社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE

# ニューラルネットワークを用いたコマンド予測シェルの試作

北川 俊広† 杉山 安洋‡

†日本大学大学院工学研究科 〒963-8642 福島県郡山市田村町徳定字中河原 1
‡日本大学工学部 〒963-8642 福島県郡山市田村町徳定字中河原 1
E-mail: †kitagawa@ssl.ce.nihon-u.ac.jp, ‡sugiyama@ce.nihon-u.ac.jp

**あらまし** ソフトウェア開発において，個々の開発者のソフトウェアプロセスを共有できれば，開発作業の効率化が可能である．今回はシェルに着目し，コマンド列をソフトウェアプロセスとしてとらえた．本研究ではニューラルネットワークを用いて次のコマンドを予測するシェルを試作した．ニューラルネットワークは開発者が打ち込むコマンドの順序を学習し，それを基に次のコマンドを予測する．本稿では，そのシェルの構想，現在の実装，そして動作検証について述べる．

**キーワード** シェル，ソフトウェアプロセス，ニューラルネットワーク

# Building a Command Shell that Can Predict Next Commands Using a Neural Network

Toshihiro KITAGAWA† Yasuhiro SUGIYAMA‡

†Graduate School of Engineering, Nihon University, Koriyama, Fukushima, 963-8642 Japan
‡Department of Computer Science, College of Engineering, Nihon University, Koriyama, Fukushima, 963-8642 Japan
E-mail: †kitagawa@ssl.ce.nihon-u.ac.jp, ‡sugiyama@ce.nihon-u.ac.jp

**Abstract** If developers can share each developer's software process, software development can be more efficient. Authors focused on the command shells, and regarded a sequence of commands as a software process. We developed a prototype of a command shell that can predict next commands using a neural network. The neural network learns developer's command sequences and predicts next commands. This paper will give a brief overview of our shell, the current implementation and the evaluation of the prototype system.

**Keyword** Shell, Software Process, Neural Network

## 1. はじめに

ソフトウェア開発は通常，チームを組んで共通のソフトウェアプロセスに沿って行われる．そして，個々の開発者はそのプロセスに従いながらも，細かい部分では，今までの経験で得た独自のプロセスを用いて開発作業を行っていく．そのときもし，個々の開発者独自のプロセスを自動的にモデル化し，そのプロセスをチームで共有できれば，ソフトウェア開発効率の向上が可能である．また，モデル化することで改善すべき点を発見することも容易になり，失敗を共有することによって，結果的にソフトウェアの品質向上にもつながる．

しかし一般的なソフトウェアプロセスは扱う対象が広く，研究対象として最初から直接取り扱うことが困難である．そこで，今回はシェルに着目し，コマンド列をソフトウェアプロセスとしてとらえた．また，そのコマンド列のパターンを蓄積する機構としてニューラルネットワークを用いた．

本研究で開発しているシェルは，それまでユーザが実行してきたコマンドの順序から知的に次のコマンドを予測する機能を持つものである．そこでこのシェルをインテリジェントシェル（以下 ish と呼ぶ）と名づけた．

本稿の構成は次の通りである．まず，第 2 章ではニューラルネットワークの概要について説明する．そして，第 3 章で，ish の概要について説明し，第 4 章で，ish の実装を説明する．続いて，第 5 章でその動作を検証し，最後に第 6 章で本研究のまとめと今後の課題を述べる．

なお，本論を述べる前に断っておきたい点は，本稿は完成した研究の報告ではないということである．ish は現在，初期のプロトタイプが完成した段階である．したがって，数多くの問題が存在する．本稿では，現在までの研究の途中経過と問題点を述べ，今後の研究

の指針とすることを目的とする．

## 2. ニューラルネットワークの概要

ニューラルネットワークは，人間の脳の働きをコンピュータでまねることを目的に生まれた情報処理方法で，これまでの情報処理方法にはない多くの長所がある．次にそれを述べる．

### 2.1. 学習機能

従来の情報処理方法は，数値演算や論理演算などアルゴリズムで表現された問題に対して高速で正確な処理を行うことができる．したがって，アルゴリズムが明確にできない問題を解くことはできない．一方，ニューラルネットワークは数値演算や論理演算を高速で正確に実行するのには向いていない．しかし，アルゴリズムが不明で，従来の情報処理では解決できない問題を解くことができる．それは，入力と出力の関係から処理方法を自ら導き出す機能を持っているからである．この機能を学習機能と呼ぶ．たとえば，熟練したソフトウェア開発者は，問題に対して正しい判断ができるが，その知識やルールはうまく取り出せないことが多い．このような場合，開発者のプロセスを多数集めて，ニューラルネットワークに学習させれば，そのニューラルネットワークが的確な判断をできるようになる可能性がある．

