平成14年度　修士論文

学習すべき課題を自律的に選択する学習システムの研究

電気通信大学 大学院情報システム学研究科
情報ネットワーク学専攻
ヒューマンインターフェース学講座

0151004　内田一樹

指導教官
阪口　　豊
出澤　正徳
長岡　浩司

平成15年2月3日 提出

目 次

第1章　序論

複雑な問題を学習システムに自動的に学習させるのが難しいことは, 理論的にも経験的にも明らかである. そのため, 複雑な問題を学習させる際には, 問題を分割したりサブゴールを与えたりして問題を単純化し, それらを学習させた上で複合的な問題を解決する方法がとられる [1][2][3].

例えば, 実ロボットによる起立運動を獲得するのにあたって, 実ロボットでの試行錯誤を減らすため, まず, 内部モデルを用いた仮想的な試行錯誤を通じた学習を行い, それらの経験をもとに実ロボットでのタスク獲得を可能にしている [4][5].

また, ニューラルネットにおいても, 問題を分割する流れがあり, ネットワークにおける異なる部位が独立した情報処理を実現し, これらの部位が相互に影響を及ぼしあうことのできるモジュール構造ニューラルネットへという流れがある [6].

このように, 複雑な問題を解くために, 問題の分割やサブゴールを与える方法は考えられているが, 人間の場合は, 2 桁の掛け算を解けるようになるためには, まず, 1 桁の掛け算を解くというように, 複雑な問題をいきなり解くのではなく, 簡単な問題から順次解いていく方法もある.

学習システムにおいても, いくつか問題が用意されていて, その中から簡単な問題を自ら選択し, その問題ができるようになったら次に簡単な問題を選択し, 最終的には複雑な問題が解けるようなシステムも考えることができる.

しかし, このような解くべき課題を自動的に選択する学習システムの研究は, 今のところ筆者の知る限りでは, 見当たらない. この場合, まず, どのようにして自動的に選択させるのかが問題になる.

ところで, 人間がある学習をする際, 目にみえてパフォーマンスが向上すれば, 人間はそれに面白さや喜びを感じ, 自ら進んでその課題に取り組むことができる. 逆に, いくら繰り返してもパフォーマンスが向上しないときや常によい結果を出せるときには, やる気を

失う.

このように, 人間の学習過程には, 自分の能力と取り組む課題の難しさが整合しているかどうかによってパフォーマンスの時間変化が決まり, それが学習の動機づけを決定しているという側面がある. このことは結果的に能力を伸ばせるように, 自力で問題を選択する機能を生み出しているといえる.

本研究ではその点に着目し, 自らの能力にあった難しさの問題を選びながら, 問題解決能力を自己組織的に拡張する学習アルゴリズムを提案する.

第2章　強化学習の基礎

序論で述べた通り, 本研究では強化学習のアルゴリズムを用いる. この章では本研究で用いる強化学習について説明する.

2.1　強化学習とは

強化学習 (reinforcement learning) とは, 試行錯誤を通じて環境に適応する学習制御の枠組みである. 基本的には, 学習者 (エージェント) がゴールを達成するために環境の状態 (state) を観測し, その状態観測に応じて行動 (action) を決定する. そして, 学習者の行動により環境の状態に遷移し, その遷移に応じた報酬 (reward) を学習者が得る. このやりとりを繰り返すことにより, 最終的に学習者は最も良い報酬が得られる行動を, 選択するようになる.

2.2　強化学習の構成要素

強化学習の構成要素として,

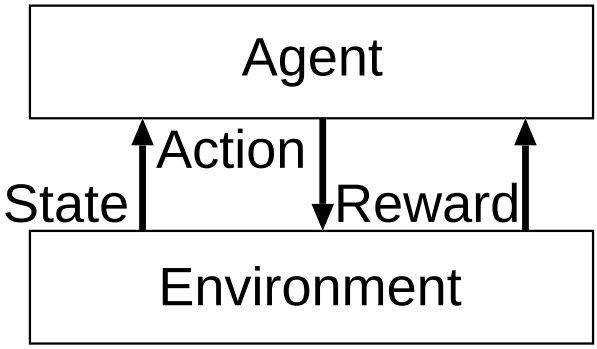

図 2.1: 強化学習

- 方策 (policy)
- 報酬関数 (reward function)
- 価値関数 (value function)
- 環境のモデル (model)

がある.

2.2.1 方策 (policy)

方策とは, ある時点 (状態) において, 学習者のふるまい方を定義するもの, すなわち, 学習者の行動を決めるためのルールである. 状態 $s \in \mathcal{S}$ で行動 $a \in \mathcal{A}(s)$ をとるという方策を $\pi(s, a)$ と表す. この方策は一般的には確率的であり, 方策をきめる方法として, ϵ-greedy 法や, softmax 法等がある.

方策は心理学において, 刺激-応答規則 (stimulus-response rules) あるいは連想 (association) と呼ばれているものに相当する.

2.2.2 報酬関数 (reward function)

報酬関数は強化学習問題において目標を定義するもの, すなわち, ある状態において獲得できる報酬のことである. つまり, 報酬関数は学習者にとって何がよい出来事と悪い出来事を定義している. 強化学習では, 環境から得られる累積報酬を最終的に最大化することを目標として学習を行う.

累積報酬は, 以下の式で与えられる.

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1} \tag{2.1}$$

ここで, T は最終時刻, γ は割引率 (discount factor)$(0 \leq \gamma \leq 1)$ といい, 遠い将来に得られる報酬ほど割り引いて評価するための値である. また, 一般的に報酬関数は確率的である.

2.2.3 価値関数 (value function)

価値関数とは将来にわたって獲得できる報酬の総和の期待値である. つまり, 報酬関数が即時的な意味合いで何がよいのかを示しているのに対し, 価値関数は最終的に何が良いのかを計る関数である.

方策 π のもとで, 状態 s の価値は, 以下のように定式化できる.

状態価値関数 (state-value function)

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\} \tag{2.2}$$

同様に, 方策 π のもとで, 状態 s において行動 a を取ることの価値は, 以下のように定義できる.

行動価値関数 (action-value function)

$$Q^{\pi}(s,a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\} \tag{2.3}$$

2.2.4 環境のモデル (model)

環境のモデルとは, 学習者がある状態においてある行動をとったきに, どの状態にいるのかを予測することである. つまり, モデルによって, 1 個の状態 s と 1 個の行動 a が与えられたとき, 次の状態 s' と報酬の予想が作りだされる. モデルが確率的なものであれば, 可能な次の状態と報酬は複数存在し, その各々が何らかの生起確率を持つ.

強化学習の理論では, 状態 s_t で行動 a_t をとったときに状態 s_{t+1} に遷移する確率が, s_t と a_t だけで決まるというマルコフ性を前提とすることが多い.

状態遷移確率

$$\mathcal{P}^{a}_{ss'} = Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \tag{2.4}$$

期待報酬

$$\mathcal{R}^{a}_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \tag{2.5}$$

2.3 強化学習を解くための手法

2.3.1 Bellman 最適方程式

Bellman 方程式は, 全ての行動に対して, 次におこると期待される全ての状態における価値とその報酬の和を, 生起確率で重みづけしたもので, 以下の式で定義される.

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\{r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s\} \tag{2.6}$$

これを, ちがう形で書くと次のようになる.

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s') \tag{2.7}$$

ここで, 有限マルコフ決定過程 (マルコフ性を満足する強化学習のこと) において,

$$\pi \geq \pi' \quad \text{if and only if} \quad V^{\pi} \geq V^{\pi'} \quad \text{for all} \quad s \in S \tag{2.8}$$

が成り立つ.

したがって, 最適状態関数 V^* は次のように求められる.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \text{for all} \quad s \in S \tag{2.9}$$

同様に, 最適行動関数 Q^* が次のように求められる.

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad \text{for all} \quad s \in S \quad \text{and} \quad a \in A(s) \tag{2.10}$$

以上より, V についての Bellman 最適方程式は

$$\begin{aligned} V^* &= \max_{a \in A(s)} Q^{\pi^*}(s,a) \\ &= \max_{a \in A(s)} E_{\pi^*}\{r_{t+1} \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \max_{a \in A(s)} E_{\pi^*} \sum_{s'} \mathcal{P}^{a}_{ss'} \{\mathcal{R}^{a}_{ss'} + \gamma V^*(s')\} \end{aligned} \tag{2.11}$$

と表せる. また, Q についても,

$$\begin{aligned} Q^* &= E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\} \\ &= \sum_{s'} \mathcal{P}^{a}_{ss'} (\mathcal{R}^{a}_{ss'} + \gamma V^*(s')) \end{aligned} \tag{2.12}$$

として, 求めることができる.

Q^* が与えられれば, 最適な方策 π^* が与えられ,

$$\pi_s^* = \arg\max_{a \in A(s)} Q*(s,a) \tag{2.13}$$

となる. ここで, $\arg\max_a f(a)$ とは $f(a)$ を最大にするような a の値である.

これより, 具体的に Bellman 最適方程式を解くことで, 最適な方策 π を求めることができる. 最適方程式を解く方程式を解く方法として, 動的計画法や Monte Carlo 法, TD 学習等がある. 本研究では, TD 学習を用いているので, 動的計画法と Monte Carlo 法については軽く触れるだけにする. 動的計画法とは, 環境に関する完全知識を仮定して最適な価値関数を求める方法である. 次に Monte Carlo 法とは, 動的計画法とは逆に知識を仮定せず, 実際に行動して, 終端状態に達するまでに得られた報酬をもとに価値関数を推定する方法である.

2.4 TD 学習

TD 学習 (Temporal-Difference Learning) は, Monte Calro 法と動的計画法の中間的な方法と考えることができる. 環境の知識を仮定せず, 経験を通じて価値関数を求めるという点で Monte Carlo 法に, また, 最終結果を待たずに, 一度の行動での経験に基づいて価値関数を修正するという点で動的計画法に似ている.

2.4.1 価値関数の更新規則

TD 法では時刻 $t+1$ で直ちに目標値を作り, 観測した報酬 r_{t+1} と価値関数 $V(s_{t+1})$ を使って価値関数 $V(s_t))$ の更新を行う. 最も単純な TD 法は TD(0) と呼ばれ, 以下のように更新を行なう.

$$\begin{aligned} V(s_t) &\leftarrow V(s_t) + \alpha\delta_t \\ &= V(s_t) + \alpha\{r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\} \end{aligned} \tag{2.14}$$

ここで, α は定数で $0 < \alpha \leq 1$ である. また, $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ は TD 誤差と呼ばれている.

2.4.2 Sarsa(On-line TD learning)

行動価値関数を用いた最も自然なアルゴリズムである. 次の状態の価値関数を計算するとき, 次の時刻で実際に選んだ行動を用いる.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\} \tag{2.15}$$

2.4.3 Q学習 (Off-line TD learning)

次の状態の価値関数を計算するとき, その時刻で最大のQ値を与える行動を用いる.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\{r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\} \tag{2.16}$$

2.4.4 TD(λ) 学習

TD(λ) はTD学習とMonte Carlo法を統合した考え方であり, 過去に経験した状態に関する価値関数も更新するという考え方である. 具体的には, 履歴を管理するための変数 e を設け, その値を0として終端状態になるまで次のような更新を繰り返す.

$$e(s_t) \leftarrow e(s_t) + 1 \tag{2.17}$$

$$V(s) \leftarrow V(s) + \alpha\delta e(s) \quad \text{for all} \quad s \in S \tag{2.18}$$

$$e(s) \leftarrow \gamma\lambda e(s) \quad \text{for all} \quad s \in S \tag{2.19}$$

$\lambda = 0$ のときは通常のTD学習であり, $\lambda = 1$ のときはMonte Carlo法になる.

2.4.5 Q(λ) 学習

TD学習の更新則のQ学習と同様に, TD(λ) の更新則で, TD誤差 δ を次のように定義する.

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \tag{2.20}$$

2.4.6 行動選択

- ϵ-greedy法

  ϵ の割合でランダムに選び, そのほかの場合は, 価値関数が最大に行動を選ぶ. すな

わち,

$$\begin{aligned}
\pi_t(s,a) &= Pr\{a_t = a \mid s_t = s\} \\
&= \begin{cases} \dfrac{\epsilon}{|A(s)|} & (a \notin A'(s)) \\ \dfrac{1-\epsilon}{|A'(s)|} + \dfrac{\epsilon}{|A(s)|} & (a \in A'(s)) \end{cases} \qquad (2.21) \\
A'(s) &= \{a | Q(s,a) = \max_b Q(s,b)\} \qquad (2.22)
\end{aligned}$$

である.

- softmax 法

  以下の確率に従って行動を選ぶ.

$$\begin{aligned}
\pi_t(s,a) &= Pr\{a_t = a \mid s_t = s\} = \frac{\exp(q(s,a))}{\sum_b \exp(q(s,b))} \qquad (2.23) \\
q(s,a) &= \frac{Q(s,a)}{T} \qquad (2.24)
\end{aligned}$$

  ただし, T は温度パラメータといい, この値が大きいほどランダム性が高くなる. 上記の式から, softmax 法では $Q(s,a)$ の値が大きい a ほど選ぶ確率が高いことがわかる.

第3章　自律的に選択する学習システム

3.1　基本的な考え方

人工的な学習システムにおいて, 学習者の能力に比べて問題が難しすぎると学習が阻害されることになる. したがって, 学習を効率よく進めるためには, 学習者の能力と課題の難しさが整合していることが重要である. 例えば, 線形分離不可能な問題をパーセプトロンで学習させるとき, その問題を直接学習させるよりも, あらかじめその問題の解決に必要な中間層素子を成長させた上で学習させる方が, 結果的に少ない回数で学習が終了する [7].

このことは, 人間においても同じことが言える. 例えば, 足し算を学習したばかりで 1 桁の掛け算を習得していない小学生が, 3 桁の掛け算を学習することが困難であることは明らかであるように, 自分の能力に比べて問題が難しすぎると学習が阻害される. この小学生の例では, 3 桁の掛け算の方法だけを教わるより, まず 1 桁の掛け算を教わってから, 3 桁の掛け算を教わる方が学習が早いであろう.

このとき, この小学生にとって, 3 桁の掛け算だけを学習しようとした場合, 難しすぎて喜びや面白さを感じることができないであろう (もちろん, 難しいからこそやりがいを感じ面白いと思う人もいるかもしれない). また, この小学生に足し算ばかりをやらせた場合, この小学生は足し算に飽きてしまうだろう. 逆に, この小学生が 1 桁の掛け算を学習し, 自分の力で解けるようになって来たときに, 喜びや面白さを感じるであろう.

このように人間の場合は, 課題が難しすぎて全く解けない場合や, 課題が簡単すぎて常によい結果が出せるときにはやる気を失い, 目に見えてパフォーマンスが向上すれば, 面白さや喜びを感じ, 自ら進んでその課題に取り組むことができるといってよい. この課題の成績と面白さの関係を簡単にグラフ化したものを図 3.1 に示す.

ここで, このグラフに着目すると, 人間が「面白さ」を感じているときが, パフォーマンスが向上しているときであり, その課題に取り組むと習得が最もよいときであるといえる.

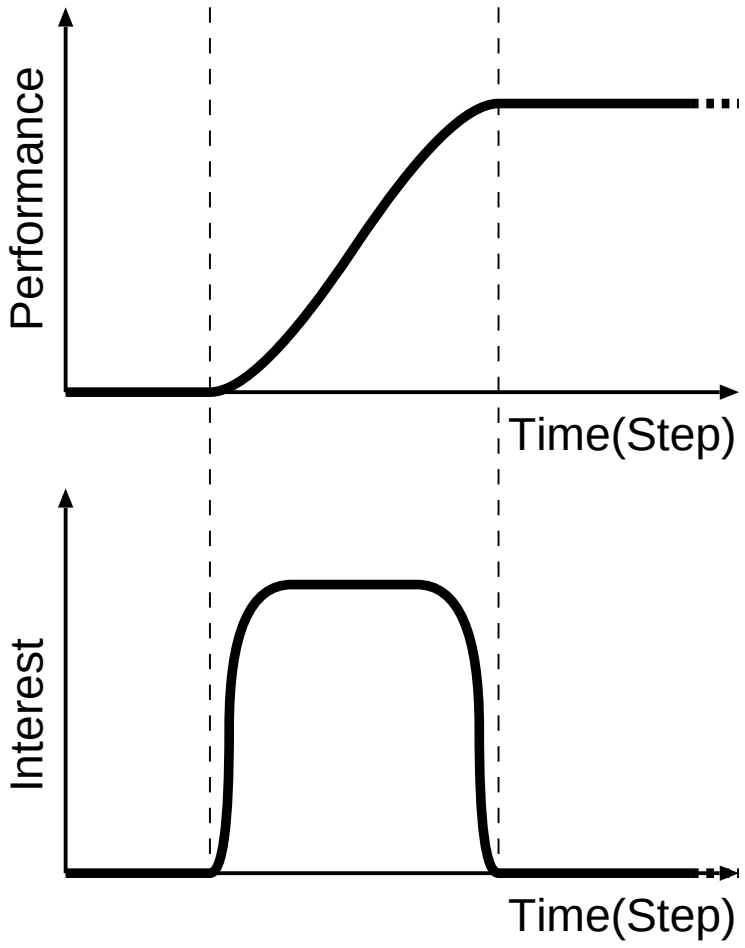

図 3.1: 課題の成績と「面白さ」の関係

つまり, この「面白さ」が学習するべき課題を選択する上でのものさしとなっていることがいえる.

この考えを人工的な学習システムにも応用することができる. つまり, 学習者と課題が整合しているかどうかは, 学習曲線の時間変化をみることによって判定できる. 学習者と課題が整合していれば, 学習は順調に進むために学習曲線は一定の傾きで増加するが, 整合していない場合は, 学習が進まないために学習曲線はほぼ水平に保たれる. よって, 図 3.1 からわかるように, 学習曲線の時間変化が人間の「面白さ」に相当する.

以上の考えに基づき, 本研究では, 学習者と課題の整合性をはかるものさしとして学習曲線の時間変化を用いる. そして, このものさしを使って, 学習者が自分の能力にあった課題を選択することを考える.

