特別研究報告書

リカレント・ニューラル・ネットワークにおける迷路の学習

指導教官　　富田眞治 教授

京都大学工学部情報学科

永野 貴宣

平成15年2月3日

リカレント・ニューラル・ネットワークにおける迷路の学習

永野 貴宣

内容梗概

回帰結合を持つニューラル・ネットワークであるリカレント・ニューラル・ネットワークは時系列情報を処理することができ、人間の高次情報処理機能の実現・解明に役立つものとして期待されている。しかし、リカレント・ニューラル・ネットワークの処理には依然解明されていない点が多く、リカレント・ニューラル・ネットワーク上で情報がどのように表現され、それが処理されるのかについて一般的に述べた研究は少ない。

こうした現状は、ある特定の機能を実現するというような、簡単な課題しかリカレント・ニューラル・ネットワークに扱わせてこなかったことにあると考えられる。このような特定の課題を実現したネットワークは、その課題を解く有限状態機械とほぼ同等の形状となっており、リカレント・ニューラル・ネットワークの特徴を生かしているとは言い難い。一般的な議論をするためには、有限状態機械では解けないような難易度の高い課題を設定する必要があると考えた。

そこで、本実験ではリカレント・ニューラル・ネットワークが従来扱ってきたものよりも難易度の高い迷路課題を、リカレント・ニューラル・ネットワークに学習させた。迷路課題とは、迷路内を動くエージェントに任意の迷路を解かせる問題であり、この問題を有限状態機械で構成しようとすると状態数が膨大になり現実的ではなく、この問題を解く機械はスタックマシンによって実現することができる。

本稿では迷路を解く順序回路を示し、この順序回路が入力を受けとってから正しい出力を出すまでにある程度の遅延が必要であることを示す。その後に、この順序回路との比較から、迷路を解くリカレント・ニューラル・ネットワークも入力を受けとってから正しい出力を出すまでにある程度の遅延が必要であるという考察を示す。

この遅延はどの程度の長さにとれば十分なのか不明なので、学習の際に任意の長さの遅延を アダプティブ に挿入して教師信号を生成する手法を提案した。また、教師を アダプティブ に生成することで、リカレント・ニューラル・ネットワークの学習に、リアルタイム通時的誤差逆伝播法よりも効率の良いアルゴ

リズムであるエポック毎通時的誤差逆伝播法を用いることができるので、これを用いた。

アダプティブ に教師を生成する手法では、リカレント・ニューラル・ネットワークの教師信号に遅延区間という区間を設け、この区間内ではリカレント・ニューラル・ネットワークの出力に任意の値を許した。この遅延区間ではリカレント・ニューラル・ネットワークの出力にかかわらず、エージェントは移動しないことにした。その上で、リカレント・ニューラル・ネットワークの出力に応じて遅延区間の長さを適応的に変化させた。

遅延区間内では教師信号の値はリカレント・ニューラル・ネットワークの出力の値をそのまま用いる。こうすることで、リカレント・ニューラル・ネットワークの出力と教師信号の差が 0 になる。通時的誤差逆伝播法では、リカレント・ニューラル・ネットワークの出力と教師信号の誤差に比例して結合重みの改良が進むが、こうすることで、遅延区間での学習は、学習による結合重みの変化に全く影響が出ないようにした。

また、リカレント・ニューラル・ネットワークの出力に応じて遅延区間の長さを変化させるアルゴリズムとして、遅延区間においてリカレント・ニューラル・ネットワークの出力が教師の進む方向にある一定値まで近づくまで遅延区間を継続し、リカレント・ニューラル・ネットワークの出力と教師の進む方向がある程度一致した時点で遅延区間をその時点までとして打ち切る、というアルゴリズムを採用した。

最後に、迷路を解くリカレント・ニューラル・ネットワークを構成した。その際、学習する迷路の選び方として、ランダムに生成した迷路について一度学習させ、その後にこれとは別のランダムに生成した迷路を解かせ、解けない迷路についてさらに学習するという方針を用いた。これらの手法を用いて迷路を解くリカレント・ニューラル・ネットワークを実現した。

Learning of Reccurent Neural Network that solves Maze Task

Takanobu Nagano

Abstract

A Recurrent Neural Network, in which Feedback Loops are included can handle time series information. Recurrent Neural Network is expected to clarify the functions of human's advanced information processing and realize them. But many aspects of how Recurrent Neural Networks deal with information have not been clarified yet. And few researches have been done to clarify how informations are expressed in Recurrent Neural Networks.

Such the present condition comes from the fact that in old researches Recurrent Neural Networks learned only specific easy tasks. A Recurrent Neural Network that has learned such an easy task forms a structure similar to a Finite State Machine that solves the same task. Such a Recurrent Neural Network cannot be said to make good use of the characteristics of Recurrent Neural Networks. We thought that to have a general argument Recurrent Neural Network should learn difficult tasks that Finite State Machine cannot solve.

Thus, in this paper, we make Recurrent Neural Network learn more difficult task than in old researches. This task is a Maze Task. We made Recurrent Neural Network move around mazes and solve mazes of arbitrary shapes , not a specific shape. If we solve this Maze Taask with Finite State Machine, the machine needs a huge number of State. So, it is not realistic to solve this task with Finite State Machine. A Stack Machine can solve Maze Task.

In this paper, we show an Sequential Circuit which solves Maze Task. And we prove that this Sequential Circuit needs some delay when it returns valid output after it has recieved an input. We compare Recurrent Neural Network with this Sequential Circuit , and show a consideration that Recurrent Neural Network which can solve Maze Task need some delay to output valid value after it has received an input.

It is unknown how much delay a Recurrent Neural Network needs to output valid value after it has received an input. So, we propose a technique to insert delay of arbitrary length adaptively when we generate a Teacher Signal. :We

call this technique “Adaptive Learning”.
And by using Adaptive Learning, we can apply Epochwise BPTT, which is more efficient algolithm than RealTime BPTT.

In Adaptive Learning, we establish Delay Sections , in which Recurrent Neural Network is allowed to output arbitrary values. In Delay Section , Agent does'nt move , regardless of the output of Recurrent Neural Network. And according to the output of the Recurrent Neural Network , we changed the length of the Delay Section in adaptation.

In the Delay Section, the value of Teacher Signal is made equal to the value of the output of a Recurrent Neural Network. By doing this , the differences of the output of a Recurrent Neural Network and the Teacher Signal are set to 0. In BPTT , the Joint Weight is improved in proportion to the margin between the output of Recurrent Neural Network and Teacher Signal. So, in the Delay Section, the Joint Weight is not changed absolutely by Learning.

According to the output of Recurrent Neural Network , we changed the length of the Delay Section in adaptation. We show the algolithm bellow. In the Delay Section, if the output of Recurrent Neural Network is close to the Teacher, we put an end to the Delay Section there. If the output of Recurrent Neural Network is not close to the Teacher , the Delay Section continue.

In Learning, we made Recurrent Neural Network learn mazes which are generated at random. After this, we made it's Network solve other mazes which are generated at random. And we made it's Network learn mazes again which it could'nt solve.

At last, we implemented a Recurrent Neural Network which solves Maze Task.

