TR-AC-0051

003

相手モデル学習を取り入れた マルチエージェント系の学習モデル （プログラムFD付）

松野 陽一郎* 山崎 達也
松田 潤** 石井 信*

*奈良先端科学技術大学院大学 **大阪学院大学

2001. 1.10

ATR環境適応通信研究所

相手モデル学習を取り入れたマルチエージェント系の学習モデル

松野 陽一郎[1]　山崎達也[2]　松田潤[3]　石井 信[1]

[1] 奈良先端科学技術大学院大学　知識工学講座
(e-mail)yoichi-m@is.aist-nara.ac.jp
[2] (株)ATR 環境適応通信研究所　第一研究室
[3] 大阪学院大学 情報学部

あらまし　マルチエージェント系の一例としてカードゲームである Hearts を取り上げ，そこでのエージェントの行動学習として，Actor-Critic 型強化学習アルゴリズムと相手モデル学習を組み合わせた確率的なモデルを提案する．マルチエージェント系では、環境的な部分観測問題と、他エージェントの内部状態の非観測性と戦略による非マルコフ性といった様々な問題を内包している。相手モデルを学習することで、マルチエージェント系における非マルコフ過程問題に取り組み、また、確率的な戦略や状態推定をおこなうことで、部分観測問題へも適用を可能にしている。また、検証のためにシミュレーション実験を行いその結果を比較する。

A Multi-agent Reinforcement Learning Method for a Partially-Observable Compatitive Game

Yoichiro Matsuno[1]　Tatsuya Yamazaki[2]　Jun Matsuda[3]　Shin Ishii[1]

[1] Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma-shi, Nara 630-0101
(e-mail)yoichi-m@is.aist-nara.ac.jp
[2] ATR Adaptive Communications Research Lab
Department 1
[3] Osaka Gakuin university

Abstract　This article proposes a reinforcement learning (RL) method based on the Actor-Critic architecture, which can be applied to partially-observable multi-agent competitive games. As an example, we consider a card game "Hearts". The RL then becomes a partially-observable Markov decision process (POMDP). However, the card distribution becomes inferable from the disclosed information as a single game proceeds. In addition, the strategy (model) of the other players can be learnable from their actual plays by repeating games. In our method, a single Hearts game is divided into three stages, and three actors are prepared so that one of them plays and learns separately in each stage. In particular, the actor for the middle stage plays so as to enlarge the expected temporal-difference (TD) error, which is calculated using the evaluation function approximated by the critic and the estimated state transition. After a learning player trained by our RL method plays several thousands training games with three heuristic players, the RL player becomes strong enough to beat the heuristic players.

1 まえがき

近年計算機システムがネットワークにより相互に接続され，複数のユーザや自律エージェントが協調あるいは競合する問題を取り扱う必要が出てきている．このような通信システム全体はマルチエージェント系として考えることができる [1]．こうした系においてエージェントが問題解決する場合，環境の動的な変化や膨大な状態数が原因となり，予測可能な事例をデータベース化する手法を適用することは難しい．変化する環境に応じて各エージェントが局所的な最適化を行うことによって全体的な問題解決が図れるという強化学習が適している [2]．強化学習とはある行動をとった時に得られる報酬により環境に適応するように学習を行う機械学習の方式である．

マルチエージェント系における強化学習は，行動の結果の評価（payoff）の設定により，協力的状況と，妥協的，競争的状況の二つに分けることができる [3][4]．前者はすべてのエージェントが同一の payoff 関数を持つような場合であり，追跡問題 [5][6] などがそれに相当する．一方後者の妥協的，競争的状況に関しては，問題がより困難なものとなるため，他者の行動を推定する過程が重要である．例えば Littman[7] はサッカーゲームを題材に，相手の戦略は学習者に提示されるという仮定に基づき，マルチエージェント系の強化学習を提案した．またマルチエージェントの競争的状況の典型例としてはゲームが考えられ，実際にブラックジャック [9]，オセロ [8]，バックギャモン [10] のような 2 人ゲームへ強化学習を適用した研究例がある．しかし，プレイヤーの数が 3 人以上の非協力ゲーム [11] では，状態の予測や相手の内部モデルの構築が難しいため強化学習による戦略獲得の研究は少ない．

本論文ではマルチエージェント系の競争的状況として非協力 n 人ゲーム（$n > 2$）のカードゲームである Hearts を取り上げ，強化学習の枠組みで自身の戦略を環境（相手プレイヤー）に応じて改良していく学習モデルを提案する．提案するモデルでは Actor-Critic アーキテクチャ [14] を用いる．Critic がプレイヤーの状態評価を行い，Actor が状況に応じた行動制御を行う．具体的には．Hearts のゲームを序盤，中盤，終盤と分け，状況に応じて得られる知識を活用し，相手の状態や行動のモデルを推定することでより最適な制御を行っている．これは文献 [12] に示されている複雑なタスクのサブタスクへの分解と考えることもできる．

以下，2．においてマルチエージェント系における学習モデルを提案し，3．で研究題材として用いた Hearts について説明し，提案モデルの Hearts への適用に関して述べる．4．において計算機実験結果とその評価を示した後，5．でまとめを行う．

2 モデル

マルチエージェント系では，各エージェントが自律的に行動選択を行う．自分および他のエージェントすべてを含む系が環境である．ただし，この環境は他エージェントの学習によって変化する．また，他エージェントの内部状態は一般に観測できないため部分観測である．このような環境下で，エージェントがある目標を達成するように戦略を学習するモデルの構築を目的とする．以降本節では，時刻 $t \in \{0,1,2,\ldots\}$ における観測状態を $x_t \in X$，その時エージェントのとる行動を $a_t \in A$，とった行動により得られる報酬を $r_t = r(a_t, x_t) \in \mathcal{R}$ と表す．ここで X は全観測状態の集合，A はエージェントの行動の選択可能な集合を表し，$\mathcal{R}$ は実数空間とする．一般には A は状態ごとに異なるが，以下では紛らわしくない場合はその依存性を省略する．

2.1 Actor-Critic アルゴリズム

通常，相手のエージェントの戦略や内部状態は観測できないので，経験から相手の内部状態や戦略を推定し，自分自身の戦略を確率的に学習する枠組みが重要である．実際に簡単な部分観測状況であっても，決定論的な戦略では悪い戦略となり得ることが知られている [13]．そのような学習方法として Actor-Critic

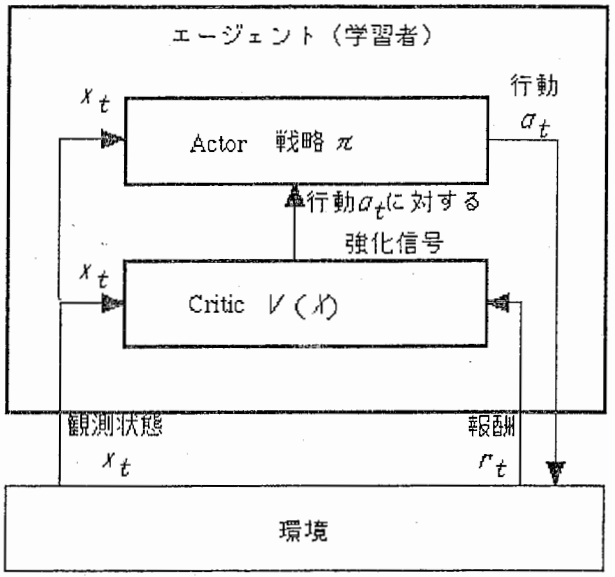

図 1: Actor-Critic 法の構成

アルゴリズム [14] と相手モデル学習を組み合わせたモデルを提案する．

図 1 に示される Actor-Critic アルゴリズムによる強化学習では，学習者であるエージェントが内部に Actor と Critic を持ち，相手エージェントを含む環境で観測された状態を Critic によって評価すると同時に，その時の行動制御を Actor によって行う．Critic は行動の結果として報酬を受取り，次の観測状態に基づき式 (1) で表される TD 誤差 δ_t を計算する．

$$\delta_t = \{r_t + \gamma V(x_{t+1})\} - V(x_t) \tag{1}$$

ここで $V(x_t)$ は観測状態 x_t の Critic による評価値で，$\gamma(0 \leq \gamma \leq 1)$ は割引率である．TD 誤差は，現在の状態における Critic の評価値 $V(x_t)$ と一つ先の状態 x_{t+1} で評価した x_t に対する評価値 $r_t + \gamma V(x_{t+1})$ の差である．一般的な Actor-Critic アルゴリズムでは，この TD 誤差を用いて Critic の状態評価関数 $V(x)$ および Actor の戦略を以下のように更新していく．

$$V(x_t) \leftarrow V(x_t) + \alpha\delta_t \tag{2}$$

$$p(x_t, a_t) \leftarrow p(x_t, a_t) + \alpha\delta_t \tag{3}$$

ここで α は学習係数，$p(x_t, a_t)$ は状態 x_t における行動 a_t に対する効用関数であり，Actor はこれを基に行動を決定する．

Actor-Critic アルゴリズムでは，現在の評価関数を用いて行動が良いか悪いかを表す TD 誤差により行動を強化している．しかし，マルチエージェント系のように相手の内部状態や戦略といった観測できない環境がある場合，それらの非観測部分が複雑に絡み合いながら状態遷移に影響するためにうまく学習ができない．

本研究では，Actor が選択する行動を現在の環境のダイナミクスから得られる観測状態遷移確率 $P^{\phi}(x_{t+1}|x_t, a_t)$ を考慮して決定する．ここで ϕ は，相手エージェントの内部状態や戦略といった学習エージェントからは観測できない環境のダイナミクスを表している．この観測状態遷移確率は，

$$P^{\phi}(x_{t+1}|x_t, a_t) = \sum_{s_t \in \mathcal{S}_t} \sum_{s_{t+1} \in \mathcal{S}_{t+1}} P(s_t|x_t) P^{\phi}(s_{t+1}|s_t, a_t) P(x_{t+1}|s_{t+1}) \tag{4}$$

と表せる．ここで，集合 $\mathcal{S}_{t+1}$ は状態 s_{t+1} で可能な真の状態の集合であり，集合 $\mathcal{S}_t$ は観測状態 x_t での可能な真の状態の集合である．「真の」状態とは相手の内部状態を含めた状態を意味する．また $P(x_{t+1}|s_{t+1})$ は

状態 s_{t+1} において観測状態 x_{t+1} が得られる条件付き確率を表す．通常，状態 s から観測状態 x は一意に決まるので，x_{t+1} が実際の観測状態である場合，この確率は 1 となる．$P^{\phi}(s_{t+1}|s_t, a_t)$ は環境のダイナミクス ϕ で行動 a_t を選択したときの状態 s_t から s_{t+1} への遷移確率を示す．ただし，こちらの ϕ には相手エージェントの内部状態は含まず，相手エージェントの戦略を含んでいる．さらに，$P(s_t|x_t)$ は観測状態 x_t の時に真の状態が s_t である確率である．観測状態遷移確率を求めるためには，状態遷移確率 $P^{\phi}(s_{t+1}|s_t, a_t)$ と $P(s_t|x_t)$ を求める必要がある．状態遷移確率 $P^{\phi}(s_{t+1}|s_t, a_t)$ は，相手エージェント達の戦略に依存するので，相手エージェントのモデルを学習することで求めることができる．また $P(s_t|x_t)$ は，問題依存ではあるが近似的に求めることが可能である．これらについては，3.2 節で議論する．

非観測部分の影響を考慮した期待 TD 誤差 $\langle \delta_t \rangle_a$ は以下のように求められる．

$$\langle \delta_t \rangle_a = \{\langle r_t \rangle_a + \gamma \langle V(x_{t+1}) \rangle_a\} - V(x_t) \tag{5}$$

ここで $\langle \rangle_a$ は $P^{\phi}(x_{t+1}|x_t, a_t)$ による期待値を表す．環境のダイナミクスが ϕ の下では，この期待 TD 誤差の値が大きい行動ほど，将来に渡ってより多く報酬を得られる可能性が高い行動であることを意味する．よって，期待 TD 誤差の値を効用として行動選択を行う．

2.2 確率モデルを用いた関数近似器とその学習

状態空間が非常に広い場合，すべての状態を経験することは困難であるため，未知の状態に関してもなんらかの汎化によって出力を推定することが望ましい．提案モデルでは，Critic の状態評価関数，Actor の効用関数，および相手エージェント行動予測器として，正規化ガウス関数ネットワーク (NGnet)[15] と呼ばれる関数近似器を用いる．この近似器は，マルチエージェント系のような刻一刻とダイナミクスが変化するような環境においても，オンライン EM アルゴリズムを用いて学習ができる [16]．NGnet の入出力関係は $\boldsymbol{I}$ を入力の N 次元ベクトル，$\boldsymbol{O}$ を出力の N_a 次元ベクトルとして以下のように表せる．

$$\boldsymbol{O} = \sum_{i=1}^{M} (\boldsymbol{W}_i \boldsymbol{I} + \boldsymbol{b}_i) NG_i(\boldsymbol{I}) \tag{6}$$

$$NG_i(\boldsymbol{I}) = G_i(\boldsymbol{I}) / \sum_{j=1}^{M} G_j(\boldsymbol{I}) \tag{7}$$

$$G_i(\boldsymbol{I}) = (2\pi)^{-N/2} |\boldsymbol{\Sigma}_i|^{-1/2} \exp\left[-\frac{1}{2}(\boldsymbol{I} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\boldsymbol{I} - \boldsymbol{\mu}_i)\right] \tag{8}$$

ここで M は中間ユニットの数である．$\boldsymbol{\Sigma}_i$ は $N \times N$ の入力ベクトルの共分散行列を，$\boldsymbol{\mu}_i$ は N 次元の入力の平均値ベクトルを表す．また $\boldsymbol{W}_i$ は，$N_a \times M$ の重み行列であり，$\boldsymbol{b}_i$ は N_a 次元のバイアスベクトルである．関数の近似精度は，ユニット数 M に依存するが，ユニットの自動生成や消滅を行うことで適切な M を選択することが可能である [16]．

NGnet を Critic の状態評価関数に用いた場合は観測状態 x_t が入力となり，それに対する $V(x_t)$ が出力となる．また Actor における行動評価器あるいは行動予測器として用いた場合は，観測状態 x_t と行動 a_t が入力となり，それに対して行動制御を行うための効用 $p(x_t, a_t)$ が出力される．

3 マルチエージェント系としての Hearts

提案するエージェントの強化学習モデルを検証するため，競争的状況にあるマルチエージェント系の例として，4 人のプレイヤーによるカードゲームである Hearts へ適用する．

3.1 Heartsのルール

本節では本論文で用いたHeartsのルールを簡単に説明する．

Heartsでは参加するプレイヤーは4名であり，通常のトランプカード52枚を用いる．カードの4つのマーク（スペード，ハート，クラブ，ダイヤ）をスートと呼び，各スート毎にA，K，Q，...，4，3，2の順で強さの順位付けがされている．スート間の優劣はない．カードを各プレイヤーに順番に配り，各プレイヤーはゲームの始めに13枚のカードを手札として持つ．次に各プレイヤーが自身の左隣（次のゲームでは向かい側，その次のゲームでは右隣，その次のゲームでは左隣，…）のプレーヤーへ3枚のカードを渡すことにより，カードの交換を行う．その後以下のルールに従い，順番に右回りに場へカードを出す．なお任意のプレイヤーから始めて全員が1枚ずつのカードを出すことをトリックと呼び，13トリックを連続して行うことで1ゲームが終了する．また一つのトリックで最初に出されるカードをリーディングカードと呼ぶ．

(ルール1) 第一トリックを除き，リーディングカードは前回のトリックの勝者が場に出す．

(ルール2) 第一トリックでは，クラブの2がリーディングカードとなり，このカードを持つプレイヤーから始める．

(ルール3) 各プレイヤーはリーディングカードと同じスートのカードを出さなければならない．

(ルール4) リーディングカードと同じスートのカードがない場合は，任意のカードを場に出すことができる．この場合そのゲームで初めてハートを出すことをbreaking heartsと呼ぶ．

(ルール5) 手札にハートしかない場合を除き，ハートのカードはbreaking heartsが起こるまで，リーディングカードとして出すことはできない．

(ルール6) 1トリック終了後，場札の中でリーディングカードと同じスートの最も強いカードを出したプレイヤーが，そのトリックの勝者となる．

(ルール7) 場札のうちハートは1点，スペードのQは13点としてトリックの勝者に罰点を加算する．

以上のルールにもとづき，1ゲーム終了時の罰点の合計により優劣を決める．罰点の合計が小さいほどよいものとする．

3.2 Heartsへの提案モデルの適用

以下のモデル化においては，既にカードの交換は済んでいるものと考える．Heartsに参加しているエージェントのうち，学習エージェントをM_L，それ以外のエージェントをM_A，M_B，M_Cで表す．系全体でみると，いずれかのエージェントがカードを場に出すことにより状態の遷移が観測されることになるが，実際に行動をとれるのは自分の手番の時だけである．したがってM_Lの手番にあたる状態にのみ注目し，状態系列を$s_0, s_1, \ldots, s_{13}$と表す．ここでs_tはM_Lが$(t+1)$番目のカードを出す直前の状態を表し，s_{13}はゲームの終了状態を表す．

Heartsにおいて，カードの所在はゲームが進行するにつれて明らかになってゆく．つまり，観測できない状態変数がゲームの進行に伴って減少する．それによって，中盤以降には観測できる状態xをもとに真の環境の状態sの分布$P(s|x)$を近似的に推定する事ができる．したがって式(4)の観測状態遷移確率を求めるためには，状態の遷移確率$P^{\phi}(s_{t+1}|s_t, a_t)$が分かればよい．遷移確率は，相手エージェント達の戦略が分かればゲームのルールから決まる．相手エージェントM_Zの戦略は，過去のゲームでのM_Zの観測状態とその時の行動との入出力関係から，関数近似器によって学習可能である．よって，中盤以降は提案したモデルを使って行動制御が行える．

しかし，ゲームの序盤では観測できない状態変数が多いため，観測状態遷移確率の推定精度が悪いことが予想される．そこで，ゲームの序盤，中盤，終盤においてActor内の戦略を選択的に変更する選択的Actor-Criticモデルを提案する．選択的Actor-Criticモデルでは，ゲーム中はCriticが一貫して状態評価

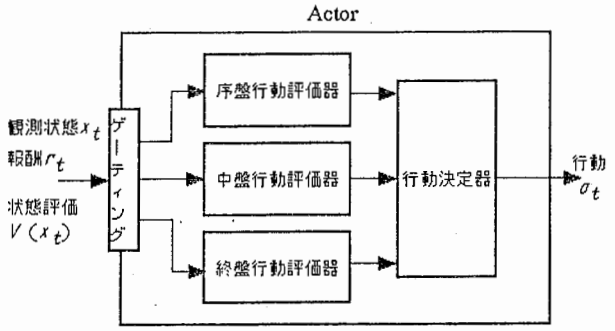

図 2: 選択的 Actor-Critic モデルにおける Actor 部の構成

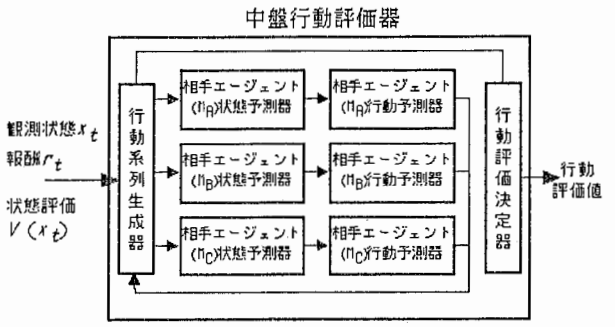

図 3: 中盤行動評価器の構成

を行う．この際，Critic における NGnet のパラメータは，時刻 t における入力 x_t に対する出力 $V(x)$ の学習ターゲットを $\{r_t + \gamma V(x_{t+1})\}$ として，オンライン EM アルゴリズム [16] によって更新する．一方，Actor は図 2 に示すように局面に応じて行動評価器を切り換え行動制御を行う．

3.2.1 序盤におけるモデル

序盤では相手エージェントの内部状態に対する情報がほとんど得られない．そこで，観測状態 x_t が得られた時行動 a_t をとることによって得られる効用 $U_{open}(x_t, a_t)$ を，序盤のために用意された NGnet により近似的に計算する．これを用いて以下の式で計算される行動確率 $P(a_t|x_t)$ にもとづき，行動を確率的に選択する．

$$P(a_t|x_t) = \frac{\exp(U_{open}(x_t, a_t)/T)}{\sum_{a_t \in A_t} \exp(U_{open}(x_t, a_t)/T)} \tag{9}$$

ここで T は温度パラメータである．効用関数の学習は，一般的な Actor-Critic アルゴリズムを用いる．時刻 t における入力 x_t と行動 a_t の組に対して，学習ターゲットを $U_{open}(x_t, a_t) + \alpha\delta_t$ として，NGnet のパラメータをオンライン EM アルゴリズムによって更新する．すなわち，式 (3) と同じである．

3.2.2 中盤におけるモデル

図 3 に中盤における行動評価器の構成を示す．状態 s_t から状態 s_{t+1} までの状態遷移に介在する相手エージェントの戦略を考慮した行動評価を行う．その際，真の状態の遷移ごとに相手の内部状態を推定する．図 4 に示すように，学習エージェントが状態 s_t において行動 a_t を選択したとき，次の状態 $s_{t+1} = s_t^{n_\tau}$ で再び自分の行動が回ってくるまでには，相手エージェント達の行動の系列 $a_t^\tau = [a_t^1, a_t^2, \cdots, a_t^{n_\tau - 1}]$ に依存して，状態遷移の経路 $\tau = [s_t^1, s_t^2, \cdots, s_t^{n_\tau}]$ が複数存在し得る．ここで $n_\tau - 1$ は各経路に介在した相手エー

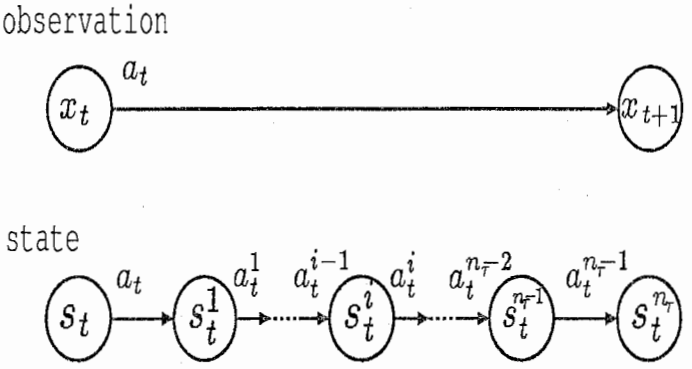

図 4: 真の状態遷移と観測状態遷移

ジェントの数である．介在するエージェントの数や順番は，リーディングカードを出すエージェントが誰になるかに依存する．s_t^1 は状態 s_t において学習エージェントが行動 a_t を選択した直後の状態を表す．直後にカードを出す相手エージェントは，独自の戦略を用いて行動 a_t^1 を選択することによって，次の状態 s_t^2 へと遷移する．以後，同様に繰り返し，再び自分の番の順番になったときの状態が $s_{t+1} = s_t^{n_\tau}$ となる．学習エージェントの観測状態 x_t から x_{t+1} への遷移の裏には，実際にはこのような状態遷移が起こっている．

s_t から s_{t+1} への状態遷移は，各エージェントの可能な行動によって分岐する木で表現する事ができ，深さ優先探索と適当な枝切りによって，効率よく探索できる．以下のアルゴリズムにしたがって，深さ優先探索を行い，各経路の状態遷移確率 $P^\phi(x_{t+1}|x_t, a_t)$，状態評価 $V(x_{t+1})$ と報酬 r_t を求め，最終的に期待 TD 誤差を計算する．

(a) トリック t での観測状態 x_t とエージェント達の行動 $a_t, a_t^1, \cdots, a_t^{i-1}$ から，図 3 の相手エージェント状態予測器によって状態 $\hat{s}_t^i$ の分布 $P(\hat{s}_t^i|x_t, a_t, a_t^1, \cdots, a_t^{i-1})$ を推定する．この推定については後述する．

(b) 状態の予測値 $\hat{s}_t^i$ に対して，図 3 の相手エージェント行動予測器によって相手エージェントの行動確率 $P(a_t^i|\hat{s}_t^i)$ を後述の式 (15) によって求める．

