強化学習を用いた
拡張型七五三ゲームの自動生成

執筆者：田邊　繁

指導教官：新納　浩幸

平成１４年３月１日

目次

図目次

表目次

第１章

序論

１．１　概要

現在，機械学習において様々な問題の解決が試みられているが，その多くは分類問題である．しかし，現実の問題では分類問題として定式化できない問題も数多く存在する．

その中の一つに相互作用型の問題というものがある．これはある目的を持ったエージェントと提供される環境が相互に影響を与えあう問題で，強化学習を用いることが問題解決に有効である．

本研究では強化学習を実際に相互作用問題に適用できるようにすることを目的としてその例を解くものである．

これには状態数が非常に多数である場合の汎化手法が重要な問題となっている．そこで，今回は七五三ゲーム及び七五三ゲームを拡張したゲームの最適な行動を検討することとする．次の流れで実験を行う．

１　七五三ゲームでの最適解の取得

これにより実際に相互作用問題に対して強化学習を利用することができることを確認する．また強化学習の手法を学ぶ．

2　拡張型七五三ゲームでの最適解の取得

最も重要な問題である状態数が非常に多い問題の場合の汎化を検討し，最適方策の取得を試みる．

本研究の目的は状態数が非常に多い問題である拡張型七五三ゲームの最適な方策を，強化学習を用いて取得することである．

１．２　本論文の構成

第２章において教科学習について述べる．強化学習にはいくつかの手法がありそれぞれについての説明をする．本研究では最終的にはTD学習を使用する．

第３章では汎化の手法として今回利用したニューラルネットワークについて述べる．基本的な内容と実際に用いるアルゴリズムである逆誤差伝播法について説明する．

第４章では七五三ゲームにおける最適方策を得るための実験について述べる．

第５章では拡張型七五三ゲームの最適方策の取得を試みる実験について述べる．概要，工夫点，また実験に使うアルゴリズムも同時に説明する．また実験結果とその評価を述べる．

第６章では実験の結果の考察，さらに本研究の目的である状態数が多数である問題の最適方策を得るための方針を述べる．

第２章

強化学習

２．１　強化学習

ここではまず強化学習について説明する．

まず，強化学習とは相互作用から学習して目標を達成する問題の枠組みそのものである．そこで最初に問題の枠組みから説明する．学習と意思決定を行うものは「エージェント」と呼ばれる．このエージェント外部から構成され，エージェントが相互作用を行う対象は「環境」と呼ばれる．両者は継続的に相互作用を行い，エージェントは行動を選択し，環境はその行動に応答し，エージェントに新しい状況を提示する．環境は報酬の発生源でもあり，この報酬はエージェントが時間の経過の中で最大化しようとする特別な数値である．エージェントの目標は最終的に受け取る報酬を最大化することである．

もっと詳細に言うならば，エージェントと環境は離散的な時間スッテプ．$t = 0,1,2,3...$の各々のおいて相互作用を行う．各時間スッテプtにおいて，エージェントは何らかの環境の状態(state)の表現 $s_t \in S$ (S は可能な状態の集合)を受け取り，これに基づいて行動 $a_t \in A(s_t)$を選択する($A(s_t)$は状態 $s_t \in S$ において選択することのできる全ての行動の集合である)．１時間スッテプ後に，エージェントはその結果として数値化された報酬 $r_{t+1} \in \Re$ を受け取り，新しい状態 s_{t+1} にいることを知る．図にエージェントと環境との相互作用を示す．

この枠組みは抽象的かつ柔軟で，種々の問題に対して種々の方法で適用することができる．例えば，時間スッテプを固定間隔の実時間で参照する必要はない．時間スッテプは意思決定と行動の連続的な任意の段階とすることもできる．

このように考えることによって今回用いる七五三ゲーム，もしくはその他のゲームに対応できる．

次に学習について述べる．

強化学習問題を解くことは大雑把に言えば，最終的に多くの報酬獲得を達成するような方策(最適方策)を見つけることを意味する．これを価値関数に基づく評価により，行っている．この関数は状態(あるいは状態行動対)の関数で，エージェントがある状態にいることがどれだけ良いのか(もしくはある状態において．ある行動を行うことがどれだけ良いか)を評価する．全ての状態に対する最適な価値関数が得られたなら最適方策を得ることは容易である．１スッテプ探索によって最適な行動が得られるのだ．

よって，強化学習とは最適な価値関数を得ることによって最適方策を得るものであるということができる．

次に最適価値関数を得るための学習アルゴリズムについて述べる．

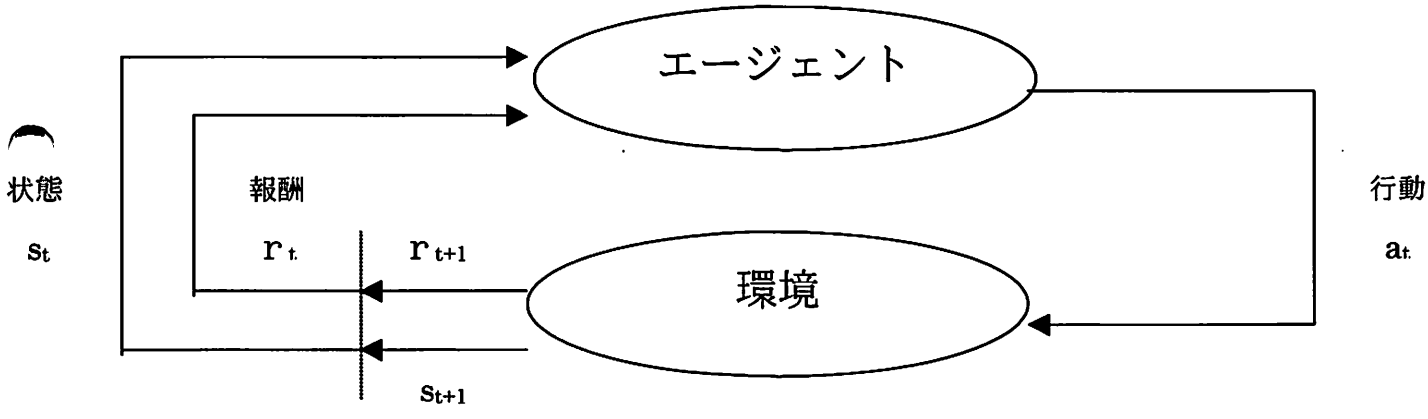

図２．１　強化学習におけるエージェントと環境間の相互作用

２．２　動的計画法

２．２．１　動的計画法

動的計画法(ＤＰ)とは，最適方策を計算するためのアルゴリズムであり，環境の完全なモデルがマルコフ決定過程として与えられる場合に適用できる.ＤＰアルゴリズムは、完全なモデルを前提としていて計算量も多くなる古典的なものだが理論的にはこの後の方法につながる重要なものである.

まずマルコフ決定過程について述べる.

状態信号が過去の知覚をコンパクトに集約しさらに過去の関連情報を全て保持していることをマルコフ性を持つという.このマルコフ性を満たす強化学習タスクをマルコフ決定過程という．また状態と行動空間が有限であれば有限マルコフ決定過程となる．　ＤＰは主に方策評価，方策改善，方策反復の三つからなる.

手法としてはさきも述べたように最適価値関数を用いたものである．価値関数には状態価値関数と行動価値関数があり状態価値関数は状態そのものの価値行動価値関数はその行動がどの程度有効かを表す．下に式に示す.

$$V^*(s) = \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) | s_t = s, a_t = a\}$$

$$= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$$

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_a Q^*(s_{t+1}, a') | s_t = s, a_t = a\}$$

$$= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_a Q^*(s', a')]$$

２．２．２　方策評価

方策評価とは任意の方策πに対する状態価値関数V^{π}を計算するものである．これは予測問題とも呼ばれる．
まず，状態価値は以下の式で示す．

$$V^{\pi}(s) = E_{\pi}\{r_{t+1} + \gamma r_{t+2} + \cdots | s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')]$$

$\pi(s,a)$は方策πの下で，状態 s で行動ａを選択する確率を表す．添え字πはπに関する条件付であることを示す．割引率$\gamma<1$であるか，または方策πの下で最終的にすべての状態から終端状態に至ることが保証されるなら，V^{π}の存在と一意性は保証される．

環境のダイナミクスが完全に知れている場合これは簡単に求められる．計算時間のかかる場合もあるが，反復解法を用いるのが適している．V^{π}を１ステップ観測で以下のように近似できる．これを全ての状態を対象としてバックアップを行うので完全バックアップと呼ぶ．

$$V_{k+1}(s) = E_{\pi}\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s\}$$
$$= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

これを各状態ｓに適用してV_kから続くV_{k+1}への近似を得る．つまり以前のｓの価値と即時報酬の期待値を使って新たなｓの価値を計算し，それを以前のｓの価値と置き換え，現在評価している方策の下で可能な 1 スッテプ遷移のすべてに対してこの操作を行う．このような操作を完全バックアップと呼び，これを方策評価の中でも特に，反復方策評価と呼ぶ．

評価対象の方策πを入力
すべての$s \in S$に対して$V(s) = 0$とする
繰り返し:
$\Delta \leftarrow 0$
各$s \in S$について:
$v \leftarrow V(s)$
$V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
$\Delta < \theta$(正の小さな値)　ならば，繰り返しを終了

図２。２　反復方策評価アルゴリズム

２．２．３　方策改善

方策に対して評価関数を計算することで，さらに良い手がかりが得られる．いま決定論的な方策πに対して，価値関数V^{π}がきまっていたとする時，ある行動$a \neq \pi(s)$を行うことが良いことかを知ることがそのまま方策の改善につながる．そこで状態 s で行動 a を選択し，その後は既存の方策πに従うことで得られるのが以下の式でこれを行動価値関数という．

$$Q^{\pi}(s,a) = E_{\pi}\{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\}$$
$$= \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^{\pi}(s')]$$