### 2.2. 汎化能力

ニューラルネットワークは，学習した入力以外の未学習の入力に対しても，適切な出力を与える能力を持つ．この能力を汎化能力という．この能力をプロセスに応用すると，学習したプロセスだけではなく，それに類似したプロセスについても適切な予測を行うことができる．

### 2.3. パターン認識機能

パターン認識機能は，ニューラルネットワークの大きな特徴の一つである．パターン認識とは文字，図形，信号，音声などの各種パターンから特徴を抽出し判断する機能である．この機能をプロセスに応用すると，コマンド列中に本質的ではないコマンドがいくつか含まれた場合でも特徴を見出し，それをパターンとして判断することができる．

## 3. ish の概要

ish は主にコマンド実行機能，学習機能，ヒストリ機能，予測機能から構成される．コマンド実行機能とヒストリ機能は，他の一般的なシェルにも存在するが，学習機能と予測機能は ish 独自のものである．図１に ish の概要を示す．

ish のユーザがこれらの機能を利用するには，Enter キーと上下のカーソルキーを使用する．Enter キーは，入力したコマンドを実行するときに使用する．このキーが押されると，ish はコマンド実行機能を用いて，指定されたコマンドを独立したプロセスで実行する．そしてその後，学習機能によりユーザが実行したコマンド順序の学習が行われる．

次に，上向きのカーソルキーは，実行したコマンドのヒストリを得るときに使用する．このキーが押されるごとに，ish はヒストリ機能を用いて，実行したコマンドをさかのぼって返す．

最後に，下向きのカーソルキーは，次のコマンドを予測して得るときに使用する．このキーが押されると，ish は予測機能を用いて，次のコマンドを予測して返す．

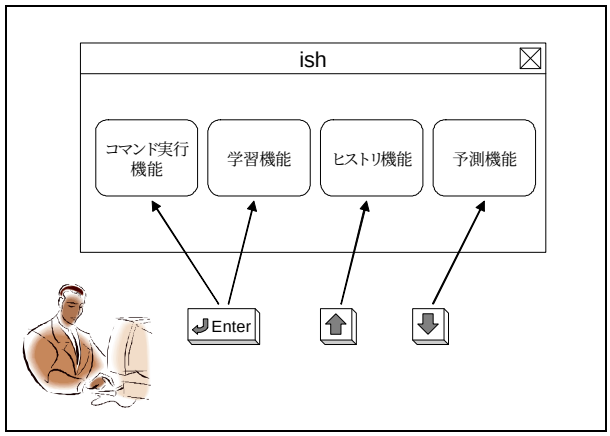


**図１　ish の概要**

ish によるコマンド順序の学習は，ユーザが意識することなく行われる．そのため，ソフトウェア開発者がこのシェルを一般的なシェルと同様に使用してソフトウェア開発作業を行うことで，そのプロセスが ish に蓄積される．

### 3.1. 学習機能

ユーザのプロセスを学習するための学習機能は，ニューラルネットワークによって提供される．そして，そのプロセスの学習はコマンド実行後に行われる．図２に学習機能の概要を示す．

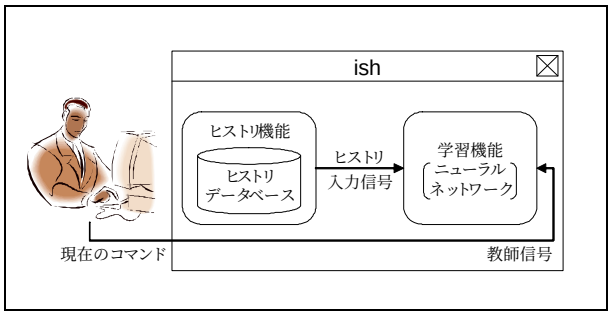


**図２　学習機能の概要**

ユーザがコマンドを実行すると，そのコマンドの情報がニューラルネットワークに渡される．ニューラルネットワークはその情報を教師信号とし，また，ヒス

トリを入力信号として学習を行う．そうすることでヒストリを入力すると，次のコマンドを出力するニューラルネットワークを構築する．ただし，ヒストリはすべてを用いるのではなく，範囲を限定した直前のものを用いる．

