A Pulsed Neural Network for Language Understanding
- Discrete-Event Simulation of a Short-Term Memory Mechanism and Sentence Understanding -
言語理解のためのパルス神経回路網
- 短期記憶機構と文理解の離散イベント式シミュレーション -

by

MAKINO Takaki
牧野 貴樹

Ph. D. Dissertation
博士論文

Submitted to
the Graduate School of Science
the University of Tokyo
on December 21, 2001
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Science

Thesis Supervisor: TSUJII Jun-ichi　辻井 潤一
Professor of Information Science

ABSTRACT

Various language processing algorithms have been studied to find the algorithm used in the human language understanding, but no algorithm has proven its existence by physiological evidences. In such a situation, we should consider an approach to pursue implementational constraints and preferences from a computational theory of the language understanding process.

In this paper, we study the model of a short-term memory mechanism of the human brain suitable for language understanding. Specifically, the following three topics are pursued.
(1) The exploration of the element necessary for building a short-term memory mechanism suitable for language understanding in the framework of neural network
(2) The techniques for an efficient simulation of general pulse neural networks in a continuous time.
(3) Construction of a primitive simulation of language understanding based on (1) and (2).

In (1), we clarify the following on the language-understanding neural networks. i) A binding problem has to be solved in order to represent a result of language understanding, and the most promising way is to utilize behavior in the time domain of a neural network. ii) Requirement of phase arbitration causes us to build a structural time-series memory on a neural network. iii) Application of grammatical rules can be implemented in the same way as a prediction of a time series.

In (2), we studied the event-driven pulse neural network simulator. In order to research complex operations in a time domain, such as phase mediation, the network simulation with high time precision is demanded, while conventional discrete-time systems is limited in simulation speed. On the other hand, discrete-event systems have difficulty in handling delayed firing for general neuron models. In this study, we show that our new technique with the second-order incremental partitioning method enables us to build an event-driven pulse network simulator in general neuron models by numerical calculation of delayed firing times. We also describe technique for more efficient handling of delayed firing by filtering redundant calculations of delayed firings.

Finally, in (3), we build a neural network simulation model, which understands the simple sentence of 3 to 4 words, in order to demonstrate the studies of language understandings in (1) and (2). We discuss our language-understanding system in various aspects and future directions of research for better understanding of a sentence.

論文要旨

　人間の言語理解がどのようなアルゴリズムで実現されているかを調べるために、さまざまな言語処理アルゴリズムが研究されてきたが、生理学的証拠と結びつくものは見つかっていない。このような状況では、言語理解過程に関する計算理論からわかる言語理解メカニズムの実装上の制約や有利な構造を調べるというアプローチが有効である。

　本論文では、言語理解に適した人間の短期記憶機構のモデルと、そのシミュレーションに必要な技術について提案する。具体的には、以下の３項目についての研究を行う。

(1) 言語理解に適した短期記憶機構を神経回路網で構築するときに必要となる要素の探究

(2) 連続時間での高速かつ一般的なパルス神経回路網シミュレーション技術

(3) (1) 及び (2) を利用した、言語理解の原始的なシミュレーション

　ここで (1) については、言語を取り扱う神経回路網が備えているべき要素について、次のようなことを明らかにした。i) 言語理解の結果を表現するには束縛問題を解決することが必要であり、そのためには神経回路網の時間領域での振る舞いを利用することが最も有望であること。ii) 位相調停を実現するために、構造的な時系列記憶システムが構築されるべきであること。iii) 時系列予測の形式で文法規則の適用が実装できること。

　また、(2) に関しては、イベント離散方式によるパルス神経回路網シミュレーション方式を研究した。位相調停など複雑な時間領域での操作を研究するために必要となる、時間精度の高い回路網シミュレーションは、時間離散方式による従来の手法ではシミュレーション速度に限界があった。一方、イベント離散方式には一般的なニューロンモデルに対し遅延発火が扱えないという問題があった。本研究で提案する二次漸進分割法は、遅延発火を解析的に計算することにより、一般的なニューロンモデルに対してもイベント離散方式でパルス回路網シミュレーションが構築できることを示す。また、冗長な遅延発火計算を検出することで遅延発火の処理を高速化する手法についても述べる。

　そして、(3) においては、(1) 及び (2) の研究を例証するために、３～４単語程度の単純な文を理解する神経回路網シミュレーションを構築する。ここで構築した言語理解システムをさまざまな側面から検討し、よりよい文の理解のために研究すべき方向について考察する。

Acknowledgments

First of all, I would like to thank Prof. Tsujii Jun-ichi for his invaluable supervision. I'm indebted to Prof. Aihara Kazuyuki for his kind help and precious directions for the research. Many thanks go to the colleagues in Tsujii group at Tokyo university. Dr. Ninomiya Takashi and Mr. Miyao Yusuke patiently read my puzzling draft and gave me many advices and discussion; Dr. Bekki Daisuke gave me invaluable advices on the strategy of the dissertation, which helped this dissertation so much; Prof. Torisawa Kentaro and Dr. Edson T. Miyamoto shared precious discussions with me; and Dr. Tateisi Yuka and Dr. Ohta Tomoko supported my health with a lot of delicious oranges. Finally, I'd like to thank Ms. Mino Yukiko, for being beside me and encouraging me all the time. I would not complete this dissertation without her.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We dream, if a computer could understand language, as humans do. No more than a dream, at least up to now. At present, computers cannot see meanings. Since language understanding is to obtain meanings from a sentence, nothing can be understood without meanings. In these days, elaborate computer programs can correctly deal with grammars, and can even play language games with us by juggling words. But still they never understand languages. It is impossible; until, at least, they can handle meanings.

What, however, is a meaning, in the first place? Although many studies are trying to describe meanings (or 'semantics') by symbolic systems, they do not tell us what is understanding. Even if we made a program to convert from a symbolic system (language) to another symbolic system (semantic representation) [35, 31, 34], we couldn't say the program understands something. Some other studies try to emulate semantic behavior in language by using statistics and thesaurus, achieving a higher precision of natural language processing

[23, 25]. However, it is far from understanding language.

One approach to know meanings is to investigate how a meaning is handled by human beings. We know it happens nowhere but in the brain, the seat of the thought, memory and consciousness. The brain, which can understand a sentence, is able to associate the sentence with the brain's surrounding world, sensory input and experience. This association should be a meaning, what the current computers don't have. The way of making associations is unique to the architecture of the brain; we can pursue the mechanism to achieve our dream.

Due to the high complexity of the brain, the physiological analysis of the brain does not reach the representation and handling of meanings. The long history of brain science has succeeded to elucidate shallow parts of the brain function, such as sensory analysis and motor control mechanisms [4]. Recent studies enlightened more deeper parts of the animal brain, such as place-coding in the rat hippocampus [39]. However, since the meanings and languages are in the deepest part in the human brain, our knowledge is too little to see the mechanism.

Here we should come back to our starting point, language. Language is an expression of a meaning in the brain, which has evolved along with the human beings over thousands of years. Structures, constructions, and contents of sentences are all the reflection of the brain mechanism of meanings. It is said that the language also reflects cognitive frameworks [56]. Language gives us both requirements and clues to the meanings in the brain; they can be the key to solve the mechanism of meanings. The previous research to build a language-processing model on a brain-like architecture [8, 18] lacks this point of view: the model must be able to represent a meaning carried by a language.

In this dissertation, we pursue the goal by constructing a minimal model of language understanding in the human brain, focusing on its short-term memory. When a person reads or listens to a text or discourse, short-term memory plays a critical role in storing the intermediate and final products of his/her computation, as he/she constructs and integrates a meaning representation from a stream of successive words from a text or discourse. In addition, the short-term memory can also be viewed as the pool of operational resources that perform the symbolic computations and thereby generate the intermediate and final meaning representation [22]. Language poses requirements to the capability of the memory mechanism; such a mechanism should be able to keep meaning representations and compute meanings from the stream of words.

How is the meaning represented in the brain? How is the language understood, that is, how does the brain construct a representation of a meaning from a sentence? Although it is impossible to construct a perfect model, we think that our goal is supported by not only the model itself but also the process of the model construction.

1.1 Position of This Study in the Research Program

Studies in a large and deep research domain, such as the human language understanding mechanism, need concrete research programs to support each other; otherwise, the studies rarely contribute to the domain.

Marr's research program of the process of vision [36] is a good starting point to consider the research program of language understanding. He proposed three

Computational Theory	What is the goal of the computation, why is the goal appropriate, and what is the logic of the accomplishable strategy?
Representation and Algorithm	How can the computational theory be realized? In particular, what is the representation of its input and output? What is an algorithm for the conversion?
Hardware Implementation	How are the representation and algorithm materialized?

Table 1.1: Three levels required to describe a machine that performs an information processing task, proposed in Marr's research program [36].

levels to describe of a machine that performs an information processing task, as shown in Table 1.1. The first level, computational theory, concerns what should be performed in the process, and which principles should be satisfied by the process and its input/output. The second level, representation and algorithm, concerns how the input/output of the process is represented, and what procedure processes the representation to achieve the principles studied in the first level. The third level, hardware implementation, concerns the device and mechanism to realize the representation and algorithm studied in the second level.

A computational theory is described with various representations and algorithms, and an algorithm is materialized by various implementations. Nevertheless, we first need a computational theory of the domain of interest; otherwise we cannot evaluate achievements of the algorithms. Human vision is an implementation of a computational theory of vision processing, which looks easy but involves complex calculations to solve difficulties. Study of the human visual cortex (an implementation) will hardly reveal anything without knowledge of the difficulties in the algorithm of vision processing; study of a visual processing algorithm cannot be evaluated without a computational theory to be achieved.

Marr claims that the human vision should be studied from the upper level to the lower one. First, a computational theory of vision should be clearly declared, without concerning the actual algorithm and representation. Thereafter, the representations and algorithms for the theory should be studied; through this study, we know the difficulties of the process and ways to solve the difficulties. Then we can study the implementation to see which algorithm and which representation are constructed on the human vision mechanism. In the domain of vision processing, this approach is successfully revealing the mechanisms of visual cognition. The study of a vision processing algorithm has shown that efficient vision processing consists of a series of stages of algorithms that calculate from local features to global features; this clarified the role of the layered structure of neurons in the visual cortex, in which later layers respond to more global features [36, 3].

The same research program has been being tried in the domain of human language ability. Researchers of linguistics have tried to clarify the computational theory of language [7, 26, 42]. The representations and algorithms are studied in computational linguistics, to successfully show various efficient algorithms of linguistic computation [24, 49, 57, 48].

However, it is not sure that the research of language goes well with this research program as the research of vision does. Although current studies suppose that a linguistic process is divided into several stages as in the process of vision, researchers of neurolinguistics are still unable to see the evidences of the staged processing in the brain. We have to face the possibility that the cerebral implementation of human language processing performs at so different stages of processing from our research that we cannot easily associate the implementation to the studied algorithms. If this is the case, how can we reach the implementation model of human language ability?

Our idea is to pursue the relation between the first level and the third level. Even if we have no knowledge of the algorithms and representations of a process, we are still able to suggest a possible implementation from the required properties of the process. If a corresponding mechanism to the suggested implementation is found in the human brain, it will help the research on the second level by narrowing possibilities of the representation and algorithm used in the brain.

We also claim that this approach should be accompanied with computational simulation of the implementation. Because of the inherent complexity of language processing, representation on the third level will possibly be incomprehensible, unlike the representation on the first and second levels. Only the computational simulation can reveal the implication of the theory in the third level; without simulation, this approach will be difficult to make an impact on the research on the second level

This dissertation consists of three parts. The first part, Chapter 3, explores the requirement of the process to pursue the plausible way of implementation and mechanisms. The second part, Chapter 4, concerns the simulation techniques of a neural network with high temporal precision. The third part, Chapter 5 describes a design and simulation experiments of a neural network language understanding mechanism inspired by the first part.

We believe this series of studies contributes the research of human language ability. The requirement discussed in Chapter 3 suggests the existence of a global mechanism over the cerebral portion of language processing, despite the evasion of a global property in the research of connectionist models. The simulation techniques presented in Chapter 3 lay a foundation of a precise, efficient, and large-scale simulation of pulsed neural networks, which is a convincing framework of the implementation of human language ability. Moreover, the simulation experiments in Chapter 5 lead us to the fruitful discussions of possible implementations and algorithms of the human language ability.

1.2 Exploration of the Requirements

We first explore requirements of possible implementations for the language understanding process in the brain. In Chapter 3, we point out that the representation of *bindings*, an important part of the meanings, demands *temporal coding*. A binding is a relation of an attribute and an object. For example, a sentence 'John loves Mary' is supposed to have two bindings, 'John — lover' and 'Mary — beloved'. Since we make a clear distinction between the bindings and another set of bindings, 'Mary — lover' and 'John — beloved', the binding should be an important part of the meanings; thus we assume bindings should be represented in the brain. However, this is not an easy task for neural network architectures. We argue a possible mechanism of a binding in the brain, and state an advantage of temporal coding using oscillation phases.

Following that, we pursue the implementation of language understanding under an assumption of temporal coding. We discuss that a special mechanism is required to manage inputs to the temporal coding, which we call *phase arbitration*. Since a sentence input to the brain is a temporal sequence of words, direct connections from the input to the temporal coding of bindings causes unexpected collisions in the pulse timings (phases). Thus the brain should have some mechanism to arbitrate phases for a stable construction of temporal coding from a sentence input. We discuss that the phase arbitration is hard to be implemented by local mechanisms, and claim the usage of a global signal in the phase arbitration mechanism, despite the evasion of a global property in the research of connectionist models [45].

1.3 Simulation Techniques

We also focus on the simulation techniques for our model. It is an important step to validate a model, at least empirically, by a computer simulation. Since our exploration discovers the importance of the temporal aspects of neural behaviors, we need efficient simulation techniques for a high temporal precision, being capable of handling general neuron models. A discrete-time simulation framework, which is used in most neural network simulators, loses efficiency for a high temporal precision. On the other hand, a discrete-event simulation framework provides a high temporal precision and is suitable for the simulation of pulsed neural networks. However, existing discrete-event simulators rely on simple neuron models, because delayed firings of general neuron models pose difficulty on a discrete-event framework.

In Chapter 4, we introduce several techniques for a discrete-event pulsed network simulator. We present a *second-order incremental partitioning* method, which is able to solve delayed firings for any Spike-Response neuron model with finite discontinuity. Moreover, in order to achieve efficiency at the handling of delayed firings, we developed the *gradient limit checking* technique. We show that the resulting neural network simulator, PUNNETS, is able to simulate a large-scale network efficiently.

1.4 Language-Understanding Simulation

Finally, in Chapter 5, we design a small neural network model that converts a simple input sentence into a set of binding representation in the temporal coding, and test the model on the simulation. The purpose of the simulation is to validate the requirements explored in Chapter 3. This network model can be an important step towards a language-understanding neural network model, because binding representations are a critical portion of semantic representations in our assumption. The experiments on the network model show that a neural network model can be constructed within the requirements and preferences we explored in Chapter 3. Although the model is too small to discuss the linguistic problems, the following discussion reveals some important differences of the model from human language understanding, and points to further directions of research.

Chapter 2

Background

As background matters we first introduce related studies in two areas, connectionist approaches for language processing and the temporal coding of neurons. Thereafter we describe a brief overview of the Spike-Response model, which is a general neuron model for pulsed neural networks.

2.1 Related Work

In this section we give a brief overview of some relevant studies. First we have a glance on the history of the temporal coding on neurons. After that, we survey the connectionist approaches for natural language processing.

2.1.1 Temporal Coding on Neurons

It is known that neurons, the cells constituting a brain, transmit information by voltage pulses of membrane potential [14]. Since all pulses of a given neuron look alike, the form of pulses does not carry any information. Rather, it is the number and the timing of pulses which matter.

Perceptron, an early neuron model in artificial neural networks, uses binary output, which represents all-or-nothing nature of a pulse. A learning algorithm called Perceptron Convergence Process is proposed, but this algorithm has a limited power of learning [38]. More recently, generalized delta rule, also called as error back-propagation, is developed, which uses a sigmoidal gate function to provide a continous value as an output of a neuron [45]. This continuous value is supposed as a 'rate-coded' value, a mean firing rate of a neuron or a group of neurons. Since this model has an ability to learn a complex function, it is applied to many complex problems.

