修 士 論 文

再帰型回路網による文法の獲得

北陸先端科学技術大学院大学

知識科学研究科知識システム基礎学専攻

原田 哲治

2001 年 3 月

修 士 論 文

再帰型回路網による文法の獲得

指導教官　　櫻井 彰人 教授

北陸先端科学技術大学院大学
知識科学研究科知識システム基礎学専攻

950071　原田 哲治

審査委員 :　櫻井 彰人 教授 (主査)
林 幸雄 助教授
橋本 敬 助教授

2001 年 2 月

目 次

第1章　はじめに

1.1　まえがき

人はどのようにして言語を獲得するのだろうか．この大きな疑問に答えるべく，言語学，心理学，認知科学，脳科学，計算機科学など多くの分野で，さまざまな角度から研究が続けられている．その中において本研究は，脳の言語獲得モデルの構築[1]とその計算機上への実装を念頭においている．

言語知識には，統語論 (syntax)・意味論 (semantics)・音韻論 (phonology) の 3 要素がある．幼児の言語獲得の際にはこれらは不可分であるが，言語獲得問題を考える上では，これらを同時に扱うのは現状では至難である．本研究では統語論，すなわち文法の獲得に的を絞る．

これまでの言語獲得のモデルは，Chomsky の生成文法理論を背景に，記号処理をベースとするシステムとして構築されたものが多い．一方，その記号処理とは対照的な計算方法である人工神経回路網 (artificial neural network, ANN) は，学習能力，汎化能力，並列化による高速な計算，ロバスト性などに優れるが，構造をもつデータの取り扱いが不得意であり，記号処理，そしてその典型である自然言語処理には向いていないとされてきた．しかし 1990 年前後から，構造データを取り扱う ANN モデルが多数提案されてきており，コネクショニスト的記号処理 (connectionist symbol processing) 研究が盛んになっている[2]．本研究では，先に挙げたような ANN の優れた点を活かした，再帰型回路網 (recurrent neural network, RNN) による文法の獲得モデルの構築を，具体的な実現目標とする．また，RNN としては，その最も基本的なモデルのひとつである単純再帰型回路網 (simple recurrent network, SRN) を使う．

現在の connectionist symbol processing の大きな問題のひとつに，構造を含むデータ（たとえば，リストや木など）を ANN でどのように取り扱うか，という問題がある．構造を含む情報は可変長であるが，ANN での情報処理の基本は固定長パターンの処理である．構造を含む可変長のデータをどのようにして固定長で表現するのか．その解決方法のひとつとして，Pollack の RAAM (recursive auto-associative memory)[15] があり，本研究では，構文解析木の表現にこの RAAM を使う．

形式言語理論によれば，自然言語の重要な構造である入れ子構造は正規文法 (regular grammar, RG) では記述できない．入れ子構造を表現する再帰的規則をもつ文法クラスとして，文脈自由文法 (context-free grammar, CFG) クラスがある．RNN で例文から正規言語 (regular language, RL) の受理を学習することが可能であることは多くの研究によって示されてきた．しかし CFG の学習が困難であることもやはり多くの研究で示されており，例

[1] 実現への諸課題は文献 [19] に概説されている．

[2] 当時の様子は 文献 [14] に簡単にまとめられている．また現在については文献 [1] を参照のこと．

文からの CFG の獲得は現在の connectionist symbol processing 研究の焦点のひとつとなっている．この困難の最大の原因は，CFG の再帰的規則にある．文脈自由言語 (context-free language, CFL) の構文解析 (parsing) にはスタックが必要であり，RNN はスタックの表現・操作を獲得しなければならない．現時点ではスタックを深さ制限付で獲得しているモデルはあるものの，再帰的規則を獲得していると言えるものはまだない．

本研究では，RNN が自然言語風のある CFG から生成される例文から，その文法に基づく構文木を生成すること（構文解析）を学習する，というタスクを採用する．学習に使った例文に含まれる名詞句の埋め込み深さ以上の深さをもつテスト文を，学習後の RNN が正しく構文解析できれば，その RNN はスタックの表現とその操作を獲得していると推定できるのである．

1.2 本研究の概要

提案する再帰型回路網モデルは RAAM および SRN で構成され，学習対象のそれぞれの文法から生成される例文から，その構文解析を学習する．実験では，英語風文法，日本語風文法を学習対象の自然言語風文法として採用する一方，それに先立って，モデルの基礎的な再帰的規則の獲得能力を見るために，言語 $\{a^n b^n\}$ を生成する簡単な CFG も学習対象に加えた．RAAM は構文木を分散圧縮表現することを，SRN は構文解析を実行することを学習する．RAAM は，学習に成功すれば，小さな構文木を繰り返し埋め込んでいくことで，任意の深さの構文木を固定長のパタン（これを本稿では “構文木の RAAM 表現” と呼ぶ）として表現することが可能になる．SRN は文中の単語を前からひとつずつ読み，その時点で可能な最大の構文木（の RAAM 表現）を出力することを学習する（すなわち，本モデルは LR パーザ[3] を ANNs で学習するものだと言うことができる）．RAAM は SRN に先だって学習しておき，SRN の学習の際には，RAAM は SRN に教師信号としての構文木 RAAM 表現を提示する．英語風および日本語風の自然言語風言語の学習の際には，SRN には先読みを許した．

本モデルを計算機に実装して実験を行なった結果，モデルは，単純な CFG から生成される言語 $\{a^n b^n\}$ の構文解析について，学習した文の 2 倍以上の長さの文を構文解析可能であるという汎化能力を示した．ただし，学習した SRN が文の途中と最後とを区別しているかどうかは，この実験では定かではない．

自然言語風の 2 種類の CFG（英語風，日本語風）については，長さ 10 程度（埋め込み深さ 4 程度）までの例文について学習が可能であった．学習に使用したすべての文について，文途中の構文木も含めて，正しい構文木の出力に成功している例が見つかっている．これにより，提案するモデルに基本的な構文解析能力があることが確認された．また，より長いテスト文の構文解析を正しく行なうという汎化能力も若干ながら確認された．しかし，埋め込み深さについての SRN の汎化能力は確認されなかった．ただし，RAAM については，深さ 3 までの文で学習して，深さ 4 の文を正しく表現する例があった．

本研究は，次のような点で新規性がある．コネクショニスト・モデルで LR パーザを学習する研究で，ネットワーク・アーキテクチャは本研究と同様な先例があるが，学習方法

[3]CFL を統語解析するパーザのひとつ．スタックを用い，シフト操作と還元操作を施しながら，ボトムアップ的に統語解析を行なう．詳しくは文献 [22] を参照のこと．

まで含めて本研究と同様なモデルを使っている事例はない．コネクショニスト・モデルによる文法獲得研究において，文の長さおよび関係節の埋め込み深さについての汎化能力の詳細な評価をしている例もこれまでない．さらに，右回帰的要素を含む文法と，左回帰的なそれとを同等のネットワーク・アーキテクチャで学習を試みたところも本研究の特徴のひとつである．また，$\{a^n b^n\}$ の埋め込み深さに対する RAAM の汎化能力を示した実験もおそらく本研究が初めてである．

本稿の構成は次の通りである．まず，次章で connectionist symbol processing の主要な研究を概観し，この分野における諸課題と，本研究の位置づけとを明確にする．第 3 章では，本研究で提案するモデルの詳細な説明を行ない，第 4 章において，モデルを計算機上で実装し，シミュレーション実験を行なった結果を述べる．第 5 章では，実験結果の検討を行ない，提案モデルの問題点，コネクショニスト・モデルを使った構文解析について議論する．最後の第 6 章では，本研究での結論を示す．

第2章　Connectionist symbol processing

コネクショニスト的記号処理の研究は古くからなされている．その主要な研究を概観する．

2.1　RAAM: 可変長のデータを固定長で表現する

Connectionist symbol processing の大きな問題点のひとつとして，構造を持つデータがうまく扱えないということがあった．構造を持つデータ（リストや木など）は，データが可変長になる場合がある．ANN のアーキテクチャは一般に有限固定であり，可変長データの取り扱いがうまくできない．可変長データを如何に固定長で表現するか，が問題になる．その有力な解決策のひとつとして，Pollack が提案した RAAM (recursive auto-associative memory)[15] がある．たとえば Rumelhart ら [18] は，互いに垂直な 8 ビットのビット・パターンを，入力値自身を出力ターゲットとする自己連想の形でフィード・フォワード型ネットワークを学習させることにより，ネットワークの 3 ビットの隠れ層に，それぞれの入力パターンに対応する表現を得ている．このときネットワークは 8 ビットの情報を 3 ビットで表現することを学習した．つまりこれは，情報の圧縮をしたことに相当する．RAAM はこれを拡張したようなモデルだと言うことができる．

Pollack は RAAM を，スタックの状態を固定長で表現することに使った．RAAM は図 2.1 のような 3 層のフィード・フォワード型ネットワークで，自己連想を誤差逆伝播法（backpropagation,BP）[18] で学習することにより,（入出力層よりユニット数の少ない）隠れ層に入力データの内部表現を得るものである．

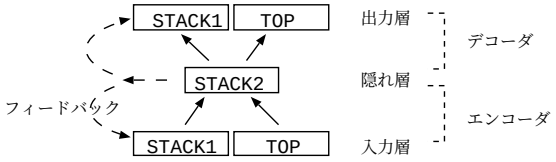


図 2.1: Pollack の RAAM．エンコード部では，STACK1 に TOP がプッシュされ，隠れ層に STACK2 が得られる．デコード部では，STACK2 から TOP がポップされ，STACK1 と TOP に戻る．

スタックの状態と，それにプッシュする表現とを RAAM ネットワークへの入力とし，それと同じパターンをターゲットとして学習すれば，隠れ層に表現されているパターンは，入力のスタックに，それと一緒に入力した表現がプッシュされた，スタックの新たな状態の表現であると Pollack は考えた．このとき RAAM の入力層から隠れ層までをエンコーダ，隠れ層から出力層までをデコーダとすれば，エンコーダはプッシュの，デコーダはポップの役割を果たす．

この RAAM ネットワークが学習に成功すれば，ネットワークの隠れ層に得られるスタック表現を再帰的に用いることにより，任意の深さのスタックの表現が可能になる．可変長のデータが，固定長で表現される例の一つである．また，この RAAM は，スタックの表

現だけでなく，木構造の表現などにも用いることができる．Pollack は RAAM を，簡単な自然言語風の CFG から生成される構文木の表現に用いた．その結果，RAAM が学習していない文の構造も表現する汎化能力を示したと報告している．しかしそこでは，名詞句の再帰的な埋め込みや，学習した文より長い文に対する汎化力の評価はなされていない．

Chalmers は．この RAAM 技術を使い，簡単な英文の能動態／受動態変換を ANNs で実現した [4]．ここで RAAM は能動態／受動態のそれぞれの文を圧縮表現するのに使われる（図 2.2）．能動文の圧縮分散表現と，それに対応する受動文の圧縮分散表現との間の相互変換は，フィード・フォワード型のネットワークが行なう．RAAM と フィード・フォワード型ネットワークとを組み合わせたこのシステムは，学習に使っていない文を変換する汎化能力を見せた．

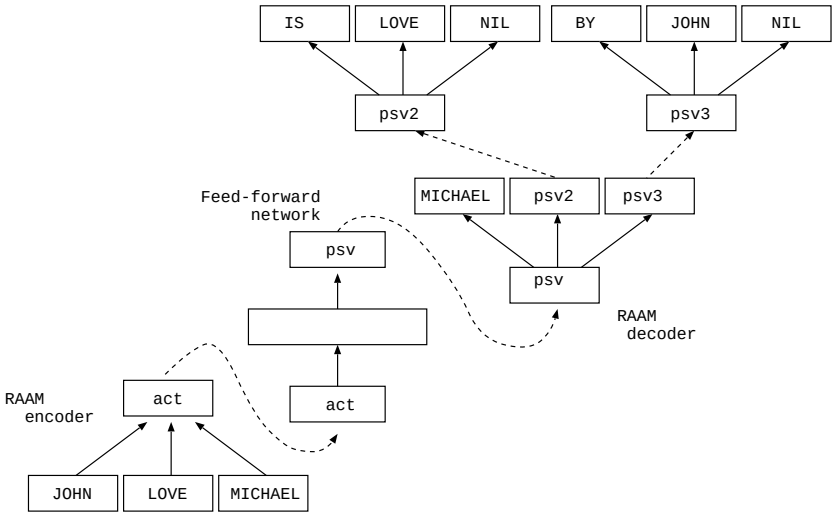


図 2.2: Chalmers の RAAM を使った能動文／受動文変換の概略図．例は “JOHN LOVE MICHAEL” という能動文を，対応する受動文 “MICHAEL IS LOVE BY JOHN” に変換している．act, psv などは RAAM によって作られた圧縮分散表現である．“NIL” はダミー表現．RAAM エンコーダは能動文を圧縮している．フィード・フォワード型ネットワークは文の圧縮分散表現の間の変換を行ない，RAAM デコーダは圧縮表現から単語を取り出す．Chalmers の実験では，受動文の圧縮分散表現に対しては 2 回のデコードが必要である．

Chalmers の実験の重要な点は，圧縮分散表現された文の間の変換を，その文の個々の要素（ここでは単語）をひとつひとつ取り出すことなく，全体として一括並列処理することによって実現している，ということにある．このことは holistic computation と呼ばれている．holistic computation については 2.4 節で詳述する．

2.2 SRN: 時系列情報を処理する

前節で見たように構造データをうまく表現していると思われる RAAM だが，そのままでは時系列的なデータの取り扱いがうまくできない．その点で好都合なのは RNN であり，その最も単純なネットワークのひとつが Elman により提案された SRN (simple recurrent

network, SRN)[5] である．SRN は，フィード・フォワード型ネットワークの，1 時刻前の

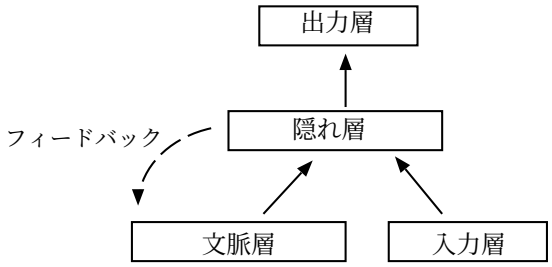


図 2.3: SRN の基本形．文脈層には，1 時刻前の隠れ層の出力がそのままコピーされる．これにより SRN は，時系列パターンの学習をすることができる．

隠れ層の出力をそのまま現入力の文脈層にコピーして利用する（図 2.3）．学習アルゴリズムには BP を用いることができる．過去の情報を持つ文脈層を入力に加えることにより，SRN は時系列パターンを学習することができる．Elman はこのネットワークに，XOR の時系列入力に対して次のビットを予測するタスクを学習させた．これはある種の有限オートマトンを学習させることに相当する．

また，Servan-Schreiber らは，有限状態オートマトンを，それが生成する正規言語の例文を SRN で学習することにより同定する実験を行なった [20]．学習したネットワークの隠れ層の活性パターンのクラスタが，生成されたオートマトンの各状態を表している，と彼らは主張している．