(c) 再び学習エージェントの順番が来るまで，(a)(b) の手続きを繰り返す．

(d) 学習エージェントの順番が来たときは，その経路 τ で得られた報酬 r_t と状態評価 $V(x_{t+1})$ を計算して，次の経路 τ' の探索を行う．

上記 (a)(b)(c)(d) から観測状態遷移確率は式 (10) によって求められる．

$$P^\phi(x_{t+1}|x_t, a_t) = \sum_{\tau \in \mathcal{T}} \sum_{a_t^1, \cdots, a_t^{n-1}} \prod_{i=1}^{n_\tau} P(\hat{s}_t^i|x_t, a_t, a_t^1, \cdots, a_t^{i-1}) P^\phi(a_t^i|\hat{s}_t^i) P(x_{t+1}|\hat{s}_t^{n_\tau}) \tag{10}$$

ここで，n_τ は再び学習エージェントの順番が回るまでに相手エージェント達を遷移した回数，また $\mathcal{T}$ は可能な状態遷移経路の集合を意味する．実際には，x_{t+1} は $\hat{s}_t^{n_\tau}$ から一意に決まるので，$P(x_{t+1}|\hat{s}_t^{n_\tau})$ は 1 である．なお式 (10) は式 (4) の特別な場合である．

式 (10) を用いて，トリック $(t+1)$ で得られる報酬の期待値と状態評価の期待値は，以下のようになる．

$$\langle r_t \rangle_{a_t} = \sum_{x_{t+1} \in X} P^\phi(x_{t+1}|x_t, a_t) r_t^{a_t} \tag{11}$$

$$\langle V(x_{t+1}) \rangle_{a_t} = \sum_{x_{t+1} \in X} P^\phi(x_{t+1}|x_t, a_t) V(x_{t+1}) \tag{12}$$

学習エージェントは，自分にとって最も利益があると期待される行動を選択するべきである．行動評価決定器において，M_L の得られる効用 $U^L_{mid}(x_t,a_t)$ は，行動 a_t を選択することによって得られる期待 TD 誤差を用いて以下のように計算される．

$$U^L_{mid}(x_t,a_t) = \langle r_{t+1}\rangle_{a_t} + \gamma\langle V(x_{t+1})\rangle_{a_t} - V(x_t) \tag{13}$$

行動決定は式 (14) により確率的に行う．

$$P(a_t|x_t) = \frac{\exp(U^L_{mid}(x_t,a_t)/T)}{\sum_{a_t}\exp(U^L_{mid}(x_t,a_t)/T)} \tag{14}$$

ここで T は温度パラメータである．

$\hat{s}^i_t$ の分布推定は，以下の二種の知識を用いて M_Z があるカード C を持つ確率 $P(M_Z,C)$ を推定することで行う．

1: ゲームのルール上の情報から、確実にあるカードを持っていたり、持っていなかったりする場合は、その結果を反映する。

たとえば、

- 他エージェントとのカードの交換によって、分かっている情報。
- leading スートを出せなかった時。

などは、エージェント j がカード C を持っている事が分かっている場合、その所持確率 $P_j(C) = 1$ であるし、持っていないことが分かっている場合、所持確率 $P_j(C) = 0$ である。

2: 情報が曖昧な場合は、各カード所持確率を近似的に計算する

出されたカードの履歴によって、各カードの所持確率は決定されるはずである。しかし、各エージェントの戦略に依存するために、正確な計算は行えない。ここでは近似的に、同じスートのカードについては同じ所持確率を持つとして計算を行う。

各状態で分かっている情報は、

- スート i カードのうち情報が曖昧なカードの枚数 N^C_i
- 自分がスート i について配られた枚数 N^I_i
- 相手エージェント j が現状態までにスート i を出した枚数 B_{ij}
- 各スートについての残りのカードは何であるか。
- スート i を持つ相手エージェントの数 N^A_i

などである。これらの情報から、まず、相手エージェント N^A_i 人にスート i のカードが $M_i = (N_i - N^I_i)$ 枚分配される。そして、相手エージェント j がスート i のカード n 枚を持っている確率は、

$$P_{ij}(n) = \frac{M_i!}{(M_i - B_{ij} - n)!(n + B_{ij})!}\left(\frac{1}{N^A_i}\right)^{n+B_{ij}}\left(\frac{N^A_i - 1}{N^A_i}\right)^{M_i - B_{ij} - n} \tag{15}$$

で与えられる。

これから、スート i を持つ確率 P^{all}_{ij} も求めることが出来る。

$$P^{all}_{ij} = \sum_{n}^{M_i - B_{ij}} P_{ij}(n) \tag{16}$$

また、エージェント j がスート i のカード C_i を持つ確率はそれぞれ等確率であるとし、

$$P_{ij}(C_i) = \sum_{n}^{M_i - Bij} P_{ij}(n) \frac{n}{M_i - B_{ij}} \tag{17}$$

となる。

相手エージェントの行動評価器が，予測状態 $\hat{s}_t^i$ を入力としたときに，各行動 a_t^i の効用 $U_{mid}^Z(\hat{s}_t^i, a_t^i)$ を出力するように NGnet を用いて学習させる．ここでは，経路 τ の i 番目のプレイをする相手エージェント M_Z の効用関数を考える．効用関数の学習は，1ゲーム終了ごとに状態 s_t^i から一意に定まる相手エージェントの観測状態 H_Z を入力，その時の a_t^i を出力ターゲットとしてオンライン EM アルゴリズムを用いて更新を行う．

効用 $U_{mid}^Z(\hat{s}_t^i, a_t^i)$ から，相手エージェントの行動確率 $P(a_t^i|\hat{s}_t^i)$ は，

$$P(a_t^i|s_t^i) = \frac{\exp(U_{mid}^Z(\hat{s}_t^i, a_t^i)/T_Z)}{\sum_{a_t^i} \exp(U_{mid}^Z(\hat{s}_t^i, a_t^i)/T_Z)} \tag{18}$$

と推定する．ここで T_Z は相手エージェント M_Z の温度パラメータである．

3.2.3 終盤におけるモデル

終盤においては探索空間が狭まるため，全探索により最も良い行動を選択する．しかし，マルチエージェント系では各エージェントの利害関係や戦略が複雑に絡み合っているため，最良性を一意に定義することは難しい．最良の行動を選択する基準としては以下のようなものが考えられる．

(a) 効用を最大にする行動を最良とする．

(b) 最悪報酬が最小の行動を最良とする．

(c) 平均報酬を最大にする行動を最良とする．

本研究においては (a) の基準を採用する．

4 計算機実験

4.1 相手状態予測実験

中盤における相手エージェントの状態予測の効果を検証するため，簡単な実験を行った．まず，ゲームのルールに則してランダムなプレーを行い，対象となるエージェントが前回のトリックでの勝者である状態を作り出す．そのため今回のトリックでは，エージェントが出すカードがリーディングカードとなる．エージェントは，リーディングカードを出すときに，そのリーディングカードを出すことによって得られる予測期待報酬 $\langle \hat{r} \rangle$ を計算する．そして，トリック終了時に実際の報酬 r を得る．相手エージェントは所持しているカードからランダムにカードを選択するものとする．以下の二つのエージェントを用いて，40000 回実験を行った．

エージェント 1 3.2.2 節における (a)(b) を考慮してカードの所在確率を予測し，その存在確率に比例して各エージェントが出すと予測するモデル．

エージェント 2 現在見えていないカードの中でそのトリックで出すことが可能なカードを，各エージェントが等確率で出すと予測する，つまり相手の手の内を予測しないモデル．

表 1: 相手エージェントの状態予測を行った場合と行わなかった場合の正規化平均二乗誤差

trick	NMSE with other agents' state inference	NMSE without other agents' state inference
2	0.797058	0.852562
3	0.810739	0.978432
4	0.833514	0.91074
5	0.823951	0.924043
6	0.836233	0.956581
7	0.864077	1.01079
8	0.881718	1.05967
9	0.86458	1.08681
10	0.968697	1.26099
11	1.12188	1.50487
12	1.3336	1.81433

各トリックにおいて予測期待報酬 $\langle \hat{r} \rangle$ と実際に得られた報酬 r との正規化平均二乗誤差を，上記のエージェントについて比較したものが表 1 である．

表 1 より、エージェント 1 の相手の手の内を予測したモデルの方が，エージェント 2 の予測しないモデルと比較して誤差が小さくなることが分かる．また，トリックが増えるにつれて状態予測の効果が大きくなる傾向にあるが．これは次第に相手の状態を知る手がかりが増えることが理由であると思われる．この結果は，3.2.2 節における真の状態の予測が有効であることを示している．

4.2 Hearts による実験

提案した学習モデルを持つ学習エージェント 1 体と，ルールベースの相手エージェント 3 体を使って実験を行った．

観測状態は，52 次元のベクトル $x = [x_1, x_2, \cdots, x_{57}]$ で表現する．最初の 52 次元は各カードの状態表現で，「過去のトリックで出てしまっている場合」を -1，「自分が現在所持している場合」を 1，「それ以外の場合」を 0 と表現する．残りの 5 次元は，現在のトリック，リーディングカードから数えて何番目に出すか，リーディングカード，現在場に出ているリーディングカード以外のカード等の情報を表現する．各次元には，スートはクラブ，ダイヤ，スペード，ハートの順に，数字は 2 から A の順にカードを割り当てる．つまり，x_1 はクラブの 2，x_2 はクラブの 3，$\cdots x_{13}$ はクラブの A，$\cdots x_{14}$ はダイヤの 2，$\cdots$ の状態を表現する．たとえば，ダイヤの 2 を所持している場合は $x_{14} = 1$，スペードの Q がすでに場に出ている場合は $x_{37} = -1$ となる．

学習エージェントの状態評価器，序盤行動制御器，3 つの相手エージェント行動予測器をそれぞれ 1 つ，計 5 つの NGnet が関数近似を行う．状態評価器は，57 次元の観測状態 x_t を入力とし，1 次元の状態評価 $V(x_t)$ を出力する．また，序盤行動評価器は観測状態 x_t を入力とし，学習エージェントの行動 a_t の効用 $U_{open}(a_t, x_t)$ を出力する．行動 a_t の効用は NGnet により，52 枚のカードすべてについて出力される．つまり，57 次元入力 52 次元出力の関数近似を行う．相手エージェント行動予測器は，推定された相手エージェントの状態 $\hat{s}_t^i$ を入力として，行動 a_t^i の効用 $U_{mid}^Z(\hat{s}_t^i, a_t^i)$ を出力する．各 NGnet には，それぞれ初期ユニットを 50 個与えた．

推定相手状態 $\hat{s}_t^i$ は，学習エージェントの状態表現と同様に 57 次元のベクトルで表現される．ただし，各

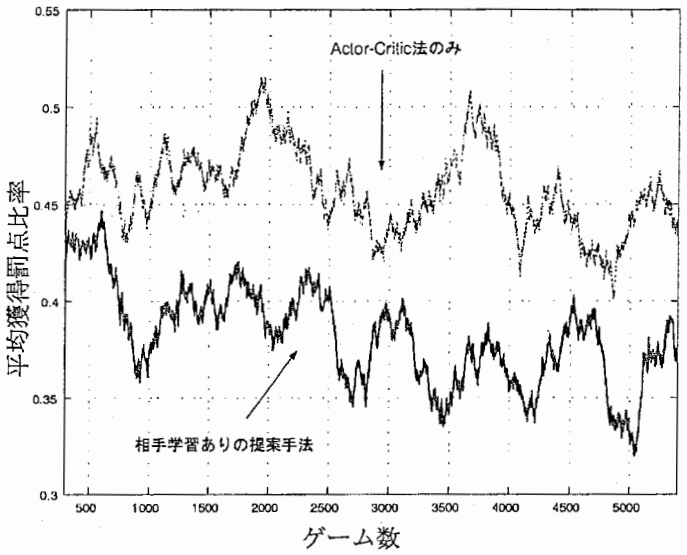

図 5: 300 回平均した学習エージェント獲得罰点比率

カードの状態表現は,「過去のトリックで出てしまっている場合」を -1,「それ以外の場合」はカードの予測所持確率 $P(M_Z,C)$ で与える．この時，確実に所持している場合は 1 に，確実に持っていない場合は 0 になる．

実験に用いたルールベースの相手エージェントは 80 以上のルールを持ち，熟練レベルの強さを持つ．持札からランダムにカードを出すだけのエージェントが，このルールベースのエージェント 3 体と対戦した場合の獲得罰点比率は，0.47 であった．獲得罰点比率は，4 体のエージェントの合計罰点に対する学習エージェントの獲得罰点の比率である．つまり，ランダムエージェントはルールベースエージェントの 2.6 倍程度の罰点を平均して獲得している．図 5 において Actro-Critic 法のみの方は，行動制御器の中盤モデルを用いずにすべて序盤モデルを用いて学習を行った結果であり，相手学習ありの提案手法の方は，トリック 6 から中盤モデルを用いて学習を行った場合の結果である．序盤モデルにおける Actro-Critic アルゴリズムでは環境のダイナミクスによる知識を用いてないのに対し，提案手法では相手エージェントのモデルを用いて環境のダイナミクスを学習している．さらに提案手法では現在得られている知識を用いて観測できない変数の推定を行っているので，Actro-Critic 法のみの手法と比べより忠実に環境のモデル化を行っていると言える．図 5 において，相手学習なしの Actro-Critic 法は平均獲得罰点比率がランダムエージェントと同程度であるが，提案手法によると平均的に約 0.05 の獲得罰点比率の改善が見られる．この差は環境のダイナミクスを学習を中盤モデルに取り入れたことにより効果であると考えられる．さらにゲーム数が増えるにつれ，Actro-Critic 法のみの手法と提案手法の両者とも学習により平均獲得罰点比率が改善していくことが観測されるが，学習の効果がより現れているのがわかる．これは，提案手法における相手行動の学習の枠組みが他の関数近似器とは独立に行われており，学習速度の遅い状態評価 $V(x)$ の学習に依存している序盤モデルのみの場合よりも，比較的早い学習が可能であるためと考えられる．

5 むすび

本論文では，マルチエージェント系として非協力 n 人ゲーム（$n > 2$）であるカードゲーム Hearts を取り上げ，部分的に非観測な状態においても自らの持つ環境モデルを強化学習により更新し，より多くの報酬を得るようにふるまうエージェントを設計した．学習モデルとしては，Actor-Critic 強化学習アルゴリズムに NGnet を用いている．さらに学習エージェントはゲームの局面に応じて複数の戦略を選択的に変更する．特に中盤の戦略は相手エージェントモデルを内包することで，マルチエージェント系における部分

観測性に取り組むことが可能であることを示した．

将来的にエージェントは様々な場面で計算機システムとユーザの中間に介在し，ユーザの思考を支援していくものと考えられるが，その場合状態が全て観測できるわけではなく，エージェント自らが環境モデルを持ち，利用可能な知識にもとづき，学習によって環境の同定を行うようなアプローチは今後ますます重要になっていくものと考えられる．特にユーザが計算機システムを使う時に，ユーザの意図を把握した上でシステムの利用を支援するエージェントは身近な例である．この場合にもエージェントが場面に応じて選択的に切り換えられるユーザモデルを切り換え，かつそれらを学習により適合させていく本アプローチは利用可能と考えられる．

参考文献

[1] 村上国男, "マルチエージェントシステムとその応用," 信学誌, 78(6), 570-577, 1995.

[2] 山村雅幸, 宮崎和光, 小林重信, "エージェントの学習," 人工知能学会誌, 10(5), 683-689, 1995.

[3] 三上貞芳, "強化学習のマルチエージェント系への応用," 人工知能学会誌, 12(6), 845-849, 1997.

[4] 石田亨, 片桐恭弘, 桑原和宏, "分散人工知能," コロナ社, 東京, 1996.

[5] Tan, M."Multi-agent reinforcement learning: independent vs. cooperative agents." *Proceedings of the Tenth International Conference on Machine Learning*, 330-337, 1993.

[6] Nagayuki, Y., Ishii, S., and Doya, K."Multi-agent reinforcement learning: an approach based on the other agent's internal model." *Proceedings of the fourth International Conference on MultiAgent Systems*, 215-221, 2000.

[7] Littman, M.L., "Markov games as a framework for multi-agent reinforcement learning," *Proceedings of the Eleventh International Conference Machines Learning*, 157-163, 1994.

[8] Yoshioka, T., Ishii, S., and Ito, M., "Strategy acquisition for game Othello based on min-max reinforcement learning," *IEICE Transactions on Information and Systems*, E82-D(12), 1618-1626, 1999.

[9] Pérez-Uribe, A,. and Sanchez, A., "Blackjack as a test bed for learning strategies in neural networks." *Proceedings of the IEEE International Joint Conference Neural Networks*, 3, 2022-2027, 1998.

[10] Tesauro, G., "TD-Gammon, a self-teaching Backgammon program, achieves master-level play," *Neural Computation*, 6(2), 215-219, 1994.

[11] 坂口実, "ゲームの理論," 森北出版, 東京, 1969.

[12] 浅田稔, "強化学習の実ロボットへの応用とその課題," 人工知能学会誌, 12(6), 834-836, 1997.

[13] Singh, S.P.,Jaakkola, T., and Jordan, M.I, "Learning without state-estimation in partially observable Markovian decision process," *Proceedings of the Eleventh International Conference on Machine Learning*, 284-292. 1994.

[14] Barto, A.G., Sutton, R.S., and Anderson, C.W., "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems Man and Cybernetics*, 13, 834-846, 1983.

[15] Moody, J., and Darken, C.J., "Fast learning in networks of locally-tuned processing units," *Neural Computation*, 1(2), 281-294, 1989.

[16] Sato, M., and Ishii, S., "On-line EM algorithm for the normalized Gaussian network," *Neural Computation*, 12(2), 407-432, 2000.

付録１，学習エージェントのプログラム Qlearnagent (v1.0)と使い方について

１，学習エージェントプログラムソースについて

学習エージェントプログラム testtpage は、従来のテスト用のエージェントプログラムを拡張した構成になっています。また、このエージェントプログラム実行の際には、３つの共有リンクライブラリを用いています。コンパイルは maketesttpage を参考に Makefile を作成して行って下さい。共有リンクライブラリの参照パスなどが環境によって変える必要があると思います。

学習エージェントをハーツの選択エージェントに加える方法については、別の資料（実験手順書）を参照して下さい。

１－１、プログラムソースファイル

プログラムのソースファイルは、/Qlearnagent/ディレクトリの下にあります。以下に列挙します。

testtpagent.c	学習エージェントのメインプログラム。
Qlearn.c	学習エージェントのゲームプレイ用関数を集めたソース
log_read.c	過去のログを読み Ngnet の初期化を行う。
agent_sock.c	以下は、他のエージェントプログラムと共通なので、他の資料を参照。
agent_ruledata.c	
agent_txtgrp.c	
agent_util.c	
agent_widget.c	
tp_card.c	
tp_disp.c	
tp_graph.c	
tp_mainwin.c	
tp_rule.c	
Qlearn.h	学習エージェントのゲームプレイ用関数のヘッダ
log_read.h	過去のログを読み Ngnet の初期化のためのヘッダ

agent_common.h	以下は他のエージェントプログラムと共通なので、他の資料を参照。
agent_data.h	
tp_global.h	
agent_sock.h	
agent_ruledata.h	
agent_txtgrp.h	
agent_util.h	
agent_widget.h	
tp_card.h	
tp_disp.h	
tp_graph.h	
tp_mainwin.h	
tp_rule.h	

１－２、共有リンクライブラリについて

必要な共有リンクライブラリ

libNGnetOEM.so	NGnet のオンライン EM アルゴリズムによる学習用ライブラリ
libMatrix.so	行列演算用ライブラリ
libRandom.so	乱数精製用ライブラリ

これらの共有リンクライブラリをあらかじめ作っておく必要があります。
以下、一応 Makefile によって gmake コマンドでコンパイルできるようにはなっていますが、環境によるのでパスなどを書き換える必要があると思います。gmake の無い環境では、Makefile を make 用に書き換える必要があります。

１－２－１、libRandom.so について

/Qlearnagent/Random/にある
Random.h
Random.c
を共有リンクライブラリとして libRandom.so コンパイルして下さい
一応 Makefile は gmake コマンドでコンパイルできるようにはなっていますが、環境によるのでパスなどを書き換える必要があると思います。

１－２－２，libMatrix.so について

/Qlearnagent/Matrix/にある
Matrix.h
Matrix.c
MatrixUtil.h
MatrixUtil.c
Main.c　　test 用
を共有リンクライブラリ libMatrix.so としてコンパイルして下さい。
一応 Makefile は gmake コマンドでコンパイルできるようにはなっていますが、環境によるのでパスなどを書き換える必要があると思います。

１－２－３、libNGnetOEM.so について

使用に当たって必要なファイル
NGnetOEM.h
NGnetOEM.c
Bool.h
Matrix.h
MatrixUtil.h
LibMatrix.so
LibRandom.so

ソースファイルについては/Qlearnagent/NGnetOEM/にあるので共有リンクライブラリ libNGnetOEM.so としてコンパイルしてください。一応 Makefile は gmake コマンドでコンパイルできるようにはなっていますが、環境によるのでパスなどを書き換える必要があると思います。

2、パラメータについて

この学習エージェントを実験に用いる際に、必要となるのがパラメータの設定です。主に設定すべきパラメータは学習に関するパラメータで、その中で特に設定に注意が必要なものが NGnet に関するものです。NGnet のオンライン学習は、その能力と学習速度をうまく引き出すためには、パラメータをうまく設定する必要があります。

以下では、パラメータの意味を説明します。

- **BETA　　(0<BETA<1)**

Qlearn.h にあります。序盤 Actor の学習係数です。大きいほど actor が早く学習を行いますが、critic の学習よりも早いとエージェントが不安定になるので、あまり大きくすべきではありません。

- T　　(0 < T < ∞)

Qlearn.h に global 変数として定義されています。Actor の行動決定、予測の際の温度パラメータになっています。基本値は 1.0 ですが、大きいほどランダムか行動選択を行います。小さいほど決定論的に行動を選択します。現在のプログラムでは、スケジューリングを行っています。

次に、NGnet のオンライン学習の初期化パラメータの設定です。NGnetOEM.h にある NGnet の初期化関数のヘッダーを使って簡単に説明します。詳しい内容など詳細は論文[16]を参照して下さい。NGnet の初期化関数は、以下のような関数になっています。

```
NGnetOEM * ngnet_oem_init( unsigned long iInputs, unsigned long iHidden, unsigned long iOutputs,
                           double iEtaInit, double iLambdaInit, unsigned long iRoundTime,
                           double iEtaReduceFactor,
                           double iPCreateLog, double iPDelete,
                           double iAlpha,
                           double iBeta1, double iBeta2,
                           double iMinEta, double iMinXVar, double iMinSigma2,
                           unsigned long iSample, const double ** iSampleInputs, const
double ** iSampleOutputs);
```

パラメータ設定に必要な変数を説明します。

- iLambdaInit　　(0 < iLambdaInit < 1)

NGnet は Online 学習の忘却係数。各 NGnet は 1/(1- iLambdaInit)のステップについての記憶を持っている。ハーツの場合、1 ゲームについて１３回学習を行うので、100 ゲームを見るなら 0.999 前後、1000 ゲームを見るなら 0.9999 前後の値を取ればよいことになる。ただし、このパラメータ値は初期値の設定で、その値はスケジューリングによって次第に１に近づく。

- iRoundTime　　(1 < iRoundTime)

忘却係数のスケジューリングは、この iRoundTime ごとに起こしている。毎ステップごとに少しずつ変化させるよりもしばらくは止めておく方が良いというシミュレーション実験

結果が出ているため、このパラメータを導入している。もちろん、毎ステップごとに変化させたい場合には、この値を１に設定すればよい。

- iEtaReduceFactor　　(0 < iEtaReduceFactor <1)

忘却係数のスケジューリング変数で、この値が１に近いほど忘却係数はより早く１に近づく。よってより長いタイムステップの情報を保持する要になるまでのステップが早くなる。

- iPCreateLog　　(iPCreateLog <0)

NGnet のユニットの生成に関するパラメータで、この値が大きいほどユニットを生成しやすくなる。この値を大きくすると対数スケールでユニットを生成しやすくなる。

- iPDelete　　(0 < iPDelete <1)

NGnet のユニットの消滅に関するパラメータで、この値が大きいほどユニットを消滅しやすくなる。

- iAlpha　　(0 < iAlpha <1)

NGnet のユニットに対する regularization に関するパラメータで、大きいほどユニットが象徴するガウス関数の分散がなまされるが、計算の安定性が上がる。小さい値にすると計算が発散し易くなる。

- iSample

NGnet を初期化する際の初期化サンプルの数。
NGnet を初期化する際にはサンプルデータが必要となる。これはユニットに配置やその他パラメータの値をうまく初期化しないと発散してしまうためである。NGnetOEM のライブラリでは、サンプルデータを k-means 法によってクラスタリングしたものを初期化に用いている。

- iSampleInputs
- iSampleOutputs

NGnet を初期化する際の初期化サンプルの入出力データ

３，使用前の準備

/Qlearnagent/ディレクトリの下に test4 というバイナリーファイルがありますが、このファイルは、NGnet 初期化の際のサンプルデータとして使用します。実行の際に、Hearts/server/LOG/ディレクトリの下に移して下さい。

A Multi-Agent Reinforcement Learning Method for a Partially-Observable Competitive Game

Yoichiro Matsuno[1] Tatsuya Yamazaki[2] Jun Matsuda[3] Shin Ishii[1]

[1] Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma-shi, Nara 630-0101
(e-mail)yoichi-m@is.aist-nara.ac.jp
[2] ATR Adaptive Communications Research Lab .
Department1
Kyoto, 619-0288
[3] Osaka Gakuin university
2-36-1 Kishibe-Minami Suita-shi Osaka

Abstract This article proposes a reinforcement learning (RL) method based on the Actor-Critic architecture, which can be applied to partially-observable multi-agent competitive games. As an example, we consider a card game "Hearts". In the Hearts game, cards held by the other players and the strategy of the other players are not directly observable by the learning agent. The RL then becomes a partially-observable Markov decision process (POMDP). However, the card distribution becomes inferable from the disclosed information as a single game proceeds. In addition, the strategy (model) of the other players can be learnable from their actual plays by repeating games. In our method, a single Hearts game is divided into three stages, and three actors are prepared so that one of them plays and learns separately in each stage. In particular, the actor for the middle stage plays so as to enlarge the expected temporal-difference (TD) error, which is calculated using the evaluation function approximated by the critic and the estimated state transition. Although the state transition is not well-defined in a POMDP, it can be estimated by taking the inferred card distribution and the other player's models into account. If the state transition is well estimated, the problem becomes a Markov decision process (MDP), and hence the RL method works well. We conduct an experiment to evaluate our method. After a learning player trained by our RL method plays several thousands training games with three heuristic players, the RL player becomes strong enough to beat the heuristic players.