### 3.2. 予測機能

次のコマンドを予測するための予測機能は，学習機能と同じニューラルネットワークによって提供される．図３に予測機能の概要を示す．

ユーザが下向きのカーソルキーを押すと，範囲を限定した直前のヒストリが入力信号としてニューラルネットワークに入力される．そして，次のコマンドを出力信号として得る．

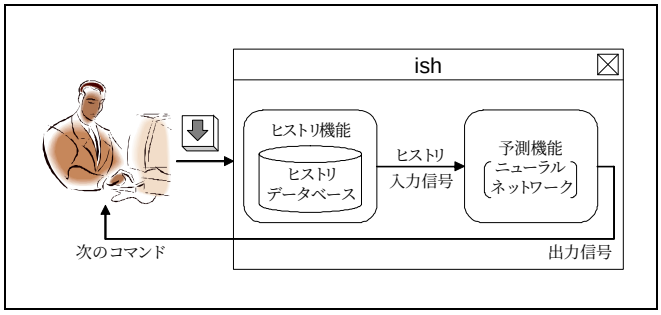


**図３　予測機能の概要**

### 3.3. プロセスのパターン化

ユーザのコマンド手順をそのまま学習すると，学習したコマンド手順からしか次のコマンドを予測できない．そのため，未学習のコマンド手順から予測を行うには，ユーザの詳細な手順をパターン化する必要がある．図５にパターン化の例を示す．

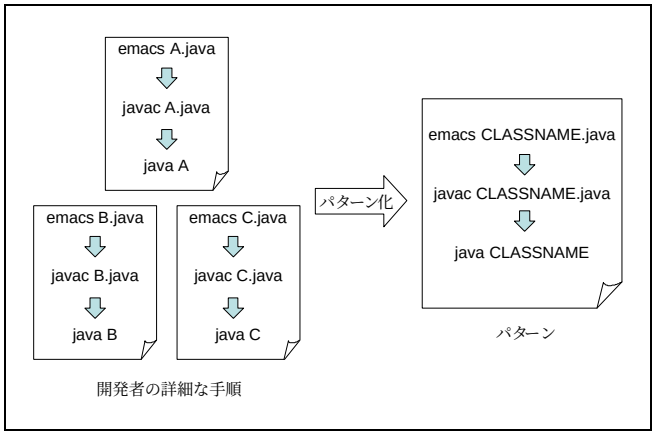


**図５　パターン化の例**

図の左側のように，開発者の詳細な手順をそのまま記憶すると，もう一度開発者が emacs A.java を実行したときには次のコマンドを正しく予測できるが，新たに emacs D.java を実行したときはまったく予測ができない．そこで，パターン化を行う必要がある．図の右側のようにクラス名という汎用的な言葉に置き換えて記憶するのである．こうすることで，新たに emacs D.java が実行されても D がクラス名であると認識できれば，次のコマンド javac D.java を予測可能になる．

### 3.4. プロセスの共有

ユーザ間のプロセスの共有は，ニューラルネットワークをサーバ上に置き，それをクライアントである ish が利用することによって実現される．図４にプロセスの共有を示す．

この図は，それぞれのクライアントがサーバ上のニューラルネットワークを使用し，そのとき，クライアントＡの学習機能を有効にし，クライアントＢの学習機能を無効にすることを表している．

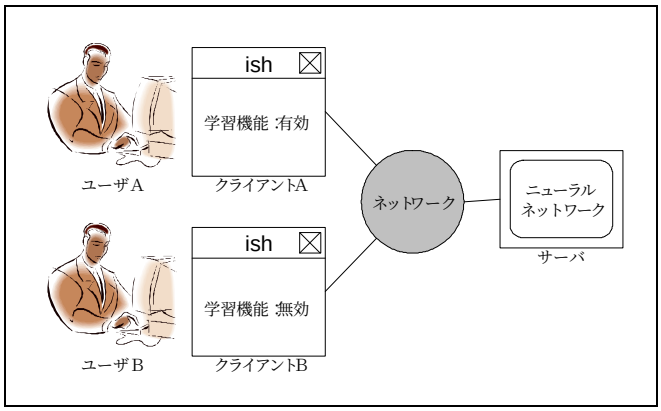


**図４　プロセスの共有**

こうすることでニューラルネットワークにはユーザＡのプロセスのみが蓄積される．ここでユーザＢが ish の予測機能を用いると，ユーザＡの今までのプロセスから予測された次のコマンドを得る．これより結果的にユーザＢはユーザＡのプロセスを使用できる．