SRN は正規文法を学習することができるようである．しかし，再帰的規則をもつ文脈自由文法についてはどうなのか？ これについて Elman は重要な実験を行なっている．Elman は，SRN を用いて，関係節生成を含む簡単な CFG で生成される単語列を対象として，単語列中の単語が一つずつ入力されるとき，次に来るべき単語を予測するような学習を行なった [6]．その結果，埋め込み文を超えた主語と述語の一致を予測することが確認された．しかし深さ 3 を超える関係節を含む文には対応できておらず，関係節の埋め込みの深さや，文の長さに対する SRN の汎化能力についてもこの実験では明確にされていない．

Elman のこの実験のもうひとつ重要な点は，隠れ層のダイナミクスを観察していることである．ネットワークの隠れ層のダイナミクスとして文法が獲得されていることを示唆している．

RNN による再帰的規則の獲得については，Rodriguez らが，RNN で簡単な CFG である言語 $\{a^n b^n\}$ の予測タスクを学習し得ることを示した研究 [17] がある．彼らの実験では，ネットワークは，$\{a^n b^n\}$ の $n \leq 11$ の例文で学習して，$n \leq 16$ まで汎化して認識することに成功した．また彼らは，RNN の隠れユニットの挙動を非線形ダイナミクスとして詳細に観察し，隠れユニットがカウンタの役割を果たしていると考察している．

Rodriguez らの研究は，RNN が再帰的規則を獲得しうることを示したという点で重要である．しかし，$\{a^n b^n\}$ の単語予測タスクでは，RNN がスタックを獲得したとまでは言えない．また，彼らの研究では，ネットワークが構文解析をしているかどうかは明示的ではない．

2.3 ANNs で構文解析をする

構造データ表現ができるとされる RAAM と，時系列情報を扱うことができる SRN とを組み合わせて，言語処理をしようという発想は自然である．本研究で考察するモデルも RAAM と SRN とを組み合わせている．その先駆けは，Reilly の実験 [16] に見ることができる．

Reilly のモデルは，簡単な自然言語風の CFG から生成される文を RAAM で学習して構文木の RAAM 表現を作り，その RAAM 表現を出力ターゲットとして SRN を学習させる，というものである．SRN には文中の単語を一つずつ入力するが，文の最初から最後の単語まで一貫して，ターゲットは完成した文の RAAM 表現である．結果，SRN はサンプル文のうちのいくつかを学習することができなかった．また，汎化能力も極めて限定されたもので，ひとつのテスト文を最初から最後まで正確に構文解析できた例はなかった．しかし，RAAM と SRN との組合せで構文解析をする可能性を示したという点で，この実験の意義は大きい．

本研究にもっとも近い手法を研究しているもののひとつとして，Mayberry & Miikkulainen の研究 [13] をあげることができる．Reilly のモデルと同じく，彼らのモデルも RAAM と SRN とをベースにしており，さらに文構造情報を保持する SOM (self-organization map) を SRN への入力に組み入れたシステムとなっている．Reilly との大きな違いは，Mayberry らは，SRN への単語の入力に対して，ターゲットとして途中の構文木を与えていることである．つまり，ANN に LR パーザそのものを学習させようとしている．この点において，手法としては本研究のとる方法とほぼ同一である．しかし，彼らの研究ではシフトと還元の操作を SRN で学習することと， SRN の過去情報保持能力の向上に焦点を当てており，名詞句の再帰的規則の学習については検討を避けている．

RAAM および SRN を応用したいくつかのパーザについて，Ho & Chan はそれらの性能比較を行なっており [11]，Reilly，Berg [3]，Sharkey & Sharkey [21]，Ho & Chan [12] らのモデルを取りあげている．性能比較は汎化能力（未知の文の構文解析）とロバスト性（非文の訂正能力）について行なわれているが，文の長さ，深い関係節の埋め込みなどの汎化能力については検討されていない．

2.4 Holistic computation

以上に見てきたように，コネクショニスト・モデルは，記号処理，構文解析の能力の可能性を秘めているようである．しかしそれは，従来の記号操作をベースとする AI 手法で既に行なわれていることである．確かに ANN は学習能力などに優れるが，記号処理はもともと不得手である．それでもあえてコネクショニスト・モデルで記号処理をするということに，一体どういう意義があるのだろうか．

Fodor & Pylyshyn [9] は，次のように痛烈にコネクショニズムを批判した．i) コネクショニスト・モデルでは記号を明示的に取り扱うことができない，したがって人間の高次認知機能を説明するモデルにはなりえない. ii) 仮にコネクショニスト・モデルで記号処理が可能だとしても，それは AI における記号処理の焼き直しに過ぎず，それを超えるものでは

ない．

それに対する反論が，Van Gelder [23]によって提出されている．それによれば，コネクショニスト･モデルが扱う構造情報は functional compositionality であって，従来の AI における構造情報である concatenative compositionality とは区別される．concatenative compositionality では，記号トークンは明示的に結合 (concatenate) されて構造情報となる．しかし，functional compositionality では，トークンは表現上に文字通りに現れていなくてもよい．表現された構造情報と，トークンを combination/decomposition する function とがうまく機能しさえすれば，それは functionally compositional であるということができる．すなわち，大きな違いは，コネクショニスト・モデルでは，トークンの操作が複雑な function によって実現されているところにある．

コネクショニスト・モデルでは，functionally compositional に表現された構造データを，その中に暗示的に含まれているであろうトークンひとつひとつを抽出することなく，包括的に取り扱うことが可能になる．このことは，holistic computation と呼ばれている．この能力はただの記号操作の焼き直しではない，と Chalmers は主張している [4]．コネクショニスト・モデルは実数の連続空間を利用して，より豊かな構造情報の表現と，その操作が可能になるというのである．

Hammerton [10] は，holistic computation の定義を次のように提案している．

> holistic computation とは，オブジェクトのすべての構成要素 (constituents) に対して，構成要素のある位置を探したり，個々の構成要素にアクセスすることなく，同時に作用する計算過程のことである．(文献 [10], p.12)

holistic computation の利点のひとつに，オブジェクトの大きさや構造によらず，すべての構成要素に同時に作用するような計算ができる，ということがある．Hammerton の定義によれば，holistic computation には必ずしも holistic representation が必要なわけではない．しかし，従来の AI における記号操作をベースとしたシステムでは，記号をひとつひとつ処理していく必要があり，holistic computation を実現することは困難である．このことからも，コネクショニスト・モデルは holistic computation を行なうシステムとして有望であり，従来の AI 的手法の単なる焼き直しではなく，それと並んで人間の高次認知機能に迫るモデルの選択肢のひとつとなり得ると考えられている．

本研究は，このアプローチを採用し，holistic computation を扱っていると考えられる RAAM と SRN とを用いた文法獲得モデルを提案する．

第3章　モデルと実験手法

この章では，本研究で提案するモデルを説明し，その能力を確認するための実験方法を述べる．実験は，すべて計算機上でのシミュレーションで行なう．実験過程は大きく三つに分けられる．すなわち，文法からの文生成，RAAM の学習，および SRN の学習，である．

3.1　文脈自由文法からの文生成

3.1.1　学習対象となる文法

実験に用いる 3 種類の文脈自由文法を表 3.1 に示す．i) は ANN による文法獲得実験においてベンチマークとしてよく用いられている，単純な再帰的規則をもつ文脈自由文法である．本研究においても提案するモデルが基本的な再帰的規則を獲得する能力を有するかどうかのテストとして，まず i) の文法を学習する実験を行なう．ii)，iii) は自然言語風に，関係節の埋め込み文を生成する再帰的規則をもつ．ii) は右回帰的な要素を含む文法であり，iii) は逆に左回帰的な要素を含んでいる．X バー理論でいえばこの違いはヘッドがあとになるか先になるかの違いであり，この違いは人間が言語を獲得する段階で接する例文によりセットされるパラメータの違いである，とされる．本研究では，同一アーキテクチャと学習方法で，例文の違いだけから右回帰・左回帰双方の文法がそれぞれ獲得できるかどうかを確認するために，この二つの文法を採用する．

表 3.1: 実験に用いる文脈自由文法．S, NP, VP は非終端記号，それ以外は終端記号として扱う．ii), iii) においてそれぞれの記号の意味は; S: 文，NP: 名詞句，VP: 動詞句，N: 名詞，Vi: 自動詞，Vt: 他動詞，who: 関係代名詞，“.”: ピリオド，である．

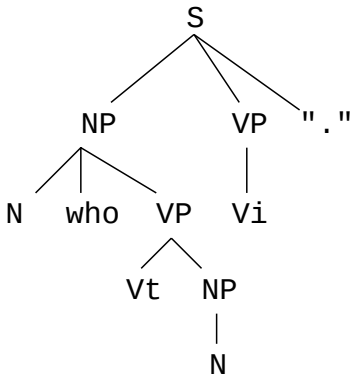
i) $a^n b^n$ 文法	iii) 日本語風文法
S → a b | a S b	S → VP “.”
ii) 英語風文法	VP → NP Vi | NP NP Vt
S → NP VP “.”	NP → N | NP Vt N | Vi N
NP → N | N who VP | N who NP Vt	
VP → Vi | Vt NP	

なお，ii) および iii) の自然言語風文法は，文の前から順に単語をひとつづつを読んだ際，現在読んでいる単語に対して，どの書き替え規則を適用すべきか，一意に決定することが

できない場合がある．その決定には，現在読んでいる単語の次の単語を先読みする必要がある．このような文法の代表的なものとして，LALR(1) 文法がある．ここで用いる ii) および iii) の文法は，LALR(1) 文法である．

3.1.2 文生成アルゴリズム

本研究では，構文木を記号列で表現する[1]．図 3.1 で説明しよう．木のノードにあるのが非終端記号であり，木の葉が終端記号である．あるノード以下の要素は，そのノードの非終端記号の右横に括弧で閉じる形で左から順に並べることにする．たとえば S から深さ 1 にあるノード NP には，そのひとつ深い要素として N, who, VP があり，これを NP(N who VP) のように表現する．この操作をここでは，非終端記号を展開する，ということにする．ところでこの VP にはさらに要素があり，VP(Vt NP) と展開できるから，先ほどの NP は NP(N who VP(Vt NP)) と表現できる．さらに VP の下の NP にも…，というように，それ以上展開できない終端記号で止まるまで，非終端記号を括弧で展開して表現していく．



a: S(NP(NwhoVP(VtNP(N)))VP(Vi).)
b: N who Vt N Vi .
c: dog who bite boy run .

図 3.1: 構文木の例．a: その記号列表現．b: その終端記号だけからなる文．c: b に具体的な名詞，動詞を当てはめた例.

文脈自由文法からの文[2]生成には，横型探索方式で，指定長さ[3]以下の文を網羅的に生成する方法を用いた（図 3.2）．そのアルゴリズムは次のようになる．

i) キューを用意し，それに開始記号 S を投入する．

ii) キューの先頭要素（記号列）を取り出す.

iii) 記号列を左から走査し，展開されていない最初の非終端記号を探す.

iv) 該当する非終端記号について規則を適用し，右辺の記号列に書き換える．一つの終端記号について複数の規則があったら，それらをそれぞれ適用して，新たな記号列を作る．

[1]以降，とくに区別する必要があるときには，これを “構文木シーケンス表現” と呼ぶ.
[2]本研究では，終端記号のみからなる記号列を “文” と呼ぶ.
[3]長さとは，本実験では，文中の終端記号の数を言う．ただし自然言語風の場合はピリオドは数えていない.

v) 該当する非終端記号が記号列中になければ，それは文とする．

vi) 書き換え規則適用後の記号列を，キューに追加する．ただし，指定長さ以上の記号列はキューに追加せずに捨てる．

vii) ii) ～ v) を，キューが空になるまで繰り返す．

このようにして，文脈自由文法から，指定長さ以下の文を網羅的に生成する．

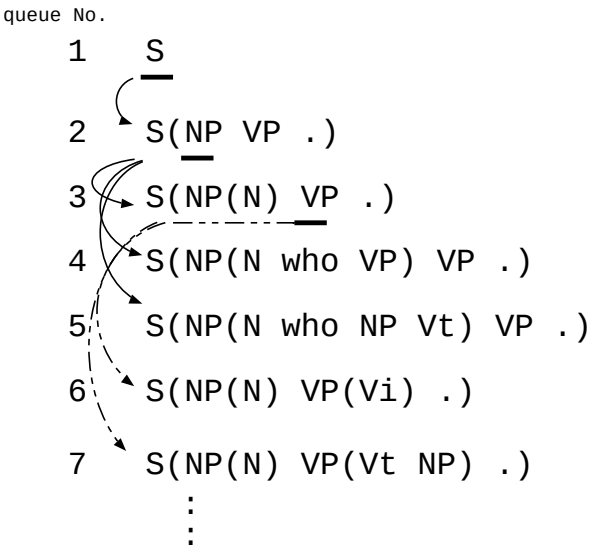


図 3.2: 英語風文法からの文生成の様子．記号列を左から走査して，未展開の最初の非終端記号に書き替え規則を順次適用していく．

実験では，簡単のため，文は具体的な単語レベルに展開しないものを用いた．すなわち，N を boy や dog に，Vi を run などに，Vt を bite などに展開しない．これには学習対象となる文の数を減らすという効果と，単語を名詞，自動詞，他動詞などにカテゴライズするというタスクを省く効果とがある．単語のカテゴリ識別を学習することは自然言語獲得を考える上では重要であるが，本研究ではとくに再帰的規則の獲得に焦点を当てるために，あえてこのタスクを省いて学習の負担を減らした．

このようにして生成された構造情報を含む文を，ネットワークの学習および汎化テストに使用する[4]．なお，以下の実験では正文（学習対象となる文法にかなった文）のみを用いている．

3.2 モデル概要

本研究の目的は，自然言語風の文脈自由言語の構文解析をすることを，その例文から学習するような再帰型回路網を構築することである．回路網は，与えられた文の単語を前から順にひとつづつ入力され，それに対応する構文木を出力するというタスクを学習する．ネットワークは，文の最後の入力で完全な構文木を出力するだけでなく，文途中の単語の入力に対しても，その時点で可能な最大の構文木を出力することを要求される．

[4]付録 A に，自然言語風の文法から生成される文一覧がある．

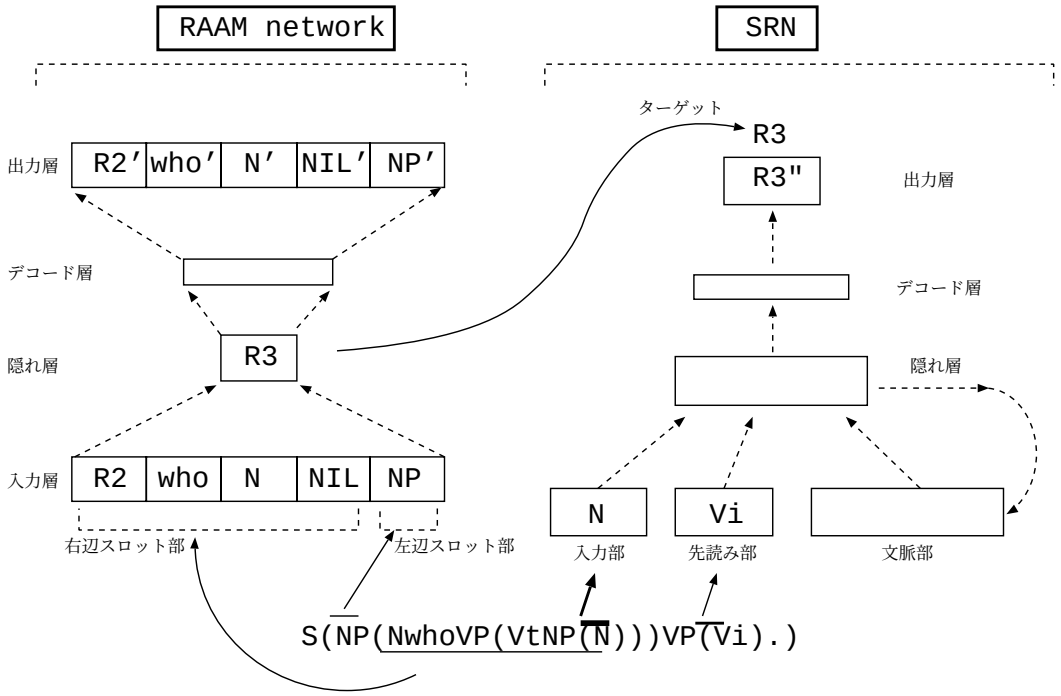


図 3.3: モデル概略図．例は NP(N who VP(Vt NP(N))) の処理を学習しているところ．R3 が NP(N who VP(Vt NP(N))) の RAAM 表現．R2 は前もって作られている VP(Vt NP(N)) の RAAM 表現．詳しくは図 3.4 および図 3.5 を参照のこと．