この値を$V^{\pi}(s)$と比べた場合の大小が判断基準となる．つまり，これを利用して，すべての状態で，すべての可能な行動に関して，各様態で$Q^{\pi}(s,a)$が最良となる行動を選択する変更が有効である.このようにして決定した方策をグリーディ方策といい次式で表せる．

$$\pi'(a) = \arg\max_a Q^{\pi}(s,a)$$

$$= \arg,\max_a E\ \{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s, a_t = a\}$$

$$= \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')]$$

この方策改善をすることにより最適な方策へ収束していく．

2．2．4　方策反復

これまでの流れで方策評価によって得られたV^{π}を使って方策改善を行い，より良い方策π’を得できた．これを反復して行うことで最適方策を得るのが方策反復である．以下に詳しいアルゴリズムを載せる．

$$\pi_0 \xrightarrow{E} V^{\pi 0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi 1} \xrightarrow{I} \pi_2 \rightarrow \cdots \rightarrow \pi^* \xrightarrow{E} V^*$$

のようにπが更新されていく．ここで$\xrightarrow{E}$は方策評価を$\xrightarrow{I}$は方策改善を表す．

この場合の方策評価の開始点は一つ前に方策評価をして得た値となるので方策評価の収束スピードは 2 回目からは大幅に速くなる．また、この繰り返しで得られた方策は常に最小ステップで終端状態へ行くような最適方策となる．

しかし，大規模な問題においては方策評価と方策改善の繰り返しがたくさん必要となってしまうことがある．

以下に方策反復のアルゴリズムを示す．

1. 初期化

$s \in S$に対して$V(s) \in \Re$と$\pi(s) \in A(s)$を任意に初期化する

2．方策評価

3. 方策改善

$policy - stable \leftarrow true$

各$s \in S$について：

$b \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

もし$b \neq \pi(s)$ならば$policy - stable \leftarrow false$

もし$policy - stable$ならば，終了;それ以外は、2から繰り返す

図2．3　反復方策アルゴリズム

2．2．5　価値反復

先ほど述べたように反復方策は大規模な問題だと手間がかかる．しかも，方策評価の際にスイープ操作を繰り返し行わなければならない．さらに厳密な収束は極限においてのみ得ることができるため，その際の反復の停止条件にも疑問が残る．

そこで考えられたのが価値反復である．これは方策評価と方策改善を同時に行うもので常に最適な行動，つまりグリーディ方策のみを1スイープ操作するものです．厳密に言えば，価値観反復とは方策改善と方策評価のステップ打ち切りを組み合わせたものです．これはきわめて単純なバックアップ操作で，次の式で表される．

$$V_{k+1}(s) = \max_a E_\pi \{r_{t+1} + \gamma V_k(s_{t+1}) | s_t = s, a_t = a\}$$

$$= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V_k(s')]$$

厳密には方策評価同様に無限回の反復で極限となりV*へ収束するが，実際の終了条件は，一回のスイープ操作による価値関数の変化量が十分に小さくなった時とする．

この価値反復は有限のマルコフ決定過程である強化学習問題にはとても有効である．

また動的計画法のモデル図を載せる．

Vを任意の値で初期化
繰り返し:
　$\Delta \leftarrow 0$
　各$s \in S$について：
　　$v \leftarrow V(s)$
　　$V(s) \leftarrow \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$
　　$\Delta \leftarrow \max(\Delta | v - V(s))$
$\Delta < \theta$(正の小さな数)ならば，繰り返しを終了

次式の決定論的方策πを出力
　$\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

図２．４　価値反復アルゴリズム

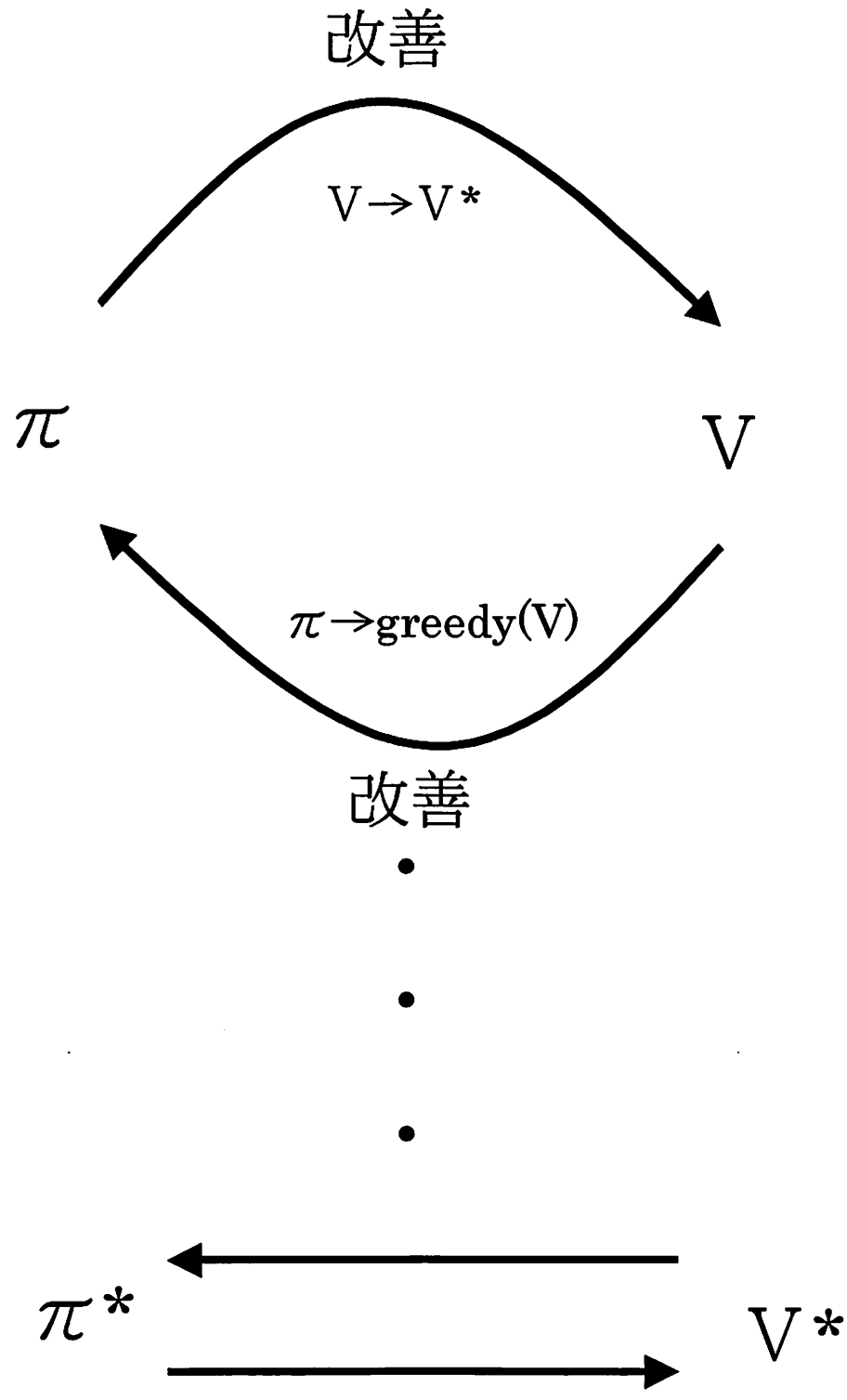

図２．５　動的計画法の学習モデル

2．3　モンテカルロ法

前述の動的計画法は環境の完全なモデルが必要な学習アルゴリズムであったが，モンテカルロ法は環境のダイナミクスを必要としない，経験のみを必要とするものです．つまり、モンテカルロ法で必要としているのは遷移のサンプル生成だけで，遷移の確率分布がわからない場合にも使うことが出来ます．

モンテカルロ法は，サンプル収益を平均化することに基づいて強化学習を解く方法であり，また，エピソード的なタスクのみ定義できます．経験はエピソードごとで，どのような行動が選択された結果，どのような最終結果を得るかを元にエピソードの終了時に価値の推定と方策の改善がされます．

モンテカルロ法では大数の法則から期待値を推測するのであらゆるすべての状態に無限回訪れることが理想です．しかし，方策によっては訪れることの出来ない状態がある場合があります．そこでε-グリーディ方策が用いられることがあります．これは確率εでランダムな行動をとるもので，これにより幅広い行動を助ける事ができる．

初期化：
　π ← 評価すべき方策
　V ← 任意の状態価値関数
　returns(s) ← 空のリスト(すべてのs∈Sに対して)

繰り返し：
　(a)πを用いてエピソードを生成する
　(b)エピソード中に出現する各状態sについて：
　　R ← sの発生後の収益
　　*R*を*returns*(s)に追加する
　　$V(s)$ ← *average*(*returns*(s))

図2．6　モンテカルロ法アルゴリズム

２．４　ＴＤ学習

２．４．１　ＴＤ予測

動的計画法，モンテカルロ法これらを進化させたのがＴＤ学習法である．ＴＤとモンテカルロ法は，いずれも経験を用いて予測問題を解決し，方策πに従って経験をいくつか得ることでV^{π}の推定値Ｖを更新する．しかし，ＴＤはモンテカルロ法とは違い，エピソードの終わりまで更新を待つ必要がない．動的計画法と同様にそれまでの学習から得た次の状態の推定値を用いて１スッテプごとに更新する．これを，ブートストラップという．

つまり動的計画法とモンテカルロ法をあわせたのがＴＤと言うことが出来る．

時刻ｔで訪問した非終端状態を s_t として，その後に起きる出来事によってＶ(s_t)を更新する．式に表すと次のようになる．

$$V(s_t) \leftarrow V(s_t) + \alpha[R_t + V(s_t)]$$
$$V(s_t) \leftarrow V(s_t) + \alpha[r_{+1t} + \gamma V(s_{t+1}) - V(s_t)]$$

αはスッテプサイズ･パラメータを表す定数で，更新する幅を指定する．

また状態価値関数は以下になる．

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\}$$

$$= E_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\}$$

$$= E_{\pi}\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) | s_t = s\}$$

これを利用してV^{π}の推定値を予測する．

$V(s)$を任意に初期化し，πを評価対象の方策に初期化する
各エピソードに対して繰り返し:
　sを初期化
　エピソードの各スッテプに対して繰り返し:
　　$a \leftarrow s$に対してπで与えられる行動
　　行動aをとり，報酬rと次状態s'を観測する
　　$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
　　$s \leftarrow s'$
　sが終端状態ならば繰り返し終了

図２．７　ＴＤを用いたV^{π}の予測アルゴリズム

２．４．２　方策オン型ＴＤ制御

次にＴＤ予測法を制御問題に適用する方法について述べる．評価を予測部分にＴＤ法を利用し，一般化方策反復の考え方を適用していく．この時，調査と知識の利用のトレードオフの問題がある．これに対するアプローチには方策オン型と方策オフ型の二つの手法がある．

方策オン型のＴＤ制御では，現在の方策πにおける状態ｓと行動ａの全てに対して$Q^{\pi}(s,a)$を評価しなければならない.これはＴＤ予測法を行動価値関数に置き換えることによって簡単に定義できる.

