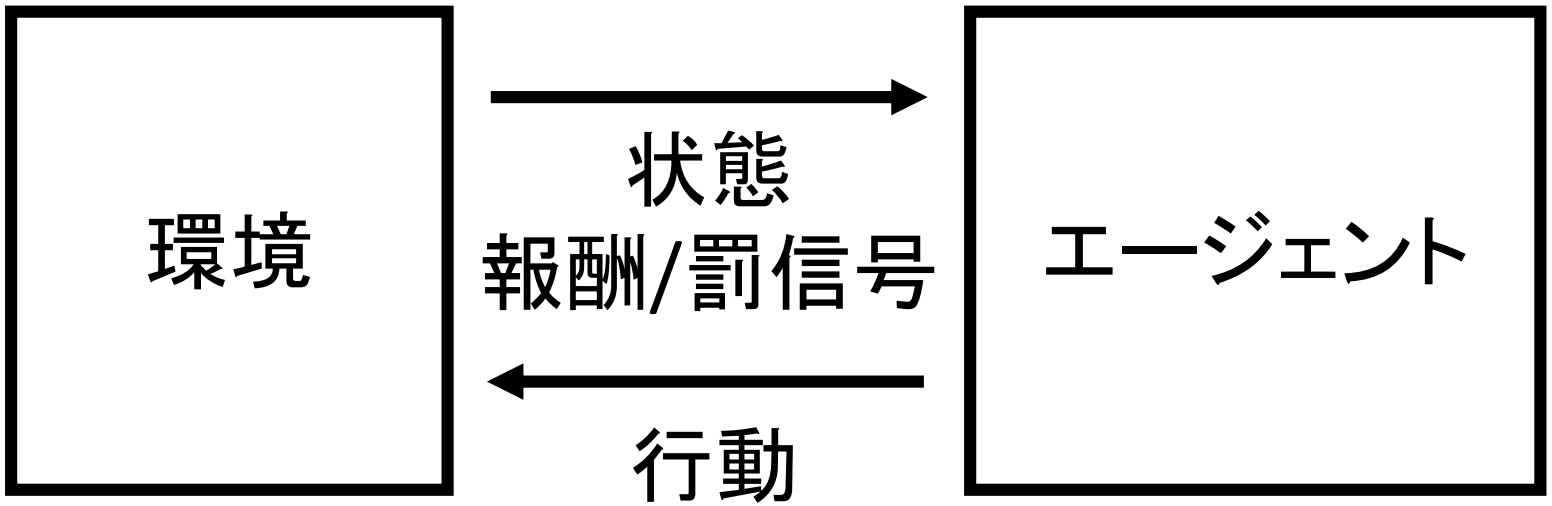
通常の強化学習



2.4.1 強化学習と XoLの違い

経験強化型強化学習(XoL)

悪い報酬を受け取りたくないから
その道は捨ててしまおう
(その行動をとるのをやめよう)



このルールを学習
最適性は捨てた
合理的な判断

計算量が小さい

2.4.1 XoLの特徴

<特徴>

①最適性よりも, 少ない試行回数での合理性

②教師信号は, 目的達成による報酬と制約違反による
罰信号を考えて, 設定が容易でない場合は行わない

	経験強化型学習XoL	DPベース強化学習
試行錯誤回数	少ない	多い
報酬と罰の与え方	優先順位をgiven	値を設定
MDPs下の最適性	合理性の追求	最適性の追求

2.4.1 XoLの手法

- 1種類の報酬のみが存在する問題クラス

⇒Profit Sharing(PS)の合理性定理

合理的政策形成アルゴリズム(RPM)

PS-r*

PS-r#

罰回避政策形成アルゴリズム(PARP)

連続空間に対応した手法[8]

補足：ルール

状態-行動ペアのことをルールという

初期状態あるいは罰を得た直後から次の罰までの
ルール系列をエピソードという

直接罰を得たことのあるルールを罰ルールという
※選択可能なルールが罰ルールまたは無効ルールのみ
　⇒その状態を罰状態という
※罰状態に遷移する可能性のあるルールも罰ルール

補足：有効ルールと無効ルール

あるエピソードで，同一状態で異なるルールを選択
⇒そのあいだのルール系列を迂回系列という

現在までのすべてのエピソードで，常に迂回系列上に
あるルールを無効ルールという
それ以外のルールを有効ルールという

補足：タイプ2の混合

タイプ2の混合とは，
あるとき有効ルールと判定されたルールが，
ある時点以降，常に迂回系列上に存在する
(異なるルールを選択する)ことをいう

補足：タイプ2の混合の例

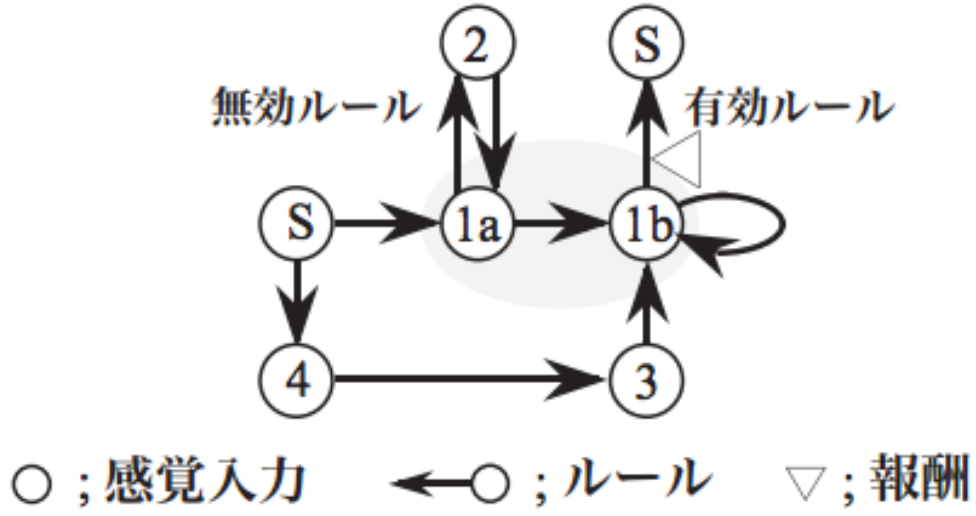


図 3　タイプ 2 の混同の例.

POMDPの考え方で, 実際には1aと1bの状態は別物だが,
強化学習機にとっては1aと1bは信念状態1だと信じているとする

補足：タイプ2の混合の例

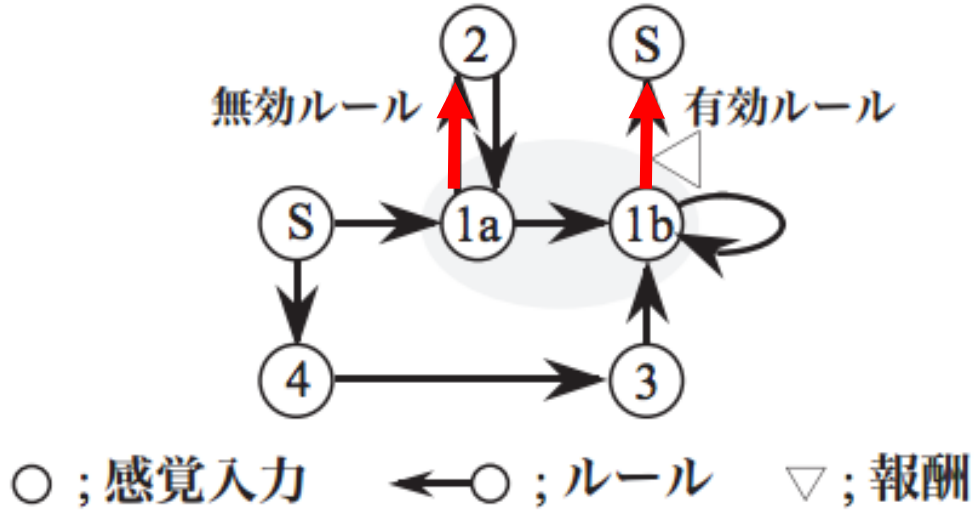


図 3　タイプ 2 の混同の例.