構文解析をする再帰型回路網として，SRN を用いる．SRN は文中の現在読んでいる単語と，それまでの文脈とを入力とし，それらに応じた構文木を出力するが，ネットワーク・アーキテクチャが固定されているため，可変長の構文木の表現にはそのままでは対応できない．可変長の構文木を固定長で表現するために，RAAM を導入する．RAAM は，構文木の自己埋め込みをフィード・フォワード型ネットワークで学習することにより，隠れ層に入力構文木の圧縮分散表現[5]を得るものである．学習に成功すれば，これを再帰的に用いて，実数の小数展開した下位部分を用いて表現することにより，任意の深さの構文木を固定長の表現に変換できることになる．SRN は，あらかじめ学習済みの RAAM が与える構文木の RAAM 表現を教師信号として学習を行なう．つまり，SRN は，構文木の RAAM 表現を出力対象とする．

一回の試行で，学習対象となる文法から生成された文のうち，指定長さ以下の文をすべて学習に用いる．学習に用いる文をサンプル文と呼び，それらの集合をサンプル文セットと呼ぶ．汎化テストは，学習に用いたものより長い文を使って行なう．これらの文をテスト文と呼び，それらの集合をテスト文セットと呼ぶ．

[5] この隠れ層での構文木の表現のことを，本稿では RAAM 表現と呼ぶ．

3.3 構文解析とネットワークの学習方法

学習する構文解析方法は，LR パーザ [22] と同等である．この方法ではスタックをひとつ用意し，シフト操作 (shift)，還元操作 (reduce) という操作を順次適用しながら，決定的にボトムアップ解析を行なう．簡単に言えば，前節で述べた文生成アルゴリズムの逆，すなわち，生成規則の右側に一致する記号列を見つけたら，それを左側の非終端記号に置き換えていく（これを “還元する” という），という操作を，最初の開始記号が現れてこれ以上還元できない，というところまで続ける．

本実験で用いる自然言語風のふたつの文法（英語風，日本語風）は，統語解析アルゴリズムとしては，単語をひとつ先読みをすれば決定的であるが，先読みなしでは非決定的である．すなわち，ある単語の入力に対して，そこまでに入力された単語だけでは，還元に用いるべき規則が一意に定められない場合がある．その場合はふつう，パーザに先読みを許すことによって構文解析を行なう．

先読みをしない場合（表 3.2）は，次の単語を読んでからはじめて前に読んだ単語の処理が決定されるので，先読みする場合に比べて，操作が少し複雑になる．この場合，還元に用いるべき規則が確定するまで，途中経過をスタックに積みつつ，新たに単語を読み続ける．すなわち，先読みしさえすれば還元できるときにも，シフトをすることになる．この結果，後の還元が複雑になる．今回の実験では，入力単語に対して，それを右端に含む最大の構文木 (subtree) を出力するよう学習することを課題としているため，先読みしなければ還元に用いるべき規則が決定できないようなときには，先読みなしの場合には入力単語をそのまま出力するよう学習させることになる．

一方，先読みを許す場合（表 3.3）では，入力単語毎に動作が決定する．すなわち，単語とスタック上の値とを用いて還元して，新たな非終端記号をスタックにプッシュするか，還元操作が適用できない場合は，そのままシフトする．これにより，現在読んでいる単語に対して，最大の構文木は常にスタックのトップにあることになる．

では，本実験で提案するモデルでは，RAAM/SRN はどのように動作するのか．RAAM は，可変長の構文木を固定長で表現することを学習する．RAAM の行なうことは，構文解析過程では構文木を作ることに相当する．例えば R1 を NP(N) の RAAM 表現だとすれば，RAAM は，VP(Vi R1) という構文木の RAAM 表現を，VP, Vi, R1 という要素の入力から作ることになる．

SRN の行なうことは，入力単語を右端の葉とする最大の構文木の RAAM 表現を出力することである．それは，構文解析過程では，単語を読み，還元かシフトかを決定し，その操作を行ない，その結果としてスタックのトップにある構文木を出力する，という操作に相当する．

本実験では，先読みをしない場合と，する場合とで実験を行なう．また，RAAM および SRN は，使用する文は同じだが，個別に学習を行なう．

表 3.2: 先読みをしない場合の，スタックを使った構文解析と，それに対応する RAAM/SRN 処理過程の例（英語風）．RAAM 欄: VP=VtR1→R2 は，VtR1 を NP に還元し，RAAM が作るその構文木の RAAM 表現を R2 と呼ぶことを意味する．SRN 欄: Vi → R4 は，SRN に Vi を入力し，その出力ターゲットが R4 であることを意味する．stack 欄: スタックには例文の括弧，非終端記号，終端記号がプッシュされている．記号列の左端がボトム，右端がトップを表す．

例文:　S(NP(NwhoVP(VtNP(N)))VP(Vi).)

input	action	stack	RAAM	SRN
N	shift	S(NP(N		N → N
who	shift	S(NP(Nwho		who → who
Vt	shift	S(NP(NwhoVP(Vt		Vt → Vt
N	shift	S(NP(NwhoVP(VtNP(N		N → N
Vi	reduce	S(NP(NwhoVP(VtNP(N)	NP=N→R1	
	shift	S(NP(NwhoVP(VtR1		
	reduce	S(NP(NwhoVP(VtR1)	VP=VtR1→R2	
	shift	S(NP(NwhoR2		
	reduce	S(NP(NwhoR2)	NP=NwhoR2→R3	
	shift	S(R3		
	shift	S(R3VP(Vi		
	reduce	S(R3VP(Vi)	VP=Vi→ R4	
“.”	shift	S(R3R4		Vi → R4
	shift	S(R3R4.		
	reduce	S(R3R4.)	S=R3R4.→R5	
	shift	R5		“.” → R5

表 3.3: 先読みをする場合の，スタックを使った構文解析と，それに対応する RAAM/SRN 処理過程の例（英語風）．lookahead 欄は先読みする単語を意味する．“NIL” はダミー表現である．SRN 欄では，例えば “N, who → N” は，現在読んでいる単語 N と，先読み単語 who とを入力とし，N を出力対象とすることを意味する．その他の諸説明は表 3.2 を参照のこと．

例文:　S(NP(NwhoVP(VtNP(N)))VP(Vi).)

input	lookahead	action	stack	RAAM	SRN
N	who	shift	S(NP(N		N,who → N
who	Vt	shift	S(NP(Nwho		who,Vt → who
Vt	N	shift	S(NP(NwhoVP(Vt		Vt,N → Vt
N	Vi	reduce	S(NP(NwhoVP(VtNP(N)	NP=N → R1	
		shift	S(NP(NwhoVP(VtR1		N,Vi → R1
		reduce	S(NP(NwhoVP(VtR1)	VP=VtR1→R2	
		shift	S(NP(NwhoR2		
		reduce	S(NP(NwhoR2)	NP=NwhoR2→R3	
		shift	S(R3		
Vi	“.”	reduce	S(R3VP(Vi)	VP=Vi→ R4	
		shift	S(R3R4		Vi, “.” → R4
“.”	NIL	reduce	S(R3R4.)	S=R3R4.→R5	
		shift	R5		“.”,NIL → R5

3.4 ネットワーク・アーキテクチャ

3.4.1 記号の表現形態

ネットワークで使う記号の表現形態には，localist 表現を用いる（表 3.4）．すなわち，使用する記号数に応じた ID 部と，終端記号／非終端記号識別フラグ部，冗長ユニット部の三つの部分からなるビット列を用い，ID 部で各記号に応じたビットをひとつだけ 1 にして，他ではすべて -1 にする（本実験では，あとで述べるように，シグモイド関数として区間 $(-1, 1)$ を値域とする bipolar 型[6]を採用しているため，$1, -1$ の表現を用いることになる．表では見やすさのため -1 を 0 としている）．終端記号／非終端記号識別フラグ部は，記号が非終端記号なら 1，そうでなければ -1 とする．これらの表現により，終端記号，非終端記号の区別以外は，記号間の関係を示唆するような情報は表現には盛り込まれず，表現形態による恣意性を極力排除する形になっている．

表 3.4: 記号の表現形態．実験では bipolar 型の sigmoid 関数を用いているため，ここで示した表現において，0 は実際には -1 としている．

i) 言語 $\{a^n b^n\}$

記号	ID 部	フラグ部	冗長部
S	1 0 0	1	0 0
a	0 1 0	0	0 0
b	0 0 1	0	0 0
NIL	0 0 0	0	0 0

ii) 英語風言語

記号	ID 部	フラグ部	冗長部
S	1 0 0 0 0 0 0 0	1	0 0
NP	0 1 0 0 0 0 0 0	1	0 0
VP	0 0 1 0 0 0 0 0	1	0 0
N	0 0 0 1 0 0 0 0	0	0 0
Vi	0 0 0 0 1 0 0 0	0	0 0
Vt	0 0 0 0 0 1 0 0	0	0 0
who	0 0 0 0 0 0 1 0	0	0 0
“.”	0 0 0 0 0 0 0 1	0	0 0
NIL	0 0 0 0 0 0 0 0	0	0 0

iii) 日本語風言語

記号	ID 部	フラグ部	冗長部
S	1 0 0 0 0 0 0	1	0 0
NP	0 1 0 0 0 0 0	1	0 0
VP	0 0 1 0 0 0 0	1	0 0
N	0 0 0 1 0 0 0	0	0 0
Vi	0 0 0 0 1 0 0	0	0 0
Vt	0 0 0 0 0 1 0	0	0 0
“.”	0 0 0 0 0 0 1	0	0 0
NIL	0 0 0 0 0 0 0	0	0 0

RAAM/SRN では，終端記号，非終端記号，および構文木の RAAM 表現を，いずれも同じユニット数で表現しなければならない．ユニット数が多ければそれだけ多くの情報を表現できるので，複雑な構文木の表現に対応するために，記号表現に冗長ユニット部を導入する．これは記号表現の上では何もしないが，構文木の表現の際に使われる．また，これらのユニットは，記号と構文木 RAAM 表現との識別にも役立つことになる．

各文法で使用している記号の他に，各言語とも NIL 表現を導入する．これは，記号でも RAAM 表現でもなく，何もないことを示す表現である．すべてのビットで -1 とする．

[6] $f(x) = (1 - e^{-x})/(1 + e^{-x})$.

3.4.2 RAAM ネットワーク・アーキテクチャ

RAAM は $km-m-km$ 型のアーキテクチャ[7]を持つフィード・フォワード型ネットワーク[8]で，入力値と同じ値を出力の目標値とする自己連想を学習することにより，隠れ層に入力値の圧縮分散表現を得る（図 3.4）．本実験に則していえば，m はひとつの記号を表現するのに使う素子の数で，k は構文木を作る際に同時に使いうる要素（記号・非記号）の最大数である．例えば英語風文法の場合では，ひとつの構文木を作る際に最大で NP(N who NP Vt) というような構造に対応しなければならない．つまり，生成規則のうちでもっとも多い右辺記号数を持つ規則に対応しなければならない．左辺の非終端記号も含めて，この場合は $k=5$ が必要になる．RAAM の入力層は，m 個のユニットをもつ k 個の部分（これをスロットと呼ぶ）に分かれており，スロットひとつに，単語または構文木の RAAM 表現がひとつ，入力される．

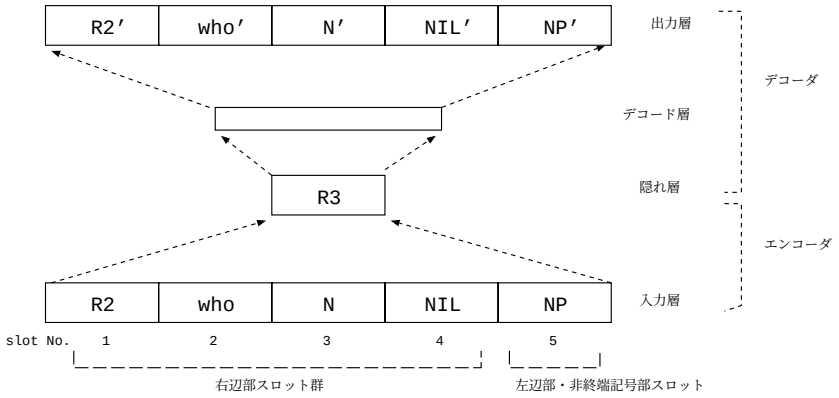


図 3.4: RAAM による NP(N who VP(Vt NP(N))) 構文木の圧縮分散表現の作成例，4 層型の場合．3 層型ではデコード層を省く．R2 は VP(Vt NP(N)) の RAAM 表現とする．

スロットを記号で埋めていく手順を，図 3.4 の例で NP(N who VP(Vt NP(N))) という名詞句の構文木を作ることで考えよう．入力層のスロットを左から順にスロット 1 〜 スロット k とする．VP(VtNP(N)) に相当する構文木（サブツリー）に対応する RAAM 表現が作られているとして，それをここでは R2 と表記する．すると，目的の構文木は NP(N who R2) となる．本実験の RAAM では，これを入力値とするときに，この構文木シーケンスを右側からひとつづつ読む．右辺部スロットである スロット 1 から順に，読んだ表現で埋めていく．左括弧まで読んだら右辺部スロットを埋める処理を一時中断して，左括弧の次の非終端記号を読み，左辺・非終端記号部であるスロット 5 に挿入する．最後に，残っていた右辺部スロット（いまの場合はスロット 4）を NIL で埋める．

これらをフィード・フォワードして隠れ層に得られる表現（ここでは R3）が，
NP(N who VP(Vt NP(N))) の RAAM 表現である．

Pollack が提案したオリジナルの RAAM ネットワークは 3 層のフィード・フォワード型ネットワークだが，本実験では，ネットワークの表現力を上げるために，デコーダ部に層

[7]本稿では，ネットワークのアーキテクチャとは，素子間の結合形態およびその結合荷重値（重み）を指す．
[8]実験で用いた RAAM および後に詳述する SRN では，層を超えての結合はなく，次の層との結合は完全結合である．