具体的には, 学習曲線の変化率を「報酬」, 自分の解くべき問題の選択を「行動」とする強化学習系を考える. なお, この強化学習には「状態」に対応する概念がない (学習者のその時点での能力を状態とみなすことができるが, 状態と行動を結びつける方策を学習する必要がないので, 状態を明示的に考える必要はない).

学習者は, 難易度の異なる複数の問題のクラスの中から一つ選んで, 一定期間そのクラスに属する問題の学習に取り組む. その期間に, パフォーマンスが向上して報酬が得られれば, そのクラスに対する価値関数の値を高める. 逆に, パフォーマンスが向上しないときや既に飽和してしまっているときは, 報酬が得られないので, そのクラスに対する価値関数の値を低下させる. 学習者が価値関数の大きさを反映して問題のクラスを選択すれば, 学習者は大きな報酬が得られる問題を優先して選ぶことになるので, そのクラスについて集中的に学習を行うことができる.

一方, 図 3.2 からわかるように, 易しい問題が安定して解けるようになるとパフォーマンスの時間変化がなくなるため, 学習者は他のクラスの問題, すなわち, より難しい問題を選択する比重を高めることになる. したがって, 易しい問題を解く過程で体得した能力が難しい問題を解くときに利用できる場合には, 学習者は自然に自分の能力を発展させていくことができる.

つまり, 本研究では, 自分の能力と整合していない難しい問題は最初に解かず, 簡単な問題から順次解いて行くが, それは, 環境から解くべき課題を与えられるのではなく, 学習曲線の時間変化を用いることによって, 自律的に選択するようになる学習システムを提案

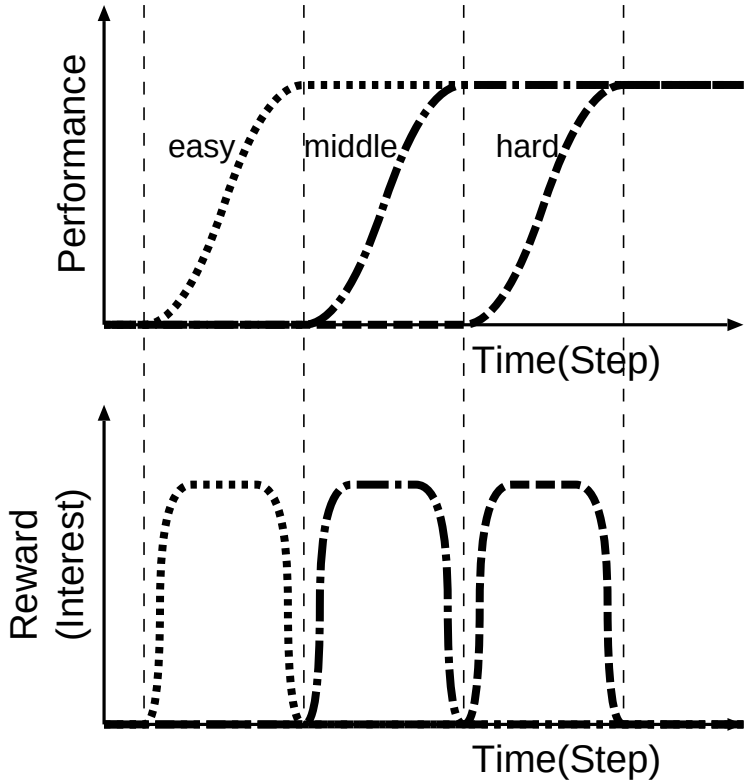

図 3.2: 学習曲線の時間変化

する.

3.2 学習システムの構成

図 3.3 に提案する学習システムの一般的な構成を示す. 図に示したように, このシステムは二つの強化学習系から構成されている. 一方は, 問題を解決するための学習系で, 環境から得られた入力 (問題) に応じて適切な行動 (答え) を選択して出力する. この強化学習系は, 問題を解決するための学習系で, 解くべき問題のクラスを出力する. 前節で述べたように, この学習系は, パフォーマンス向上率を報酬として学習を行う.

前者のアルゴリズムには, 解くべき問題の内容に合わせて前章で述べたような通常の強化学習を用いればよい. また, 問題解決系に強化学習を用いる他教師あり学習等を用いてもよい. 以下では, 本研究のポイントである後者のアルゴリズムについて述べる.

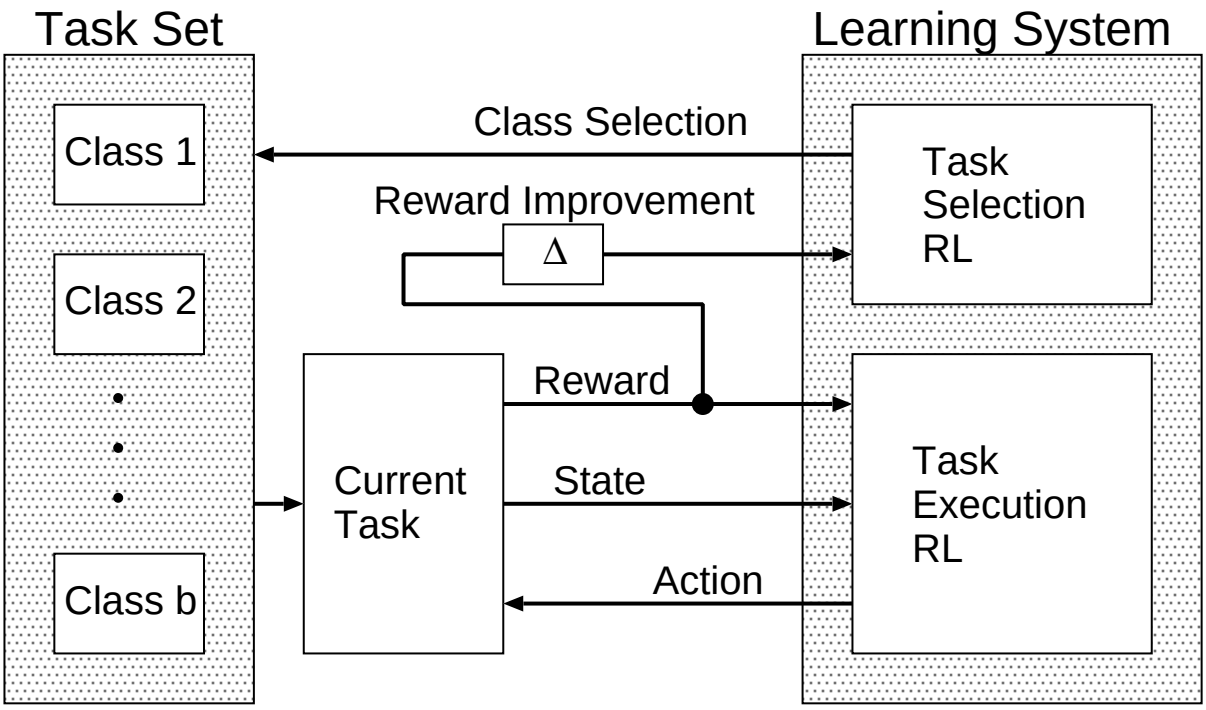

図 3.3: モデルの一般的構成

3.3 問題選択系強化学習アルゴリズム

まず, 問題のクラス b に対して価値関数 $Q(b)$ を定義する. 問題選択系が受け取る報酬 D は, 問題を解くごとに問題解決系が得た報酬 (つまり, パフォーマンス) R の時間変化によって決まる. ただし, 報酬の値は問題ごとに細かく変動すると考えられるので, ここでは, 同じクラスの問題に $2N$ 回続けて取り組んだときに (以後これを学習ブロックと呼ぶ), その後半 N 回に得られた平均報酬と前半 N 回に得られた平均報酬の差を用いる (この平均報酬の差を用いることの他に, 最小二乗法を使って相関係数を求めそれを報酬とする方法もある).

以上の設定の下で, 問題選択系の行動決定則は以下の式に従って定められる. まず, 解くべき問題のクラスは, 各クラスに与えられた価値関数に基づき前章で説明した Softmax 法により決定される.

$$P(b) = \frac{\exp(\frac{Q(b)}{T})}{\sum_{b'} \exp(\frac{Q(b')}{T})} \tag{3.1}$$

一方, 価値関数は以下の式に従って更新する.

$$\Delta Q(b) = c(D - Q(b)) \tag{3.2}$$

c は学習定数である. これらの規則は学習ブロックごとに適用する.

3.4 問題選択系の報酬を TD 誤差とする場合

問題解決系の報酬の変化率を報酬とする以外のものとして, 問題解決系の TD 誤差を選択系の報酬とする方法が考えられる. TD 誤差は, 実際にもらえる報酬と期待している報酬 (価値関数) との差であるといえる. したがって TD 誤差を選択系の報酬とすることは, 予想以上の報酬を得られたときには, その課題に興味や喜びを感じ, その課題に取り組みたくなり, 逆に, 予想以下の報酬や予想通りの報酬だと, その課題には興味を持たなくなり, 他の課題に取り組みたくなることだと言える. また, また常に TD 誤差が大きいときは, 学習者は学習しているということである. したがって, 問題解決系の報酬の変化率を選択系の報酬としているときと同様に, 課題に対してパフォーマンスが向上しているときが, 学習者が最もその課題に取り組みたくなるときだと予想できる.

学習ブロックで得られる TD 誤差を $\{\delta_t | 1 \leq t \leq T\}$ (学習ブロック中 T は TD 誤差を得る回数) とすると, 選択系の報酬 D は以下の式で表せる.

- ステップ数で平均する場合

$$D = \frac{\sum_{1 \leq t \leq T} |\delta_t|}{T} \tag{3.3}$$

- ステップ数で平均しない場合 (h は課題同士共通の定数)

$$D = \frac{\sum_{1 \leq t \leq T} |\delta_t|}{h} \tag{3.4}$$

ステップ数で平均するということは 1 ステップの TD 誤差を比較するということを重視した方法であり. 平均しない場合は, 終端状態の TD 誤差と途中の TD 誤差は異なるので, 単純に平均してはならないとうことを重視した方法である.

次章では, 問題解決系の報酬の変化率を選択系の報酬とする場合と, TD 誤差を報酬とする場合を比較し, さらに, TD 誤差をステップ数で平均する場合としない場合を比較し検討する.

第4章　例題1 二分木(1)

4.1　例題の意義と目的

本章では, 本研究の提案する学習システムがうまく機能するかどうかを調べるために具体的な例題を用いて検討する. ここでは, 議論が明確になることに重点をおき, 状態遷移の構造が単純な完全二分木構造で記述できる問題を例題としてとりあげる.

本章で扱う二分木問題は, 二分木のルートからパスを辿って正解のあるリーフを目指すという課題である. 正解のリーフが一つしかない条件では, 二分木の層の深さに応じて探索空間は大きくなるので, 層の数を変えることで, 課題の難しさが制御できることが予想できる.

例えば, 3 層の二分木 (図 4.1) ではリーフの数に対する正解の割合が 1/4 であるのに対し, 5 層の二分木 (図 4.2) では, その割合が 1/16 で, その分, 解を見つけるのに必要な試行数が多くなる. したがって, 5 層ある二分木の方が難しい課題であると予想できる.

以下, (1) 層の深さが異なる課題を個別に学習させる実験を行い, 層が深いほど難しい課題であるかどうかを検証し, 次に (2) 複数ある課題群の中からランダムに選択した場合と, 本学習システムを用いて自律的に選択した場合について実験を行ない, システムの学習速度を比較する.

また, 複数の課題に対する, 行動価値関数を共有する方法と課題間の価値関数を複写させる方法について, それぞれに対し (1), (2) のような実験を行い, それらの性質を検討する.

さらに, 選択系の報酬をパフォーマンスの学習曲線の時間変化とした場合と TD 誤差とした場合の比較も行う.

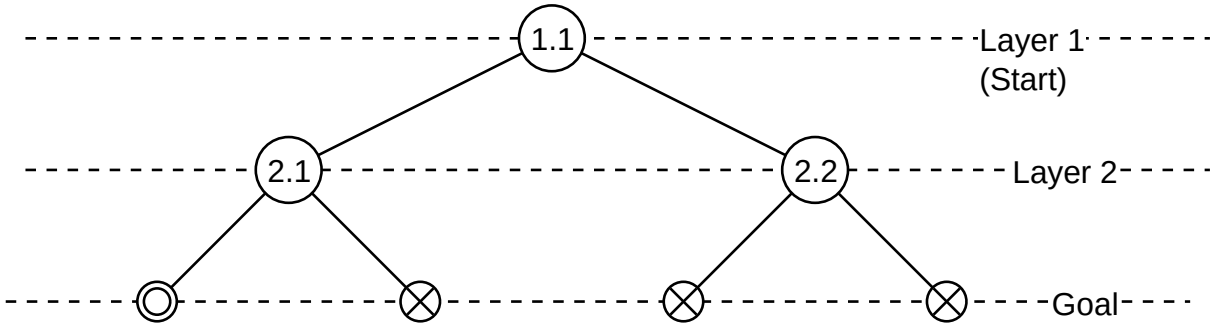

図 4.1: Class 2 の課題

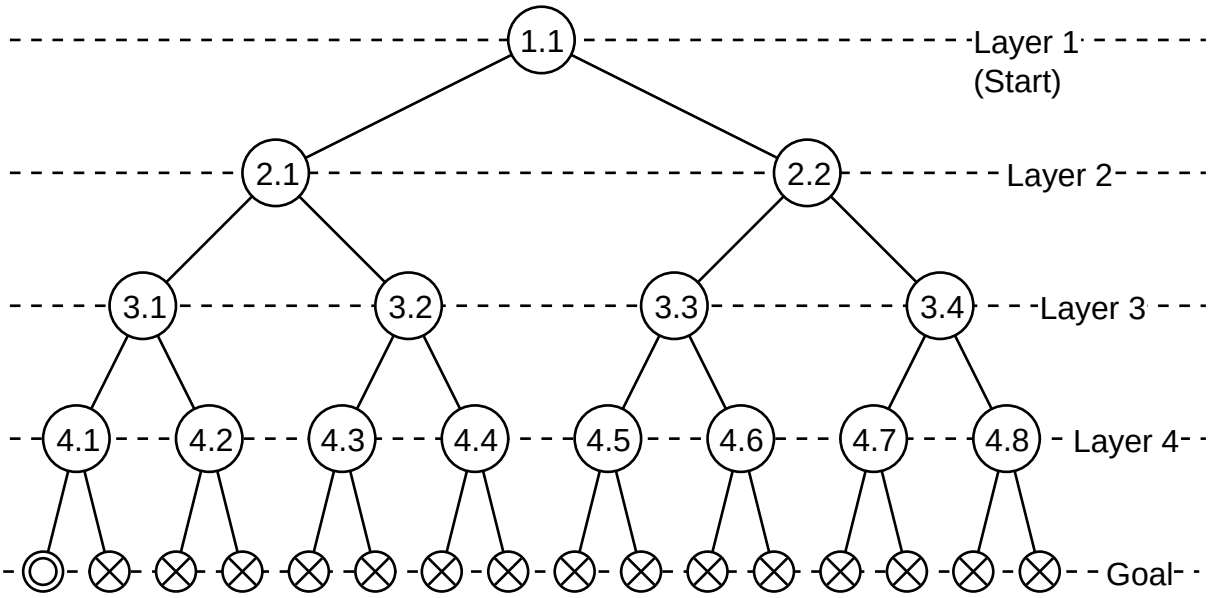

図 4.2: 二分木を用いた問題

4.2 例題の内容と実験条件

4.2.1 実験条件

用いる二分木は図4.2のように完全二分木であり, かつ, 第 N 層のノード数が 2^{N-1} であるようになっている.

この課題では, ゴールとなるリーフの中で報酬が得られるリーフは一つであり, すなわち, 正解につながるパスは一本ということになる. さらに, 学習者はどのパスを辿ると報酬を得られるのかについての知識を事前に持っていない.

ここで, 深さ $N+1$ ある場合その課題を Class N と呼ぶことにすると, 前節で述べたよ

うに, 課題 Class N は, ルートを第 1 層, リーフを第 $N+1$ 層とする完全二分木を場とし, 学習者は二分木の第 1 層のノード 1.1(ルート) を出発点とする.

問題解決系は, 各ノードにおいて左右の 2 方向のどちらに移動するかを決めるための行動を出力する. すなわち, 各ノードを状態 s とし, そのノードからの移動方向を行動 a とする学習系を設定した. また, その学習系に対して, 行動価値関数 $Q(s,a)$(初期値は 0) を設定した. また, 行動選択には Softmax 法 (温度パラメータの値は 0.1) を用いた. 例えば, 状態 s がノード 1.1 である場合,

$$Pr\{a_t = \text{left} | s_t = 1.1\} = \frac{\exp(\frac{Q(1.1,\text{left})}{T})}{\exp(\frac{Q(1.1,\text{left})}{T}) + \exp(\frac{Q(1.1,\text{right})}{T})} \tag{4.1}$$

の確率で左に移動, すなわち, ノード 1.1 の下の層であるノード 2.1 に状態遷移し,

$$Pr\{a_t = \text{right} | s_t = 1.1\} = \frac{\exp(\frac{Q(1.1,\text{right})}{T})}{\exp(\frac{Q(1.1,\text{left})}{T}) + \exp(\frac{Q(1.1,\text{right})}{T})} \tag{4.2}$$

の確率で右に移動, すなわち, ノード 2.2 に状態遷移する.

さらに, 学習者は左右の選択を繰り返すことにより, 次々に下の層に状態遷移し, 最終的にゴールであるリーフまで辿り着く. 各リーフには〇か×のどちらかが印してあり, もし〇であれば正解として 1 の報酬を与え, ×であれば不正解として 0 の報酬を与えた. ただし, 〇は一番左のリーフにのみ印してあるものとした (このように設定しても一般性は失われない). 行動価値関数の学習には Q 学習をアルゴリズムを用い, 学習係数 α_1 は 0.005, 割引率は 1 とした. すなわち, 状態 s から移動後の状態 s' がゴール (リーフ) でなければ,

$$Q(s,a) \leftarrow Q(s,a) + \alpha_1(\max_b Q(s',b) - Q(s,a)) \tag{4.3}$$

のように更新し, 移動後の状態がゴールならば,

$$Q(s,a) \leftarrow Q(s,a) + \alpha_1(r - Q(s,a)) \tag{4.4}$$

のように更新した (ただし, 移動後の状態が〇ならば $r=1$, ×ならば $r=0$ である).