## 4. ish の実装

### 4.1. ニューラルネットワークの実装

ニューラルネットワークは様々な種類が存在するが，本研究では現在最も一般的である学習法にバックプロパゲーション[1][2]を用いた階層型ニューラルネットワークを開発し使用した．

#### 4.1.1. 階層型ニューラルネットワーク

ニューラルネットワークは，ニューロンが他の多数のニューロンと結合して構成される．ここで言うニューロンは神経細胞の処理をモデル化したもので，そのモデルは，入力値に重みをかけて総和をとり，シグモイド関数を通して出力するものである．

階層型ニューラルネットワークは，明確に区別された入力層と出力層を持つニューラルネットワークである．現在は，それらの層の間に隠れ層と呼ばれる一層以上の中間層を加えたものがよく用いられる．図７に階層型ニューラルネットワークの構造を示す．図中の円はニューロンを表している．

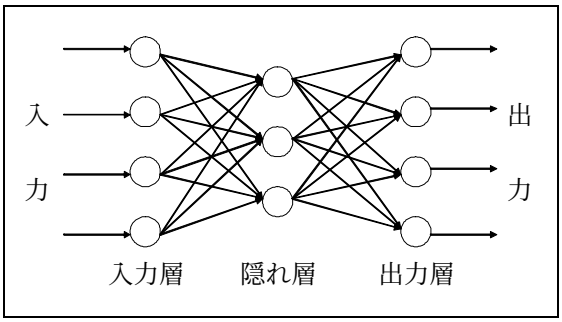


**図７　階層型ニューラルネットワーク**

階層型ニューラルネットワークは，入力層に与えた信号が結合の重みによって変換されながら，出力層のニューロンから出力されるという前向きの信号伝播を行う．ただし，入力層のニューロンは前節で説明したものとは異なり，与えられた入力信号をそのまま出力する．つまり，シグモイド関数や閾値は持たない．

### 4.1.2. バックプロパゲーション

ニューラルネットワークは重みを調節することで，望ましい出力を出すようになる．その重みを調節する方法のひとつがバックプロパゲーションである．バックプロパゲーションの概要を図８に示す．このアルゴリズムは，まず，ニューラルネットワークに入力信号を入力し，出力信号を得る．そのとき，出力層に教師信号（望まれる出力）を与え，教師信号と出力信号の誤差を計算する．そして，その誤差が０になるように出力層から入力層の方向にニューロン間の重みを調整する．

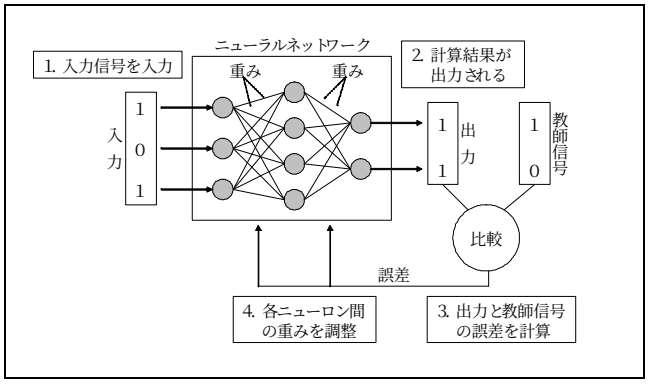


**図８　バックプロパゲーションの概要**

## 4.2. 学習機能

### 4.2.1. 学習機能の実装

ニューラルネットワークには,ish 上で実行されたコマンドの順序を学習させる．そのとき，コマンドによって生成されたプロセスの終了コード（０が正常終了を示す)とセットにして学習作業を行う.その方法は，今，ユーザがあるコマンドを実行したと仮定すると，それ以前に実行したいくつかのコマンドをひとかたまりと考え,それらの終了コードも含めて入力信号とし，今実行したコマンドを教師信号としてニューラルネットワークに学習させる．ただし，コマンドは文字列のためニューラルネットに直接入力することができない．そのため，後述するコマンド・２進数変換機能を用いてコマンドを一意な２進数に変換して学習を行う．図９に直前の２つのコマンドを入力信号とした場合の学習機能の実装を示す．