1 Introduction

In recent years, computer systems have come to be mutually connected via networks and the importance has emerged for multiple users and autonomous agents to handle problems for cooperation and competition. Such telecommunication systems as a whole can be considered as multi-agent systems. Creating databases of inferable examples is one idea to solve such multi-agent problems. However, this approach has difficult due to the dynamic changes in the environments and the enormous number of possible states. According to reinforcement learning, on the other hand, it is expected that the whole problem can be solved by individual agents, in which each agent carries out local optimization in response to the dynamic environments. The reinforcement learning (RL) is such a machine learning method that the learning agent adapts to the environment based on rewards received when certain actions are taken.

The RL in a multi-agent system can be divided into (1) cooperation system and (2) compromise/competition system, with respect to payoffs for action consequences. The former corresponds to a case where all agents have the same payoff function; typical examples are pursuit problems [1][2]. In the latter case in contrast, the estimation of the actions of other agents is important, because the problems are qualitatively more difficult than the former. As an existing study, Littman [3] proposed a min-max Q-learning method, which was a modification of the Q-learning, in a zero-sum game for two agents. Since the objectives of the two agents become opposites in a zero-sum game, the profit of one agent is always disadvantage of the other agent. In the min-max Q-learning method, the problem is reduced to a single-agent problem, by using the assumption that the opponent agent executes the action that minimizes the profit of the considering agent.

Games are typical instances of the competitive mult-agent problems, and there have been studies applying RL methods to actual two-player games like Othello [4], Blackjack [5], and Backgammon [6]. In non-cooperative games whose number of players is more than two, however, there are few RL applications to strategy acquirement, because the inference of states and strategies of opponent players involves difficulty.

In this article, we propose an RL method that proceeds to improve one's strategy in response to the environment ,i.e., the other players, which can be applied to multi-agent competitive games. As a multi-agent competitive game, i.e., a non-cooperative n-player game ($n > 2$), we deal in particular with "Hearts", which is a four-player card game. Our method employs an Actor-Critic architecture [8]. The critic carries out evaluation on the states of the players, and the actor performs action controls in response to the states. More concretely, a single game of Hearts is divided into three phases: early phase, middle phase, and final phase. For the three phases, we prepare different actors, and they conduct appropriate controls by estimating opponent states and action models. We conduct an experiment to evaluate our method. After a learning player trained by our RL method plays several thousands training games with three heuristic players, the RL player becomes strong enough to beat the heuristic players.

2 Model

In a multi-agent system, each agent autonomously carries out action selection. The environment can be formulated as a system including the considering agent ("agent") itself and all of the other agents. This environment, however, changes over time according to the learning of the other agents. If the internal states of the other agents are not observable by the agent, the problem for the agent becomes partial observation. In the Hearts game we consider, the cards delivered to the opponents players constitute partial observation. In such a situation, we aim to construct a model in which the agent learns a strategy so as to do his best to achieve his own objective. In the following sections, an observable state at time $t \in \{0, 1, 2, \ldots\}$ is denoted by $x_t \in \mathcal{X}$, the action taken by the considering agent at that time is denoted by $a_t \in \mathcal{A}$,

and the reward obtained by taking the action is denoted by $r_t = r(a_t, x_t) \in \mathcal{R}$. Here, $\mathcal{X}$ is the set of all observable states, $\mathcal{A}$ shows the set of possible actions of the agent, and $\mathcal{R}$ is a real number space. Although $\mathcal{A}$ differs dependent on each state, in the following, we neglect the dependence when there is no ambiguity.

2.1 Actor-Critic algorithm

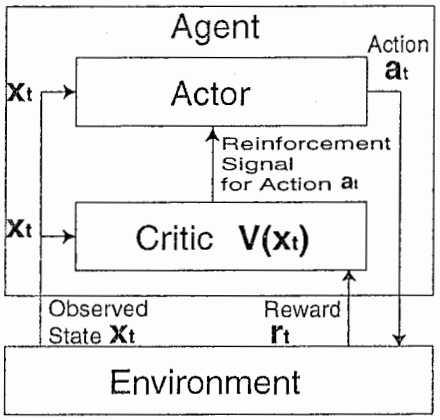

図 1: Actor-Critic architecture

When the strategies and internal states of opponent agents cannot be observed, it is important to measure them from experience. Even though the learning of the opponent agents is ignored, the problem is known to be a partially observable Markov decision process (POMDP), due to the opponents' unobserved states. In a POMDP, a way to probabilistically learn its own policy is often important. Even for a very simple POMDP, it is actually known that a deterministic policy is necessarily worse than a probabilistic policy [7]. As such a learning method, we propose a method that combines opponent policy (model) learning with the Actor-Critic method [8].

In our Actor-Critic algorithm shown in Fig. 1, the learning agent has an actor and a critic. The critic evaluates the goodness of the current observed state x_t, which is represented by the value function $V(x_t)$, in the environment including the agent and the other opponent agents. The actor carries out an action control a_t for the current observed state x_t.

After the actor emits an action control a_t, the observed state changes into x_{t+1}. After that, the critic receives a reward as a result of the state change, and calculates the TD error δ_t shown by Eq. (1) below using the new value function $V(x_{t+1})$.

$$\delta_t = \{r_t + \gamma V(x_{t+1})\} - V(x_t) \tag{1}$$

Here, $V(x_t)$ is the value function for the current observed state x_t, which is approximated by the critic. $\gamma (0 \leq \gamma \leq 1)$ is a discount rate. The TD error is the difference between the old evaluation value $V(x_t)$ for the current state x_t evaluated at time t and the new evaluation value $r_t + \gamma V(x_{t+1})$ for x_t evaluated at time $t+1$. With the general Actor-Critic algorithm, both of the value function $V(x)$ approximated by the critic and the strategy represented by the actor are updated by using the TD error, as follows:

$$V(x_t) \leftarrow V(x_t) + \alpha \delta_t \tag{2}$$

$$p(x_t, a_t) \leftarrow p(x_t, a_t) + \alpha \delta_t \tag{3}$$

Here, α is a learning coefficient, $p(x_t, a_t)$ is a merit function that represents the goodness of action a_t in state x_t. The actor determines an action based on the merit function. Specific details will be described in Section 3.2.

With the Actor-Critic algorithm, the probability of actions is changed via the merit function by using the TD error which shows, as a consequent of a single state transition, whether the current action is good or bad with respect to the current evaluation function. In multi-agent problems we consider, the environment includes unobservable factors, namely, internal states and strategies of the opponents. In such a problem, the learning based on the TD error is not effective, because a single state transition in the observed state space consists of a lot of possible state transitions in the actual state space. Namely, the unobservable factors make the state transitions complicated and intertwined.

In this study, we determine the agent actions based on the state transition probability $P^{\phi}(x_{t+1}|x_t, a_t)$ from the current observed state x_t to the next observed state x_{t+1} under the consid-

eration of the unobservable dynamics of the environment. Here, ϕ denotes the dynamics of the environment unable to be observed from the learning agent, namely, the internal states of the opponent agents.

This observed state transition probability satisfies

$$P^{\phi}(x_{t+1}|x_t, a_t) = \sum_{s_t \in \mathcal{S}_t} \sum_{s_{t+1} \in \mathcal{S}_{t+1}} P(s_t|x_t) \cdot P^{\phi}(s_{t+1}|s_t, a_t) \cdot P(x_{t+1}|s_{t+1}), \quad (4)$$

where $\mathcal{S}_t$ or $\mathcal{S}_{t+1}$ denotes the set of possible true states that correspond to the observed state x_t or x_{t+1}, respectively. "True states" mean the states including both of the observed state and the unobserved state like the internal states of opponents. $P(x_{t+1}|s_{t+1})$ denotes the conditional probability that the observed state x_{t+1} is obtained in state s_{t+1}. Especially when x_{t+1} is the actual observed state, this conditional probability comes to have a value of 1, because the observed state x is uniquely determined from state s. $P^{\phi}(s_{t+1}|s_t, a_t)$ denotes the transition probability from a true state s_t to a true state s_{t+1} when action a_t is selected under the dynamics of the environment, ϕ. It should be noted that ϕ in the right hand side of equation (4) does not include the internal states of the opponent agents but includes the strategies of the opponent agents. In addition, $P(s_t|x_t)$ is the probability that the true state is s_t for the observed state x_t

In order to obtain the observed state transition probability (4), we need to know $P(s_t|x_t)$ and true state transition probability $P^{\phi}(s_{t+1}|s_t, a_t)$. Since the true state transition probability $P^{\phi}(s_{t+1}|s_t, a_t)$ depends on the strategies of the opponent agents, the probability can be approximated by learning of the models of the opponent agents. In addition, depending on the problem, we can approximate the probability $P(s_t|x_t)$. Further details are discussed in Section 3.2.

We here define the expected TD error $\langle \delta_t \rangle_a$ with respect to the observed state transition (4) as follows.

$$\langle \delta_t \rangle_a = \{\langle r_t \rangle_a + \gamma \langle V(x_{t+1}) \rangle_a\} - V(x_t) \quad (5)$$

Here, $\langle \cdot \rangle_a$ denotes the expected value with respect to $P^{\phi}(x_{t+1}|x_t, a_t)$. When the dynamics of the environment is given by ϕ, an action having large expected TD error implies an action for which the possibility that a large reward will be obtained over the future is large under the environmental dynamics. Accordingly, the action selection based on the expected TD error value is considered to be appropriate.

2.2 Stochastic function approximator and its learning

Since it is difficult to experience every state especially when the state space is large, it is desirable to estimate by some way the values of necessary function (e.g.,the value function or merit function) for an unknown state. Generalization from the function values for the known states using a function approximator is one such way. In our RL method, we use function approximators called normalized Gaussian networks (NGnets) [9] for the value function of the critic, the merit function of the actor, and action inference devices for the opponent agents. These approximators are trained [10] by the on-line EM algorithm, which provides an appropriate learning framework in dynamic environments like a multi-agent environment. The output relationship of By an NGnet, an N_a-dimensional output vector $\boldsymbol{O}$ for an N-dimensional input vector $\boldsymbol{I}$ is given as follows.

$$\boldsymbol{O} = \sum_{i=1}^{M} (\boldsymbol{W}_i \boldsymbol{I} + \boldsymbol{b}_i) NG_i(\boldsymbol{I}) \quad (6)$$

$$NG_i(\boldsymbol{I}) = G_i(\boldsymbol{I}) / \sum_{j=1}^{M} G_j(\boldsymbol{I}) \quad (7)$$

$$G_i(\boldsymbol{I}) = (2\pi)^{-N/2} |\Sigma_i|^{-1/2} \cdot \exp\left[-\frac{1}{2}(\boldsymbol{I} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\boldsymbol{I} - \boldsymbol{\mu}_i)\right] \quad (8)$$

Here, M is the number of units. N-by-N matrix Σ_i and N-dimensional vector $\boldsymbol{\mu}_i$ are the covariance matrix and the mean vector, respectively, for the

i-th unit. N_a-by-M matrix W_i is the linear regression matrix and $\boldsymbol{b}_i$ denotes the N_a-dimensional bias vector. Although the approximation ability of the NGnet depends on the number of units M, it is possible to select an appropriate M by carrying out the dynamic production and deletion mechanisms of units [10].

When an NGnet approximates the value function of the critic, the input is the observed state x_t and the target output is $V(x_t)$. When an NGnet approximates the merit function of the actor and the action evaluation function (it is actually the merit function of the opponent agents) of the inference devises, the input is the pair of the observed state x_t and action a_t, and the target output is the merit value $p(x_t, a_t)$.

3 Hearts as a Multi-agent System

In order to evaluate the applicability of our RL method to a multi-agent competitive system, it is applied to the Hearts game which is a card game played by four players.

3.1 Rules of Hearts

Here, we briefly explain the rules of Hearts, which is used in our study.

The game Hearts is played by four players and use the ordinary 52 cards. There are four suits, i.e., spades, hearts, diamonds, and clubs, and there is an order of strength within each suit (i.e., A, K, Q,..., 2). There is no strength order among the suits. Cards are distributed to each player in order, and each player has 13 cards at the beginning of the game as his hand. After that, each player exchanges cards by handing three cards to the player on his left hand. Thereafter, according to the rules below, each player plays a card clock-wisely in order. When each of the four players plays a card, it is called a trick. When a trick begins a card played by a player, the card is called the leading card and the player is called the leading player. A single game ends when 13 tricks are carried out one after another.

Rule 1) Except for the first trick, the winner of the current trick is the leading player of the subsequent trick.

Rule 2) In the first trick, the two of clubs comes to be the leading card, implying that the player holding this card is the leading player.

Rule 3) Each player must play a card of the same suit with the leading card.

Rule 4) If a player does not have a card of the same suit with the leading card, he can play any card. When a heart card is in such a case played at the first time of a single game, the play is called "breaking hearts".

Rule 5) Until the breaking hearts occurs, the leading player may not play a heart card. If the leading player has no suits without hearts, it is a exceptional case and the player may lead a heart card.

Rule 6) After a trick, the player that has played the strongest card of the same suit with the leading card is the winner of that trick.

Rule 7) A heart card has one point and the Q of spade has 13 points. The winner of a trick receives all of the points of the cards played in the trick.

According to the rules above, a single game is processed, and at the end of the single game, the score of each player is determined as the sum of the received points. The smaller the score, the better.

3.2 Application of the proposed model to Hearts

In this section, we apply our RL method to the game Hearts. In the followings, it is assumed that the exchange of cards has already been completed. Among the agents participating in Hearts, the learning agent is denoted by M_L, and the other agents are denoted by M_A, M_B, and M_C. Although the true state transition occurs at each play of a single agent the action of the learning agent can be taken only at his turn. Therefore, we focus only on the states for the turns of M_L, and the state series are described by $s_0, s_1, \cdots s_{13}$. Here, s_t de-

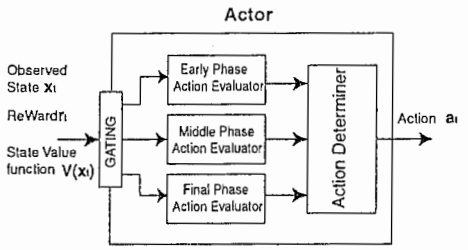

図 2: Actor Architecture in the Selective Actor-Critic Model

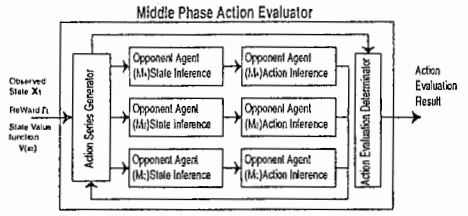

図 3: Architecture of Action Evaluator in the Middle Phase

notes the true state just before M_L plays a card in his $(t+1)$-th turn, and s_{13} denotes the final state of the game.

In Hearts, the unobserved card distributions become disclosed as a single game proceeds. That is, the unobservable state variables decrease as the game proceeds. This fact implies that it is possible to estimate the distribution $P(s|x)$, which denotes the probability of the true state s in the condition of the observed state x. In order to determine the observed state transition probability using Eq. (4), it is then necessary to know the transition probability $P^{\phi}(s_{t+1}|s_t, a_t)$ for the true state. If it is possible to estimate the strategies of the opponent agents, the true transition probability is automatically determined by the game rules. The strategy of an opponent agent M_Z can be learned by a function approximator from input/output relationships in the past games. In a past game, the state observed by M_Z and his play can be definitely determined for each turn of M_Z, in a backward fashion from the final game state according to the game rule. When the learning becomes accurate, the action taken by M_Z can be estimated using the function approximator.

In the early phase of a single game, however, it is conceivable that the estimation accuracy of the observed state transition probability is poor, since the unobserved state variables are larger than the observed state variables. In such a case, the accuracy of the expected TD error (5) will be poor. Therefore, this study employs a selective Actor-Critic model to selectively switch one of the actors that correspond to the early, middle, and final phases of a single game. In the selective Actor-Critic model, the critic carries out consistent state evaluations during the whole single game. Namely, there is a unique critic that approximates the value function for every state in a single game. The NGnet for the critic is trained by the on-line EM algorithm [10]; the target output for input x_t is given by $\{r_t + \gamma V(x_{t+1})\}$. On the other hand, according to the game phase, the actor is selected out of three actors, and the selected actor performs action controls, as shown in Fig. 2.

3.2.1 Model in the early phase

In the early phase of a single game, little information can be obtained on the internal states of the opponent agents. Therefore, the actor for the early phase calculates merit $U_{open}(x_t, a_t)$ for each action a_t in the observed state x_t. This calculation is done by the NGnet prepared for the early phase. Using the merit function, the actor selects an action according to the action probability $P(a_t|x_t)$ defined by the following equation.

$$P(a_t|x_t) = \frac{\exp(U_{open}(x_t, a_t)/T)}{\sum_{a_t \in A_t} \exp(U_{open}(x_t, a_t)/T)}, \qquad (9)$$

where, T is the temperature parameter that determines the randomness of the actor. This learning method of the merit function is the same as that in the conventional Actor-Critic algorithm. The actor NGnet is trained by the on-line EM algorithm, where the target output is given by $U_{open}(x_t, a_t) + \alpha\delta_t$, for the pair of input x_t and action a_t. This learning is identical to that given by Eq. (3).

3.2.2 Model in the middle phase

Fig. 3, shows the construction of the actor in the middle phase. The action evaluator carries out action evaluations considering the estimation of plays of the opponent agents that intervene in the state transition between state s_t and state s_{t+1}. It estimates the internal states of the opponents for each transition of the true state. As shown in Fig. 4, after the learning agent selects an action for state s_t, until the turn of the agent comes again in the next state $s_{t+1} = s_t^{n_\tau}$, the possible true state transition is dependent on the possible action sequence $a_t^\tau = [a_t^1, a_t^2, \cdots, a_t^{n_\tau - 1}]$ that consist of actions taken by the opponent agents. According to the possible action sequence, there are a lot of possible paths for the true state transition $\tau = [s_t^1, s_t^2, \cdots, s_t^{n_\tau}]$.

Here, $n_\tau - 1$ is the number of opponent agents intervening in a single path. The number and order of intervening agents depend on who is the leading agent. s_t^1 denotes the state just after the learning agent selects action a_t in state s_t. After that, the next (opponent) agent takes an action a_t^1 (,i.e., plays a card) using its own strategy, and the true state transits to the next state s_t^2. This repeats in the same manner, and the turn of the learning agent comes again as $s_{t+1} = s_t^{n_\tau}$. Behind the transition from state x_t to x_{t+1} observed by the learning agent such a complicated state transition in the space of the true state variables. The possible state transitions from state s_t to s_{t+1} can be expressed by a tree whose branch corresponds to an action taken by each individual agent. The tree can then be efficiently retrieved based on the depth priority search and appropriate branch cutting heuristics.

According to the following algorithm, we carry out the depth priority search retrieving the state transition tree. Each retrieval of the transition tree obtains the state transition probability $P^\phi(x_{t+1}|x_t, a_t)$, value function $V(x_{t+1})$, and reward r_t for the transition path. By achieving the retrieval of the tree, the expected TD error is calculated.

(a) Estimate distribution $P(\hat{s}_t^i|x_t, a_t, a_t^1, \cdots, a_t^{i-1})$ of state $\hat{s}_t^i$ with the opponent agent state inference in Fig. 3, from observed state x_t for trick t and action $a_t, a_t^1, \cdots, a_t^{i-1}$ of the opponent agent. We will explain this estimation later.

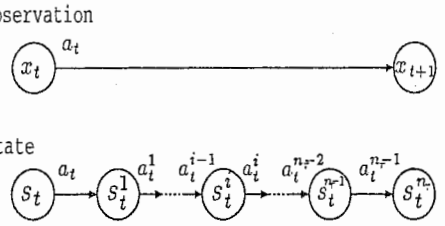

図 4: Real state transition and observable state transition

(b) Search for action probability $P(a_t^i|\hat{s}_t^i)$ of the opponent agent with the action inference in Fig. 3, for inferred value $\hat{s}_t^i$ of the state, by Eq. (15) (shown later).

(c) Repeat steps (a) and (b) until the turn of the learning agent comes again.

(d) Calculate the reward r_t and state value function $V(x_{t+1})$ obtained with the τ when the turn of the learning agent comes, and then carry out a search for the next path τ'.

The observed state transition probability is obtained from Eq. (10), from (a), (b), (c), and (d) above.

$$P^\phi(x_{t+1}|x_t, a_t) = \cdot \sum_{\tau \in \mathcal{T}} \sum_{a_t^1, \cdots, a_t^{n-1}} \prod_{i=1}^{n_\tau} P(\hat{s}_t^i|x_t, a_t, a_t^1, \cdots, a_t^{i-1}) \cdot P^\phi(a_t^i|\hat{s}_t^i) P(x_{t+1}|\hat{s}_t^{n_\tau}) \quad (10)$$

Here, n_τ is the number of turns transiting the opponent agents until the turn of the learning agent comes again, and $\mathcal{T}$ indicates the set of possible state transition paths. Actually, $P(x_{t+1}|\hat{s}_t^{n_\tau})$ is 1, since x_{t+1} is uniquely determined from s_t. Incidentally, Eq. (10) is a special case of Eq. (4).