神様の視点

(POMDPの状態がすべてわかっている)

状態1aで上にいっちゃダメ

状態1bで上にいくとよい

補足：タイプ2の混合の例

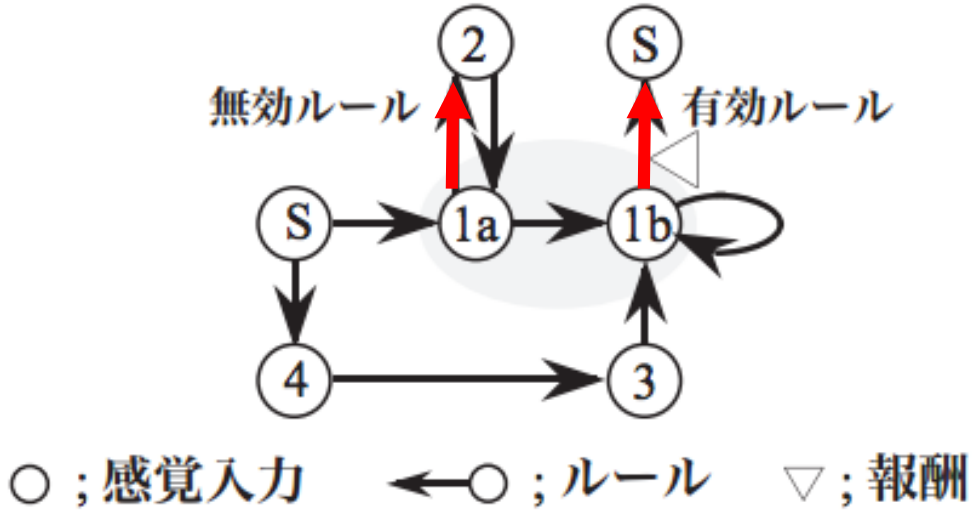


図 3　タイプ 2 の混同の例.

強化学習機の視点

(状態1aと1bを信念状態1とみている)

状態1で上にいく

・1bにいるときは**有効ルール**で正解

・1aにいるときは**無効ルール**に入る

⇒**迂回系列上に有効ルールが存在**

⇒タイプ2の混合を起こす！

補足：タイプ2の混合の例まとめ

	神様の視点	強化学習機
見えない状態の区別	ついている ⇒MDP	ついていない ⇒POMDP
信念状態1での行動によるルール	1aなら右, 1bなら上 ⇒無効ルールに行かない	1aでも1bでも上 ⇒有効ルールと無効ルールの存在 ⇒タイプ2の混合

補足：タイプ1の混合(本の範囲外)

タイプ1の混合とは,

タイプ2の混合と同様に, 状態の価値を混同すること

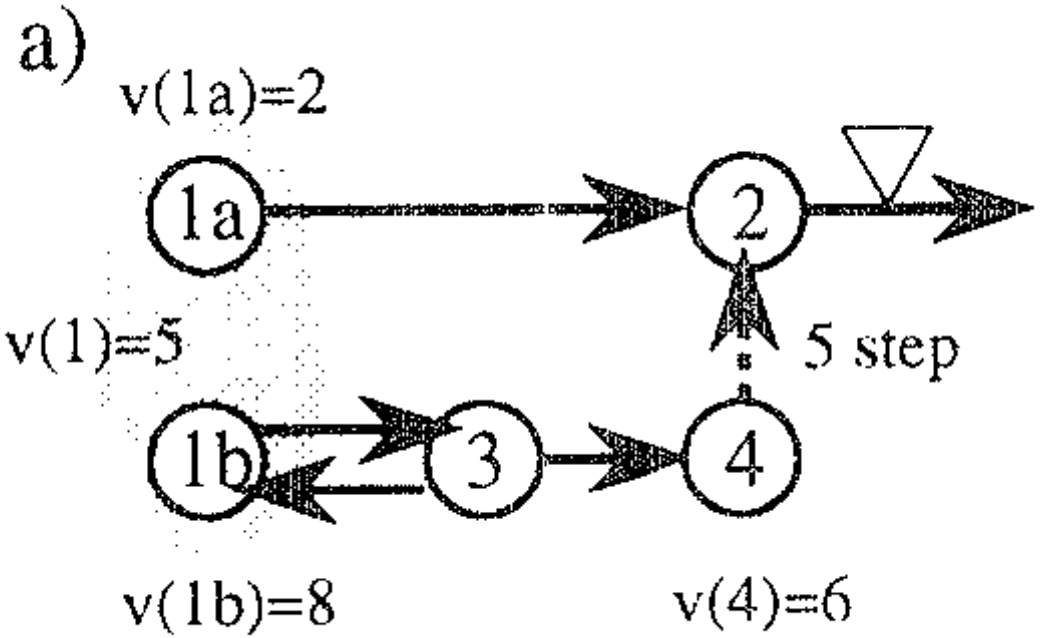
⇒状態の価値を計算するアルゴリズムでは問題

ex)Q-learning , TD(λ)

XoLは, 良いルールを探すので問題なし

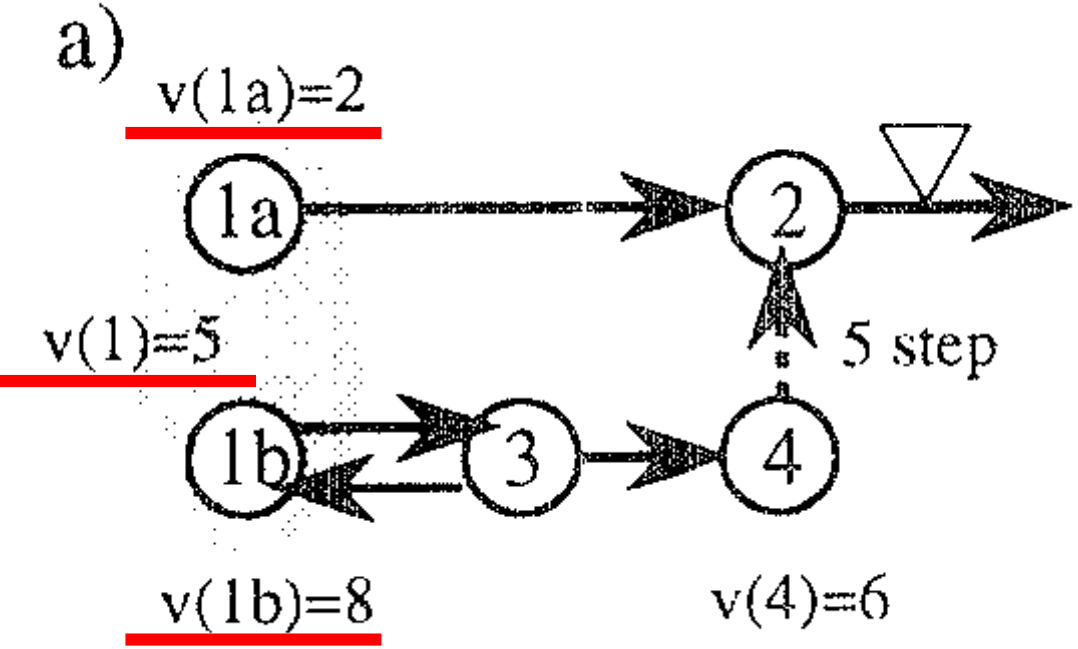
補足：タイプ1の混合の例(本の範囲外)