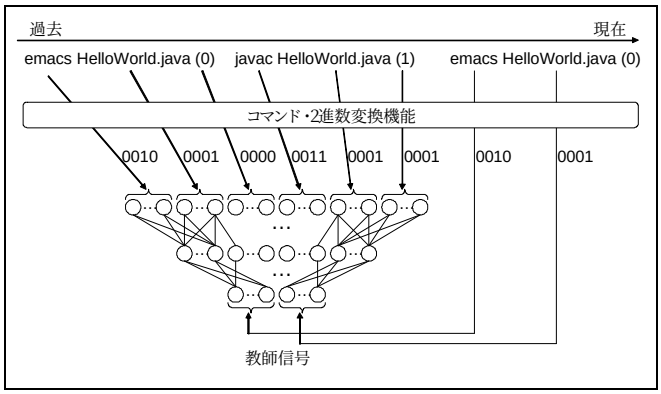


**図９　学習機能の実装**

図中のコマンドラインの括弧で囲まれた数字は終了コードを表している．ここでは，コマンド，引数，終了コードをそれぞれ４ビットの２進数に変換しているが，現在のプロトタイプの ish ではコマンドと引数は８ビットの２進数に変換している．この方法で学習を行うことで，直前のいくつかのコマンドを変換機能で２進数化したものを入力すると，２進数で表現された次のコマンドを出力するニューラルネットワークが構築される．

### 4.2.2. コマンド・２進数変換機能

コマンド・２進数変換機能は，文字列であるコマンドと引数，そして，１０進数である終了コードを２進数に変換し，ニューラルネットワークに直接入力できる形にする機能である．終了コードは４ビットの２進数に変換し，コマンドと引数はそれぞれ渡された順に８ビットの２進数を割り振り変換を行う．そして，その対応関係は随時記録され，以降同じコマンドや引数が渡されたときは，以前割り当てた２進数を返す．また，この機能は記録された対応関係を用い２進数から文字列への変換も行う．２進数からコマンド文字列への変換は，ニューラルネットワークの出力をコマンド文字列に変換するときに使用される．

## 4.3. 予測機能

次に，予測機能について説明する．ユーザが，emacs HelloWorld.java，javac HelloWorld.java という順序でコマンドを実行し,次のコマンドを予測すると仮定する．図１０に予測機能の実装を示す.まずコマンド,引数，終了コードはコマンド・２進数変換機能で２進数に変換され，ニューラルネットワークに入力する．そしてニューラルネットワーク内で計算が行われ，以前学習

した次のコマンドをあらわす２進数の出力を得る．この出力を変換機能を用いてコマンド文字列に変換し出力する．ここでは，emacs HelloWorld.java という結果が得られている．

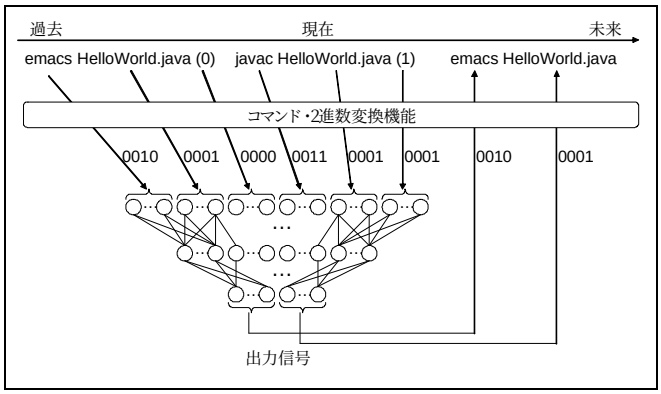


**図１０　予測機能の詳細**

### 4.4. パターン化機能

パターン化機能は本来ならばニューラルネットワークのパターン化機能を用いて実装する予定であったが，現在はプロトタイプということで，パターン化の効果を確認するために次の方法を採用している．