リカレント・ニューラル・ネットワークにおける迷路の学習

目次

第1章　はじめに

リカレント・ニューラル・ネットワーク(Recurrent Neural Network, 以下**RNN**とする)は、回帰結合を持つ**ニューラル・ネットワーク**(Neural Network,以下**NN**)である。この回帰結合が、順序回路における回帰結合と同様、入力の履歴を代表することができる。[3]RNN による時系列データ処理は、人間の高次情報処理機能の実現・解明に役立つものと期待されている。

しかし、RNN の処理には依然解明されていない点が多く、RNN 上で情報がどのように表現され、それが処理されるのかについて一般的に述べた研究は少ない。個別の課題に対しては、Cleeremans[1] や Jun.T[2] らも触れているが、包括的に議論されてはいないのが現状である。

こうした現状は、従来研究において RNN に扱わせてきた課題が有限状態機械で容易に構成できるような簡単なものにに限られてきたからであると考えられる。これらの研究では学習後のネットワークの形状は、同等の問題を解く有限状態機械とほぼ同等のものとなっており、RNN の性質を生かしきっているとは言い難い。一般的な議論をするためには、従来よりも難易度の高い、有限状態機械では簡単には構成できないような課題を設定する必要があるだろう。

そこで我々は、RNN に迷路課題を学習させることにした。迷路課題とは、迷路内を動く**エージェント**(RNN)の視点によって任意の迷路を解く課題である。迷路課題を解く機械を有限状態機械で構成しようとすると、状態数が膨大な量になり現実的ではない。スタックマシンを用いて構成すると容易に実現できる。つまり迷路課題の難易度は、有限状態機械で構成できる問題より高いと考えられる。迷路課題を学習後の RNN は、任意の迷路を解くためのテクニックを獲得したことになる。

本稿ではまず2章において、RNN について詳しく説明し、RNN に対する学習アルゴリズムについて述べる。また、関連研究の紹介もする。3章では、迷路課題を解く RNN の入出力について述べ、迷路を学習する際の教師を**アダプティブ**に定義する方法について詳しく説明する。4章では、実験の結果と考察を述べる。

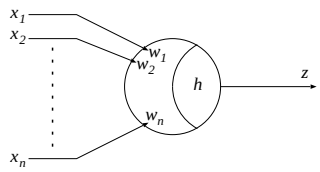

図1: ニューロン・モデル

第2章　研究の背景

この章では RNN について概説する。2.1 節では、まずニューロン・モデルの説明をし、そこから RNN が実際に時系列処理が可能な事を示す。

続いて 2.2 節では関連研究として、Jun.T らによる、RNN に迷路を解かせる研究について述べ、その研究と我々の研究を比較することで、研究の目的をより明確にする。また 2.3 節では、RNN の学習アルゴリズムについても述べる。RNN の学習アルゴリズムは階層型 NN と同様に最急降下法に基づくものが一般に用いられる。そこで、その代表的なものについて述べ、後に迷路を学習する際に用いたアルゴリズムの詳しい説明をする。

2.1　リカレント・ニューラル・ネットワーク

この節ではまず RNN の構成要素であるニューロン・モデルについて説明する。階層型 NN において誤差逆伝播法を用いる場合に、最も単純で一般に用いられるニューロン・モデルが**準線型素子モデル**であるが、RNN で用いられるニューロン・モデルも、この準線型素子モデルである。

次にその準線型素子モデルを用いて構成される RNN について説明する。

2.1.1　ニューロン・モデル

準線型素子モデルを図 1 に示す。

このモデルでは一つのニューロンは n 個のニューロンからそれぞれ、$x_1, x_2, \cdots, x_n$ の入力信号を受けとり、その内部状態 u および出力 z は、

$$u = \sum_{i=1}^{n} w_i x_i \tag{1}$$

$$z = f(u - h) \tag{2}$$

で表される。

ここで h は**閾値**であり、w_i は i 番目のニューロンとの間の結合効率を表す量

で、**結合荷重**または**結合重み**と呼ばれる。

ニューロンの**評価関数**fは、一般に微分可能な単調増加関数および単調非減少関数が用いられる。これはNNの構成法として、誤差逆伝播法のような最急降下法に基づく学習アルゴリズムを用いて学習を行う場合、重みの変量を求める際に微分されるからである。一般には、

$$f(x) = \frac{1}{1 + exp(-x)} \tag{3}$$

のシグモイド関数が用いられ、この関数の値域よりニューロンの入出力は(0,1)の連続値をとる。

数式の取り扱いの簡単化のために、閾値hのかわりに、仮想的に常に1を与える$n+1$番目の入力x_{n+1}を設けて、新たに$n+1$番目の結合重みを$-h$と定義すれば、式1と2で定義される入出力関係は、次のように簡単化される。

$$u - h = \sum_{i=1}^{n} w_i x_i + w_{n+1} = \sum_{i=1}^{n+1} w_i x_i, w_{n+1} = -h \tag{4}$$

$$z = f(\sum_{i=1}^{n+1} w_i x_i) \tag{5}$$

このw_iは**バイアス(bias)**と呼ばれる。以上のようにすることで、閾値の学習も結合重みの場合と同様に行うことができる。以降、この表現式で表されるニューロン・モデルを用いる。

2.1.2 RNN

1章でも述べたように、RNNとはフィードバック・ループを持つNNのことである。RNNの例を図2に示す。四角形が入力ユニット、2重丸が出力ユニット、丸型が入出力のどちらでもない隠れユニットを表している。

RNNが時系列処理を実現できる事は前章で述べたが、ここではその具体的な例を示す。例として、入力系列のパリティを求める装置を考える。入力系列のパリティを求める装置とは、$t = T$における出力が、$t = T - 1$における出力と、$t = T - 1$における入力のXORであるような装置のことである。

順序回路でこれを実現するには図3のように、XOR素子の入力の1つを外部入力ユニットにつなぎ、もうひとつの入力を1つ前の時間の出力を記憶しているフリップ・フロップの出力から取ればよい。

これに対して、4のような結合重みを持つようなRNNは、入力系列のパリ

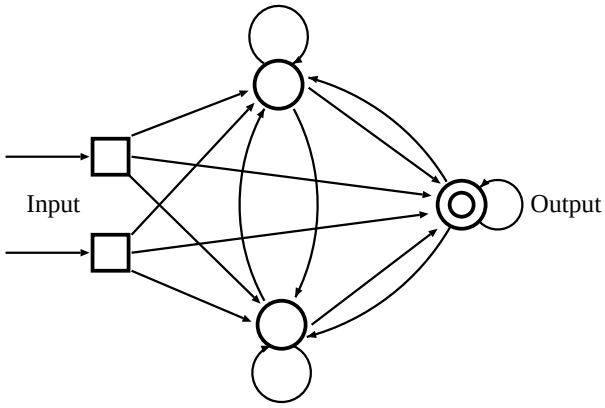

図2: リカレント・ニューラル・ネットワーク

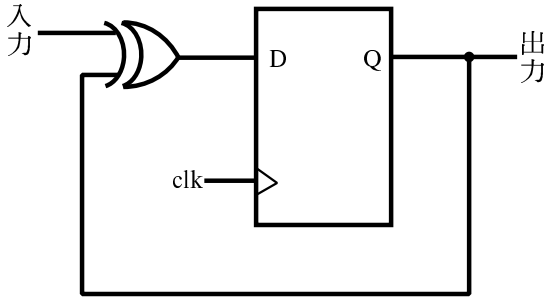

図3: 入力系列のパリティを求める順序回路

ティを求めることができる。

図の#1,#2,#3 はニューロンのインデックスを表している。また図の各矢印は、向きが信号が伝播する方向を、そこの付された数値はその結合重みを表している。

この RNN において、$t = 0$ のとき外部入力が 1 という入力系列を受けとったとする。この時の図3 の順序回路の動作と、図 4 の RNN 動作を比較してみる。

図 3 において、まず XOR の入力が外部入力の 1 とフィードバック・ループ側の 0 なので、XOR の出力は 1 となる。そこにクロック (図の clk) が入ることによって、D フリップ・フロップに 1 が蓄えられる。

まず $t = 0$ において、外部入力の 1 とフィードバック・ループからの入力の初期値である 0 が、#1,#2 の入力となる。このとき、先に述べたニューロンの評価式 3 に基づいて出力が計算されるが、この事を **Forward Propagation** (以