状態の価値：報酬への最短ステップ数



POMDPの考え方で，実際には1aと1bの状態は別物だが，
強化学習機にとっては1aと1bは信念状態1だと信じているとする

補足：タイプ1の混合の例(本の範囲外)



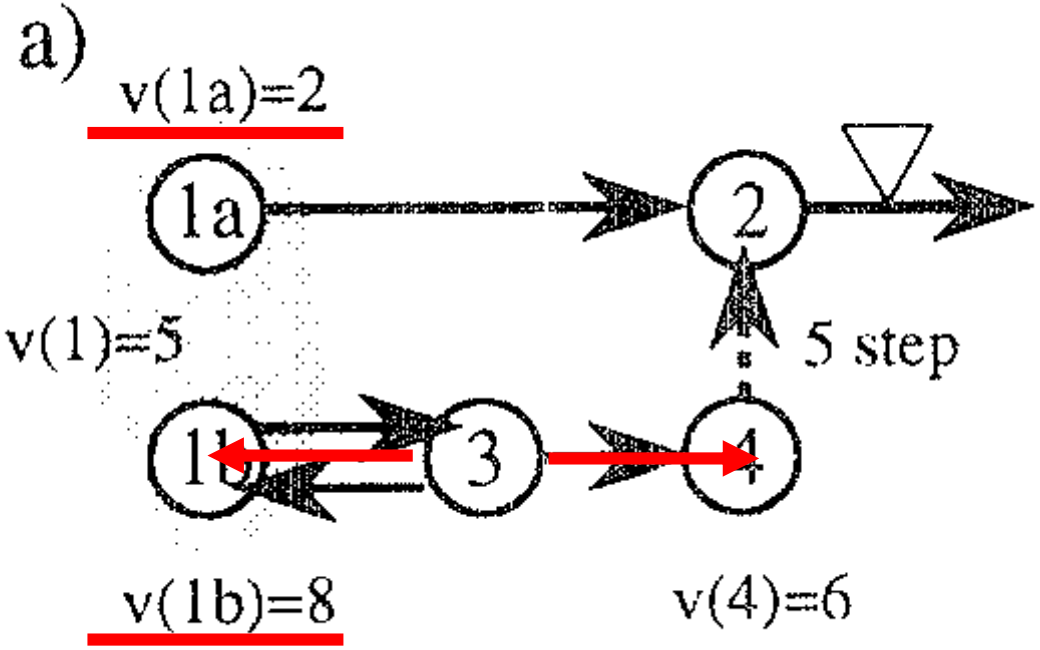
1aの価値はv(1a)=2

1bの価値はv(1b)=8　図から読み取る

状態1aと1bを同じ確率で経験

⇒信念状態1の価値関数v(1)=5

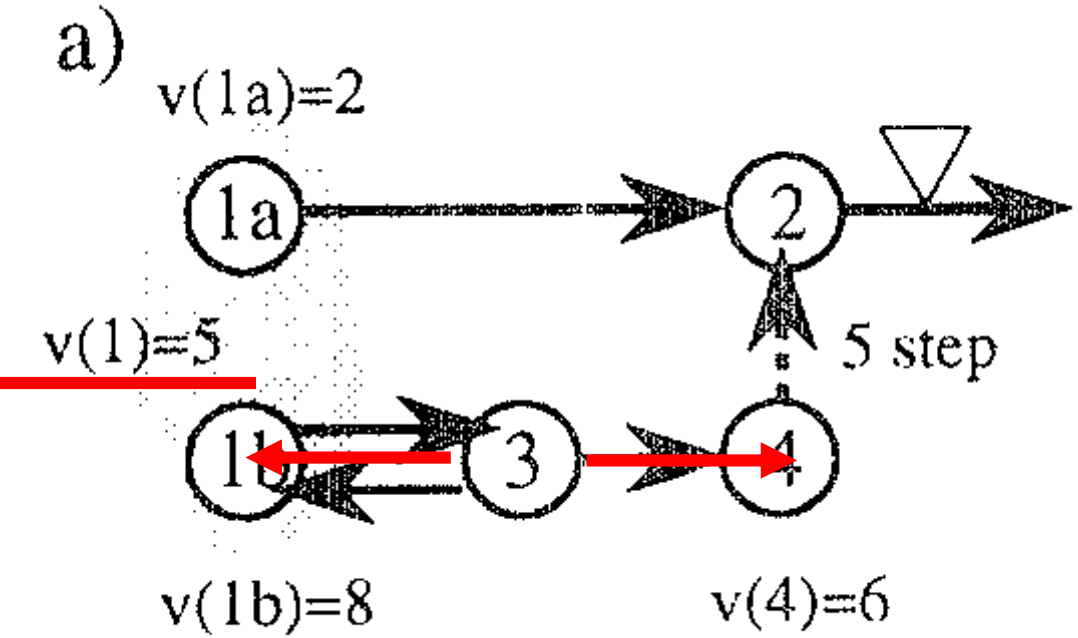
補足：タイプ1の混合の例(本の範囲外)



神様の視点
(POMDPの状態がすべてわかっている)

状態3からどっちに移動すればよい？
・左にいくと v(1b)=8
・右にいくと v(4)=6
つまり右に行くほうが近そう！(正しい)

補足：タイプ1の混合の例(本の範囲外)



強化学習機の視点

(状態1aと1bを信念状態1とみている)

状態3からどっちに移動すればよい？

・左にいくと v(1)=5

・右にいくと v(4)=6

つまり左に行くほうが近そう！(?)

⇒3-1-3-1-・・・というループになる

補足：タイプ1の混合の例まとめ

	神様の視点	強化学習機
見えない状態の区別	ついている ⇒MDP	ついていない ⇒POMDP
3における方策	3で右に行く (正解)	3で左に行く (不正解, ループに) ⇒タイプ1の混合

補足：タイプ1の混合とタイプ2の混合まとめ

重要な概念なのでまとめておきます

	タイプ1の混合	タイプ2の混合
定義	状態の価値を混合	状態のルールの 価値を混合
使えない アルゴリズム	・Q-learning ・TD(λ) などの状態価値を 用いるもの	Profit-Sharing (後述)

補足：合理的政策

各状態に対し，選択すべき行動を与える関数を
政策(=方策, policy)という

単位行動当たりの**期待獲得報酬量が正**である政策を
合理的政策という
(罰ルールを回避して報酬を得ることのできる政策？)