By using Eq. (10), the expected value of the reward and the expected value of the state evaluation obtained at trick $t+1$ become as follows.

$$\langle r_t \rangle_{a_t} = \sum_{x_{t+1} \in X} P^\phi(x_{t+1}|x_t, a_t) r_t^{a_t} \quad (11)$$

$$\langle V(x_{t+1})\rangle_{a_t} = \sum_{x_{t+1}\in X} P^{\phi}(x_{t+1}|x_t, a_t) \cdot V(x_{t+1}) \quad (12)$$

The learning agent should select an action that is expected if it believes it can obtain the most profit. In the action evaluation determinator, utility $U^L_{mid}(x_t, a_t)$ obtained from M_L is calculated as follows using the expected TD error obtained by selecting action at.

$$U^L_{mid}(x_t, a_t) = \langle r_{t+1}\rangle_{a_t} + \gamma\langle V(x_{t+1})\rangle_{a_t} - V(x_t\rangle \quad (13)$$

The action determination is probabilistically carried out by Eq. (14).

$$P(a_t|x_t) = \frac{\exp(U^L_{mid}(x_t, a_t)/T)}{\sum_{a_t}\exp(U^L_{mid}(x_t, a_t)/T)} \quad (14)$$

Here, T is the temperature parameter.

The distribution estimation of $\hat{s}^i_t$ is carried out by estimating probability $P(M_Z, C)$ with which M_Z has some card C by using the following two kinds of knowledge.

(a) For example, if the fact was observed that M_Z did not take out a card of the same suit as the leading card in a previous trick, the probability is assumed to be 0 that M_Z has a card of that suit.

(b) Focusing on the cards of some suit, only the number of remaining cards excluding the number of cards one himself possesses as his hand and the number of cards the opponent agents take out to that point can exist with the opponents.

By using these two kinds of knowledge, we carry out estimations of distributions by assuming the allocations of possible cards to be completely random.

The action evaluators of the opponent agents are made to learn by using the Ngnets so as to output utility $U^Z_{mid}(\hat{s}^i_t, a^i_t)$ of each action a^i_t when inferred state $\hat{s}^i_t$ is assumed to be the input. Here, we consider the utility function of opponent agent M_Z to perform the ith play of path τ. The learning of the utility function results in updating by using the online EM algorithm, assuming observed state H_Z of the opponent agent uniquely determined from state s^i_t at the end of each game to be the input, and a^i_t at that time to be the output target. From merit $U^Z_{mid}(\hat{s}^i_t, a^i_t)$, the action probability $P(a^i_t|\hat{s}^i_t)$ of the opponent agent is estimated as

$$P(a^i_t|s^i_t) = \frac{\exp(U^Z_{mid}(\hat{s}^i_t, a^i_t)/T_Z)}{\sum_{a^i_t}\exp(U^Z_{mid}(\hat{s}^i_t, a^i_t)/T_Z)} \quad (15)$$

Here, T_Z is the temperature parameter used for the estimation of the opponent action.

3.2.3 Model in the final phase

In the final phase, we select the best action based on an entire search, because the possible state transition space is tractable. However, it is difficult to uniquely define the best action, since the relationships and strategies of each agent are complexly intertwined in the multi-agent system. As standards for selecting the best action, the following can be considered.

(a) The action maximizing the merit is assumed to be the best.

(b) The action that maximizes the reward for the possible worst case is assumed to be the best.

(c) The action maximizing the average reward is assumed to be the best.

We adopt standard (a) in this study.

4 Computer Experiments

4.1 Opponent state inference experiments

The major novelty of our study is the actor for the middle phase. It conducts the state inference of the opponent agents. In order to evaluate the state inference, a simple experiment is done. First, we prepare states at which the considering agent is the leading player by random plays according to the game rules. Namely, the card that the agent plays is the leading cards for the current trick in

each of those states. The agent estimates the expected reward $\langle \hat{r} \rangle$ that will be recieved by playing the leading card. This estimation is compared with the actual reward r at the end of that trick. The opponent agents randomly select cards that they possess and can play according to the rules. By using the following two types of agents, 40000 different tricks are played.

Agent 1 An agent that estimates the probability of card distribution (corresponding to step (a) in Section 3.2.2) infers that each opponent agent will play according to the distribution probability.

Agent 2 An agent model that infers that each opponent agent will play a removable card with equal probability among the cards that have not seen at present, namely, an agent model that does not infer the opponent's hands.

Table 1 shows the normalized mean square error between the estimated reward $\langle \hat{r} \rangle$ and the actual reward r recieved in an individual trick, for the two agent models above.

From Table 1, it can be seen that the error for the model that infers the opponent's hands is smaller than that of the model that does not infer the opponent's hands. In addition, the effect of the state inference increases as the number of tricks enlarges. This may be due to the fact that the information on the unobserved states gradually increase as the single game proceeds. This result shows that the inference of unobserved states (opponent's hand) is effective to estimate the recieved reward.

4.2 Experiments with Hearts

We carried out experiments using one learning agent based on the proposed learning model and three rule-based opponent agents.

The observed state is expressed with 57-dimensional vector x. The first 52 dimensions are the state expressions of the individual cards; -1 denotes "has already played in a previous trick," 1 "when the agent owns in his hand," or 0 "for the other cases." The remaining five dimensions express information on such things as the present trick, who is the leading player in that trick, the leading card, the other cards that have been played in that trick, and so on. For the earlier 52 dimensions, a number is assigned to each card in the order of from 2 to A, in the suit order of clubs, diamonds, spades, and hearts. That is, x_1 is the 2 of clubs, x_2 is the 3 of clubs, ..., x_{13} is the A of clubs, x_{14} is the 2 of diamonds, and so on. For example, when the agent has the 2 of diamonds in his hand, $x_{14} = 1$. When the Q of spades has already been played, $x_{37} = -1$.

Five NGnet's are prepared for the function approximation: the value function (i.e., the critic) of the learning agent, the merit function (i.e., the actor) for the early phase, and action inference devices for the three opponent agents. For the value function, input x_t is a 57-dimensional vector, and outputs is one-dimensional value function $V(x_t)$. The merit function for the early phase, the init is state x_t, and the output is merit $U_{open}(a_t, x_t)$ for each action a_t. The merit function outputs the merit for all of the 52 cards. In other words, the NGnet for the merit function carries out function approximation whose input and output dimensions are 57 and 52, respectively. For the action inference device of each opponent agent, the input is the estimated state $\hat{s}_t^i$, and the output is the merit $U_{mid}^Z(a_t^i, \hat{s}_t^i)$ for action a_t. Each of the five NGnets has 50 units.

The estimated state $\hat{s}_t^i$ is expressed by a 57-dimensional vector in a similar way to the state expression of the learning agent. The state expression of each card is given by -1 when the card has been played in a previous trick, or in the other cases, it is given by the inferred possession probability $P(M_Z, C)$ where C denotes the card. Namely, the value is 1 when agent M_Z has definitely the card, and the values is 0 when agent M_Z has no chance to have the card.

On the other hand, the rule-based agent used in the experiments has more than 80 rules so that it is "experienced" level player of the game Hearts.

The acquired penalty ratio was 0.47 when an agent who only took out cards at random from its hand challenged the three rule-based agents. The acquired penalty ratio is the ratio of the acquired

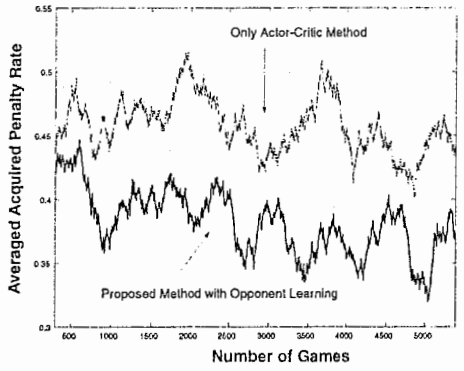

図 5: 300-times averaged penalty ratio acquired by the learning agent

penalty of the learning agent to the total penalty of the four agents. That is, a random agent acquires about a 2.6-fold penalty of rule-based agents on average.

In Fig. 5, the Actor-Critic method alone shows results obtained by carrying out learning using all of the early phase models (without using the middle phase models) of an action controller, and the proposed model with opponent learning shows results obtained by carrying out learning using the middle phase models from six tricks. With the Actor-Critic algorithm in early phase models, while knowledge via the dynamics of the environment is not utilized, with the proposed method, the dynamics of the environment is learned by using the models of the opponent agents. With the proposed method, moreover, it can be said that models of the environment are created as expected, from a comparison with the Actor-Critic method alone, since estimations are carried out on unobservable variables by using the knowledge presently obtained. In Fig. 5, although the averaged acquired penalty ratio for the Actor-Critic method without opponent learning would be about equivalent to a random agent, an improvement can be seen in the approximate acquired penalty ratio (on average) according to the proposed method. This difference can be considered to be due to an effect of the dynamics of the environment, from the adoption of learning into the middle phase models. It can also be observed that the averaged acquired penalty ratio proceeds to improve with learning with the increase in the number of games, for both the Actor-Critic method alone and the proposed method (even though it is understood that the effect of the learning does lead to further development). This can be considered to be due to the following: the framework of the learning of the opponent actions in the proposed method is carried out independently from other function approximators, and therefore there is the possibility of a comparatively quicker learning than the case of early phase models alone, which depend on the learning of state value function $V(x)$ at a slow learning speed.

5 Conclusion

In this paper, we adopted the card game Hearts, i.e., a non- cooperative n-player game ($n > 2$), as a multi-agent system, and designed agents to behave in such a way as to partially update the environment models they themselves maintain even in non- observable states and to obtain more rewards. As the learning model, we use NGnets in an Actor-Critic reinforcement learning algorithm. The learning agent selectively updates multiple strategies in response to aspects of the game. In particular, the strategies of the middle phase showed that it is possible to accommodate partial observations in a multi-agent system by including opponent agent models.

In the future, although agents can be expected to intervene between computer systems and users in various ways and to support the considerations of users, at such a time, approaches where the agents themselves maintain their own environment models (since all states will naturally be unable to be observed) to carry out the identification of the environment via learning based on the usable knowledge, can be expected to come to be increasingly more important. In particular, a familiar example is an agent supporting a user using a computer with the use of the system after grasping the intention of the user. It is considered that the proposed approach can be used even in this case, where the agent changes user models (that are changed selectively) in response to the aspect, and then proceeds

to make the models fit through learning.

REFERENCES

[1] Tan, M."Multi-agent reinforcement learning: independent vs. cooperative agents." *Proceedings of the Tenth International Conference on Machine Learning*, 330-337, 1993.

[2] Nagayuki, Y., Ishii, S., and Doya, K."Multi-agent reinforcement learning: an approach based on the other agent's internal model." *Proceedings of the fourth International Conference on MultiAgent Systems*, 215-221, 2000.

[3] Littman, M.L., "Markov games as a framework for multi-agent reinforcement learning," *Proceedings of the Eleventh International Conference Machines Learning*, 157-163, 1994.

[4] Yoshioka, T., Ishii, S., and Ito, M., "Strategy acquisition for game Othello based on min-max reinforcement learning," *IEICE Transactions on Information and Systems*, E82-D(12), 1618-1626, 1999.

[5] Pérez-Uribe, A,. and Sanchez, A., "Blackjack as a test bed for learning strategies in neural networks." *Proceedings of the IEEE International Joint Conference Neural Networks*, 3, 2022-2027, 1998.

[6] Tesauro, G., "TD-Gammon, a self-teaching Backgammon program, achieves master-level play," *Neural Computation*, 6(2), 215-219, 1994.

[7] Singh, S.P.,Jaakkola, T., and Jordan, M.I, "Learning without state-estimation in partially observable Markovian decision process," *Proceedings of the Eleventh International Conference on Machine Learning*, 284-292. 1994.

[8] Barto, A.G., Sutton, R.S., and Anderson, C.W., "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems Man and Cybernetics*, 13, 834-846, 1983.

[9] Moody, J., and Darken, C.J., "Fast learning in networks of locally-tuned processing units," *Neural Computation*, 1(2), 281-294, 1989.

[10] Sato, M., and Ishii, S., "On-line EM algorithm for the normalized Gaussian network," *Neural Computation*, 12(2), 407-432, 2000.

表 1: Normalized Mean Square Error(NMSE) with and without the state inference

trick	NMSE with other agents' state inference	NMSE without other agents' state inference
2	0.797058	0.852562
3	0.810739	0.978432
4	0.833514	0.91074
5	0.823951	0.924043
6	0.836233	0.956581
7	0.864077	1.01079
8	0.881718	1.05967
9	0.86458	1.08681
10	0.968697	1.26099
11	1.12188	1.50487
12	1.3336	1.81433

On-line EM Algorithm for the Normalized Gaussian Network

Masa-aki Sato † and Shin Ishii ††

† ATR Human Information Processing Research Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan
TEL: (+81)-774-95-1039 FAX: (+81)-774-95-1008
E-mail: masaaki@hip.atr.co.jp

†† Nara Institute of Science and Technology
8916-5 Takayama-cho, Ikoma-shi, Nara 630-0101, Japan

Abstract

Normalized Gaussian Network (NGnet) (Moody and Darken 1989) is a network of local linear regression units. The model softly partitions the input space by normalized Gaussian functions and each local unit linearly approximates the output within the partition.

In this article, we propose a new on-line EM algorithm for the NGnet, which is derived from the batch EM algorithm (Xu, Jordan and Hinton 1995) by introducing a discount factor. We show that the on-line EM algorithm is equivalent to the batch EM algorithm if a specific scheduling of the discount factor is employed. In addition, we show that the on-line EM algorithm can be considered as a stochastic approximation method to find the maximum likelihood estimator. A new regularization method is proposed in order to deal with a singular input distribution. In order to manage dynamic environments, where the input-output distribution of data changes with time, unit manipulation mechanisms such as unit production, unit deletion, and unit division are also introduced based on the probabilistic interpretation.

Experimental results show that our approach is suitable for function approximation problems in dynamical environments. We also applied our on-line EM algorithm to a reinforcement learning problem. It is shown that the NGnet, when using the on-line EM algorithm, learns the value function much faster than the method based on the gradient descent algorithm.

1 Introduction

Normalized Gaussian Network (NGnet) (Moody and Darken 1989) is a network of local linear regression units. The model softly partitions the input space by normalized Gaussian functions and each local unit linearly approximates the output within the partition. Since the NGnet is a local model, it is possible to change parameters of several units in order to learn a single datum. Therefore, the learning process becomes easier than that of the global models such as the multi-layered perceptron. In a local model, on the other hand, the number of necessary units grows exponentially as the input dimension increases, if one wants to approximates the input-output relationship over the whole input space. This results in a computational explosion, which is often called the "curse of dimensionality". However, actual data will often distribute in a lower dimensional space than the input space, such as in attractors of dynamical systems.

The NGnet, which is a kind of "Mixtures of Experts" model (Jacobs, Jordan, Nowlan and Hinton 1991 ; Jordan and Jacobs 1994), can be interpreted as an output of a stochastic model with hidden variables. The model parameters can be determined by the maximum likelihood estimation method. In particular, the EM algorithm for the NGnet was derived by Xu, Jordan and Hinton (1995). In this article, we propose a new on-line EM algorithm for the NGnet. The on-line EM algorithm is derived from the batch EM algorithm (Xu, Jordan and Hinton 1995) by introducing a discount factor. Although the derivation is straightforward, important modifications are necessary for practical applications. We will discuss these points in detail and propose a modified version of the on-line EM algorithm. We show that the on-line EM algorithm is equivalent to the batch EM algorithm if a specific scheduling of the discount factor is employed. In addition, we show that the on-line EM algorithm can be considered as a stochastic approximation method to find the maximum likelihood estimator. A new regularization method is proposed in order to deal with a singular input distribution. Unit manipulation mechanisms based on the probabilistic interpretation are also introduced in order to manage dynamic environments. These mechanisms are unit production, unit deletion, and unit division.

In order to investigate the performance of our new on-line EM algorithm, experiments for function approximation problems are conducted for three circumstances. The first one is a usual function approximation problem for a set of observed data. The second one is a function approximation in which the distribution of the input data changes with time. This experiment is performed to check the applicability of our approach to the dynamic environments. The third one is a function approximation in which the distribution of the input data is singular, i.e., the dimension of the input data distribution is smaller than the input space dimension. In this case, a straightforward application of the basic on-line EM algorithm does not work, because the covariance matrices used in the NGnet become singular. Our modified on-line EM algorithm is shown to work well even in this situation.

Encouraged by these positive results, we applied our on-line EM algorithm to a reinforcement learning problem. In the actor-critic model (Barto, Sutton and Anderson 1983), learning of the value function for a current policy can be regarded as a function approximation problem in a dynamic environment, since the policy changes with time as the learning proceeds. As an example, we examined a task for swinging-up an inverted pendulum (Doya 1996). The experimental result shows that the NGnet, when using our new on-line EM algorithm, learns the value function much faster than the method based on the gradient descent algorithm.

The paper is organized as follows. The NGnet and its stochastic model are explained in Section 2. The batch EM algorithm is introduced in Section 3. These sections are based on previous papers (Moody and Darken 1989; Xu, Jordan and Hinton 1995). The basic on-line EM algorithm is derived in Section 4. The modifications of the basic on-line EM algorithm are discussed in Sections 5, 6 and 7. The experimental results are shown in Section 8, and Section 9 sums up the paper.

2 Normalized Gaussian network

2.1 NGnet

The Normalized Gaussian Network (NGnet) model (Moody and Darken 1989), which transforms an N-dimensional input vector x to a D-dimensional output vector y, is defined by the following equations.

$$y = \sum_{i=1}^{M} (W_i x + b_i) \mathcal{N}_i(x) \tag{2.1a}$$

$$\mathcal{N}_i(x) \equiv G_i(x) / \sum_{j=1}^{M} G_j(x) \tag{2.1b}$$

$$G_i(x) \equiv (2\pi)^{-N/2} |\Sigma_i|^{-1/2} \exp\left[-\frac{1}{2} (x - \mu_i)' \Sigma_i^{-1} (x - \mu_i) \right] . \tag{2.1c}$$

M denotes the number of units, and the prime ($'$) denotes a transpose. $G_i(x)$ is an N-dimensional Gaussian function; its center is an N-dimensional vector μ_i and its covariance matrix is an $(N \times N)$-dimensional matrix Σ_i. $|\Sigma_i|$ is the determinant of the matrix Σ_i. $\mathcal{N}_i(x)$ is the i-th normalized Gaussian function. W_i and b_i are a $(D \times N)$-dimensional linear regression matrix and a D-dimensional bias vector, respectively. Subsequently, we use notations: $\tilde{W}_i \equiv (W_i, b_i)$, and $\tilde{x}' \equiv (x', 1)$. With these notations, (2.1a) is rewritten as

$$y = \sum_{i=1}^{M} \mathcal{N}_i(x) \tilde{W}_i \tilde{x} . \tag{2.2}$$

The Gaussian function $G_i(x)$ is a kind of radial basis function (Poggio and Girosi 1990). However, the normalized Gaussian functions, $\mathcal{N}_i(x)$ $(i = 1, ..., M)$, do not have a radial symmetry. They softly partition the input space into M regions. The i-th unit linearly approximates its output by $\tilde{W}_i \tilde{x}$ within the corresponding region. An output of the NGnet is given by a summation of these outputs weighted by the normalized Gaussian functions.

2.2 Stochastic model of NGnet

The NGnet (2.1) can be interpreted as a stochastic model, in which a pair of an input and an output (x, y) is a stochastic (incomplete) event. For each event, a single unit is assumed to be selected from a set of classes $\{i | i = 1, ..., M\}$. The unit index i is regarded as a hidden variable. A triplet (x, y, i) is called a complete event. The stochastic model is defined by the

probability distribution for a complete event (Xu, Jordan and Hinton 1995):

$$P(x,y,i|\theta) = (2\pi)^{-(D+N)/2}\sigma_i^{-D}|\Sigma_i|^{-1/2}M^{-1}\exp\left[-\frac{1}{2}(x-\mu_i)'\Sigma_i^{-1}(x-\mu_i) - \frac{1}{2\sigma_i^2}(y-\tilde{W}_i\tilde{x})^2\right], \tag{2.3}$$

where $\theta \equiv \{\mu_i, \Sigma_i, \sigma_i^2, \tilde{W}_i \mid i = 1,...,M\}$ is a set of model parameters. From the probability distribution (2.3), the following probabilities are obtained.

$$P(i|\theta) = 1/M \tag{2.4a}$$
$$P(x|i,\theta) = G_i(x) \tag{2.4b}$$
$$P(y|x,i,\theta) = (2\pi)^{-D/2}\sigma_i^{-D}\exp\left[-\frac{1}{2\sigma_i^2}(y-\tilde{W}_i\tilde{x})^2\right]. \tag{2.4c}$$

These probabilities define a stochastic data generation model. First, a unit is selected randomly with an equal probability (2.4a). If the i-th unit is selected, the input x is generated according to the Gaussian distribution (2.4b). Given the unit index i and the input x, the output y is generated according to the Gaussian distribution (2.4c), which has the mean $\tilde{W}_i\tilde{x}$ and the variance σ_i^2.

When the input x is observed, the probability that the output value becomes y in this model, turns out to be

$$P(y|x,\theta) = \sum_{i=1}^{M}\mathcal{N}_i(x)P(y|x,i,\theta). \tag{2.5}$$

From this conditional probability, the expectation value of the output y for a given input x is obtained as:

$$E[y|x] \equiv \int yP(y|x,\theta)dy = \sum_{i=1}^{M}\mathcal{N}_i(x)\tilde{W}_i\tilde{x}, \tag{2.6}$$

which is equivalent to the output of the NGnet (2.2). Namely, the probability distribution (2.3) provides a stochastic model for the NGnet.

3 EM algorithm

From a set of T events (observed data), $(\{x\},\{y\}) \equiv \{(x(t),y(t)) \mid t = 1,...,T\}$, the model parameter θ of the stochastic model (2.3) can be determined by the maximum likelihood estimation method. In particular, the EM algorithm (Dempster, Laird and Rubin 1977) can be applied to models having hidden variables. The EM algorithm repeats the following E-step and M-step. Since the likelihood for the set of observations increases (or does not change) after an E- and M-step, the maximum likelihood estimator is asymptotically obtained by repeating the E- and M-steps.

- E (Estimation) step
  Let $\bar{\theta}$ be the present estimator. By using $\bar{\theta}$, the posterior probability that the i-th unit is selected for each observation $(x(t),y(t))$ is calculated according to the Bayes rule.

$$P(i|x(t),y(t),\bar{\theta}) = P(x(t),y(t),i|\bar{\theta})/\sum_{j=1}^{M}P(x(t),y(t),j|\bar{\theta}). \tag{3.1}$$

- M (Maximization) step
  By using the posterior probability (3.1), the expected log-likelihood $Q(\theta|\bar{\theta},\{x\},\{y\})$ for the complete events is defined by

$$Q(\theta|\bar{\theta},\{x\},\{y\}) = \sum_{t=1}^{T}\sum_{i=1}^{M}P(i|x(t),y(t),\bar{\theta})\log P(x(t),y(t),i|\theta). \tag{3.2}$$

  On the other hand, the log-likelihood of the observed data $(\{x\},\{y\})$ is given by

$$L(\theta|\{x\},\{y\}) = \sum_{t=1}^{T}\log P(x(t),y(t)|\theta) = \sum_{t=1}^{T}\log\left(\sum_{i=1}^{M}P(x(t),y(t),i|\theta)\right). \tag{3.3}$$

  Since an increase of $Q(\theta|\bar{\theta},\{x\},\{y\})$ implies an increase of the log-likelihood $L(\theta|\{x\},\{y\})$ (Dempster, Laird and Rubin 1977), $Q(\theta|\bar{\theta},\{x\},\{y\})$ is maximized with respect to the estimator θ. A solution of the necessity condition $\partial Q/\partial\theta = 0$ is given (Xu, Jordan and Hinton 1995) by

$$\mu_i = \langle x\rangle_i(T)/\langle 1\rangle_i(T) \tag{3.4a}$$
$$\Sigma_i = \langle(x-\mu_i)(x-\mu_i)'\rangle_i(T)/\langle 1\rangle_i(T) = \langle xx'\rangle_i(T)/\langle 1\rangle_i(T) - \mu_i(T)\mu_i'(T) \tag{3.4b}$$
$$\tilde{W}_i\langle\tilde{x}\tilde{x}'\rangle_i(T) = \langle y\tilde{x}'\rangle_i(T) \tag{3.4c}$$
$$\sigma_i^2 = \frac{1}{D}\langle|y-\tilde{W}_i\tilde{x}|^2\rangle_i(T)/\langle 1\rangle_i(T) = \frac{1}{D}\left[\langle|y^2|\rangle_i(T) - \mathrm{Tr}\left(\tilde{W}_i\langle\tilde{x}y'\rangle_i(T)\right)\right]/\langle 1\rangle_i(T), \tag{3.4d}$$

  where $\mathrm{Tr}(\cdot)$ denotes a matrix trace. A symbol $\langle\cdot\rangle_i$ denotes a weighted mean with respect to the posterior probability (3.1) and it is defined by

$$\langle f(x,y)\rangle_i(T) \equiv \frac{1}{T}\sum_{t=1}^{T}f(x(t),y(t))P(i|x(t),y(t),\bar{\theta}). \tag{3.5}$$

If an infinite number of data, which are drawn independently according to an unknown data distribution density $\rho(x,y)$, are given, the weighted mean (3.5) converges to the following expectation value.

$$\langle f(x,y)\rangle_i(T) \stackrel{T\to\infty}{\longrightarrow} E[f(x,y)P(i|x,y,\bar{\theta})]_\rho, \tag{3.6}$$

where $E[\cdot]_\rho$ denotes the expectation value with respect to the data distribution density $\rho(x,y)$. In this case, the M-step equation (3.4) can be written as

$$g_i \equiv E[P(i|x,y,\bar{\theta})]_\rho \tag{3.7a}$$
$$\mu_i = E[xP(i|x,y,\bar{\theta})]_\rho/g_i \tag{3.7b}$$
$$\Sigma_i = E[xx'P(i|x,y,\bar{\theta})]_\rho/g_i - \mu_i\mu_i' \tag{3.7c}$$
$$\tilde{W}_iE[\tilde{x}\tilde{x}'P(i|x,y,\bar{\theta})]_\rho = E[y\tilde{x}'P(i|x,y,\bar{\theta})]_\rho \tag{3.7d}$$
$$\sigma_i^2 = \frac{1}{D}\left(E[|y|^2P(i|x,y,\bar{\theta})]_\rho - \mathrm{Tr}(\tilde{W}_iE[\tilde{x}y'P(i|x,y,\bar{\theta})]_\rho)\right)/g_i, \tag{3.7e}$$

where the new parameter set $\theta = \{\mu_i,\Sigma_i,\tilde{W}_i,\sigma_i^2|i = 1,...,M\}$ is calculated by using the old parameter set $\bar{\theta} = \{\bar{\mu}_i,\bar{\Sigma}_i,\bar{\tilde{W}}_i,\bar{\sigma}_i^2|i = 1,...,M\}$.