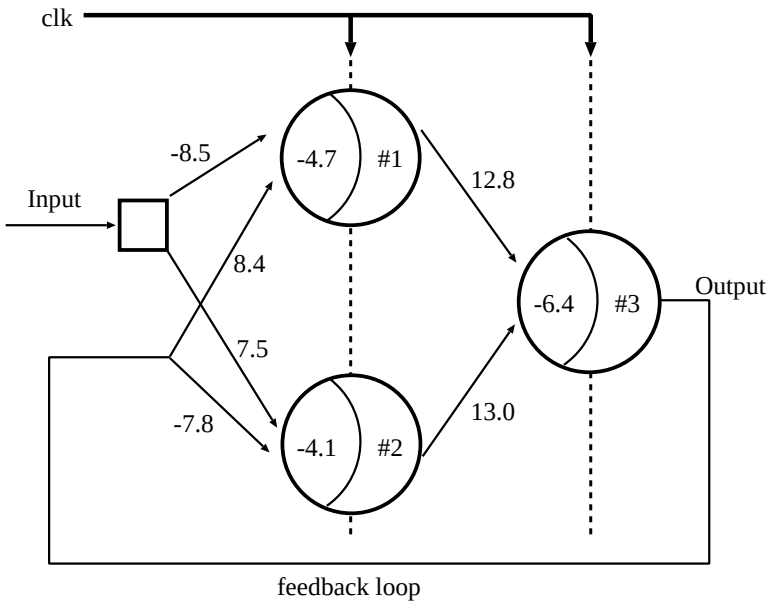

図4: 入力系列のパリティを求める RNN

下、**FP**) と言う。この時点でのニューロン#1 の内部状態は、

$$u = \sum_{i=1}^{n+1} w_i x_i = -8.5 \times 1 + 8.4 \times 0 + -4.7 \times 1 = -13.2 \tag{6}$$

である。#1 の出力は、

$$z = f(u) = \frac{1}{1 + exp(-(-13.2))} = 1.9 * 10^{-6} \tag{7}$$

となり、ほぼ 0 である。同様の計算で、#2 の出力は 0.97 と 1 に近い値となる。さらに#3 についても計算をすると、#3 の出力は 0.99 となり、ほぼ 1 である。これは XOR(0,1) の結果である。

この計算で、図の clk の位置、つまり RNN における同一の段で同期がとられる。例えば、#1 と#2 を同一の段として計算の同期がとられる。次の時刻にその結果が#3 のニューロンの伝播され、その位置を一つの段として同期がとられ、計算がなされる。このように、RNN には図の clk のような仮想的なクロックが存在している。

以上のように、RNN は順序回路と同等な時系列処理を行うことができる。また、RNN の動作は順序回路の動作と対応していることが言える。

2.2 関連研究

我々の研究に関連のある研究として、Jun.T らによる迷路を解くロボットの研究がある。Jun.T らは、壁で仕切られたある一定の空間内に障害物を置き、ロボットにその特定の空間を壁や障害物に接触しないように進ませる研究をしている。

このロボットは可動式のカメラを持ち、カメラによって周りの状況を確認しながら進む方向を決めていく。この時、どの方向に壁や障害物があるかを予想して、カメラを向ける方向を決定している。この壁や障害物の位置の予想に、RNN が用いられている。

RNN の学習は、ロボットが迷路を一度解くごとに行われる。学習において RNN が覚える情報は、各時刻において「どの位置」に「どのような障害物」があるか、という情報である。これらを記憶することで、RNN はカメラをどのようなタイミングでどの方向に向けるかを正しく予想することができる。

この研究において、ある時刻における RNN の出力が示すものは、次の時刻における RNN の入力である。次の時刻での入力を予め知ることで、ロボットは次に進む方向を決定している。これは即ち、RNN は迷路の形状を記憶していて、進行方向を次々に出力しており、ロボットはそれに従って移動している、と言える。つまり、RNN はある特定の迷路の形状を記憶している。

この研究で RNN が覚えさせているものは、特定の迷路空間において、どの位置に壁や障害物があるかという情報である。つまり RNN が記憶している情報は、特定の迷路空間の形状であると言える。

これに対して我々の研究では 1 章で述べたように、特定の迷路ではなく任意の迷路を解く RNN を学習により構成することで、学習によって RNN が覚えるものは、特定の迷路の形状ではなく、迷路を解くテクニックのようなものであると言える。

2.3 学習アルゴリズム

NN は一般に学習によって構成される。フィードバック・ループを持つ NN である RNN でもそれは同様である。

RNN の学習アルゴリズムには様々なものが存在するが、一般には、通常の NN における一般的な学習アルゴリズムである誤差逆伝播法(**Back Propagation**、

以下 **BP**) と同様に、最急降下法に基づいた学習アルゴリズムが用いられる。

この節では、RNN 上に学習をさせるのに使用するアルゴリズムである**通時的誤差逆伝播法**(**Back Propagation Through Time**、以下**BPTT**) の概念とその種類を説明をする。

その上で、実際に迷路を解くネットワークを構成する際に用いたアルゴリズムである**エポック毎通時的誤差逆伝播法**(**Epochwise BPTT**、以下**EBPTT**) について、何故このアルゴリズムを用いたかの理由を交えつつ詳しく説明をしていく事にする。

2.3.1 BPTT

一般に RNN の学習には、先に述べた BPTT アルゴリズムを用いる事ができる。BPTT は、RNN を時間展開することで回帰結合を持たない NN に変換し、BP を適用する手法である。

図5,6 は、RNN およびそれを時間展開した NN である。下図における各層は同一時刻のニューロンであり、各ニューロンの index は上図のものと対応している。図に示すように、RNN を時間展開することで、回帰結合を持たない NN を得ることができる。

時間展開することにより得られる NN は通常の NN とはやや異なっている。通常の NN は最初の層 (入力層) にのみ外部入力を持ち、最後の層 (出力層) にのみ外部出力を持つ。それに対し、得られた NN は各層に外部入出力を持つ。

このため、通常の BP をそのまま適用することはできない。通常の BP は、出力層に入力信号が伝播した時点における出力信号と**教師信号**との二乗誤差を最小化するアルゴリズムである。一方、得られた NN には入力層・出力層が存在しないため、BPTT における最小化する対象は通常の BP とは異なる。

最小化する対象の違いにより、BPTT には以下の 2 つのアルゴリズムが提案されている。

Epochwise BPTT(EBPTT)　最後の層までの各時刻における出力信号と教師信号との二乗誤差の総和を最小化

Real-Time BPTT(RTBPTT)　現時刻における出力信号と教師信号との二乗誤差を最小化

2 つの BPTT における最小化する対象の違いは、それぞれの学習用途に起因する。

ロボット工学等の分野では、学習は off-line と on-line の 2 つに大別される。

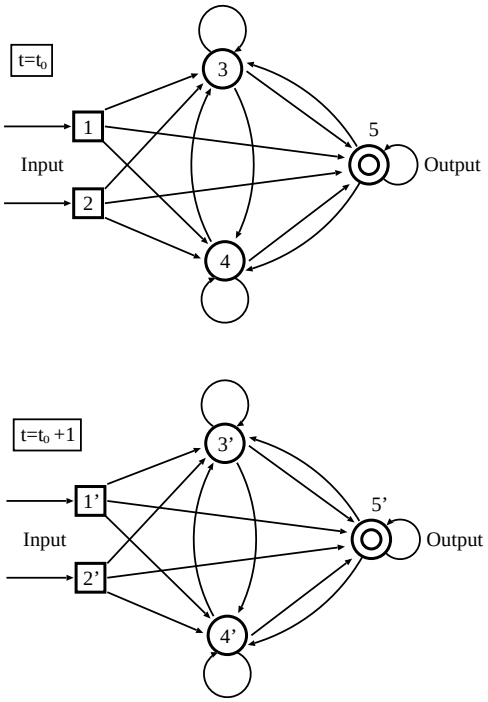

図 5: $t = t_0$,$t = t_0 + 1$ における RNN

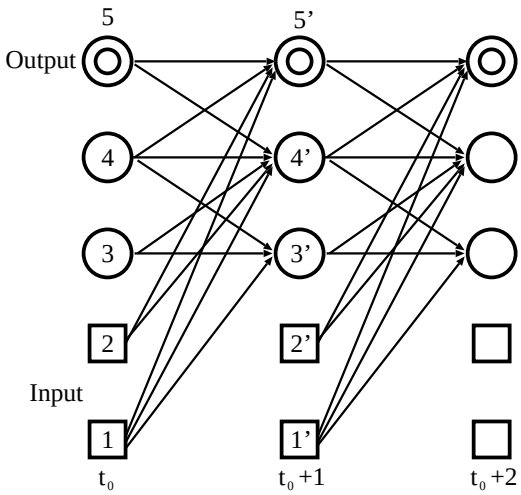

図 6: 時間展開した RNN

前者は学習用の時系列パターンを用意し、それに対し行う学習である。それに対し、後者は学習するシステムの稼働 (入力に対する処理) と並行して、その時系列パターンに対し行う学習である。一般に on-line 学習の方が解空間が次々と変化するために、学習の効率が悪い。