学習系が自ら課題を選択する場合は, 各学習ブロックの試行数を $N=20$ とし, 問題解決系の報酬差を選択系の報酬とした. また, 問題選択系の行動選択則の温度パラメータは 0.04, 価値関数の学習をする際の学習係数を 0.2 とした.

4.2.2 課題同士の相互作用

このような難易度の異なる課題 (難易度の異なる課題の集合を課題群と呼ぶことにする) を取り組むことによって得た知識を他の課題を取り組む際に使えるようにしなければ, 本研究の学習システムは意味をなさない. その知識を使える方法として, 複数の課題に対する一つの価値関数を共有する方法と, 価値関数を課題毎に一つずつ用意して, 学習がほぼ完了した課題で得た知識 (価値関数) を学習が進んでいない課題の価値関数に複写する方法が考えられる. 具体的な価値関数の複写の方法としては, あるクラス a で学習ブロック中に 9 割以上〇に辿り着くようになったら, 2 割以下しか〇に辿り着けないクラス b の価値関数をクラス a の価値関数と同じ値にする. ただし, 価値関数を複写し続けると, 複写されるクラスの学習は進まないので, 複写を行なう回数はクラス毎に 1 回のみとする.

例えば, Class 2 と Class 4 で共通の行動価値関数を使用すると, 図 4.3 の $Q(1.1, \mathrm{left})$, $Q(1.1, \mathrm{right})$, $\cdots$, $Q(2.2, \mathrm{right})$ の第 3 層への行動価値関数が, Class 2 と Class 4 の共通となる. 仮に Class 2 で続けて学習した後に Class 4 を学習するとすると, Class 4 を学習しはじめる際に, 左端へ移動すればよいという知識が第 3 層まではすでにできあがっている. したがって, Class 4 で正解に辿り着きやすくなると予想できる.

4.3 結果

4.3.1 個別に学習させた場合

まず, 図 4.4 は問題選択系を使用せずに, Class 2, 4, 6, 8, 10 を各 20 回ずつ個別に学習させた場合の実験結果の平均である (以降, 断りがない限り, 実験結果の図はすべて 20 回の平均とする). 図の横軸は学習ブロック数, 縦軸は学習ブロック当たりに得られた報酬の合計を表している. この図から, 予想通りクラスが低いほど学習曲線の立ち上がりが早く, 層が深くなるほど課題を解くことが困難であることがわかる.

4.3.2 価値関数を共有し, 問題をランダムに選択した場合

図 4.5 と図 4.6 はそれぞれ価値関数を共有させたときの Class 2, 4, 6, 8, 10 の課題群と Class 4, 8, 12, 16, 20 の課題群の中からランダムに選択した場合の結果である実験中に選

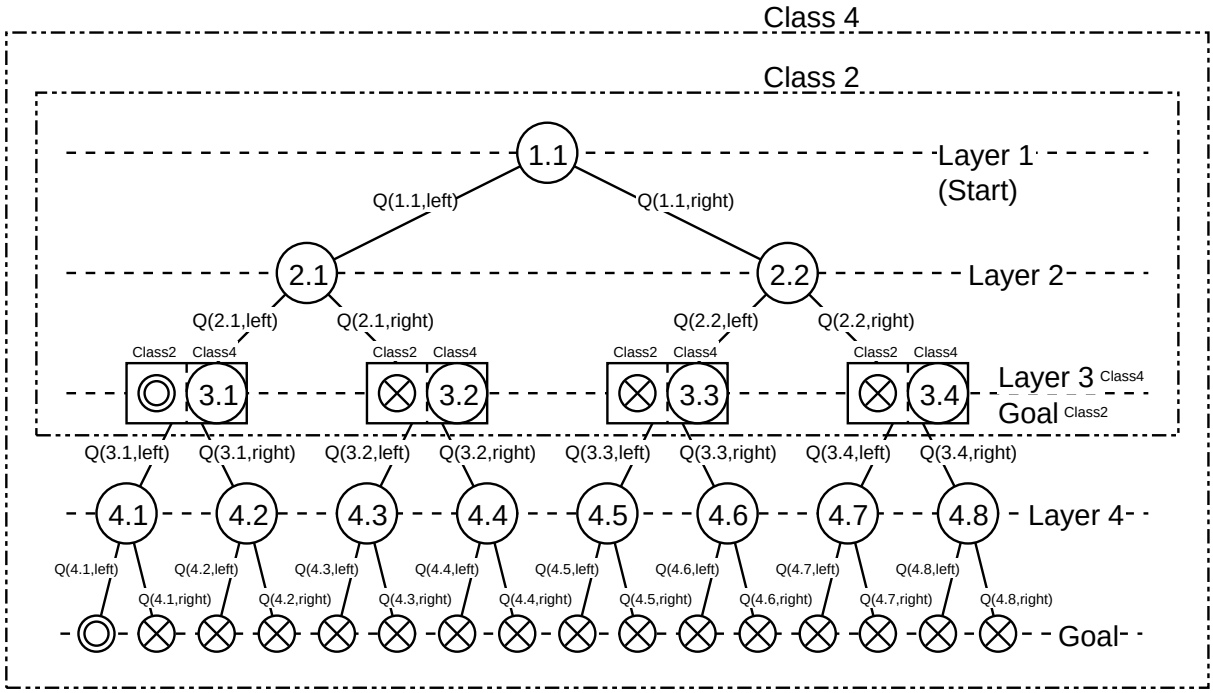

図 4.3: Class 2 と Class 4

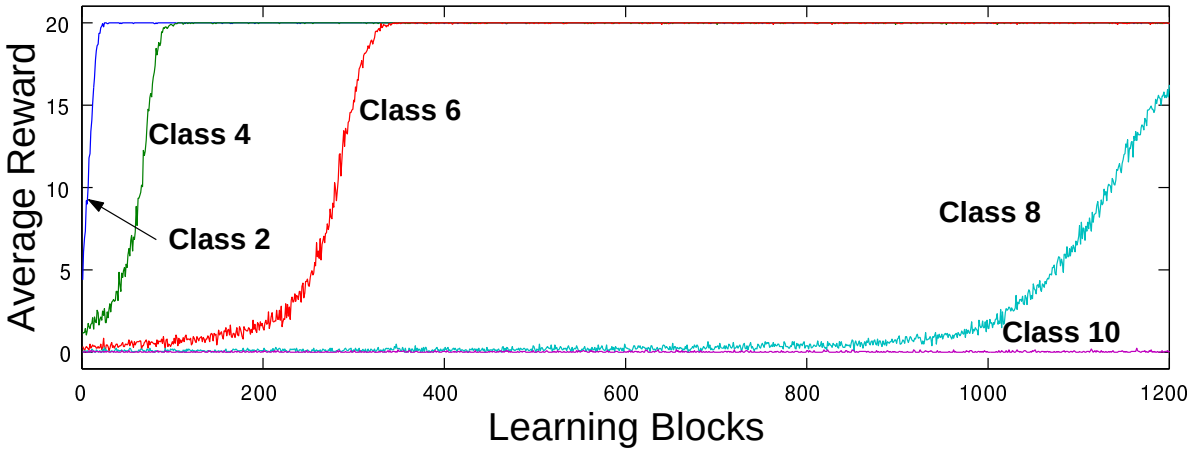

図 4.4: Class 2, 4, 6, 8, 10 を個別に学習させた場合の結果

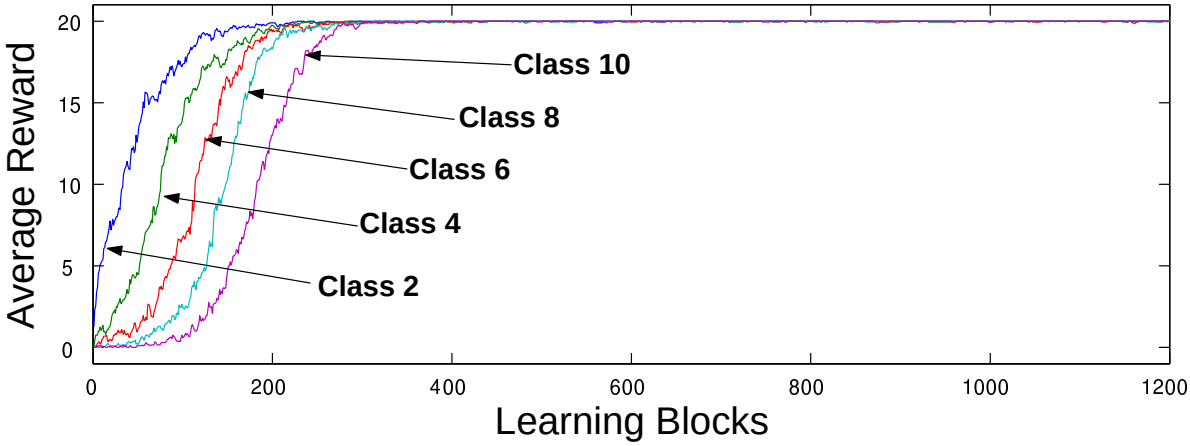

図 4.5: Class 2, 4, 6, 8, 10 の課題群の中からランダムに選択した場合の結果

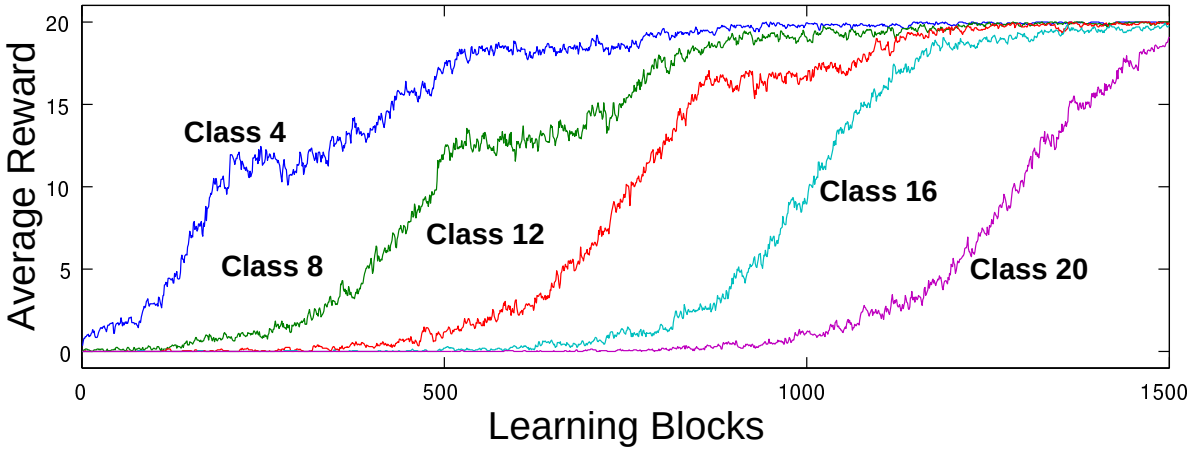

図 4.6: Class 4, 8, 12, 16, 20 の課題群の中からランダムに選択した場合の結果

ばれる課題はブロックごとに変化するが, ここでは, 課題ごとにその課題を使ったブロックの成績だけを抜きだしてグラフを描いている. これらの図と個別に学習させたときの結果である図 4.4 を比較すると, 易しい課題を混ぜることで学習が速く進むことがわかる.

4.3.3 価値関数を共有し, 問題をランダムに選択した場合の考察

個別に学習させたときより, ランダムに学習させた方が学習が速く進む理由を以下に述べる. 説明を簡単にするために, 9 層構造の二分木で Class 4 と Class 8 で比較することによって説明する.

まず, 個別に学習する場合について考える. 学習を始めるときは, 各パスの価値関数の値は 0 であるから, 学習者はパス上をランダムに行動するので, Class 4 で試行すると正解に辿り着ける確率は $1/2^4$ であり, Class 8 で試行すると正解に辿り着ける確率は $1/2^8$ である. したがって, 不正解の場合の報酬は 0 であり, 価値関数の値は 0 なので, 不正解になると価値関数は更新されず, 正解に辿り着ける確率の低い Class 8 より確率の高い Class 4 の方が早く更新される.

ここで, 仮に Class 8 第 5 層までの一番左側のパス上の価値関数 (正解に通じるパス上の価値関数) が, 学習者が常に正解のパスを選択するぐらいの値だとすると (他のパス上の価値関数の値はすべて 0), 学習者は第 5 層までは正解のパスを移動する. その第 5 層からの正解に辿りつく確率は $1/2^4$ であり, 価値関数の値がすべて 0 のときより, 大幅に正解に辿り着く確率があがる.

ランダムに選択する場合は, 上述の仮定ようなことが学習中に起こる.

Class 8 と比べ, Class 4 の方が正解に辿り着くので, 第 5 層までの価値関数も先に更新される. Class 4 の学習曲線が飽和すると, 上述の仮定のような状態になり, Class 8 もすぐに学習できるようになるので, 個別に学習させるより, 問題を同時に学習させる方すなわち, 問題をランダムに選択させる方が学習が速くなる.

4.3.4 価値関数を共有し, 問題を自律的に選択した場合

図 4.7 と図 4.8 はそれぞれ Class 2, 4, 6, 8, 10 の課題群, Class 4, 8, 12, 16, 20 の課題群の中から選択系を使用して自律的に選択した場合の結果である. 上図は学習曲線, 下図は