Recently, researchers have started focusing on a temporal aspect of the neuron behavior. Although the rate-coded model was powerful, it used only a number of spikes as information and ignored the timing information of pulses. If it is used asynchronously, with analog values encoded by a temporal pattern of firing times, a spiking neuron has in principle not only more computational power than a perceptron neuron, but also more computational power than a sigmoidal gate neuron [30]. Temporal aspect of coding is also said to be used in various other domains in the human brain [12].

One notable work is SHRUTI [46], which proved that the temporal coding can represent dynamic bindings. This system is capable of reflexive reasoning in a parallel way, representing multiple binding at the same time. However, the system uses various artificial nodes, and it is not clear how the connections between nodes are learned. Moreover, phases are determined artificially, although

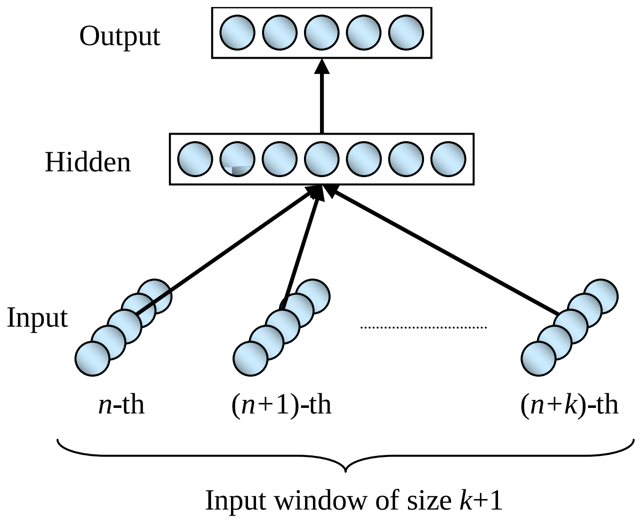


Figure 2.1: Spatial Configuration of a Neural Network. A sequence of linguistic elements (such as letters and words) are spatially extracted as input to a feed-forward network. Note that the input is limited by the $(k + 1)$-sized window, so that no relation over the window size can be captured.

it is a critical problem in the brain.

2.1.2 Connectionist Approaches for Language Processing

Several studies tackle to the problem of linguistic processing by an connectionist approach, although they do not reach to the semantics. In this section, we have a brief overview on two representational studies.

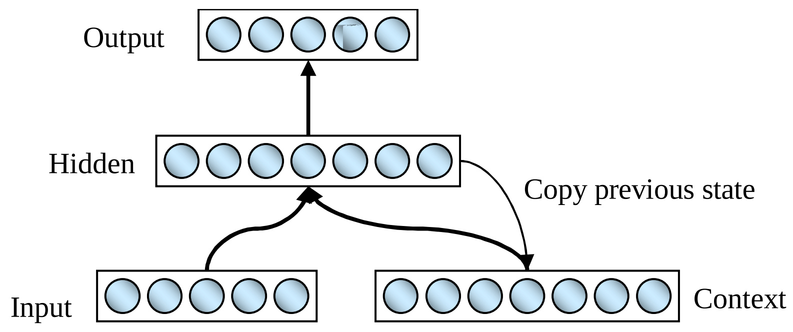


Figure 2.2: Simple Recurrent Network Model by Elman, in which activations are copied from hidden layer to context layer.

Simple Recurrent Network

Natural language processing has been one of the most difficult challenges for an connectionist approach. Spatial configuration, as shown in Figure 2.1, was used in earlier studies, such as pronunciation estimation from spelling [11]. However, the configuration had an substantial problem for application to language processing. The width of the input is constant, that is, the network can never handle relation beyond the constant-sized window, such as a subject-verb agreement with a long relative clause.

A notion of time was introduced to deal with this problem. Elman applied to language processing the Simple Recurrent Network [8] (in Figure 2.2), which uses a copy of hidden layer to represent its context. Then the word sequence is input to the network, one word at a time. The network is trained to predict the next word from the current stream of input words. He claims that this network can discover word segmentation and lexical classes [8], word clustering [9] and grammar with sentence embedding sentence [10].

This configuration (we call *temporal configuration*) has an advantage over spatial configuration in the following points. First, the context layer works as a memory of the past inputs. Although the context layer seems to keep only the previous state of the network, it is calculated using one more previous state. By recursion, the context layer becomes a memory, which can represent longer dependence of temporal events.

The second advantage of the temporal configuration is the resemblence of the processing to the language understanding of human. We cannot say that an SRN understands a sentence in any way, since the network is just trained for the assigned task, which is far from understanding. However, it is sure that a person processes a sentence in temporally divided way. Listening to a narration, she/he receives a sentence as a stream of sound. Reading a text, she/he sequentially picks up words by eyes. It is apparent that a person stores previous readings in a memory to understand a sentence, as an SRN does.

However, we cannot use this model directly for sentence understanding. We discuss this point in Chapter 3.

Henderson's Connectionist Parsers

Henderson extended the idea of SHRUTI, that is, to introduce temporal synchrony as a medium of structural information. His first work [17] shows that the reflexive reasoning of the SHRUTI system can applied to the parsing problem. His connectionist parser is based on Structure Unification Grammar and capable of parsing sentences, in which he argues that constraints incorporated by the connectionist architecture helps prediction of sentence acceptability. The parser receives each word for one cycle, and emits a tree fragment as soon as it is completed. When the parsing finishes, all the tree fragments are emitted so that the fragments constitutes the full parsing tree.

He also applied the idea of SHRUTI to a network which learns to parse [18]. Backpropagation through time [54] is used to train the Simple Synchrony Network, which is like Elman's SRN with synchrony representation, to produce SUG parsing trees from a sequence of words. He reports that the parser achieved high performance than statistical parsers when trained with relatively small corpus.

Both networks are targetted to the syntactic property of the language and they are not intended as a language understanding model. Even if we regard the tree fragments are meaning representation, the connectionist parser 'forgets' about the emitted tree fragment, and the parser memory becomes empty as the parsing finishes. Our assumption, the language has a simple association to the short-term memory for meaning, contradicts such a mechanism. Moreover, the learning work uses a supervised learning rule, and there is no discussion on how the human brain learns to understand sentences.

2.2 Common Definitions

In this dissertation, we model the language understanding in a form of neural networks. For readability we use the common notations and definitions for the network modeling. In this section we describe a *spike-response model* of a pulsed neuron, following the definition in [14].

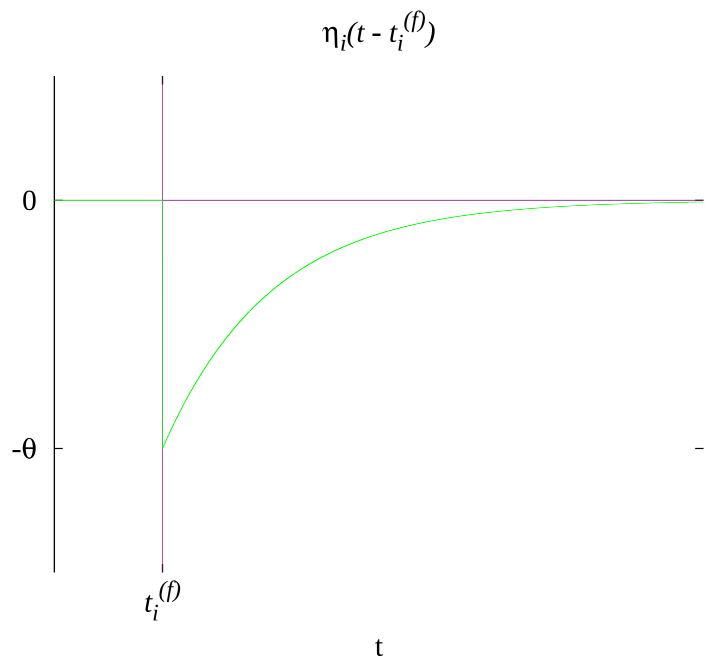


Figure 2.3: Sample η Kernel Function.

2.2.1 Spike Response Model

The state of neuron i is described by a state variable u_i, which may be interpreted as the electrical membrane potential of a neuron in the biological context. The neuron is said to fire, if u_i reaches a threshold θ. The moment of threshold crossing defines the firing time $t_i^{(f)}$, which means the fth firing time of neuron i. The set of all firing times of neuron i is denoted by

$$\mathcal{F}_i = \{t_i^{(f)}; 1 \leq f \leq n\} = \{t | u_i(t) = \theta\} \tag{2.1}$$

Two different processes contribute to the value of the state variable u_i.

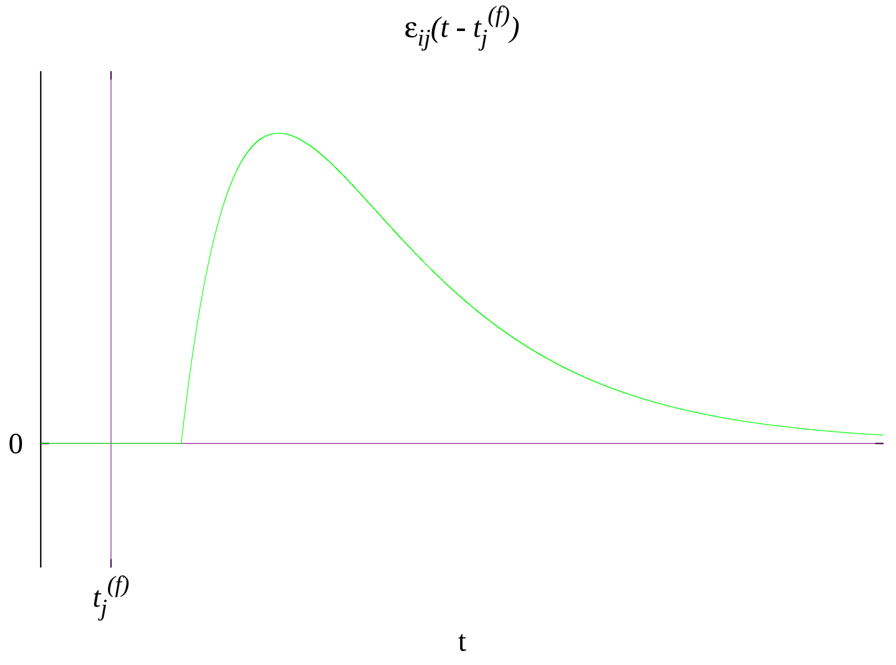


Figure 2.4: Sample ϵ Kernel Function

First, immediately after firing an output spike at $t_i^{(f)}$, the variable u_i is lowered or 'reset.' Mathematically, this is done by adding a negative contribution $\eta_i(t - t_i^{(f)})$ to the state variable u_i. The kernel $\eta_i(s)$ vanishes for $s \leq 0$ and decays to zero for $s \to \infty$; See the sample in the Figure 2.3.

Second, the model neuron may receive input from presynaptic neurons $j \in \Gamma_i$ where

$$\Gamma_i = \{j | j \text{ presynaptic to } i\}. \tag{2.2}$$

A presynaptic spike at time $t_j^{(f)}$ increases (or decreases) the state u_i of neuron i for $t > t_j^{(f)}$ by an amount $w_{ij}\epsilon_{ij}(t - t_j(f))$. The weight w_{ij} is a factor which accounts for the strength of the connection. An example of an ϵ_{ij} function is shown in Figure 2.4. The effect of a presynaptic spike may be positive (excitatory) or negative (inhibitory). Because of causality, the kernel $\epsilon_{ij}(s)$ must vanish for $s \leq 0$. A transmission delay may be included in the definition of ϵ_{ij}; see Figure 2.4.

The state $u_i(t)$ of model neuron i at time t is given by the linear superposition of all contributions,

$$u_i(t) = \sum_{t_i^{(f)} \in \mathcal{F}_i} \eta_i(t - t_i^{(f)}) + \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)}) \tag{2.3}$$

An interpretation of the terms on the right-hand side of (2.3) is straightforward. The η_i contribution describes the response of neuron i to its own spikes. The ϵ_{ij} kernels model the neurons response to presynaptic spikes. We will refer to (2.1), (2.2), and (2.3) as the Spike Response Model (SRM). In the biological context, the state variable u_i may be interpreted as the electrical membrane potential. The kernels ϵ_{ij} are the postsynaptic potentials and η_i accounts for neuronal refractoriness.

2.2.2 Common Variants of the Spike Response Model

We often use a variant of the Spike Response Model. In the following we show some common variants of the model, which is used throughout this dissertation.

Short-term memory We sometimes assume that only the last firing contributes to refractoriness. In this case, we can simplify (2.3) slightly and only keep the influence of the *most recent* spike in th sum over the η contributions. Formally, we make the replacement

$$\sum_{t_i^{(f)} \in \mathcal{F}_i} \eta_i(t - t_i^{(f)}) \longrightarrow \eta(t - \hat{t}_i) \tag{2.4}$$

where $\hat{t}_i < t$ denotes the most recent firing of neuron i. We refer to this simplification as a neuron with short term memory. Instead of (2.3), the membrane potential of neuron i is now

$$u_i(t) = \eta(t - \hat{t}_i) + \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)}) \ . \tag{2.5}$$

External input External input in the neuron model is often considered. In addition to (or, instead of) spike input from other neurons, a neuron may receive an analog input current $\mathcal{I}^{\text{ext}}(t)$, for example, from a non-spiking sensory neuron. In this case, we add on the right-hand side of (2.3) a term

$$h^{\text{ext}}(t) = \int_0^\infty \tilde{\epsilon}(s) \mathcal{I}^{\text{ext}}(t - s) ds \ . \tag{2.6}$$

Here $\tilde{\epsilon}$ is another kernel, which describes the response of the membrane potential to an external input pulse. As a notational convenience, we introduce a new variable h which summarizes all contributions from other neurons and from external sources

$$h(t) = \sum_{j \in \Gamma_i} \sum_{t_j^{(f)} \in \mathcal{F}_j} w_{ij} \epsilon_{ij}(t - t_j^{(f)}) + h^{\text{ext}}(t) \ . \tag{2.7}$$

The membrane potential of a neuron with a short term memory is then simply

$$u_i(t) = \eta(t - \hat{t}_i) + h(t) \ . \tag{2.8}$$

In this dissertation, we suppose tue membrane potential of neurons is modeled by the equation (2.8).

Chapter 3

Phase Arbitration for Binding Representation in Language Understanding

This chapter explores a mechanism of representation in the human brain from a viewpoint of language understanding. We point out that, in order to represent a meaning explicitly on a neural network, a memory mechanism of the network should have some coding more complex than the traditional localist binary representation, preferably a coding with oscillation phases. We also pursue a mechanism required to compose the coding from a linguistic input, and claim an existence of a global mechanism to arbitrate phases.

Discussion in this chapter is rather abstracted. We try to find a common feature of mechanisms that explains the process of language understanding. It turns into a clue to inspecting the human brain for seeking the implementation of the language understanding process.

3.1 Introduction

Memory plays a critical role in human language processing. Since a person cannot process long sentences at once, one divides a sentence into words (or some other units) and processes them word by word, while keeping partially processed results in the brain; it is memory that keeps such information over time. Moreover, the semantic information of the sentence is also supposed to be stored in memory with the partial results. Thus, a model of sentence understanding cannot be described without a model of memory. In other words, we are able to use a sentence-understanding task to justify a model of the human memory mechanism.

However, past studies of natural language processing based on artificial neural networks are not enough to explain the memory mechanism used in sentence understanding. For example, a memory model in a simple recurrent network [8] suffers from a feature binding problem restricting capacity of semantic representation. Although the temporal coding as in Henderson's connectionist parser [17] seems promising, it is still an important open problem to pursue a better model of human memory along this line.

In this chapter, we explore models of the memory mechanism in the human brain from a viewpoint of sentence understanding. Especially, we point out an advantage of the temporal coding in representing binding information, and the necessity of *phase arbitration*, a mechanism that allocates an unused pulse phase to a newly memorized item.