off-line 学習を行う場合は、当然ではあるが、学習時に学習する時系列パターン (**トレーニング・パターン**) が明らかになっている。一方、本来最小化したい対象は NN の出力系列のトレーニング・パターンに対する二乗誤差であり、これを求めることは可能である。従って、off-line 学習には EBPTT が用いられる。

これに対し、on-line 学習時には、トレーニング・パターンが明らかでない。明らかになっているのはその時刻までの系列のみであり、今後の入力系列および教師系列はその時点では明らかでない。従って、上述のように本来最小化したい対象である、トレーニング・パターンに対する二乗誤差を求めることができない。そこで、各時刻においてその時刻の二乗誤差を最小化することにより将来的な誤差を小さくする RTBPTT が用いられる。

1 章で述べたように、我々は RNN に迷路を解くことを学習させる。迷路課題

は、3.1 節で述べるように RNN 自身の出力 (進んだ方向) が次の入力 (迷路における状況) に影響するため、off-line 学習向きとは言えないだろう。しかし、後で述べるように アダプティブ にトレーニング・パターンを変えることで、迷路課題においても、より効率の良い off-line 学習を用いることができる。

また、迷路に対してリアルタイムに学習をしてしまうと、迷路の最初の方の道について最も多く学習が行われる。その結果、RNN は最初の方に学習した道順について、強く記憶しすぎてしまう。そのため、局所解に陥いりやすいという問題がある。

局所解に陥いったネットワークは一般的に、出力が教師とは大きくかけ離れた値を出す。この時、BPTT の学習率を決定するシグモイド関数の微分係数の値が極めて小さくなるので、BPTT によって結合重みの修正される度合いが極めて小さくなる。そのため、局所解に陥いったネットワークは BPTT による学習があまり進まず、局所解からの脱出が困難とされている。

一方エポックワイズに学習すると、パターン全体の誤差を最小化できるので、ネットワークが局所解に陥いる事を避ける事ができる。これらの理由により学習アルゴリズムには EBPTT を選択した。

そこで以下では、off-line 学習向けアルゴリズムである EBPTT について詳しく述べる。

2.3.2 EBPTT

EBPTT とは、Epoch 全体の誤差を最小化するアルゴリズムである。トレーニングパターンの先頭から最後尾までを Epoch とし、Epoch 全体の教師信号との 2 乗誤差を用いて BPTT する事で、トレーニングパターン全体の誤差を最小化する事ができる。

EBPTT のアルゴリズムを以下に示す。

1. 結合重みの初期値**初期重み**を 0 以外の小さな乱数に設定する。
2. パターンの入力系列 ($t = 0$ から $t = N$ とする) を FP して得られた出力と教師信号との 2 乗誤差を、$t = 0$ から $t = N$ までについて足しあわせる
3. 上で求めた 2 乗誤差の和が一定値以下なら学習が成功したとなし、終了する
4. 得られた 2 乗誤差の和に比例してネットワークの重みを変化させ、2 に戻る

結合重みの初期値を 0 以外の小さな乱数に設定するのは、もし全ての初期重みを、例えば 0 のような同じ値に設定して学習すれば、後述するように、誤差は結合重みに比例して伝播されるので、すべての結合重みは同じように変化し

てしまい、非対称な解は決して得られないからである。

式を用いてFPの誤差が最小化される事を示す。

RNNにおいては、階層型ネットワークのような層の概念は明確にはないので、ネットワークを構成するユニットは入力ユニット(I)、出力ユニット(O)、入出力どちらにも属さない隠れユニット(H)の3つに分類される。図2のように、入力ユニットは全ての出力ユニットと隠れユニットに結合され、出力ユニットと隠れユニットはお互いに結合されている。

RNNにおいて、時刻tにおけるユニットiの出力を$z_i(t)$とする。入力ユニットの出力を$x_i(t)$、隠れユニットと出力ユニットの出力を$y_i(t)$とすれば、$z_i(t)$は、

$$z_i(t) = \begin{cases} x_i(t), & i \in I \text{ のとき} \\ y_i(t), & i \in H \cup O \text{ のとき} \end{cases} \tag{8}$$

のように表される。ここでI, H, Oはそれぞれ外部入力、外部出力以外のニューロン、外部出力のニューロンの インデックス の集合を表す。

離散時刻$t+1$における隠れユニットと出力ユニットの入出力関係は次のように表される。

$$y_i(t+1) = f(u_i(t+1))), \quad i\epsilon H \cup O \tag{9}$$

$$u_i(t+1) = \sum_{j\epsilon I} w_{ij} x_j(t) + \sum_{j\epsilon H \cup O} w_{ij} y_j(t) = \sum_{j\epsilon I \cup H \cup O} w_{ij} z_i(t) \tag{10}$$

ここで、w_{ij}はユニットjからユニットiへの結合重み、$u_i(t+1)$はユニットiの内部状態を表す。また、$f(u_i(t))$は単調増加で微分可能な関数であり、ここでは$f(u_i(t))$には2.2節で紹介したシグモイド関数を用いる。

$$f(u_i(t)) = \frac{1}{1 + exp(-u_i(t))} \tag{11}$$

シグモイド関数の微分は、次式によって簡単に求めることができる。

$$f'(u_i(t)) = f(u_i(t)) * (1 - f(u_i(t))) \tag{12}$$

図7に示されているように、RNNに対して、離散時刻$t_0, t_0+1, \cdots, t_1-1$において、それぞれ入力$x_i(t_0), x_i(t_0+1), \cdots, x_i(t_1-1)$を与え、時刻$t_0+1, \cdots, t_1-1, t_1$

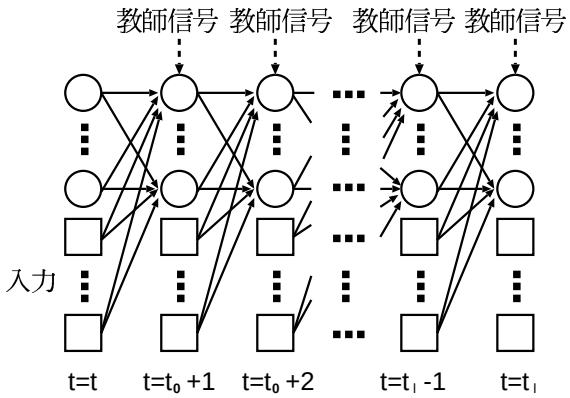

図 7: エポック毎通時的誤差逆伝播法

において、それぞれ教師信号 $d_i(t_0+1), \cdots, d_i(t_1-1), d_i(t_1)$ を与えるとする。

このときネットワーク全体の誤差関数は、各時刻における誤差の総和

$$J(t_0, t_1) = \sum_{\tau=t_0+1}^{t_1} E(\tau) = \frac{1}{2} \sum_{\tau=t_0+1}^{t_1} \sum_{i \epsilon H \cup O} e_i(\tau)^2 \tag{13}$$