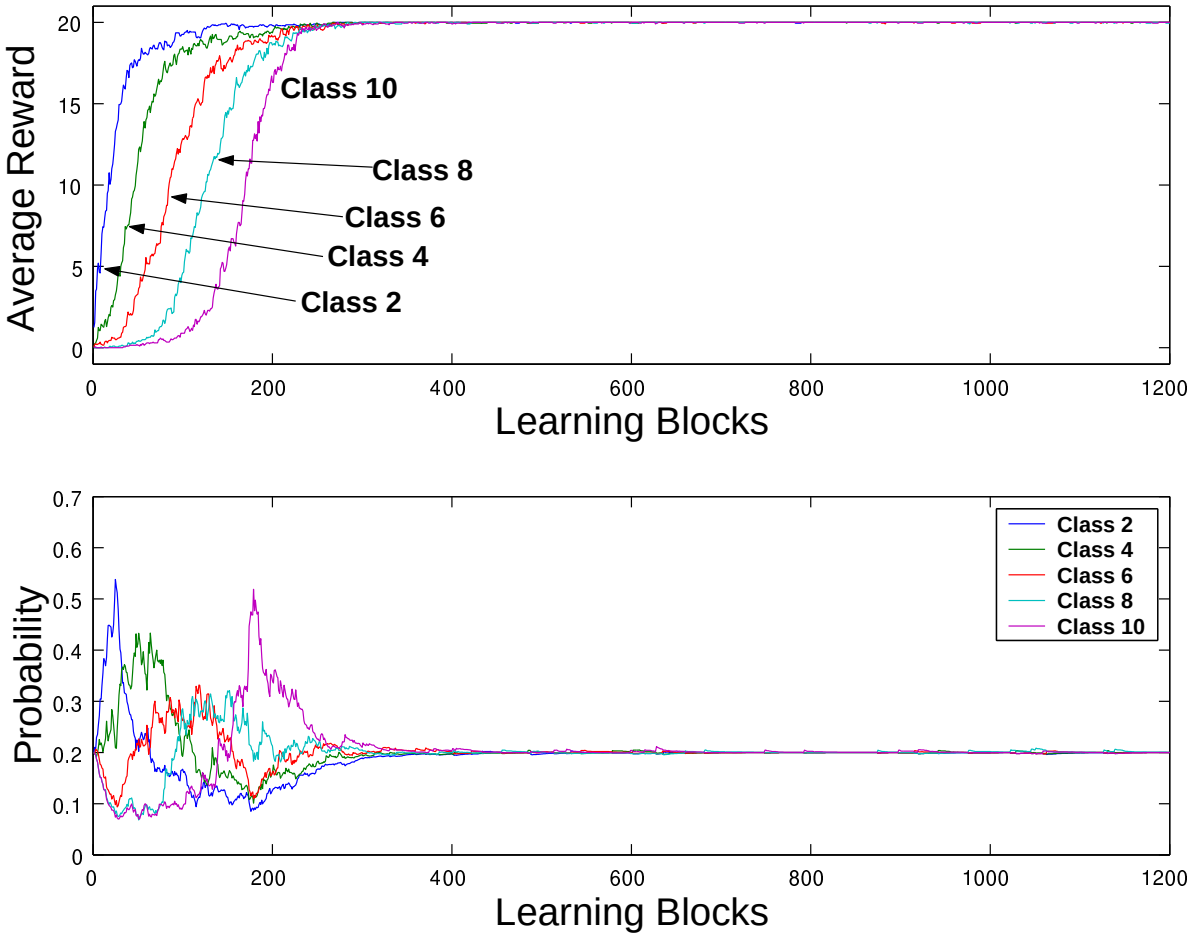

図 4.7: Class 2, 4, 6, 8, 10 の課題群の中から自律的に選択した場合の結果

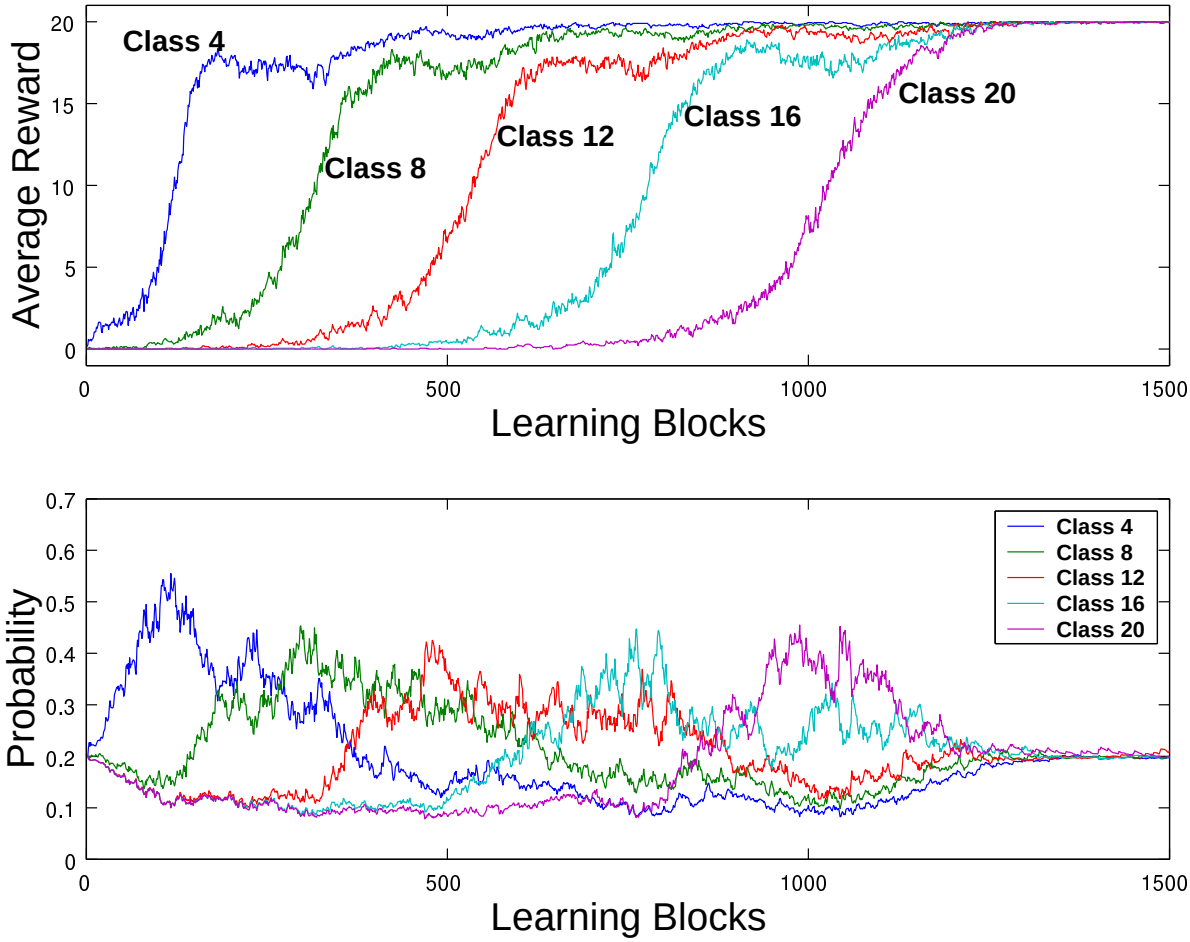

図 4.8: Class 4, 8, 12, 16, 20 の課題群の中から自律的に選択した場合の結果

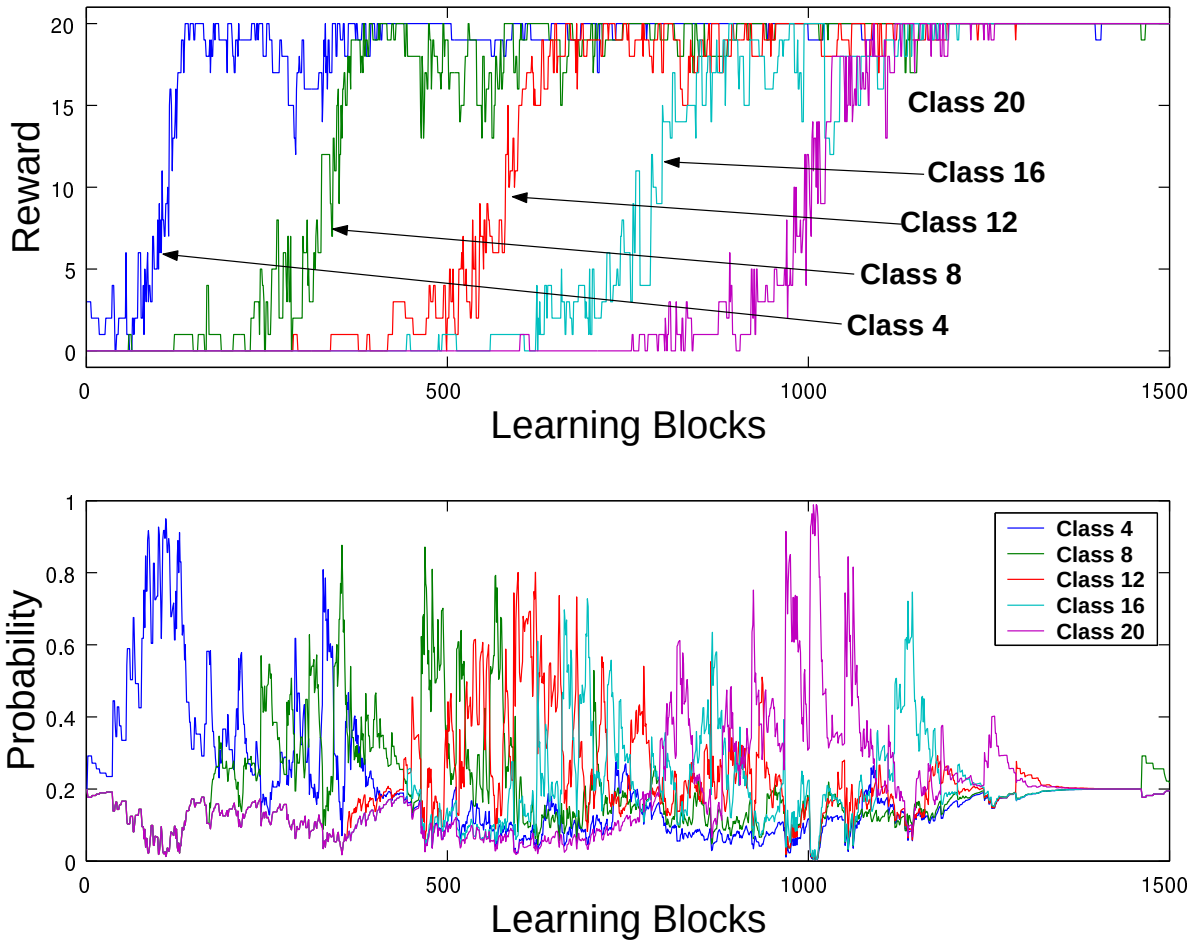

図 4.9: 価値関数を共有し Class 4, 8, 12, 16, 20 の課題群の中から自律的に選択した場合の結果 (1 回の試行)

課題を選択する確率を表しており, 横軸は学習ブロック数, 縦軸はクラスを選ぶ確率を表している.

まず, 学習曲線と選択確率曲線を対比させると, ある課題の学習曲線が立ち上がっているときに, その課題を選択する確率が大きいことがわかる.

次に, Class 2, 4, 6, 8, 10 において, 自律的に選択した場合である図 4.7 とランダムに選択した場合である図 4.5 を比較すると, 学習の速さの差はほとんどないが, Class 4, 8, 12, 16, 20 において同様に図 4.8 と図 4.6 を比較すると, 自律的に選択した場合の方が学習が速いことがわかり, 問題選択系が機能している.

また, 図 4.9 は試行回数は 20 回ではなく 1 回の実験結果であり, Class 4 の学習曲線が飽和して, Class 8 の学習曲線が立ち上がり始めたときに, Class 4 の学習曲線は下がってし

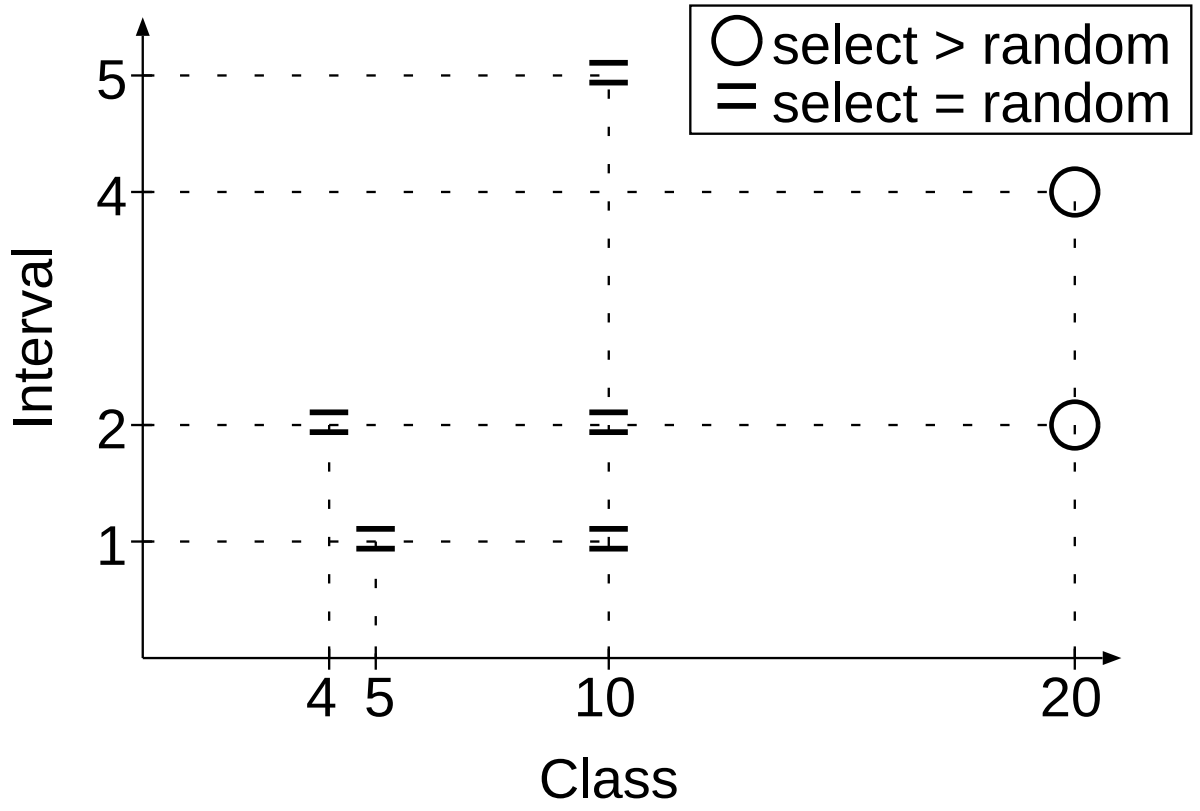

図 4.10: 選択系とランダムの比較

まっている. この原因は 4.3.9 節の考察で述べる.

4.3.5 問題の規模による能力の変化

先の実験に加え, 他の課題群で実験を行なった結果をまとめたものが図 4.10 である.

図の横軸は課題群の中で最も難しい (深い) 層の数であり, 縦軸は課題群の中に何層毎に課題があるのかを表している. また, ○は選択系を使った方が学習が明らかに速いことを表しており, = は選択系とランダムの学習の差がほとんどみられなかったことを表している.

この図から, 選択系の有効性は最終的に解くべき課題が困難であるときに顕著に現れることがわかる. また, 縦軸方向で見ると, 課題群中の課題同士の間隔や, 課題群の中で最も深い層の値を間隔で割った値である課題数は, 選択系の有効性には関係がないと考えられる.

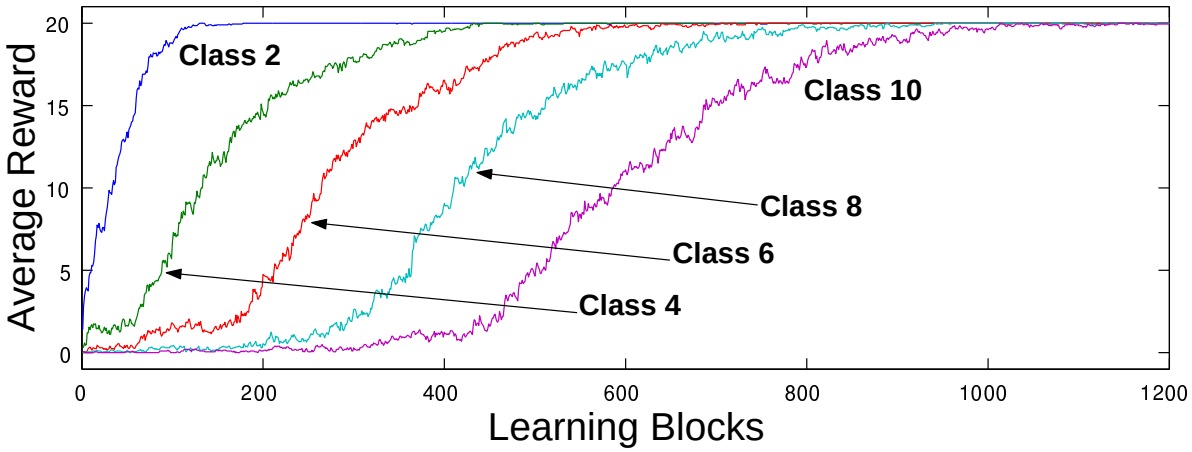

図 4.11: Class 2, 4, 6, 8, 10 の課題群の中からランダムに選択した場合の結果

したがって, 価値関数を共有した場合は, この学習システムをある問題に応用する際に, その問題が難しければ難しいほど, この学習システムが有効である可能性が十分にあるといえる.

4.3.6 課題毎に価値関数を用意し, 課題をランダムに選択した場合

図 4.11 は課題毎に価値関数を用意し複写する場合で Class 2, 4, 6, 8, 10 の課題群の中からランダムに選択した場合の結果である. この図と図 4.4 を比較すると, 価値関数を共有する場合と同様に, 易しい課題を混ぜることで学習が早くなっていることがわかる. 共有した場合との比較は後で述べる.

4.3.7 課題毎に価値関数を用意し, 課題を自律的に選択した場合

図 4.12 は, Class 2, 4, 6, 8, 10 の課題群の中から選択系を使用して自律的に選択した場合の結果である. この図と図 4.4 を比較すると, 価値関数を共有する場合と同様に, 易しい課題を混ぜることで学習が早くなっていることがわかる.

また, 図 4.13 は課題毎に価値関数を用意したときの実験結果 (試行回数は 1 回) であり, Class 4 で学習することにより得た知識 (価値関数) を Class 8 に複写するのことにより, 若干であるが学習曲線が少し立ち上がるが, 学習曲線がすぐに下がってしまうことがわかる.