We first show that traditional neural networks are prevented from achieving the task of sentence understanding by the *feature binding problem*. We then investigate possible sources of complexity to be added to the neural network, and show an advantage of temporal complexity. Further, we show the necessity of the phase arbitration in a network involving temporal complexity. We also discuss two models of phase arbitration mechanism, a local model and a global model, and show that the local model is inappropriate to perform the arbitration.

In Section 3.2, the sentence-understanding task is outlined. In Section 3.3, we explain the feature binding problem and possible solutions by various codings. Section 3.4 shows the necessity of the phase arbitration mechanism for temporal coding. Finally, we discuss implementations of the mechanism and its implications in Section 3.5.

3.2 Computational Theory of Language Understanding

At the beginning, we assume the computational theory of language understanding. As described in Section 1.1, we study the possibilities of implementation from the theoretical requirement of language understanding. The computational theory of the language understanding gives the property of the process, which should be satisfied by any algorithm performing language understanding process. The computational theory we discuss in this section is not a complete one, but focused on a critical part of the theory, feature binding and memory.

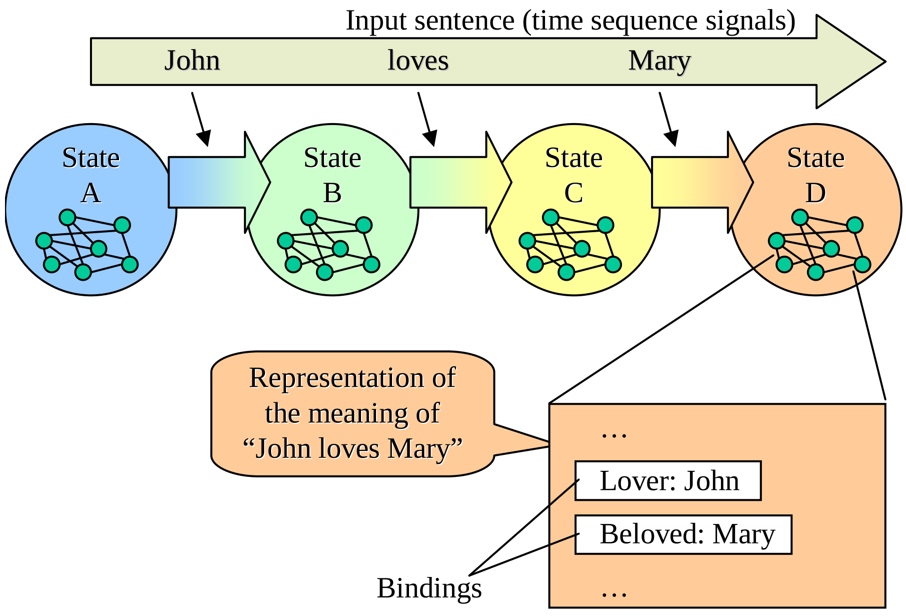

Input sentence (time sequence signals)
John
loves
Mary
State A
State B
State C
State D
Representation of the meaning of “John loves Mary”
…
Lover: John
Beloved: Mary
…
Bindings

Figure 3.1: Language understanding as a dynamical system. The figure of the interconnected nodes represents the language-understanding system.

Language understanding can be regarded as a process, which takes a linguistic expression as an input and produces its meaning. Here we assume that the meaning includes associations with the invariants provided by other perceptual inputs. For example, the process of vision is supposed to extract invariants from the image, such as objects, colors and shapes; we think that the language is useful because it can be related to the invariants provided by other perceptual inputs. Supposing the brain receives a phrase 'blue circle' and produces its meaning in the brain, the meaning should have association to the output of the visional process from a scene with a blue circle. We call the representation of meanings in the brain as *semantic representation*.

We assume that *feature bindings*, or simply bindings, are included in the semantic representation. A meaning usually contains information of relations between entities or concepts. For example, a meaning of a phrase 'blue circle' contains a relation between a concept 'blue' and an entity 'circle', which has a clear distinction from the visual scene containing something blue and a colorless circle. We call this relation binding.

An input of the process is distributed over time. Since a person cannot receive a long sentence at once, one divides the sentence into some units, possibly words, to receive the whole sentence. Moreover, since one can understand a sentence without an explicit end-of-input marker, understanding process should be working all over the time, receiving one unit at a time. In order to integrate the meaning of the whole sentence, the intermediate and final products of the language understanding process should be stored in the processor; it should be, in some form, memorized in the brain. Then we can assume that the intermediate products should also include bindings and association to perceptual inputs as the final product does.

The assumed process is illustrated in Figure 3.1. An implementation of the external verbal input causes the transition of the state. When the system receives the last word of the sentence, it is expected that the state contains the semantic representation of the sentence, including the bindings in the meaning of the sentence.

In the following, we focus on the representation of the bindings on the implementation of the human brain. In order to consist a part of a semantic representation, a representation of binding should satisfy the following principles.

1. *Dynamicity.* A semantic representation is dynamic, that is, available immediately after understanding. Although static memory mechanism (such as change of wiring) may concern background knowledge of semantics, it is too slow to be used in the following processes. A semantic representation and, consequently, a binding representation should be on a more dynamic and flexible medium, such as change of electric potential and functional connectivity. It is expected that the linguistic computation of a short sentence should be finished within a couple of seconds.

2. *Memorability.* A semantic representation is memorable in the brain. Namely, the brain does not understand a sentence without keeping the semantic representation, including bindings, for a certain period. We require the period is substantially longer than the time span of understanding process. It is not desirable that a semantic representation acquired in two seconds is lost in three seconds; we expect it persists for e.g. two

minutes.

3. *Concurrency.* The brain can represent multiple bindings at the same time. A person has an ability to make inference using two or more bindings. We require that a semantic representation can be used for such an inference, in other words, concurrent representation of multiple bindings.

4. *Generalizability.* A binding representation is generalizable to unencountered bindings using the representation of known entities and bindings. A language can easily represent and deliver bindings, which the hearer/reader have never encountered. For example, although few people have heard of either 'flying sandwich', 'noisy coffee' or 'colorless green,' most of the people are able to receive the bindings in the phrases (even if difficult to imagine). We regard that the ability of a language to represent unencountered bindings is the source of the generalizability; without it, human beings cannot communicate new ideas and happenings each other.

Especially the last point constrains a possible coding of a binding representation, which we pursue in the next section.

3.3 Complexity in Memory Coding

This section pursues binding representations in the brain, based on assumptions enumerated in the previous section. The argument is not based on any presumption of a specific mechanism or coding in the brain, such as distributed representation [19] and population coding [14], or the push-down stack[44]. Rather, we discuss the conditions, which must be satisfied by any mechanism that performs language understanding.

3.3.1 Requirement of Additiveness

Binding representations can be classified into *additive* ones and *multiplicative* ones. If the representation of a binding 'A = B' can be composed from an activity of 'A' and that of 'B', the representation is called additive. Otherwise, every binding has a representation depending on a particular activity appearing only on the certain binding, which is called as multiplicative[1].

The generalizability requirement, requirement 4 in Section 3.2, contradicts the multiplicative representation. If the binding representation depends on a particular activity for a certain binding, the brain cannot represent a novel binding; even if it is represented, the brain has no mechanism to decode the representation, since it is a novel representation for the novel binding. Such a system loses generalizability, that is, it cannot understand a novel idea in a sentence.

Thus we can conclude that, in order to perform the sentence-understanding task, a system has to use additive binding representation. With an appropriate mechanism, a system with additive representation is able to generalize a novel idea in a sentence into bindings the system never used before, and still able to use the unencountered bindings for later inference.

[1]This does not mean that the additive representation totally excludes non-compositional representation. For example, a phrase 'blue cheese' has a different meaning from the compositionally constructed meanings from 'blue' and 'cheese'; such a meaning, which depends on a specific binding, may use the representation that cannot be decomposed, even in the additive representation. On the other hand, in the multiplicative representation, any binding representation cannot be constructed without using binding-specific activity.

3.3.2 Feature Binding Problem

The additiveness requirement forces us to face with a feature binding problem [12], as illustrated in Figure 3.2. The recognizer outputs the interpretation of the scene as a set of binary signals. If the output satisfies binding additiveness, the interpretation of a scene with two colored objects becomes a superposition of the representations of two colors and two objects. Then we cannot distinguish which color is bound to which object in the output; this is called a feature binding problem.

It should be noted that a distributed representation [19] also suffers from this problem. We assume that semantic information can be extracted from the distributed representation; otherwise it cannot be regarded as a semantic representation. Then, the extraction mechanism is either multiplicative or additive, depending on the utilization of binding-specific activity patterns. If the distributed representation uses such an activity, it is multiplicative, and lose potential to represent unencountered bindings; otherwise, it is additive, and is caught by feature binding problem.

Although some people argue that a selective attention mechanism solves the feature binding problem in the recognition [50], we don't think the argument is applicable to the language understanding. Since a relation of a noun phrase and its case or role is a binding, a simple sentence with a transitive verb contains two bindings as shown in Figure 3.3. If the selective attention is used to solve the feature binding, the brain can pay attention to only one binding at a time, which makes the brain unable to handle the relation between entities in a sentence, such as John and Mary.

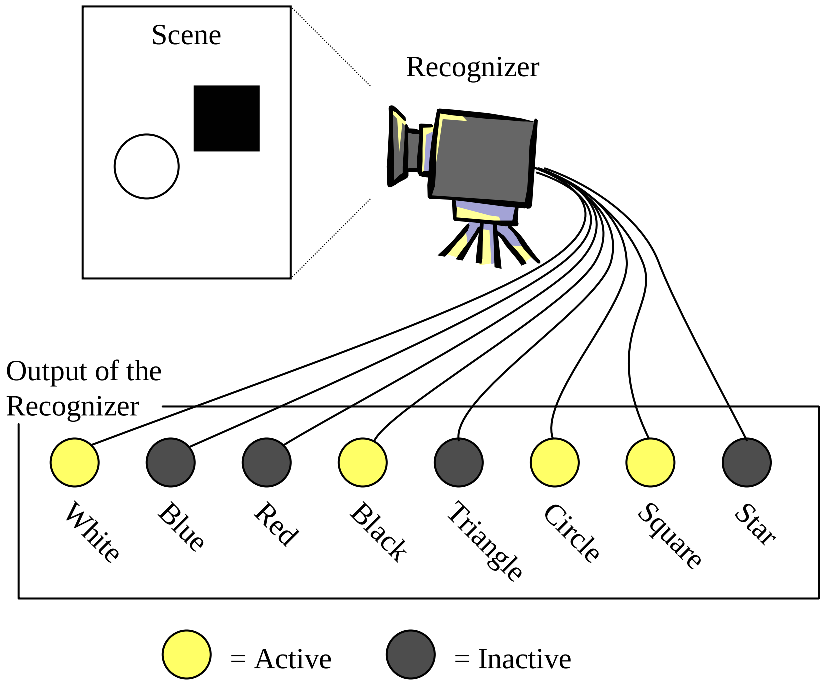


Figure 3.2: Feature binding problem in a scene recognition. A scene with two figures, a white circle and a black square, is posed to a recognizer. Because of additiveness, the output of the system is a superposed representation of 'white' and 'circle', 'black' and 'square'. However, the output is indistinguishable from the recognition of another scene, a black circle and a white square.

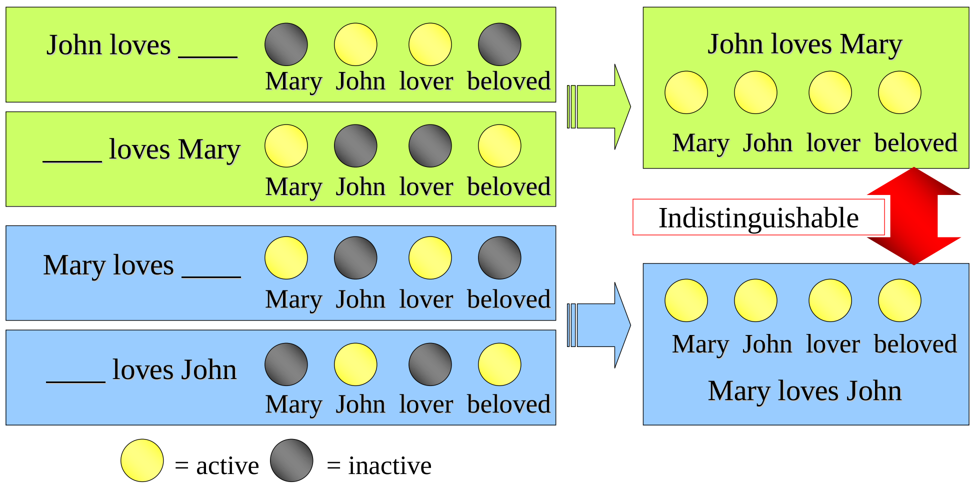


Figure 3.3: Feature binding problem in a semantic representation. Binding of an attribute "lover" and a value "John" is represented as simultaneous activities of "lover" and "John." However, when we try to represent two binding relations, "John — lover" and "Mary — beloved," the activity becomes a mixture of "John," "Mary," "lover," and "beloved," which is not distinguishable from another set of binding "Mary — lover" and "John — beloved." To simplify, we drew this figure with the localist representation, but the problem is not restricted to this representation.

Since a person rarely makes such a mistake of dynamic bindings, some inherent representation that solves this problem should be used in the brain. In Section 3.2, we assumed that the bindings are explicitly represented in the human brain. Moreover, the concurrency requirement, requirement 3 of the sentence-understanding task, states that the brain can represent multiple bindings at the same time[2]. It is clear that the brain uses some representation more complex than a simple set of binary signals, so that the problem is solved.

It is important to explore the representation because the representation characterizes the algorithm in the brain, and consequently, the mechanism of language understanding. In the following, we discuss possible sources of complexity to be incorporated into the binding representation.

3.3.3 Solution in Computers

Modern computers represent bindings by a vector of bits. In one case, each object has its uniquely assigned ID and the ID is expressed in an attribute representation. In another case, an ID (also called as a marker) is assigned to each binding so that binding is represented by passing the binding ID between an attribute representation and an entity representation. Anyway, a vector of bits (e.g. 32-bit length) is used in the binding representation. Indirect memory reference is often used in combination, for example, using a chain of references to represent one binding [2]. This representation makes it possible to keep additiveness while retaining information of bindings. We agree that this way of the binding representation is powerful and efficient. It is also discussed that a

[2] Otherwise a human cannot understand both two bindings at a time, e.g. "John — lover" and "Mary — beloved."

connectionist model based on a bit-vector binding representation [6].

However, we claim that the human brain does not depend on such a relocatable bit-vector representation of bindings. No physiological evidence shows existence of either trunked signal lines or bit-vector comparators. Instead, evidences point to the opposite direction, in which every signal is assigned an individual role. Synaptic plasticity [1] seems dependent only on presynaptic and postsynaptic activity, thus independent of neighborhood activities. Even if some sort of ID or marker is used to represent the bindings, the ID would be encoded in a totally different way.

It is worth pursuing the complexity which is used by the brain in a binding representation. The way of the binding representation characterizes process and memory of information. It would also be deeply related to the mechanism the brain associates the external non-verbal inputs to its internal representation, which constitutes the meaning. In the following, we explore the possible ways of binding representation used in the brain.

3.3.4 Possible Source of Complexity

Binding representations can be classified into three categories according to the complexity used in the representation: space, intensity, and time. We argue the advantage of temporal complexity over other two sources in detail.

Spatial Complexity

The first candidate, spatial complexity, is to use more neurons and synapses to represent bindings. A simple example is to introduce a neuron for each possible binding, such as ‘John=lover’ neuron, ‘Mary=beloved’ neuron and so on. However, this is obviously ‘multiplicative’ representation and violates additiveness requirement.

More sophisticated usage of spatial complexity is to represent IDs by bit-vectors, as just discussed in Section 3.3.3. However, in this dissertation, we pursue possible binding representations in the brain other than bit-vector encoding.

Intensive Complexity

The second candidate, which we name intensive complexity, uses intensity (strength) of signals to store binding information. In other words, multi- or continous-valued signals are utilized as a medium of binding representations, not as a stressed representation of signal appearance. Since a sigmoidal neuron, a mainstream model of neural networks, uses continuous-valued signals, this complexity seemed convincing.