で与えられる。ただし、$e_i(\tau)$ は、

$$e_i = \begin{cases} d_i(\tau) - y_i(\tau), & i \in O \text{ のとき} \\ 0, & \text{その他のとき} \end{cases} \tag{14}$$

である。

最急降下法に基づいて結合重みを変更するためには、結合重みの変量を Δw_{ij} として、

$$\Delta w_{ij} = -\eta \frac{\partial J(t_0, t_1)}{\partial w_{ij}} \tag{15}$$

に従って結合重みを変更すればよい。ここで η は学習率と呼ばれる。

時間展開された RNN の各時刻 τ ごとの結合重み $w_{ij}(\tau)$ は時刻 τ に依存しない定数 $w_{ij}(\tau) = w_{ij}$ とであるとすれば、誤差関数 $J(t_0, t_1)$ を結合重みで偏微分すれば、

$$\frac{J(t_0, t_1)}{dw_{ij}} = \sum_{\tau=t_0}^{t_1-1} \frac{\partial J(t_0, t_1)}{\partial u_i(\tau+1)} \frac{\partial u(\tau+1)}{\partial w_{ij}(\tau)} \frac{\partial w_{ij}(\tau)}{\partial w_{ij}} = \sum_{\tau=t_0}^{t_1-1} z_j(\tau) \frac{\partial J(t_0, t_1)}{\partial u_i(\tau+1)} \tag{16}$$

と計算される。

この結合重みの過程に基づく近似により、以下の計算を行うことで、EBPTT

ではニューロン数を N とした場合、その計算量は $O(N^4)$ から $0(N^2)$ へ、メモリ量は $O(N^3)$ から $O(N)$ へ減らすことができる。

そこで、

$$\delta_i(\tau) = \frac{\partial J(t_0, t_1)}{\partial u_i(\tau)}, \quad t_0 \leq \tau \leq t_1 \tag{17}$$

とおけば、$J(t_0, t_1)$ を極小化するような学習則は、$\tau = t_1, t_0 + 1 \leq \tau \leq t_1 - 1$ に依存して、

$$\delta_i(t_1) = (y_i(t_1) - d_i(t_1)) f'(u_i(t_1)) \tag{18}$$

$$\delta_i(\tau) = f'(u_i(\tau)) \left((y_i(\tau) - d_i(\tau)) + \sum_{l \epsilon H \cup O} w_{li} d_l(\tau + 1) \right) \tag{19}$$

と与えられる。この $\delta_i(t_1)$ と $\delta_(\tau)$ が BP における δ 項に対応している。

したがって、結合重みの交信は、

$$\Delta w_{ij}(t) = -\eta \sum_{\tau=t_0}^{t_1 - 1} \delta_i(\tau + 1) z_i(\tau) \tag{20}$$

に従って行えばよい。

第3章　リカレント・ニューラル・ネットワークにおける迷路の学習

迷路内を動くエージェントが迷路を解くとき、分岐点を記憶する事なしに解くような方法は使わない場合は、現在どの方向に向かって進んでいるのか、1つ前の分岐点でどの方向に進んだことがあってどの方向には進んだことがないのか、等の情報を記憶しておく必要がある。ある時刻において、これらの情報はそれまでの入力系列に依存するので、迷路を解くエージェントは時系列に処理をする能力が必要である。

この章では実際に迷路を解く RNN を構成する手順を示す。

まず最初に迷路課題に関しての説明をする。ここではエージェントが解く迷路課題について例を用いて説明する。また、RNN の入出力の設定の説明も行

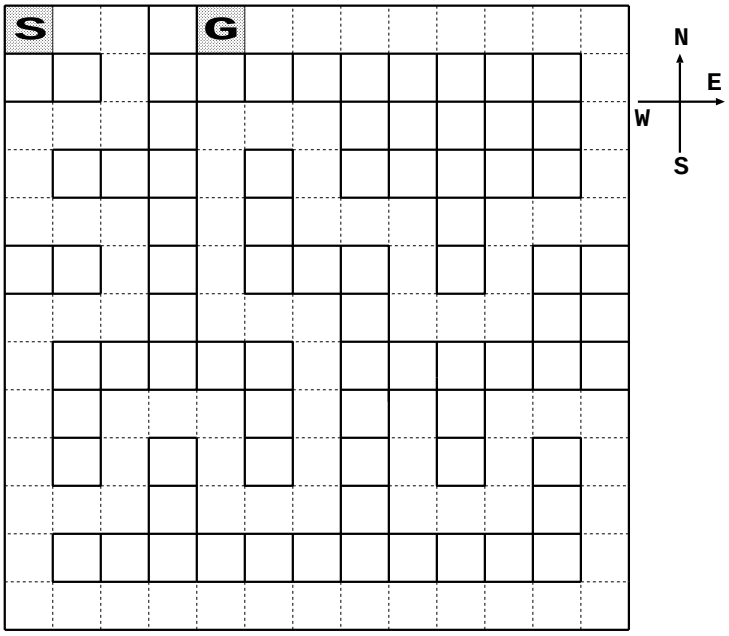

図8: 迷路の例

う。そこでは、RNN とのインターフェイスについて詳しく説明する。

次の節では、教師信号の生成法について説明する。そこでは、学習に off-line 学習である EBPTT を適用するために、教師をアダプティブ に変化させる方法を示す。

最後の節で、学習に用いるトレーニング・セットの選び方について説明する。

3.1 迷路課題

エージェントが解く迷路の例を図8 に示す。迷路は2 次元平面上の空間で、図に示すように複数のマス目に区切られている。図中の S はスタート地点を、G はゴール地点を表している。また、実線で表しているのは壁で、エージェントはこれを通り抜けることができない。破線で表している境界は、そこには壁がないことを示しており、エージェントはこれを通り抜けることができる。エージェントは時刻 $t = 0$ においてスタート地点から出発し、ゴールを目指す。エージェントがゴールに到着する事ができれば、迷路を解いたとみなされる。

ここで、時間の単位を導入する。エージェントにとって最小の時間単位とは何かを考えると、それはエージェントの出力が変化するまでの最小時間である。この時間は2.1 章で述べたように、あるニューロンからそのニューロンに接続された隣のニューロンに信号が伝播するのにかかる時間に等しい。すなわち、FP

1 回にかかる時間がエージェントにとっての最小の時間である。この FP 1 回にかかる時間を **1 単位時間**として、説明を続ける。

エージェントは 1 単位時間の間に、

- (N,E,S,W) いずれか方向の 1 つ隣のマスに移動する
- どの方向にも移動せずに、今いるマスに滞まる

のいずれかの動作をする。この動作はエージェントの出力の値に応じてなされる。

次節からこの迷路を解くネットワークを構成するための入出力環境や、EBPTT の教師信号の生成の仕方について整理する。

3.1.1 リカレント・ニューラル・ネットワークとのインターフェイス

エージェントである RNN が迷路内を歩く際には、観測することのできる壁の位置などから判断して次に進む方向を決める。エージェントは迷路内を歩くので、エージェントが観測できる壁の位置関係は自分の周囲に限られている。また、分岐点においてどちらに進むのかを決定するのに、自分から見てゴールがどの方向にあるのか、という**匂い**の情報もエージェントは利用できるものとする。(どのように匂いを利用するのかは、教師の項で後述する)

このように、エージェントに与えられる様々な情報を、RNN の入力として与える。また RNN の出力を、エージェントが進む方向を表すように定義する。

入力 エージェントが観測できる情報は、自分の東西南北それぞれが壁であるか否かと、ゴールが自分から見てどの方向にあるかという「匂い」の情報のみであるとする。ここで、壁が (N,E,S,W) の 4 方向分の入力があるのと同様に、匂いにも (N,E,S,W) の 4 方向についての入力があるとする。