At an equilibrium point of this EM algorithm, the old and new parameters become the same, i.e., $\theta = \bar{\theta}$. The equilibrium condition of the EM algorithm, i.e., (3.7) along with $\theta = \bar{\theta}$, is equivalent to the maximum likelihood condition,

$$\partial L(\theta)/\partial\theta = 0, \tag{3.8}$$

where the log-likelihood function is given by

$$L(\theta) = E[\log \sum_{i=1}^{M} P(x, y, i|\theta)]_\rho. \tag{3.9}$$

4 On-line EM algorithm

The EM algorithm introduced in the previous section is based on a batch learning (Xu, Jordan and Hinton 1995), namely, the parameters are updated after seeing all the observed data $(\{x\}, \{y\})$. In this section, we derive an on-line version of the EM algorithm. Since the estimator is changed after each observation, let $\theta(t)$ be the estimator after the t-th observation $(x(t), y(t))$. In the on-line EM algorithm, the weighted mean (3.5) is replaced by:

$$\ll f(x,y) \gg_i (T) \equiv \eta(T) \sum_{t=1}^{T} \lambda^{T-t} f(x(t), y(t)) P(i|x(t), y(t), \theta(t-1)) \tag{4.1a}$$

$$\eta(T) \equiv (\sum_{t=1}^{T} \lambda^{T-t})^{-1} = (1-\lambda)/(1-\lambda^T). \tag{4.1b}$$

Here, the parameter λ $(0 < \lambda < 1)$ is a discount factor, which is introduced for forgetting the effect of the old posterior values employing the earlier inaccurate estimator. $\eta(T)$ is a normalization coefficient and plays a role like a learning rate. The modified weighted mean $\ll \cdot \gg_i$ can be obtained by the step-wise equation:

$$\ll f(x,y) \gg_i (t) = \ll f(x,y) \gg_i (t-1) + \eta(t) \left[f(x(t), y(t)) P_i(t) - \ll f(x,y) \gg_i (t-1) \right], \tag{4.2}$$

where $P_i(t) \equiv P(i|x(t), y(t), \theta(t-1))$. After calculating the modified weighted means, i.e., $\ll 1 \gg_i (t)$, $\ll x \gg_i (t)$, $\ll xx' \gg_i (t)$, $\ll |y|^2 \gg_i (t)$, and $\ll y\tilde{x}' \gg_i (t)$, according to (4.2), the new estimator $\theta(t)$ is obtained by the following equations.

$$\mu_i(t) = \ll x \gg_i (t) / \ll 1 \gg_i (t) \tag{4.3a}$$

$$\Sigma_i^{-1}(t) = [\ll xx' \gg_i (t) / \ll 1 \gg_i (t) - \mu_i(t)\mu_i'(t)]^{-1} \tag{4.3b}$$

$$\tilde{W}_i(t) = \ll y\tilde{x}' \gg_i (t) [\ll \tilde{x}\tilde{x}' \gg_i (t)]^{-1} \tag{4.3c}$$

$$\sigma_i^2(t) = \frac{1}{D} \left[\ll |y|^2 \gg_i (t) - \mathrm{Tr} \left(\tilde{W}_i(t) \ll \tilde{x}y' \gg_i (t) \right) \right] / \ll 1 \gg_i (t). \tag{4.3d}$$

This defines the on-line EM algorithm. In this algorithm, calculations of the inverse matrices are necessary at each time step. A recursive formula for Σ_i^{-1} and $\tilde{W}_i$ can be derived, in which there is no need to calculate the matrix inverse, by using a standard method (Jürgen 1996).

Let us define the weighted inverse covariance matrix of $\tilde{x}$: $\tilde{\Lambda}_i(t) \equiv (\ll \tilde{x}\tilde{x}' \gg_i (t))^{-1}$. This quantity can be obtained by the step-wise equation:

$$\tilde{\Lambda}_i(t) = \frac{1}{1-\eta(t)} \left[\tilde{\Lambda}_i(t-1) - \frac{P_i(t)\tilde{\Lambda}_i(t-1)\tilde{x}(t)\tilde{x}'(t)\tilde{\Lambda}_i(t-1)}{(1/\eta(t)-1) + P_i(t)\tilde{x}'(t)\tilde{\Lambda}_i(t-1)\tilde{x}(t)} \right]. \tag{4.4}$$

$\Sigma_i^{-1}(t)$ can be obtained from the following relation with $\tilde{\Lambda}_i(t)$.

$$\tilde{\Lambda}_i(t) \ll 1 \gg_i (t) = \begin{pmatrix} \Sigma_i^{-1}(t) & -\Sigma_i^{-1}(t)\mu_i(t) \\ -\mu_i'(t)\Sigma_i^{-1}(t) & 1 + \mu_i'(t)\Sigma_i^{-1}(t)\mu_i(t) \end{pmatrix}. \tag{4.5}$$

The estimator for the linear regression matrix, $\tilde{W}_i$, is given by

$$\tilde{W}_i(t) = \ll y\tilde{x}' \gg_i (t) \tilde{\Lambda}_i(t) \tag{4.6a}$$

$$= \tilde{W}_i(t-1) + \eta(t) P_i(t) (y(t) - \tilde{W}_i(t-1)\tilde{x}(t)) \tilde{x}'(t) \tilde{\Lambda}_i(t). \tag{4.6b}$$

Thus, the basic on-line EM algorithm is summarized as follows. For each observation $(x(t), y(t))$, the weighted means, i.e., $\ll 1 \gg_i (t)$, $\ll x \gg_i (t)$, $\ll |y|^2 \gg_i (t)$, and $\ll \tilde{x}y' \gg_i (t)$, are calculated by using the step-wise equation (4.2). $\tilde{\Lambda}_i(t)$ is also calculated by the step-wise equation (4.4). Subsequently, the estimators for the model parameters are obtained by (4.3a), (4.3d), (4.5), and (4.6b).

5 Scheduling of discount factor

5.1 Time-dependent discount factor

In the previous section, we assumed that the discount factor λ is a constant. If this is the case, the estimators obtained by the on-line EM algorithm and the batch EM algorithm differ from each other due to the presence of λ in (4.1). Here, we assume that λ is a function of the time t, so that λ^{T-t} in (4.1a) is replaced by $\Pi_{s=t+1}^{T} \lambda(s)$. The normalization coefficient $\eta(T)$ (4.1b) is redefined by

$$\eta(T) \equiv \left(\sum_{t=1}^{T} \prod_{s=t+1}^{T} \lambda(s) \right)^{-1}. \tag{5.1}$$

It is calculated by the step-wise equation:

$$\eta(t) = (1 + \lambda(t)/\eta(t-1))^{-1}. \tag{5.2}$$

There is no need to redefine the step-wise equations (4.2), (4.4) and (4.6b), if the effective learning coefficient $\eta(t)$ is calculated by (5.2).

The constraint, $0 \leq \lambda(t) \leq 1$, gives the constraint on $\eta(t)$:

$$1 \geq \eta(t) \geq 1/t. \tag{5.3}$$

If this constraint is satisfied, the equation (5.2) can be solved for $\lambda(t)$ as:

$$\lambda(t) = \eta(t-1)(1/\eta(t) - 1). \tag{5.4}$$

5.2 Equivalence between on-line and batch EM algorithms

In the next part, we show that the on-line EM algorithm defined by (3.1), (4.2) and (4.3) is equivalent to the batch EM algorithm defined by (3.1), (3.4) and (3.5), with an appropriate choice of $\lambda(t)$.

It is assumed that the same set of data, $\{(x(t), y(t))|t = 1, ..., T\}$, is repeatedly supplied to the learning system, i.e., $x(t+T) = x(t)$ and $y(t+T) = y(t)$. The time t is represented by an epoch index k $(= 1, 2, ...)$ and a data number index m $(= 1, ..., T)$ within an epoch, i.e., $t = (k-1)T + m$. Let us assume that the model parameter θ is updated at the end of each epoch, i.e., when $m = T$, and it is denoted by $\theta(k)$. Let us suppose that $\lambda(t)$ is given by

$$\lambda(t) = \begin{cases} 0 & \text{if } t = (k-1)T+1 \\ 1 & \text{otherwise} \end{cases}. \tag{5.5}$$

The corresponding $\eta(t)$ is given by $\eta((k-1)T+m) = 1/m$. Then, the weighted mean $\ll f(x,y) \gg_i (t)$ is initialized to $f(x(1), y(1))P(i|x(1), y(1), \theta(k-1))$ at the beginning of an epoch. At the end of an epoch, $\ll f(x,y) \gg_i (kT) = \langle f(x,y) \rangle_i(T)$ is satisfied for $\bar{\theta} = \theta(k-1)$. This shows that the on-line EM algorithm together with $\lambda(t)$ given by (5.5) is equivalent to the batch EM algorithm. It should be noted that the step-wise equation (4.4) for $\bar{\Lambda}_i(t)$ can not be used in this case, since the equation (4.4) becomes singular for $\eta(t) = 1$.

5.3 Stochastic approximation

If an infinite number of data, which are drawn independently according to the data distribution density $\rho(x, y)$, are available, the on-line EM algorithm can be considered as a stochastic approximation (Kushner and Yin 1997) for obtaining the maximum likelihood estimator, as demonstrated below.

Let ϕ be a compact notation for the weighted mean, i.e., $\phi(t) \equiv \{\ll 1 \gg_i (t), \ll x \gg_i (t), \ll xx' \gg_i (t), \ll y\bar{x}' \gg_i (t), \ll |y|^2 \gg_i (t) | i = 1, ..., M\}$. The on-line EM algorithm, (4.2) and (4.3), can be written in an abstract form:

$$\delta\phi(t) \equiv \phi(t) - \phi(t-1) = \eta(t)[F(x(t), y(t), \theta(t-1)) - \phi(t-1)] \tag{5.6a}$$

$$\theta(t) = H(\phi(t)). \tag{5.6b}$$

It can be easily proved that the set of equations:

$$\phi = E[F(x, y, \theta)]_\rho \tag{5.7a}$$

$$\theta = H(\phi) \tag{5.7b}$$

is equivalent to the maximum likelihood condition (3.8). Then, the on-line EM algorithm can be written as

$$\delta\phi(t) = \eta(t)(E[F(x, y, H(\phi(t-1)))]_\rho - \phi(t-1)) + \eta(t)\zeta(x(t), y(t), \phi(t-1)), \tag{5.8}$$

where the stochastic noise term ζ is defined by

$$\zeta(x(t), y(t), \phi(t-1)) \equiv F(x(t), y(t), H(\phi(t-1)) - E[F(x, y, H(\phi(t-1)))]_\rho, \tag{5.9}$$

and it satisfies

$$E[\zeta(x, y, \phi)]_\rho = 0. \tag{5.10}$$

The equation (5.8) has the same form as the Robbins-Monro stochastic approximation (Kushner and Yin 1997), which finds the maximum likelihood estimator given by (5.7). The effective learning coefficient $\eta(t)$ should satisfy the condition

$$\eta(t) \overset{t\to\infty}{\longrightarrow} 0, \quad \sum_{t=1}^{\infty} \eta(t) = \infty, \quad \sum_{t=1}^{\infty} \eta^2(t) < \infty. \tag{5.11}$$

Typically, $\eta(t)$, which satisfies the conditions (5.3) and (5.11), is given by

$$\eta(t) \overset{t\to\infty}{\longrightarrow} \frac{1}{at+b} \quad (1 > a > 0). \tag{5.12}$$

The corresponding discount factor is given by

$$\lambda(t) \overset{t\to\infty}{\longrightarrow} 1 - \frac{1-a}{at+(b-a)}, \tag{5.13}$$

namely, $\lambda(t)$ is increased such that $\lambda(t)$ approaches 1 as $t \to \infty$.

For the convergence proof of the stochastic approximation (5.8), the boundedness of the noise variance is necessary (Kushner and Yin 1997). The noise variance is given by

$$E[\zeta(x, y, \phi)^2]_\rho = E[F(x, y, H(\phi))^2]_\rho - E[F(x, y, H(\phi))]_\rho^2. \tag{5.14}$$

Both terms on the right hand side in (5.14) are finite, if we assume that the data distribution density $\rho(x, y)$ has a compact support. Consequently, the noise variance becomes finite under this assumption. This assumption is not so restrictive, because actual data always distribute in a finite domain. We can weaken this assumption such that $\rho(x, y)$ decreases exponentially as $|x|$ or $|y|$ goes to infinity.

It should be noted that the stochastic approximation (5.8) for finding the maximum likelihood estimator is not a stochastic gradient ascent algorithm for the log-likelihood function (3.9). The on-line EM algorithm (5.8) is faster than the stochastic gradient ascent algorithm, because the M-step is solved exactly. We have so far used the on-line EM algorithm defined by (4.2) and (4.3), which is equivalent to the basic on-line EM algorithm defined by (4.2), (4.3a), (4.3d), (4.4), (4.5), and (4.6b), if $\eta(t) \neq 1$. Therefore, the basic on-line EM algorithm is also a stochastic approximation.

In practical applications, the learning coefficient $\eta(t)$ given by (5.12) becomes too small for a very large t, such that the parameters could not be changed significantly in a finite period. In order to avoid this situation, $\eta(t)$ is often set at a small positive value η_{min} for a very large t. This clamp corresponds to $\lambda(t) = 1 - \eta_{min}$. If the input and/or output distribution for the observed data change with time, it is not necessary that $\eta(t)$ converges to zero. This dynamical situation will be considered later in our experiments.

6 Regularization

6.1 Regularization of the covariance matrix

We have assumed so far that the covariance matrix of the input data is not singular in each region, namely, the inverse matrix for every Σ_i $(i = 1, ..., M)$ exists. In actual applications of

the algorithm, however, this assumption often fails. A typical case occurs when the dimension of the input data distribution is smaller than the dimension of the input space. In such a case, Σ_i^{-1} can not be calculated and $\bar{\Lambda}_i$ diverges exponentially with the time t.

There are several methods for dealing with this problem. They are the introduction of Bayes priors, the singular value decomposition, the ridge regression, etc. However, they are not satisfactory for our purpose. Here, we will propose a simple new method that performs well.

Let us first consider a regularization of an $(N \times N)$-dimensional covariance matrix, Σ, for the observed data. Its eigen values and normalized eigen vectors are denoted by ξ_n (≥ 0) and ψ_n $(n = 1, ..., N)$, respectively.

$$\Sigma\psi_n \equiv \xi_n\psi_n \qquad (n = 1, ..., N). \tag{6.1}$$

The set of the eigen vectors, $\{\psi_n | n = 1, ..., N\}$, forms a set of orthonormal bases. The condition number of the covariance matrix Σ is defined by

$$\upsilon \equiv \xi_{min}/\xi_{max}, \tag{6.2}$$

where ξ_{min} and ξ_{max} are the minimum and maximal eigen values of Σ, respectively. So that $0 \leq \upsilon \leq 1$. If $\upsilon = 0$, the covariance matrix Σ is singular. If $\upsilon \approx 1$, the covariance matrix Σ is regular and the calculation of its inverse is numerically stable. If $0 < \upsilon \ll 1$, the covariance matrix is regular and Σ^{-1} is given by

$$\Sigma^{-1} = \sum_{n=1}^{N} \xi_n^{-1}\psi_n\psi_n'. \tag{6.3}$$

The inverse covariance matrix Σ^{-1} (6.3) is dominated by ξ_{min}, which may contain a numerical noise. As a consequence, the inverse matrix Σ^{-1} becomes sensitive to numerical noises.

In order to deal with this problem, we propose the following regularized covariance matrix Σ_R for the data covariance matrix Σ.

$$\Sigma_R = \Sigma + \alpha\Delta^2 I_N \qquad (0 < \alpha < 1) \tag{6.4a}$$
$$\Delta^2 = \mathrm{Tr}(\Sigma)/N, \tag{6.4b}$$

where I_N is an $(N \times N)$-dimensional identity matrix. The data variance Δ^2 satisfies the inequality

$$\xi_{max} \geq \Delta^2 = \frac{1}{N}\sum_{n=1}^{N} \xi_n \geq \xi_{max}/N. \tag{6.5}$$

The eigen values and eigen vectors of the regularized matrix Σ_R are given by $(\xi_n + \alpha\Delta^2)$ and ψ_n, respectively. From the inequality (6.5), it can be proved that the condition number of the regularized matrix Σ_R satisfies the condition:

$$\upsilon_R \geq \alpha/(N(1+\alpha)). \tag{6.6}$$

Then, the condition number of Σ_R is bounded below and it is controlled by the small constant α. The rate of change for the regularized eigen value is defined by

$$R(\xi_n) \equiv ((\xi_n + \alpha\Delta^2) - \xi_n)/\xi_n = \alpha(\Delta^2/\xi_n). \tag{6.7}$$

From the inequality (6.5), the ratio (6.7) satisfies the condition:

$$R(\xi_n) \geq R(\xi_{max}) \geq \alpha/N \tag{6.8a}$$
$$R(\xi_n) \leq R(\xi_{min}) \leq \alpha/\upsilon. \tag{6.8b}$$

If all the eigen values of Σ are the same order, i.e., $\upsilon \approx O(1)$, the rate of change becomes small, i.e., $R(\xi_n) \approx O(\alpha)$, so that the effect of the regularization is negligible. If the data covariance matrix Σ is singular, i.e., $\upsilon = 0$, the zero eigen value of Σ is replaced by $\alpha\Delta^2$. Thus, the regularized matrix Σ_R becomes regular and its condition number is bounded by (6.6). In general, the eigen values, which are larger than their average, are slightly affected, while the very small eigen values are changed so as to be nearly equal to $\alpha\Delta^2$.

By introducing a Bayes prior for a regularized matrix Σ_B as

$$P(\Sigma_B) = (\kappa/2)^N \exp\left(-(\kappa/2)\mathrm{Tr}(\Sigma_B^{-1})\right), \tag{6.9}$$

one can get a similar regularization equation

$$\Sigma_B = \Sigma + \kappa I_N, \tag{6.10}$$

where the regularization parameter κ is a given constant. However, it is rather difficult to determine the value of κ without knowledge of the data covariance matrix. It is especially difficult for the NGnet, since there are M independent local covariance matrices Σ_i $(i = 1, ..., M)$. The proposed method (6.4) automatically adjusts this constant by using the data variance Δ^2, which is easily calculated in an on-line manner.

If $\Sigma = 0$, Δ^2 becomes zero and the regularization (6.4) does not work. Therefore, if Δ^2 is smaller than a threshold value Δ^2_{min}, Δ^2 is set to Δ^2_{min}. This prevents Σ_R from being singular even when $\Sigma = 0$.

6.2 Regularization of on-line EM algorithm

Based on the consideration above, the i-th weighted covariance matrix is redefined in our on-line EM algorithm as:

$$\Sigma_i^{-1}(t) = \left[\left(\ll xx' \gg_i (t) - \mu_i(t)\mu_i'(t) \ll 1 \gg_i (t) + \alpha \ll \Delta_i^2 \gg_i (t) I_N\right) / \ll 1 \gg_i (t)\right]^{-1}, \tag{6.11}$$

where

$$\ll \Delta_i^2 \gg_i (t) \equiv \ll |x - \mu_i(t)|^2 \gg_i (t)/N = \left(\ll |x|^2 \gg_i (t) - |\mu_i(t)|^2 \ll 1 \gg_i (t)\right)/N. \tag{6.12}$$

The regularized Σ_i^{-1} (6.11) can be obtained from the relation (4.5) by using the following regularized $\bar{\Lambda}_i$:

$$\bar{\Lambda}_i(t) = \left(\ll \tilde{x}\tilde{x}' \gg_i (t) + \alpha \ll \Delta_i^2 \gg_i (t) \tilde{I}_N\right)^{-1}. \tag{6.13}$$

The $((N+1) \times (N+1))$-dimensional matrix, $\tilde{I}_N$, is defined by

$$\tilde{I}_N \equiv \begin{pmatrix} I_N & 0 \\ 0 & 0 \end{pmatrix} = \sum_{n=1}^{N} \tilde{e}_n \tilde{e}_n', \tag{6.14}$$

where $\tilde{e}_n$ is an $(N+1)$-dimensional unit vector; its n-th element is equal to 1 and the other elements are equal to 0. The regularization term in (6.13) can be calculated in an on-line manner. From the definition (6.12), $\ll \Delta_i^2 \gg_i (t)$ can be written as

$$\ll \Delta_i^2 \gg_i (t) = \ll \Delta_i^2 \gg_i (t-1) + \eta(t)(\Delta_i^2(t)P_i(t) - \ll \Delta_i^2 \gg_i (t-1)), \tag{6.15}$$

where $\Delta_i^2(t)$ is given by

$$\eta(t)P_i(t)\Delta_i^2(t) = \eta(t)P_i(t)|x(t) - \mu_i(t)|^2/N + (1-\eta(t))|\mu_i(t) - \mu_i(t-1)|^2 \ll 1 \gg_i (t-1)/N. \tag{6.16}$$

The second term on the right hand side in (6.16) comes from the difference between $\ll |x - \mu_i(t)|^2 \gg_i (t-1)$ and $\ll |x - \mu_i(t-1)|^2 \gg_i (t-1)$. Using (6.14) and (6.15), (6.13) can be rewritten as

$$\bar{\Lambda}_i(t) = \left[(1-\eta(t))\bar{\Lambda}_i^{-1}(t-1) + \eta(t)\left(\tilde{x}(t)\tilde{x}'(t) + \sum_{n=1}^{N} \bar{\nu}_n(t)\bar{\nu}_n'(t)\right) P_i(t)\right]^{-1} \tag{6.17a}$$

$$\bar{\nu}_n(t) \equiv \sqrt{\alpha}\Delta_i(t)\tilde{e}_n. \tag{6.17b}$$

As a result, the regularized $\bar{\Lambda}_i(t)$ (6.13) can be calculated as follows. For a given input data $x(t)$, $\bar{\Lambda}_i(t)$ is calculated using the step-wise equation (4.4). After that, $\bar{\Lambda}_i(t)$ is updated by

$$\bar{\Lambda}_i(t) := \bar{\Lambda}_i(t) - \frac{\eta(t)P_i(t)\bar{\Lambda}_i(t)\bar{\nu}_n(t)\bar{\nu}_n'(t)\bar{\Lambda}_i(t)}{1 + \eta(t)P_i(t)\bar{\nu}_n'(t)\bar{\Lambda}_i(t)\bar{\nu}_n(t)}, \tag{6.18}$$

using the virtual data $\{\bar{\nu}_n(t)|n = 1, ..., N\}$.