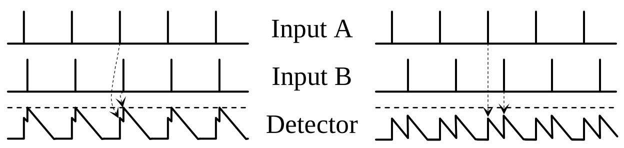
Here we discuss two possible ways of utilization. One is to use signal intensity as a shared binding marker. Bound attributes and objects share the same signal intensity, and different bindings are distinguished by the difference of the signal intensity. The performance of this representation depends on the precision of signal levels; if the signal levels are precise enough to keep eight different levels (including inactiveness), up to seven different bindings can be represented at the same time.

The other is to use signal intensity as a storage of nested information. Suppose that the signal level x is represented in a value between 0 and 1; we can write down the value in a binary digits, $x = 0.x_1x_2x_3x_4\cdots \quad (x_i \in \{0,1\})$. These x_is can be independent storages. Moreover, it is easy to store nested information by shift operators. Dividing x by 2 is equivalent to right-shift operation, that is, $x_{i+1} \leftarrow x_i$; Multiplication of x by 2 is the reversal operation. Combination of these operators can form a push-down stack, which is suitable for handling nested information.

It is notable that Elman [10] is standing on this point of view. He claims that a simple recurrent network trained with center-embedding sentences can generalize the rule to more deeper nestings of center-embeddings, because such a network learns to store information of the embedding (outer) sentence into smaller portion of the value range at the beginning of the embedded (inner) sentence, and to enlarge the portion at the end of the embedded sentence. This is exactly the utilization of signal value precision as a storage of nested information.

We, however, claim that the utilization of signal intensity are not suitable for representing bindings in the brain. In the brain, neurons intercommunicate by spikes, where every spike from a neuron looks alike [14]. This fact tells that the information is conveyed not by the strength nor form of the spike, but by the presence (or absence) and the timing of spikes. There is no reason to add the intensional complexity for a spike.

Although it is said that density of a group of spikes can convey intensive information by rate-coding, we claim that binding representation is not likely to depend on such a coding in the brain. One of the major reasons is the precision. Since binding detectors in the brain have to obtain signal intensity by taking



(a) Two inputs with almost coincident phases. The detector's signal level reaches the threshold and the detector fires periodically.

b) Two inputs with different phases. Detector does not fire at all, since its signal level is always lower than the threshold.

Figure 3.4: Phase Coincidence Detection by an Integrate-Fire Neuron.

an average of the spike rate for a time range, the precision of the intensity gets far from required precision for binding representation. In order to achieve higher precision, the time range for averaging must be increased; however, it must not be longer than several hundreds of milliseconds, or the representation of binding violates the dynamicity requirement, requirement 1 in the sentence-understanding task, and consequently fails constructing a meaning of a sentence within a couple of seconds

It is still possible to construct a language-understanding system with intensive complexity to represent bindings. However, it would be a highly complex and artificial system, far from the human brain. We should pursue another approach first.

Temporal Complexity

The last candidate, temporal complexity, uses temporal position of signals to represent binding information. This focuses on dynamic aspects of the neural network, while the intensive complexity focuses on static aspects. Regarding a timing of an activity as another source of continuous value, we can use similar approaches in the intensive complexity. For example, timing can be seen as an ID of a binding; this leads us to synchronized firing in order to represent a binding.

This seems to violate the memorability requirement of the semantic representation, since temporally transient activities of neurons cannot be kept over time. However, periodic activities such as oscillation can stay unchanged for a certain time. Moreover, a detector of temporal coincidence is easily constructed by neural devices. A single integrate-and-fire neuron can detect coincidence of arriving phases (temporal positions of periodic activity) among multiple neurons with high precision [12], as illustrated in Figure 3.4.

From these arguments, we conclude that the temporal complexity is the first candidate for the brain simulation of the language understanding. In the following, we consider the coding of bindings using the oscillation phases. Note that we do not commit any specific pattern of oscillation, any specific delay of phase coincidence (synchronized or in a specific delay), nor any specific amount of neurons used to represent an attribute or entity. Any binding coding depending on oscillation phases is subject to the following discussion.

3.3.5 Related Work on Temporal Coding

It was known that temporal coincidence of signals can be used to represent bindings. Several studies suggested that temporal correlation of activities may be utilized as a coding in the brain in order to avoid the feature binding problem

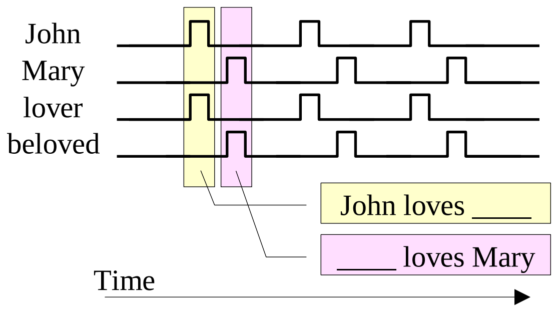


Figure 3.5: Synchrony-based Coding.

[12, 51].

We also have several implementation of the binding representation with temporal coding. One of the simplest implementations is a synchrony-based coding used in SHRUTI system [46]. In their coding, a neuron oscillating by itself denotes either an attribute or a value, and synchronization of the oscillation denotes binding between them (Figure 3.5).

Henderson implemented a connectionist parser based on this coding [17] and succeeded to make a neural network learn to parse by back-propagation through time [18]. His architecture, Simple Synchrony Network, is generally an extension of Simple Recurrent Network by the synchrony-based coding. He notes that the limitation of the synchrony-based coding, e.g. capacity constraint caused by lack of phases, can predict human unacceptability of some sentences.

These studies encourage a model of temporal coding as an additional complexity for binding representations, which constitutes a part of a semantic representation. However, we should be careful that the time itself is used at the input of a sentence. In the next section we discuss the influence of the additional role of time in the context of language understanding.

3.4 Phase Arbitration Mechanism

Although a network with temporal complexity looks quite promising, we found that our binding representation cannot be applied directly to the sentence-understanding task: We have to assign double roles to the time.

In the language understanding process, a sentence is input to the system by distribution over time. Since the input timing is not synchronized to the internal oscillation timing used for the binding representation, some synchronization mechanism seems necessary. It is more difficult than a simple synchronization of internal oscillation and external signal; in order to avoid accidental binding between internal oscillation and external signals, the synchronization mechanism has to be able to assign *unused phase* for the new oscillation caused by the external input. Moreover, since the phases are a finite resource, it is necessary to free a used phase, namely, forgetting.

Due to the different assumption of the computational model, existing studies with temporal coding solve this problem artificially. The SHRUTI system [46] determines every pulse phase by an external signal, and cannot forget items unless the systems are reset to the original state. Henderson's SSN [18] learns to use an unused phase for a new item, but it is based on the teacher signals in back-propagation. Moreover, SSN forgets an item when its syntactic requirement is completed: this contradicts our stance, in which results of the syntactic processing, such as semantic information, is stored in the working memory.

In this study, we name the allocation of an unused phase as *phase arbitration*,

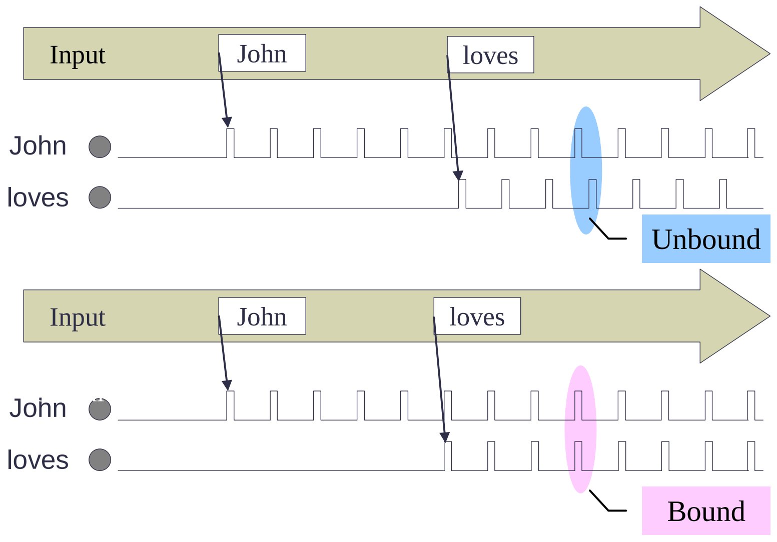


Figure 3.6: Temporal coding without phase arbitration. Two signals (John and loves) are unbound (not synchronized) in the upper figure, and bound (synchronized) in the lower figure. This large difference is caused by the subtle difference of the input timing of 'loves'.

and pursue the way to implement phase arbitration on a temporal-coding neural network. Since the phase arbitration mechanism determines the usage of phases, it will characterize the information processing of the network.

3.4.1 Necessity of Phase Arbitration Mechanism

We need a mechanism to arbitrate phases for stable encoding into temporal oscillation. The mechanism may be very simple; just a single signal is sufficient if it is properly generated and used. However, it is certain that we need a mechanism, which allocates unused phases and assign them to new inputs.

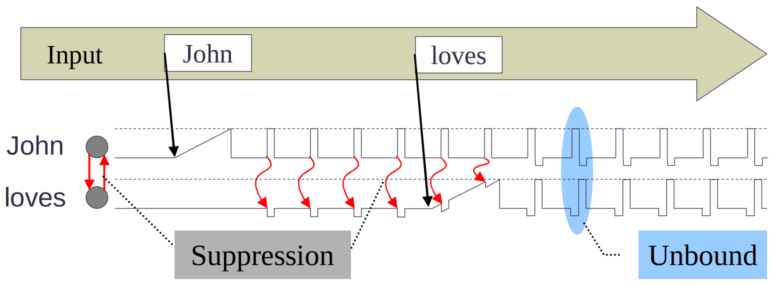
If a neural network has no such mechanism, phase of signals caused by an input word becomes dependent on the input timing. In this case, small turbulence of input timings disturbs the phase to cause collision with another phase used in the representation, resulting accidental binding representation (shown in Figure 3.6). This is not practical, because information coded in oscillation phase becomes unstable. Note that, in our claim, such a mechanism is necessary not only in the simulation of language understanding but also in the brain that performs language understanding. The phase arbitration mechanism is necessary when a language-understanding system uses temporal coding. As we pointed out in Section 3.3.4, it is highly possible that the brain also uses temporal coding. It is important to study the phase arbitration mechanism used in the brain, which characterizes information processing of the brain[3].

3.4.2 Implementation of Phase Arbitration

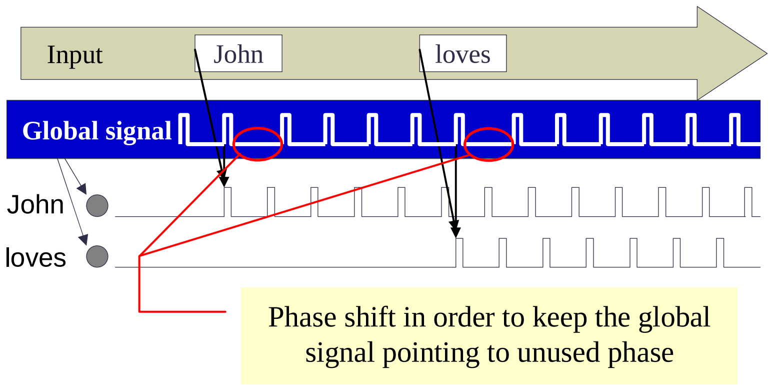
A phase arbitration mechanism can be classified into *local* and *global*. If a phase arbitration mechanism uses some information source globally shared among temporal-coding neurons, it is global; otherwise, it controls phases by mutual connections between temporal-coding neurons, and called as local.

Figure 3.7(a) shows an example of a local phase arbitration mechanism. In the example, memory neurons are mutually connected by inhibitory synapses so that the accidental binding representation is suppressed. The raise of the potential of a memory neuron is controlled to be slow in order to cause firing at just after the inhibitory signals. It is possible that the mutual connections

[3]Here we do not commit whether the mechanism is innate or acquired. However, even if it is acquired by learning, it is worth discussing what kind of mechanism is used after acquisition.



(a) Example of a local phase arbitration mechanism.


Input
John
loves
Global signal
John
loves
Phase shift in order to keep the global signal pointing to unused phase

(b) Example of a global phase arbitration mechanism.

Figure 3.7: Possible implementations of phase arbitration mechanisms.

are excitatory, but the base of the idea is the same.

A global phase arbitration mechanism is illustrated in Figure 3.7(b). The mechanism uses some shared signal that represents a global phase of a network. Since the signal is used to assign unused phases, it is supposed that the signal points an unused phase in some way. In this case, when the pointed phase is used, a phase of either global signal or memory neuron needs to shift so that the global signal points to a new unused phase.

3.4.3 Problem of Local Phase Arbitration

Intuitively, the local phase arbitration mechanism seems more feasible than the global one, because of the distributed style of the information processing in the brain. It is said that the advantage of the connectionist architecture is its parallel and distributed processing manner [19], while the introduction of a global signal seems to sacrifice the parallel processing power.

However, we found that the local mechanism is not suitable for the phase arbitration. When we tried to implement the local mechanism as described in Section 3.4.2, we have to face a number of obstacles. One problem is the difficulty to decide the raising speed of the potential caused by external input. If it is too fast, the firing cannot be controlled during consecutive suppressions; if too slow, dynamicity is lost.

Another problem is the accumulation of inhibition/excitation. When many activities are overlaid in an additive representation, inhibition/excitation caused by mutual connections is accumulated to cause prohibition of necessary activity or induction of unrelated activity (See Figure 3.8. Apart from the problem,

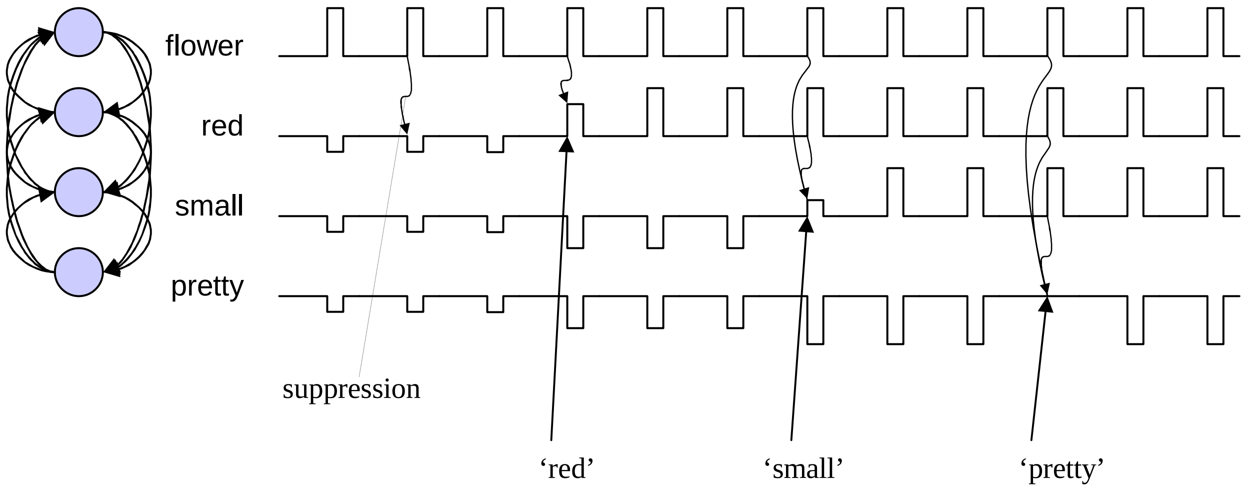


Figure 3.8: Problem of Local Phase Arbitration. Four neurons are interconnected so that accidental synchronizations are suppressed. However, when flower is synchronized to many other attributes, the suppression becomes stronger as the number of synchronizing activity increases, causing binding failure for an attribute pretty.

since the raising speed of the potential is deeply related to the strength of inhibition/excitation, such an accumulation effect makes it more difficult to decide the raising speed. Since a preconfigured network suffers from the difficulty, it is much more difficult to acquire such a configuration by learning[4].