壁の入力 各方向 (N,E,S,W) に壁があるか否かを 0,1 の値で表現する。

図 9 に壁の入力を示す。図のように、エージェントの隣に壁があればその方向の壁の値は 1 とし、隣に壁がなければ 0 とする。東西南北の壁の値がとるのはこの 0, 1 のいずれかである。

匂いの入力 ゴールがエージェントから見てどの角度にあるかを (N,E,S,W) の 4 方向について、[0,1] の連続値で表現する。

ある方向にゴールが近い角度にあれば、その方向の匂いは 1 に近くなり、逆にゴールがあるのとは正反対の方向の匂いは 0 に近くなる。また、ゴールがある方向から見てちょうど中間の角度にあるとき、例えば、ゴールが向かってちょうど N 方向にある時の E 方向や W 方向の匂いの値は、ちょうど 0.5 になる。

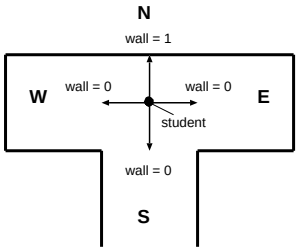

図 9: 壁の入力

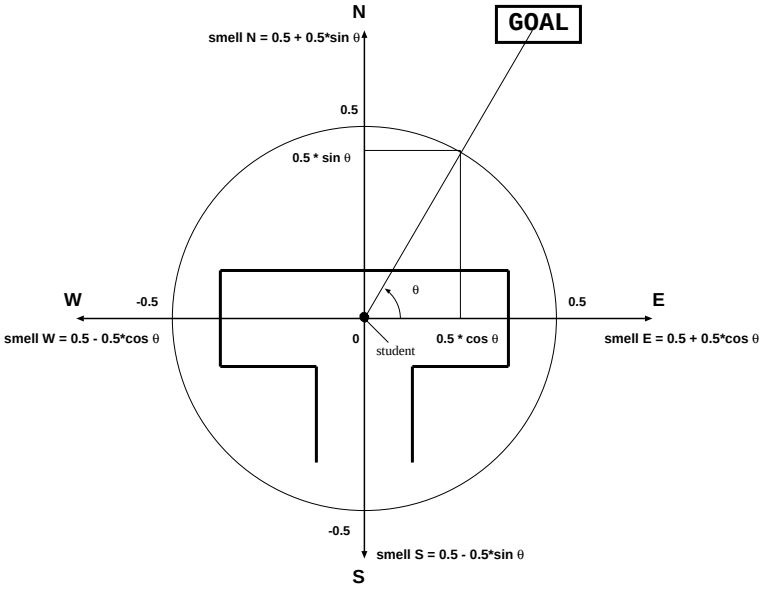

図 10: 匂いの入力

図 10 に匂いの入力が計算される図を示す。

具体的な計算方法を示すと、直交する NS 軸,EW 軸による座標系をとり、エージェントからゴールを見た角度を θ とおく。N 方向については $\frac{1}{2}(1+\sin\theta)$、E 方向については $\frac{1}{2}(1+\cos\theta)$、S 方向については $\frac{1}{2}(1-\sin\theta)$、W 方向については $\frac{1}{2}(1-\cos\theta)$、がそれぞれの匂いの値となる。

以上の 8 本に、RNN に閾値 (**Threshold**) を学習させるために常に 1 を入力し続ける入力 1 本を加えた合計 9 本を RNN の入力とする。(2.1.1 項参照)

出力　エージェントの出力は、(N,E,S,W) について計 4 本ある。出力がとりうる値の範囲は、RNN の出力のとりうる値の範囲と同じなので、$[0,1]$ の連続値が出力のとりうる値となっている。

ある方向の出力値が閾値 **Threshold** 以上でなおかつ、他 3 方向の出力値

が Threshold 未満のときに、エージェントはその方向に進みたいとみなす。本実験では、0 と 1 の中間値として、Threshold = 0.5 とした。

その他の条件、例えば、2 方向の出力が Threshold 以上の時や、どの方向の出力も Threshold を越えない時は、エージェントはどの方向にも進みたくないとみなす。

出力は全時刻において有効(**valid**)であるとし、1 単位時間ごとにこの出力の値に応じての判断が行われる。

この出力の値に応じてエージェントは移動し、移動先の壁と匂いの値が次時刻でのエージェントの入力となる。

このような入出力環境の下で、EBPTT を用いて RNN に迷路を学習させていく。

3.2 教師

2.3 節において、学習アルゴリズムとして EBPTT を用いる事を説明する際に、教師信号を アダプティブ に生成すると述べた。ここでまず、教師信号をアダプティブ に生成することについて、我々の提案する方法が、通常の BPTT における教師信号とどのように違うのか比較して説明する。

通常の BPTT において、例えば N 方向に 1 歩進むということを RNN に学習させようとするときを考える。教師信号は (N,E,S,W) について、N 方向に 1、その他の方向について 0、というように $(1,0,0,0)$ が BPTT の教師信号となる。通常の BPTT における学習では、この $(1,0,0,0)$ を 1 単位時間の間だけ RNN に教える。

我々の提案する方法では、この $(1,0,0,0)$ を RNN に教えるのに、複数の単位時間をかけて教えている。さらに、その間ずっと $(1,0,0,0)$ を教師信号として教えているのではなく、RNN の出力に応じて教師信号を変化せながら $(1,0,0,0)$ に 1 歩進むことを教えている。

この節では、迷路を解く順序回路について考察し、その考察から、迷路内を歩く RNN が 1 歩進むには何単位時間かの待ち時間が必要であるという考察を示す。

次にその考察に基づいて、教師信号に アダプティブ に遅延を挿入して学習する手法について詳しく説明する。

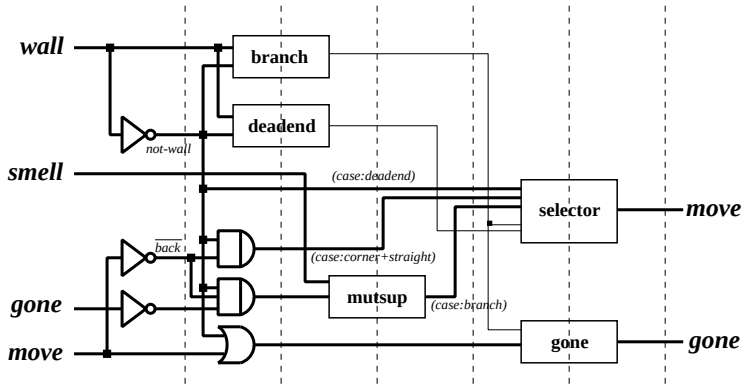

図11: 迷路を解く順序回路

3.2.1 迷路を解く順序回路についての考察

迷路を解く順序回路を図11に示す。これは分岐数1の迷路を解く順序回路である。この順序回路はRNNと同一の入出力を持ち、図のwall、smellがそれぞれ壁と匂いの入力を、moveが進む方向を示している。この順序回路の動作を簡単に説明すると。goneは分岐点において既に行ったことのある道を表していて、これをループさせることで既に行ったことのある道を記憶している。deadend、branchはそれぞれ行き止まり、分岐点を表しており、このユニットによって現在いる地点が行き止まりや分岐点であるか否かを判断する。mutsupは3.2節の教師アルゴリズムの項でも後述するが、分岐点において匂いの強さによって進む方向を決定するユニットである。

この順序回路において、ある時刻で壁と匂いの入力を受けとってから、回路が正しい出力を出すのに必要な時間について考える。図11において、縦に入った破線は回路の段数を示している。正しい出力を得るには、この段数の数だけ待たなければならない。図で段数の数は6段なので、この順序回路が入力を受けとってから、有効な出力を得るのには5段分の遅延が必要である。入力を受けとってからそれまでの時間は、回路は有効でない出力を出している。言いかえると、この順序回路が有効な出力を出した後は必ず、有効でない出力が何段か続く。