を 1 つ増やした 4 層型のフィード・フォワード型ネットワークも用いた．

3.4.3 RAAM の学習

RAAM の学習アルゴリズムには誤差逆伝播法（backpropagation, BP）を用いる．シグモイド関数には bipolar 型（値域 $(-1, 1)$）を用いる．文中の単語を前から順にひとつづつ読み，還元すべき箇所で前述のように各記号および RAAM 表現をスロットにセットして，処理を行なう（このことを，以降では “RAAM 処理” と呼ぶ）．自己連想学習であるから，ターゲットは入力パターンと同一である．

一回の試行では，指定長さ以下の文をすべて用いる．ネットワークに対する文の提示順はランダムである．重みの初期値は $[-1.0, 1.0]$ のランダムな値である．各出力ユニットにおける誤差が，あらかじめ指定する許容ユニット誤差範囲内の場合には，そのユニットについては誤差逆伝播を行なわない．学習対象となるすべての文のすべての RAAM プロセスにおいて，すべてのユニットが許容ユニット誤差範囲内になったら収束したとして学習を一旦中断し，出力層のパターンを RAAM デコーダ部でデコードして，構文木のシーケンス表現が復元できるかどうかテストを行なう（その方法は次節で述べる）．すべての文のすべての RAAM 処理において正確な構文木のシーケンス表現を復元することができれば，そこで学習を成功とし，テスト・セットによる汎化能力の評価をする．構文木が復元できなれければ，許容ユニット誤差を小さくして，さらに学習を続ける．許容ユニット誤差をだんだん小さくしながら学習を進めていくことを，ここではアニーリング学習と呼ぶことにする．

許容ユニット誤差は，学習する言語や文長さにより $0.3 \sim 0.5$ から始める．学習係数は 0.1 で始めて，指定エポック内（$3000 \sim 5000$ epochs．エポックとは，ここでは学習対象の文の総数と同数の文を処理することで 1 エポックと呼ぶ．つまり，32 の文が学習対象であれば，32 文を処理すれば，1 エポックである．文の提示順はランダムなので，1 エポック内で学習対象の全種類の文が処理されるとは限らない）に収束しなければ，以降では学習係数を 0.05 に切替える．慣性項は用いない．アニーリング学習での許容ユニット誤差減少ステップは 0.01 とする．学習係数が 0.05 でも指定エポック内で収束しなかった場合，および許容ユニット誤差が 0.01 で収束しても正確なデコードができなかった場合は，学習は失敗としてその試行を終了する．

3.4.4 RAAM 表現のデコード処理

RAAM によって作られる構文木の RAAM 表現自体には，記号は明示的に表現されていない．この RAAM 表現から，構文木のシーケンス表現を復元するには，学習した RAAM のデコーダ部を使って，順次 RAAM 表現をデコードし，記号表現を明示的に取り出す必要がある．すなわち，RAAM の隠れ層に構文木の RAAM 表現をセットし，フィード・フォワードすることにより出力層に得られたものをデコード結果とする．

図 3.4 における例で言えば，R3 という RAAM 表現は，デコーダ部でデコードされてはじめて NP(N who R2) という構文木シーケンス表現に復元できる．しかし，R3 を一度デコードしただけでは，R2 という RAAM 表現がまだ残っている．このサブツリーを復元

するために，さらに R2 をデコーダ部によってデコードする．R2 は VP(Vt R1) という構文木シーケンス表現に復元され，先ほどの NP(N who R2) と合わせて，NP(N who VP(Vt R1)) という構文木シーケンス表現が得られる．R1 は RAAM 表現なので，さらにデコード処理を適用して，NP(N) を得る．この中にはこれ以上デコードすべき RAAM 表現がないので，ここでデコードを終了する．こうして R3 の最終的な構文木シーケンス表現 NP(NwhoVP(VtNP(N))) が得られる．もしさらに RAAM 表現があれば，RAAM 表現が出現しなくなるまでデコードを繰り返す．指定回数以上デコードしても構文木シーケンスが確定しない場合，および出力層の左辺・非終端記号部に以下の基準で判別不能な表現をしている場合は，デコード失敗として処理を終了する．

デコードの際は，RAAM 隠れ層にセットする表現およびその出力結果である出力層の各スロットにおける表現が，記号なのか RAAM 表現なのかあるいは NIL なのか，記号であればどの記号なのか，を判別しなければならない．RAAM 出力層のうち，右辺部スロット群には，終端記号と RAAM 表現および NIL 表現が出現するので，その判別は次のような基準で行なう（括弧内は $\{a^n b^n\}$ 処理の場合）．

- まず RAAM 表現かどうかを見分ける．次の場合は RAAM 表現とみなす
  - 0.1 (-0.5) 以上の出力が 2 ユニット以上
  - または，ID 部の終端記号以外のユニットが 0.1 (0.0) 以上
  - （または，フラグ部が -0.5 以上）
- つぎに，終端記号を見分ける
  - RAAM 表現ではないこと
  - 最大出力ユニットの値が正で，そのユニットが終端記号 ID のいずれかであること．その最大出力ユニットをして当該記号とする
- RAAM 表現，終端記号のいずれでもなければ NIL とみなす

一方，左辺・非終端記号部スロットには，非終端記号以外は現れないはずなので，その判別は次のようになる．

- ID 部の非終端記号部の最大出力ユニットの値が正で，識別フラグ以外のユニットが正でないこと．その最大出力ユニットをして当該記号とする

3.4.5 SRN アーキテクチャ

SRN は構造としてはふつうのフィード・フォワード型ネットワークである（図 3.5）が，前回の隠れ層の活性パターンが，そのまま次回入力の文脈部として入力されるフィード・バック機能をもつ．これによりネットワークは，前回の状態と今回の入力とから出力を決めること，つまり時系列処理を行なうことができる．本実験でいえば，入力が文中の現在読んでいる単語で，文脈部は前回の隠れ層の活性パターンである．先読みをする場合には，文中の現在読んでいる単語の次の単語を，先読み部に入れる．

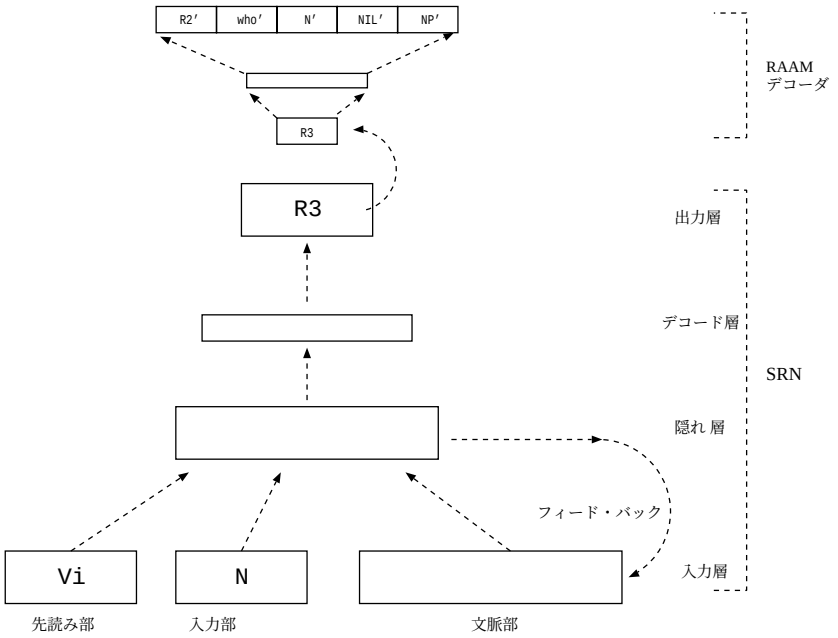


図 3.5: 先読みをする 4 層型 SRN の構造．例では N を入力として，対応する構文木 NP(NwhoVP(VtNP(N))) の RAAM 表現 R3 を出力することを示している．R3 は RAAM デコード 部を使ってデコード され，構文木シーケンスの表現に復元される．先読みをしない場合は先読み部は省く．また，3 層型の場合はデコード 層を省く．

SRN に提示するターゲットは，入力単語に対応する構文木の RAAM 表現を，あらかじめ学習済みの RAAM のエンコード部を使って生成する．入力単語に対するターゲットが構文木ではなく入力単語自身の場合もあり，その際はその単語表現をそのままターゲットとする．

なお，RAAM と同じく SRN でも，通常の 3 層型に加えて，隠れ層と出力層との間にデコード層を設け，4 層型としたものも用いて実験を行なった．

3.4.6 SRN の学習

諸条件は RAAM の場合とほぼ同じである．さらに，文脈部のすべてのユニットは各文の処理開始時に -1 に初期化される．先読みをする場合は，最後の単語の入力の際に先読み部には NIL が入る．出力対象は構文木の RAAM 表現なので，これを構文木シーケンスに復元するには RAAM と同じデコード処理をする必要がある．

3.5 ネットワークの汎化能力の評価

アニーリング学習が終了するまでに学習に用いた文（サンプル文セット）の学習に成功していれば，テスト文セットによる汎化能力の評価を行なう．テスト文セットは，各言語とも，サンプル文セットよりも長い文からなる．過学習を回避するために，一回の試行において学習が終了するまでに何度か汎化能力評価を行ない，もっとも汎化能力の高い時点をしてその試行における最適解とする．

評価は，正確に構文木を復元できたテスト文の数による．文の構文木を正確に復元するとは，文の各単語の入力に対して，途中経過の構文木を含めて，すべてのプロセスで正しい構文木を出力した場合を言う．

第4章　計算機実験結果

4.1　$a^n b^n$ 文法の学習

学習対象とした 3 つの文法（図 3.1 参照）のうち，まず，もっとも単純な $a^n b^n$ 文法の RAAM/SRN 学習実験を行なう．

4.1.1　RAAM 学習

学習に使う文（サンプル文）の長さ (l) 別に，各種ネットワーク・アーキテクチャで実験を行なった．結果を表 4.1 に示す．実験は $n \leq 2$（$l \leq 4$）の文を使った場合から，$n \leq 10$（$l \leq 20$）の文を使った場合までのうち，いくつかの場合について行なった．

$n \leq 6$ から $n \leq 10$ の文で学習した場合は，80 % 前後の率でサンプル文の学習に成功した．学習した文長さの倍以上の長さの文についても正確に構文木の RAAM 表現を生成する（RAAM 処理する) RAAM が見つかった．しかし，どの長さの場合でも，そのような高い汎化能力を学習で獲得する率は 50 回の試行[1]中 1 ～ 2 回程度である．

$n \leq 2$ の文で学習した場合は，サンプル文の学習には 100 % 成功するが，それ以上の長さの文を RAAM 処理するネットワークは得られなかった．

これらの試行で得られた，高い汎化能力の獲得に成功した RAAM の主な例を表 4.2 に示す．これらを，以降の SRN 学習に使う．

4.1.2　SRN 学習

表 4.2 に示した高い汎化能力をもつ RAAM を使って，SRN の学習実験を行なった．SRN は，使用する RAAM が学習に使ったものと同じサンプル文を使って学習する．$\{a^n b^n\}$ の構文解析に先読みは必要ないので，先読みをしない SRN を使う．その実験結果を表 4.3 に示す．

学習に使う文が長くなれば成功率は下がるものの，サンプル文の学習には 80% 前後で成功する．これは，正確な構文木シーケンス表現を出力することに成功する率であり，ネットワークが学習で収束するかしないかでいえば，ユニット毎平均誤差 0.1 程度でほぼ 100 % 収束する．ただ，SRN が出力した RAAM 表現の正確なデコードができない場合があるので，構文木出力の成功率が下がっている．

SRN が，学習した文長さよりも 2 倍を超えて長いテスト文に対して正しい構文木を出力する汎化能力を獲得する率は，$n \leq 10$ のどの長さまでで学習しても 5% 前後である．高い汎化能力を獲得した SRN のうちの主なものを 4.4 に示す．

[1] 1 回のアニーリング学習を 1 試行と呼ぶ.

表 4.1: $\{a^n b^n\}$ の RAAM 学習結果．学習長さは，学習に使った文の長さ n の上限値．最大汎化長さは，そのアーキテクチャのネットワークを使った試行のうち，最も長い文の RAAM 表現を作ることに成功したときの，その長さである．アーキテクチャは，ここではネットワークの層毎の素子数で表現されている．左から右に，入力層 – 隠れ層 – デコード層 – 出力層の素子数を示す．3 層型の場合にはデコード層を省く．倍汎化成功率は，学習に使った文の長さの，2 倍を超えて長いテスト文の構文木 RAAM 表現の生成（RAAM 処理）に成功した試行の回数を，全試行回数で割ったものである．学習成功率は，汎化能力を獲得したものも含めて，サンプル文セットの正確な RAAM 処理に成功した試行の回数を，全試行回数で割ったものである．これらは，以降の表でも同様である．

i) $n \leq 3$ の文で学習した場合．

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	倍汎化成功率	学習成功率
3	14	$24-6-24$	1/50	46/50
3	4	$24-6-8-24$	0/50	47/50
3	4	$24-6-12-24$	0/50	42/50
3	12	$24-6-15-24$	1/50	49/50
3	2	$24-6-18-24$	0/50	48/50

ii) $n \leq 5$ の文で学習した場合．

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	倍汎化成功率	学習成功率
5	12	$24-6-24$	1/50	45/50
5	14	$24-6-5-24$	1/50	45/50
5	12	$24-6-10-24$	1/50	47/50
5	16	$24-6-15-24$	3/50	48/50
5	8	$24-6-20-24$	0/50	47/50

iii) $n \leq 8$ の文で学習した場合．

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	倍汎化成功率	学習成功率
8	14	$24-6-24$	0/50	42/50
8	10	$24-6-5-24$	0/50	38/50
8	12	$24-6-10-24$	0/50	46/50
8	25	$24-6-15-24$	1/50	47/50

iv) $n \leq 10$ の文で学習した場合．

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	倍汎化成功率	学習成功率
10	13	$24-6-24$	0/50	37/50
10	22	$24-6-5-24$	1/50	34/50
10	16	$24-6-10-24$	0/50	43/50
10	27	$24-6-15-24$	2/50	42/50
10	16	$24-6-20-24$	0/50	39/50

表 4.2: $\{a^n b^n\}$ の汎化に成功した主な RAAM ネットワーク例．学習平均誤差は，ネットワークが最大の汎化能力を示したときの，サンプル文セットに対するユニット毎の平均誤差．重み更新回数は，ネットワークが最大の汎化能力を示すまでのもの．これらは，以降の表でも同様である．

No.	学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	学習平均誤差	学習文平均長さ (l)	重み更新回数
AR3-1	3	9	$24-6-15-24$	0.0892	3.5	702
AR5-1	5	16	$24-6-15-24$	0.0369	5.8	9521
AR8-1	8	25	$24-6-15-24$	0.0312	8.0	6526
AR10-1	10	28	$24-6-15-24$	0.0486	10.3	7298

表 4.3: $\{a^n b^n\}$ の SRN 学習結果

i) $n \leq 3$ の文で 学習した場合.

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	RAAM 重み	倍汎化成功率	学習成功率
3	4	$14-8-6$	AR3-1	2/50	50/50
3	5	$14-8-8-6$	AR3-1	5/50	50/50
3	8	$14-8-10-6$	AR3-1	5/50	50/50
3	5	$14-8-12-6$	AR3-1	5/50	50/50
3	6	$14-8-16-6$	AR3-1	5/50	50/50
3	7	$14-2-18-6$	AR3-1	2/50	50/50

ii) $n \leq 5$ の文で 学習した場合.