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

この更新式は状態が遷移するたびに更新されるが,とるべき行動はエピソードの前にあらかじめ決められた行動となる.つまり,はじめは$Q^{\pi 1}$そった行動を行い,次のエピソードでは$Q^{\pi 2}$のそった行動を採る.

ある状態行動対から次の状態行動対への遷移を定義する

$$s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$$

から Sarsa と呼ばれる.

2．4．3　方策オフ型ＴＤ制御

方策オン型はエピソードの開始時点ですべての状態に対する行動を観測しておくものだったが，それに対して方策オフ型はステップごとに行動を決定する．今回は方策オフ型の手法を用いてＴＤ学習を行うこととした．これを特にＱ学習と呼ぶ.

Ｑ学習は以下のように定義される.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a)]$$

ここで学習される行動価値関数は，使われている方策とは独立に，Q^*を直接近似する．これによって１ステップごとに新しい方策を生成，使用することになるので，収束が早まる．今回の実験ではこれを状態価値関数に置き換え使用した.

$Q(s,a)$を任意に初期化
各エピソードに対して 繰り返し：
　sを初期化
　エピソードの各ステ ップに対して繰り返し：
　　Qから導かれる方策を使 い，
　　　　sでの行動aを選択する
　　行動aを取り，報酬rと次状態s'を観測する
　　$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
　　$s \leftarrow s'$
　sが終端状態ならば繰り 返しを終了

図２．８　Q学習：方策オフ型ＴＤ制御アルゴリズム

図２．９　TD 学習のモデル図

第3章

ニューラルネットワーク

3．1　ニューラルネットワークの基礎

今回，状態をニューラルネットワークを用いた関数近似とするため，ニューラルネットワークに基礎と実験で使用したバックプロパゲーション法について述べる．

ニューラルネットワークとは，生物の神経系の特徴的な機能に着目してそのモデル化を行ったものである．簡単に言うとある信号がシナプスへ入力として送られるとシナプス結合強度により変化した信号が出力される，ニューロンをモデル化したものである．

このニューロンモデルを，複数接続することによってニューラルネットワークを作ることができる．このとき，ユーロンの接続のしかたでネットワークの構造と特徴が決まる．ニューラルネットワークは構造によって大きく分けて2つの種類に分けられる．1つは階層型ネットワークといわれるもの．これはニューロンを層状にならべ，前の層から次の層へ一方向にのみ信号が伝わるというネットワークである．このタイプのネットワークは入力層のニューロンに加えた信号（入力信号）に対して出力層のニューロンの出力（出力信号）が一意的に定まる．今回使用するバックプロパゲーション法はこれにあたる．

他には相互結合型ネットワーク，フィードバック付き相互結合型ネットワークがある．これらは流れが一方向に定まっていず，また，フィードバックを持つ．

３．２　バックプロパゲーション法

バックプロパゲーション法は階層型のニューラルネットワークで最急降下法と呼ばれる，誤差Ｅが小さくなるようにする結合荷重ωを求める方法を利用したものである．この方法では、関数のある点での勾配を求めて誤差Ｅが少なくなる方向へ結合荷重ωを変化させていくこのとになる．

ＢＰで最急降下法の考え方を使うために，１つ準備しておくことがある．それは，微分可能な連続関数を用意することで，そこで使うものが関数をシグモイド関数である．これを用いると連続な関数になると共に出力が０～１の値になる．

$$f(x) = \frac{1}{1+\exp(-x)}$$

上の関数がシグモイド関数である．

次にバックプロパゲーション法での出力について述べる．これは

$$x_{ip} = h_i(z_{ip})$$
$$z_{ip} = \sum_j \omega_{ij} y_{jp}$$

で求められる．これは入力側のニューロンの値とそれぞれの層の間の結合強度をかけた値，これをすべて足したものこれが出力となることを表している．関数ｈはシグモイド関数である．このように出力は前向きの演算を行うことによって求められる．

次に近似関数の更新．これは先ほども述べたように結合荷重ωによる近似関数なのでωを更新していく．まず，簡単に仕組みを説明する．

ある入力と出力をの時，ＢＰ法は教師あり学習なので，本来出力されるべき値，教師信号がある．この教師信号と出力の値の誤差を小さくしていくのが最急降下法という手法である．この誤差(Ｐ番目の入力に対するものとする)を誤差関数$E_p(\omega)$

といい，すべての入力に対する誤差を相誤差関数E(ω)と言う．

$$E_p(\omega) = \frac{1}{2}\sum_i (\bar{x}_{ip} - d_{ip})$$

$$E\ (\omega) = \sum_p E_p(\omega)$$

ＢＰ法はこの相誤差関数を最小化するため，各パターンに対する誤差関数の勾配ベクトルを計算し，その逆方向にωを改良する．ωはいくつもの成分を持つ多次元の関数なのでそれぞれの成分ごとに考える．

$$\frac{\partial E_p(\omega)}{\partial \omega_{ij}} = \frac{\partial E_p(\omega)}{\partial z_{ip}} \frac{\partial z_{ip}}{\partial \omega_{ij}}$$

$$= \frac{\partial E_p(\omega)}{\partial z_{ip}} y_{jp}$$

$$= -\delta_{ip} y_{jp}$$

となる．よって$-\delta_{ip}$を求めることによってωのそれぞれの変化させる量がわかることになる．この値は第ｉニューロンが出力層の場合とそうでない場合によって違う値となる．

(ｉ)第ｉニューロンが出力層の場合

出力x_{ip}は相誤差関数における$\bar{x}_{ip}$に相当するので

$$\frac{\partial E_p(\omega)}{\partial \omega_{ij}} = \bar{x}_{ip} - d_{ip}$$

となるので

$$\delta_{ip} = -(\bar{x}_{ip} - d_{ip}) h_i'(z_{ip})$$

となる．

(ii) 第 i ニューロンが出力層でない場合

第 i ニューロンを s 層とすると出力 x_{ip} が変化することによって(s+1)層の各ニューロンへの入力の総量 z_{kp} が変化し(s+1)層の各ニューロンからの出力も変化する．よってこれらの変化が(s+1)層以降のニューロンを伝わっていき $E_p(\omega)$ の値を変動させる．

$$\frac{\partial E_p(\omega)}{\partial x_{ip}} = \sum_k \frac{\partial E_p(\omega)}{\partial z_{kp}} \frac{\partial z_{kp}}{\partial x_{ip}}$$

$$\frac{\partial E_p(\omega)}{\partial z_{kj}} = \frac{\partial E_p(\omega)}{\partial x_{kp}} h_k'(z_{kp}) = -\delta_{kp}$$

$$\frac{\partial z_{kp}}{\partial x_{ip}} = \omega_{ki}$$

以上の式から，

$$\delta_{ip} = h_i'(z_{ip}) \sum_k \delta_{kp} \omega_{ki}$$

となる．

よって，以下のような形で記述される結合強度ベクトル変更則により変更される．η は固定された値で，更新する幅，ステップ幅を表している．

$$\omega^{(k+1)} = \omega^{(k)} - \eta \frac{\partial E_p(\omega)}{\partial \omega_{ij}} \Big| \omega = \omega^{(k)}$$