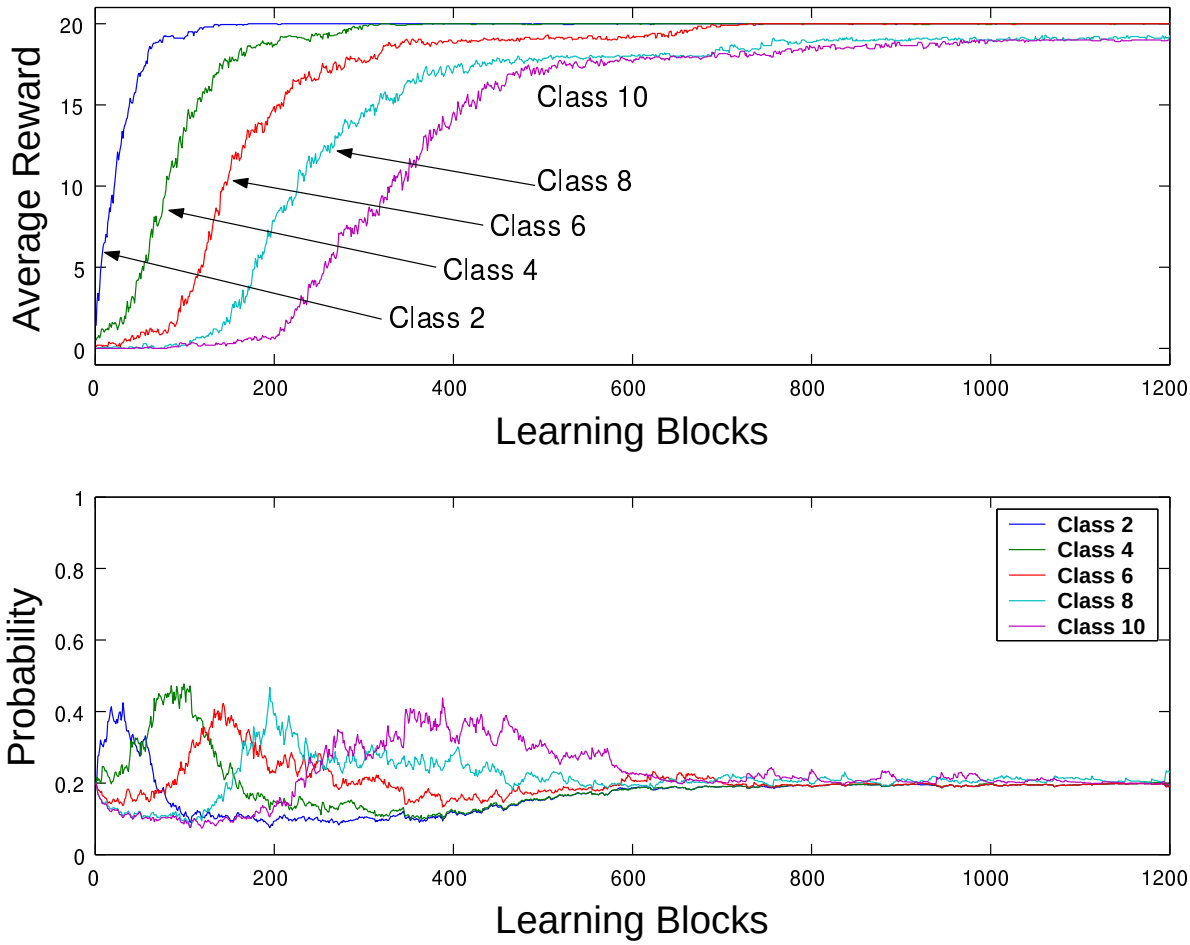

図 4.12: Class 2, 4, 6, 8, 10 の課題群の中から自律的に選択した場合の結果

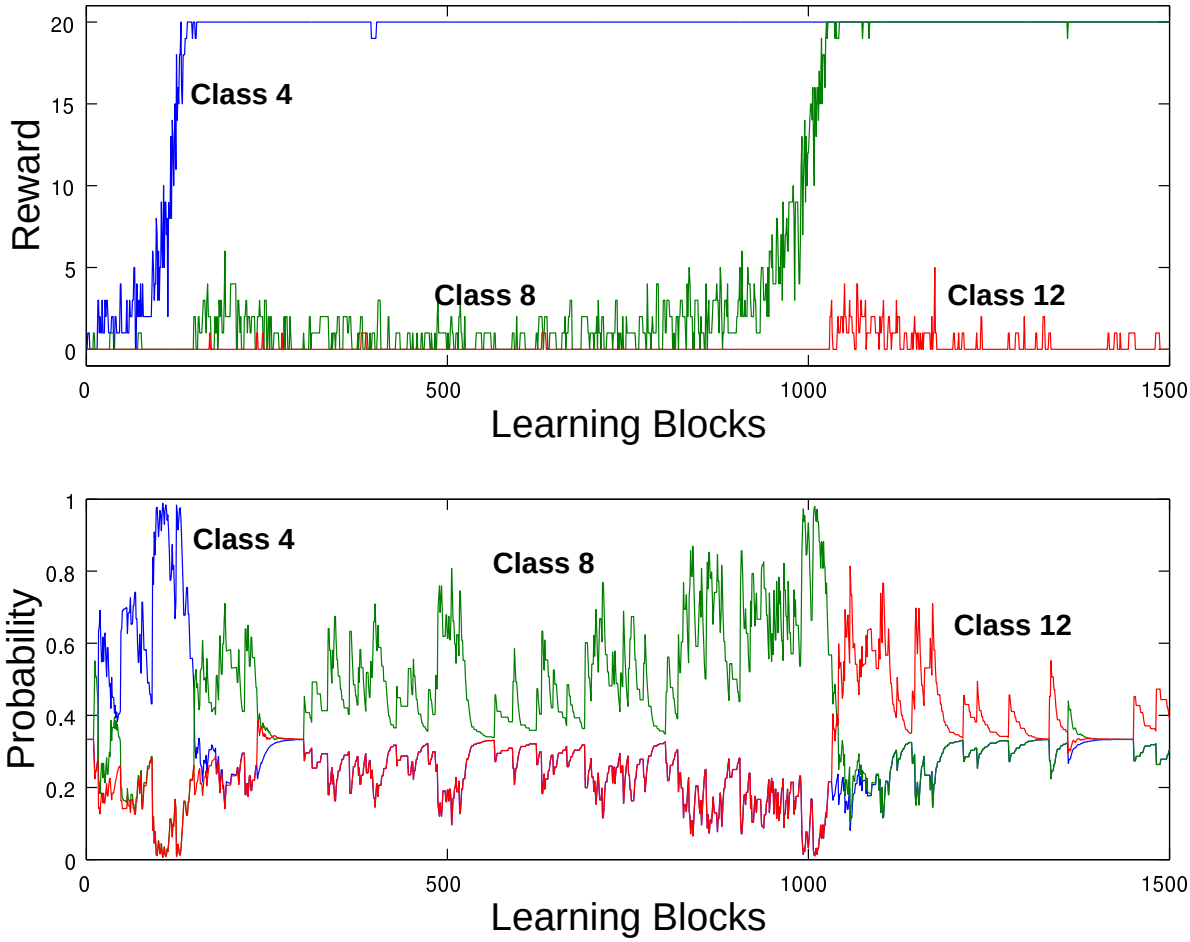

図 4.13: 課題毎に価値関数を用意し Class 4, 8, 12 の課題群の中から自律的に選択した場合の結果 (1 回の試行)

4.3.8 課題毎に価値関数を用意し, 価値関数を複写する場合の考察

価値関数を複写する場合, ランダムに選択するより, 自律的に学習させた方が早い. その理由は, 易しい課題の価値関数から難しい課題の価値関数への複写を行うということは, それまでに, 難しい課題で更新してきた価値関数がすべて書き直されるということである. したがって, 易しい課題から価値関数の値を得る前は, なるべく難しい課題で学習しない方が無駄がなく, よいということがわかる. よって, 課題をランダムに選択した場合より, 易しい問題から順次学習している選択系を使用した場合の方が学習が速く進む.

4.3.9 価値関数を共有する方法と課題毎に価値関数を用意する方法の比較

価値関数を共有する方法と課題毎に価値関数を用意し複写する方法を比較する.

まず, 価値関数を共有した場合の問題点は, 難しい課題の学習過程が易しい問題の価値関数に影響を及ぼしてしまう点である. 例えば, 価値関数を共有した場合である図 4.9 では, Class 4 の学習曲線は一旦飽和するが, Class 8 が学習曲線が立ち上がり始めると, Class 4 の学習曲線はまた下がり始める. この原因は, 第 4 層と第 5 層の間のパス上の価値関数が Class 4 の課題の場合は, 第 5 層がゴールなので, 価値関数の更新は報酬をもとに更新されるが, Class 8 の課題の場合はその価値関数の次の価値関数は, まだ更新されていない価値関数 (価値関数の値はほとんど 0) なので, その価値関数をもとに更新すると, 第 4 層と第 5 層の間の価値関数の値は下がってしまうからである (初期値を 1 にすると, 第 4 層と第 5 層の間のパス上の価値関数を一つ下のパス上の価値関数をもとに更新しても, そのもとにする価値関数の値が 1 に近いかもしれないので, 更新した価値関数の値はさがらないかもしれない). このような影響が強く現れると, 難しい問題に取り組んでいる間に, それまでは解くことができた易しい問題まで解けなくなってしまうことがある. この問題は課題毎に価値関数を用意することにより, 回避することができる.

しかし, 課題毎に価値関数を用意し複写する場合の問題点は, 1 回しか複写しない点である. 例えば, 価値関数を複写する場合である図 4.13 では, 易しい課題で学習した知識を受け取ってパフォーマンスが一旦向上しても, また下降することがおこっている. これは, 易しい課題で得た知識を一度しか利用していないために起こる問題である. 価値関数を共有する場合は常に易しい課題で得た知識を利用していることと同等と考えることができ

るので, このような問題は起こらない. さらに, 複写することの最大の問題点は学習者が, 期待報酬を事前に知っていることである. そうしなければ, 価値関数を複写するタイミングがわからないからである. 強化学習の利点の一つとして, 事前に知識がなくても試行錯誤によって学習することであるから, この方法だとその利点失うことになる.

したがって, 課題毎に価値関数を用意させ, 各課題で学習した知識を破壊せず, 期待報酬を事前に知らなくても, 易しい課題を解いたときに得る知識を常に利用できるシステムを考えなければならない.

例えば, 学習曲線が飽和するときの報酬の値を知らなくても, Class 4 の学習曲線が立ち上がり, 飽和し始めてきたときに, Class 4 の価値関数を他のクラスの価値関数に複写する方法を考える. 学習曲線が飽和し始めているときは, 学習曲線が上に凸で, 学習曲線の変化率が 0 に近いときである. つまり, 学習曲線が連続であれば, 学習曲線の一回微分の値は 0 に近く, 二回微分の値は負であることである. したがって, 学習曲線の時間変化が 0 で, さらに学習曲線の時間変化をグラフ化し, その時間変化が負になるところで, 価値関数を複写すればよい.

また, 図 4.13 のように複写直後にパフォーマンスの学習曲線が下がる限り, 学習に関与した部分のみ価値関数を複写し続けるという方法が考えられる.

4.3.10 価値関数を共有し, TD 誤差を選択系の報酬とした場合

図 4.14 と図 4.15 はそれぞれ, TD 誤差を選択系の報酬とした場合で TD 誤差を平均化する場合と平均化しなかった場合の図であり, 上から学習曲線, 学習ブロック得られた TD 誤差 (TD 誤差を平均化した場合はその平均値しない場合は定数 400 で割った値をあらわしている), 選択確率曲線を表している.

まず, 学習曲線を比較すると, 平均化しない方が学習が速く進んでいることがわかる. さらに, 選択確率曲線を比較すると, 平均化しない場合は学習曲線が立ち上がっているときのその課題を選択している確率が高く, 平均化しない場合は, 常に Class 4 を選ぶ確率が高いということがわかる.

また, TD 誤差を平均化しないときと, 解決系の学習曲線の時間変化を報酬として自律的に選択した場合である図 4.8 を比較すると, 学習の速さはほとんどかわらず, TD 誤差を

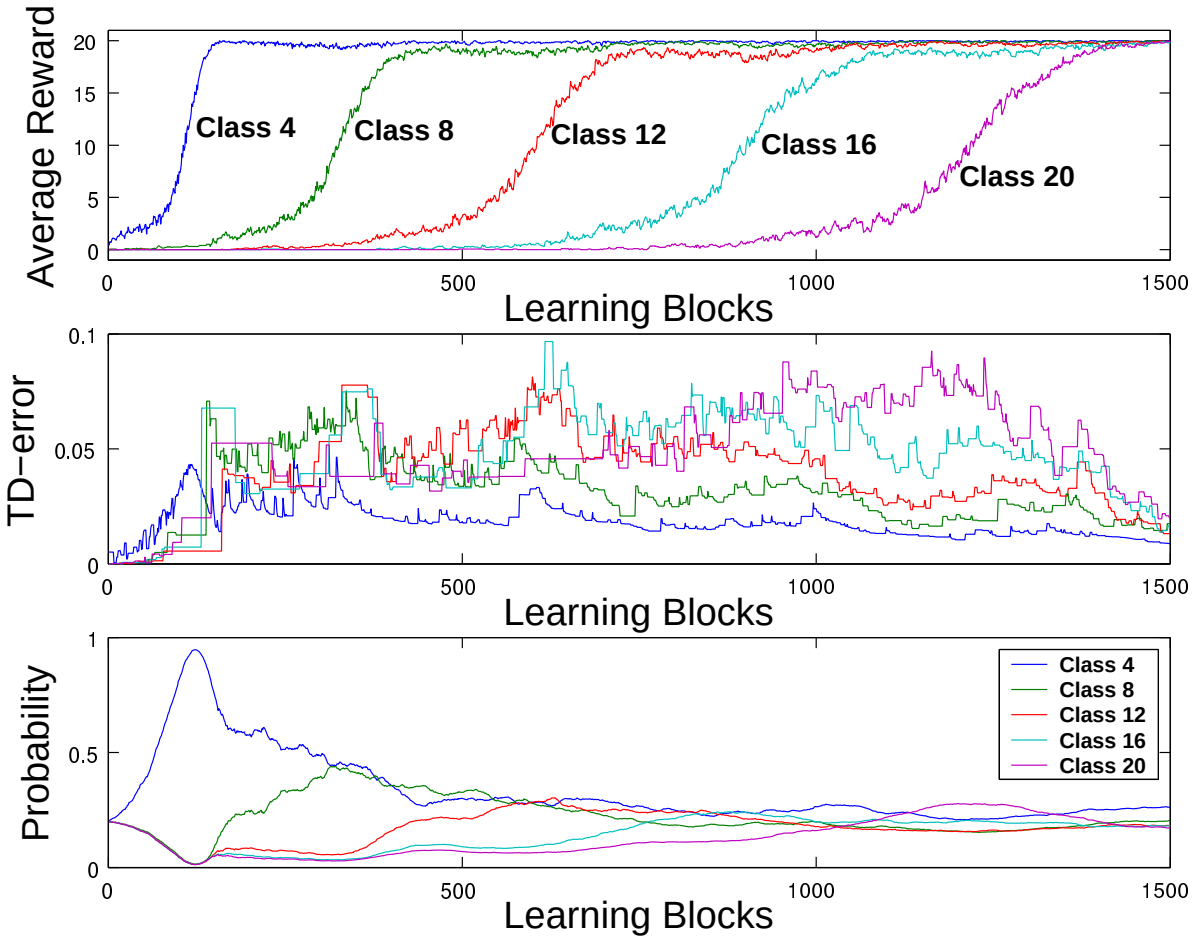

図 4.14: TD 誤差を報酬とした場合 (TD 誤差を平均化した場合)

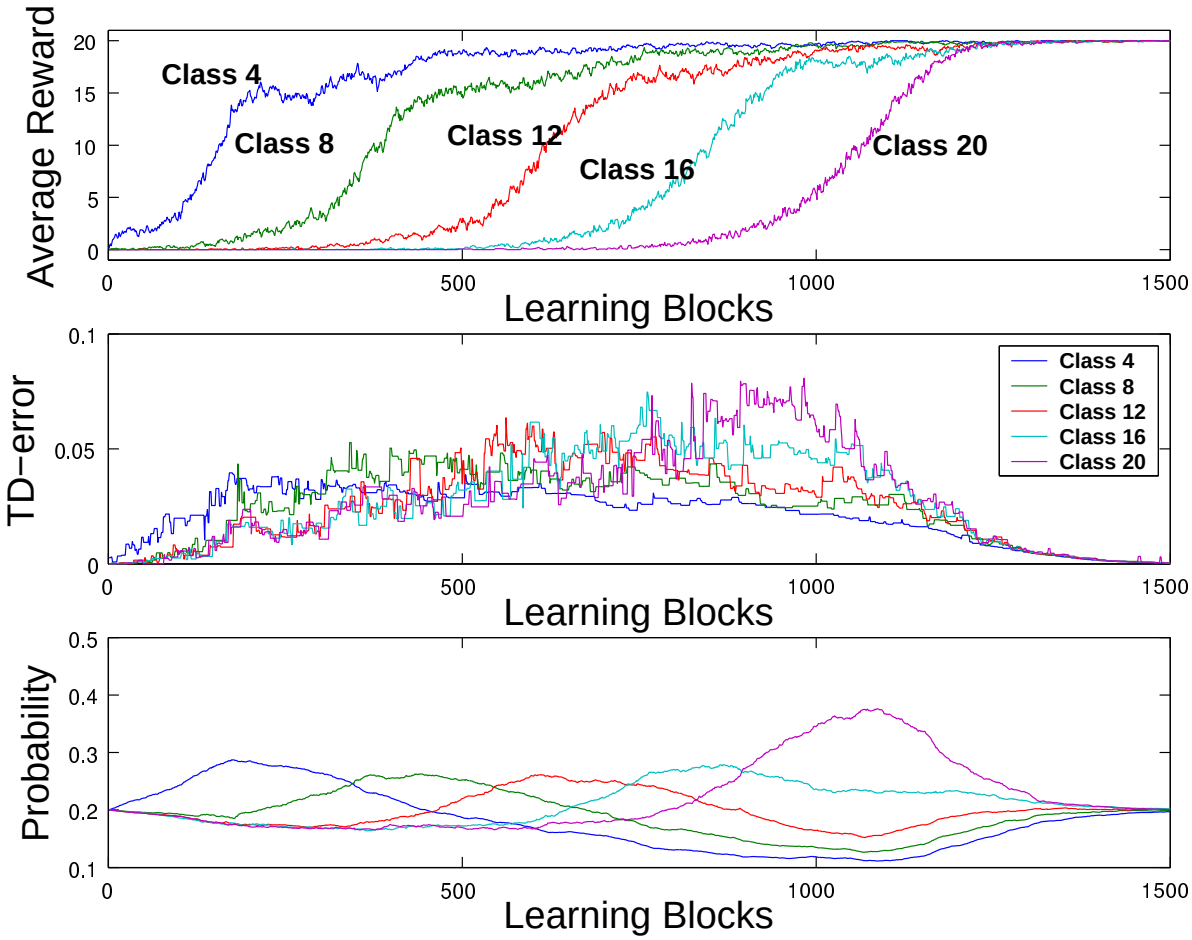

図 4.15: TD 誤差を報酬とした場合 (TD 誤差を平均化しなかった場合)

報酬としたときも学習システムがうまく機能していることがわかる.

4.3.11 価値関数を共有し, TD 誤差を選択系の報酬とした場合の考察

まず, TD 誤差を平均するより, 平均しない方が学習が早い原因を考える. 以下の考察では, 行動価値関数の初期値を 0, 正解の場合の報酬は 1, 不正解の場合の報酬は 0, γ の値を 1 と仮定している.

報酬を得ることができるパス上にある行動価値関数 $Q(i.1, \mathrm{left})$ を改めて Q_i とおくと, Class 4 の学習曲線が飽和したときのエピソード中の総和 Δ_4 は (TD 誤差の平均は $\Delta_4/4$), $Q_1 \leq Q_2 \leq Q_3 \leq Q_4 \leq r = 1$ (この場合は, 終端状態以外は報酬が得られないので, 次の価値関数の値をもとに更新しなければならないからである) であるから,

$$\begin{aligned}
\Delta_4 &= \sum_{1 \leq t \leq 4} |\delta_t| \\
&= |r - Q_4| + |Q_4 - Q_3| + |Q_3 - Q_2| + |Q_2 - Q_1| \\
&= (r - Q_4) + (Q_4 - Q_3) + (Q_3 - Q_2) + (Q_2 - Q_1) \\
&= r - Q_1 \qquad (4.5)
\end{aligned}$$

である. 一方, Class 8 の学習曲線が飽和したときのエピソード中の総和 Δ_8 は (TD 誤差の平均は $\Delta_8/8$), Class 4 でも学習する分 Q_4 の値の方が Q_5 の値より大きいため,

$$\begin{aligned}
\Delta_8 &= \sum_{1 \leq t \leq 8} |\delta_t| \\
&= |r - Q_8| + \cdots + |Q_2 - Q_1| \\
&= (r - Q_8) + \cdots + (Q_6 - Q_5) + (Q_4 - Q_5) + (Q_4 - Q_3) + \cdots + (Q_2 - Q_1) \\
&= (r - Q_1) + 2(Q_4 - Q_5) \qquad (4.6)
\end{aligned}$$

である. これら 2 つの式を比較すると, したがって, TD 誤差を平均化し, $\Delta_4/4, \Delta_8/8$ を選択系の報酬とすると, $r - Q_1$ の値が $(Q_4 - Q_5)$ に比べて大きいほど, Δ_4 と Δ_8 の値の差が小さくなることがわかる. ほとんどの場合 $\Delta_4/4$ の方が$\Delta_8/8$ より大きくなり, Class 4 は無条件に選択される確率が高くなる. 図 4.14 の結果は正にこれが現実に起きることを示している.