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	RAAM 重み	倍汎化成功率	学習成功率
5	13	$10-4-6$	AR5-1	1/50	50/50
5	12	$8-2-8-6$	AR5-1	2/50	43/50
5	11	$8-2-10-6$	AR5-1	2/50	46/50
5	12	$8-2-12-6$	AR5-1	6/50	46/50
5	9	$8-8-8-6$	AR5-1	0/50	50/50
5	10	$8-3-8-6$	AR5-1	1/50	50/50

ii) $n \leq 10$ の文で 学習した場合.

学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	RAAM 重み	倍汎化成功率	学習成功率
10	22	$14-8-6$	AR10-1	5/50	50/50
10	25	$14-8-8-6$	AR10-1	4/50	50/50
10	25	$14-8-12-6$	AR10-1	6/50	50/50
10	25	$14-2-9-6$	AR10-1	3/50	39/50
10	25	$14-2-12-6$	AR10-1	4/50	39/50

表 4.4: $\{a^n b^n\}$ の汎化処理に成功した主な SRN

No.	学習長さ (n)	最大汎化長さ (n)	アーキテクチャ	学習平均誤差	学習文平均長さ (l)	重み更新回数
AS3-1	3	8	$14-8-10-6$	0.0914	3.7	1029
AS5-1	5	13	$10-4-6$	0.0326	5.9	20925
AS10-1	10	25	$14-2-12-6$	0.0238	10.9	88796

4.1.3 SRN 隠れユニットの挙動

表 4.4 の AS10-1 は，$n \leq 10$ の文で学習して $n \leq 25$ の文の正確な構文解析に成功した．このネットワークは隠れユニットが 2 つであり，その活性パターン変化を平面上にプロットして観察することができる．SRN に $n = 25$ の文を処理させたときの隠れユニットの活性の様子を図 4.1 に示す．

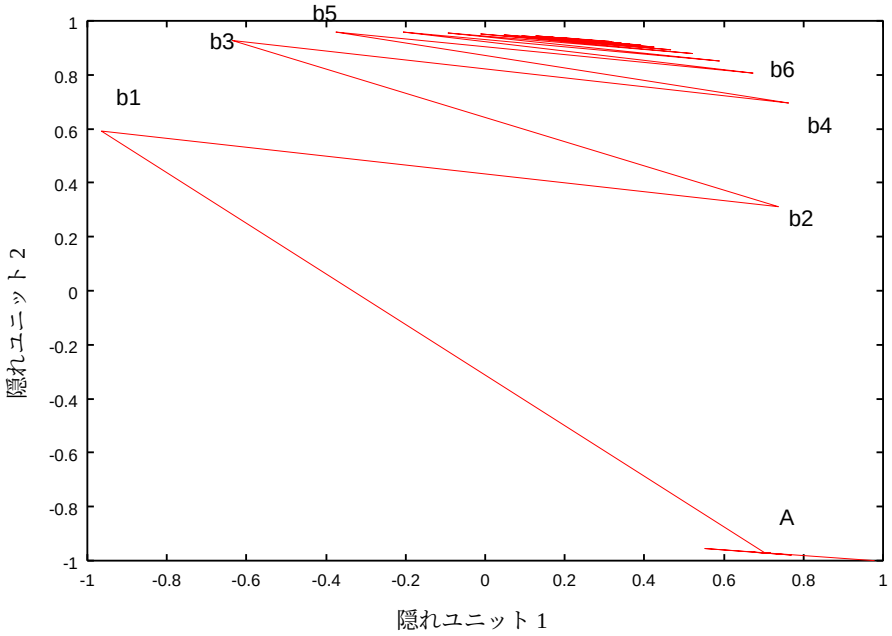


図 4.1: SRN 隠れユニットのダイナミクスの例．$n = 25$ の文を入力したとき．a の入力が続く間は，図中の A 付近で振動しながら $(0.70, -0.97)$ 付近に収束する．b の入力に切り替わると，図中 b1，b2，…と変遷しながら，やはりある収束点 $(0.25, 0.93)$ 付近に向かう．

隠れユニットは，a が入力し続けている間は $(0.70, -0.97)$ 付近で細かく振動しながら一点に収束するような動きをしている．a から b に入力が切り替わると，ユニットの活性は急激に変化し，$(-0.96, 0.59)$ 付近にジャンプする．そこからは b の入力のたびに大きく振動しながら，やはりある収束点 $(0.25, 0.93)$ 付近に向かう．

4.2 英語風文法の学習

前節の $a^n b^n$ 学習実験で，RAAM/SRN の基本的な学習能力が確認された．これを受け，文法を自然言語風のものに拡張して実験を続ける．まず，英語風文法の学習実験を行なう．

4.2.1 RAAM 学習

学習に使う文の長さ (l) 別に，各種ネットワーク・アーキテクチャで実験を行なった．結果を表 4.5 に示す．$l \leq 7$ または $l \leq 8$ の文で学習した場合は，サンプル文セットの学習には高い率で成功し，学習に成功すれば汎化能力をもつ傾向がある．なお，以降の自然言語風言語の学習実験では，汎化能力のテストには，RAAM および SRN の双方について，ネットワークが学習に使った文よりも 1 ないし 2 だけ長い文をテスト文セットとして使う．

$l \leq 10$ の文および $l \leq 11$ の文それぞれの学習の場合でも，学習に成功すれば汎化能力をもつ傾向があるが，学習に成功する率は低くなる．

$l \leq 9$ の文での学習の場合，および $l \leq 12$ の文での学習の場合では，RAAM の学習は成功しなかった．とくに $l \leq 9$ の文での学習の場合は，サンプル文セットのうちひとつだけ学習できないという事例が多かった．

ネットワークのアーキテクチャによる学習能力の違いを見てみると，3 層型よりは 4 層型のほうが，4 層型でもデコード層のユニット数の多いほうが，サンプル文の学習成功率は高い．しかし $l \leq 7$ および $l \leq 8$ の文での学習の場合については，最大の汎化能力を示したネットワークはいずれも 3 層型であった．サンプル文の最長長さが $l \geq 10$ の場合は，3 層型ではサンプル文の学習さえ困難であり，4 層型でなければ学習できなかった．

これらの試行で得られた，比較的高い汎化能力の獲得に成功した RAAM の主な例を表 4.6 に示す．これらを，以降の SRN の学習に使う．

表 4.6 中の ER7-2 および ER10-1 は，長さだけでなく，埋め込み深さについても RAAM 処理の汎化に成功している．

$l \leq 7$ の文では，名詞句の埋め込み深さは最深で 2 であるが，ER7-2 は，埋め込み深さが 3 である次のふたつの文を正しく RAAM 処理することに成功している．

e19[2]: S(NP(NwhoVP(VtNP(NwhoNP(N)Vt)))VP(Vi).)
e28: S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt))))).)

ER10-1 は $l \leq 10$ の文で学習しているが，このサンプル文での名詞句の埋め込み深さは最深で 3 である．この ER-10 が汎化処理に成功している長さ 11 ～ 12 の文のうち，次の文は最深の埋め込み深さが 4 である．

e222: S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))))).)

[2]文番号．以降も同じ．詳しくは文一覧 A を参照のこと．

表 4.5: 英語風言語の RAAM 学習結果．汎化文数は，学習に使った文より長さ 1 ないし 2 だけ長い文でテストをしたときに，文のすべてのプロセスで正しい構文木を出力する率を示す.（正解文数）／（総テスト文数）である．以降の表でも同様である．

i) $l \leq 7$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
7	11	8/24	$55-11-55$	38/50	41/50
7	11	5/24	$55-11-12-55$	37/50	43/50
7	11	6/24	$55-11-16-55$	37/50	47/50
7	11	7/24	$55-11-24-55$	44/50	46/50
7	11	6/24	$55-11-32-55$	38/50	45/50

ii) $l \leq 8$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
8	26	8/20	$55-11-55$	7/30	7/30
8	26	3/20	$55-11-12-55$	13/30	14/30
8	26	7/20	$55-11-16-55$	13/30	16/30
8	26	7/20	$55-11-24-55$	15/30	18/30
8	26	7/20	$55-11-32-55$	25/30	25/30

iii) $l \leq 10$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
10	46	–	$55-11-55$	0/30	0/30
10	46	19/64	$55-11-24-55$	11/30	11/30
10	46	15/64	$55-11-32-55$	17/30	17/30
10	46	17/64	$55-11-40-55$	14/30	14/30
10	46	14/64	$55-11-48-55$	13/30	13/30

iv) $l \leq 11$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
11	78	–	$55-11-55$	0/30	0/30
11	78	–	$55-11-24-55$	0/30	0/30
11	78	15/52	$55-11-32-55$	4/30	4/30
11	78	17/52	$55-11-40-55$	5/30	5/30
11	78	14/52	$55-11-48-55$	4/30	4/30

表 4.6: 英語風言語学習で汎化に成功した主な RAAM ネットワーク例．汎化プロセス正解率は，文途中も含めて，構文木表現の生成を 1 プロセスとしたときに，全テスト文のプロセス総計に対して，ネットワークが正しい構文木を出力した割合である．これは，以降の表でも同様である．

No.	学習長さ	汎化成功率	アーキテクチャ	汎化プロセス正解率	学習平均誤差	学習文平均長さ	重み更新回数
ER7-1	7	8/24	$55-11-55$	130/160	0.0312	5.3	56802
ER7-2	7	7/24	$55-11-24-55$	125/160	0.0450	5.2	24625
ER8-1	8	8/20	$55-11-55$	122/142	0.0304	6.5	125268
ER10-1	10	19/64	$55-11-24-55$	448/524	0.0288	7.8	71230

4.2.2 SRN 学習

表 4.6 に示した，比較的高い汎化能力をもつ RAAM を使って，SRN の学習実験を行なった．SRN は，使用する RAAM が学習に使ったものと同じサンプル文を使って学習する．

まず，$l \leq 7$ の文で先読みをしないで学習して，$l \leq 8$ および $l \leq 9$ の文についての汎化能力をみる実験を行なった．その結果を表 4.7 に示す．この実験により，$l \leq 7$ の文での学習では，隠れユニットを多くすれば，サンプル文の学習はほぼ 100% 可能であることがわかった．しかし，より長い文の正確な構文木出力をする汎化能力を示すものはほとんどなく，あってもわずか 1 文が正しく処理できるにとどまった．また，$l \leq 8$ の文で学習した場合には，サンプル文の学習さえ困難だった．

表 4.7: 英語風言語の $l \leq 7$ の文による SRN 学習，先読みしない場合．

i) 隠れユニット数が 12 の場合

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
7	14	0/22	$23-12-11$	ER7-2	1/50
7	14	–	$23-12-6-11$	ER7-2	0/50
7	14	0/22	$23-12-8-11$	ER7-2	15/50
7	14	0/22	$23-12-10-11$	ER7-2	24/50
7	14	0/22	$23-12-12-11$	ER7-2	31/50

ii) 隠れユニット数が 16 の場合

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
7	14	0/22	$27-16-8-11$	ER7-2	45/50
7	14	1/22	$27-16-10-11$	ER7-2	47/50
7	14	1/22	$27-16-12-11$	ER7-2	49/50
7	14	1/22	$27-16-14-11$	ER7-2	49/50
7	14	1/22	$27-16-16-11$	ER7-2	48/50

先読みをしない学習では $l \leq 8$ の文でさえ学習できないので，次に，先読みを許す形で実験を行なった．その結果を表 4.8 に示す．先読みをすれば，$l \leq 10$ の文による学習の場

合でも，SRN の学習が可能である．ただし，$l \leq 7$ の文での学習では汎化能力をもつネットワークが見つからなかった．$l \leq 9$ の文での学習では，1 文を汎化処理するネットワークがひとつ見つかったのみである．$l \leq 10$ の文での学習でもサンプル文の学習は可能だったが，汎化能力をもつネットワークは見つかっていない．

$l \leq 8$ の文での学習の場合について，$l \leq 4$ の文の割合をそれまでの 0.115 から 0.3 に増やして学習させる措置をとってみたが，学習成績に大きな違いはなく，汎化能力をもつネットワークも得られなかった．

表 4.8: 英語風言語の SRN 学習，先読みする場合．

i) $l \leq 7$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
7	14	0/24	$46-24-8-11$	ER7-1	5/50
7	14	0/24	$46-24-12-11$	ER7-1	10/50
7	14	0/24	$46-24-16-11$	ER7-1	13/50
7	14	0/24	$46-24-20-11$	ER7-1	27/50
7	14	0/24	$46-24-24-11$	ER7-1	36/50

ii) $l \leq 8$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
8	26	1/20	$46-24-8-11$	ER8-1	7/50
8	26	0/20	$46-24-12-11$	ER8-1	2/50
8	26	0/20	$46-24-16-11$	ER8-1	3/50
8	26	0/20	$46-24-20-11$	ER8-1	10/50

iii) $l \leq 10$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
10	48	–	$46-24-28-11$	ER10-1	0/50
10	48	0/64	$54-32-28-11$	ER10-1	1/50
10	48	0/64	$70-48-32-11$	ER10-1	3/10
10	48	0/64	$70-48-40-11$	ER10-1	1/10
10	48	0/64	$70-48-48-11$	ER10-1	1/10

4.2.3 SRN 汎化成功事例の観察

表 4.7，表 4.8 に示した試行のうち，テスト文の構文解析に成功した SRN の主な例を表 4.9 に示す．

$l \leq 7$ の文で学習して，汎化に成功した SRN は表 4.9 に示した ES7-1 を含め全部で 7 例，$l \leq 8$ 以下の文の学習でのそれは ESL8-1 のみである．それらがテスト文に対して正しく構文木を出力することに成功した例は，次の 3 文に限られていた．

e32: S(NP(NwhoVP(Vi))VP(VtNP(NwhoVP(VtNP(N))))).

表 4.9: 英語風言語の汎化処理に成功した主な SRN 例.

No.	学習長さ	汎化文数	先読み	アーキテクチャ	汎化プロセス正解率	平均誤差	学習文平均長さ	重み更新回数
ES7-1	7	1/24	なし	$27-16-14-11$	189/228	0.1647	5.1	9594
ESL8-1	8	1/20	あり	$46-24-24-11$	175/208	0.0914	6.0	140880

e35: S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(Vi))).)
e70: S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(VtNP(N)))).)

汎化に成功した各ネットワークは，これらの文のうちどれか 1 文しか正確に構文木を出力できない．また，これらの文は名詞句の最深埋め込み深さが 2 であり，埋め込み深さ 3 の文に対して汎化能力をもつネットワークは見つかっていない．

サンプル文の学習に成功した SRN は，テスト文のほとんどすべての文について，一文を通して正しい構文木を出力することには失敗しているが，文を読んでいる途中の単語入力に対しては，多くの場合，正確な構文木を出力している．表 4.9 中のプロセス正解率は，そのような途中の出力も含む全出力プロセスで見た場合の正解率であり，これは 80% 以上である．

SRN の出力の様子を観察すると，どのテスト文の構文解析の際にも，文のはじめのほうの単語入力に対しては，ほぼ間違うことなく正確な構文木を出力している．間違った出力をするのは，多くの場合，文の後半部の名詞句の深い埋め込みに対する構文木を出力しようとするときである．例えば ESL8-1 で $l = 9$ の次の文を構文解析する例を見てみる．

```
e75: S(NP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))VP(VtNP(N)).)
                                        *1          *2
```

ネットワークは，*1 の N の入力まで，正しい構文木を出力している．*1 での正しい構文木は NP(NwhoVP(VtNP(NwhoVP(VtNP(N))))) である．*2 の N の入力でも正しい構文木 VP(VtNP(N)) を出力する．しかし，最後の単語であるピリオドが入力されたとき，ネットワークは

S(NP(NwhoVP(VtNP(N)))VP(VtNP(N)).)