ただし，BP 法では ω の初期値によっては，$E(\omega)$ の大域的最小値へ収束するとは限らず，局所的最小値に陥ってしまう場合がある．

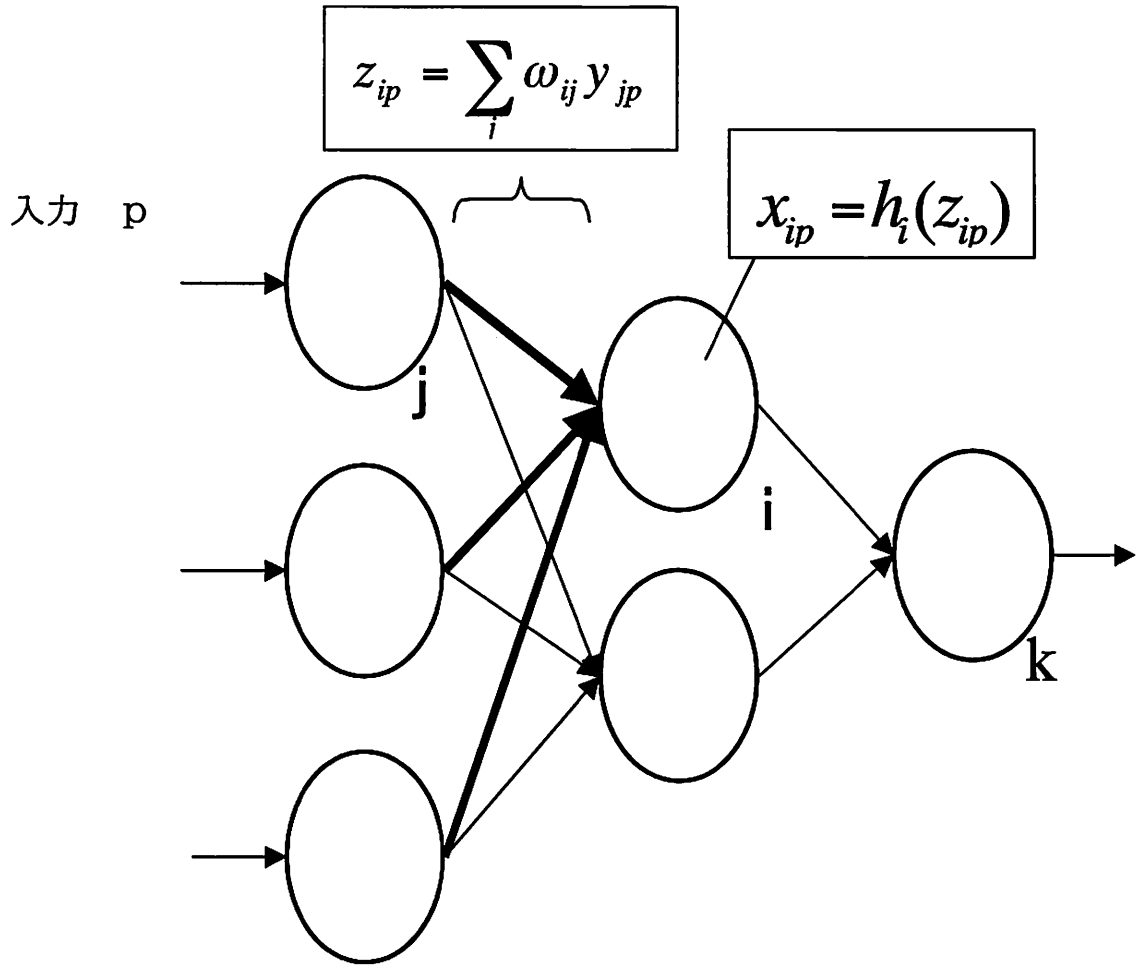

図３．１　バックプロパゲーションにおける出力のモデル図

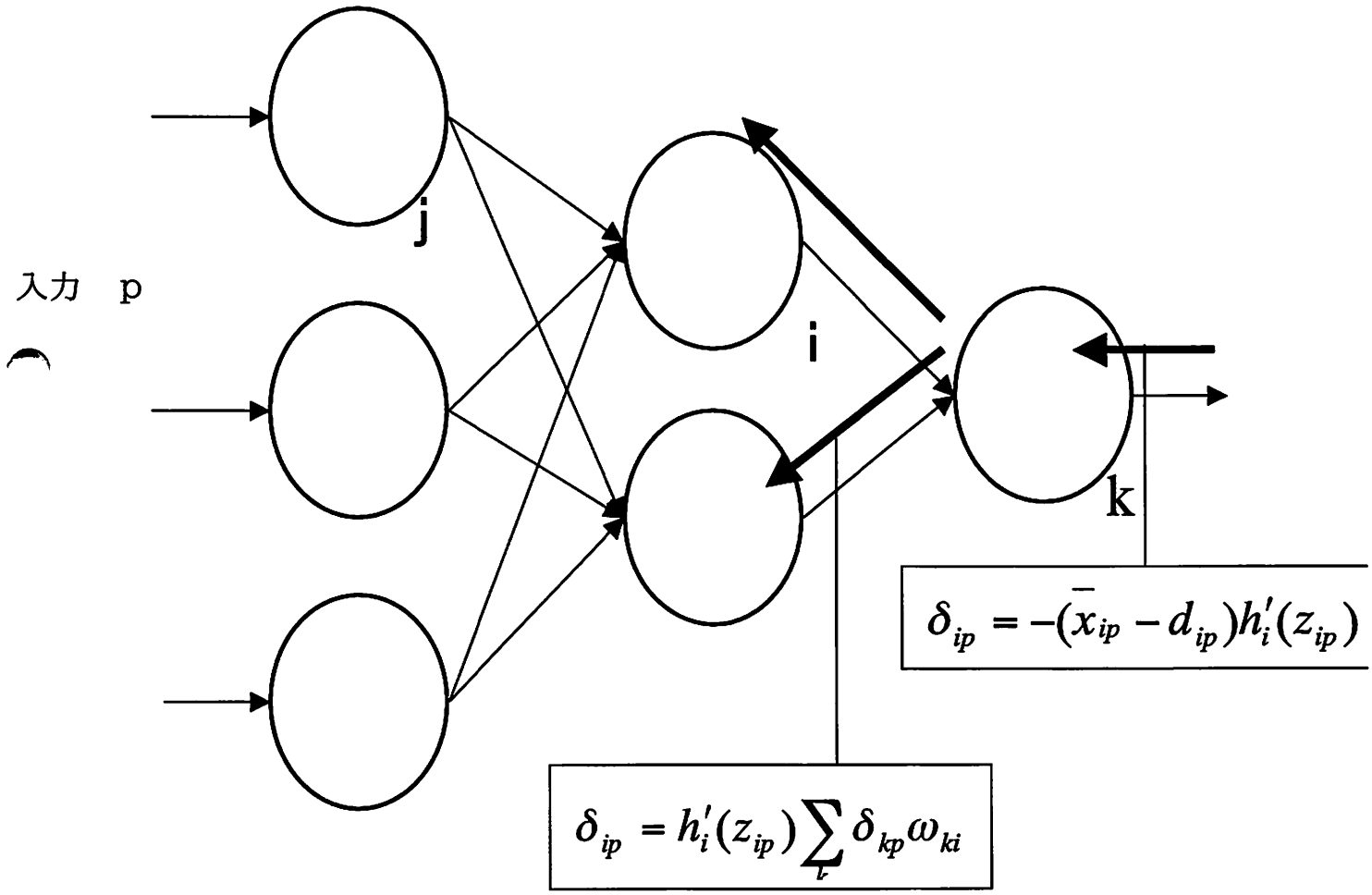

図３．２　ＢＰ法におけるωの更新のモデル図

第４章

七五三ゲーム

４．１　七五三ゲーム

一つ目の実験として実際に七五三ゲームにおける最適方策の取得を行う．

まず，七五三ゲームについての説明をする．七五三ゲームは３本，５本，７本の棒を二人で交互に消していくゲームである．最後の棒を消した方が負けとなる．細かいルールは，一つで棒の消し方である．これにはいくつかの規則があり一度に消す棒の数が制限される．まず，棒を横に並べた塊(それぞれ 3 本，5 本，7 本)がある訳だが，棒を複数消す場合にはそれらの塊の一つからしか消すことが出来ない．また，隣り合う棒がすでに消されている場合，それを飛び越えてさらにもう一つ隣の棒を消すことは出来ない．

このゲームではすでに必勝法，つまり最適方策が存在するので，ここでは強化学習による最適方策の取得が有効であることの確認，検討を行う．

４．２　最適方策の取得

次の流れにそって最適方策取得，取得した方策の評価をする

（１）七五三ゲームのモデル化
（２）DP による強化学習
（３）VB によるインターフェイスの作成

４．２．１　七五三ゲームのモデル化

七五三ゲームを数学的なモデルに置き換える．七五三ゲームのような盤面によるゲームはすべての盤面の状態を変数として持つことがもっとも簡単である．しかし，それでは状態数が増え計算量が増えることになる．よって，ゲームの特性に合わせて状態を定義することが有効になる．

七五三ゲームでは「何本の棒の集まり(消されていない棒)がいくつあるか？」がもっとも重要になるため，すべての状態を「Ｎ本の集まりがM個残っている状態」として定義する．

次にこのモデルにそった状態の抽出と，それぞれの状態における行動とそれによる状態の遷移を抽出する．状態と状態の遷移の抽出には JAVA を用いたプログラムを作成．状態は，すべての状態における N(１～７)に対する M を調べ重複を取り除くことで抽出する．行動は，対象とする N 本の塊をどのように消すか，つまり消す対象と消し方の二つに分けた行動を行い，それぞれの行動後の状態を観測，先に得た状態と対応させた．以下の結果となる．

状態数	２５１
行動(すべての状態において共通)	５１

表４．１　七五三ゲームにおける状態数と行動

４．２．２　ＤＰによる強化学習

七五三ゲームは状態数２５１と比較的小さな問題であり　環境(行動後の詳細な遷移の分布)が得られる．また七五三ゲームはすべての状態においてすべての棒が消される終端状態への遷移が約束されたマルコフ決定過程なのでＤＰを用いることができる．

アルゴリズムは反復方策を用いる．方策評価を行う value メソッドと方策改

善を行う kaizen メソッドからなる．ゲーム性からγ=0 で割引はないものとする．また方策評価の際の終了条件は$\theta = 0.001$とする．

巻末にプログラムのソースを添付する．

４．２．３　ＶＢによるインターフェイスの作成

ＶＢを使い強化学習で導き出した方策を実行するゲームを作成する．この方策とはすべての状態の最適勝ち関数を指す．ゲームでは先攻，後攻を任意に選択でき，それぞれの場合での結果を評価する．