最大化：最適政策

2.4.2 1種類の報酬に対応した XoL 手法

2.4.2 Profit Sharing

- Profit Sharing

報酬を得たときに, エピソード単位で評価値を強化

$$\omega_{r_i} = \omega_{r_i} + f_i$$

ω_{r_i} : ルールr_iの評価値

f_i : 報酬からiステップ前の強化値

2.4.2 Profit Sharingの合理性定理

- Profit Sharingの合理性定理

各状態で最も評価値の高いルールを選択し続ける政策が
合理的政策となるための強化関数の必要十分条件

$$\forall i = 1, 2, \ldots, W, L \sum_{j=i}^{W} f_j < f_{i-1}$$

W：エピソードの最大長

L：同一状態下に存在する有効ルールの最大個数

未知だが，$L =$ 可能な行動の種類-1で十分

2.4.2 強化関数の例

- Profit Sharingの合理性定理を満たす強化関数

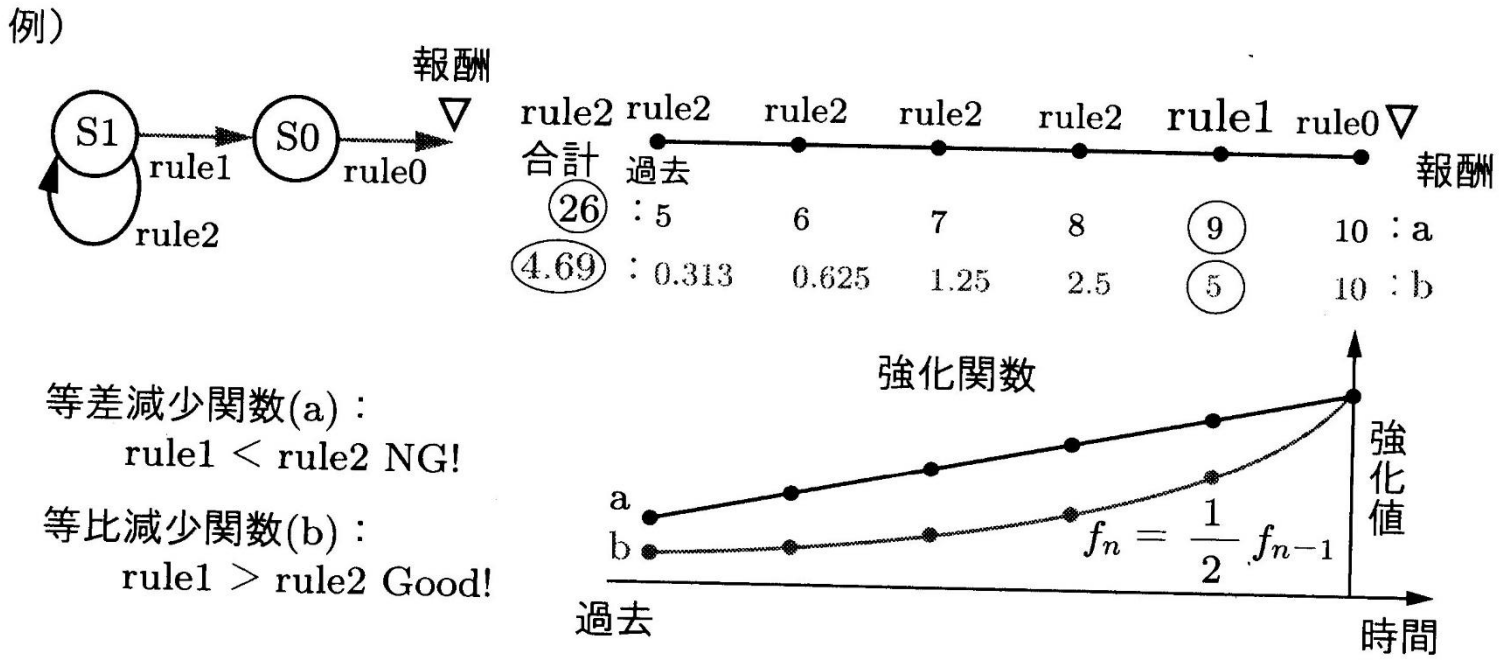


図 2.4.1 定理を満たす等比減少関数の例

行動の種類が 2 であるエージェントを例に示す環境に適用した場合，公比 2 の等比減少関数 (b) を用いれば，状態 S1 において，有効ルールである rule1 の評価値は 5 となり，無効ルールである rule2 の評価値 $4.69(= 2.5 + 1.25 + 0.625 + 0.313)$ よりも大きくなる．一方，公差 -1 の等差減少関数 (a) を用いた場合は，rule2 の評価値 $(= 26)$ が rule1 の評価値 $(= 9)$ よりも大きくなり，有効ルールよりも無効ルールが強化されてしまう結果となる．

公比
1/(行動の種類)

2.4.2 合理的政策形成アルゴリズム(RPM)

Profit Sharingが基本なので飛ばします

• RPM

あるエピソードにおいて同一の状態に対する行動選択のなかで, 報酬に最も近い位置で選択された行動を含むルールが有効ルール

⇒発見した有効ルールを選び続けると合理的政策を獲得

2.4.2 タイプ2を含む場合

Profit Sharingはタイプ2の混合があると使えない
⇒以下のアルゴリズムを用いる

- RPM

マルチスタート法により合理的政策の獲得可能

- PS-r*

RPMの拡張, 統計的検定の選択回数に着目

- PS-r#

RPMの拡張, ルールの選択回数に着目

2.4.3 報酬および罰に対応した XoL 手法

2.4.3 罰回避政策形成アルゴリズム(PARP)

• PARP

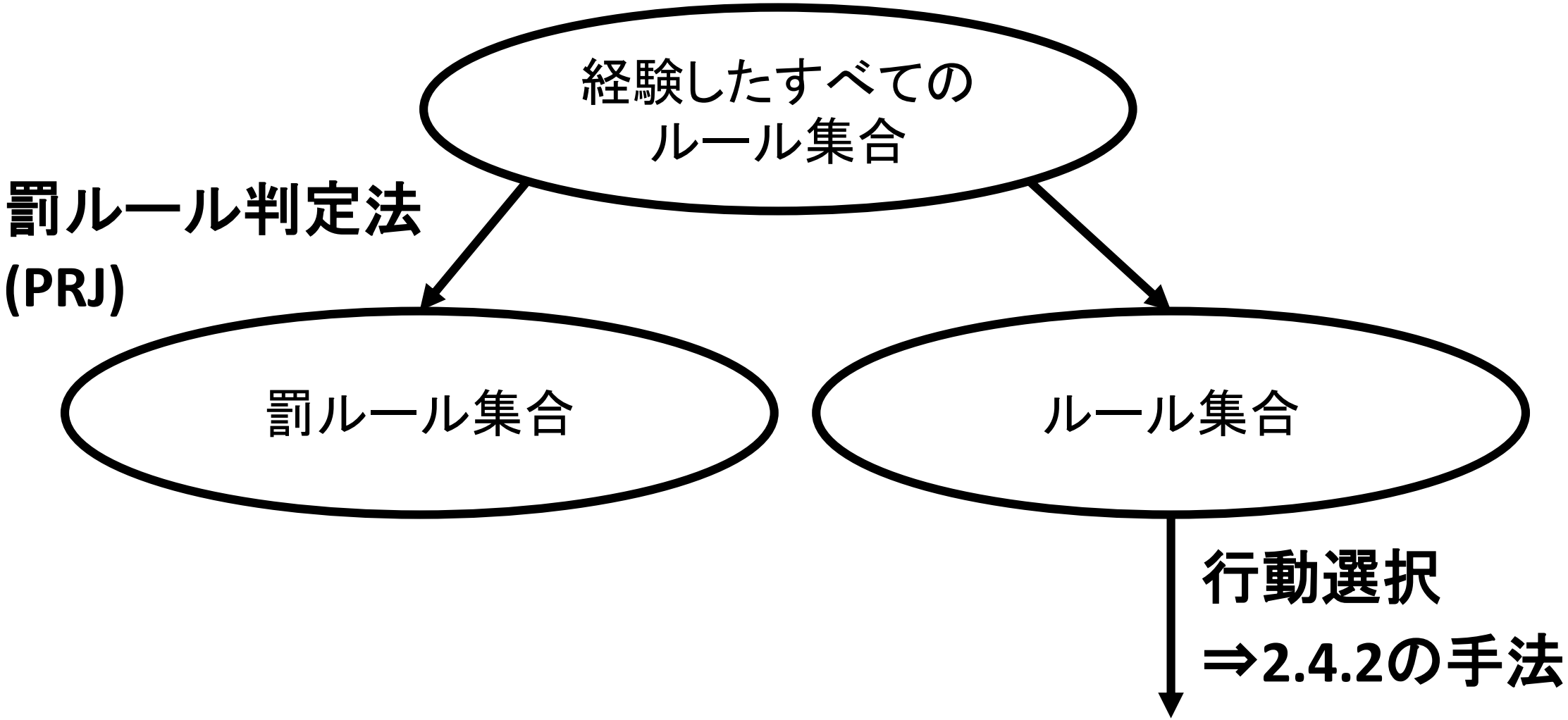
報酬と罰がたかだか1種類ずつ存在する問題クラスに対応

経験したすべてのルール集合から罰ルールを発見

⇒除外したルール集合を作成し,

　その集合のなかで行動選択

2.4.3 罰回避政策形成アルゴリズム(PARP)