一方, 平均化せず Δ_4, Δ_8 をそのまま選択系の報酬とする場合は, $Q_4 - Q_5 \geq 0$ であるため, Class 8 が立ち上がり始めると, Class 4 より Class 8 の方が選ばれる確率は高くなる. 次に, Class 4 では正解に辿り着くが, Class 8 ではなかなか正解に辿り着かない場合を考える. 正解からはずれる時刻を $t = k$ とすると, それ以降の価値関数の値は不正解の方なので更新されず 0 であるので (時刻 k 以降の不正解につながるパス上の価値関数を q_t とする),

$$\begin{aligned}
\Delta_8 &= \sum_{1 \leq t \leq 8} |\delta_t| \\
&= |0 - q_8| + \cdots + |q_{k+2} - q_{k+1}| + |Q_{k+1} - Q_k| + |Q_k - Q_{k-1}| \\
&\quad + \cdots + |Q_5 - Q_4| + |Q_4 - Q_3| + \cdots + |Q_2 - Q_1| \\
&= (0 - 0) + \cdots + (0 - 0) + (Q_{k+1} - Q_k) + (Q_k - Q_{k-1}) \\
&\quad + \cdots + (Q_4 - Q_5) + (Q_4 - Q_3) + \cdots + (Q_2 - Q_1) \\
&= (Q_{k+1} - Q_1) + 2(Q_4 - Q_5) \\
&\doteq 2Q_4 - Q_1 \qquad (4.7)
\end{aligned}$$

である. したがって, 平均化しない場合において $r > 2Q_4$ のときは, Class 4 を選ぶ確率が高くなる. その結果, 図 4.15 のように学習システムがうまく機能する.

しかし, 逆にいえば, $r > 2Q_4$ が成り立たなければ, Class 8 が学習できていないにもかかわらず, Class 8 を選択する確率が高くなり, 学習システムがうまく機能しないかもしれない.

最後に, 選択系の報酬として解決系の報酬差を用いた場合と, TD 誤差を用いた場合を比較する. 解決系の報酬差を選択系の報酬とするときの欠点は, 学習曲線が向上しているときにその課題を選択する比率が増えるので, 学習曲線がほぼ垂直に立ち上がるときは, 選択系がうまく機能しないことである. TD 誤差を選択系の報酬とした場合は, 価値関数は徐々に更新され, その増加分である TD 誤差を報酬としているのでこのような問題はおこらない.

一方, TD 誤差を選択系の報酬とする場合にも欠点がある. 図 4.14 と図 4.15 の選択系の報酬である TD 誤差をみると, 選択系の報酬が平均化する場合はクラスの低い方が大きく, 平均化しない場合はクラスの高い方が大きくなり, 全体的にみて, 課題毎に報酬の大きさ

が異なるという問題がある. 解決系のパフォーマンスの時間変化を報酬とする場合は, 課題同士パフォーマンスの時間変化が同じふるまいをする限りは, 得られる報酬は等しいので問題はない.

そこで, これらの欠点を補ってくれる可能性があることとして, TD 誤差を報酬とすることと解決系の報酬差を報酬とすることとをなんらかの方法をもちいてバランスよく組み合わせることにより, 選択系がより効果的になると考えられる. 例えば, 学習曲線が立ち上がる前は TD 誤差を使用し, 立ち上がっている最中や飽和したときは解決系の報酬の時間変化に切り替える方法が考えられる. しかし, 学習者が飽和する値を事前に知らなければならないという問題があるので, その問題を解消しなければならない.

第5章　例題2 二分木(2)

5.1　例題の意義と目的

例題1では易しい問題が難しい問題のサブゴールを与える場合を考えた. 本章では, 難しい問題でとれる行動が易しい問題でもその行動がとれる場合, すなわち, 易しい問題で報酬が得られる行動系列の集合が難しい問題で報酬が得られる行動系列の集合を包含する場合について検討する. 例えば, ラケットで球を打ち返す課題では, 小さなラケットで球を打ち返すときろ同じ動きをすれば, 大きなラケットを使ったときでも打ち返すことができる. しかし, その逆は必ずしも成り立たない. したがって, 大きなラケットで球を打ち返すことのできる動きが, 小さなラケットで球を打ち返すことのできる動きを包含していることになる. 本章では, 小さなラケットでいきなり練習するのではなく, まず, 大きなラケットで練習してから小さなラケットを練習すると習得が速くなる場合について考える.

この問題を議論するため, 本章でも二分木構造をもった問題を例題としてとりあげる. 例題1と異なる点は, 課題の違いを層の深さではなく, リーフの正解数とした点である. 正解数が多いほど (例えるとラケットが大きいほど), 課題が簡単だと予想できる.

結論から述べると, この例題についてはTD学習では, 学習システムはうまく機能しないことが明らかになった. その原因を考察した結果, 上の層の価値関数を更新する際に, 得られた報酬の値を直接用いた方がよいと考え (TD学習では, 上の層の価値関数の更新する際に, 直接報酬の値を用いていない), そのように更新することができるTD(λ)学習を用いることが有効であることを述べる.

5.2　例題の内容と実験条件

前節で述べたように, 本章の例題では, 層の深さを固定させた完全二分木構造とする. 例えば, Class 2 と Class 4 からなる課題群であるとすると, 図5.1のように, Class 2 におい

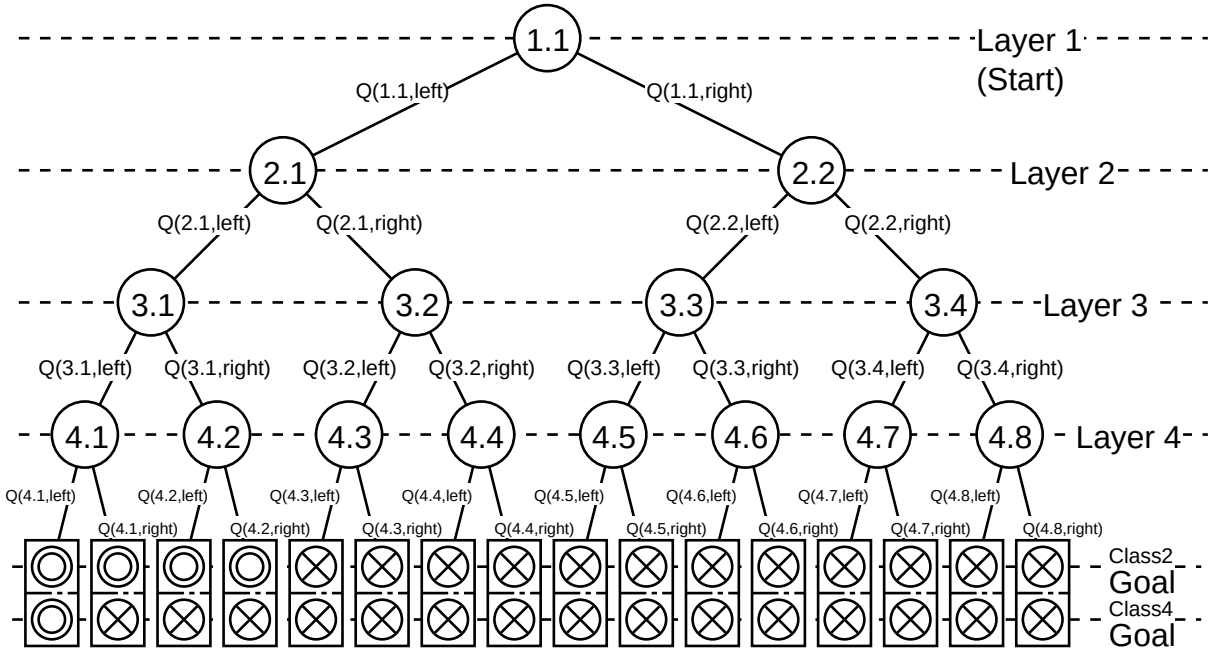

図 5.1: Class 2 と Class 4

ても Class 4 においても, 学習者は第 5 層まで移動する.

課題 Class N_m の場合, ノード $(N+1).1$ を通過すると, 第 N_M+1 層で 1 の報酬が得られる. 例えば, 図 5.1 において, Class 2 を選択して試行する場合, ノード 3.1 を通過すると, 第 5 層で 1 の報酬を得ることができ, それ以外のノードでは報酬を得ることができない (0 の報酬を得る). また, Class 4 を選択して試行する場合は, ノード 4.1 の左のリーフに移動しなければ報酬は得ることができない.

以上の条件以外は, 例題 1 と同様に問題解決系の行動価値関数の学習には Q 学習と Q(λ) 学習 (Q(λ) 学習とは, TD 誤差を Q 学習の定義で決めた TD(λ) 学習のことであり, λ の値を 0.8 とする) を用いた (行動価値関数の初期値は 0). 学習係数は 0.02, 割引率を 1 とし, 行動選択則の温度パラメータを 0.1 とした. また, 問題選択系の行動価値関数の学習係数を 0.02 (ただし図 5.3 の学習係数は 0.005) とし, 行動選択則の温度パラメータは 0.02 とした. また, Class 2, 4, 6, 8 を同時に学習させるときは 9 層構造の二分木を環境とし, Class 4, 8, 12 を同時に学習させるときは 13 層構造の二分木を環境とした.

また, 課題同士の相互作用として, 例題 1 で行った課題毎に価値関数を用意し複写する方法を採用した.

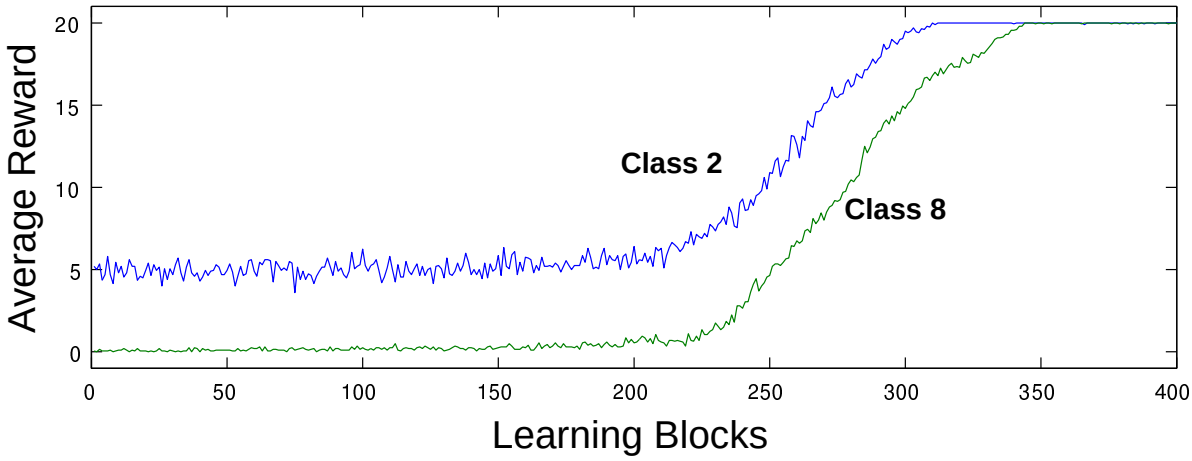

図 5.2: Class 2, 8 を個別に学習させた場合の実験結果 (学習係数=0.02)

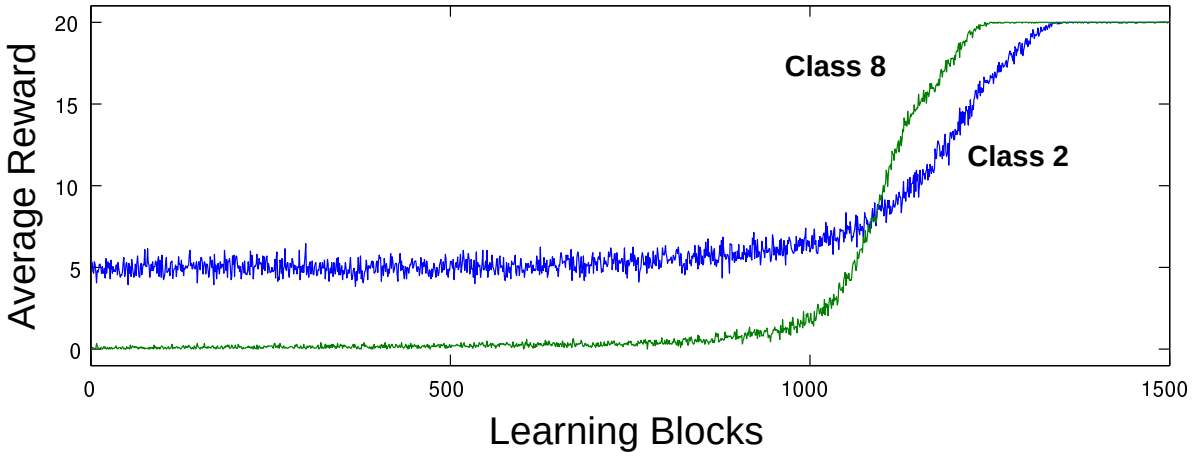

図 5.3: Class 2, 8 を個別に学習させた場合の実験結果 (学習係数=0.005)

5.3 問題解決系の学習アルゴリズムに Q 学習を用いた場合

5.3.1 個別に学習させた場合

問題解決系の学習アルゴリズムに Q 学習を用い Class 2, 8 を個別に学習させた場合で, 図 5.2 は学習係数が 0.02 のとき, 図 5.3 は学習係数が 0.005 のときの実験結果である.

これらの図より, 学習係数が 0.02 はクラスの低い課題のほうが学習が速いが, 学習係数を小さくした場合には, クラスの高い方が学習が速くなっていることがわかる.

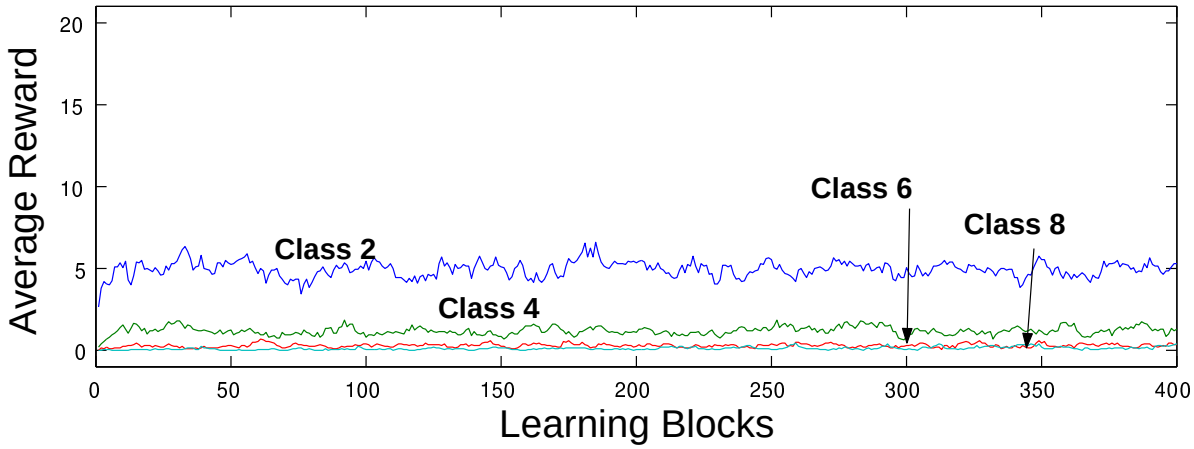

図 5.4: Class 2, 4, 6, 8 の課題群の中からランダムに選択した場合の結果

5.3.2 問題をランダムに選択した場合

図 5.5 は Class 2, 4, 6, 8 の課題群の中からランダムに選択した場合の結果である．この図と図 5.2 を比較すると，個別に学習させるほうが学習曲線の立ち上がりが早いことがわかる．

5.3.3 問題を自律的に選択した場合

図 5.5 は Class 2, 4, 6, 8 の課題群の中から自律的に選択した場合の結果である．この図と図 5.2 を比較すると，予想に反して，個別に学習させるほうが学習曲線の立ち上がりが早く，選択系が機能していないことがわかる．

5.3.4 考察

まず，個別に学習させた場合において，学習係数が小さいとクラスの高い方が学習が速くなる原因について考察する．

学習係数が小さいほど，価値関数の更新される量が小さいので，学習し始めは，学習者はランダムに移動する．(1) ランダムに移動すると，同層で正解につながるパス上の価値関数同士はほぼ平等に更新される．(2) 平等に更新されると，正解につながるパスの中でランダムに行動する．このように，(1), (2) の繰り返しになり，正解のパス上の価値関数をすべて更新する分，正解のパスを一つ見つけて集中的に更新するより，正解を選ぶ確率が向上