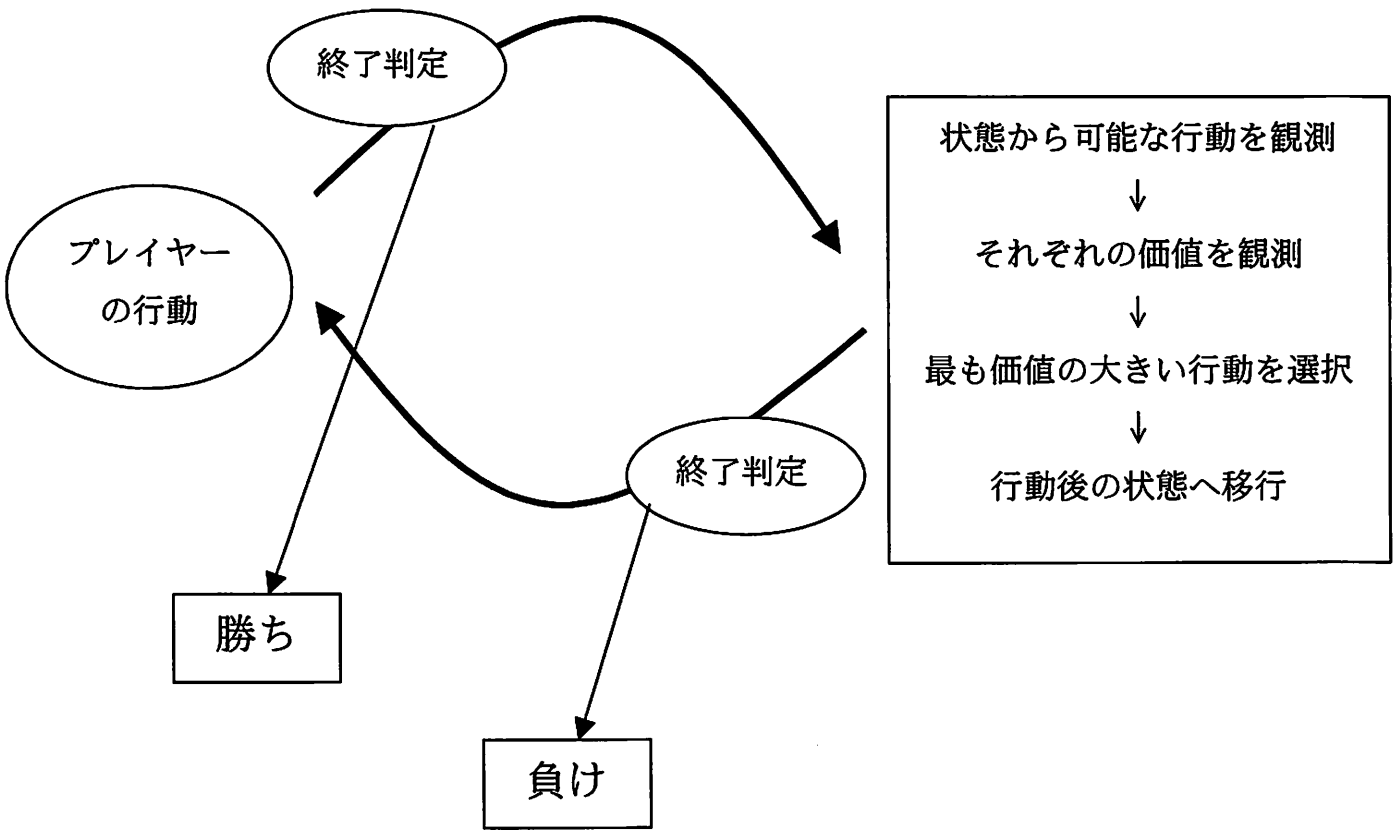

図４．１　VB による七五三ゲームの流れ

４．３　　評価

実際にＶＢによる最適方策を搭載したゲームでの試行を行った結果，最適な方策がエラ亭たことが確認された．

後攻(プレイヤー)の場合，このゲームで勝つことは不可能となった．これは既存の七五三ゲームの必勝法とも合致する．先攻(プレイヤー)においても，必勝法を知らない者だと，つまり一度でも手を間違えると勝利することは出来なかった．

このことから強化学習を用いることによって七五三ゲームの最適方策を得ることができることが確認された．

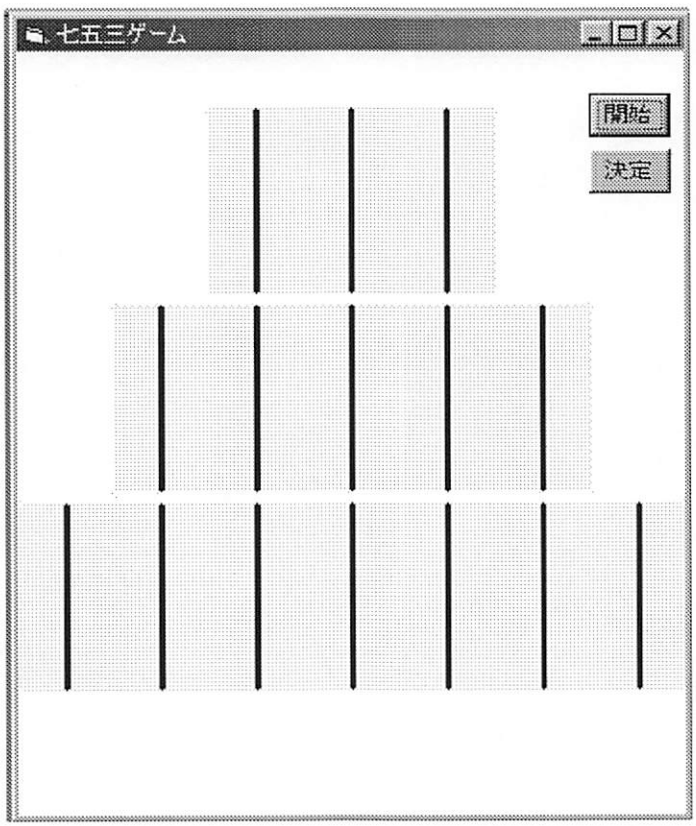

図４．２　ＶＢによる七五三ゲームインターフェイス

第５章

拡張型七五三ゲーム

５．１　　拡張型七五三ゲーム

先ほどは七五三ゲームの最適方策の取得を行った．ここで状態数が非常にｓ多数である問題として拡張型七五三ゲームを提案する．

拡張型七五三ゲームとは，先ほどの七五三ゲームを拡張したもので七五三ゲームが３，５，７本の棒の集まりを状態としたのに対して，８行８列の升目状の状態をもつゲームである．

これによって状態数はすべてのマスの状態が，消されたか，消されていないかによって決定するので．

$2^{(8\times8)} = 2^{64}$

の状態数を持つことになる．さらにルールを一部改良した．升目状にしたことにより一度に複数のマスを横方向だけでなく，縦方向にも消せるように変更した．

その他の基本的なゲームの流れは七五三ゲームと同様とする．

実験は以下の流れで行う．

（１）拡張型七五三ゲームのモデル化
（２）学習アルゴリズムの模索
（３）学習プログラムの作成
（４）取得した最適方策の評価

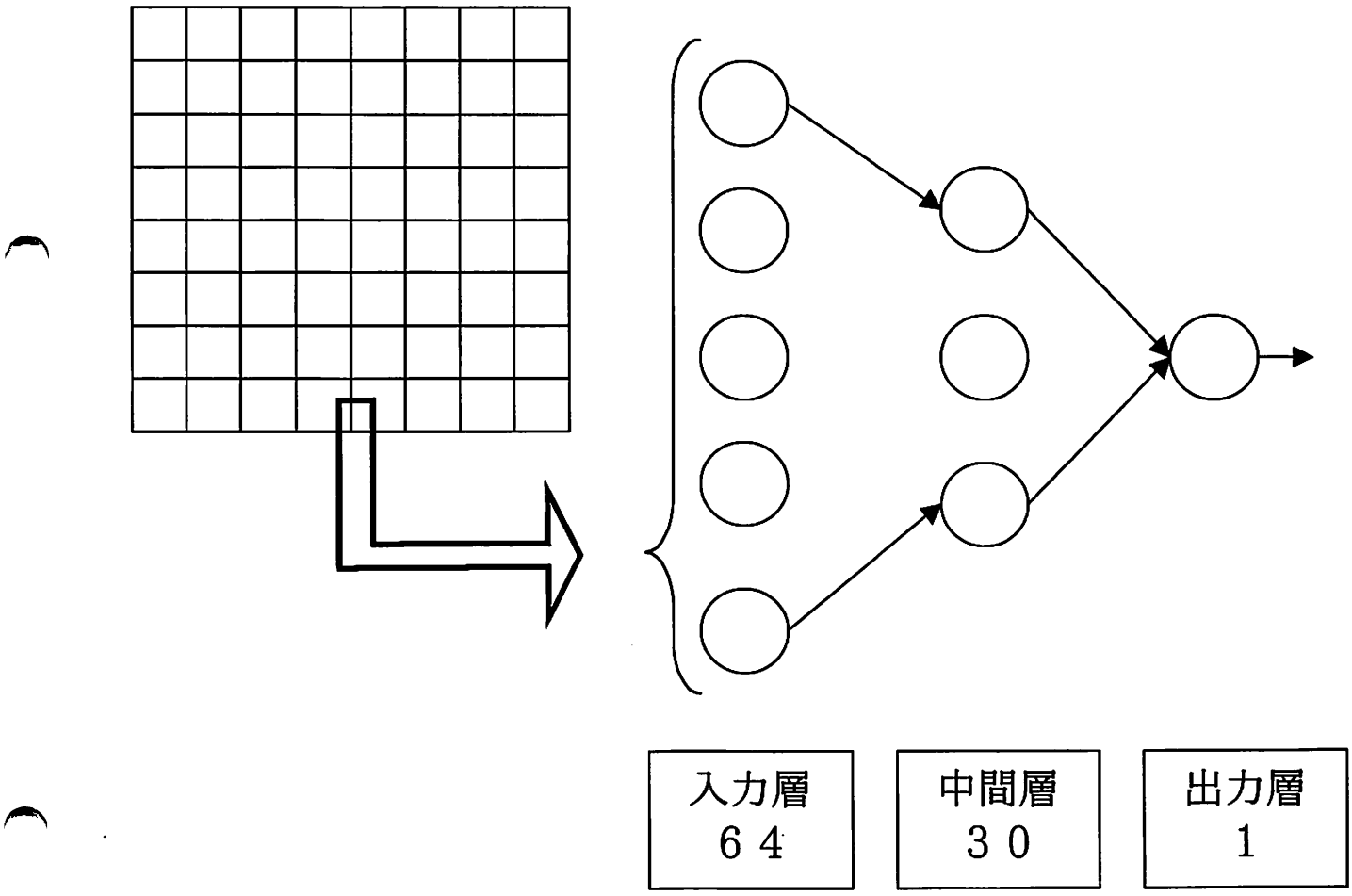

図5．1　ニューラルネットワークを用いたモデル化

５．２　　モデル化

拡張型七五三をモデル化しなければならない訳だが，七五三ゲームと同様に等価値の状態動詞をまとめたとしても，状態数が非常に多くすべての状態を変数として持つことは難しくメモリの確保も出来ない．