という構文木を出力する．最後の入力で文であると認識し，最後の動詞句の構造も正しいが，途中で正しく処理をした深い埋め込みを持つ名詞句を，最後で正しく処理できていないのである．

4.3 日本語風文法の学習

続いて，日本語風の文法についての実験を行なう．

4.3.1 RAAM 学習

文の長さ (l) 別に，各種ネットワーク・アーキテクチャで学習実験を行なった．結果を表 4.10 に示す．全般的に学習に成功する率が高く，学習に成功すれば汎化能力ももつ，という傾向がある．

表 4.10: 日本語風言語の RAAM 学習結果．各項目の意味は，4.1 および 4.2 の各表を参照のこと．

i) $l \leq 7$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
7	21	8/15	$50-10-50$	18/50	19/50
7	21	5/15	$50-10-16-50$	36/50	39/50
7	21	6/15	$50-10-24-50$	34/50	36/50
7	21	7/15	$50-10-32-50$	34/50	38/50
7	21	6/15	$50-10-40-50$	35/50	37/50

ii) $l \leq 8$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
8	28	10/17	$50-10-50$	7/50	8/50
8	28	7/17	$50-10-16-50$	36/50	37/50
8	28	9/17	$50-10-24-50$	38/50	38/50
8	28	7/17	$50-10-32-50$	38/50	40/50
8	28	6/17	$50-10-40-50$	35/50	36/50

iii) $l \leq 9$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
9	36	8/19	$50-10-50$	27/50	27/50
9	36	10/19	$50-10-16-50$	34/50	34/50
9	36	13/19	$50-10-24-50$	35/50	36/50
9	36	9/19	$50-10-32-50$	27/50	27/50
9	36	10/19	$50-10-40-50$	30/50	32/50

iv) $l \leq 10$ の文による学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	汎化成功率	学習成功率
10	45	9/21	$50-10-50$	11/50	11/50
10	45	12/21	$50-10-16-50$	21/50	22/50

表 4.10 に示した結果のうち，高い汎化能力の獲得に成功した RAAM の主な例を表 4.11

に示す．これらを，以降の SRN の学習に使う．

表 4.11: 日本語風言語の汎化処理に成功した主な RAAM ネットワーク例．各項目の説明は表 4.6 を参照のこと．

No.	学習長さ	汎化成功率	アーキテクチャ	汎化プロセス正解率	学習平均誤差	学習文平均長さ	重み更新回数
JR7-1	7	8/15	$50-10-50$	83/94	0.0581	5.3	49605
JR8-1	8	10/17	$50-10-50$	102/115	0.0546	6.0	139055
JR9-1	9	13/19	$50-10-32-50$	129/138	0.0266	6.6	159114
JR10-1	10	12/21	$50-10-24-50$	150/163	0.0247	7.3	303801

これらのネットワークのうち，JR7-1 と JR9-1 とは，名詞句の埋め込み深さについてわずかながら汎化能力を示した．$l \leq 7$ の文では，名詞句の最深埋め込み深さは 3 であるが，JR7-1 は，次に示す深さ 4 の文の正確な RAAM 処理に成功している．

j19: S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)Vi).)
j35: S(VP(NP(N)NP(NP(NP(NP(N)VtN)VtN)VtN)Vt).)

$l \leq 9$ の文では，名詞句の最深埋め込み深さは 4 であるが，JR9-1 は，次に示す深さ 5 の文の RAAM 処理に成功している．

j34: S(VP(NP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)VtN)Vi).)

4.3.2 SRN 学習

前節に見た実験で得られた，高い汎化能力をもつ RAAM を使って，SRN の学習実験を行なった．SRN は，使用する RAAM が学習に使ったものと同じサンプル文を使って学習する．なお，日本語風言語では，先読みをしない学習の処理が技術的に困難だったため，先読みをする場合についてのみ実験を行なった．

まず，$l \leq 7$ の文を使って学習する実験を行なった．その結果を表 4.12 に示す．この実験では 3 層型および 4 層型で，隠れユニット数が 16 および 24 のときの各種ネットワーク・アーキテクチャでの学習を見ているが，SRN は比較的高い学習・汎化能力を示しているように見える．そこで，これと同様なネットワーク・アーキテクチャによる実験を，$l \leq 8$ の文での学習に拡張したところ，SRN はサンプル文の学習にさえ一度も成功しなかった．

そこで，隠れユニットを大幅に増やして，より長い文の学習実験を行なった．その結果を表 4.13 に示す．隠れユニット数が 16 または 24 では成功しなかった，$l \leq 8$ の文での学習が，隠れユニット数を大幅に増やすことで可能になったことがわかる．しかし，サンプル文の学習に成功した例はあるものの，$l \leq 8$ の文での学習で，より長い文に対して汎化能力を示した例は見つかっていない．$l \leq 9$ の文で学習した場合には，$l \geq 10$ の文を 1 文汎化処理した例が見つかっている．

表 4.12: 日本語風言語の SRN 学習結果 I, $l \leq 7$ の文を使う場合.

i) 隠れユニット数 16 の場合

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
7	21	0/15	$36-16-10$	JR7-1	2/50
7	21	0/15	$36-16-8-10$	JR7-1	4/50
7	21	2/15	$36-16-12-10$	JR7-1	4/50
7	21	1/15	$36-16-16-10$	JR7-1	16/50
7	21	2/15	$36-16-20-10$	JR7-1	12/50

ii) 隠れユニット数 24 の場合

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
7	21	2/15	$44-24-8-10$	JR7-1	8/50
7	21	3/15	$44-24-12-10$	JR7-1	18/50
7	21	2/15	$44-24-16-10$	JR7-1	31/50
7	21	2/15	$44-24-20-10$	JR7-1	38/50
7	21	4/15	$44-24-24-10$	JR7-1	37/50

表 4.13: 日本語風言語の SRN 学習結果 II

i) $l \leq 8$ の文での学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
8	28	0/17	$52-32-32-10$	JR8-1	1/10
8	28	0/17	$68-48-32-10$	JR8-1	1/10
8	28	0/17	$68-48-48-10$	JR8-1	4/10
8	28	0/17	$68-60-48-10$	JR8-1	7/10

ii) $l \leq 9$ の文での学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
9	36	0/19	$52-32-32-10$	JR9-1	9/10
9	36	1/19	$68-48-48-10$	JR9-1	8/10
9	36	1/19	$80-60-48-10$	JR9-1	1/10

iii) $l \leq 10$ の文での学習

学習長さ	学習文数	最大汎化成功率	アーキテクチャ	RAAM 重み	学習成功率
10	45	0/21	$52-32-32-10$	JR10-1	6/10
10	45	0/21	$68-48-48-10$	JR10-1	10/10
10	45	0/21	$80-60-48-10$	JR10-1	10/10

4.3.3 SRN 汎化成功事例の観察

表 4.12，表 4.13 に示した結果のうち，汎化能力を獲得することに成功した SRN の主な例を表 4.14 に示す.

表 4.14: 日本語風言語の汎化処理に成功した主な SRN 例.

No.	学習長さ	汎化文数	アーキテクチャ	汎化プロセス正解率	学習平均誤差	学習文平均長さ	重み更新回数
JS7-1	7	4/15	$44-24-24-10$	118/143	0.0463	5.3	75404
JS7-2	7	3/15	$44-24-24-10$	115/143	0.0711	5.3	23589
JS7-3	7	3/15	$44-24-24-10$	111/143	0.0720	5.3	31392
JS7-4	7	3/15	$44-24-12-10$	113/143	0.0498	5.3	91866
JS7-5	9	1/19	$68-48-48-10$	177/219	0.0515	6.6	170701
JS7-6	9	1/19	$68-48-48-10$	179/219	0.0628	6.6	59911
JS7-7	9	1/19	$80-60-48-10$	175/219	0.0533	6.6	305439

表 4.14 のうちの，JS7-1~4 で正しい構文木出力に成功したテスト文は，次の 5 つの文のいずれかに限られた.

j22: S(VP(NP(N)NP(NP(NP(ViN)VtN)VtN)Vt).)
j27: S(VP(NP(NP(NP(ViN)VtN)VtN)NP(N)Vt).)
j28: S(VP(NP(NP(NP(ViN)VtN)VtN)NP(ViN)Vt).)
j29: S(VP(NP(NP(ViN)VtN)NP(NP(N)VtN)Vt).)
j30: S(VP(NP(NP(ViN)VtN)NP(NP(ViN)VtN)Vt).)

$l \leq 7$ の文では，名詞句の最深埋め込み深さは 3 であり，上記の 5 つの文中にも深さ 4 以上の名詞句の埋め込みはない.

一方，JS7-5~7 では，各ネットワークは次のふたつのテスト文のうち，いずれかひとつのみ，正しい構文木の出力に成功した.

j36: S(VP(NP(N)NP(NP(NP(NP(ViN)VtN)VtN)VtN)Vt).)
j42: S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)NP(ViN)Vt).)

$l \leq 9$ の文では，名詞句の最深埋め込み深さは 4 である．j36，j42 の文もやはり最深の埋め込み深さは 4 である.

名詞句の埋め込み深さについて汎化能力を示した SRN の事例は，この実験では見つかっていない.

第5章　考察と議論

5.1　実験結果のまとめと考察

本実験では，三種類の CFG（$a^n b^n$，英語風，日本語風）について，提案するモデルが，文中の単語を前からひとつづつ順に読んでいった際に，その単語に応じた構文木を出力する能力，またその汎化能力を観てきた．ここではまず，実験結果を簡単にまとめて，考察する．

5.1.1　$a^n b^n$ 文法の学習について

$\{a^n b^n\}$ の学習では，$n \leq 5$ の文で学習して，$n \leq 16$ の文について途中経過も含めて正しい構文木を出力することが可能であることが確認された．さらに長い文，$n \leq 10$ の文で学習した場合には，最高で $n \leq 25$ の処理に成功している．短いほうでは，同じタスクについて，埋め込み深さが最深で 3 までしかない $n \leq 3$ の文で学習した場合にも，2 倍以上の長さの文について汎化処理することが可能だった．ただしこれらの学習では，汎化能力をもつ RAAM が 50 回の試行で 2 つ程度しかなく，その，高い汎化能力をもつ RAAM を使った SRN の学習でも，2 倍程度の長さの文について汎化能力をもつ SRN はやはり 50 試行で 2 つ程度しかないなど，汎化能力獲得の成功率は低い．

$\{a^n b^n\}$ の構文解析タスクでは，ネットワークは高い汎化能力を示した．しかし，ここで行なった実験手法では，SRN は $\{a^n b^n\}$ の処理の際に明示的に文の終りを認識している必要がない．$\{a^n b^n\}$ の処理の際には，入力された b に対する最大の構文木は常に文の形をしており，構文木だけでは文の途中なのかそれとも文の終りなのかを示すことはできない．文の終りを認識する必要がなければ，ネットワークは入力された b の数に応じて構文木を出力すればよい．a の数と b の数の一致をネットワークが認識しているかどうかは，このタスクでは明確にはならない．したがってこのタスクだけでは，ネットワークは構文解析はしているものの，カウンタ[1]能力を獲得したかどうかは不明確である．図 4.1 の隠れユニットのダイナミクスでは，a の入力時の挙動があまりに小さい．このことを考えると，$\{a^n b^n\}$ の構文解析に成功した SRN でも，a の数を数えていないことは十分に考えられる．この点を明確にするには，非文を入力した際にネットワークがどのような出力をするのか確認すること，$\{a^n b^n\}$ の処理の際には，構文木だけではなく，それが文の途中なのか終りなのかも出力させること，などの対策をとらなければならないが，それらは今後の課題である．

しかし，$\{a^n b^n\}$ の RAAM 学習実験では，RAAM が構文木の再帰的な埋め込みを学習し，その汎化能力も持ちうることを示した．SRN も，その構文木の RAAM 表現を出力す

[1]簡単に言ってしまえば，プッシュできる文字を一つに制限したスタックである．

るという汎化能力を示している．これは，SRN でも深い埋め込み構造をもつ RAAM 表現を取り扱いうることを示している．文の途中と終りとを明示的に区別しながら構文解析をするようネットワークをトレーニングすれば，汎化成績は今回の実験で示したよりは下がるであろうが，ネットワークは入力された a の数と b の数との一致を認識できると考えている．

RAAM の学習は収束が難しく，累積誤差が拡大するなどの欠点が指摘されている [2]. Reilly の実験 [16] でも，汎化能力が伸びなかったのは RAAM の汎化能力に原因があるのではないかとされた．しかし，本実験では，デコード層を増やした RAAM が $\{a^n b^n\}$ の埋め込み深さについて大きな汎化能力を示すことが確認されており，その際の収束も早い．RAAM は基本的な再帰的埋め込みの表現を学習することが可能であると言える．ただし，成功率は低いし，自然言語風言語の学習の際には，長い文の学習は困難であった．

5.1.2 自然言語風文法の学習について

英語風，日本語風の自然言語風言語の構文解析タスクは，$\{a^n b^n\}$ の場合とは違って，S 記号は文の終りに対してしか現れないので，明確に文の終りを認識しなければ構文解析ができない．したがって自然言語風言語の構文解析学習タスクでは，ネットワークはスタックを実現する能力の獲得を要求されていることになると考える．

スタック能力の獲得を要求される自然言語風の学習は，$\{a^n b^n\}$ に比べると格段に難しくなる．英語風言語の学習では，RAAM については長さ (l) が 11 までの文で学習して，$l \geq 12$ の 18 文について正確に RAAM 処理する汎化能力の獲得に成功したが，$l \leq 12$ の文で学習した場合は，サンプル文の学習に成功しなかった．汎化能力の獲得成功率は $l \leq 11$ の文の学習で約 10 % である．

SRN については，$l \leq 7$，および $l \leq 8$ の文で学習したそれぞれの SRN が，学習に使った文の長さより長い文をひとつだけ正確に構文解析することができた，という事例がいくつかある程度である．これら事例でも，汎化に成功した文は，埋め込み深さでは学習した文での深さを超えるものはなく，長さについての汎化はわずかならがしているものの，埋め込み深さについての汎化はしていない．$l \leq 9$，および $l \leq 10$ の文での学習では，サンプル文の学習は可能だったが，学習した長さ以上の文を正確に構文解析した例はなかった．

日本語風言語の学習では，$l \leq 7$ の文で学習した場合に，$l \geq 8$ の文を最高で 4 文，構文解析することに成功する汎化能力を示した．しかしそれ以上の長さの文で学習した場合には，$l \leq 9$ の文で学習したときに，$l = 10$ の文を 1 つだけ処理することに成功した事例があるのみである．

英語風，日本語風のいずれの言語の学習についても，ひとつのテスト文のすべての入力単語に対して正しい構文木の出力に成功した例は極めて少なかったが，文途中の入力単語に対する構文木の出力は概ね正しかった．テスト文の途中まではサンプル文と同じパターンがあるから，テスト文でもサンプル文と同じ長さのところまでは正しく出力しても当然に思われる．4.2.3 節で見た例はその典型で，ピリオドの手前までは正しい構文木を出力していた．しかし最後のピリオドの入力で，それまで正しく出力してきた構文木を一つにまとめて完全な文としての構文木を出力することができなかった．このことは非常に重要なことなので，5.2 節であらためて議論する．