We claim that any local mechanism of phase arbitration suffers from similar problems. Since the oscillation phases can represent bindings between any representation of attributes and entities, the oscillation phases have to be global. It is difficult to avoid the global phenomenon, an accidental binding by a phase collision, using local mechanisms; if the phase collision is solved by a local mechanism, it will fall into a reproduction of global information at each local position, which is equivalent to the sharing of the global information.

Note that the globalism of the mechanism is not limited to the phase coding. Since generalizability of binding representation requires the coding should be able to represent *any* binding of an attribute and an entity, a coding of bindings should have a global property to represent any binding, including spatial and intensive complexity. Thus, accidental bindings of unrelated entities are also common to any coding, which is solved by a global mechanism, specific to the coding. In the case of the temporal coding, the global mechanism turns out to be a shared signal of a global phase.

[4]We should also consider additiveness for the learning of phase arbitration. If the learning is multiplicative (specific to each binding), the brain cannot keep a phase coding for unencountered bindings.

3.5 Discussion

The theory of phase arbitration is only a part of the language understanding process; it is still far from a complete model of language understanding in the brain. In addition, the detail of the phase arbitration mechanism is unknown. We cannot conclude which mechanism is actually used in the brain and which mechanism should be implemented in the language-understanding simulation.

However, we are now able to look into the brain physiology to find a similar mechanism in the brain, because we succeeded to suggest the appearance of the global phase arbitration mechanism. It is expected that the features in the suggestion, such as utilization of a global signal and phase shifting, helps us to find a mechanism, which performs phase arbitration.

Actually, some mechanisms studied in the brain science are similar to global phase arbitration. O'Keefe and Recce [39] report that phase precession occurs in a rat hippocampus. Place-coding cells, which correspond to the current position of the rat, first become active in a specific phase to the Theta oscillation, and then shift their phase gradually to make phase difference to the next activation of other place-coding cells. This mechanism, which is supposed to provide short-term episodic memory, can also be regarded as a global phase arbitration mechanism using Theta oscillation as a global signal. It is possible that the phase arbitration for language is provided in such an episodic memory mechanism, as some research on neurolinguistics [22] suggests the relation between sentence understanding and short-term memory capacity.

However, the mechanism that causes phase precession is not known. Lisman's mathematical model of oscillation subcycles [29] is simple and useful, except the conflict of the direction of phase precession against the recent physiology (Lisman's model goes delaying, while recent studies point to advancing).

No study is known about the memory deactivation mechanism in the brain, except old memories spilling out from the width of Theta oscillation. However, Ono [40] reports that, in a mathematical model of phase precession [29], storage of multiple patterns sharing neurons to be active may cause interference between patterns to deactivate one of the patterns. In a sentence parsing and understanding task, it is likely that a pattern of partial parsing results shares neurons with another partial result that covers the former result, thus this type of interference may occur on human memory. Since deactivation by interference suggests another memory structure different from stack, sentence parsing and understanding based on such a memory structure is worth to be studied in future.

3.6 Summary

We explored a model of human working memory mechanism from a viewpoint of sentence understanding. We found that the temporal complexity is likely to be used in solving the feature binding problem than the spatial and intensive complexity. We also pointed out that the oscillation phase coding based on the temporal complexity poses a new problem to the memory model, i.e. phase arbitration. We discussed the mechanism of phase arbitration and suggested an existence of a global phase arbitration mechanism in the language understanding mechanism in the brain.

Chapter 4

A Discrete-Event Simulator for General Neuron Model

A high-precision and efficient simulator for pulsed neural networks is demanded to verify the model of human language understanding pursued in the previous chapter, in which the importance of the temporal complexity of neuronal activity is revealed. However, existing simulators cannot provide both precision and efficiency, because of the lack of appropriate simulation techniques. In this chapter, we describe techniques for discrete-event simulation of pulsed neural networks, applicable to arbitrary spike-response model neurons with finite discontinuities.

4.1 Introduction

The importance of time in a neural network simulation is increasing. Emerging research areas, such as simulation of memory and context handling in a neural network, are requiring simulation of temporal transitions of the network. Recent studies pointed out that temporal coincidence of pulses has various roles in the brain, including binding encoding [33] and functional connectivity [12]. A high-precision and efficient simulator for pulsed neural networks is demanded for studying temporal behavior of the brain.

Most existing simulators are based on a discrete-time simulation framework (also known as synchronous simulation) [5, 41]. Although this framework is easy to develop, it inevitably requires a large amount of computation to increase temporal precision. If the temporal precision is reduced to achieve efficiency, pulse timings are restricted and the expressive power of temporal coding decreases.

It is widely known that a discrete-event simulation framework, also called event-driven simulation, can simulate a neural network with high temporal precision. Studies on discrete-event neural network simulation were pioneered by Watts [53], and application to a larger network has been investigated by various researchers [16, 37]. However, the neuron model in the existing simulators is restricted to a rather simple class, in which the future transition of the neuron is easily predictable. Techniques to simulate a more complex class of neuron models are thus being demanded by advanced simulation tasks, such as the simulation of the short-term memory model of hippocampus, .

It is known that most of the demanded neuron models can be described by the Spike-Response model (in Section 2.2), whose state is described as a summation of presynaptic pulse-response functions, a self-spike response function and an external input function. This model includes a large class of neurons, such as leaky integrate-and-fire neurons [14]. However, its high expressive power makes it difficult to predict the future behavior of a neuron, especially to detect the nearest threshold-crossing point that corresponds to the next firing time.

In response to the above-described situation, we developed a *second-order incremental partitioning method*, which is a general solver to detect the nearest threshold-crossing point by using linear envelopes of a function and its derivatives. The linear envelopes can be defined for any C_1-class continuous function; even when the function has incontinuities, we can partition the function into continuous parts. Moreover, since linear envelopes of various functions can be summed, this method is easily applicable to a neuron model with any functions splittable into finite ranges of second-order differentiable functions, including the Spike-Response model of a neuron.

We also devised a filtering technique for reducing the cost of the partitioning method. Since the partitioning method is based on prediction of the future, every arrival of a pulse causes recalculation of the prediction, which degrades the efficiency. Our technique, *maximum gradient checking*, effectively reduces the number of predictions by filtering out unnecessary ones prediction by concerning the next known pulse arrival at a neuron, i.e., arrival time.

4.2 Discrete-event Neural Network Simulation

Numerical simulation of neural networks is commonly based on a discrete-time simulation framework. In discrete-time simulation, the temporal transition of neural states are represented in a form of associated differential equations. The values of state variables are then updated synchronously for each time step Δt, using a finite integration method such as Euler or Runge-Kutta. Δt gives temporal resolution of the simulation in a sense that the simulator cannot reproduce dynamics in a time span less than Δt. Since the simulation cost is inversely proportional to Δt, a coarse temporal resolution must be used for large-scale network simulations.

For the simulation of pulsed neural networks, the discrete-time simulation framework is not suitable. To simulate the temporal correlation of pulses, Δt must be significantly less than the correlating pulses, so the performance of the simulation degrades drastically. In addition, when the framework is applied to pulsed neural networks, most of the calculation is a deterministic update of neuron states. In a pulsed neural network, neurons intercommunicate with pulses. The transition of a neuron state between receiving pulses is deterministic. In the case of a fine-grained time step, most of the synchronous updates in discrete-time simulation concern deterministic evolution of neuron states. If this evolution were properly calculated, such synchronous updates could be reduced.

Elaborating this idea, we obtain a different framework of simulation, which is called a discrete-event simulation framework. An arrival of a pulse to a neuron is regarded as an event; the state of the neuron is calculated only at the time an event occurs. This process may cause the neuron to fire, which causes new pulses to be sent, each of which turns into another event. This framework is called discrete-event because it cannot simulate continuous interaction of neurons; that is, it can only simulate a discrete sequence of events. However, it is a suitable framework for pulsed neural networks, in which every interaction of neurons is a discrete pulse.

4.2.1 Discrete-Event Simulation of a Neural Network

Figure 4.1 sketches a discrete-event simulation process with a simple integrate-and-fire neuron model. The simulator keeps information of each neuron as a pair consisting of the last simulation time and the value of the state variable at that time, which are denoted in the figure as ‘Last’ and ‘Sig’, respectively. A scheduling queue keeps pending events in the order of arrival time.

The simulation process consists of the repeated deliveries of the earliest pending event in the scheduling queue to the neuron. In the figure, the event arriving at neuron A at time 5.0 is the earliest pending event; thus it is delivered to neuron A. Then the state of the neuron is updated to the time of the event. In this case, the last simulation of neuron A was at time 4.0, and the state variable at that time was 0.7. As the event arrived at time $t = 5.0$, the state of neuron A is updated to time 5.0: Last becomes 5.0, and Sig is updated to 0.4, i.e., the decayed value at $t = 5.0$. Note that, in this update process, other neurons such as neuron B are kept unchanged. The calculation of the state of A presumes no other pulse arrives at A before that time, although the state of neuron B, which may send pulses to A, is left uncalculated from $t = 3.3$. This is because we know that neuron B never fires unless it receives an external pulse, and pulses for B are absent between the last calculation of the state of B ($t = 3.3$) and the calculation of the state of A ($t = 5.0$). The absence of the pulses is ensured by the scheduling queue, which stores events and serves them

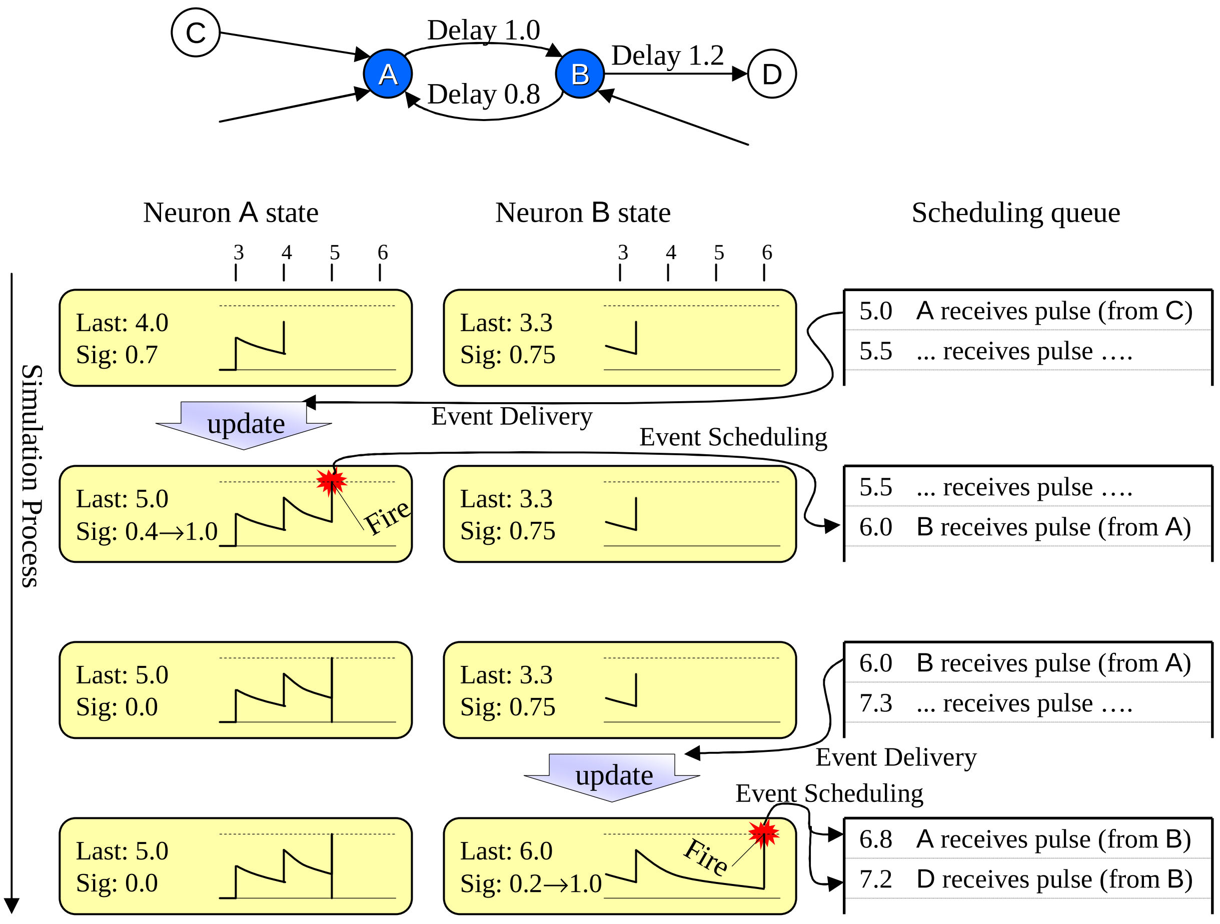


Figure 4.1: Discrete-event simulation model

in order of arrival time. In this way, discrete-event simulation keeps the whole network consistent while minimizing the neuron states to be updated.

Thereafter, the effect of the pulse is added to neuron A, which causes A to fire at time 5.0. As a result, A sends a pulse to neuron B, with a delay of time 1.0. Thus, an event of pulse arrival at B is scheduled at time 6.0. When the event comes to the top of the queue, it is delivered to neuron B, and at that time the state of B is updated. If the event caused firing, then another set of new events is scheduled. In this way, the repeated deliveries constitutes the simulation.

As described above, in a discrete-event simulation framework, the update process of states no longer relies on synchronous processing of neurons in Δt steps, but on calculation based on event arrivals. This advantage makes it easy to achieve high temporal precision efficiently with pulsed neural networks.

4.2.2 Delayed Firing

One remaining problem is the handling of *delayed firings*. In some cases, the effect of an event on a neuron is not instant. In the upper part of Figure 4.2, the pulse itself does not cause immediate firing, but causes the neuron to fire at a later time. The handling of such a firing, which we call delayed firing, poses a problem for discrete-event simulation. Namely, since the neuron state is not calculated until the arrival of the next event, the delayed firing is 'ignored' until the arrival of the next event. If the pulses produced by the delayed firing are not simulated in order of arrival time, the causality of the simulation system is violated.

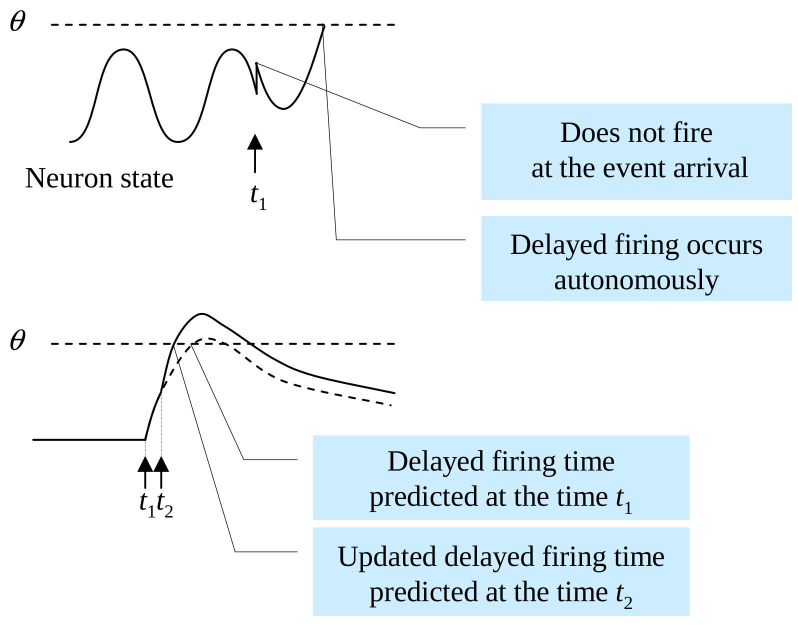

θ
Neuron state
t_1
Does not fire at the event arrival
Delayed firing occurs autonomously
θ
t_1 t_2
Delayed firing time predicted at the time t_1
Updated delayed firing time predicted at the time t_2

Figure 4.2: Delayed firing of a neuron. The upper part shows a simple sine function with an immediate response for the pulse at time t_1. The lower part shows spike-response functions for the pulses at t_1 and t_2. In the latter case, the first prediction of the firing time at t_1 is changed by another pulse arrival at t_2.