また状態の遷移も複雑で完全な状態の遷移モデルを作るのは困難である．

よって，サンプル生成だけを必要とするＴＤ学習とニューラルネットワークを使った近似関数による汎化を用いてモデル化を行う．

ニューラルネットワークでの近似関数のωはランダムに初期値を決定することにする．ニューラルネットワークの設定は以下の表のとおりとする．これはニューラルネットワークの学習の際のデータと参考資料から決定した．

入力層	６４（８×８）
中間層	３０（入力半分程度）
出力層	1
ステップ幅	０．１３
ωの初期値	ランダム

表５．１　ニューラルネットワークの設定

５．３　学習アルゴリズム

５．２のような近似関数を用いてＴＤ学習を行う．よってＴＤ学習を近似関数を用いた形に改良する必要がある．

ＴＤとは簡単に言えば，次状態の価値を観測し，現在の状態の価値をそれに近づけることによって最適状態価値観を見つけることによって最適方策を得るものである．

これに対してニューラルネットワーク，特にＢＰとはある入力に対する出力

を教師信号と比較し，この誤差からωを改善していくものである．

両者に共通する点は，向かうべき目標値と現在提示されている推定値の誤差から改善を行うことである．そのことからＴＤ学習から今日しい信号を生成するという手法を試みることとする．

ニューラルネットワークのＢＰ法は本来教師あり学習である．しかし，強化学習は教師を必要としない学習なのでＢＰ法における教師を与えることが出来ない．そこで，ＴＤ学習の手法を用いて教師信号を生成することにする．

具体的に言うと，ある状態と次状態があった時，次状態を入力した時の出力をそのまま，現在の状態を入力した時の教師信号として利用する．これをエピソードの毎ステップ行う．そして状態が終端状態になった時，終端状態を入力，報酬である０を教師信号としてＢＰによる関数近似を行う．

するとまず終端状態を入力とした時の出力が０へと改善され，以降はＴＤ学習のように一つずつ前の状態へと伝播されて行く．

この手法を用いて学習プログラムを作成．作成にはJAVAを用いる．

評価方法として，学習プログラムから得られた近似関数を用いて次状態の価値を観測しグリーディ方策をとるものと，ランダムな行動を選ぶプログラムとの対戦を一万回戦行うプログラムを作成．先手後手を順番に行う．同様にJAVAを用いて作成．

またVBによるインターフェイスも作成する．

５．４　評価結果

一万回の対戦を行う評価プログラムの結果，勝率はわずかに５５％しか得ることは出来なかった．また最適方策が得られればどちらかが絶対的に有利になると思われるが，今回の実験では先手，後手どちらもほぼ変わらない結果となった．　またＶＢにより作成したゲームにおいては人間に勝つことはほとんどなかった．

ニューラルネットワークの結合荷重ωをランダムに初期化

エピソードの対して繰り返し:
  状態sを初期化
  各ステップに対して繰り返し:
    sで可能な行動をＡを検索
    確率εでランダムに行動
      (i)ε
        a∈Ａをランダムに選択する
        行動aをとり次状態s'を観測する
      (ii)1-ε
        すべてのＡの次状態S'を観測する
        S'を入力としてニューラルネットワークによる近似関数から
          状態価値V'を観測する
        V'から行動a∈Ａを選択する
        行動aをとり次状態s'と次状態の価値$V(s')$を観測する

        ＢＰ法により関数を近似する，ωを更新する
        (入力をs、教師信号を$V(s')$とする)

    s←s'
    sが終端状態なら
      ＢＰ法により関数を近似する，ωを更新する
      (入力をs(終端状態)、教師信号を０とする)
      終了

図５．３　ＴＤとＢＰ法を用いた学習アルゴリズム

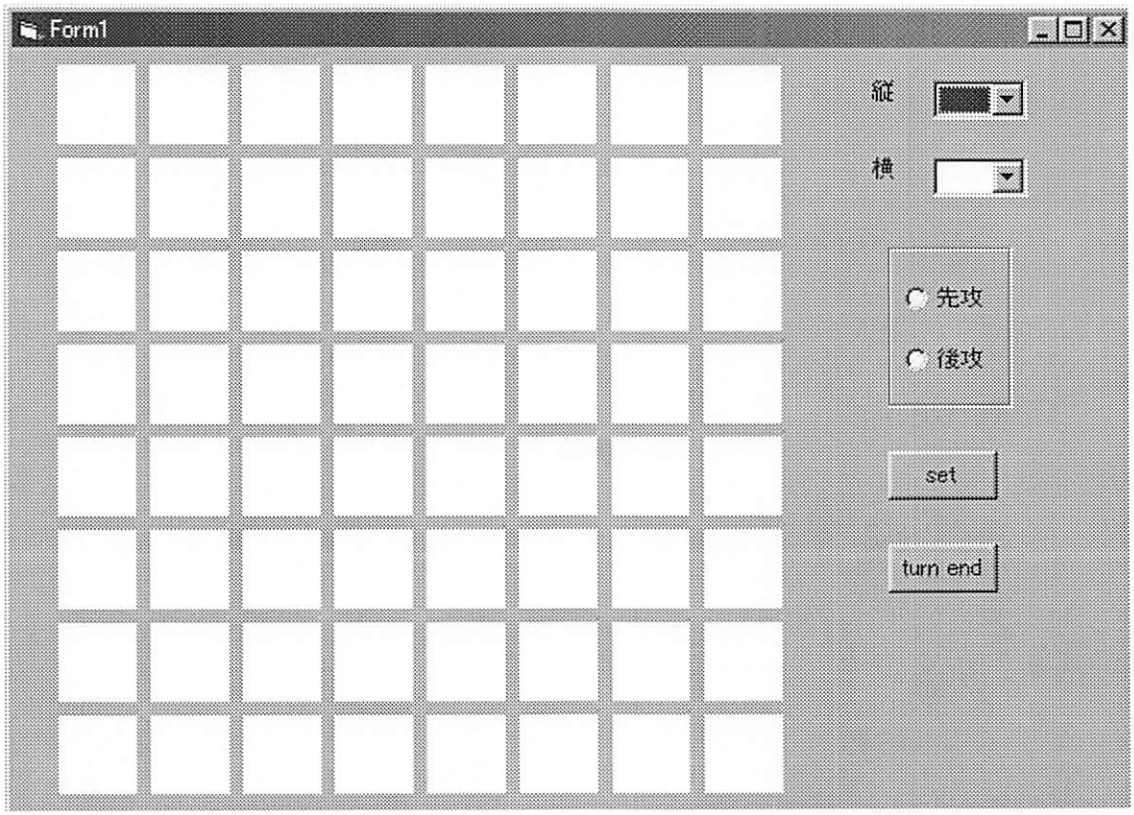

図５．４　ＶＢにより作成した拡張型七五三ゲームのインターフェイス

縦と横のマスの長さを自由に設定できる，また先攻後攻を選択できる

第６章

考察

実験の結果，最適方策を得るには至らなかった．つまり近似関数を使い最適状態価値関数を得ることが出来なかった．

いくつかの状態においては状態価値が最適な値を指していたが，その他の多くの状態においてはうまく状態価値をいることが出来ていなかった．このためある状態においては最適な行動を選ぶが，その他の状態では全く無意味な行動をしてしまうと言うことがおこった．

しかし，いくつかの状態では最適状態価値を得ることが出来たことから手法としての間違いではないと思われる．

原因として考えられるのは以下の２点である．

（１）学習不足

（２）結合強度ωの初期値によるＢＰの誤作動

（１）

もともとＴＤ学習も，ＢＰ法も幾度もの改善を繰り返し解を得る手法であり．それを併用することによって，単なる２倍以上の時間がかかることになった．

このアルゴリズムを用いた学習だと終端状態に対する近似関数を得る必要がある．そこからＴＤによって伝播する各ステップ毎に禁じ関数を得なくてはならない．このようなことから膨大な時間がかかることになる．

さらにＴＤでは全ての状態に無限回訪問するのが理想だが，大きな問題になるにつれそれぞれの状態への訪問に偏りなどがでるため．一つのルートを何度も学習することになる．それで，いくつかの状態では学習が正確に行われたものと思われる．

（2）

第３章でも少し触れた通り，ＢＰ法にはωの初期値によっては局所解に陥ってしまう欠点がある．この欠点を補うためにはランダムでωを与え，これまで学習してきた物との性能の比較を行うなどの対策があった．しかし，これには正確な教師信号が必要となり，教師信号にもそれまで学習から得られた推定値を用いる今回のアルゴリズムには適用できなかった．

現在採るこのできる対策としてはωの初期値をかえて何度も試行することしかない．しかし，この方法だと，一度でさえ時間がかかるので，さらに時間を要するようになり現実的で有効な手法とは言えなくなってしまう．