再帰的規則の獲得と並んで本研究の重要な目的のひとつである右回帰的，左回帰的な双方の文法の学習は，提案するモデルが，両者ともほぼ同等の成績で学習可能だということができる．

5.1.3 結果の全体的傾向

ネットワーク・アーキテクチャは，3 層型と 4 層型について，さまざまな素子数で実験した．定性的な考察では，全般的に 3 層型よりは 4 層型のほうが，隠れユニットは少ないよりは多いほうが，学習能力が高い，という傾向が認められる．これは当然予想されたことである．さらに付け加えると，それぞれの文法の学習において，サンプル文が短いときは，学習能力は 4 層型のほうが高いものの，汎化能力は 3 層型のほうが高い傾向がある．この原因として，4 層型のほうが過学習に陥りやすいということが考えられる．しかし，サンプル文が長くなると 3 層型では学習困難となり，4 層型でないと学習できないケースが目立った．

今回の実験では，RAAM 表現のデコード基準 (3.4.4 節参照) が大きな問題となった．扱う言語にもよるが，デコードの際の RAAM 表現と記号・その他との判別基準，および記号種別の判別基準が，単なるベクトル距離では困難が予想されたため，半ば恣意的な判別基準を用いた．このデコード基準が改善されれば，構文木シーケンス表現の正解率はあるいは向上する可能性もある．ただし，どのようにデコード基準を決定するかは大きな課題である．

SRN の学習を効率良く成功させるには，短い文から学習を始めることが必要だ，という Elman のいわゆる “starting small” の主張 [7] がある．本実験でも自然言語風言語を長い文で学習しているときには，SRN が短いサンプル文の構文解析すら間違える傾向があることを確認した．全体のサンプル文に対して，短いサンプル文の割合を増やしてみて学習させてみたところ，短い文に対しては確かに改善される傾向があったが，全体の学習成績に大きな向上はなかった．これは，本実験の学習アルゴリズムが，許容ユニット誤差範囲内なら重み更新をしないという方式ため，よく学習できている文についてはただ単に無視されるだけだからであると考える．Elman の starting small の考え方では，文の割合だけでなく，ネットワークのワーキング・メモリの大きさも順次大きくしていくとよい[2]ということであり，本実験では，starting small の効果は正確には評価できない．

5.2 議論： 再帰的規則の獲得が困難なのは何故か

自然言語風言語の学習では，文の長さについて SRN はわずかならが汎化能力を示したが，名詞句の埋め込み深さについて汎化能力を示した SRN は見つからなかった．この困難の原因は何だろうか．

第一に考えられるのは，SRN の過去情報保持能力の問題である．SRN は，最近の入力に強く影響される．4.2.3 節で見たように，過去の情報は現在の情報に上書きされるように消えてゆくのである．文を読んでいけば，終りの単語を読んでいる頃には，最初に読んだ単語の記憶は微かである．この困難の解決方法のひとつとして，隠れユニットを増やす

[2]文脈層を文の途中でも数単語毎にクリアすることで，ワーキング・メモリの限定効果を出すようである．

ということがまず考えられる．隠れ層（すなわち文脈層も）のユニット数が増えれば，原理的には記憶保持能力があがるはずである．しかし，表 4.8 の iii) の結果では，隠れ層を増やしてみても汎化能力の向上は見られない．そうなると，次に考えられるのは，計算精度の問題である．SRN は過去のフィード・バックを入力値の一部として計算している，いわばダイナミカル・システムである．初期の誤差は，フィード・バックを繰り返すたびに増幅されてゆく．そのような累積誤差が，文の最後のほうの処理で影響してくることは十分に考えられることである．

第二に，RAAM の構造データ表現能力の問題がある．本実験の RAAM では，そもそも文の表現に使用する記号の数に 3 つユニットを足しただけのユニット数で，隠れユニット数が固定されている．この隠れユニット数で表現できる構文木の深さに限界があることが考えられる．また，RAAM でも計算精度の問題がある．例えば深く埋め込まれた構文木の RAAM 表現を，順次デコードしていくことを考えよう．RAAM のデコーダ部を使うことになるが，ユニットの閾値関数としてシグモイド関数を使っている限り，デコードには誤差が付きまとう．つまり，出力に得られた構文木の RAAM 表現には誤差が含まれている．終端記号が取り出せるまで，誤差を含んだ RAAM 表現のデコードを繰り返すことになる．このような累積誤差は，深く埋め込まれた構文木の記憶に，大きな影響を及ぼすことになるだろう．

第三に，RAAM/SRN システムでは，再帰的な埋め込みを，実数値である隠れユニット活性値を下位に小数展開していくことで表現しているのではないだろうか？図 4.1 では，（ネットワークがカウンタ機能を獲得しているかどうかは不明確であるが,）b の入力が続くに従って，二つの隠れユニットの挙動は微小になっていく．これは実は a についても同じで，図の A の部分で二つの隠れユニットは a の入力が続くに従って挙動が微小になっていく．これらは，そもそもの再帰的な埋め込みの表現自体が，非常に微小な数値の差異によって成り立っていることを示唆する．深く埋め込まれた表現ほど，わずかな誤差に影響を受けやすくなるのである．

ここまでの議論では，スタック獲得の困難さを考える上で計算精度の問題がクローズ・アップされてきたが，今度はもう少し方向を変えて考察してみよう．すなわち，学習の成功率の問題である．今回の実験では，そもそも学習の成功率が低い．これは，解空間に対して求める解の領域が狭いということを意味する．人間の言語獲得がほぼ 100% 成功することを考えると，このままでは本研究で提案するモデルの，脳の言語獲得モデルとしてのもっともらしさはない．学習の成功率を上げるためには，モデルに何らかの制約条件が必要になると考える．この制約条件を考えることが，Chomsky の言う脳内の生得的[3]な言語獲得装置の仕組みを解明する糸口になるかもしれない．

5.3 つぎの課題

$\{a^n b^n\}$ の学習実験で，文の途中と終りとを区別させるタスクで構文解析を学習させ，RAAM/SRN にカウンタを獲得する能力があるかどうか確認する必要がある．さらに，文脈依存言語であり，2-カウンタ実時間言語でもある $\{a^n b^n c^n\}$ の学習実験で，カウンタの拡張が可能であるかどうかを確かめる．また，括弧言語や回文言語など，簡単な CFG で

[3]言語の生得性とコネクショニズムとの関わりについては，文献 [8] に詳しい．

$\{a^n b^n\}$ の拡張となるような言語の学習を試みて，カウンタを超える機能を持つスタックの獲得が可能なのかどうかも実験する．

自然言語風言語の学習には，現在のモデルにさらなる工夫が必要になると考える．すなわち，前節で議論したような困難の原因を克服する必要がある．SRN の拡張として，i) 過去情報を出力しながら学習する．ii) 数時刻前までの過去情報をフィード・バックさせながら学習する．iii) 学習アルゴリズムを強力にする（backpropagation through time などを利用する），などが考えられる．また，RAAM の表現能力を上げるために，記号表現に使うユニット数を大幅に増やす，などの方法がある．そして，根本的な改善として，計算精度を上げる必要がある．

第6章　結論

本研究では，RAAM と SRN を組み合わせた LR パーザという，新しいコネクショニスト・パーザを提案した．さらにこれを計算機上に実装し，シミュレーション実験を行なった．その結果，簡単な CFG である $\{a^n b^n\}$ について，学習した文の長さの 2 倍以上の長さの文を正確に構文解析する汎化能力を獲得した．しかしこの言語については文の終りを認識する必要がなく，ネットワークがカウンタ能力を獲得しているかどうかは明確ではない．また，自然言語風の二つの文法（英語風，日本語風）についても実験を行なった．これらの言語については，構文解析の汎化能力は極めて限られたものしか得られなかった．しかし，学習に使った長さ 10，名詞句の深さ 4 までのサンプル文セットについて，正確な構文解析が可能であることが示された．また，RAAM は埋め込み深さについて汎化能力を持ちうることを確認した．右回帰的要素を含む文法，左回帰的なそれの双方について，同等のモデルで学習が可能であることを確認した．

実験により，提案するモデルの，LR パーザとしての基本的能力，および基本的な再帰的規則の学習能力は示されたと考える．しかし，スタック能力を獲得したとは言えない．このことについて，さらに工夫を加えて研究を継続する必要がある．holistic なパーザの研究が，人間の言語獲得の仕組みの解明に多少なりとも寄与することを確信している．

謝辞

本研究を御指導いただいた櫻井彰人教授，および多くの助言をしてくださった荒木修助手に感謝します．また，計算機環境の整備に多大な支援をしてくださった櫻井研究室の梅津亮氏に感謝します．最後に，本研究の計算機実験・論文作成の際に使用した，Linux，Emacs，LaTeX をはじめとする，多くのフリー・ソフトウェアを公開・配布してくださっている関係者の皆様にも感謝します．

参考文献

[1] Connectionist symbol processing: Dead or alive? In A. Jagota, T. Plato, L. Shastri, and L. Sun eds., *Neural Computing Survey*, Vol. 2, pp. 1–40, 1999. http://www.berkeley.edu/ jagota/NCS.

[2] M. Adamson and R. Damper. B-RAAM: A connectionist model which develops holistic internal representations of symbolic structures. *Connection Science*, 11(1):41–71, 1999.

[3] G. Berg. A connectionist parser with recursive sentence structure and lexical disambiguation. In *Proceedings Tenth National Conference on Artificial Intelligence(AAAI-92)*, pp. 32–37, San Jose,CA, 1992.

[4] D. Chalmers. Syntactic transformations on distributed representations. *Connection Science*, 2:53–62, 1990.

[5] J. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[6] J. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[7] J. Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48:71–99, 1993.

[8] J. Elman, E. Bates, M. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett. *Rethinking Innateness: A Connectionist Perspective on Cevelopment*. The MIT Press, 1996. (乾他訳, 認知発達と生得性，共立出版，1998).

[9] J. Fodor and Z. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71, 1988.

[10] J. Hammerton. Holistic computation: Reconstructing a muddled concept. *Connection Science*, 10(1):3–19, 1998.

[11] E. K. S. Ho and L. W. Chan. How to design a connectionist holistic parser. *Neural Computation*, 11:1995–2016, 1999.

[12] K. Ho and L. Chan. Confluent preorder parsing of deterministic grammars. *Connection Science*, 9(3):269–293, 1997.

[13] M. R. Mayberry and R. Miikkulainen. Combining maps and distributed representations for shift-reduce parsing. In S. Wermter and R. Sun eds., *Hybrid Neural Systems, LNAI 1778*, pp. 144–157. Springer Verlag, 2000.

[14] 野田. コネクショニスト的シンボル処理の現状. システム／制御／情報, 36(10):661–668, 1992.

[15] J. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[16] R. Reilly. Connectionist techniques for on-line parsing. *Network*, 3:37–45, 1992.

[17] P. Rodriguez, J. Wiles, and J. Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999.

[18] D. Rumelhart, G. Hinton, and R. Williams. Learning internal representations by error back-propagation. In J. McClelland, D. Rumelhart, and the PDP Researh Group eds., *Parallel Distributed Processing: Experiments in the Microstructure of Cognition 1: Foundations*, chapter 8, pp. 318–362. The MIT Press, Cambridge, MA, 1986.

[19] 櫻井, 酒井. 言語獲得のモデル. 数理科学, (444):45–51, June 2000.

[20] D. Servan-Schreiber, A. Cleeremans, and J. McClelland. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7:161–193, 1991.

[21] N. Sharkey and A. Sharkey. A modular design for connectionist parsing. In *Twente Workshop on Language Technology 3: Connectionism and Natural Language Processing*, pp. 87–96, Enschede, The Netherlands, 1992. Department of Computer Science, University of Twente.

[22] 田中. 自然言語解析の基礎. 産業図書, 1989.

[23] T. Van Gelder. Compositionality: A connectionist variation on a classical theme. *Cognitive Science*, 14:355–384, 1990.