In a general neuron model such as the Spike-Response model, delayed firing is not a special case. If a response function such as the one in the lower part of Figure 4.2 is used, a firing is always delayed from the last pulse arrival. Moreover, a superposed response function from a later arrival of another pulse causes the change in the delayed firing time. Such a change poses more difficulty for the simulation.

To avoid this problem, delayed firing has to be scheduled in the pending event queue, which requires prediction of the precise timing of the delayed firing when the previous event is processed. This firing prediction is undoubtedly the key to precise simulation of pulsed neural networks. However, it is difficult to predict firing for a complex neuron model such as the Spike-Response model, as described in the next section.

4.2.3 Difficulty of Delayed Firing Prediction

Simulating complex neuron models, including the Spike-Response model of pulsed neural networks, are demanded in neural modeling of human memory and high-level information tasks using human memory[33]. Such a neuron model is described by a summation of a number of functions of time t, including exponential and trigonometric functions. However, it is difficult to predict the time of delayed firing for such a neuron.

The difficulty is caused by the mathematical complexity involved in finding the time of delayed firing. Even if we can give a functional expression to the state variable $u_i(t)$, it is different from finding roots of the equation $u_i(t) = \theta$, which gives the firing time. Analytical methods for finding a root are restricted

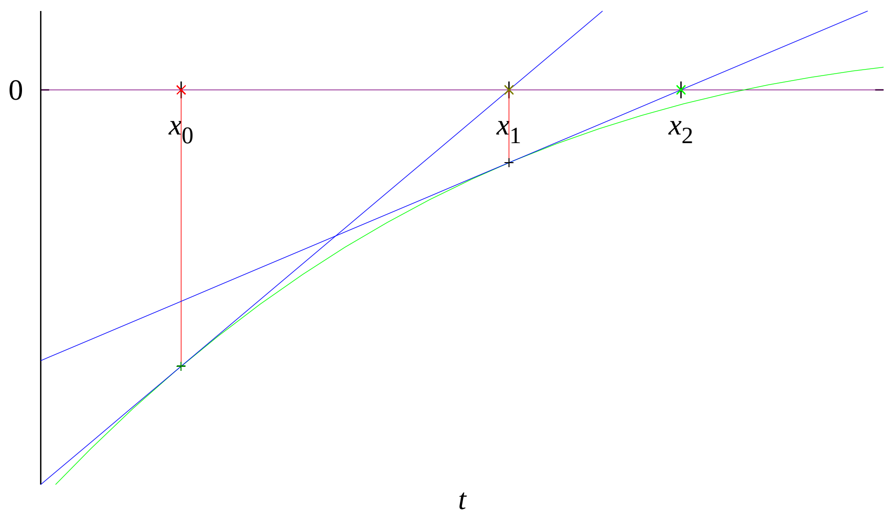


Figure 4.3: Newton-Raphson method finding a root. Starting at the point x_0, the method calculates a tangent line of the function at x_0 and moves to the intersection of the line and the x-axis, which is x_1. One more application gives another point x_2, and repeating this process numerically gives the crossing point of the function and the x-axis.

to simple functions, such as a linear function and a simple exponential function. In general, we cannot analytically find the roots for an equation that is a summation of several exponential and trigonometric functions; it is more difficult than finding roots of higher-order polynomial equations.

However, we can solve such an equation numerically. The Newton-Raphson method is one of the best-known and most powerful methods to give a numerical solution to an equation. Figure 4.3 illustrates the process. Basically, in solving an equation $f(x) = 0$, the method repeatedly moves variable x to a crossing point of the x-axis and the tangent line of $f(x)$ at point x until x converges on a root.

Although the simple application of the Newton-Raphson method sometimes fails to find a root, it is known that the Newton-Raphson method combined with the bisection method can safely find a root if we enclose the root in a range [43]. Here, *enclosing* means finding a range (x_1, x_2) for a function $f(x)$ in which the values $f(x_1)$ and $f(x_2)$ have the opposite signs; at least one root exists in the range because the function is continuous. Since the bisection-combined Newton-Raphson method is applicable to any differentiable function, it is suitable to find a root of $u_i(t) = \theta$, where u_i is a sum of differentiable functions.

Nevertheless, the method is still incomplete; i.e., it cannot predict the delayed firing time. Figure 4.4 illustrates a situation comprising a sum of a linear function and a sine function. A prediction algorithm of the delayed firing time should correctly find the first point beyond the threshold, which is time t_0 in the figure. However, we cannot control which root is calculated by the Newton-Raphson method; namely, the method may converge to any root, such as t_S, the second crossing of the threshold.

For accurate simulation we have to guarantee that the solver finds the first threshold-crossing point. However, it is difficult to distinguish it from *false crossing*, such as t_F, where the state variable approaches but does not go beyond the threshold. When the solver finds t_0 as a root, how can we guarantee that all previous approaches to the threshold are all false crossings? This is a difficult question for a Spike-Response-model neuron, because many exponential and trigonometric functions are superimposed to form its state function. The next

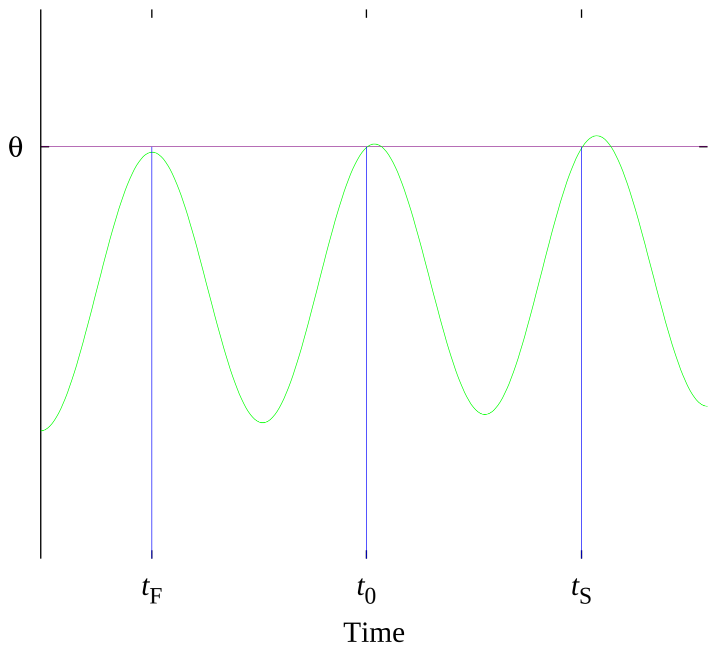


Figure 4.4: Difficulty in finding the firing. We want to find t_0, the first threshold-crossing time. However, it is difficult for a solver to distinguish it from t_F, where the state variable approaches but does not go beyond the threshold, and from t_S, the second threshold-crossing time.

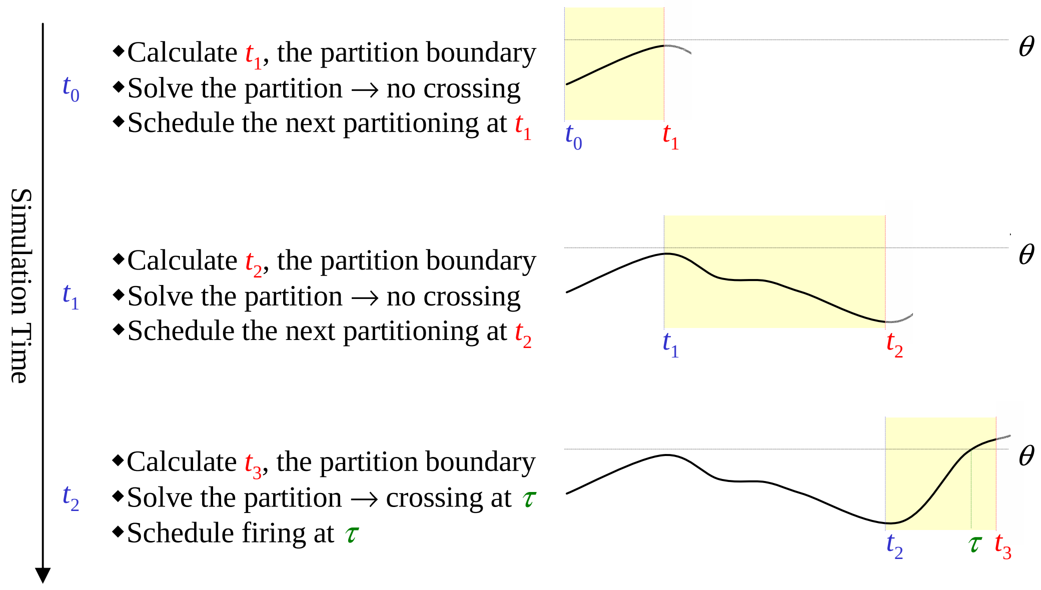

Simulation Time
t_0
◆Calculate t_1, the partition boundary
◆Solve the partition → no crossing
◆Schedule the next partitioning at t_1
t_1
◆Calculate t_2, the partition boundary
◆Solve the partition → no crossing
◆Schedule the next partitioning at t_2
t_2
◆Calculate t_3, the partition boundary
◆Solve the partition → crossing at τ
◆Schedule firing at τ
θ
t_0
t_1
t_2
τ
t_3

Figure 4.5: Incremental partitioning method. Blue denotes the current simulation time, red denotes the end of the partition, and green denotes the time of delayed firing.

section describes our method to solve this problem.

4.3 Incremental Partitioning Method

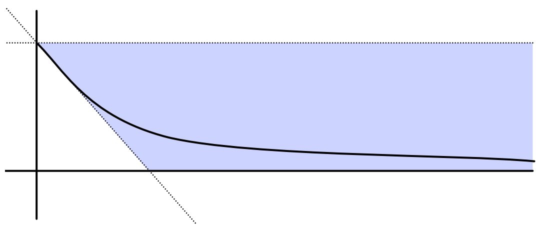
4.3.1 Overview of the Incremental Partitioning Method

Partitioning is a simple idea to solve the difficulty concerning the Newton-Raphson method. We divide the function into *partition*s, each of which has at most one threshold-crossing point. After that, we check each partition to see whether it has a crossing point, and apply the Newton-Raphson method for the first partition containing the crossing point. This method can find the first crossing point, i.e., the time of the delayed firing, without mistakenly finding the second and later crossing points.

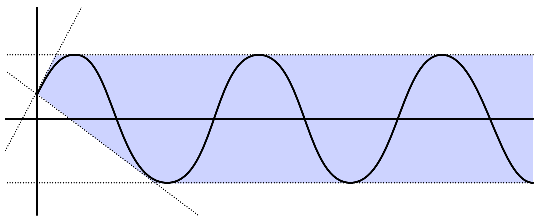
In the simulation, all partitions do not need to be solved at once; they can be calculated and solved one partition at a time. Figure 4.5 illustrates this process. When a partition containing the current simulation time t is solved but no crossing in the partition is found, calculating and solving the next partition can be postponed until the simulation time reaches the end of the partition. The postponement is done by scheduling the solution of the next partition as an event. We call this method *incremental partitioning*.

This method is suitable for discrete-event neural network simulation for the following reasons. First, scheduling of the next partitioning can be implemented in a consistent way with scheduling of other events, such as firing and pulse arrival. Second, it uses more computing power for the near future; since a new arrival of pulses easily changes the state of the neuron, it is often redundant to predict firings in the distant future.

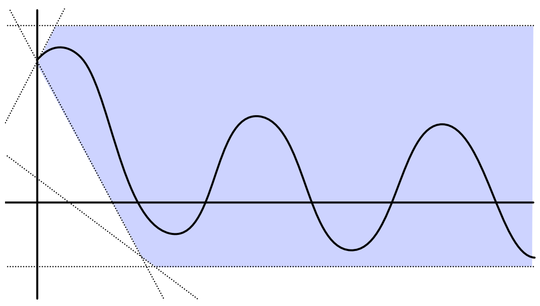
The remaining problem is providing an algorithm for partitioning. If this method requires too fine-grained partitioning, the discrete-event simulation will lose its advantage over discrete-time simulation. The rest of this section describes the partitioning algorithm, which uses *linear envelopes* of the function.



(a) A linear envelope for an exponential-decay function



(b) A linear envelope for a sine-wave function



(c) A linear envelope for the summed function of the above two, composed of the linear envelopes shown in (a) and (b)

Figure 4.6: Linear envelopes for non-linear functions. Although the composed envelope shown in (c) is looser than the envelopes shown in (a) and (b), it correctly encloses the function.

4.3.2 Linear Envelopes

To perform partitioning efficiently, we calculate *linear envelopes* of functions to estimate the range of function values. In short, a linear envelope provides a convenient way to cover possible values of a function with a linear region. Since a linear envelope of a sum of functions can be easily composed from linear envelopes of addend functions, we can cover a complex summed function with a linear envelope.

Linear envelope $\mathcal{L}(f, t_0)$ of function $f(t)$ is a region, whose edge is a set of linear equations and contains any point $(t, f(t))$ such that t is greater than a given starting point t_0. Figure 4.6 shows examples of linear envelopes. In Figure 4.6(a), an exponential decay function is enclosed by a linear envelope consisting of three linear inequality expressions (shown as dotted lines). In Figure 4.6(b), a sine wave function is enclosed by a linear envelope consisting of four inequality expressions. Note that a linear envelope is not unique, even if $f(t)$ and t_0 are given.

It is notable that we can easily compose a linear envelope for a summed function of several nonlinear functions from the linear envelopes of the addend functions. Figure 4.6(c) shows a composed linear envelope of a function, which is a summation of the above two functions. This property enables us to calculate linear envelopes for many complex functions.

In the simulator PUNNETS, the linear envelopes are calculated using tangent gradients and their approximations. See Section 4.9for the actual formulas used in the PUNNETS system.

In the following, we also use linear envelopes of the derivatives of a function.

We call a linear envelope of a first-order derivative a *first-order linear envelope*, and that of a second-order derivative a *second-order linear envelope*. In need of distinction, we call a linear envelope of a non-derived function a *zeroth-order linear envelope*.

4.3.3 Incremental Partitioning with Linear Envelopes

It is certain that a function never crosses a threshold when the threshold is out of a linear envelope of the function. We can thus partition the function at the first point where the linear envelope touches the threshold. As illustrated in Figure 4.7, repeated application of this process constitutes incremental partitioning, which we call *zeroth-order incremental partitioning*.

Note that this partitioning never produces a partition that contains threshold-crossing. The closer the threshold-crossing is, the smaller the partition becomes; we never reach the threshold-crossing, as when Achilles could not catch up the turtle. One solution to escape from this paradox is to introduce a minimum partition size Δt; in other words, fallback to discrete-time simulation. Such fallback often degrades the simulation efficiency.