2.4.3 罰ルール判定法(PRJ)

---

procedure 罰ルール判定法 (PRJ)
begin
 これまで経験したエピソードのなかで直接罰を得たことのあるルールにマークする
 do
  以下の条件が成立する状態にマークする
   その状態で選択可能なルールが無効ルールまたはマークされたルールのみである
  以下の条件が成立するルールにマークする
   そのルールで遷移可能な状態のなかの少なくとも一つがマークされている
 while 新たにマークされた状態が存在する
end

図 **2.4.2**　罰ルール判定法 (PRJ)[9]

2.4.3 改良型罰回避政策形成アルゴリズム

- 改良型PARP

PARPは状態数2乗オーダーの記憶容量が必要

⇒記憶容量の改善

2.4.3 改良型罰回避政策形成アルゴリズム

罰ルール度 $PL(xa) = \frac{N_p(xa)}{N(xa)}$

xa : ルール

$0 \leq PL(xa) \leq 1$: 各罰ルールの罰状態への遷移確率

$N_p(xa)$: xaが罰と判定された回数

$N(xa)$: xaを選んだ回数

$PL(xa) > \gamma$ のときxaを罰ルールとする(確率 + 閾値)

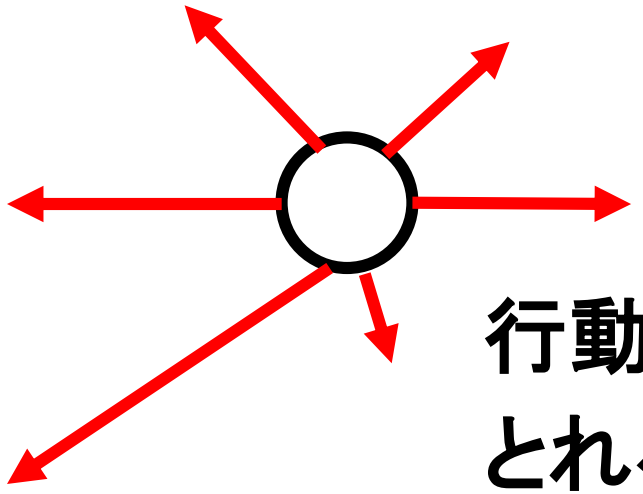
2.4.4 連続値で与えられる感覚入力への対応

2.4.4 連続値の問題点

連続値だと, タイプ2の混合が生じる

⇒離散化する必要

⇒離散化は基底関数を用いる



行動は全方位に
とれるのでは？

2.4.4 基底関数(とばす)

S_t : 時刻tでの感覚入力

a_t : S_tで選択した行動

S_{t+1} : S_tでa_tを選択したときの遷移先

基底関数

$$f(d) = e^{-\frac{1}{2}\sum_{i=1}^{n}\frac{(\mu_i - d_i)^2}{\sigma_i^2}}$$

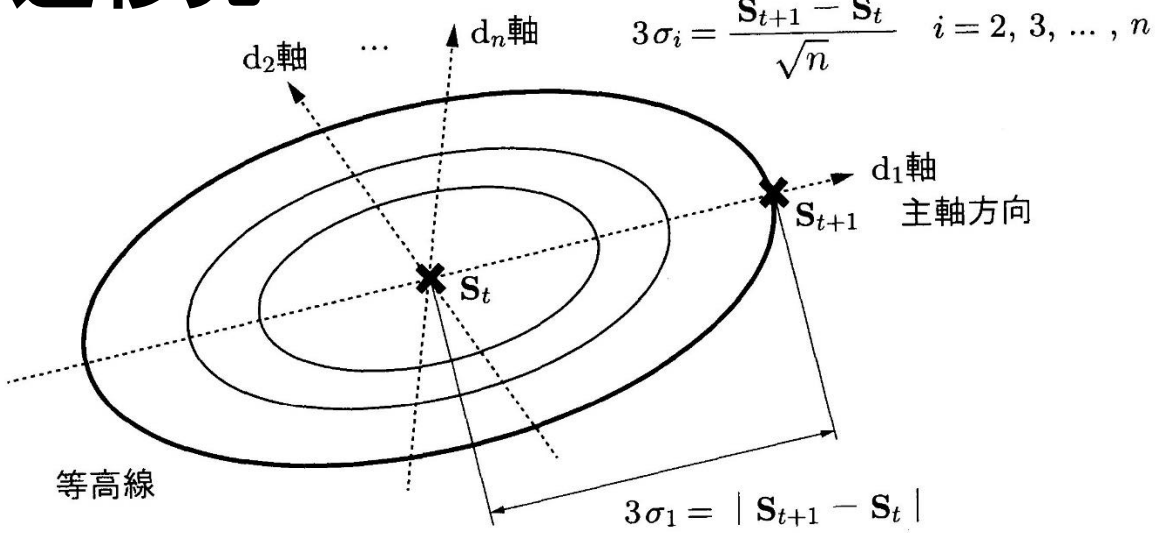


図 2.4.3 基底関数の形状

関数の主軸方向は $\mathbf{S}_{t+1}-\mathbf{S}_t$ で定義する (d_1 軸)．主軸以外の方向は，各々が直交するようにグラムシュミットの直交化法などを用いて生成する ($d_2, \ldots, d_n$ 軸)．主軸の裾野の広さは，$3\sigma_1 = |\mathbf{S}_{t+1}-\mathbf{S}_t|$，主軸以外の裾野の広さは，$3\sigma_i = \frac{|\mathbf{S}_{t+1}-\mathbf{S}_t|}{\sqrt{n}}$ $(i = 2, 3, \ldots, n)$ とする．ここで主軸以外の方向の裾野の広さに $\frac{1}{\sqrt{n}}$ を乗じているのは，経験した方向にバイアスをかけた関数を得るためである．なお，$\mathbf{S}_t = (\mu_1, \ldots, \mu_n)$ である．

2.4.4 基底関数(とばす)