付 録A　実験に使用した文一覧

実験に使用した，英語風言語，日本語風言語の文を示す．番号，長さ，文の順．長さは文中の終端記号の数だが，ピリオドは含めていない．

A.1　英語風言語の文

```
e1,2,S(NP(N)VP(Vi).)
e2,3,S(NP(N)VP(VtNP(N)).)
e3,4,S(NP(NwhoVP(Vi))VP(Vi).)
e4,5,S(NP(NwhoNP(N)Vt)VP(Vi).)
e5,5,S(NP(N)VP(VtNP(NwhoVP(Vi))).)
e6,6,S(NP(N)VP(VtNP(NwhoNP(N)Vt)).)
e7,5,S(NP(NwhoVP(Vi))VP(VtNP(N)).)
e8,5,S(NP(NwhoVP(VtNP(N)))VP(Vi).)
e9,6,S(NP(NwhoNP(N)Vt)VP(VtNP(N)).)
e10,7,S(NP(NwhoNP(NwhoVP(Vi))Vt)VP(Vi).)
e11,8,S(NP(NwhoNP(NwhoNP(N)Vt)Vt)VP(Vi).)
e12,6,S(NP(N)VP(VtNP(NwhoVP(VtNP(N)))).)
e13,8,S(NP(N)VP(VtNP(NwhoNP(NwhoVP(Vi))Vt)).)
e14,9,S(NP(N)VP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)).)
e15,7,S(NP(NwhoVP(Vi))VP(VtNP(NwhoVP(Vi))).)
e16,8,S(NP(NwhoVP(Vi))VP(VtNP(NwhoNP(N)Vt)).)
e17,6,S(NP(NwhoVP(VtNP(N)))VP(VtNP(N)).)
e18,7,S(NP(NwhoVP(VtNP(NwhoVP(Vi))))VP(Vi).)
e19,8,S(NP(NwhoVP(VtNP(NwhoNP(N)Vt)))VP(Vi).)
e20,8,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoVP(Vi))).)
e21,9,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoNP(N)Vt)).)
e22,8,S(NP(NwhoNP(NwhoVP(Vi))Vt)VP(VtNP(N)).)
e23,8,S(NP(NwhoNP(NwhoVP(VtNP(N)))Vt)VP(Vi).)
e24,9,S(NP(NwhoNP(NwhoNP(N)Vt)Vt)VP(VtNP(N)).)
e25,10,S(NP(NwhoNP(NwhoNP(NwhoVP(Vi))Vt)Vt)VP(Vi).)
e26,11,S(NP(NwhoNP(NwhoNP(NwhoNP(N)Vt)Vt)Vt)VP(Vi).)
e27,8,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoVP(Vi))))).)
e28,9,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt))))).)
e29,9,S(NP(N)VP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt)).)
e30,11,S(NP(N)VP(VtNP(NwhoNP(NwhoNP(NwhoVP(Vi))Vt)Vt)).)
e31,12,S(NP(N)VP(VtNP(NwhoNP(NwhoNP(NwhoNP(N)Vt)Vt)Vt)).)
e32,8,S(NP(NwhoVP(Vi))VP(VtNP(NwhoVP(VtNP(N)))).)
e33,10,S(NP(NwhoVP(Vi))VP(VtNP(NwhoNP(NwhoVP(Vi))Vt)).)
e34,11,S(NP(NwhoVP(Vi))VP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)).)
e35,8,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(Vi))).)
e36,9,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoNP(N)Vt)).)
e37,8,S(NP(NwhoVP(VtNP(NwhoVP(Vi))))VP(VtNP(N)).)
e38,8,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))VP(Vi).)
```

```
e39,9,S(NP(NwhoVP(VtNP(NwhoNP(N)Vt)))VP(VtNP(N)).)
e40,10,S(NP(NwhoVP(VtNP(NwhoNP(NwhoVP(Vi))Vt)))VP(Vi).)
e41,11,S(NP(NwhoVP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)))VP(Vi).)
e42,9,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoVP(VtNP(N)))).)
e43,11,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoNP(NwhoVP(Vi))Vt)).)
e44,12,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)).)
e45,10,S(NP(NwhoNP(NwhoVP(Vi))Vt)VP(VtNP(NwhoVP(Vi))).)
e46,11,S(NP(NwhoNP(NwhoVP(Vi))Vt)VP(VtNP(NwhoNP(N)Vt)).)
e47,9,S(NP(NwhoNP(NwhoVP(VtNP(N)))Vt)VP(VtNP(N)).)
e48,10,S(NP(NwhoNP(NwhoVP(VtNP(NwhoVP(Vi))))Vt)VP(Vi).)
e49,11,S(NP(NwhoNP(NwhoVP(VtNP(NwhoNP(N)Vt)))Vt)VP(Vi).)
e50,11,S(NP(NwhoNP(NwhoNP(N)Vt)Vt)VP(VtNP(NwhoVP(Vi))).)
e51,12,S(NP(NwhoNP(NwhoNP(N)Vt)Vt)VP(VtNP(NwhoNP(N)Vt)).)
e52,11,S(NP(NwhoNP(NwhoNP(NwhoVP(Vi))Vt)Vt)VP(VtNP(N)).)
e53,11,S(NP(NwhoNP(NwhoNP(NwhoVP(VtNP(N)))Vt)Vt)VP(Vi).)
e54,12,S(NP(NwhoNP(NwhoNP(NwhoNP(N)Vt)Vt)Vt)VP(VtNP(N)).)
e57,9,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))).)
e58,11,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoNP(NwhoVP(Vi))Vt)))).)
e59,12,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)))).)
e60,11,S(NP(N)VP(VtNP(NwhoNP(NwhoVP(VtNP(NwhoVP(Vi))))Vt)).)
e61,12,S(NP(N)VP(VtNP(NwhoNP(NwhoVP(VtNP(NwhoNP(N)Vt)))Vt)).)
e62,12,S(NP(N)VP(VtNP(NwhoNP(NwhoNP(NwhoVP(VtNP(N)))Vt)Vt)).)
e65,10,S(NP(NwhoVP(Vi))VP(VtNP(NwhoVP(VtNP(NwhoVP(Vi)))))).)
e66,11,S(NP(NwhoVP(Vi))VP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt)))).)
e67,11,S(NP(NwhoVP(Vi))VP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt)).)
e70,9,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(VtNP(N)))).)
e71,11,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoNP(NwhoVP(Vi))Vt)).)
e72,12,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)).)
e73,10,S(NP(NwhoVP(VtNP(NwhoVP(Vi))))VP(VtNP(NwhoVP(Vi))).)
e74,11,S(NP(NwhoVP(VtNP(NwhoVP(Vi))))VP(VtNP(NwhoNP(N)Vt)).)
e75,9,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))VP(VtNP(N)).)
e76,10,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(Vi)))))VP(Vi).)
e77,11,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt)))))VP(Vi).)
e78,11,S(NP(NwhoVP(VtNP(NwhoNP(N)Vt)))VP(VtNP(NwhoVP(Vi))).)
e79,12,S(NP(NwhoVP(VtNP(NwhoNP(N)Vt)))VP(VtNP(NwhoNP(N)Vt)).)
e80,11,S(NP(NwhoVP(VtNP(NwhoNP(NwhoVP(Vi))Vt)))VP(VtNP(N)).)
e81,11,S(NP(NwhoVP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt)))VP(Vi).)
e82,12,S(NP(NwhoVP(VtNP(NwhoNP(NwhoNP(N)Vt)Vt)))VP(VtNP(N)).)
e85,11,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoVP(VtNP(NwhoVP(Vi)))))).)
e86,12,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt)))).)
e87,12,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt)).)
e90,11,S(NP(NwhoNP(NwhoVP(Vi))Vt)VP(VtNP(NwhoVP(VtNP(N)))).)
e93,11,S(NP(NwhoNP(NwhoVP(VtNP(N)))Vt)VP(VtNP(NwhoVP(Vi))).)
e94,12,S(NP(NwhoNP(NwhoVP(VtNP(N)))Vt)VP(VtNP(NwhoNP(N)Vt)).)
e95,11,S(NP(NwhoNP(NwhoVP(VtNP(NwhoVP(Vi))))Vt)VP(VtNP(N)).)
e96,11,S(NP(NwhoNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))Vt)VP(Vi).)
e97,12,S(NP(NwhoNP(NwhoVP(VtNP(NwhoNP(N)Vt)))Vt)VP(VtNP(N)).)
e100,12,S(NP(NwhoNP(NwhoNP(N)Vt)Vt)VP(VtNP(NwhoVP(VtNP(N)))).)
e105,12,S(NP(NwhoNP(NwhoNP(NwhoVP(VtNP(N)))Vt)Vt)VP(VtNP(N)).)
e115,11,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(Vi)))))))).)
e116,12,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt)))))).)
e117,12,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt))))).)
e120,12,S(NP(N)VP(VtNP(NwhoNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))Vt)).)
```

```
e128,11,S(NP(NwhoVP(Vi))VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))).)
e136,11,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(VtNP(NwhoVP(Vi)))))).)
e137,12,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt))))).)
e138,12,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt)).)
e141,11,S(NP(NwhoVP(VtNP(NwhoVP(Vi))))VP(VtNP(NwhoVP(VtNP(N))))).)
e144,11,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))VP(VtNP(NwhoVP(Vi)))).)
e145,12,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))VP(VtNP(NwhoNP(N)Vt)).)
e146,11,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(Vi))))))VP(VtNP(N)).)
e147,11,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))))VP(Vi).)
e148,12,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoNP(N)Vt)))))VP(VtNP(N)).)
e151,12,S(NP(NwhoVP(VtNP(NwhoNP(N)Vt)))VP(VtNP(NwhoVP(VtNP(N))))).)
e156,12,S(NP(NwhoVP(VtNP(NwhoNP(NwhoVP(VtNP(N)))Vt)))VP(VtNP(N)).)
e166,12,S(NP(NwhoNP(N)Vt)VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))).)
e179,12,S(NP(NwhoNP(NwhoVP(VtNP(N)))Vt)VP(VtNP(NwhoVP(VtNP(N))))).)
e184,12,S(NP(NwhoNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))Vt)VP(VtNP(N)).)
e222,12,S(NP(N)VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))))).)
e252,12,S(NP(NwhoVP(VtNP(N)))VP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))).)
e265,12,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))VP(VtNP(NwhoVP(VtNP(N))))).)
e270,12,S(NP(NwhoVP(VtNP(NwhoVP(VtNP(NwhoVP(VtNP(N)))))))VP(VtNP(N)).)
```

A.2　日本語風言語の文

```
j1,2,S(VP(NP(N)Vi).)
j2,3,S(VP(NP(ViN)Vi).)
j3,4,S(VP(NP(NP(N)VtN)Vi).)
j4,5,S(VP(NP(NP(ViN)VtN)Vi).)
j5,3,S(VP(NP(N)NP(N)Vt).)
j6,4,S(VP(NP(N)NP(ViN)Vt).)
j7,4,S(VP(NP(ViN)NP(N)Vt).)
j8,5,S(VP(NP(ViN)NP(ViN)Vt).)
j9,6,S(VP(NP(NP(NP(N)VtN)VtN)Vi).)
j10,7,S(VP(NP(NP(NP(ViN)VtN)VtN)Vi).)
j11,5,S(VP(NP(N)NP(NP(N)VtN)Vt).)
j12,6,S(VP(NP(N)NP(NP(ViN)VtN)Vt).)
j13,5,S(VP(NP(NP(N)VtN)NP(N)Vt).)
j14,6,S(VP(NP(NP(N)VtN)NP(ViN)Vt).)
j15,6,S(VP(NP(NP(ViN)VtN)NP(N)Vt).)
j16,7,S(VP(NP(NP(ViN)VtN)NP(ViN)Vt).)
j17,6,S(VP(NP(ViN)NP(NP(N)VtN)Vt).)
j18,7,S(VP(NP(ViN)NP(NP(ViN)VtN)Vt).)
j19,8,S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)Vi).)
j20,9,S(VP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)Vi).)
j21,7,S(VP(NP(N)NP(NP(NP(N)VtN)VtN)Vt).)
j22,8,S(VP(NP(N)NP(NP(NP(ViN)VtN)VtN)Vt).)
j23,7,S(VP(NP(NP(N)VtN)NP(NP(N)VtN)Vt).)
j24,8,S(VP(NP(NP(N)VtN)NP(NP(ViN)VtN)Vt).)
j25,7,S(VP(NP(NP(NP(N)VtN)VtN)NP(N)Vt).)
j26,8,S(VP(NP(NP(NP(N)VtN)VtN)NP(ViN)Vt).)
j27,8,S(VP(NP(NP(NP(ViN)VtN)VtN)NP(N)Vt).)
j28,9,S(VP(NP(NP(NP(ViN)VtN)VtN)NP(ViN)Vt).)
j29,8,S(VP(NP(NP(ViN)VtN)NP(NP(N)VtN)Vt).)
```

```
j30,9,S(VP(NP(NP(ViN)VtN)NP(NP(ViN)VtN)Vt).)
j31,8,S(VP(NP(ViN)NP(NP(NP(N)VtN)VtN)Vt).)
j32,9,S(VP(NP(ViN)NP(NP(NP(ViN)VtN)VtN)Vt).)
j33,10,S(VP(NP(NP(NP(NP(NP(N)VtN)VtN)VtN)VtN)Vi).)
j34,11,S(VP(NP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)VtN)Vi).)
j35,9,S(VP(NP(N)NP(NP(NP(NP(N)VtN)VtN)VtN)Vt).)
j36,10,S(VP(NP(N)NP(NP(NP(NP(ViN)VtN)VtN)VtN)Vt).)
j37,9,S(VP(NP(NP(N)VtN)NP(NP(NP(N)VtN)VtN)Vt).)
j38,10,S(VP(NP(NP(N)VtN)NP(NP(NP(ViN)VtN)VtN)Vt).)
j39,9,S(VP(NP(NP(NP(N)VtN)VtN)NP(NP(N)VtN)Vt).)
j40,10,S(VP(NP(NP(NP(N)VtN)VtN)NP(NP(ViN)VtN)Vt).)
j41,9,S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)NP(N)Vt).)
j42,10,S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)NP(ViN)Vt).)
j43,10,S(VP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)NP(N)Vt).)
j44,11,S(VP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)NP(ViN)Vt).)
j45,10,S(VP(NP(NP(NP(ViN)VtN)VtN)NP(NP(N)VtN)Vt).)
j46,11,S(VP(NP(NP(NP(ViN)VtN)VtN)NP(NP(ViN)VtN)Vt).)
j47,10,S(VP(NP(NP(ViN)VtN)NP(NP(NP(N)VtN)VtN)Vt).)
j48,11,S(VP(NP(NP(ViN)VtN)NP(NP(NP(ViN)VtN)VtN)Vt).)
j49,10,S(VP(NP(ViN)NP(NP(NP(NP(N)VtN)VtN)VtN)Vt).)
j50,11,S(VP(NP(ViN)NP(NP(NP(NP(ViN)VtN)VtN)VtN)Vt).)
j51,12,S(VP(NP(NP(NP(NP(NP(NP(N)VtN)VtN)VtN)VtN)VtN)Vi).)
j53,11,S(VP(NP(N)NP(NP(NP(NP(NP(N)VtN)VtN)VtN)VtN)Vt).)
j54,12,S(VP(NP(N)NP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)VtN)Vt).)
j55,11,S(VP(NP(NP(N)VtN)NP(NP(NP(NP(N)VtN)VtN)VtN)Vt).)
j56,12,S(VP(NP(NP(N)VtN)NP(NP(NP(NP(ViN)VtN)VtN)VtN)Vt).)
j57,11,S(VP(NP(NP(NP(N)VtN)VtN)NP(NP(NP(N)VtN)VtN)Vt).)
j58,12,S(VP(NP(NP(NP(N)VtN)VtN)NP(NP(NP(ViN)VtN)VtN)Vt).)
j59,11,S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)NP(NP(N)VtN)Vt).)
j60,12,S(VP(NP(NP(NP(NP(N)VtN)VtN)VtN)NP(NP(ViN)VtN)Vt).)
j61,11,S(VP(NP(NP(NP(NP(NP(N)VtN)VtN)VtN)VtN)NP(N)Vt).)
j62,12,S(VP(NP(NP(NP(NP(NP(N)VtN)VtN)VtN)VtN)NP(ViN)Vt).)
j63,12,S(VP(NP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)VtN)NP(N)Vt).)
j65,12,S(VP(NP(NP(NP(NP(ViN)VtN)VtN)VtN)NP(NP(N)VtN)Vt).)
j67,12,S(VP(NP(NP(NP(ViN)VtN)VtN)NP(NP(NP(N)VtN)VtN)Vt).)
j69,12,S(VP(NP(NP(ViN)VtN)NP(NP(NP(NP(N)VtN)VtN)VtN)Vt).)
j71,12,S(VP(NP(ViN)NP(NP(NP(NP(NP(N)VtN)VtN)VtN)VtN)Vt).)
```

付 録B　シミュレーション・プログラム概要

計算機実験のために実装したプログラムの概要を述べる．大きく分けると，文生成用のプログラム，RAAM/SRN 用のプログラム，それと RAAM/SRN を使ってパラメータ探索実験をするためのスクリプトとがある．スクリプトは Perl で書かれ，他はすべて C 言語で書かれた．

C プログラムは，GCC-2.95.2，PGCC-2.95.2，および Compaq C Compiler にてコンパイルされ，GCC 版は Pentium machine の Debian GNU/Linux および Soralis 上で，PGCC 版は Pentium machine の Debian GNU/Linux 2.2 上で，CCC 版は Alpha machine の Linux 上で，それぞれ実験に使われた．開発は Pentium の Linux 上で mule2 を使って行ない，デバッグには GDB を使用した．また，メモリ・リークのチェックにはフリー・ソフトウェアの Checker および Memprof を使った．

また，ライブラリ作成にあったっては，一部をフリー・ソフトウェアである Publib_0.28 のソースの一部を改変して利用した．ここに記して感謝する．

以下に，プログラムおよびライブラリ別にコメントする．

- ライブラリ

  libbp　Backpropagation 基本ライブラリ

  libprs　libbp を構文解析学習で利用するためのライブラリ

  libstack　libprs で使うスタック機能

  libqueue　libprs で使うキュー機能

- アプリケーション

  sg3　文生成

  raam22　RAAM ネットワーク

  srn22　SRN ネットワーク

- スクリプト
  RAAM および SRN のアプリケーションを使い，最善の汎化能力をもつネットワークのパラメータ・セットを探索する．