A more sophisticated partitioning method uses linear envelopes of the derived function. The derived function never reaches zero in a range that the linear envelope of the derivative never touches zero; in other words, the function either monotonously increases or monotonously decreases in the range. Thus, if we partition the function in the range, the partition will have, at most, one threshold-crossing point. Moreover, we can see the existence of the threshold-crossing by checking the signs of function values at both ends of the partition; if

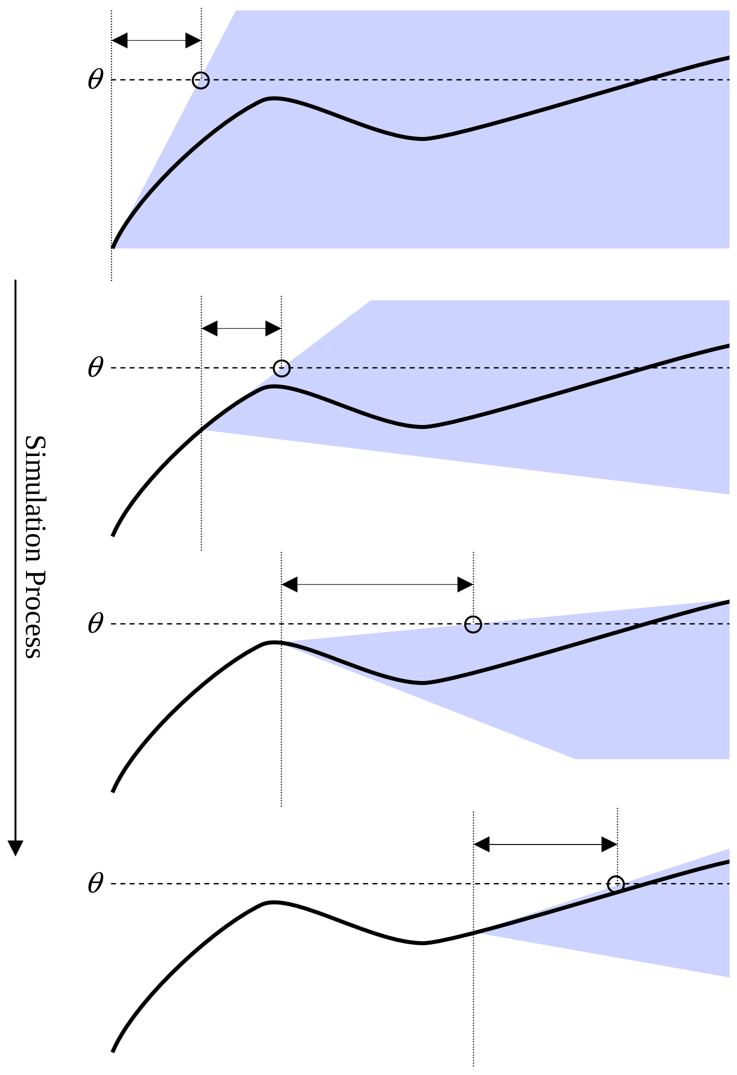


Figure 4.7: Zeroth-order incremental partitioning. The arrows denote ranges of the partitions. The end of a partition is given by an intersection point of the envelope edge and the threshold (indicated by a small circle), which is in turn the start of the next partition.

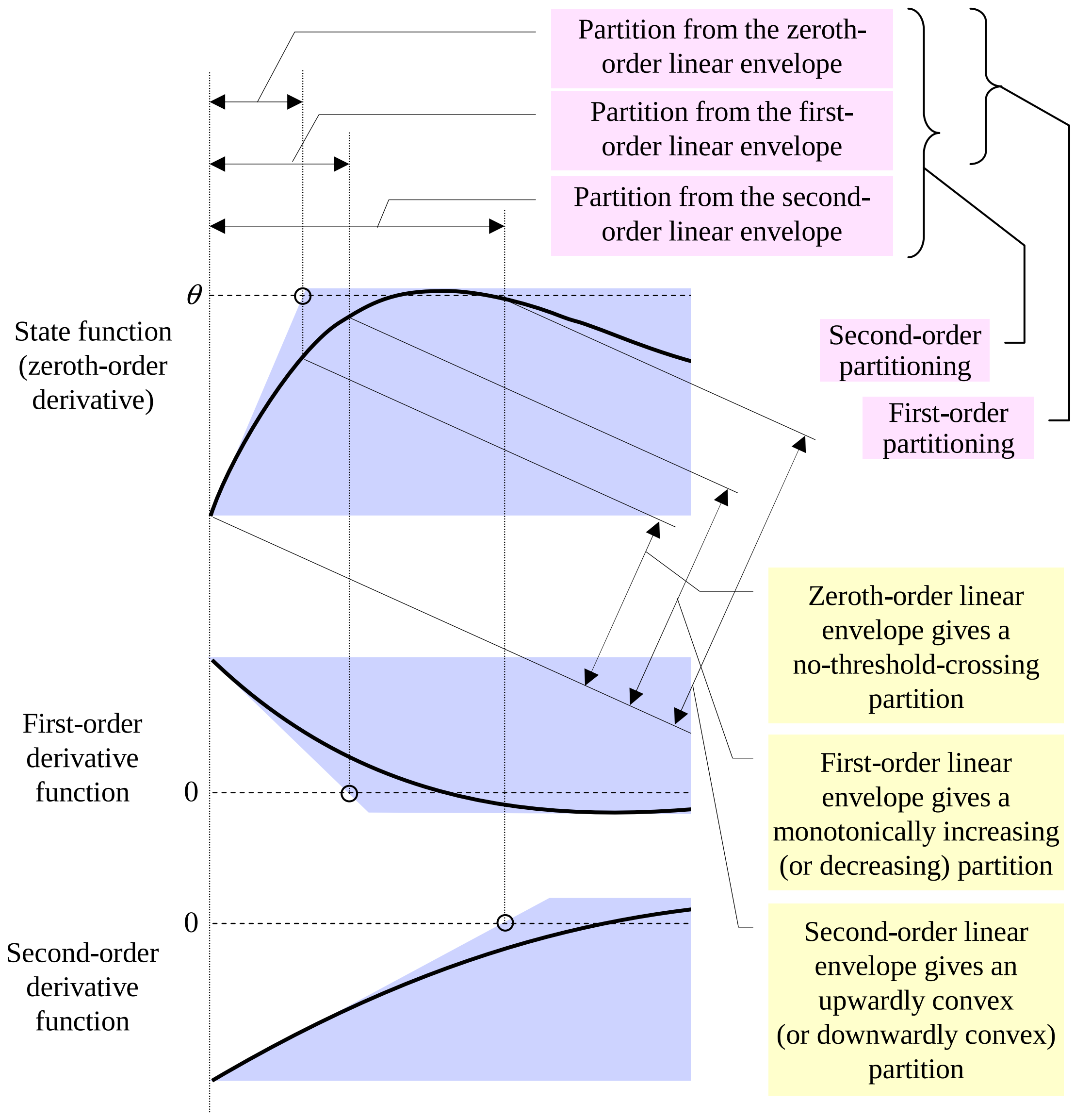

Partition from the zeroth-order linear envelope
Partition from the first-order linear envelope
Partition from the second-order linear envelope
Second-order partitioning
First-order partitioning
θ
State function (zeroth-order derivative)
First-order derivative function
0
0
Second-order derivative function
Zeroth-order linear envelope gives a no-threshold-crossing partition
First-order linear envelope gives a monotonically increasing (or decreasing) partition
Second-order linear envelope gives an upwardly convex (or downwardly convex) partition

Figure 4.8: First-order and second-order incremental partitioning methods. First-order partitioning uses larger one of the above two partitions, while second-order partitioning uses the largest of the three partitions.

the signs are opposite, a threshold-crossing is in the partition, and at the same time, the crossing is *enclose*d in the partition so that the bisection-combined Newton-Raphson method is applicable. As shown in Figure 4.8, the *first-order incremental partitioning* uses two linear envelopes, a linear envelope of the state function and a linear envelope of the derivative, and uses a larger partition from two envelopes; the method reverts to the minimum partition size Δt as before, but it relies less on the Δt fallback.

We can enlarge this approach to second-order linear envelopes as *second-order incremental partitioning*, which uses the largest partition obtained from the three linear envelopes. In a partition where the second-order derivative never touches zero, the function is either upward convex or downward convex, as shown in Figure 4.8. If the function values of the both partition ends have opposite signs, we can apply the enclosed Newton-Raphson method safely. However, the problem occurs in the case with the same signs, as shown in Figure 4.9(a), since the partition may have either zero or two threshold-crossings. In this case, we first discriminate the existence of the threshold-crossings by enclosed peak searching with parabola approximation [43] (see Figure 4.9(b)). If the peak is beyond the threshold, we can enclose the crossing between an end of the partition and the peak; otherwise, it is analytically discriminated that the partition has no crossings. Because of the convexness of the function, the discrimination can be finished before the real peak is found (see Figure 4.9(c)).

It is noteworthy that the effects of the three envelopes are complementary. When the function value is far from the threshold, the zeroth-order linear envelope usually makes the best and the largest partitioning. In the case that the

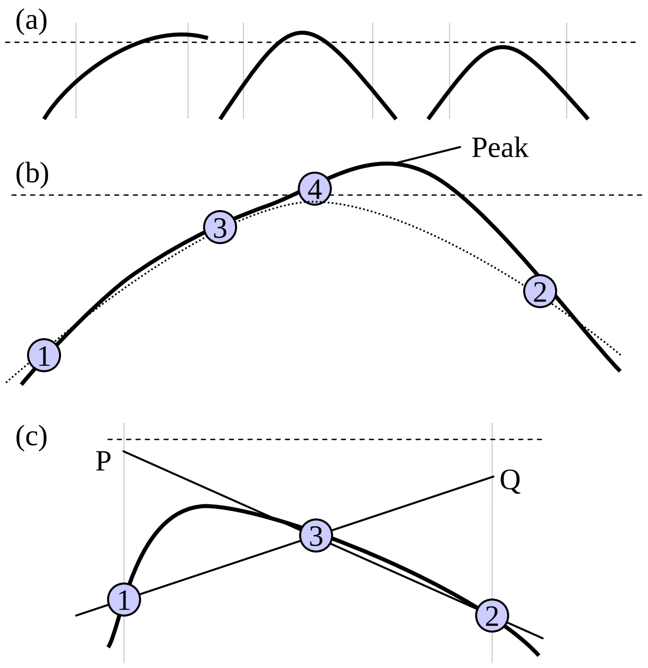


Figure 4.9: Discrimination of threshold crossing. (a) Existence of threshold-crossing. Signs of the function at both ends of the partition show the left case has crossing, but cannot discriminate the middle and the right cases. (b) Enclosed peak searching. The triplet of points 1, 2, and 3 are said to be enclosing the peak ($f(x_1) < f(x_3)$, $f(x_3) > f(x_2)$, $x_1 < x_3 < x_2$). Parabola approximation (dotted line) suggests the peak point as 4, so we can narrow the enclosing to points 3, 2, and 4. The peak can be found by repeating this process. Actually point 4 is above the threshold, thus no more peak searching is required (crossing is enclosed between 3 and 4). (c) Discrimination of threshold-crossing using convexness. Since both points P and Q are below the threshold, this function has no threshold-crossing in the enclosed region. In this case, we can safely abort the enclosed peak searching.

function value is close to the threshold and the gradient is large, a first-order linear envelope gives the large partition that encloses the crossing. If the function value is close to the threshold and the gradient is also small, the second-order linear envelope will make a partition that contains the convex curve to be solved by peak search. Since it is theoretically possible that all three approaches may fail to produce a good partition, it is still necessary to revert to the minimum partition size Δt; however, this rarely happens in actual simulation.

4.3.4 Applicability of the Incremental Partitioning Method

The incremental partitioning method uses linear envelopes. For calculation of a second-order linear envelope, the function must be second-order differentiable. Note that any second-order differentiable function satisfies C_1-class continuity, that is, the requirement of the Newton-Raphson method.

Moreover, to perform partitioning effectively, the vertical range of linear envelopes at a given starting point t is expected to converge to point $(t, f(t))$. This ensures that the linear envelopes give better prediction for the nearer future.

These requirements can be relaxed by introducing additional partitions. For example, if a function with incontinuities can be split into finite ranges of continuous functions by additional partitions, the function can be handled by the incremental partitioning method. Some functions such as $f(t) = t^2$, which is unable to maintain convergence of the linear envelopes to the starting point, can be split by additional partitions to satisfy the convergence expectation.

As a result, the incremental partitioning method can be applied to any function splittable into finite ranges of second-order differentiable functions. Although the method is unapplicable to some ill-natured functions (such as a function with an infinite number of incontinuities in a finite range), we can say the method covers any arbitrary function for the purpose of neural network simulation.

4.4 Efficient Simulation Techniques

The previous section introduced the incremental partitioning method, which predicts the delayed firing for a neuron model with practically any arbitrary function. However, naive application of the method causes inefficient simulation. Since the prediction is based on an assumption that no further pulses arrive, it has to be updated each time a new pulse arrives at the neuron. This degrades the performance of the simulation. Moreover, the update of the prediction changes the time of the scheduled events, which stresses the scheduling mechanism.

We have developed an efficient technique that solves these problems: *maximum gradient checking.* It utilizes the next known pulse arrival to suppress redundant predictions. It also suppresses the changes to scheduled events, so it reduces the simulation cost.

4.4.1 Quick Filtering

The incremental partitioning algorithm predicts the delayed firing time of a neuron in the case that the neuron receives no more pulses after the last deliv-

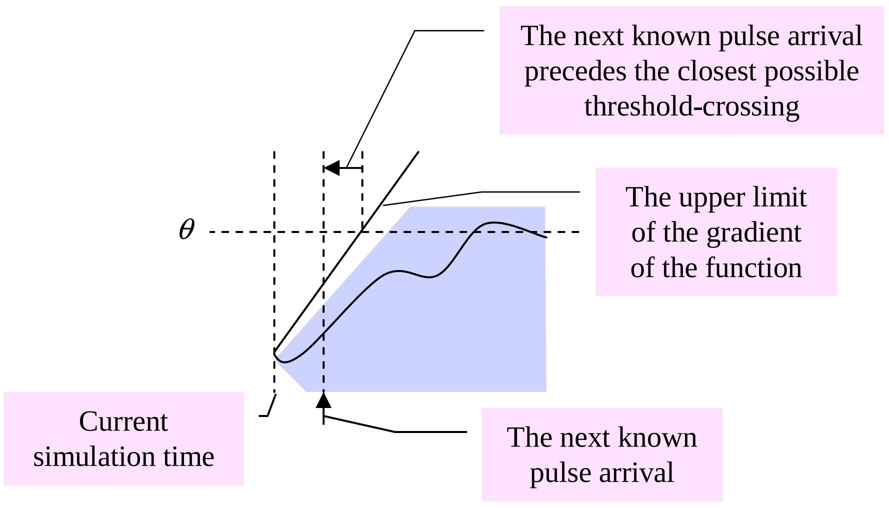

The next known pulse arrival precedes the closest possible threshold-crossing
The upper limit of the gradient of the function
θ
Current simulation time
The next known pulse arrival

Figure 4.10: Filtering a redundant prediction by gradient-limit checking

ered event. However, it is often the case that the next pulse arrival is already scheduled but not delivered yet. Utilizing this information, we can decrease the cost of the prediction.

Quick filtering is a technique that uses the time of the next-known-pulse arrival to filter out unnecessary predictions. The prediction based on linear envelopes can be suppressed if we can confirm that no threshold-crossing occurs till the arrival of the next known pulse. In such a case, it is not necessary to schedule the end of the partition as an event, since the state of the neuron is anyway recalculated at the time of the next-known-pulse arrival. If the confirmation is efficient enough, the decrease of the cost of prediction and scheduling exceeds the additional cost of confirmation.

For this purpose, we introduce *zeroth-order linear envelope checking* and *gradient limit checking*. The work of zeroth-order linear envelope checking is to check the precedence of the next-known-pulse arrival by using a zeroth-order linear envelope; in this case, the calculation cost of first- and second-order linear envelopes can be suppressed. However, to reduce the cost of calculating zeroth-order linear envelopes, we introduce a quicker checking method that uses the upper limit of the gradient of the function for quick checking. Since the upper limit of the gradient is a constant for each function, it can be calculated before starting simulation. Moreover, the upper limit of the summed function can be easily calculated by summing up the upper limits of the gradients of component functions.

Figure 4.10 shows an example of our quick checking method. When the next known pulse arrival is close to the current time (which is often the case in handling pulse bursts), the pulse arrives before the gradient upper limit line reaches the threshold. The filtering technique thus reduces the cost of re-scheduling as well as the cost of calculating linear envelopes.

4.4.2 Queuing Model for Quick Filtering

Many discrete-event simulators have a single queue to schedule all events, e.g., pulse arrivals. However, in a single-queue model, it is difficult to find the next pulse arrival for a specified neuron. To apply the quick-filtering technique, another queuing model should be used in order to allow a quick retrieval of the information of the next known pulse arrival.

To meet this requirement, we introduce another queuing model, in which each neuron has a local event queue. The local queue of a neuron holds all pending events that affect the neuron. A main schedule queue keeps neurons sorted according to the first event time of their event queues. In this model, the next event of a neuron can be easily found at the top of the neuron's local event queue.