第7章

結論

本研究で提案したアルゴリズム，作成したプログラムでは膨大な時間がかかりうまくいかなかった．収束に時間のかかる手法を用いたことが原因であり，うまく汎化は行うことが出来なかった．

今後はプログラムの高速化を計り短時間で方策を得ることができる必要になる．また局所解に陥った時の脱出法を考える必要がある．

もしくはニューラルネットワーク以外の関数近似、もしく別のアプローチによる汎化手法を検討する必要がある．

謝辞

本研究の遂行及び論文の作成に多大な御助言及び御指導を賜った新納 浩幸 教官（茨城大学システム工学科）に深い感謝の意を表します．

また，御指導を頂きましたシステム工学科計算機応用学講座の教官の方々にも深く感謝いたします．

最後に，本研究を進めるにあたり助言，協力を頂きました，同研究室の阿部修也氏（茨城大学大学院理工学研究科システム工学専攻），高橋篤史氏（茨城大学大学院理工学研究科システム工学専攻），進藤邦夫氏（茨城大学システム工学科 4 回生），杉田尚史氏（茨城大学システム工学科４回生）に深く感謝いたします．

参考文献

[1] Ricard S. Stton and Andrew G. Barto (訳 三上貞芳, 皆川雅章):
'強化学習', 森北出版 (2000)

[2] 馬場規夫, 小島史男, 小澤誠一:'ニューラルネットの基礎と応用',
共立出版 (1994)

[3] ジョゼフ・オニール:'独習 Java', トップスタジオ訳, 武藤武志監修,
翔泳社 (1999)

[4] 玉井浩:'VisualBasic プログラミング', サイエンス社 (2000)

付録A　プログラムスリスト
(七五三ゲーム)

動的計画による七五三ゲームの最適方策学習プログラム

```
import t13;
import t14;
//状態と遷移のリスト(詳細省略)

class Keisan{
   int i,j;
   public int s[][]=new int[252][8];
   int k[][]=new int[1500][8];
   double r[][]=new double[252][51];
   double val[]=new double[252];
   double va[][]=new double[252][51];
   double vs[]=new double[252];

Keisan(){

for(int m1=0;m1<=4;m1++){
 for(int m5=0;m5<=10;m5++){
  for(int m7=0;m7<=21;m7++){
      i++;

      if(m1==0){k[i][3]++;}
      if(m1==1){k[i][2]++;}
      if(m1==2){k[i][1]++;}
      if(m1==3){k[i][1]+=2;}
      if(m5==0){k[i][5]++;}
      if(m5==1){k[i][4]++;}
```

```
if(m5==2){k[i][3]++;}
if(m5==3){k[i][2]++;}
if(m5==4){k[i][1]++;}
if(m5==5){k[i][3]++;
          k[i][1]++;}
if(m5==6){k[i][2]++;
          k[i][1]++;}
if(m5==7){k[i][1]+=2;}
if(m5==8){k[i][2]+=2;}
if(m5==9){k[i][1]+=3;}
if(m7==0){k[i][7]++;}
if(m7==1){k[i][6]++;}
if(m7==2){k[i][5]++;}
if(m7==3){k[i][4]++;}
if(m7==4){k[i][3]++;}
if(m7==5){k[i][2]++;}
if(m7==6){k[i][1]++;}
if(m7==7){k[i][5]++;
          k[i][1]++;}
if(m7==8){k[i][4]++;
          k[i][1]++;}
if(m7==9){k[i][3]++;
          k[i][1]++;}
if(m7==10){k[i][2]++;
           k[i][1]++;}
if(m7==11){k[i][1]+=2;}
if(m7==12){k[i][4]++;
           k[i][2]++;}
if(m7==13){k[i][3]++;
           k[i][2]++;}
if(m7==14){k[i][2]+=2;}
if(m7==15){k[i][3]+=2;}
if(m7==16){k[i][1]+=4;}
if(m7==17){k[i][1]+=2;
           k[i][3]++;}
if(m7==18){k[i][1]+=2;
```

```
                k[i][2]++;}
      if(m7==19){k[i][1]+=3;}
      if(m7==20){k[i][1]++;
                k[i][2]+=2;}
}}}

int n=1;
   for(int i7=1;i7>=0;i7--){
    for(int i6=1;i6>=0;i6--){
     for(int i5=2;i5>=0;i5--){
      for(int i4=2;i4>=0;i4--){
       for(int i3=4;i3>=0;i3--){
        for(int i2=5;i2>=0;i2--){
         for(int i1=9;i1>=0;i1--){

      for(int j=1;j<=1210;j++){
       if(k[j][1]==i1){
        if(k[j][2]==i2){
         if(k[j][3]==i3){
          if(k[j][4]==i4){
           if(k[j][5]==i5){
            if(k[j][6]==i6){
             if(k[j][7]==i7){

System.out.println(n+";"+"["+i7+"]["+i6+"]["+i5+"]["+i4+"]["+i3+"]["+i2+"]["+i1+
"]");
              s[n][1]=i1;
              s[n][2]=i2;
              s[n][3]=i3;
              s[n][4]=i4;
              s[n][5]=i5;
              s[n][6]=i6;
              s[n][7]=i7;
              n++;
              break;
      }}}}}}}}
```

```
        }}}}}}}

    val[251]=1;
    vs[251]=1;
    t13.def();
    t14.defp();

    //System.out.println("ロード完 "+t13.a[1][1]);
    }

double ab=0.0;
double delta=0.0;
double v=0.0;
void value(){
        for(i=1;i<=250;i++){
           vs[i]=0;
        }
        do{
           delta=0;

           for(i=1;i<=250;i++){
             v = val[i];
             val[i] = atai1(i);
             ab = v-val[i];
             if(ab<=0){ab=ab*(-1);}
             if(ab>delta){delta=ab;}
           }
           for(i=1;i<=250;i++){
              vs[i]=val[i];
           }
      }while(delta>=0.001);
}

double atai1(int n){

       double ganma=1.0;
```

```
        double valu=0.0;
        for(int y=1;y<=50;y++){
        va[n][y]=r[n][y]-ganma*vs[t13.a[n][y]];
        valu=valu+t14.p[n][y]*(r[n][y]-ganma*vs[t13.a[n][y]]);
        }

        //System.out.println("a "+n);
        return(valu);
}

void kaizen(){

        for(i=1;i<=250;i++){
        double c=0;
        double vm=-1;
        for(j=1;j<=50;j++){
           if(t14.p[i][j]>0){
            if(vm<va[i][j]){vm=va[i][j];}
        }}

        for(j=1;j<=50;j++){
           if(t14.p[i][j]>0){
            if(va[i][j]==vm){c++;}
        }}
        for(j=1;j<=50;j++){
           if(t14.p[i][j]>0){
            if(vm==1){
             if(va[i][j]==vm){t14.p[i][j]=1/c;}
             else{t14.p[i][j]=0;}}

if(va[i][j]==vm){//System.out.println(i+"("+s[i][1]+s[i][2]+s[i][3]+s[i][4]+s[i]
[5]+s[i][6]+s[i][7]+")"+";"+j+"("+t13.a[i][j]+") "+t14.p[i][j]+" "+vm);
                        System.out.println("s("+i+")="+j);
                        }
                }
        }}
```

```
}
}

class nana{
	public static void main(String arg[]){

	 Keisan obj =new Keisan();
	 double pai[][]=new double[252][51];
	 int time=0;
	 obj.value();
	 obj.kaizen();
	 for(int i=1;i<=250;i++){
	  for(int j=1;j<=50;j++){
	   if(pai[i][j]==t14.p[i][j]){
	   }else{time++;
		System.out.println(time);
		i=0;
		j=0;
		obj.value();
		System.out.println(time);
		i=0;
		j=0;
		for(i=1;i<=250;i++){
		 for(j=1;j<=50;j++){
		pai[i][j]=t14.p[i][j];
		}}

		i=0;
		j=0;
		obj.kaizen();
		System.out.println(time);
		i=0;
		j=0;
		}
	 }}
}}
```

付録Ｂ　プログラムリスト
(拡張型七五三ゲーム)