任意の観測dを得た時, 各基底関数への近さをfにより比較

fに$0 < f_{para} \leq 1$となる閾値を基底

⇒各基底ごとに守備範囲を制御

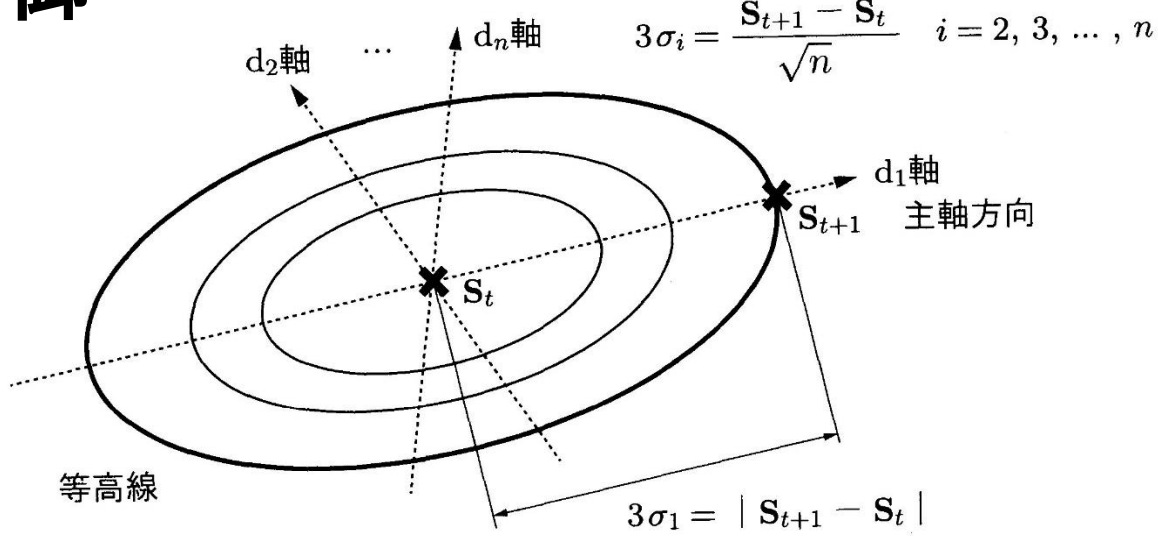


図 2.4.3 基底関数の形状

関数の主軸方向は$\mathbf{S}_{t+1}-\mathbf{S}_t$で定義する($d_1$軸). 主軸以外の方向は, 各々が直交するようにグラムシュミットの直交化法などを用いて生成する($d_2, \ldots, d_n$軸). 主軸の裾野の広さは, $3\sigma_1 = |\mathbf{S}_{t+1}-\mathbf{S}_t|$, 主軸以外の裾野の広さは, $3\sigma_i = \frac{|\mathbf{S}_{t+1}-\mathbf{S}_t|}{\sqrt{n}}$ $(i = 2, 3, \ldots, n)$ とする. ここで主軸以外の方向の裾野の広さに$\frac{1}{\sqrt{n}}$を乗じているのは, 経験した方向にバイアスをかけた関数を得るためである. なお, $\mathbf{S}_t = (\mu_1, \ldots, \mu_n)$である.

2.4.4 基底関数(とばす)

各学習の行動選択/実行ごとに基底関数を生成し,記憶する
⇒行動選択時には, 現在の感覚入力に最も合致する
　基底関数に記憶されている行動を選択すればよい

※深層学習によるXoLは
　2.4.6

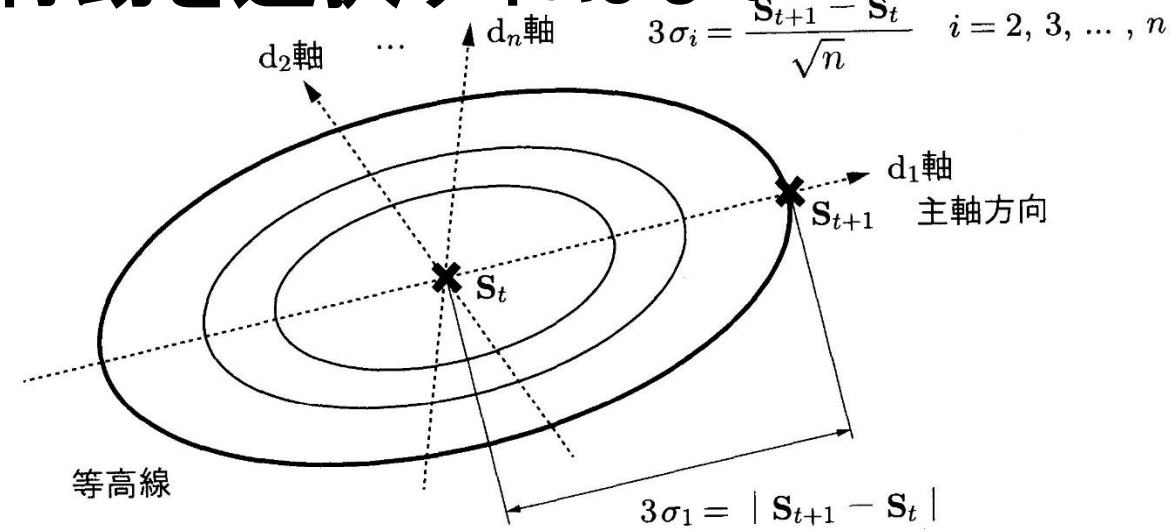


図 2.4.3　基底関数の形状
関数の主軸方向は $\mathbf{S}_{t+1}-\mathbf{S}_t$ で定義する (d_1 軸)．主軸以外の方向は, 各々が直交するようにグラムシュミットの直交化法などを用いて生成する ($d_2, \ldots, d_n$ 軸)．主軸の裾野の広さは, $3\sigma_1 = |\mathbf{S}_{t+1}-\mathbf{S}_t|$, 主軸以外の裾野の広さは, $3\sigma_i = \frac{|\mathbf{S}_{t+1}-\mathbf{S}_t|}{\sqrt{n}}$ $(i = 2, 3, \ldots, n)$ とする．ここで主軸以外の方向の裾野の広さに $\frac{1}{\sqrt{n}}$ を乗じているのは，経験した方向にバイアスをかけた関数を得るためである．なお, $\mathbf{S}_t = (\mu_1, \ldots, \mu_n)$ である．

2.4.5 XoL の応用例

2.4.5 科目分類支援システム

NIAD-QEが定める修得単位の審査の基準に合致するように科目を分類,整理, 申告する必要

表 2.4.2　「情報工学」区分における専門科目に関する修得単位の審査の基準
専門科目に関する各科目区分の名称と最低必要単位数

番号	科目区分の内容	単位数
1	情報工学基礎に関する科目	4
2	計算機システムに関する科目	4
3	情報処理に関する科目	4
4	電気電子・通信・システムに関する科目	—

2.4.5 科目分類支援システム

NIAD-QEは分類の正しさを検討
⇒めんどくさい

表 2.4.2　「情報工学」区分における専門科目に関する修得単位の審査の基準
専門科目に関する各科目区分の名称と最低必要単位数

番号	科目区分の内容	単位数
1	情報工学基礎に関する科目	4
2	計算機システムに関する科目	4
3	情報処理に関する科目	4
4	電気電子・通信・システムに関する科目	—

2.4.5 科目分類支援システム

支援するための, 科目分類支援システム(CCS)

発展形 : 分類候補数の能動的調整を可能にした

科目分類支援システム(ACCS)