After calculating the regularized $\bar{\Lambda}_i(t)$ (6.13), the linear regression matrix $\bar{W}_i$ is obtained by using (4.6b), in which the regularized $\bar{\Lambda}_i(t)$ is used. In the calculation of (4.6b), only the observed data $\{(x(t), y(t))|t = 1, 2, ...\}$ are used and the virtual data $\{\bar{\nu}_n(t)|n = 1, ..., N; t = 1, 2, ...\}$, which have been used in the calculation of the regularized $\bar{\Lambda}_i(t)$, must not be used. Therefore, equation (4.6a) does not hold in our regularization method. Since the regularized $\bar{\Lambda}_i(t)$ is positive definite, the linear regression matrix $\bar{W}_i$ at an equilibrium point of (4.6b) satisfies the condition

$$\bar{W}_i E[\tilde{x}\tilde{x}'P(i|x, y, \theta)]_\rho = E[y\tilde{x}'P(i|x, y, \theta)]_\rho, \tag{6.19}$$

which is identical to the maximum likelihood equation, (3.7d) along with $\theta = \bar{\theta}$. Although the matrix $E[\tilde{x}\tilde{x}'P(i|x, y, \theta)]_\rho$ may not have the inverse, the relation (6.19) still has a meaning. Therefore, our method does not introduce a bias on the estimation of $\bar{W}_i$.

Let us compare our regularization method, (4.6b) along with (6.13), to the Bayes prior method. By introducing a Bayes prior for $\bar{W}_i$ as

$$P(\bar{W}_i) = (\kappa/2\pi)^{D(N+1)/2} \exp\left(-(\kappa/2)\mathrm{Tr}(\bar{W}_i'\bar{W}_i)\right), \tag{6.20}$$

the regularized equation for $\bar{W}_i$ is obtained:

$$\bar{W}_i = < y\tilde{x}' >_i (\langle \tilde{x}\tilde{x}' \rangle_i + \kappa I_{N+1})^{-1}. \tag{6.21}$$

This equation is a regularized version of the equation (4.6a) instead of the equation (4.6b). At an equilibrium point of the equation (6.21), $\bar{W}_i$ satisfies

$$\bar{W}_i (E[\tilde{x}\tilde{x}'P(i|x, y, \theta)]_\rho + \kappa I_{N+1}) = E[y\tilde{x}'P(i|x, y, \theta)]_\rho, \tag{6.22}$$

implying that this Bayes prior method introduces a bias on the estimation of $\bar{W}_i$.

7 Unit manipulation

Since the EM algorithm only guarantees local optimality, the obtained estimator depends on its initial value. The initial allocation of the units in the input space is especially important for attaining a good approximation. If the initial allocation is quite different from the input distribution, much time is needed to achieve proper allocation. In order to overcome this difficulty, we introduce dynamic unit manipulation mechanisms, which are also effective for dealing with dynamic environments. These mechanisms are unit production, unit deletion, and unit division, and they are conducted in an on-line manner after observing each datum $(x(t), y(t))$.

- **Unit production**
  The probability $P(x(t), y(t), i \mid \theta(t-1))$ indicates how probable the i-th unit produces the datum $(x(t), y(t))$ with the present parameter $\theta(t-1)$. Let $0 < P_{produce} \ll 1/M$. When $\max_{i=1}^{M} P(x(t), y(t), i| \theta(t-1)) < P_{produce}$, the datum is too distant from the present units to be explained by the current stochastic model. In this case, a new unit is produced to account for the new datum. The initial parameters of the new unit are given by:

$$\mu_{M+1} = x(t) \tag{7.1a}$$

$$\Sigma_{M+1}^{-1} = \chi_{M+1}^{-2} I_N \qquad \chi_{M+1}^2 = \beta_1 \min_{i=1}^{M} |x(t) - \mu_i|^2/N \tag{7.1b}$$

$$\sigma_{M+1}^2 = \beta_2 \max_{i=1}^{M} \sigma_i^2 \tag{7.1c}$$

$$\bar{W}_{M+1} \equiv (W_{M+1}, b_{M+1}) = (0, y(t)), \tag{7.1d}$$

  where β_1 and β_2 are appropriate positive constants.

- **Unit deletion**
  The weighted mean $\ll 1 \gg_i (t)$, which is calculated by (4.2), indicates how much the i-th unit has been used to account for the data until t. Let $0 < P_{delete} \ll 1/M$. If $\ll 1 \gg_i (t) < P_{delete}$, the unit has rarely been used. In this case, the i-th unit is deleted.

- **Unit division**
  The unit error variance $\sigma_i^2(t)$ (4.3d) indicates the squared error between the i-th unit's prediction and the actual output. Let D_{divide} be a specific positive value. If $\sigma_i^2(t) > D_{divide}$, the unit's prediction is insufficient, probably because the partition in charge is too large to make a linear approximation. In this case, the i-th unit is divided into two units and the partition in charge is divided into two. The initial parameters of the two units are given by:

$$\mu_i(new) = \mu_i(old) + \beta_3\sqrt{\xi_1}\psi_1 \qquad \mu_{M+1}(new) = \mu_i(old) - \beta_3\sqrt{\xi_1}\psi_1 \tag{7.2a}$$

$$\Sigma_{M+1}^{-1}(new) = \Sigma_i^{-1}(new) = 4\xi_1^{-1}\psi_1\psi_1' + \sum_{n=2}^{N} \xi_n^{-1}\psi_n\psi_n' \tag{7.2b}$$

$$\sigma_i^2(new) = \sigma_{M+1}^2(new) = \sigma_i^2(old)/2 \tag{7.2c}$$

$$\bar{W}_i(new) = \bar{W}_{M+1}(new) = \bar{W}_i(old), \tag{7.2d}$$

  where ξ_n and ψ_n denote the eigen value and the eigen vector of the covariance matrix $\Sigma_i(old)$, respectively, and $\xi_1 = \xi_{max}$. β_3 is an appropriate positive constant.

Although similar unit manipulation mechanisms have been proposed in (Platt 1991; Schaal and Atkeson 1997), these mechanisms can be conducted with a probabilistic interpretation in our on-line EM algorithm.

Finally, we would like to comment on the extrapolation done by the NGnet after the learning phase. If an input x that is far from all the unit centers is given, the output of the NGnet is approximately $\tilde{W}_i\tilde{x}$, where i is the index of the closest unit to the input x. This implies that the output of the NGnet linearly diverges as $|x| \to \infty$ where the NGnet has never experienced the training data. A simple way to prevent this undesirable behavior is given as follows. If $\sum_{j=1}^{M} G_j(x) \leq G_{min}$ for a small threshold value G_{min}, the normalized Gaussian function $\mathcal{N}_i(x)$ defined by (2.1b) is replaced by $G_i(x)/G_{min}$. This prescription, however, has not been used in the following experiments, because the input spaces in those experiments are bounded.

8 Experiments

8.1 Function approximation in static environment

Applicability of our algorithm is investigated using the following function ($N = 2$ and $D = 1$), which was used by Schaal and Atkeson (1997).

$$y = \max\left\{e^{-10x_1^2}, e^{-50x_2^2}, 1.25e^{-5(x_1^2+x_2^2)}\right\} \qquad (-1 \leq x_1, x_2 \leq 1)\,. \tag{8.1}$$

Figure 1 shows the function shape.

By sampling the input variable vector, $x \equiv (x_1, x_2)$, uniformly from its domain, we prepared 500 input-output pairs, $\{(x(t), y(t))|t = 1, ..., 500\}$, as a training data set. A fairly large Gaussian noise $N(0, 0.1)$ is added to the outputs, where $N(0, 0.1)$ denotes a Gaussian distribution whose mean and standard deviation are 0 and 0.1, respectively. Figure 2 shows the function shape with the noise. The problem task is for the NGnet to approximate function (8.1) from the 500 noisy data. We prepared 41 × 41 mesh grids on the input domain, and the approximation accuracy was evaluated by means of the averaged squared error $nMSE$ on the grids (Schaal and Atkeson 1997). Here, $nMSE$ was normalized by the variance of the desired outputs (8.1). We compared the batch EM algorithm and the on-line EM algorithm. In this experiment, the discount factor $\lambda(t)$ was scheduled for approaching 1 as in (5.13). Figure 3 shows the time-series of $nMSE$. The abscissa denotes the learning epochs, and the 500 data points were supplied once in each epoch. The same training data set was used through the whole epochs. In the figure, we can see that both the batch EM algorithm and the on-line EM algorithm are able to approximate the target function in a small number of epochs. Although the so-called "overlearning" can be seen in both learning algorithms, it is more noticeable in the batch EM algorithm than in the on-line EM algorithm. The number of the units used in both algorithms was 50. In this task, the data distribution does not change over time. In this case, it can be thought that the batch learning is more suitable than the on-line learning because the batch learning can process the whole data at once. However, our on-line EM algorithm has ability similar to the batch algorithm. In addition, our on-line EM algorithm achieves a faster and more accurate approximation ability for this task than the RFWR (Receptive Field Weighted Regression) model proposed by Schaal and Atkeson (1997).

8.2 Function approximation in dynamic environments

The on-line EM algorithm is effective for a function approximation in a dynamic environment where the input-output distribution changes with time.

For an experiment, the distribution of the input variable x_1 in (8.1) was continuously changed in 500 epochs from the uniform distribution in the interval $[-1, -0.2]$ to that in the interval $[0.2, 1]$. At each epoch, the input variables were generated in the current domain. In the applications of the on-line EM algorithm for such dynamic environments, the learning behavior changes according to the scheduling of the discount factor $\lambda(t)$. When the discount factor is relatively small, the model tends to forget the past learning result, and to quickly adapt to the present input-output distribution. We show two typical behavioral patterns in the learning phase.

1. **Fast adaptation with forgetting past**
   In the first case, $\lambda(t)$ is initially set at a relatively small value and it is slowly increased, i.e., the effective learning coefficient $\eta(t)$ is relatively large throughout the experiment. The NGnet adapts to the input distribution change by means of relocation of the units' center. During the course of this relocation, the units' center moves to the new region, and consequently the past approximation in the old region is forgotten. Figures 4(a) and 4(b) show the center and the covariance (,i.e., two-dimensional display of the standard deviation) of all the units for the 50-th epoch and the 500-th epoch, respectively. The dots denote the 500 input points in each epoch. We can see that the NGnet adapts to the input distribution change by means of the drastic relocation of the units' center. Figure 5 shows the time-series of $nMSE$ on the current input region during the course of this learning task. We can see that the error does not grow large throughout the task. In this experiment, the number of the units does not change.

2. **Slow adaptation without forgetting past**
   In the second case, $\lambda(t)$ is rapidly increased, i.e., $\eta(t)$ rapidly approaches to zero. The NGnet adapts to the input distribution change without forgetting the past approximation result. In Figure 6, the solid, dashed, and dotted lines denote the time-series of $nMSE$ on the whole input domain, $nMSE$ on the current input domain at each epoch, and the number of units, respectively. Figure 7 shows the center and the covariance of all the units at the end of the learning task. Since the unit relocation is slow, the model adapts to the input distribution change mainly by the unit production mechanism. Consequently, the units in the region where no more input data appear remain, and the function approximation on the whole input domain is accurately maintained.

8.3 Singular input distribution

In order to evaluate our regularization method, we prepared a function approximation problem where the input variables are linearly dependent and there is an irrelevant variable. In such a case, the basic on-line EM algorithm without the regularization method obtains a poor result, because the input distribution becomes singular.

In this experiment, the output y is given by the same function as (8.1), while the input variables are given by $x \equiv (x_1, x_2, x_3, x_4, x_5) = (x_1, x_2, (x_1 + x_2)/2, (x_1 - x_2)/2, 0.1)$. When the

input data are generated, a Gaussian noise $N(0, 0.05)$ is added to x_3, x_4 and x_5. For a training set, we prepared 500 data, where x_1 and x_2 were uniformly taken from their domain, and the output y was added by a Gaussian noise $N(0, 0.05)$. For evaluation, a test set was prepared by setting x_1 and x_2 on the 41×41 mesh grids and the other variables were generated according to the above prescription. During the learning, the same training set was repeatedly supplied. In Figure 8, the solid, dashed, and dotted lines are the learning curves for $\alpha = 0.23$, $\alpha = 0.1$ and $\alpha = 0$, respectively. If no regularization method is employed ($\alpha = 0$), the error becomes large after the early learning stage. Comparing the cases of $\alpha = 0.23$ and $\alpha = 0.1$, the regularization effect seems fairly robust with respect to the parameter α value.

Let us consider another situation, where the input data distribute on a curved manifold. We consider a 3-D input space. Each input datum (x_1, x_2, x_3) is restricted on the unit sphere, i.e., $x_1^2 + x_2^2 + x_3^2 = 1$. A function defined on this unit sphere is given by

$$(x_1, x_2, x_3) = (\cos\theta_1 \cos\theta_2, \cos\theta_1 \sin\theta_2, \sin\theta_1) \tag{8.2a}$$

$$y = \cos(\theta_1)\cos(\theta_2/2) \max\left\{e^{-10(2\theta_1/\pi)^2}, e^{-50(\theta_2/\pi)^2}, 1.25e^{-5((2\theta_1/\pi)^2+(\theta_2/\pi)^2)}\right\}, \tag{8.2b}$$

where the range of the spherical coordinate is given by $-\pi/2 \le \theta_1 \le \pi/2$ and $-\pi \le \theta_2 < \pi$. The output y does not include a noise. The function (8.2b) is similar to the function (8.1), but it is changed so as to satisfy the consistency of the spherical coordinate. We prepared 2000 data points uniformly on the sphere. In each learning epoch, the same data set is repeatedly used.

The covariance matrix of the whole input data is not singular. However, the covariance matrix of each local unit is nearly degenerate, so that the calculation of the inverse covariance matrix and the linear regression matrix may include a noise without the help of the regularization method. In figure 9, the solid and dashed lines are the learning curves for $\alpha = 0.023$ and $\alpha = 0$, respectively. If no regularization method is employed ($\alpha = 0$), the error does not become small. Note, however, that the calculation of the inverse covariance matrices is possible without the regularization, since the input data is not linearly degenerate. Our regularization method ($\alpha = 0.023$) provides a fairly good result compared to the basic method without the regularization ($\alpha = 0$). The condition numbers defined by (6.2) with the regularization and without the regularization are 0.0191 ± 0.0042 and 0.0071 ± 0.0038, respectively.

The local covariance matrix, Σ_i, of each unit represents the local input data distribution fairly well in our regularized method. It has two principal components which span the tangential plane to the unit sphere at each local unit position. The third eigen value is very small and it is bounded below by the regularization term. In figure 10, receptive fields of the local units are shown. The receptive field of each unit is defined by an ellipse whose axes correspond to the two principal components of the local covariance matrix Σ_i. The center of this ellipse is set at the center of the local unit, μ_i. One can see that the receptive fields appropriately cover the unit sphere. The average cosine between the eigen vectors, which corresponds to the third eigen values of the covariance matrices, and the spherical normal vectors was 0.9966 ± 0.0157. This implies that the receptive fields are almost tangential to the unit sphere.

8.4 Reinforcement learning

We apply our new approach to a reinforcement learning problem. The task is to swing the pendulum upward by a restricted torque controller and stabilize the pendulum near the up-

right position (Doya 1996). An actor-critic model proposed by Barto et al. (Barto, Sutton & Anderson, 1983) is used for the learning system. For the current state, $x_c(t)$, of the controlled system, the actor outputs an control signal (action) $u(t)$, which is given by the policy function $\Omega(x_c(t))$, i.e., $u(t) = \Omega(x_c(t))$. The controlled system changes its state to $x_c(t+1)$ after receiving the control signal $u(t)$. Following that, a reward $r(x_c(t+1))$ is given to the learning system. It is assumed that there is no knowledge of the controlled system.

The objective of the learning system is to find the optimal policy function $\Omega^*(x_c)$ that maximizes the discounted future return defined by

$$V(x_c) \equiv \sum_{t=0}^{\infty} \gamma^t r(x_c(t+1))\Big|_{x_c(0)=x_c}, \tag{8.3}$$

where $0 < \gamma < 1$ is a discount factor and $V(x_c)$ is called the value function. The value function $V(x_c)$ is defined for the current policy function $\Omega(x_c)$ employed by the actor.

The Q-function is defined by

$$Q(x_c, u) = \gamma V(x_c(t+1)) + r(x_c(t+1)), \tag{8.4}$$

where $x_c(t) = x_c$ and $u(t) = u$ are assumed. The value function can be obtained from the Q-function:

$$V(x_c) = Q(x_c, u = \Omega(x_c)). \tag{8.5}$$

The Q-function should satisfy the consistency condition

$$Q(x_c(t), u(t)) = \gamma Q(x_c(t+1), \Omega(x_c(t+1))) + r(x_c(t+1)). \tag{8.6}$$

The critic estimates the Q-function that satisfies the consistency condition (8.6). The Q-function is approximated by the NGnet, which is called the critic-network. The input to the critic-network is the current system state $x_c(t)$ and the control signal $u(t)$. The target output for the critic-network is given by the right hand side of (8.6), in which the Q-function is calculated using the current critic-network. After obtaining the new state $x_c(t+1)$, the parameters of the critic-network are updated using the on-line EM algorithm. This learning scheme is different from TD-learning (Sutton 1988) or Q-learning (Watkins 1989), because it directly uses the target Q-function value.

In the task for swinging up the pendulum, the control signal $u(t)$ represents a torque which is applied to the controlled system, i.e., the pendulum. The policy function is approximated by an actor-network, which is a variation of the Normalized Gaussian Network:

$$u = \Omega(x_c) = u_{max} \cdot \tanh\left(\sum_{i=1}^{M_0} \omega_i \mathcal{N}_i(x_c) + \epsilon\right), \tag{8.7}$$

where a random noise ϵ is added in the training phase in order to explore the state space. Since the maximal torque is fixed at u_{max}, the output of the actor-network is filtered through the sigmoidal function, $\tanh(\cdot)$. The centers of the units are fixed at the mesh grid points in the input space. The covariance matrices are also fixed to the univariate covariance matrices with the same variance. There is no linear term. Only the bias parameters ω are updated by the gradient ascent method so that the Q-function value increases (Sofge and White 1992):

$$\Delta\omega \propto \frac{\partial\Omega}{\partial\omega}(x_c(t)) \cdot \frac{\partial Q}{\partial u}(x_c(t), u(t)). \tag{8.8}$$

The reward for the inverted pendulum is given by

$$r(x_c) = \exp(-(\dot{\theta})^2/2\iota_1^2 - \theta^2/2\iota_2^2), \tag{8.9}$$

where θ and $\dot{\theta}$ denote the angle from the upright position and the angular velocity of the pendulum, respectively, i.e., $x_c \equiv (\dot{\theta}, \theta)$. $\iota_1 (= 3\pi/5)$ and $\iota_2 (= \pi/5)$ are constant values. The reward (8.9) encourages the pendulum to stay near the upright position.

After releasing the pendulum from a vicinity of the upright position, the control and the learning process of the actor-critic network is conducted for 7 seconds. This is a single episode. The reinforcement learning is done by repeating these episodes. As the learning proceeds, the initial position of the pendulum is gradually moved away from the upright position. In order to see the learning performance, we prepared three testbeds.

- Easy initial setting: $0 \leq |\dot{\theta}| \leq 3\pi/5, 0 \leq |\theta| \leq \pi/5$
- Medium initial setting: $0 \leq |\dot{\theta}| \leq 6\pi/5, \pi/5 \leq |\theta| \leq 2\pi/5$
- Difficult initial setting: $0 \leq |\dot{\theta}| \leq 9\pi/5, 2\pi/5 \leq |\theta| \leq 3\pi/5$

After each episode, the actor-critic network is tested under the above three testbeds. Figure 11 shows the time-series of the success rate for the three testbeds. A success is determined when the final reward is larger than 0.99. In order to achieve this reward value, the pendulum should stay near the upright position, because the reward (8.9) includes a penalty term for a large velocity. After about 100 episodes, the system is able to make the pendulum achieve an upright position for the easy initial setting. After this learning stage, the success rate for easy and medium initial settings slightly decreases, because the initial position at the training session moves away from the upright position. In this learning period, the system is mainly adapting to initial states fairly distant from the upright position. This adaptation is conducted by relocation of the critic-network units. After 350 episodes, the system is able to make the pendulum achieve an upright position from almost every initial state. Since the maximal torque generated by the controller is limited, the system inverts the pendulum after swinging it several times. According to our previous experiments, in which the critic-network is the NGnet trained by the gradient descent learning algorithm, a good control was obtained after about 2000 episodes. Therefore, our new approach based on the on-line EM algorithm is able to obtain a good control much faster than that based on the gradient descent algorithm.

9 Conclusion

In this article, we proposed a new on-line EM algorithm for the NGnet. We showed that the on-line EM algorithm is equivalent to the batch EM algorithm if a specific scheduling of the discount factor is employed. In addition, we showed that the on-line EM algorithm can be considered as a stochastic approximation method to find the maximum likelihood estimator.

A new regularization method was proposed in order to deal with a singular input distribution. In order to manage the dynamic environments, unit manipulation mechanisms such as unit production, unit deletion, and unit division were also introduced based on the probabilistic interpretation.

Experimental results showed that our approach is suitable for dynamical environments where the input-output distribution of data changes with time.

We also applied our on-line EM algorithm to a reinforcement learning problem. It has been shown that the NGnet, when using the on-line EM algorithm, learns the value function much faster than the method based on the gradient descent algorithm.

In forthcoming papers, we will discuss applications for the reinforcement learning in more detail.