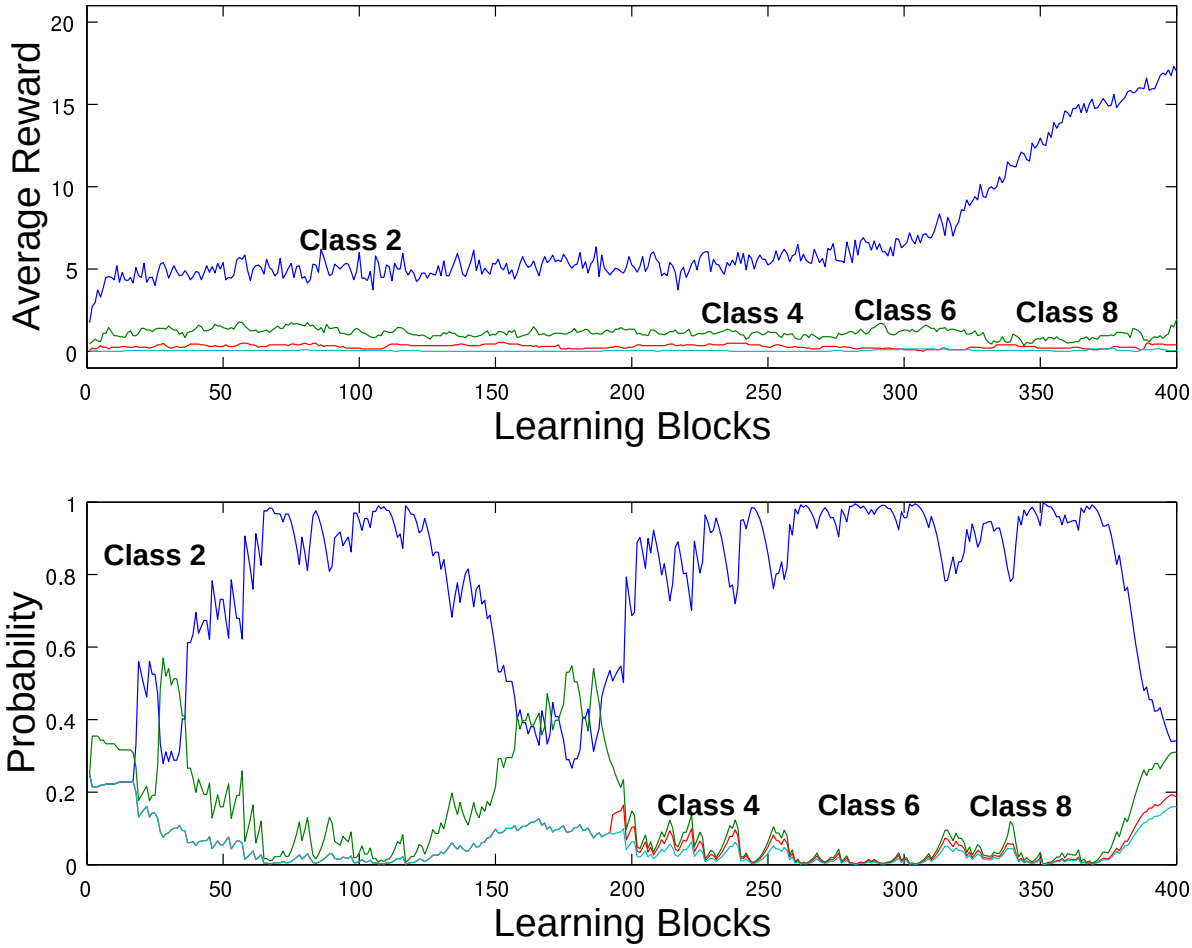

図 5.5: Class 2, 4, 6, 8 の課題群の中から自律的に選択した場合の結果

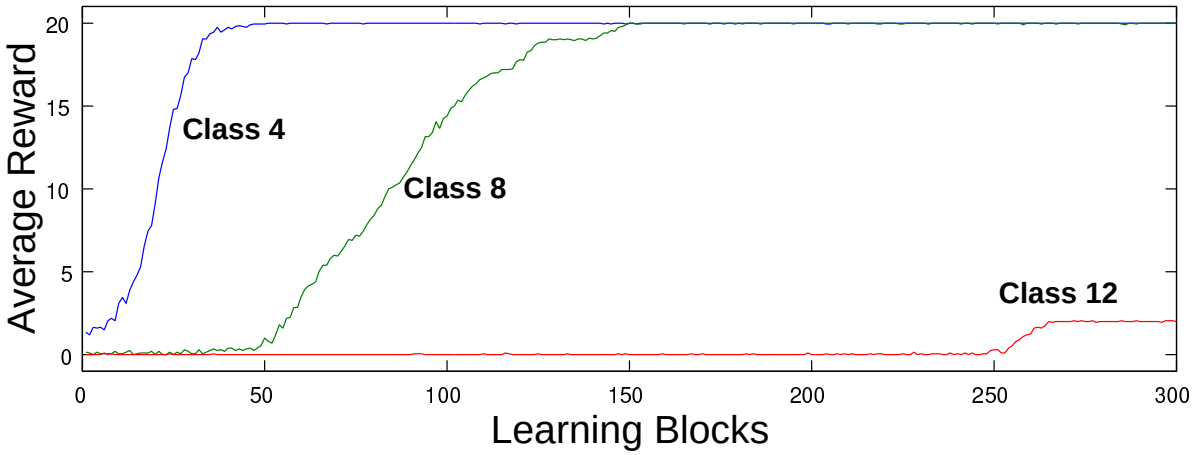

図 5.6: Class 4, 8, 12 を個別に学習させた場合の実験結果

しにくい. つまり, 学習係数が小さい場合は, 正解数が多い方が学習が遅くなると考えられる.

また, 個別に学習させるより, 選択系を用いて自律的に学習する方が学習が遅くなる原因は, 後ほど詳しく述べるが, ランダムに選択しても遅くなることから, 難易度の異なる課題を同時に学習させるからである. TD 学習では, ラケットで球を打ち返す課題に例えると, ラケットの振り始めの動きよりラケットで球に当たる瞬間の動きを学習するからであると考えられる. この例題では, 球を当てるための動きの振り始めは小さなラケットも大きなラケットも同じ動きなので, 振り始めを学習できた方がよい. しかし, TD 学習では, 球を当てる瞬間の動きから学習するので, 大きなラケットで練習すると小さなラケットでは球に当たらない振り方を覚えてしまう, つまり, 悪い癖がついてしまい, かえって, 小さなラケットでの練習の妨げにしまうと考えられる. したがって, ラケットの振り始めの動きも学習できるように, すなわち, 二分木の上部を学習させる比率をあげたほうがよいと考えられる.

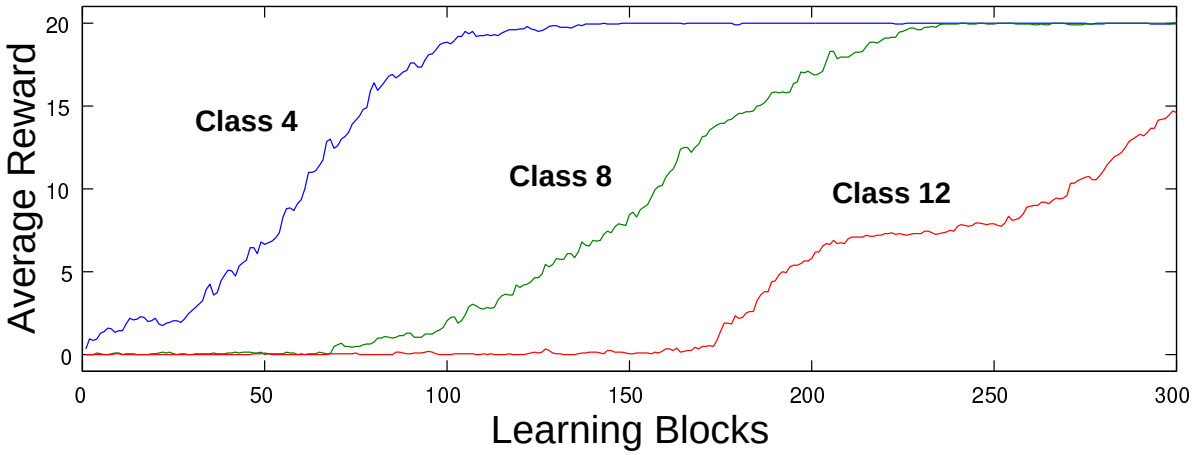

図 5.7: Class 4, 8, 12 の課題群の中からランダムに選択した場合の実験結果

5.4 問題解決系の学習アルゴリズムに Q(λ) 学習を用いた場合

5.4.1 個別に学習させた場合

問題解決系の学習アルゴリズムに Q(λ) 学習を用いて, Class 4, 8, 12 を個別に学習させた場合の結果を図 5.6 に示す. この図では, クラスの低い方から順に学習していることがわかり, 学習アルゴリズムに Q 学習を用いた場合とは異なり, 学習係数がどのような値でも, クラスの低い方から順に学習する.

5.4.2 問題をランダムに選択した場合

Class 4, 8, 12 の課題群を用意しそれらの中からランダムに選択した場合の実験結果を図 5.7 に示す, この図と図 5.6 をそれぞれ比較すると, 易しい課題を混ぜたときであるランダムに選択したときの方が学習が早く進むことがわかる.

5.4.3 問題を自律的に選択した場合

Class 4, 8, 12 の課題群の中からランダムに選択した場合の実験結果を図 5.8 に示す. まず, 課題群が Class 4, 8, 12 であるこの図と図 5.7 を比較すると, ランダムに選択した場合より, 自律的に選択した場合の方が学習が早いことがわかる. 課題群や学習係数を変更し

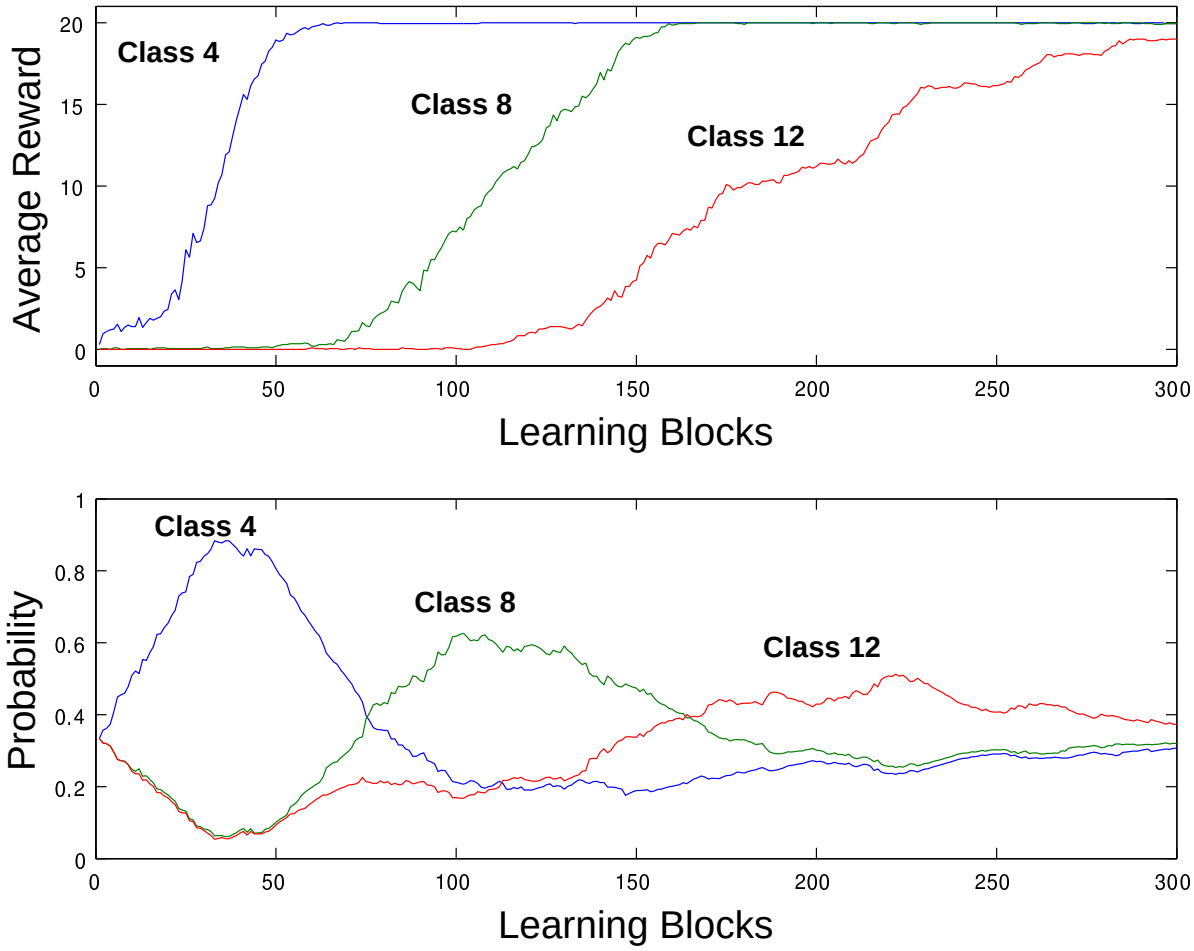

図 5.8: Class 4, 8, 12 の課題群の中から自律的に選択した場合の実験結果

ても, 自律的に選択した方が学習が早いという結果が得られている.

5.4.4 考察

難易度が異なる問題を同時に学習する場合, Q 学習では学習が遅くなるが, Q(λ) 学習では学習が速くなる原因を理解するために価値関数の値を調べた. 図 5.9 は, Class 4, 8, 12 の課題群で学習したときの, Class 8 の問題に対する報酬を得られる回数が 20 回中 18 回を越えた時点で, 16 ある正解へのパス上の価値関数の値を棒グラフに示したものである. 上下の図を比較すると, Q 学習を用いた場合には, 各ノードにおける左右のパスの価値関数の差が大きく, 上のノードほど価値関数の値が小さいことがわかる.

Class 8 の価値関数を Class 12 の価値関数を複写する際に, 第 9 層から第 13 層 (ゴール) のパス上の価値関数において, 左右のパスの価値関数を比較すると, 差がないことが理想的である. それは, Class 8 では正解であるすべてのリーフの中から, Class 12 の正解のリーフを平等に探索できるからである.

しかし, Q 学習を用いた場合のように, 左右のパスの価値関数の差が大きいと, 問題を解く際に一方のパスが集中的に選ばれるため, 広い範囲の解に到達出来なくなる. また, Q 学習では, 正解ではないリーフに到達してしまう場合, 下の層のパス上の価値関数を修正したときに, 修正し始めでは不正解に辿り着いてしまうため, 0 の報酬で価値関数は更新されることになる. したがって, 上の層のパス上の価値関数も同時に壊れて学習が不安定になりやすい. このように Q 学習では二つの問題点が共に生じるために, 期待した通りの結果が得られなかったと考えられる.

一方, Q(λ) 学習のときも, 左右のパスの価値関数の差が大きいところもあるが, Q 学習のときほど差はなく, また, 上のノードほど価値関数の値が若干大きいことがわかる. 上のノードの価値関数は大きいので, 価値関数を修正しても, 上のノードの価値関数の値も弱冠さがるものの, 下がりきらずに, 上の一番左のパスが正解のパスであるという知識が残る. したがって, Q(λ) 学習では学習が速くなると考えられる.

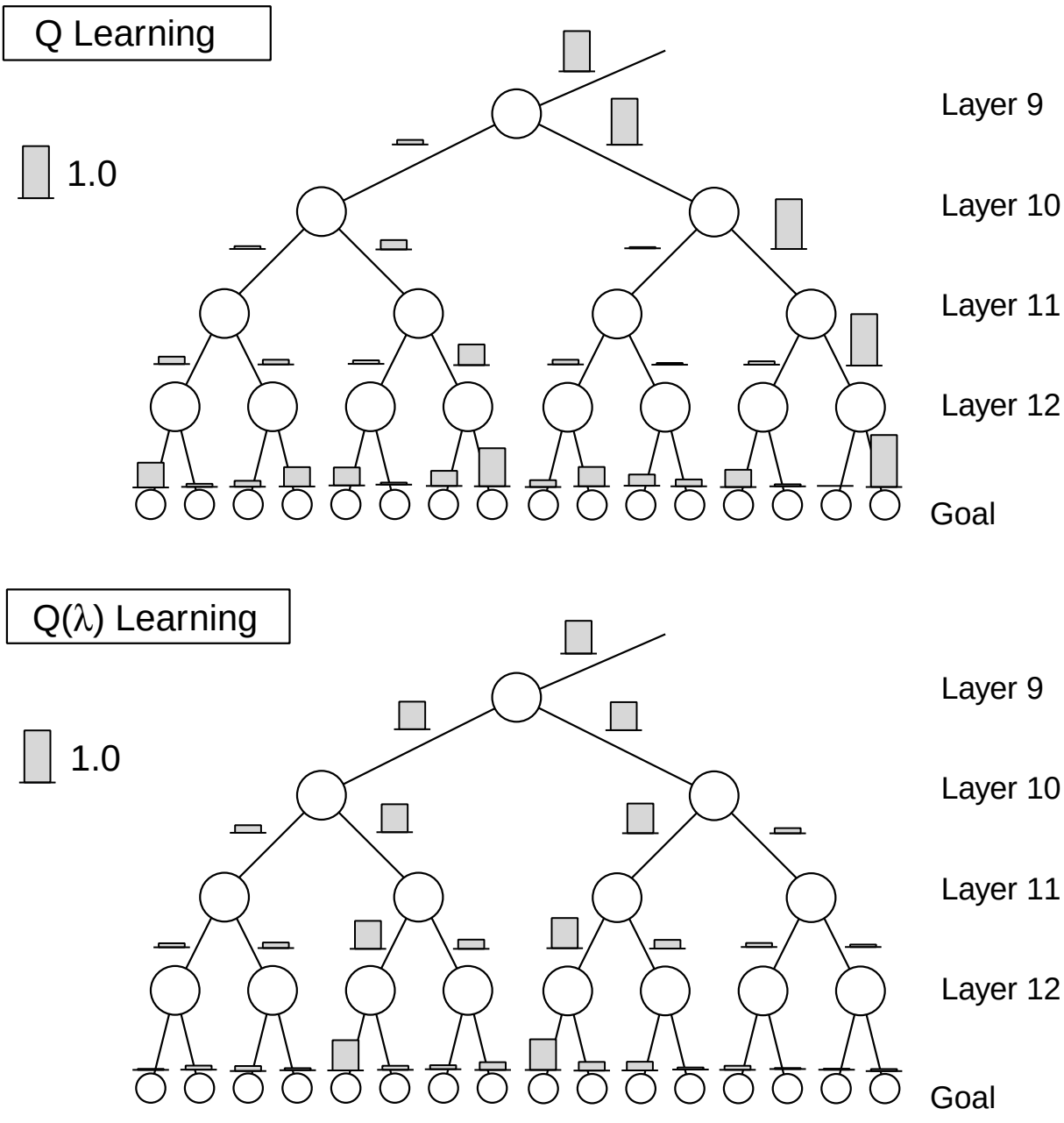

図 5.9: 解に至るパスにおける価値関数の分布

第6章　例題3 計算問題

6.1　例題の内容

これまで本研究が取り上げてきた例題では, 課題それぞれを単独で学習しても, 十分長い時間をとれば最終的に学習できるものを使用してきた. 本章では, ある課題を学習するのに前提となる知識が必要となり, その課題だけを試行しているだけでは学習できない例題を取り上げる. その例題として足し算を学習してから掛け算を学習するという計算問題を取り上げる.