これにより, 各科目区分に分類される可能性が高い科目を

優先的に専門委員に提示

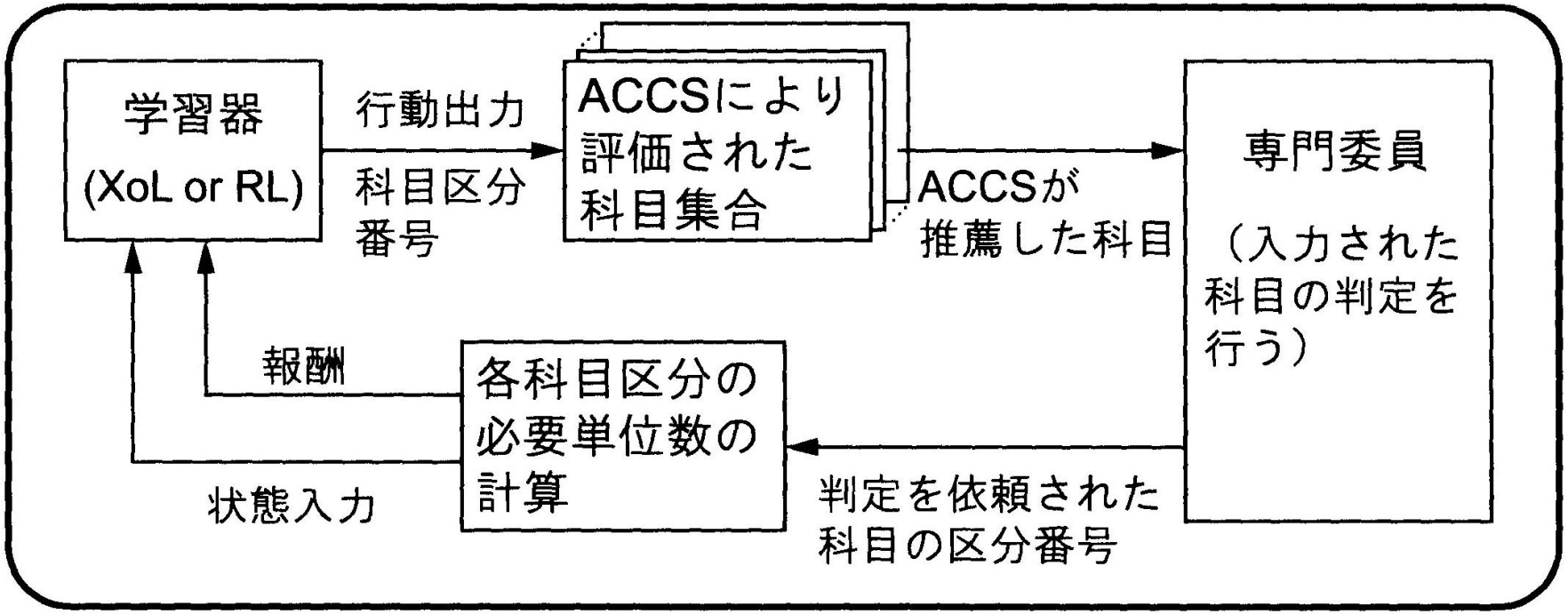
2.4.5 現実的なシステム構成

理想的には, 専攻の区分ごとに異なることが予想される

科目区分の特性を学習してほしい

⇒過去の実際の判定例からXoLであらかじめ学習！

ACCS with XoL



2.4.5 アルゴリズムの性能

シミュレーションの詳細は本に書いてある

平均科目審査数が以下の通り

表 2.4.3　各学習手法の結果

	A	B	C	D	E	F	G
Q-learning	6.9	22.6	5.2	10.2	16.4	15.0	7.3
ACCS with XoL	6.9	22.0	5.0	6.0	16.3	15.0	6.1
ACCS with XoL/PN	4.8	6.4	6.6	7.0	9.7	4.7	7.4
Random	10.0	24.2	8.9	15.9	17.1	17.2	14.2

ACCS with XoLを使うととても良い結果に！

2.4.5 腱駆動2足歩行ロボット

状態：着地している足の状態(2次元, 離散)
+ 腰の座標など(6次元, 連続)

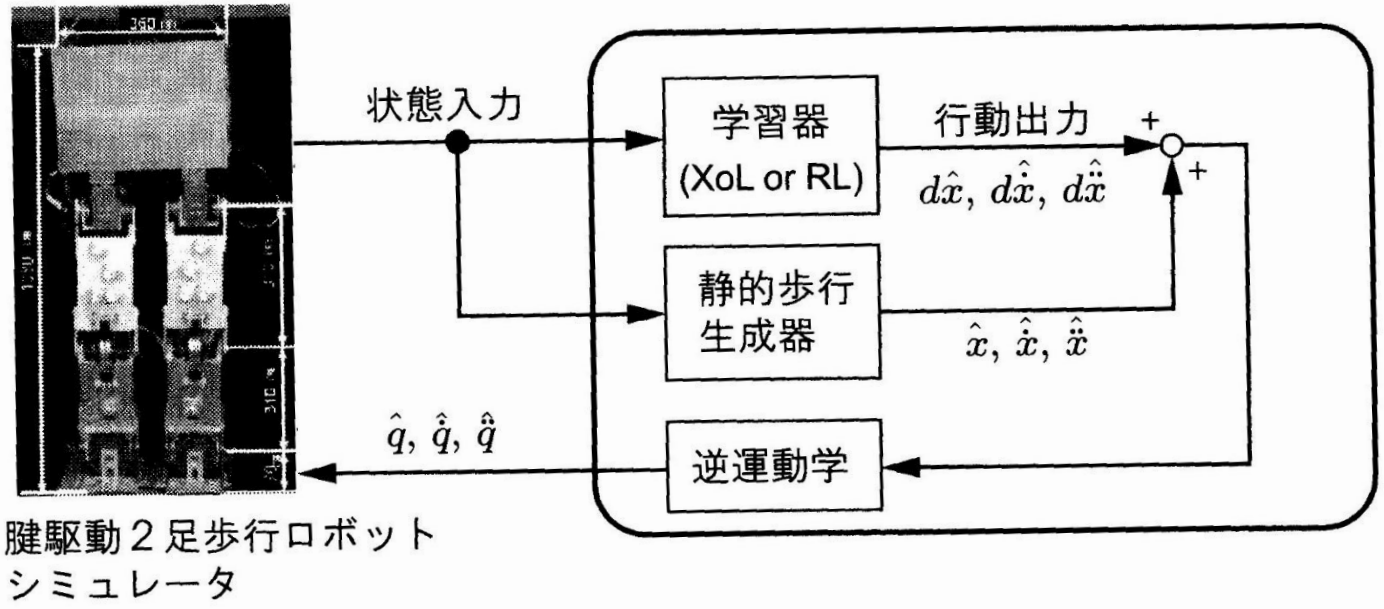


図 2.4.5 腱駆動 2 足歩行ロボットの腰軌道の学習
ロボットは，静的歩行生成器の出力に学習器の出力を加えることで動的歩行の実現を図る．その際，初期静止状態から半歩 (0.75 秒) の学習を行い，その後，その学習が $LT\,(=15)$ 回成功するたびに，目標時間をさらに半歩延長し，全体として 5 歩進み停止することを目指す．

2.4.5腱駆動2足歩行ロボット

連続値 ⇒ 2.4.4の離散化

学習の高速化 ⇒ K回報酬を受けた状態は固定状態

(最大の評価値をもつ行動のみを選択)