ＴＤとＢＰによる拡張型七五三ゲームの最適方策の学習プログラム

```
import java.io.*;
import java.lang.*;
import java.util.Random;


class game{
   /*-----------------------------------------------
           ゲームに関する設定
   -----------------------------------------------*/
   static int tate=8;
   static int yoko=8;
   static int action;

   static int state[][]=new int[yoko][tate];
   static int next_s[][]=new int[yoko][tate];
   static int koudou[][]=new int[10000][4];

   /*-----------------------------------------------
           ニューラルネットワークに関する
   -----------------------------------------------*/
   //入力層
   static int n1=tate*yoko;
   //中間層
   static int n2=30;

   //結合強度
   static double w1[][]=new double[n1][n2];
   static double w2[]=new double[n2];
```

```
/*-------------------------------------------
        ゲームに関する設定
-------------------------------------------*/
//εグリーディ方策のε
static int ipusiron=1;
static int c[]=new int[n1+1];


//              メ              ソ              ッ              ド
-------------------------------------------------------------------------------
 /*-------------------------------------------
        重みを出力する(java)
-------------------------------------------*/

 static void output(){
    for(int i=0;i<n1;i++){
       for(int j=0;j<n2;j++){
          System.out.println("w1["+i+"]["+j+"]="+w1[i][j]+";");
       }
    }
    for(int i=0;i<n2;i++){
       System.out.println("w2["+i+"]="+w2[i]+";");
    }
 }

 /*-------------------------------------------
        重みを出力する（ＶＢ）
-------------------------------------------*/

 static void outputvb(){
    for(int i=0;i<n1;i++){
       for(int j=0;j<n2;j++){
          System.out.println("w1("+i+","+j+")="+w1[i][j]);
       }
    }
    for(int i=0;i<n2;i++){
```

```
            System.out.println("w2("+i+")="+w2[i]);
        }
    }
    /*-----------------------------------------------
            エピソードの初期状態にする
    -----------------------------------------------*/

    static void s0(){
        for(int i=0;i<yoko;i++){
            for(int j=0;j<tate;j++){
                state[i][j]=0;
            }
        }
    }

    /*-----------------------------------------------
            エピソードの終端状態にする
    -----------------------------------------------*/

    static void s1(){
        for(int i=0;i<yoko;i++){
            for(int j=0;j<tate;j++){
                state[i][j]=1;
            }
        }
    }

//行動選択に関するメソッド
    /*-----------------------------------------------
            確率εでランダムに行動を選択する
    -----------------------------------------------*/

    static int koudou_1(){
        int i=(int)(Math.random()*100)%10;
        if(i<=ipusiron){
            return(1);
```

```
    }else{
        return(0);
    }
}

/*-----------------------------------------------
        選択できる行動を選ぶ
-----------------------------------------------*/

static void koudou_2(){
    int sita,migi;
    action=0;
    for(int i=0;i<yoko;i++){
        for(int j=0;j<tate;j++){
            if(state[i][j]==0){
                koudou[action][0]=i;
                koudou[action][1]=j;
                koudou[action][2]=0;
                koudou[action][3]=0;
                action++;
                //System.out.println("a"+i+" "+j);
                migi=1;
                point1: while(i+migi<yoko){
                    if(state[i+migi][j]==0){
                        koudou[action][0]=i;
                        koudou[action][1]=j;
                        koudou[action][2]=1;
                        koudou[action][3]=migi;
                        action++;
                        migi+=1;
                        //System.out.println("yoko"+"migi"+migi);
                    }else{
                        //System.out.println("break migi");
                        break point1;
                    }
                }
```

```
            sita=1;
            point2: while(j+sita<tate){
               if(state[i][j+sita]==0){
                  koudou[action][0]=i;
                  koudou[action][1]=j;
                  koudou[action][2]=2;
                  koudou[action][3]=sita;
                  action++;
                  sita+=1;
                  //System.out.println("tate"+"sita"+sita);
               }else{
                  break point2;
               }
            }
         }
      }
   }
   //action--;
   //System.out.println("a="+action);
}

/*-----------------------------------------------
        行動aの選択
-----------------------------------------------*/
static int c_a(){
   double max=0;
   int select=0;
   for(int a=0;a<action;a++){
      for(int i=0;i<yoko;i++){
         for(int j=0;j<tate;j++){
            next_s[i][j]=state[i][j];
         }
      }
      koudou_4(a);
      seiretu_1();
      //System.out.println(a+":"+nyural());
```

```
      if(max<nyural()){
          max=nyural();
          select=a;
      }
   }
   return(select);
}

/*-----------------------------------------------
        次の状態に移る
-----------------------------------------------*/

static void koudou_3(int a){
   if(koudou[a][2]==0){
      state[koudou[a][0]][koudou[a][1]]=1;
   }else{
      if(koudou[a][2]==1){
         for(int i=0;i<=koudou[a][3];i++){
            state[koudou[a][0]+i][koudou[a][1]]=1;
         }
      }else{
         for(int j=0;j<=koudou[a][3];j++){
            state[koudou[a][0]][koudou[a][1]+j]=1;
         }
      }
   }
}

/*-----------------------------------------------
        行動後の状態を観測する
-----------------------------------------------*/

static void koudou_4(int a){//----------------------
   if(koudou[a][2]==0){
      next_s[koudou[a][0]][koudou[a][1]]=1;
   }else{
```

```
        if(koudou[a][2]==1){
            for(int i=0;i<=koudou[a][3];i++){
                next_s[koudou[a][0]+i][koudou[a][1]]=1;
            }
        }else{
            for(int j=0;j<=koudou[a][3];j++){
                next_s[koudou[a][0]][koudou[a][1]+j]=1;
            }
        }
    }
}

//ニューラルネットワークに関するメソッド
    /*-----------------------------------------------
            状態をＮＮで使うために並べ替える

                [8][8]→[64]
    -----------------------------------------------*/

    static void seiretu(){
        int num=1;
        for(int j=0;j<tate;j++){
            for(int i=0;i<yoko;i++){
                c[num]=state[i][j];
                num++;
            }
        }
    }

    static void seiretu_1(){
        int num=1;
        for(int j=0;j<tate;j++){
            for(int i=0;i<yoko;i++){
                c[num]=next_s[i][j];
                num++;
            }
```

```
    }
  }

/*-----------------------------------------------
        ニューラルネットワークによる出力
-----------------------------------------------*/

static double nyural(){
        int i,j;
        double x,y;
        double x1[]=new double[n2];
        //double y1[]=new double[n3];
        //double z1[]=new double[n4];
        for(i=0;i<n2;i++){
           x = 0;
               for(j=0;j<n1;j++){
                       x += c[j+1] * w1[j][i];
               }
               x1[i] = 1.0 / (1.0 + Math.exp(-1.0 * x));
    }
        y = 0;
        for(i=0;i<n2;i++){
               y += x1[i] * w2[i];
        }
        y = 1.0 / (1.0 + Math.exp(-1.0 * y));
        return(y);
}

/*-----------------------------------------------
        wの初期化値ランダムをランダムに与える
-----------------------------------------------*/

static void syokika(){//------------------------
   for(int i=0;i<n1;i++){
      for(int j=0;j<n2;j++){
         int a=(int)(Math.random()*10);
```

```
            int zoku=(int)(Math.random()*10)%2;
            double b=Math.random();
            if(zoku==0){
            w1[i][j]=b+a;
            }else{
            w1[i][j]=-(b+a);
            }
         }
      }
      for(int i=0;i<n2;i++){
         int a=(int)(Math.random()*10);
         int zoku=(int)(Math.random()*10)%2;
         double b=Math.random();
         if(zoku==0){
         w2[i]=b+a;
         }else{
         w2[i]=-(b+a);
         }
      }
   }

   /*-----------------------------------------------
         -終端状態かの確認
   -----------------------------------------------*/

   static int end(){
      for(int j=0;j<tate;j++){
         for(int i=0;i<yoko;i++){
            if(state[i][j] != 1)return(0);
         }
      }
      //System.out.println("end");
      return(1);
   }

   /*-----------------------------------------------
```

```
        現在の状態を出力
-----------------------------------------------*/

static void s1(){
   for(int j=0;j<tate;j++){
      for(int i=0;i<yoko;i++){
         System.out.print(state[i][j]+" ");
      }System.out.println();
   }
}

/*-----------------------------------------------
        メイン
-----------------------------------------------*/

public static void main(String arg[]){

   int act=0;
   int i,j,k,l;

   //δの値
       double delta1;
       double delta2[]=new double[n2];

   //出力
       double ans;
   //誤差
   double err;
   double gosa=0.0;

   //それぞれのニューロンの値
   double x,y;
       double x1[]=new double[n2];

   //ステップ幅
       double a = 0.13;
```

```
    syokika();


for(l=0;l<1000000;l++){//----------------------------------エピソードの開始
    s0();
    do{

    seiretu();
    koudou_2();//-----------------------------------------行動の検索

    if(koudou_1()==1){//-------------------------方策に従わないランダムな行動
        act=(int)(Math.random()*100)%action;
        koudou_3(act);
    }else{//-----------------------------------------------方策に従った行動
        act=c_a();
        for(i=0;i<yoko;i++){
            for(j=0;j<tate;j++){
                next_s[i][j]=state[i][j];
            }
        }
        koudou_4(act);
        seiretu_1();
        gosa=(1.0-nyural());//-----------------------------教師信号の決定
        seiretu();//-------------------------------------現在の入力のセット

//-------------------------------------------------------------学習開始

        for(i=0;i<n2;i++){
                x = 0;
                    for(j=0;j<n1;j++){
                            x += c[j+1] * w1[j][i];
                    }
                    x1[i] = 1.0 / (1.0 + Math.exp(-1.0 * x));
        }
            y = 0;
```

```
            for(i=0;i<n2;i++){
                    y += x1[i] * w2[i];
            }

        ans = 1.0 / (1.0 + Math.exp(-1.0 * y));
        err = gosa - ans;

        delta1 = ans * (1 - ans) * err;
                for(i=0;i<n2;i++){
                    w2[i] = w2[i] + a * delta1 * x1[i];
                }
        for(i=0;i<n2;i++){
                    delta2[i] = x1[i] * (1 - x1[i]) * delta1 * w2[i];
                    for(j=0;j<n1;j++){
                       w1[j][i] = w1[j][i] +  a * delta2[i] * c[j+1];
                    }
                }
        koudou_3(act);//---------------------------------１スッテプの終了
    }
    }while(end()<1);//-------------------------------１エピソード終了

    s1();//-------------------------------------------終端状態を教師信号０
で学習
    seiretu();
        for(i=0;i<n2;i++){
                x = 0;
                    for(j=0;j<n1;j++){
                            x += c[j+1] * w1[j][i];
                    }
                    x1[i] = 1.0 / (1.0 + Math.exp(-1.0 * x));
        }
            y = 0;
            for(i=0;i<n2;i++){
                    y += x1[i] * w2[i];
            }
```

```
            ans = 1.0 / (1.0 + Math.exp(-1.0 * y));
            err = 0.0 - ans;

            delta1 = ans * (1 - ans) * err;
                    for(i=0;i<n2;i++){
                       w2[i] = w2[i] + a * delta1 * x1[i];
                    }
            for(i=0;i<n2;i++){
                       delta2[i] = x1[i] * (1 - x1[i]) * delta1 * w2[i];
                       for(j=0;j<n1;j++){
                          w1[j][i] = w1[j][i] +  a * delta2[i] * c[j+1];
                       }
                    }
            if(end()==1){
               if(l%1000==0){
                  System.out.println("learn"+l);
                  if(l%100000==0){
                     outputvb();
                     output();
                  }
               }
            }//-----------------------------------------エピソード終了

         }

      }
   }
```