このように、迷路を解く順序回路においては何段かの遅延が必要であるが、迷路を解くRNNにおいても同様に何段かの遅延が必要であると我々は考えた。

ここで、このような遅延を含んだRNNの動作の定義には2通り考えられる。1つは、遅延の間のRNNの出力がどんな値であろうがその値をinvalidである

として無視し、RNN の内部状態が安定するまで待ち、内部状態が安定した時の出力の値を valid であるとみなす手法で、もう 1 つは、RNN の出力が全時刻で valid であるとみなして、遅延の間の出力は最終的には 0 になるように学習させていく手法である。

一般に非同期式の機械の場合、ある時刻の出力の値から、その時刻における内部状態が安定しているかどうかを判断するのは困難である。前者の手法においても、RNN の内部状態が安定しているか否かを判断しつつ動作させるのは困難だと考えられる。また、BPTT での学習において、ある時刻では valid な値を学習し、別の時刻では invalid な値を学習させる事は一般に困難である。よってこれらの理由により、本実験では後者の方法を用い、前節でも述べたように RNN の出力が全時刻において valid として、遅延内では最終的に 0 を出力するように学習していく。

この遅延の間は RNN はなんらかの値を出力するが、前述のように RNN の出力は全時刻において valid であると見なしているので、遅延の間においても RNN の出力ユニットのいづれかが 1 を出力すると不都合が生じる。このことから、**遅延の間における RNN の出力は 0** を出力することが望まれる。

RNN が壁と匂いの入力を受けとってから進む方向を出力するのに必要な遅延の数は、上の順序回路が必要とする遅延の数と同じなのか、それともより少ない数の遅延で十分なのかは不明である。よって、RNN の教師信号に任意の長さの遅延を アダプティブに挿入する手法を考案した。

次の項では教師信号に遅延を アダプティブ に挿入する手法について具体的に詳しく説明をしていく。

3.2.2 教師信号の生成法

整理して説明するために、エージェントが迷路内を 1 歩進む際の教師信号を、以下に示すような 3 つの区間に分割する。

信号の立ち下がり区間 エージェントが 1 歩進む前には必ず、1 単位時間の間は 0 を出力する。

遅延区間 任意の長さの遅延を アダプティブ に挿入する。

信号の立ち上がり区間 (N,E,S,W) のいずれかの方向の出力を 1 に立ち上げる。立ち上がりは 1 単位時間の間のみで、次の時刻には出力は上の立ち下がり区間の 0 になる。

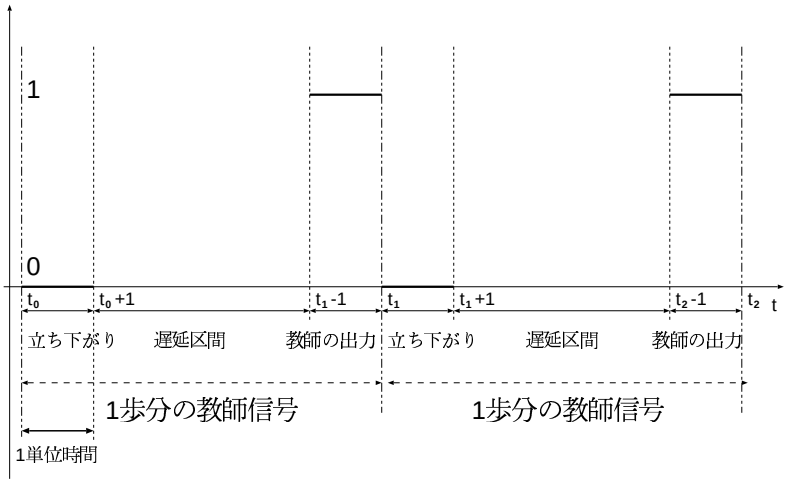

図 12: 教師信号の例

上の信号の立ち下がり区間と遅延区間を併せたものが、上で述べた順序回路における有効でない出力を出している区間にあたる。RNN の出力は全時刻について valid であると見なすので、この 2 つの区間では、最終的には RNN の出力は 0 になることが望まれる。また、信号の立ち上がり区間が、順序回路における有効な出力を出す区間にあたるので、この区間では教師が進む方向の出力が立ち上がることが望まれる。

この教師信号の例を図 12 に示す。

図 12 では、N 方向に 2 歩進むときの、N 方向の出力ユニットに対する教師信号のとる値を示している。図は時刻 $t = t_0$ から $t = t_1$、時刻 $t = t_1$ から $t = t_2$ において、それぞれ N 方向に 1 歩ずつ進むことを示している。

1 歩進むごとにこの 3 つの区間がそれぞれあり、最初に立ち下がり区間があり、この区間での信号は 0 である。次の遅延区間では教師の値が未定義になっているが、教師の値が未定義になっている場合の学習については後の項で説明をする。最後の区間で進む方向の値である 1 を出力している。

以下の項では、これら 3 つの区間について順を追って詳しく説明していく。

信号の立ち下がり区間　エージェントが 1 歩進んだ後には必ず、1 単位時間の間信号 0 を出力する区間を設ける。

これは、迷路を解く順序回路において、有効な出力の後には必ず何段かの有効でない出力が続くからである。

遅延区間　一般に RNN がある入力を受けとったときに、その入力に対する出

力を得るにはある程度の遅延が必要であることは前述した。

何ステップ必要なのかわからない遅延を教師信号に適切に挿入するために、アダプティブ に教師を定義するという方法を提案する。RNN の出力に応じて教師信号を変化させる事によって、適切な遅延の段数がわからないという問題を解決することができる。

適当な長さの遅延区間を挿入する方法として、遅延区間において、RNN の出力と教師の出力が比較的近い値ならば、遅延をそこまでとして打ち切る、という方法を考案した。

詳しいアルゴリズムを以下に示す。

$t = t_0$ から $t = t_1$ までが遅延区間であるとして、

- 時刻 $t = t_1$ であるならば、遅延区間の打ち切りをやめる。
- 時刻 t における RNN の出力と教師の出力の差を、(N,E,S,W) の 4 方向について取り、2 乗して足し合わせる。
- 上で求めた和が一定値未満なら、その時刻から $t = t_1$ までの遅延区間を削除する。和が一定値以上なら、t を 1 増やして上にもどる。

遅延区間では教師信号の値を一定値には定義せずに、その時刻での RNN の出力をそのまま教師信号の値にする。こうする事で、その時刻における教師信号と RNN の出力の差は 0 となる。BPTT による学習においては、RNN の出力と教師信号の 2 乗誤差に比例して重み変化が発生するので、この遅延区間においては、学習による重み変化に全く影響を与えないということになる。

信号の立ち上がり区間　信号の立ち上がり区間は、実際にエージェントが進む方向を示す区間である。(N,E,S,W) の 4 方向について、進みたい方向の出力値を 1、その他の方向の出力値を 0 として出力する。

エージェントが進む方向は、ある一定のアルゴリズムに従って動作する教師が与える。教師は迷路を覚えて解くようなアルゴリズムでなくてはならないので、その点をふまえた上で教師となるアルゴリズムを採用する。

この項ではこの教師アルゴリズムについて詳しく説明する。

教師アルゴリズムは RNN と同様の入出力を持つ。教師が迷路内を歩く際には、スタート地点からゴール地点までの必ずしも最短距離を進むのではなく、ある一定のアルゴリズムに従って迷路内を試行錯誤しつつ進む。

状態機械によって教師を構成した。この状態機械はスタックを持ち、分岐

点での情報をスタックに格納して保持する。分岐点での処理は、分岐点に入ったときは、入って来た方向を覚える。分岐点を出ていく方向は、まだ進んでいない方向の中からゴールの「匂い」の値が最も大きいものを選択する。