実験手順書

目次

実験手順書

本ドキュメントでは、主にバッチ処理での実験手順について説明します。バッチ処理ではハーツサーバ画面は起動されません。

システム全体のディレクトリ構成

ハーツサーバ・エージェントのシステム全体のディレクトリ構成は以下の通りです。システム全体の親ディレクトリは現在(2000年2月29日)

/export/home/matsuda/CSK

となっています。本ドキュメントではこれをPAIRHEARTSと表します。

PAIRHEARTS=/export/home/matsuda/CSK/			
server/	ハーツサーバsrvHearts/ソースコード		
	LOG/	ログファイル格納ディレクトリ	
	log_read/	ログ読みこみツールlog_read/ソースコード	
	sample/	バッチ・シェルファイルサンプル	
	resource/	画面用リソースファイル	
tp/	TPエージェントtp1/ソースコード		
	data/	tp1プログラム用データ及びTPエージェントデータ格納ディレクトリ	
		INITDATA/	TP エージェント初期状態データ(バックアップ用)
	config/	TPエディターconfig_tp1/ソースコード	
	resource/	画面用リソースファイル	
pp/	PPエージェントpp1/ソースコード		
	data/	pp1 プログラム用データ及び PP エージェントデータ格納ディレクトリ	
		INITDATA/	PP エージェント初期状態データ(バックアップ用)
	config/	PPエディターconfig_pp1/ソースコード	
	resource/	画面用リソースファイル	
exp/	実験データ格納ディレクトリ・csv作成ツールソースコード		
common/	TPエージェント・PPエージェントプログラム作成用共通ソースコード		
	config/	TPエディター・PPエディタープログラム作成用共通ソースコード	
client/	アプレット用html・画像ファイル		
	Client1/	アプレットバイトコード（バックアップ用）	
	src/	アプレットソースコード	
	gif96/	画像ファイル	
	gif80/	画像ファイル	
agent/	エージェント作成用雛型ソースコード		

各システムの動作のための注意事項

ハーツサーバ

ハーツサーバプログラム(srvHearts)が各エージェントプログラムを起動します。エージェントプログラムの起動にはエージェント名及び実行プログラム名の情報が必要です。そのため、エージェントプログラムがどこにあるかがあらかじめエージェント起動設定ファイルに書かれている必要があります。エージェント起動設定ファイル名は次の通りです。これらは PAIRHEARTS/server/に存在します。

tplist.exp, tplist.demo	(TP エージェントプログラム起動用)
pplist.exp, pplist.demo	(PP エージェントプログラム起動用)

実験を行う前にエージェント起動設定ファイルに正しくデータが設定されているかを確認して下さい。(tplist.demo, pplist.demo はハーツサーバのフリーデモ用のためのファイルです。フリーデモ用にエージェントを起動するためにはこれらのファイルは変更する必要がありますが、実験のためには変更の必要はありません。)
エージェント起動設定ファイルの書式は以下の通りです。

エージェント名　実行ファイルのパス　実行ファイル名

以下に、tplist.exp のサンプルを載せます。

```
tpLV1.6 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV2.6 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV3.6 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV1.6_a0 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV1.6_a1 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV1.6_a2 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV1.6_a3 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV2.6_a0 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV2.6_a1 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV2.6_a2 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV2.6_a3 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV3.6_a0 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV3.6_a1 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV3.6_a2 /export/home/matsuda/CSK/tp/tp1 tp1
tpLV3.6_a3 /export/home/matsuda/CSK/tp/tp1 tp1
```

srvHeartsは動作中に簡単な実験ログをPAIRHEARTS/server/のkekka.datに出力します。出力される内容は以下の通りです。

カード配布のための乱数seed
一定間隔毎の各グループの累計ペナルティ

kekka.datのフォーマットは以下の通りです。

(1行目)　seed = *numseed*
(2行目以降) hand[*numhand*]　Total Score=*s0 s1 s2 s3*

numseed	seedの値
numhand	ハンド回数
s0 s1 s2 s3	各グループの累計ペナルティ

ただし、kekka.datは追加モードで出力されますので、次回srvHeartsが起動された場合はファイルの続きから上記の情報が出力されることになります。
累計ペナルティを出力する間隔はソースファイル monitorGame.c で#define FILEOUT_INTERVALとして定義しています。現在（2000年2月29日）設定している間隔は 1000 回です。間隔を変更する場合はこの定義を変更し、PAIRHEARTS/server/で再コンパイル(make)して下さい。monitorGame.c はPAIRHEARTS/server/にあります。

ログ読みこみツール

ハーツサーバはログファイルを PAIRHEARTS/server/LOG/に出力します。ログ読みこみツールプログラム(log_read)は、ハーツサーバが出力したログファイルを読みこみ、画面に表示／印刷します。ログ読みこみツールの起動コマンドは以下の通りです。

% log_read *logfile*

log_read は、指定された名前のログファイル(*logfile*)が PAIRHEARTS/server/LOG/にあるものとして読みこみ処理を行います。ログ読みこみツールを実行する場合は、ログファイルが PAIRHEARTS/server/LOG/にあることを確認して下さい。

ログ読みこみツールについては、ドキュメント「ログ読みこみツール説明書」を参照して下さい。

TP エージェント

TP エージェントプログラム(tp1)は起動時に与えられた名前 XXX をもとに、PAIRHEARTS/tp/data/の各種 TP エージェントファイルを読みこみます。これらのファイルがこの名前の TP エージェントを特徴付けます。ファイルの内容は以下の通りです。

XXX.ele	使用できる条件部構成要素・結論部の設定ファイル
XXX.rule	行動ルールファイル
XXX.hclass	ハンド分類枠ファイル
XXX.hand	経験ハンドファイル
XXX.mode	学習モードファイル

(XXX は TP エージェント名を表します。)

ただし、XXX.hand は、実験参加の経験がない場合(初期状態)は存在しません。また、学習モードファイルは編集ツールにより初期化されている場合は存在しません。

実験を開始する前にこれらのファイルが PAIRHEARTS/tp/data/にあることを確認して下さい。

PAIRHEARTS/tp/data/INITDATA/にはこれまでに作成した各レベルの TP エージェントのファイルがバックアップ用として格納してあります。ファイルデータの内容は初期状態になっています。(XXX.hand はありません。) これらの TP エージェントを初期状態から実験に参加させるときは、PAIRHEARTS/tp/data/INITDATA/から PAIRHEARTS/tp/data/にファイルをコピーして下さい。

実験中は、一定のハンド回数間隔毎にその時点のハンド分類枠ファイルを出力するようにしています。自己学習が On の場合は、行動ルールファイルも出力するようにしています。このとき、出力されるファイル名は次のようになっています。

XXX_num.rule
XXX_num.hclass

(XXX は TP エージェント名です。
num は出力した時点のハンド回数を示します。)

この間隔はソースファイル tp1_extern.h で#define FILEOUT_INTERVAL として定義しています。現在 (2000 年 2 月 29 日) 設定している間隔は 1000 回です。間隔を変更する場合はこの定義を変更し、PAIRHEARTS/tp/で再コンパイル(make)して下さい。

PAIRHEARTS 及び TP エージェントのファイル格納先の場所を変更する場合は、ソースファイル tp1_extern.h の

#define TP1DATADIR
#define TP1CONDSCRIPT
#define TP1ACTSCRIPT
#define TP1DISTSCRIPT

及びソースファイル tp1.c の

#define APPDEF_RESOURCE

の定義を変更し、再コンパイルして下さい。
これらのソースファイルは PAIRHEARTS/tp/にあります。

TP エージェントの動作についてはドキュメント「TP エージェント」を参照して下さい。
TP エージェントプログラムの概要・ファイルフォーマットについてはドキュメント「ソフトウェア機能仕様書」を参照して下さい。
TP エージェントプログラム実行ファイル作成に必要なソースコード・データファイルについてはドキュメント「プログラム設計書」を参照して下さい。

TP エディター

TP エディター（create_tp1）は TP エージェントの編集(作成・変更)を行います。TP エージェントを新規作成すると、create_tp1 はその TP エージェントのデータファイルを PAIRHEARTS/tp/data/に出力し、新規 TP 名を PAIRHEARTS/server/tplist.exp に追加します。

新規作成時に、既に存在する TP エージェントからコピーして作成する場合には、create_tp1 はコピー元の TP エージェントのファイルを PAIRHEARTS/tp/data/から読みこんで処理を行います。そのため、コピー元の TP エージェントのデータは PAIRHEARTS/tp/data/にある必要があります。PAIRHEARTS/tp/data/INITDATA/の TP エージェントからコピーする場合は、一旦 PAIRHEARTS/tp/data/INITDATA/から PAIRHEARTS/tp/data/にファイルをコピーして下さい。
TP エージェントのデータを変更する場合も同じくそのデータが PAIRHEARTS/tp/data/にある必要があります。

変更する TP エージェントのリスト及びコピー元の TP エージェントのリストを表示するために create_tp1 は PAIRHEARTS/server/の tplist.exp を利用します。(tplist.exp に登録されている TP エージェント名をリストに表示します。) TP エージェントの変更作業・コピーを行うときは tplist.exp にこれらの TP エージェント名が登録されていることを確認して下さい。

また PAIRHEARTS の場所を変更する場合は、ソースファイル configtp1.h の

```
#define SERVERENTRYFILE
#define TP1PROGRAMPATH
#define CONDSCRIPTFILE
#define ACTSCRIPTFILE
#define DISTSCRIPTFILE
#define TP1DATADIRECTORY
```

の定義を変更し、PAIRHEARTS/tp/config/で再コンパイル(make)して下さい。
これらのソースファイルは PAIRHEARTS/tp/config/にあります。

TP エディターの操作方法についてはドキュメント「TP エディター操作説明書 1」及び「TP エディター操作説明書 2」を参照して下さい。
TP エディターの概要についてはドキュメント「ソフトウェア機能仕様書」を参照して下さい。
TP エディター実行ファイル作成に必要なソースコード・データファイルについてはドキュメント「プログラム設計書」を参照して下さい。

PP エージェント

PP エージェントプログラム(pp1)は起動時に与えられた名前 XXX をもとに、PAIRHEARTS/pp/data/の XXX ディレクトリ以下の各種 PP エージェントファイルを読みこみます。これらのファイルがこの名前の PP エージェントを特徴付けます。ファイルの内容は以下の通りです。

XXX/element	使用できる条件部構成要素・結論部の設定ファイル
XXX/rule.org	行動ルールファイル(オリジナル)
XXX/prof.org	TP プロファイル枠ファイル(オリジナル)
XXX/mode	学習モードファイル
XXX/list.prtnr	これまでにパートナーとなった TP 名のリストファイル

(XXX は指定された PP エージェント名のディレクトリを表します。)

ただし、list.prtnr は実験に参加し、パートナー TP とのゲーム経験がない場合(初期状態)は存在しません。また、学習モードファイルは編集ツールにより初期化されている場合は存在しません。

PAIRHEARTS/pp/data/INITDATA/にはこれまでに作成した PP エージェントのディレクトリがバックアップ用として格納してあります。ファイルデータの内容は初期状態になっています。(list.prtnr はありません。) これらの PP エージェントを初期状態から実験に参加させるときは、PAIRHEARTS/pp/data/INITDATA/ から PAIRHEARTS/pp/data/にディレクトリごとコピーして下さい。

また、実験開始後にパートナーの TP(PRTNR)が確定されると、pp1 は PAIRHEARTS/pp/data/XXX/のパートナー固有の各種ファイルを読みこみます。これらのファイルがパートナー TP を特徴付けます。ファイルの内容は以下の通りです。

XXX/PRTNR.rule	このパートナーに対する行動ルールファイル
XXX/PRTNR.prof	このパートナーに対する TP プロファイル枠ファイル
XXX/PRTNR.hand	このパートナーに対する経験ハンドファイル

(XXX は指定された PP エージェント名のディレクトリを表します。PRTNR はパートナー TP 名を表します。)

パートナー TP に対する各種ファイルはそのパートナー TP とのゲーム経験がない場合は存在しません。pp1 はそのパートナー TP が PP エージェントにとって始めてのパートナーである場合は、オリジナルのファイルを元にファイルを新規作成します。

実験を開始する前にこれらのディレクトリ及びファイルが PAIRHEARTS/pp/data/にあることを確認して下さい。

実験中は、自己学習が On の場合は一定のハンド回数間隔毎にその時点の行動ルールファイルを出力するようにしています。このとき、出力されるファイル名は次のようになっています。

XXX/PRTNR_num.rule

(XXX は PP エージェント名のディレクトリを表します。PRTNR は TP エージェント名です。num は出力した時点のハンド回数を示します。)

この間隔はソースファイル pp1_extern.h で#define FILEOUT_INTERVAL として定義しています。現在（2000 年 2 月 29 日）設定している間隔は 1000 回です。間隔を変更する場合はこの定義を変更し、PAIRHEARTS/pp/で再コンパイル(make)して下さい。

PAIRHEARTS の場所を変更する場合は、ソースファイル pp1_extern.h の

#define PP1DATADIR
#define PP1CONDSCRIPT
#define PP1ACTSCRIPT

及びソースファイル pp1.c の

#define APPDEF_RESOURCE

の定義を変更し、再コンパイルして下さい。
これらのソースファイルは PAIRHEARTS/pp/にあります。

PP エージェントの動作についてはドキュメント「PP エージェント」を参照して下さい。
PP エージェントプログラムの概要・ファイルフォーマットについてはドキュメント「ソフトウェア機能仕様書」を参照して下さい。
PP エージェントプログラム実行ファイル作成に必要なソースコード・データファイルについてはドキュメント「プログラム設計書」を参照して下さい。

PP エディター

PP エディター (create_pp1) は PP エージェントの編集(作成・変更)を行います。PP エージェントを新規作成すると、create_pp1 はその PP エージェント名のディレクトリを PAIRHEARTS/pp/data/に作成し、そのディレクトリ以下にデータファイルを出力します。また、新規 PP 名を PAIRHEARTS/server/pplist.exp に追加します。

新規作成時に、既に存在する PP エージェントからコピーして作成する場合には、create_tp1 はコピー元の PP エージェントのファイルを PAIRHEARTS/pp/data/のその PP エージェント名と同じ名前のディレクトリ以下から読みこんで処理を行います。そのため、コピー元の PP エージェントのデータは PAIRHEARTS/pp/data/のその PP エージェント名と同じ名前のディレクトリ以下にある必要があります。
PAIRHEARTS/pp/data/INITDATA/の PP エージェントからコピーする場合は、一旦 PAIRHEARTS/pp/data/INITDATA/から PAIRHEARTS/pp/data/にディレクトリごとコピーして下さい。
PP エージェントのデータを変更する場合も同じくディレクトリ及びデータが PAIRHEARTS/pp/data/にある必要があります。

変更する PP エージェントのリスト及びコピー元の PP エージェントのリストを表示するために create_pp1 は PAIRHEARTS/server/の pplist.exp を利用します。(pplist.exp に登録されている PP エージェント名をリストに表示します。) PP エージェントの変更作業・コピーを行うときは pplist.exp にこれらの PP エージェント名が登録されていることを確認して下さい。

また PAIRHEARTS の場所を変更する場合は、ソースファイル configpp1.h の

```
#define SERVERENTRYFILE
#define PP1PROGRAMPATH
#define CONDSCRIPTFILE
#define ACTSCRIPTFILE
#define DISTSCRIPTFILE
#define PP1DATADIRECTORY
```

の定義を変更し、PAIRHEARTS/pp/config/で再コンパイル(make)して下さい。
これらのソースファイルは PAIRHEARTS/pp/config/にあります。

PP エディターの操作方法についてはドキュメント「PP エディター操作説明書 1」及び「PP エディター操作説明書 2」を参照して下さい。
PP エディターの概要についてはドキュメント「ソフトウェア機能仕様書」を参照して下さい。
PP エディター実行ファイル作成に必要なソースコード・データファイルについてはドキュメント「プログラム設計書」を参照して下さい。

バッチ処理での実験

サーバ画面を起動しないで、実験をバッチ処理で行うことができます。srvHearts は指定されたバッチファイルの設定でゲーム条件を設定し、エージェントを起動、ゲームを開始します。バッチファイルに学習モードの設定が記述されている場合はエージェントの学習モードを設定します。ゲームが終了するとプログラムを終了します。
バッチ処理での実験の場合、ゲーム終了条件は通常終了(いずれかのグループが 100 点を越えた場合に終了)、もしくは回数指定による終了となります。回数指定の場合は、回数をバッチファイルに設定します。
シェルファイルに複数のバッチ処理での実験コマンドを書くことで、複数の実験を連続して行うことができます。

バッチファイルのフォーマット

以下のフォーマットでバッチファイルを記述して下さい。太字が設定項目のタグ、斜体が設定項目に対する実際の設定データを意味します。以下に記述されているのはサンプルデータです。実験内容に沿ってこれらの設定データを書き換えて下さい。

行番号	
1	agent
2	*0 tpLV3.6_a0 pp12_a0*
3	*1 tpLV3.6_a1 pp12_a1*
4	*2 tpLV3.6_a2 pp12_a2*
5	*3 tpLV2.6 pp12_a3*
6	shtmoon *0*
7	outfile *exp175.log*
8	gendnum *5000*
9	ppmode *0 pp12_a0 -8.0 0 -10.0 1.0*
10	ppmode *1 pp12_a1 -8.0 0 -10.0 1.0*
11	ppmode *2 pp12_a2 -8.0 0 -10.0 1.0*
12	ppmode *3 pp12_a3 -8.0 1 -10.0 1.0*
13	tpmode *0 tpLV3.6_a0 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0*
14	tpmode *1 tpLV3.6_a1 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0*
15	tpmode *2 tpLV3.6_a2 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0*
16	tpmode *3 tpLV2.6 -8.0 1 -10.0 1.0 1.0 1 -8.0 -1.0 0.0 2 1 1 1 0*

行番号１：agent
実験に参加させるエージェント名を続く４行で設定して下さい。
行番号２～５：
各グループに参加する TP エージェント名、PP エージェント名を設定します。
書式は次の通りです。
グループ番号　TP エージェント名　PP エージェント名
行番号６：shtmoon *0*
Shooting the Moon の On(1)/Off(0)を設定して下さい。

行番号７：outfile *exp175.log*
出力するログファイル名を設定して下さい。ログを取らない場合はこの行は削除して下さい。
行番号８：gendnum *5000*
実験を終了するまでのハンド回数を指定して下さい。通常終了で実験を行う場合はこの行は削除して下さい。
行番号９～１２：
PP エージェントの学習モードを設定します。
ppmode に続く書式は以下の通りです。(1 行で書いて下さい。)
グループ番号　PP エージェント名　類似ルール選択閾値
自己学習 On(1)/Off(0)　自己学習活性化閾値　慣性係数（CI）
行番号１３～１６：
TP エージェントの学習モードを設定します。
tpmode に続く書式は以下の通りです。(1 行で書いて下さい。)
グループ番号　TP エージェント名　類似ルール選択閾値
自己学習 On(1)/Off(0)　自己学習活性化閾値　慣性係数（CI）
自己学習石橋度　相手学習 On(1)/Off(0)　相手学習ルール合併閾値
相手学習活性化閾値（TML）　相手学習石橋度
相手学習モード(無指定(1)/相手指定(2))
グループ 0 の TP に対する相手学習の On(1)/Off(0)
グループ 1 の TP に対する相手学習の On(1)/Off(0)
グループ 2 の TP に対する相手学習の On(1)/Off(0)
グループ 3 の TP に対する相手学習の On(1)/Off(0)

ppmode, tpmode では上記の順番でデータを読んでいきますが、データが途中までの場合でもかまいません。また、上記以上のデータが並んでいる場合は、上記以上のデータは無視されます。ppmode, tpmode を指定しない場合は、現在の学習モード設定ファイルのデータがエージェントの学習モードとなります。学習モードの機能についてはドキュメント「ソフトウェア機能仕様書」、「TP エージェント」、「PP エージェント」を参照して下さい。

バッチ処理での実験手順

1．実験に参加させる TP エージェント、PP エージェントを起動するための設定が PAIRHEARTS/server/の tplist.exp,pplist.exp に書かれているかを確認します。
2．TP エージェントのデータ、PP エージェントのディレクトリ・データが PAIRHEARTS/tp/data/, PAIRHEARTS/pp/data/にあるかを確認します。
3．実験内容に沿ってバッチファイルを PAIRHEARTS/server/に作成します。
4．PAIRHEARTS/server/でハーツサーバを起動します。バッチ処理での起動コマンドは以下の通りです。

% srvHearts　−i　*filename*

filename にはバッチファイル名を指定します。
5．ゲームが終了すると、srvHearts は自動的に終了します。

シェルファイルによる連続実験

以下では次に示す実験１～４を連続するための手順について示します。

実験１：TP レベル 3.6x3 名+TP レベル 1.6x1 名、PPx4 名による学習無し実験（バックグランド実験)。

実験２：実験１の終了状態から TP レベル 1.6 のパートナー PP を自己学習させる実験。

実験３：実験２の終了状態から TP レベル 1.6 を石橋型自己学習させる、また、TP レベル 1.6 のパートナー PP を自己学習させる実験。

実験４：実験３の終了状態から TP レベル 1.6 を石橋型自己学習及び石橋型相手学習させ、TP レベル 1.6 のパートナー PP を自己学習させる実験。相手学習の学習相手はレベル 3.6x3 名それぞれに対して行う。

これらの実験は全て Shooting the Moon は Off とし、ハンド回数は 5000 回とします。

TP レベル 3.6x3 名は次の 3 つの TP エージェントを使用します。

tpLV3.6_a0
tpLV3.6_a1
tpLV3.6_a2

TP レベル 1.6 は次の TP エージェントを使用します。

tpLV1.6

(これらの TP エージェントの初期状態のデータは PAIRHEARTS/tp/data/INITDATA/にあります。)

PPx4 名は次の 3 つの PP エージェントを使用します。

pp12_a0
pp12_a1
pp12_a3
pp12_a4

(これらの PP エージェントの初期状態のディレクトリ・データは PAIRHEARTS/pp/data/INITDATA/にあります)。

バッチファイルの作成(例)

実験1～4のバッチファイルを次のように作成します。

実験1のバッチファイル exp1.bat

```
agent
0 tpLV3.6_a0 pp12_a0
1 tpLV3.6_a1 pp12_a1
2 tpLV3.6_a2 pp12_a2
3 tpLV1.6 pp12_a3
shtmoon 0
outfile exp1.log
gendnum 5000
ppmode 0 pp12_a0 -8.0 0 -10.0 1.0
ppmode 1 pp12_a1 -8.0 0 -10.0 1.0
ppmode 2 pp12_a2 -8.0 0 -10.0 1.0
ppmode 3 pp12_a3 -8.0 0 -10.0 1.0
tpmode 0 tpLV3.6_a0 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 1 tpLV3.6_a1 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 2 tpLV3.6_a2 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 3 tpLV1.6      -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
```

実験2のバッチファイル exp2.bat

```
agent
0 tpLV3.6_a0 pp12_a0
1 tpLV3.6_a1 pp12_a1
2 tpLV3.6_a2 pp12_a2
3 tpLV1.6 pp12_a3
shtmoon 0
outfile exp2.log
gendnum 5000
ppmode 0 pp12_a0 -8.0 0 -10.0 1.0
ppmode 1 pp12_a1 -8.0 0 -10.0 1.0
ppmode 2 pp12_a2 -8.0 0 -10.0 1.0
ppmode 3 pp12_a3 -8.0 1 -10.0 1.0
tpmode 0 tpLV3.6_a0 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 1 tpLV3.6_a1 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 2 tpLV3.6_a2 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 3 tpLV1.6      -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
```

実験３のバッチファイル exp3.bat

```
agent
0 tpLV3.6_a0 pp12_a0
1 tpLV3.6_a1 pp12_a1
2 tpLV3.6_a2 pp12_a2
3 tpLV1.6 pp12_a3
shtmoon 0
outfile exp3.log
gendnum 5000
ppmode 0 pp12_a0 -8.0 0 -10.0 1.0
ppmode 1 pp12_a1 -8.0 0 -10.0 1.0
ppmode 2 pp12_a2 -8.0 0 -10.0 1.0
ppmode 3 pp12_a3 -8.0 1 -10.0 1.0
tpmode 0 tpLV3.6_a0 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 1 tpLV3.6_a1 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 2 tpLV3.6_a2 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 3 tpLV1.6      -8.0 1 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
```

実験４のバッチファイル exp4.bat

```
agent
0 tpLV3.6_a0 pp12_a0
1 tpLV3.6_a1 pp12_a1
2 tpLV3.6_a2 pp12_a2
3 tpLV1.6 pp12_a3
shtmoon 0
outfile exp4.log
gendnum 5000
ppmode 0 pp12_a0 -8.0 0 -10.0 1.0
ppmode 1 pp12_a1 -8.0 0 -10.0 1.0
ppmode 2 pp12_a2 -8.0 0 -10.0 1.0
ppmode 3 pp12_a3 -8.0 1 -10.0 1.0
tpmode 0 tpLV3.6_a0 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 1 tpLV3.6_a1 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 2 tpLV3.6_a2 -8.0 0 -10.0 1.0 1.0 0 -8.0 -1.0 1.0
tpmode 3 tpLV1.6      -8.0 1 -10.0 1.0 1.0 1 -8.0 -1.0 1.0 2 1 1 1 0
```

シェルファイルの作成(例)

シェルファイルは PAIRHEARTS/server/以下に作成します。以下のシェルファイル(sample.sh)は 4 つの実験を連続で行うためのサンプルです。

行番号	
1	cp ../tp/data/INITDATA/tpLV3.6_a* ../tp/data/
2	cp ../tp/data/INITDATA/tpLV1.6.* ../tp/data/
3	cp -r ../pp/data/INITDATA/pp12_a* ../pp/data/
4	srvHearts -i exp1.bat
5	cd LOG
6	gzip exp1.log
7	cd ..
8	mkdir ../exp/exp1
9	mkdir ../exp/exp1/tp
10	mkdir ../exp/exp1/pp
11	mv ../tp/data/tpLV* ../exp/exp1/tp
12	mv ../pp/data/pp12_a* ../exp/exp1/pp
13	mv kekka.dat ../exp/exp1
14	mv LOG/exp1.log.gz ../exp/exp1
15	
16	cp ../exp/exp1/tp/* ../tp/data/
17	cp -r ../exp/exp1/pp/* ../pp/data/
18	srvHearts -i exp2.bat
19	cd LOG
20	gzip exp2.log
21	cd ..
22	mkdir ../exp/exp2
23	mkdir ../exp/exp2/tp
24	mkdir ../exp/exp2/pp
25	mv ../tp/data/tpLV* ../exp/exp2/tp
26	mv ../pp/data/pp12_a* ../exp/exp2/pp
27	mv kekka.dat ../exp/exp2
28	mv LOG/exp2.log.gz ../exp/exp2
29	
30	cp ../exp/exp2/tp/* ../tp/data/
31	cp -r ../exp/exp2/pp/* ../pp/data/
32	srvHearts -i exp3.bat
33	cd LOG
34	gzip exp3.log
35	cd ..
36	mkdir ../exp/exp3
37	mkdir ../exp/exp3/tp
38	mkdir ../exp/exp3/pp
39	mv ../tp/data/tpLV* ../exp/exp3/tp
40	mv ../pp/data/pp12_a* ../exp/exp3/pp
41	mv kekka.dat ../exp/exp3