この計算問題では, 学習者 (エージェント) は

- 1 桁の足し算 (addition1)
- 2 桁の足し算 (addition2)
- 1 桁の掛け算 (multiplication)

の三つの課題の中から選ぶことができる. 1 桁の足し算で学習した知識が, 2 桁の足し算の知識となり, さらに 2 桁の足し算で学習した知識が, 1 桁の掛け算の知識となるので, 最終的に 1 桁の掛け算が学習できるようになると考えられる (これから述べる計算アルゴリズムは, 人間がこのように計算しているというわけではないということを断っておく).

6.2　実験の条件

6.2.1　1 桁の足し算

1 桁の足し算を課題として選択した場合, まず学習者には 1 桁の足し算の問題

$$x_1 + x_2 = ? \quad (x_1, x_2 \in \{0, 1, \cdots, 9\}) \tag{6.1}$$

が与えられる. この式を状態 s(x_1 と x_2 の積集合) とし, この式に対しての学習者が導き出す解 (0 から 19 の 20 通り) を行動とする選択系を設定した. また, その学習系に対して, 行動価値関数 $Q_1(s,a)$(初期値は 0) を設定した.

問題解決系へは, 学習者が導き出した解と本当の解が一致していれば, 1 の報酬を与え, 一致していなければ 0 の報酬を与えた. その後, 行動価値関数 $Q_1(s,a)$ の更新を行った.

6.2.2 2 桁の足し算

2 桁の足し算を課題として選択した場合, まず, 学習者には 2 桁の足し算の問題

$$x_1 + x_2 = ? \quad (x_1, x_2 \in \{0, 1, \cdots, 99\}) \tag{6.2}$$

が与えられる. この式の x_1 の 1 桁目を x_{11}, 2 桁目を x_{12} とし, 同様に x_2 の 1 桁目を x_{21}, 2 桁目を x_{22} として, 学習者は 1 桁目から筆算を行うように計算させていくようにした.

具体的には, まず x_{11} と x_{21} の積集合を状態 s_1, 学習者が導き出す x_{11} と x_{21} の和を行動 a_1 とする学習系を設定し, 1 桁の足し算で用いた行動価値関数 $Q_1(s_1,a_1)$ を設定した. a_1 の 1 桁目を 2 桁の足し算の解の 1 桁目 y_1 とし, a_1 の 2 桁目を c_2 とし繰り上げの数とした.

次に, 2 桁目である x_{12},x_{22} と繰り上げの数 c_2 の積集合を状態 s_2, 学習者が導き出す x_{12} と x_{22},c_2 の和を行動 a_2 (0 から 19 の 20 通り) とする学習系を設定した. その学習系に対して, 行動価値関数 $Q_2(s_2,a_2)$(初期値は 0) を設定し, a_2 の 1 桁目を 2 桁の足し算の解の 2 桁目 y_2 とし, a_2 の 2 桁目を 2 桁の足し算の解の 3 桁目 y_3 とした.

問題解決系へは, 学習者が導き出した解と本当の解が一致, すなわち $100y_3 + 10y_2 + y_1 = x_1 + x_2$ となっていれば, 1 の報酬を与え, 一致していなければ, 0 の報酬を与えた. その後, 行動価値関数 $Q_2(s_2,a_2)$ のみ, 更新をおこなった (Q_1 の学習は全く行わない). このことは, 1 桁の足し算の知識があることを前堤として, 繰り上がりのある足し算を学習していると言える.

6.2.3 1 桁の掛け算

1 桁の掛け算を課題として選択した場合, まず, 学習者には 1 桁の掛け算の問題

$$x_1 \times x_2 = ? \quad (x_1, x_2 \in \{0, 1, \cdots, 9\}) \tag{6.3}$$

が与えられる. この式の x_2 を状態 s_3, x_1 を何回足せばよいかを行動 a_3 (0 から 9 の 10 通り) とする選択系を設定した. またその学習系に対して, 行動価値関数 $Q_2(s_3, a_3)$(初期値は 0) を設定した.

次に, x_1 を a_3 回繰り返し足していくが ($a_3 = 0$ の場合は 0 を学習者の導いた解とする), 足し方は 2 桁の足し算として学習系を設定し, その学習系に対して行動価値関数 Q_1,Q_2 を設定した (学習者が足し算を繰り返している途中, 解が 3 桁になった時点で学習者の解は不正解とする).

そして, 問題解決系へは, 足し算の繰り返しによって得られた解を, 学習者の導き出した解とし, 本当の解と一致していれば, 1 の報酬を与え, 一致していなければ 0 の報酬を与えた. その後, 行動価値関数 $Q_3(s_3, a_3)$ のみ, 更新を行った. このことは, 2 桁の足し算の知識があることを前提として, 何回繰り返し足せばよいのかだけを学習していると言える.

6.2.4 その他のパラメータ

以上, 行動選択には Softmax 法 (温度パラメータの値は 0.05) を用いた. 学習係数は 0.05, 割引率は 1 とした.

また, 各学習ブロックの試行数は 20 とし, ブロック前半と後半の問題解決系への平均報酬の差を問題選択系の報酬とし, 問題選択系の行動選択則の温度パラメータの値は 0.06 とした.

6.3 結果と考察

6.3.1 学習系が自ら選択した場合の結果

図 6.1 は, 学習系が自ら選択した場合の学習曲線 (上図) と, 課題の選択確率の時間変化 (下図) である. 図中の青線, 緑線, 赤線はそれぞれ 1 桁の足し算, 2 桁の足し算, 1 桁の掛け算の学習曲線を表している. また, 横軸は学習ブロックの番号, 縦軸は 1 学習ブロック中に得られる報酬の合計を表している.

この図から, 1 桁の足し算から前提知識を必要としない順に学習されていることがわかる. また, ある学習曲線が他の学習曲線と比べての変化率が大きいときに, その課題を選択する確率が最も大きくなっていることがわかる.

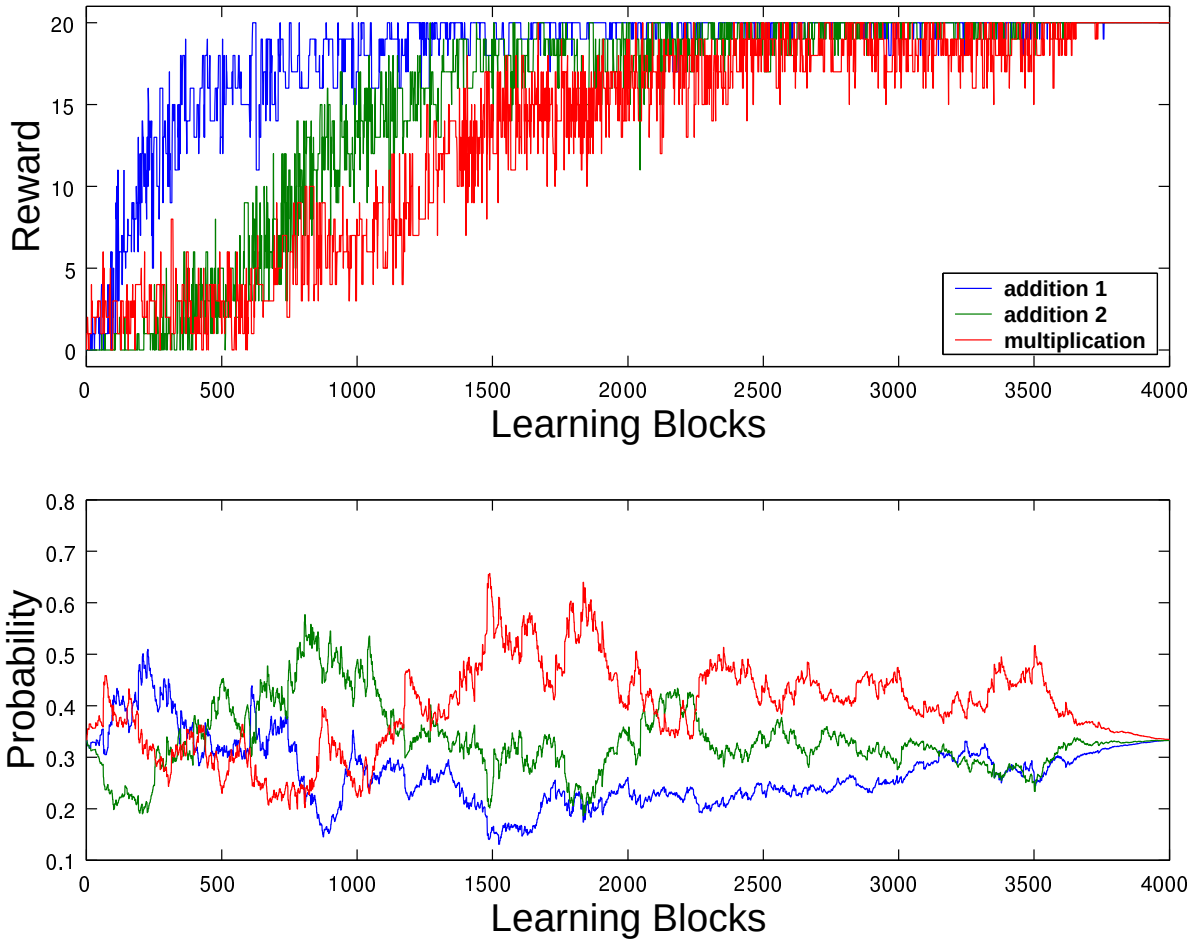

図 6.1: 自ら課題を選択した場合

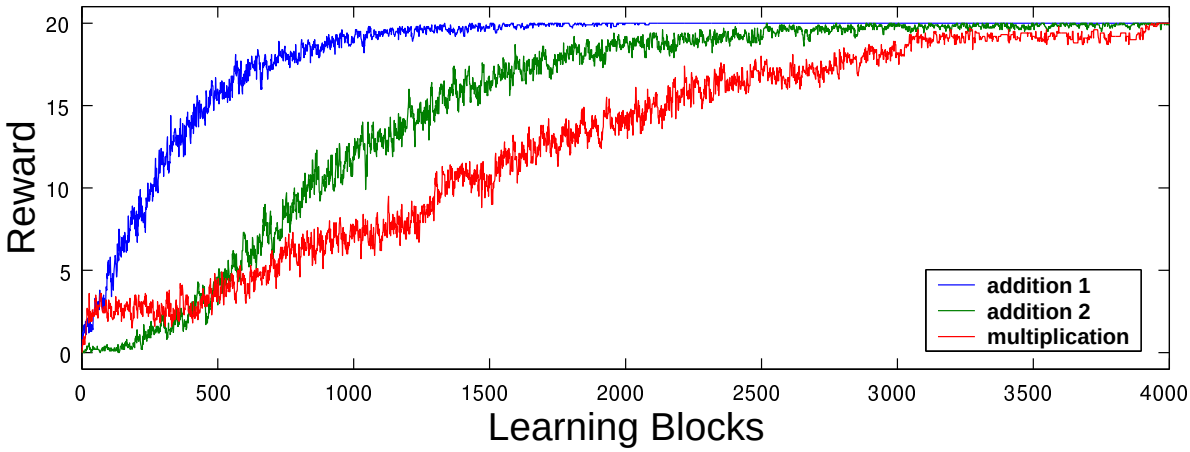

図 6.2: ランダムに課題を選択した場合

6.3.2 計算問題の考察

図 6.1 において, 1 桁の掛け算の課題で初期の段階で僅かながら報酬が得られるのは, 0 を掛ける場合は前提知識を必要としていないからである. また, 2 桁の足し算と 1 桁の掛け算を比較して, ある程度学習時間が立つと, 同時に学習しているようにみえるが, これは $11 + 13$ 等, 2 桁の足し算には 1 桁の掛け算には使用されない計算が含まれているからと考えられる.

6.3.3 ランダムに問題を選択した場合との比較

次に, 先ほどと同じ条件で, 学習系が自ら課題を選択した場合と, 課題をランダムに選択した場合とをそれぞれ, 10 回実験を繰り返し, それらの結果を平均して比較をおこなった.

図 6.2 と図 6.3 は, ランダムに課題を選択した場合の学習曲線の実験と自ら課題を選択した場合の学習曲線の実験をそれぞれ 10 回行い, 平均した結果である.

これらの図を比較すると, 1 桁の足し算と 2 桁の足し算に関しては目立った差はみられないが, 1 桁の掛け算では 2000 学習ブロック目の差等から, 自ら課題を選択させた場合の方が, 学習が早いことがわかる.

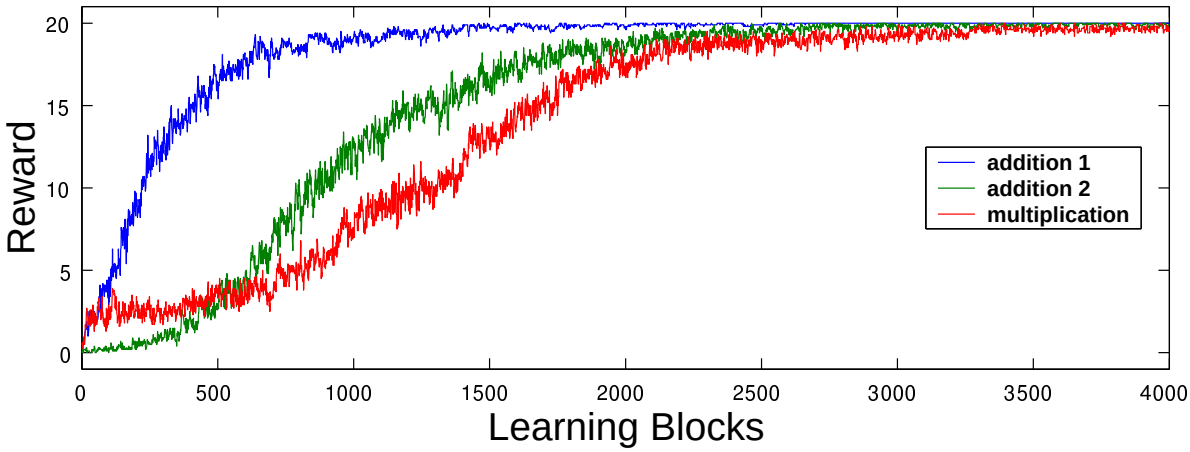

図 6.3: 自ら課題を選択した場合

6.3.4 考察

1 桁の足し算と 2 桁の足し算では差がほとんどなく, 1 桁の掛け算で差がでてきたことと, 第 4 章の結果から, さらに 2 桁の掛け算等難しい課題が増えるほど, この選択系を用いた学習システムの有効性がでてくると期待できる.

第7章　結論

以上の例題を通して, 学習曲線の変化率を報酬とする強化学習系を用いることにより, 自分の能力にあった課題のクラスを選びながら, 自己組織的に能力を向上させていく学習システムを提案した.

提案したシステムの原理を分かりやすく示すために, 第 4 章と第 5 章で単純な構造の学習者を用い, 単純な例題を用いてその振舞を例証した.

第 4 章のように, 簡単な問題が難しい問題のサブゴールとなりうるときは, この学習システムが適用できることがわかった.

問題解決系の報酬の時間変化を選択系の報酬とする方法と TD 誤差を選択系の報酬とする方法を比較し, TD 誤差を選択系の報酬とする場合でも, 易しい問題から順次解いていくような学習システムが実装できることを示した.

また, 課題毎に価値関数を用意しパフォーマンスが向上した課題の価値関数から, 向上していない課題の価値関数へ複写を行う方法と, 課題同士で価値関数を共有する方法を比較することにより, 課題毎に価値関数を用意し, 価値関数を共有した場合の利点をうまく取り入れることが必要であることがわかった.

第 5 章では, ラケットで球当ての難易度をラケットの大きさで決めるというような行動選択則を包含する場合は, 通常の TD 学習では本学習システムは有効ではないが, TD(λ) 学習のような別の学習則を用いることによって, 学習システムがうまく機能する場合があることを示せた.

さらに, 第 6 章のように, 解くべきタスクを分割した場合において, 分割したものを一つの課題として考え, この学習システムを適用させることにより, 効率の良い学習が期待できることを示した.

今後, 多くの学習システムに対して本アルゴリズムの考えが適用され, その能力が客観的に評価されることを望みたい.

謝辞

最後にこの研究をおこなうにあたって適切なご指導いただいた電気通信大学大学院情報システム研究科の指導教官である阪口豊助教授, およびご指導いただいた出澤正徳教授, 石田文彦助手, そしてヒューマンインターフェース学講座の皆様には心より感謝します.

参考文献

[1] R. Jacobs, M. Joran, S. Nowlan and G. Hinton: “Adaptive mixture of local experts”, Neural Computation, 3, 79-87, 1991.

[2] 鮫島, 片桐, 銅谷, 川人: “モジュール競合による運動パターンのシンボル化と見まね学習”, 電子通信学会論文誌, J85-D-II, 90-100, 2002.

[3] 杉本, 鮫島, 銅谷, 川人: “複数の状態予測と芳洲予測モデルによる強化学習と行動目標の推定”, 日本神経回路学会 第12回全国大会 講演論文集, 335-338, 2002.

[4] 森本, 銅谷: “階層型強化学習を用いた実ロボットによる起立運動の獲得”, 日本神経回路学会 第12回全国大会 講演論文集, 147-148, 1999.

[5] 森本, 銅谷: “階層型強化学習を用いた実ロボットによる起立運動の獲得”, 第5回ロボティクスシンポジア, 397-402, 2000.

[6] 小澤: “モジュール構造ニューラルネットの研究動向”, 計測と制御 41, 888-893, 2002.

[7] 阪口: “ニューラルネットワークとその自己組織化”, 計測と制御, 30, 336-339, 1991.

[8] 石井: “強化学習と2足歩行”, 脳情報数理科学の発展, 118-124, 2002.

[9] Richard S. Sutton and Andrew G.Barto 著, 三上 貞芳, 皆川 雅章訳, “強化学習”, 森北出版株式会社, 2000.

[10] 内田, 阪口: “学習曲線の時間変化に着目した自己組織的強化学習アルゴリズム”, 日本神経回路学会 第12回全国大会 講演論文集, 1-4, 2002.