実ロボットのシミュレータの成功率結果は以下

K	1	3	5	10	15	20	∞
Q-learning	60%	87%	80%	80%	93%	80%	73%
改良型PARP	すべて失敗						

2.4.6 XoL の発展性

2.4.6 ハイブリッド手法

複数の学習機をハイブリッドさせた手法：MarcoPolo

素早い学習のProfit Sharing

+ 環境の道程を行う k-確実探査法 の切り替え

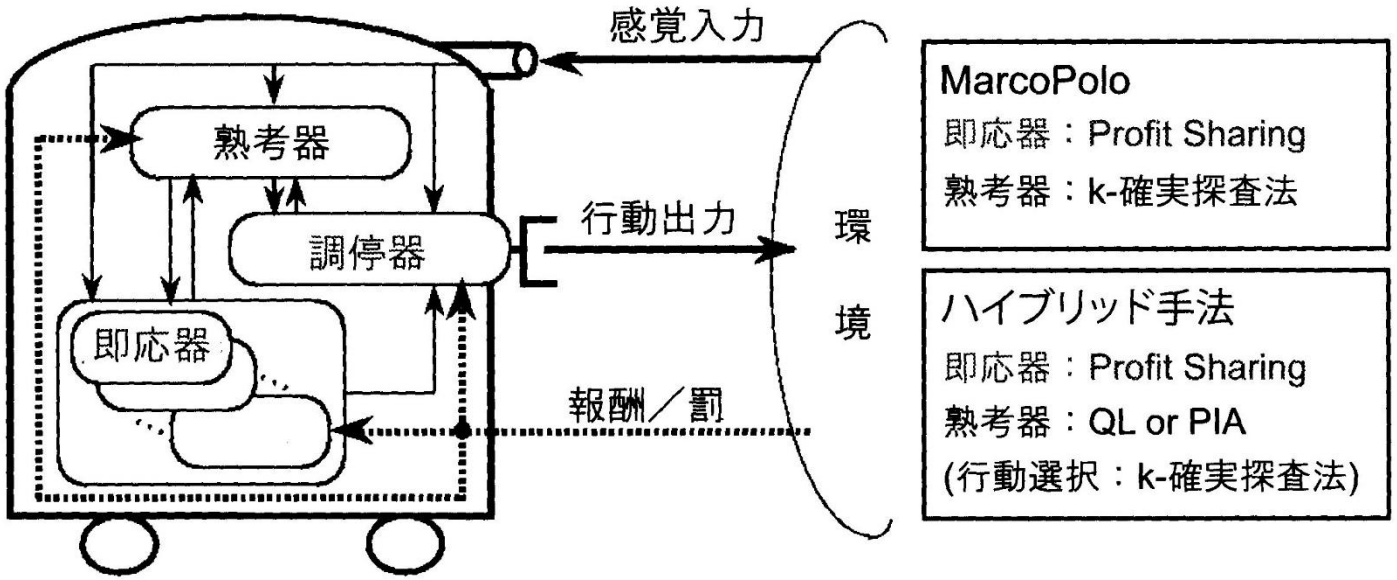


図 2.4.6　ハイブリッド手法

2.4.6 ハイブリッド手法

複数の学習機をハイブリッドさせた手法：ハイブリッド手法

学習：Profit Sharing or Q-learning or 政策反復法

行動選択：k-確実探査法

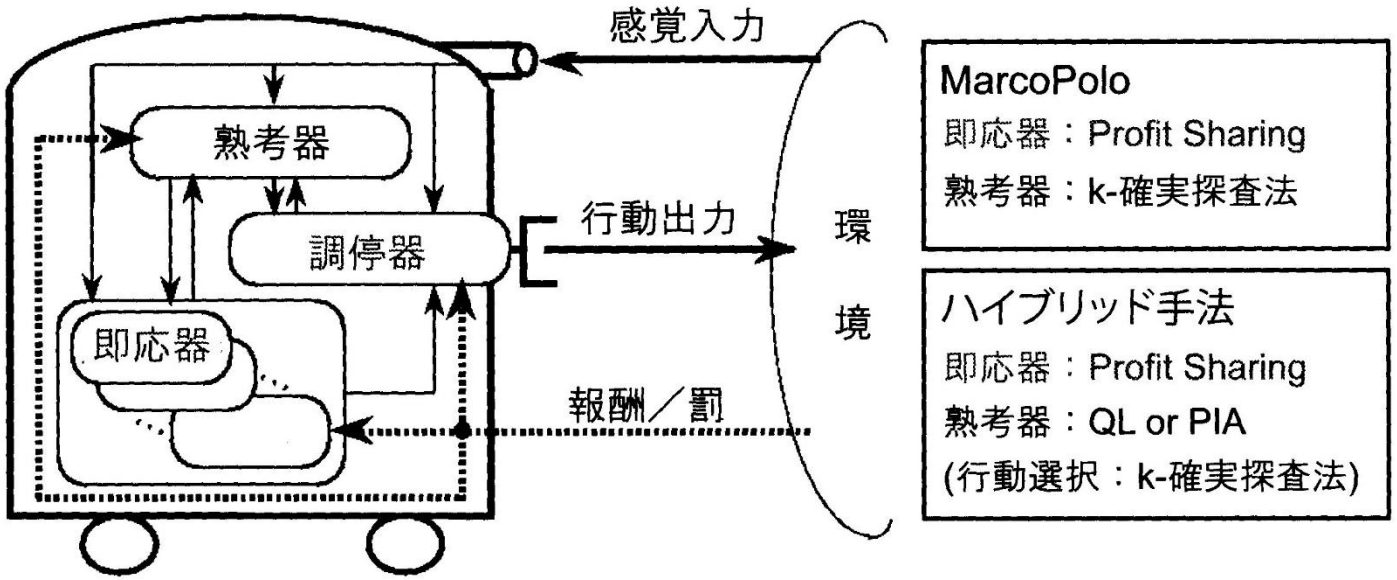


図 2.4.6　ハイブリッド手法

2.4.6 深層学習(DQN)

深層学習 + 強化学習 ⇒ Deep Q-Network (DQN)
膨大な量のデータをディープ化して処理, Q-learningで学習

ネットワークの教師信号は？
⇒Q-learningにより与えられる
⇒試行錯誤をしなければならない！
⇒結局, 計算量は膨大に
⇒XoLで一回一回の体験を貴重なものにすればよい！

2.4.6 XoLをDQNに適用

XoLの学習結果をDQNに適用(DQN with PS)

⇒実際にQ-learningの学習が進んでいなくても,

効率的なネットワークの学習を行い, 計算量が低減

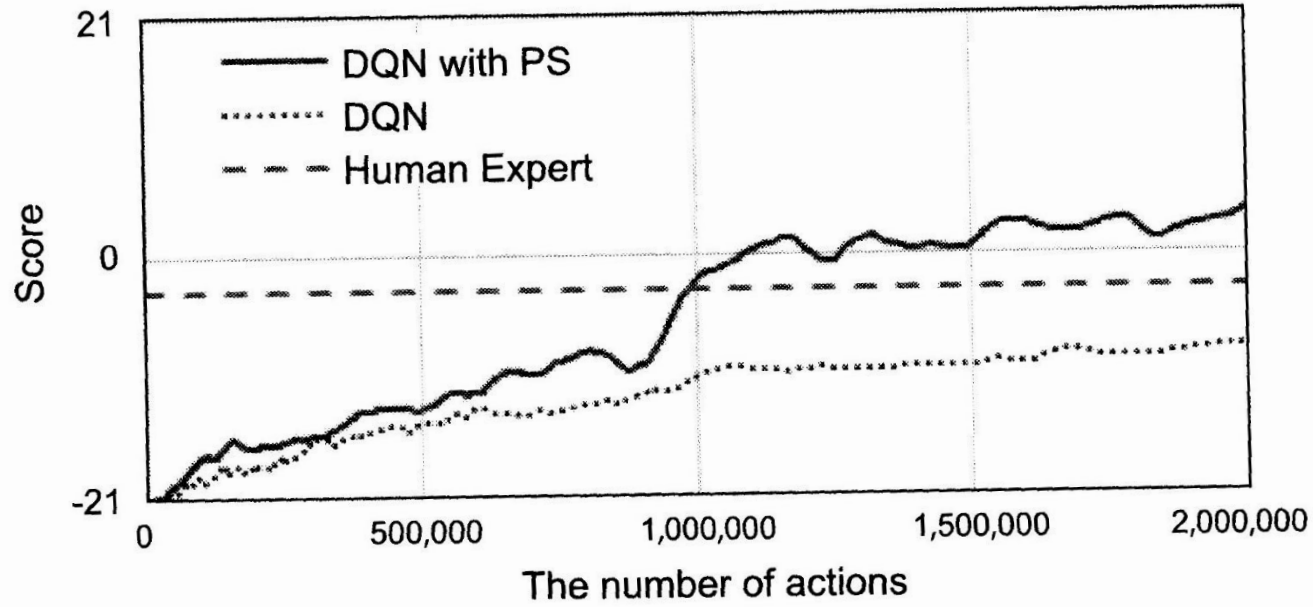


図 2.4.7　Pong における DQN と DQN with PS の比較
横軸は行動選択回数, 縦軸はスコア（最大値 21, 最小値 −21）である. なお, スコアには, 10 回行った実験の平均値をプロットした.

2.4.7 おわりに

2.4.7 まとめ

- XoLはそれ自体で完結している学習手法
- 強化学習とはバックグラウンドが異なる
- 強化学習とは異なり, 合理的な解を追及
- ほかの手法と組み合わせることで, 効率的に良い解を導くことが期待される

  ex)ハイブリッド手法, DQN with PS
- ビッグデータを扱うこれからには必要不可欠！