42	mv LOG/exp3.log.gz ../exp/exp3
43	
44	cp ../exp/exp3/tp/* ../tp/data/
45	cp -r ../exp/exp3/pp/* ../pp/data/
46	srvHearts -i exp4.bat
47	cd LOG
48	gzip exp4.log
49	cd ..
50	mkdir ../exp/exp4
51	mkdir ../exp/exp4/tp
52	mkdir ../exp/exp4/pp
53	mv ../tp/data/tpLV* ../exp/exp4/tp
54	mv ../pp/data/pp12_a* ../exp/exp4/pp
55	mv kekka.dat ../exp/exp4
56	mv LOG/exp4.log.gz ../exp/exp4

行番号 1～１４までが実験１のための記述です。

行番号１、２：

PAIRHEARTS/tp/data/INITDATA/ の初期状態のデータを PAIRHEARTS/tp/data/にコピーします。

行番号３：

PAIRHEARTS/pp/data/INITDATA/ の初期状態のデータを PAIRHEARTS/pp/data/にコピーします。

行番号４：

exp1.bat の設定でバッチ処理での実験を行います。

行番号５～７：

srvHearts が出力したログファイルを圧縮します。(ログファイルはヘッダ情報だけで約 3.3Kb、1 ハンド情報で約 1.3Kb 使用します。そのため、5000 回のハンドでは約 6.5Mb の大きさになるため、ここでは圧縮を行っています。ログ読みこみツール等でログを使用する場合は gunzip で圧縮を伸張して下さい。)

行番号８～１４：

PAIRHEARTS/exp/に実験１用のデータを保存するためのディレクトリ exp1 を作成し、さらに、TP 用、PP 用のディレクトリをその下に作成し、それぞれのデータを PAIRHEARTS/tp/data/、PAIRHEARTS/pp/data/から移動させます。また、srvHearts が出力したファイルも移動させます。

行番号１６～２８までが実験２のための記述です。

行番号１６、１７：

実験１の終了状態から実験を行うため、実験１のデータを PAIRHEARTS/exp/exp1/から PAIRHEARTS/tp/data/、PAIRHEARTS/pp/data/にコピーします。

行番号１８：

exp2.bat の設定でバッチ処理での実験を行います。

行番号１９～２８：

ログファイルの圧縮・データの移動を行います。

行番号３０～４２までが実験３のための記述です。

行番号３０、３１：

実験２の終了状態から実験を行うため、実験２のデータを PAIRHEARTS/exp/exp2/から PAIRHEARTS/tp/data/、PAIRHEARTS/pp/data/にコピーします。

行番号３２：

exp3.bat の設定でバッチ処理での実験を行います。

行番号３３～４２：

ログファイルの圧縮・データの移動を行います。

行番号４４～５６までが実験４のための記述です。

行番号４４、４５：

実験３の終了状態から実験を行うため、実験３のデータを PAIRHEARTS/exp/exp3/から PAIRHEARTS/tp/data/、PAIRHEARTS/pp/data/にコピーします。

行番号４６：

exp4.bat の設定でバッチ処理での実験を行います。

行番号４７～５６：

ログファイルの圧縮・データの移動を行います。

シェルの実行

シェルファイル sample.sh を(chmod などで)実行可能状態にしてシェルを実行します。

```
% sample.sh
```

ここで示したサンプルファイル exp1.bat, exp2.bat, exp3.bat, exp4.bat, sample.sh は PAIRHEARTS/server/sample/にあります。

サーバ画面から操作を行う場合の実験手順

1．実験に参加させる TP エージェント、PP エージェントを起動するための設定が PAIRHEARTS/server/の tplist.exp,pplist.exp に書かれているかを確認します。
2．TP エージェントのデータ、PP エージェントのディレクトリ・データが PAIRHEARTS/tp/data/, PAIRHEARTS/pp/data/以下にあるかを確認します。
3．TP エージェントの学習モードを TP エディターを用いて設定します。PP エージェントの学習モードを PP エディターを用いて設定します。エディターによる学習モードの設定の仕方はドキュメント「TP エディター操作説明書 1」、「TP エディター操作説明書 2」、「PP エディター操作説明書 1」、「PP エディター操作説明書 2」を参照して下さい。
4．PAIRHEARTS/server/でハーツサーバを起動します。起動コマンドは以下の通りです。

```
% srvHearts
```

5．サーバ画面上で以下の操作を行います。
   ゲーム条件を設定します。
      Shooting the Moon の On/Off
      ゲーム終了方法
   出力ログファイル名を設定します。
   エージェントを起動します。
   TP エージェントを相手学習させる場合は相手を指定します。
   ゲームを開始します。
6．(ゲーム終了時)ハーツサーバを終了します。

サーバ画面の操作の仕方はドキュメント「サーバ画面操作説明書」を参照して下さい。

CSV 作成ツールの使用方法

CSV 作成ツールのソースファイルは全て PAIRHEARTS/exp/にあります。使用するときは、コンパイルしてツールの実行ファイルを作成して下さい。

hclassout4

ハンド分類枠（TP プロファイル枠）のデータを CSV 形式(カンマ区切り形式)で出力するためのプログラムです。ソースファイルは hclassout4.c です。実行ファイルは以下のコマンドで作成します。

```
% cc –o hclassou4 hclassout4.c
```

このプログラムでは、エージェントのハンド分類枠として定義されているハンドクラスのデータ(平均、分散、経験回数)を実験に参加した４グループまとめて上から順に１行ずつ出力します。そのため、このツールを利用するためには、４グループのハンド分類枠が全て共通であることが前提となります。(現在、TP エージェント及び PP エージェントのハンド分類枠(TP プロファイル枠)は全て共通です。)また、全てのハンド分類枠はハンドクラスであることを前提としています。ハンド分類枠についてはドキュメント「ソフトウェア機能仕様書」を参照して下さい。

hclassout4 の実行

hclassout4 の実行コマンドは以下の通りです。

```
%  hclassout4  infile  outfile
```

infile はどのハンド分類枠データファイルを使用するかを記述した入力ファイル名です。*outfile* は CSV 形式のデータを出力するための出力先ファイル名です。
infile には以下の４行を複数回（５回まで）記述します。

グループ０のバックグランド実験ファイル名　グループ０の学習実験ファイル名
グループ１のバックグランド実験ファイル名　グループ１の学習実験ファイル名
グループ２のバックグランド実験ファイル名　グループ２の学習実験ファイル名
グループ３のバックグランド実験ファイル名　グループ３の学習実験ファイル名

hclassou4 はバックグランド実験ファイル名と学習実験ファイル名が異なるときは、バックグランド実験と学習実験のデータの差分をハンドクラスのデータとあわせて出力します。学習実験ファイル名がバックグランド実験ファイル名と同じ場合は２つの実験のデータの差分は出力しません。

以下は、シェルファイルによる連続実験の例で示した４つの実験のハンド分類枠のデータを CSV 形式で出力するための infile のサンプルです。

```
exp1/tp/tpLV3.6_a0.hclass   exp1/tp/tpLV3.6_a0.hclass
exp1/tp/tpLV3.6_a1.hclass   exp1/tp/tpLV3.6_a1.hclass
exp1/tp/tpLV3.6_a2.hclass   exp1/tp/tpLV3.6_a2.hclass
exp1/tp/tpLV1.6.hclass      exp1/tp/tpLV1.6.hclass
exp1/tp/tpLV3.6_a0.hclass   exp2/tp/tpLV3.6_a0.hclass
exp1/tp/tpLV3.6_a1.hclass   exp2/tp/tpLV3.6_a1.hclass
exp1/tp/tpLV3.6_a2.hclass   exp2/tp/tpLV3.6_a2.hclass
exp1/tp/tpLV1.6.hclass      exp2/tp/tpLV1.6.hclass
exp2/tp/tpLV3.6_a0.hclass   exp3/tp/tpLV3.6_a0.hclass
exp2/tp/tpLV3.6_a1.hclass   exp3/tp/tpLV3.6_a1.hclass
exp2/tp/tpLV3.6_a2.hclass   exp3/tp/tpLV3.6_a2.hclass
exp2/tp/tpLV1.6.hclass      exp3/tp/tpLV1.6.hclass
exp3/tp/tpLV3.6_a0.hclass   exp4/tp/tpLV3.6_a0.hclass
exp3/tp/tpLV3.6_a1.hclass   exp4/tp/tpLV3.6_a1.hclass
exp3/tp/tpLV3.6_a2.hclass   exp4/tp/tpLV3.6_a2.hclass
exp3/tp/tpLV1.6.hclass      exp4/tp/tpLV1.6.hclass
```

以下は、この infile により出力されるファイルのサンプルです。

```
6.0, 56.8, 3555, 6.4, 59.5, 3591, 6.1, 58.0, 3616, 6.4, 59.7, 3626,
6.1(6.2), 56.3(55.9), 7252(3697),6.3(6.2), 58.4(57.4), 7338(3747),6.2(6.4), 58.2(58.2), 7309(3693),6.2(5.9), 58.1(56.1), 6723(3097),
6.3(6.7), 58.3(61.7), 11019(3767),6.3(6.4), 58.3(58.0), 11126(3788),6.5(6.9), 59.6(62.1), 11063(3754),5.7(4.7), 53.1(40.1), 9698(2975),
5.9(5.7), 55.1(52.5), 11031(3779),6.2(6.1), 58.3(58.1), 11117(3779),6.2(6.1), 57.5(56.1), 11028(3719),6.4(6.8), 57.9(57.0), 9727(3004),
7.4, 60.5, 1445, 7.5, 62.8, 1409, 6.9, 56.2, 1384, 7.2, 60.5, 1374,
7.4(7.5), 60.4(60.4), 2748(1303),7.6(7.6), 62.9(62.9), 2662(1253),7.3(7.7), 59.6(62.9), 2691(1307),6.9(6.6), 58.6(57.1), 3277(1903),
7.6(8.0), 61.1(62.3), 3981(1233),7.6(7.7), 62.6(61.9), 3874(1212),7.6(8.2), 61.1(63.9), 3937(1246),6.4(5.7), 53.7(44.8), 5302(2025),
7.2(6.8), 59.0(55.5), 3969(1221),7.6(7.6), 62.6(61.9), 3883(1221),7.5(8.0), 61.3(64.5), 3972(1281),7.0(7.2), 58.6(58.5), 5273(1996),
6.3, 54.6, 666, 6.4, 57.4, 630, 5.7, 49.7, 696, 6.2, 56.4, 628,
6.2(6.1), 54.6(54.7), 1235(569),6.3(6.2), 56.5(55.4), 1199(569),6.1(6.4), 52.3(55.0), 1286(590),5.7(5.5), 53.3(51.1), 1609(981),
6.3(6.6), 55.3(56.9), 1747(512),6.4(6.6), 56.4(56.2), 1727(528),6.1(6.4), 52.5(52.9), 1840(554),5.1(4.1), 43.7(27.4), 2664(1055),
5.9(5.1), 52.2(45.9), 1773(538),6.2(6.1), 55.4(53.0), 1714(515),6.4(7.1), 56.6(65.5), 1862(576),5.7(5.7), 50.3(45.5), 2633(1024),
・・・
```

１行目が１番目の実験（実験１）のハンド分類枠の最初のハンドクラスのデータをあらわします。書式は以下のとおりです。
平均, 分散, 経験回数, 平均, 分散, 経験回数, 平均, 分散, 経験回数, 平均, 分散, 経験回数,
左から３つ目までのデータがグループ０のデータ、次の３つがグループ１のデータ、次の３つがグループ２のデータ、次の３つがグループ３のデータです。

２行目は２番目の実験（実験２）のハンド分類枠の最初のハンドクラスのデータをあらわします。書式は１行目と同じですが、() 内はバックグランドとする実験（実験１）との差分データを表します。(実験２のハンド分類枠のデータは実験１の５０００回の経験ハンドと実験２の５０００回の経験ハンドを足し合わせたデータです

が、() 内は実験２の５０００回の経験ハンドだけでハンド分類枠を構成した場合のデータになっています。)

３行目は３番目の実験（実験３）のハンド分類枠の最初のハンドクラスのデータをあらわします。() 内はバックグランドとする実験（実験２）との差分データを表します。

４行目は４番目の実験（実験４）のハンド分類枠の最初のハンドクラスのデータをあらわします。() 内はバックグランドとする実験（実験３）との差分データを表します。

５行目から８行目まではハンド分類枠の２番目のハンドクラスのデータを表します。
９行目から１２行目まではハンド分類枠の３番目のハンドクラスのデータを表します。
以下、ハンド分類枠の最後までこの形式のデータが続きます。

ruleout3

PP の行動ルールの強度（の推移）を CSV 形式(カンマ区切り形式)で出力するためのプログラムです。ソースファイルは ruleout3.c です。実行ファイルは以下のコマンドで作成します。

```
% cc –o ruleout3 ruleout3.c
```

このプログラムでは、PP エージェントの行動ルールの強度（の推移）を行動ルールファイルに記述されている順に１行ずつ出力します。

ruleout3 の実行

ruleout3 の実行コマンドは以下の通りです。

```
%  ruleout3  infile  outfile
```

*infile*はどの行動ルールファイルを使用するかを記述した入力ファイル名です。*outfile* は CSV 形式のデータを出力するための出力先ファイル名です。
infile のサンプルを以下に示します。

```
3 5
exp2/pp/pp12_a3/tpLV1.6_1000.rule
exp2/pp/pp12_a3/tpLV1.6_2000.rule
exp2/pp/pp12_a3/tpLV1.6_3000.rule
exp2/pp/pp12_a3/tpLV1.6_4000.rule
exp2/pp/pp12_a3/tpLV1.6_5000.rule
exp3/pp/pp12_a3/tpLV1.6_1000.rule
exp3/pp/pp12_a3/tpLV1.6_2000.rule
exp3/pp/pp12_a3/tpLV1.6_3000.rule
exp3/pp/pp12_a3/tpLV1.6_4000.rule
exp3/pp/pp12_a3/tpLV1.6_5000.rule
exp4/pp/pp12_a3/tpLV1.6_1000.rule
exp4/pp/pp12_a3/tpLV1.6_2000.rule
exp4/pp/pp12_a3/tpLV1.6_3000.rule
exp4/pp/pp12_a3/tpLV1.6_4000.rule
exp4/pp/pp12_a3/tpLV1.6_5000.rule
```

１行目は実験の種類が３つでその実験に対し、５段階で強度を表示することを示しています。次の５行は１番目の実験に対するそれぞれの段階の行動ルールのファイル名です。次の５行は２番目の実験に対するそれぞれの段階の行動ルールのファイル名です。次の５行は３番目の実験に対するそれぞれの段階の行動ルールのファイル名です。

以下は、このinfileにより出力されるファイルのサンプルです。

```
-6.14, -6.20, -6.26, -6.29, -6.30, -4.86, -4.94, -5.02, -5.08, -5.14, -4.23, -4.42, -4.58, -4.71, -4.78,
-6.26, -6.27, -6.29, -6.31, -6.33, -4.87, -4.94, -5.02, -5.07, -5.13, -4.48, -4.52, -4.61, -4.73, -4.79,
-6.22, -6.23, -6.26, -6.29, -6.31, -4.87, -4.95, -5.02, -5.08, -5.13, -4.45, -4.50, -4.59, -4.71, -4.78,
-6.26, -6.27, -6.30, -6.32, -6.34, -4.86, -4.94, -5.02, -5.08, -5.14, -4.49, -4.53, -4.62, -4.74, -4.80,
-6.26, -6.27, -6.30, -6.33, -6.34, -4.86, -4.94, -5.02, -5.08, -5.14, -4.49, -4.53, -4.62, -4.74, -4.80,
-6.22, -6.23, -6.27, -6.30, -6.32, -4.86, -4.94, -5.02, -5.07, -5.14, -4.46, -4.51, -4.59, -4.72, -4.79,
-6.22, -6.23, -6.27, -6.30, -6.32, -4.88, -4.95, -5.03, -5.08, -5.14, -4.45, -4.49, -4.58, -4.71, -4.78,
-6.90, -6.92, -6.93, -6.96, -6.95, -5.10, -5.15, -5.19, -5.26, -5.33, -4.77, -4.91, -5.04, -5.13, -5.17,
-2.76, -2.76, -2.59, -2.59, -2.77, -5.35, -5.35, -5.78, -4.85, -4.80, -3.83, -4.14, -3.91, -3.67, -3.47,
-6.36, -6.36, -5.78, -5.78, -5.74, -6.06, -6.06, -5.93, -5.74, -5.94, -3.81, -4.11, -4.72, -4.69, -4.69,
-6.36, -6.36, -5.78, -5.78, -5.74, -6.06, -6.06, -5.93, -5.74, -5.94, -3.81, -4.11, -4.72, -4.69, -4.69,
・・・
```

１行目は１番目の行動ルールの強度です。左から５個目のデータまでが１番目の実験(実験2)での１０００回ごとの強度の推移データです。次の５個のデータまでが２番目の実験(実験3)での１０００回ごとの強度の推移データです。次の５個のデータまでが3番目の実験(実験4)での１０００回ごとの強度の推移データです。
以下、行動ルールの最後までこのデータが続きます。

tpruleout

TP エージェントの基本行動ルールのデータ(強度)を CSV 形式(カンマ区切り形式)で出力するためのプログラムです。ソースファイルは tpruleout.c です。実行ファイルは以下のコマンドで作成します。

```
% cc –o tpruleout tpruleout.c
```

このプログラムでは、TP エージェントの基本行動ルールの強度を行動ルールファイルに記述されている順に１行ずつ出力します。このツールはレベル 1.6、2.6、3.6 に与えられている基本行動ルールのデータを出力するものです。現在(2000年2月29日)レベル 1.6、2.6、3.6 に与えられている基本行動ルールに沿ってデータを出力するため、各エージェントがどのレベルかを入力として与える必要があります。また、基本行動ルールを変更する場合は、このツールのソースコードを修正する必要があります。

tpruleout の実行

tpruleout の実行コマンドは以下の通りです。

```
%  tprueout  infile  outfile
```

*infile*はどの行動ルールファイルを使用するかを記述した入力ファイル名です。*outfile* は CSV 形式のデータを出力するための出力先ファイル名です。

infile のサンプルを以下に示します。

```
2 5
exp3/tp/tpLV1.6_1000.rule 0
exp3/tp/tpLV1.6_2000.rule 0
exp3/tp/tpLV1.6_3000.rule 0
exp3/tp/tpLV1.6_4000.rule 0
exp3/tp/tpLV1.6_5000.rule 0
exp4/tp/tpLV1.6_1000.rule 0
exp4/tp/tpLV1.6_2000.rule 0
exp4/tp/tpLV1.6_3000.rule 0
exp4/tp/tpLV1.6_4000.rule 0
exp4/tp/tpLV1.6_5000.rule 0
```

１行目は実験の種類が２つでその実験に対し、５段階で強度を表示することを示しています。次の５行は１番目の実験に対するそれぞれの段階の行動ルールのファイル名と TP エージェントのレベルに対するフラグです。次の５行は２番目の実験に対するそれぞれの段階の行動ルールのファイル名と TP エージェントのレベルに対するフラグです。フラグの値はレベル 1.6 が０、レベル 2.6 が１、レベル 3.6 が２です。

以下は、この infile により出力されるファイルのサンプルです。

```
0.00(0), 0.00(0), 0.00(0), 0.00(0), 0.00(0), 0.00(0), 0.00(0), 0.00(0), 0.00(0), 0.00(0),
 ,  ,  ,  ,  ,  ,  ,  ,  ,  ,
 ,  ,  ,  ,  ,  ,  ,  ,  ,  ,
 ,  ,  ,  ,  ,  ,  ,  ,  ,  ,
-10.25(16), -10.59(29), -10.93(46), -10.94(64), -10.80(87), -8.00(6), -7.69(16), -8.57(23), -9.53(32), -10.11(47),
-4.69(13), -7.00(26), -7.63(41), -8.36(58), -8.32(74), -10.21(14), -8.00(33), -8.89(46), -9.46(56), -9.38(71),
-11.75(4), -11.75(4), -7.83(6), -10.25(8), -10.25(8), -24.00(1), -24.00(1), -24.00(1), -24.00(1), -24.00(1),
-11.75(4), -11.75(4), -6.86(7), -9.22(9), -9.22(9), -24.00(1), -24.00(1), -24.00(1), -24.00(1), -24.00(1),
-10.74(34), -12.12(67), -11.02(92), -11.12(130), -10.43(162), -10.38(8), -8.80(20), -9.59(27), -11.65(40), -12.76(62),
・・・
```

１行目は１番目の基本行動ルールの強度です。左から５個目のデータまでが１番目の実験（実験３）での１０００回ごとの強度の推移データです。() 内は重みつき和（WS）の値です。次の５個のデータまでが２番目の実験（実験４）での１０００回ごとの強度の推移データです。
２行目～４行目まではレベル 1.6 が持っていない基本行動ルールのため空白データ行になっています。
以下、基本行動ルールの最後までこのデータが続きます。

その他の実験ツール

これらのソースファイルは全て PAIRHEARTS/exp/にあります。使用するときは、コンパイルしてツールの実行ファイルを作成して下さい。

expgzip

PAIRHEARTS/exp/に置かれた実験ファイルを圧縮するためのツールです。実験回数が多い場合、経験ハンドファイル、ハンドクラス分類枠（TP プロファイル枠）ファイル、TP の行動ルールファイルの容量が大きくなります。expgzip は圧縮及び削除によりハードディスクの占有量を小さくするためのツールです。ソースファイルは expgzip.c です。実行ファイルは以下のコマンドで作成します。

% cc –o expgzip expgzip.c

このプログラムでは、PAIRHEARTS/exp/に作成された以下の構成のディレクトリを対象とします。

expXXX/
pp/
tp/

XXX は実験番号です。

expXXX/tp/には実験 XXX に参加した TP エージェントのデータファイルが格納されているものとします。また、expXXX/pp/には実験 XXX に参加した PP エージェントのデータファイルがディレクトリごと格納されているものとします。

このとき、expgzip は、expXXX/pp/の TP プロファイル枠ファイル(拡張子.prof のファイル)及び経験ハンドファイル(拡張子.hand のファイル)を、同内容のものを TP エージェントが持っているという前提のもと、削除します。(経験ハンドファイルは TP エージェントが同じ PP エージェントとしかペアを組んでおらず、また、編集ツールで削除されていないのであれば、同一です。TP プロファイル枠、ハンド分類枠も現在(2000 年 2 月 29 日)全てのエージェントで共通のため、これも同一になります。) また、expXXX/tp/の行動ルールファイル(拡張子.rule のファイル)、ハンド分類枠ファイル(拡張子.hclass のファイル)、経験ハンドファイル(拡張子.hand のファイル)を gzip で圧縮します。

expgzip の実行コマンドは以下の通りです。

% expgzip e*xpXXX*

*expXXX*は上記のディレクトリ構成のディレクトリ名です。

expgunzip

expgunzip は expgzip で圧縮・削除されたファイルを復元するツールです。ソースファイルは expgunzip.c です。実行ファイルは以下のコマンドで作成します。

% cc –o expgunzip expgunzip.c

expgunzip の実行コマンドは以下の通りです。

% expgunzip e*xpXXX*

*expXXX*は上記のディレクトリ構成のディレクトリ名です。

expgunzip は expXXX/tp/の圧縮されたファイルを伸張します。また、経験ハンドファイル、ハンド分類枠(TP プロファイル枠)ファイルを expXXX/pp/にコピーします。