#### 4.4.1. 知識ファイル

知識ファイルは，３．３節で述べたユーザのプロセスをパターン化するために使用されるＸＭＬファイルである．図１１に知識ファイルの例を示す．

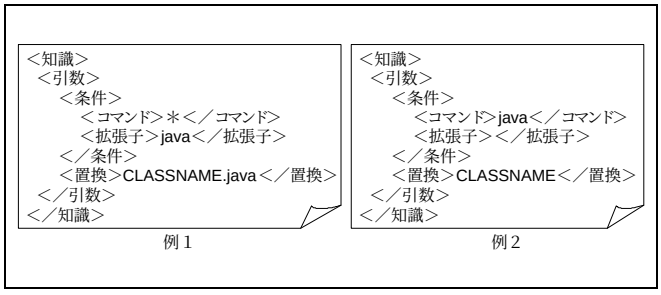


**図１１　知識ファイルの例**

この図の例１は，引数で，その引数が用いられたコマンドに指定はなく，拡張子が java の場合に CLASSNAME.java という引数名に置き換えることを表している．また，例２は，引数で，その引数が java コマンドで使用され，拡張子がない場合に CLASSNAME という引数名に置換することを表している．

#### 4.4.2. 知識ファイルを用いたパターン化

知識ファイルを用いたパターン化を図１２に示す．この図は，開発者の詳細な手順である emacs A.java，javac A.java，java A が，図１１に示した２つの知識ファイルで A の部分が CLASSNAME に置き換えられ，パターン化されたことを表している．ニューラルネットワークの学習には，このパターン化されたものを用いる．

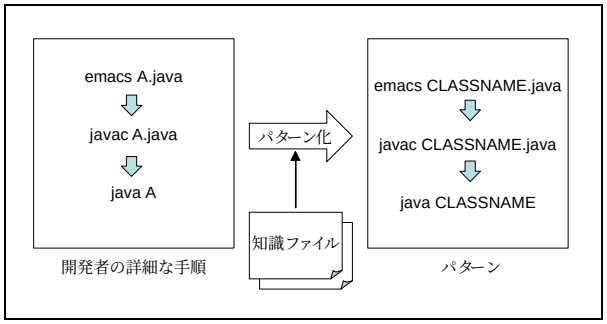


**図１２　知識ファイルを用いたパターン化**

### 4.5. 共有機能

クライアントとなる ish がネットワークを介してサーバのニューラルネットワークを利用する方式の実装には，RMI(Remote Method Invocation)を用いた．RMI は，ある JVM(Java Virtual Machine)から別の JVM にあるオブジェクトのメソッドを呼ぶための仕組みである．

### 4.6. 知識ファイルを使用した具体的な動作

具体的に，ユーザが Java 言語を用いて簡単なソフトウェア開発を行った場合を考える．そこで，ユーザが emacs A.java，javac A.java，java A という順番でコマンドを実行すると仮定する．また，説明を容易にするため，ニューラルネットワークへの入力に用いるヒストリの範囲は直前のひとつのコマンドのみを使用する．

#### 4.6.1. 学習動作

ユーザが ish で emacs A.java とコマンドを入力し Enter キーを押すと，コマンド実行機能により emacs が起動される．そして，ユーザがソースコードの編集を終え，emacs を終了させた時点で，コマンド，引数，終了コードがコマンド・２進数変換機能に渡される．変換機能では，それらの情報と知識ファイルを比較し，条件に一致するものがあれば変更を行う．今回の場合は，引数が図１１に示す知識ファイルと一致するので，emacs A.java は emacs CLASSNAME.java に変更される．そして，その変更されたそれぞれの情報に対して２進数変換を行う．このとき同時に，CLASSNAME=A という情報を記録しておく．そして，それらの変換した２進数をニューラルネットワークに入力し学習作業を行う．続いて，ユーザが javac A.java，java A と実行した場合も同様の処理が行われる．

#### 4.6.2. 予測動作

ここでは上述の学習がすでに行われ，ユーザが今 emacs B.java を実行し終えたと仮定する．このとき，ユーザが下向きのカーソルキーを押すと，直前のコマンド（emacs），引数（B.java），終了コードがコマンド・２進数変換機能に渡される．変換機能では，B.java が図１１の知識ファイルの条件と一致するため，CLASSNAME.java に変更され，CLASSNAME=B と記録される．そして，それが２進数に変換されてニューラ

ルネットワークに入力され，次のコマンドを表す２進数の出力を得る．最後に，その２進数をコマンド・２進数変換機能でコマンド文字列に変換する．このとき変換されるコマンド文字列は，javac CLASSNAME.java である．しかし，変換機能には CLASSNAME=B という情報が記録されているので，最終的には javac B.java を得ることができる．