ある分岐点について、全ての方向にすすんだにもかかわらず迷路が解けないときは、最初に入ってきた方向にもどる。

また、真っ直な道や曲がり角では道なりに進み、行き止まりでは来た方向にもどる。また、現在いる地点がゴールであれば、そこで探索を終了する。

以上まとめて書き下すと、

- ゴールかどうかの判定
- 真っ直な道 曲がり角道なりに進む
- 行き止まり来た道をもどる
- 分岐点
  - 分岐点に入ったとき入ってきた方向を覚えておく
  - 進む方向の選択まだ進んでいない方向の中からゴールの匂いが最も強い方向を選択する
  - 分岐点から全ての方向に進んだ時最初に入ってきた方向にもどる

このようなアルゴリズムによって選択した進行方向を、教師信号の立ち上がり区間として使用する。

以上のような「立ち下がり区間」「遅延区間」「教師の立ち上がり区間」の 3 つの区間で 1 歩分の教師信号を定義した。この生成法を用いて、迷路をスタートからゴールまで解いたものを EBPTT の教師信号として学習を進めていく。

第 4 章　結果

この章では、これまでに述べてきた手法によって、迷路を解く RNN を実現した結果について述べる。

まず最初に前章で記述した手法で生成した教師信号 (トレーニングパターン) を用いて、トレーニング・セットを構成する方法について述べる。

迷路を解く RNN を構成するのには、どれくらいの量のトレーニング・パターンを学習すれば良いのかは不明である。そのため、冗長なトレーニング・セットの選び方を避ける方法として、解けない迷路をトレーニング・セットに加え

ていく方法を採用した。

その方法を説明すると、

1. まず最初にランダムに生成したトレーニング・パターンをある程度の数学習させ、2 乗誤差が収束した時点で学習をやめる。
2. 得られた重みを用いて学習に使ったのとは別のランダムに生成した迷路を解かせる。
3. 解けなかった迷路について、今までのトレーニング・セットに追加して再度学習させる。

以上の手順の 2 と 3 を繰り返す事で、解けない迷路について加算的に学習を進めていく。

解けなかった迷路について、トレーニング・セットに追加して再学習する際には、それまでの学習で得られた重みを引きついで、新しいトレーニング・セットを学習する。こうすることによって、再学習にかかる時間をいくらか短縮することができる。

以上のような方法でトレーニング・セットを構成した。

また、本実験の学習パラメータには以下のようなものがある。

ニューロン数　迷路を解く RNN を構成するのにどのくらいの量のニューロン数が必要なのかは不明である。

一般に、ニューロン数を N とすると、EBPTT にかかるコストは 2.3 節で述べたように、時間コストが $0(N^2)$、必要メモリが $O(N)$ に比例して大きくなるので、ニューロン数は、迷路を解く RNN が構成できる数を満たした上で、できるだけ少なくするのが望ましい。

迷路を解く状態機械を構成するのに必要な論理素子の数は、多入力 AND や多入力 OR がニューロン 1 つで構成できるとすると、約 60 程度の論理素子で構成できる。

RNN においても状態機械に必要な論理素子数と同程度の数のニューロン数で構成できると考え、実験を行った。具体的には、入力ユニットが 9 個、出力ユニットが 4 個、隠れユニットが 64 個の合計 77 個のニューロンで実験を行った。

迷路のサイズ　本実験ではトレーニング・パターンに用いる迷路を 3.2.3 項で述べたようにランダムに生成した。この迷路のサイズは大きすぎず小さすぎず適当なものとして、13×13 に設定した。

TARGET　EBPTT による学習の終了を判定するパラメータとしてTARGETがある。学習では1つ1つのトレーニング・パターンに対してEBPTTによる2乗誤差を求めていくが、全てのトレーニング・パターンの2乗誤差の和がTARGET値以下になった時に、学習が終了したと判定する。

TARGETの値は、0.5と設定した。

η項　BPの学習率である定数η項がある。(2.3項参照) この値が小さすぎると収束が遅く、大きすぎると解空間の谷をとびこえてしまい、いつまでたっても収束しなくなるので、η項は適当な値を設定しなければならない。

実験を繰り返した経験に基づき、η項の値はパターン1種類につき0.01程度に設定した。

MORATRIUM_MAX　この実験では教師信号を アダプティブ に生成したが、その際に遅延区間の長さの最大値を設定しておく必要がある。その値がMORATRIUM_MAXである。

実験を繰り返した結果、行き止まりや分岐点においても最終的には遅延が1程度におさまった事から、MORATRIUM_MAXの値は4に設定した。

上で述べたようなトレーニング・セットの選択法と学習パラメータを用い、前章で述べたように教師信号を アダプティブ に生成する手法を用いて、迷路を解くリカレント・ニューラル・ネットワークを構成した。実験コストの関係上、全ての迷路を解くリカレント・ニューラル・ネットワークは実現できなかったが、分岐数1の迷路を解くリカレント・ニューラル・ネットワークを実現することができた。

第5章　まとめ

フィードバック・ループを持つRNNは時系列に情報を処理することができ、高次の情報処理機能の実現・解明に役立つものとされている。しかし、RNNにおいて情報がどのように表現され、処理されているのかについては、一般的に述べた研究は少ない。こうした現状はRNNに扱わせてきた課題が、ある特定の機能を実現するというような簡単なものであったからだと考えられる。このような課題を解くRNNのネットワークの形状は、同じ課題を解く有限状態機械とほぼ同等のものとなっており、RNNの性質を生かしているとは言い難い。そこで、本稿ではより難易度の高い課題として、RNNに迷路課題を学習させた。

まず、RNN に迷路を学習させる上でのアルゴリズムについて述べた。ここで RTBPTT と EBPTT という 2 つの BPTT アルゴリズムについて説明し、迷路問題の特徴について述べ、教師を アダプティブ に生成することによって、効率の良い EBPTT を採用することができる事を説明した。

また、迷路を解く順序回路を示し、この順序回路が入力を受けとってから正しい出力を得るまでにはある程度の遅延が必要であることを示した。この順序回路との比較から、迷路を解く RNN においても、入力を受けとってから正しい出力を得るまでにはある程度の遅延が必要であるという考察を示した。この考察から、RNN の教師信号に任意の長さの遅延を入れる、アダプティブ な教師信号の生成法を提案した。

この アダプティブ に教師を生成する手法では、RNN の教師信号に遅延区間という区間を設け、この区間内では RNN の出力に任意の値を許した。その上で、RNN の出力に応じて遅延区間の長さを適応的に変化させた。また遅延区間内では、教師信号の値は RNN の出力の値をそのまま用いることで、遅延区間内における学習による結合重みの変化に全く影響が出ないようにした。

さらに、RNN の学習に用いるトレーニングセットの選択法として、ランダムに迷路を選び、解けない迷路についてさらに学習していくという方法を示した。

これらの手法を用いて、分岐数 1 の迷路を解く RNN を実現した。

謝辞

本研究の機会を与えて下さり、適切な御指導を賜わりました富田眞治教授に深甚な謝意を表します。

また、貴重な御助言をいただいた森眞一郎助教授、中島康彦助教授、五島正裕助手、津村公暁助手、三輪忍氏、津田晃寿氏に深く感謝致します。

さらに、日頃暖かく御鞭撻下さった京都大学工学部情報学研究学科富田研究室の諸兄に感謝致します。

ありがとうございました。

参考文献

[1] Cleeremans, A., Servan-Schreiber, D. and McClelland, J.: Finite state automata and simple recurrent networks, *Neural computation*, Vol. 1, No. 3,

pp. 372–381 (1989).

[2] Jun, T.: Model-Based Learning for Mobile Robot Navigation from the Dynamical Systems Perspective, *IEEE Trans. on System, Man and Cybernetics Part B*, Vol. 26, No. 3, pp. 421–436 (1996).

[3] Omlin, C. and Giles, C.: Extraction of rules from discrete-time recurrent neural networks, *Neural Networks*, Vol. 9, No. 1, pp. 41–52 (1996).