## 5. ish の動作検証

ish の動作を確認するために２種類のサンプルを用意して動作検証を行った．表１にそれらのサンプルの動作検証結果を示す．ただし，条件として図１１の知識ファイルをあらかじめ用意しておくとともに，ニューラルネットワークへの入力に用いるヒストリの範囲は直前の２つのコマンドとした．

**表１　動作検証結果**

| | サンプル１ | サンプル２ |
|---|---|---|
| コマンド順序 | emacs A.java<br>javac A.java<br>java A<br>emacs B.java<br>javac B.java | emacs A.java<br>emacs B.java<br>javac A.java<br>java A<br>emacs C.java<br>emacs D.java |
| 期待する予測結果 | java B | javac C.java |
| 実際の予測結果 | java B | javac D.java |

サンプル１は，ユーザがクラス名Ａのファイルについて emacs A.java，javac A.java，java A という順序でコマンドを実行し，その後，クラス名Ｂのファイルについて emacs B.java，javac B.java を実行した場合，ish が次のコマンドを予測できるかどうかを試すものである．このサンプルを実際に実行すると，期待したとおり次のコマンドは java B.java であると予測された．このことから，プロセスのパターン化が正常に機能していることが確認できた．

サンプル２は，ユーザがクラス名Ａのファイルとクラス名Ｂのファイルを混ぜてコマンドを実行した場合である．このサンプルにおいてユーザが javac A.java でクラス名Ａのソースファイルのみをコンパイルしているのは，Java コンパイラは仕様として，あるクラスのソースファイルをコンパイルするとき，そのクラスで使用されている他のクラスのソースファイルも一緒にコンパイルする機能を持っているからである．この場合，同様に emacs C.java，emacs D.java の順序でクラス名Ｃのファイルとクラス名Ｄのファイルを混ぜてコマンドを実行した時，ish が次のコマンドをどう予測するかを試す．このサンプルを実際に実行すると，ish は次のコマンドを javac D.java と予測した．期待する予測結果は javac C.java なので，予測は正しくない．これは最後に emacs D.java を実行したことによって，最終的に CLASSNAME=D と記憶したためである．このように，単純にクラス名の部分を CLASSNAME に置き換えるだけでは正しく予測できない場合が存在することが判明した．この問題の解決策を今後の課題として次節で述べる．

## 6. まとめと今後の課題

今回は，ユーザが実行したコマンドと，それに付随する引数，終了コードの情報を２進数に変換しニューラルネットワークに学習させることで，過去の作業の流れから，次のコマンドを予測する ish を試作した．また，知識ファイルを用い，より汎用的な知識に変換してニューラルネットワークに学習させることで，限定的ではあるが未学習のことにも対応ができるようになった．しかし，動作検証の節で指摘したように知識ファイルを使用した場合，正しく予測できないことがある．そこで，今後の課題としてこの問題の解決策について表１のサンプル２を例に次に述べる．

まず実行時に知識ファイルを使用し，使われたクラス名を CLASSNAME=A,B,C,D のように記憶しておき，パターン化を行わずにそのままニューラルネットワークに学習させる．この場合，emacs C.java，emacs D.java から次のコマンドを予測することは当然できない．そこで，記憶しておいたクラス名を利用して，ＣをＡ，ＤをＢに置き換える．すなわち，emacs A.java，emacs B.java から次のコマンドを予測する．これは以前学習したプロセスと一致するので javac A.java が予測結果として得られる．そして，さらにこれをＣをＡに置き換えたという情報を用いて，逆にＡからＣに置き換えれば，最終的に次のコマンド javac C.java を得ることができる．これが現在考えている解決策である．

最後に，今回はニューラルネットワークを推論エンジンとして使用したが，これではニューラルネットワークの特徴を生かしているとは言えない．ニューラルネットワークは，もともと学習パターン以外の未学習の入力に対しても，望ましい出力を与えることができる汎化能力を備えている．もし，この汎化能力をうまく利用できれば，知識ファイルが不要になる可能性がある．今後は，この能力を考慮して研究を進めていきたいと考えている．

## 文　献


[1] 平野広美，Ｃでつくるニューラルネットワーク，パーソナルメディア株式会社，東京，2001.

[2] Joseph P. Bigus, Jennifer Bigus, Java による知的エージェント入門，井田昌之(訳)，ソフトバンク パブッリシング株式会社，東京，2002.