Note that this change of queuing model does not increase the order of scheduling cost. The complete binary tree (heap tree) algorithm, which is a most popular and empirically efficient algorithm for a priority queue [21], needs a cost of $O(\log n)$ for insertion and retrieval of an entry, where n is the entries in the queue. Suppose a neural network has N neurons and each neuron has ν pending events. In the single-queue model, the insertion/retrieval cost is $O(\log \nu N)$. On the other hand, in the object-queue model, we generally need to insert both the neuron's queue and the main queue, which keep ν and N entries, respectively. The total insertion/retrieval cost is $O(\log \nu + \log N)$, whose order is equivalent to $O(\log \nu N)$, the cost of the single-queue model.

4.5 Implementation

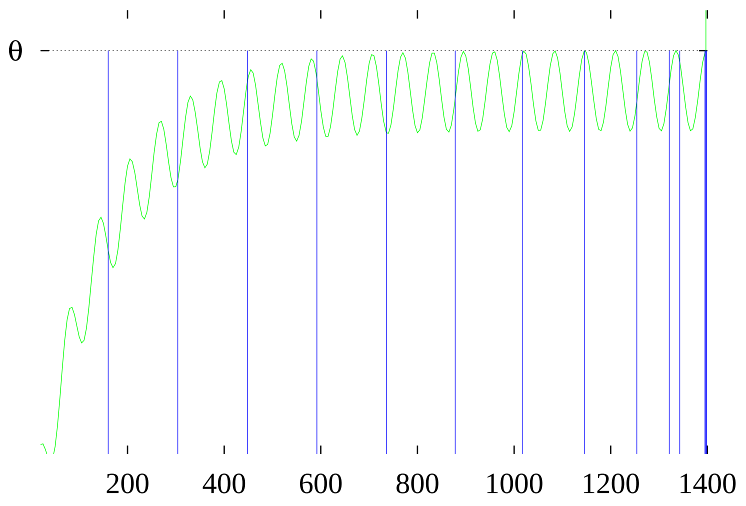
We implemented PUNNETS[32], the pulsed neural network simulator, using the techniques described in this chapter. The simulator is a 3000-step C++ program library, which is highly object-oriented and easily used by C++ programs. PUNNETS has a class that simulates any neuron based on Spike-Response model, as well as an optimized version of classes simulating an integrate-and-fire neuron with a dynamic threshold. Since neurons and synapses are designed as an object, a user can use various styles of neurons and synapses, including stochastic neurons and dynamically learning synapses. The library also has a logging ability to record the behavior of neurons as either event reports or state graphs.

4.6 Experiments

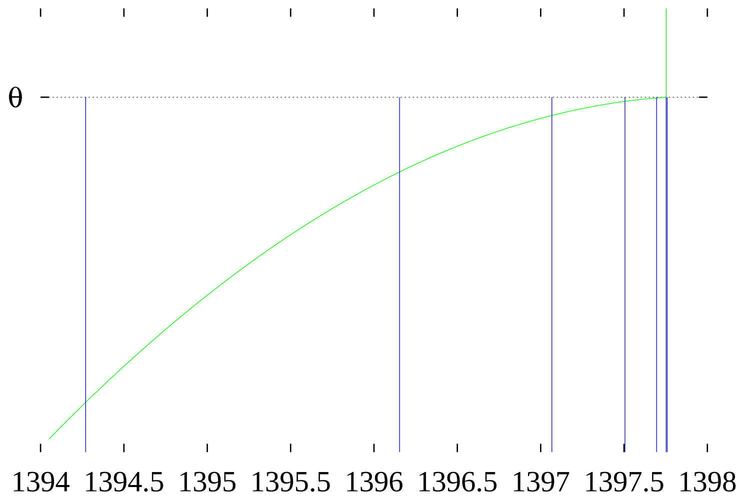
We performed a series of experiments to prove the efficiency of the incremental partitioning method and the quick filtering technique. In the experiments, we used 10^{-8} as the value of ε (the minimum movement of x for one iteration of Newton-Raphson method).

Figure 4.11(a) shows the zeroth-order incremental partitioning on a summed function consisting of a sine function and an exponential function. In this figure, the simulator makes 19 partitions, although the later partitions are too narrow to see. The last eight partitions are enlarged in Figure 4.11(b). Before the Δt-cutoff is used, the distance between the function and the threshold reaches less than the epsilon value and causes firing. If we use first- or second-order incremental partitioning, the area shown in Figure 4.11(b) is partitioned into only one partition. In this case, nine iterations of the Newton-Raphson method correctly find the firing time. The gaps between firing times of zeroth-, first-, and second-order partitioning are less than 10^{-8}. It is clear that the precision achieved by discrete-event simulation outperforms discrete-time simulation, which requires 10^{11} synchronous updates to achieve 10^{-8} precision in a 10^{3} temporal range.

Note that the number of iterations in the Newton-Raphson method (nine times) was almost the same as the number of partitions in zeroth-order partitioning (eight times). This is because, in this local range, the gradient of the edge of the linear envelopes is near to the tangent of the function, so that



(a) Incremental partitioning for the sum consisting of an exponential function and a sine function



(b) Enlarged feature around the threshold-crossing point of (a)

Figure 4.11: Application of zeroth-order incremental partitioning

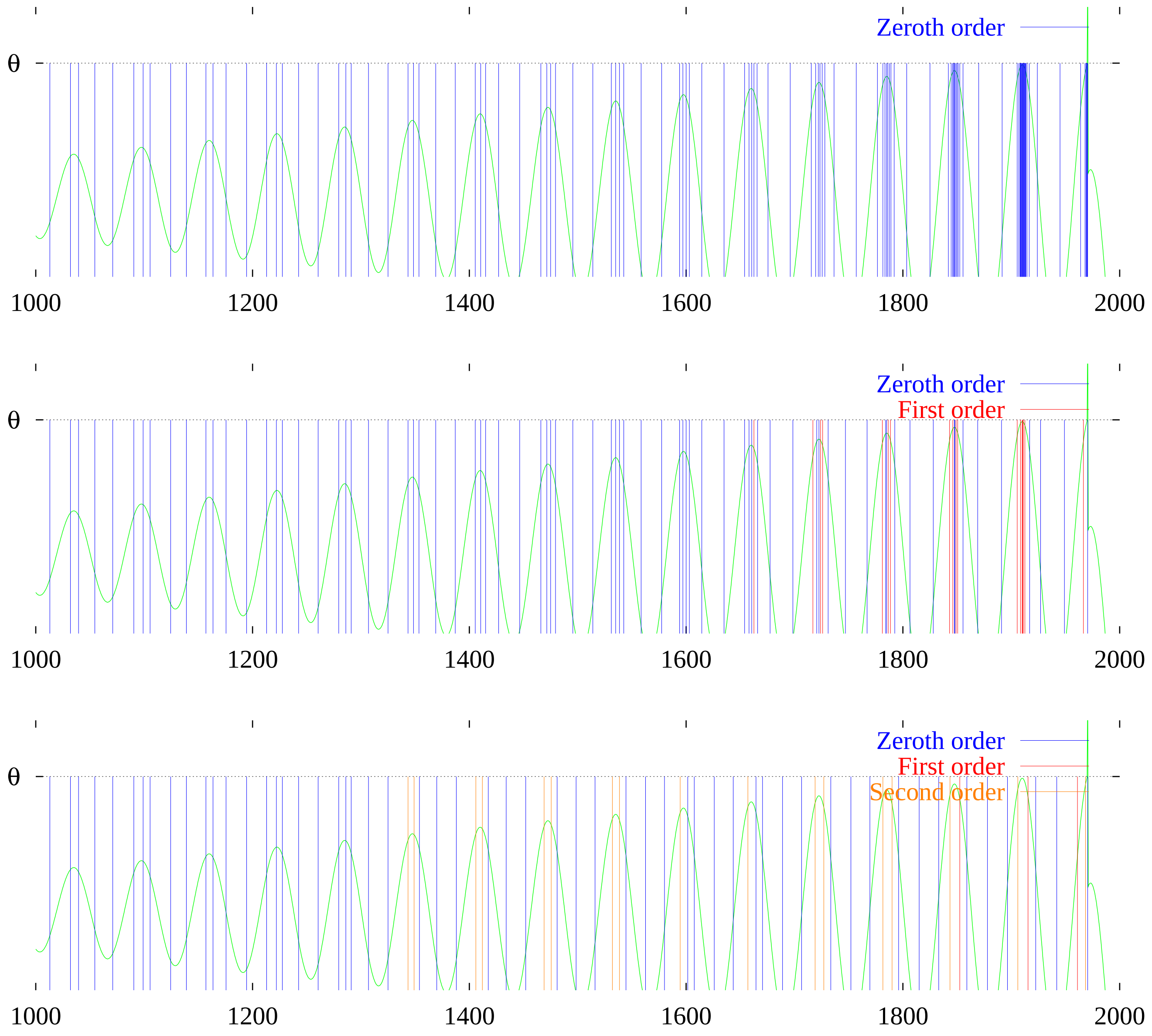


Figure 4.12: Order difference of incremental partitionings

the zeroth-order partitioning makes the same movement step as the Newton-Raphson method does. However, higher-order partitioning has the advantages that, first, zeroth-order partitioning lacks the general solving power of an equation; second, the cost of an iteration of Newton-Raphson method is lower than the cost of a partitioning.

Figure 4.12 shows the simulation result from a neuron model, that is, an addition of two sine waves with slightly different wavelength. This condition corresponds to one of the worst cases, since the composed linear envelope of the function becomes much broader than the actual range of the function.

Figure 4.12(a) shows the result from zeroth-order incremental partitioning. Despite the broad linear envelope, the simulator reaches the firing time after 152 partitions. Higher-order partitioning achieves better results: first-order partitioning shown in Figure 4.12(b) requires 97 partitions, and second-order partitioning shown in Figure 4.12(c) requires only 76 partitions to reach the firing time. These experiments show that the works of higher-order partitioning are complementary to the works of the zeroth-order partitioning.

We tested the performance of our method by simulating a large-scale network. The network consists of 100 neurons in the Spike-Response model. Every neuron has a response function $\eta(t) = \exp(-\theta_1 t)$ and sine-wave external input $H(t) = wsin(\omega t)$, and 10 connections from other neurons, each of which has the activation function $\epsilon_{ij}(t) = w_{ij}(\exp(-\theta_1 t) - \exp(-\theta_{ij} t))$, where w_{ij} and θ_{ij} are randomly determined for each connection. We also introduced 200 random pulses to the network, so 6,276 fires were observed in the range of simulation.

Table 4.1 lists the performance results carried out on a Pentium 4 Xeon

Partitioning	2nd		1st		0th	
Quick filtering	Yes	No	Yes	No	Yes	No
Time (sec)	5.09	6.02	5.21	6.00	10.88	11.21
# of Partitions	53,790	90,348	55,326	91,995	142,057	179,624
0th partitions	41,222	75,904	42,292	77,051	133,811	171,448
1st partitions	11,699	13,484	13,034	14,944	0	0
2nd partitions	869	960	0	0	0	0
Δt partitions	0	0	0	0	8176	8176
# of Re-scheduled events	23,545	56,350	23,424	56,350	22,527	56,350
# of events filtered by GLC[a]	26,720	0	26,752	0	27,218	0
# of events filtered by 0th-LE[b]	9,838	0	9,917	0	10,349	0

[a]Gradient-limit checking

[b]Zeroth-order linear envelope checking

Table 4.1: Performance experiments

2-GHz processor. The second-order partitioning method with quick filtering is the fastest of all the tested configurations. The table shows that the reduction of the number of partitions by higher-order partitioning algorithms exceeds the additional cost of the complex partitioning algorithm. In addition, the quick-filtering techniques — gradient-limit checking and zeroth-order linear checking — are effective to filter out calculation of partition ends to be re-scheduled. Note that in all tests, memory consumption was kept under 1.6 megabytes.

4.7 Related Work

Only a few studies pursue discrete-event simulation of pulsed neural networks. The first simulator by Watts, SPIKE [53] targeted a simple neuron model and a small network. He showed the advantage of the discrete-event simulation framework by simulating complex behavior on a hand-made neural network.

Mattia and Giudice [37] developed techniques for large-scale discrete-event simulation of pulsed neural networks. They achieve efficiency by grouping simultaneous pulse arrivals into one event, using a layered queue structure. Their simulator efficiently handles synaptic plasticity and Poisson-distributed random inputs. They also discuss the handling of delayed firing in the case of a neuron model with a simple differential equation, but it is not applicable to a general neuron model.

Graßmann proposed a distributed simulation of pulsed neural networks on a discrete-event simulation framework [15, 16]. He reported speedup by a factor of 2.4 on three CPUs. He also mentioned that delayed firing can be predicted by using table lookup, but details are not given[1].

[1]Although a table-lookup method can accelerate calculation of delayed firing if the pulses are represented by a simple formula, it seems inapplicable to a neuron with input of various pulse models and to a neuron with external inputs. Moreover, to achieve high precision for the calculation of time and threshold, our techniques will be also required, such as firing possibility check by peak search and refinement of the calculation by the Newton-Raphson method.

4.8 Summary and Future Work

We developed the second-order incremental partitioning method, an efficient method for predicting the delayed firing time from the function of the neuron state variable. We also devised the quick-checking technique, which uses the time of a future pulse arrival to reduce the cost of future prediction. Using these techniques, we implemented a neural network simulator that is based on a discrete-event simulation framework but is still capable of simulating practically any Spike-Response neuron model.

One of our future works is the parallelization of the simulator. Many large-scale discrete-event simulations are now performed in a parallel computing environment [13, 27]. The discrete-event simulation of a pulsed neural network seems a suitable application for parallelization, since every pulse transmission can be treated as an event, and delays between neurons enable us to use a simpler synchronization mechanism. Moreover, the queuing model used in our simulation localizes the scheduling information, so it is expected that the simulation can be speeded up more by parallelization of our queuing model than by that of a single-queue model.

4.9 Appendix: Linear Envelope Calculation in Punnets

This section illustrates calculations of linear envelopes in the PUNNETS system.

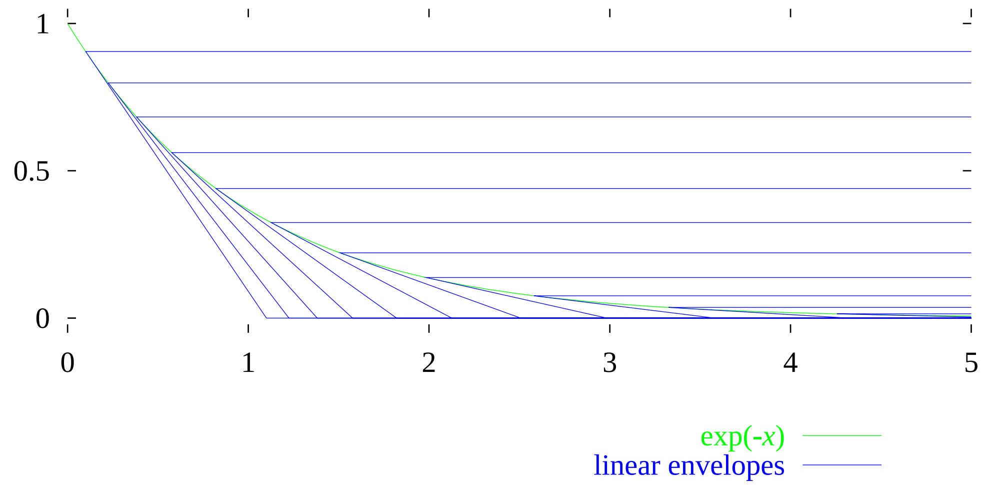


Figure 4.13: Linear envelopes of $f(x) = \exp(-x)$

4.9.1 Monotonic Convex Function

It is simple to calculate linear envelopes for monotonically increasing or monotonically decreasing and converging convex functions, such as $f(x) = \exp(-x)$.

For a monotonically decreasing function, the following definition of linear envelope for the point x_0 is used.

$$\mathcal{L}(f, x_0) = \begin{cases} y \leq f(x_0) \\ y \geq f(x_0) + \frac{df}{dx}\Big|_{x=x_0}(x - x_0) \\ y \geq f(\infty) \end{cases} \tag{4.1}$$

A monotonically increasing function can be transformed into a monotonically decreasing function, such as $g(x) = -f(x)$. Figure 4.13 illustrates linear envelopes for various points of $f(x) = exp(-x)$.

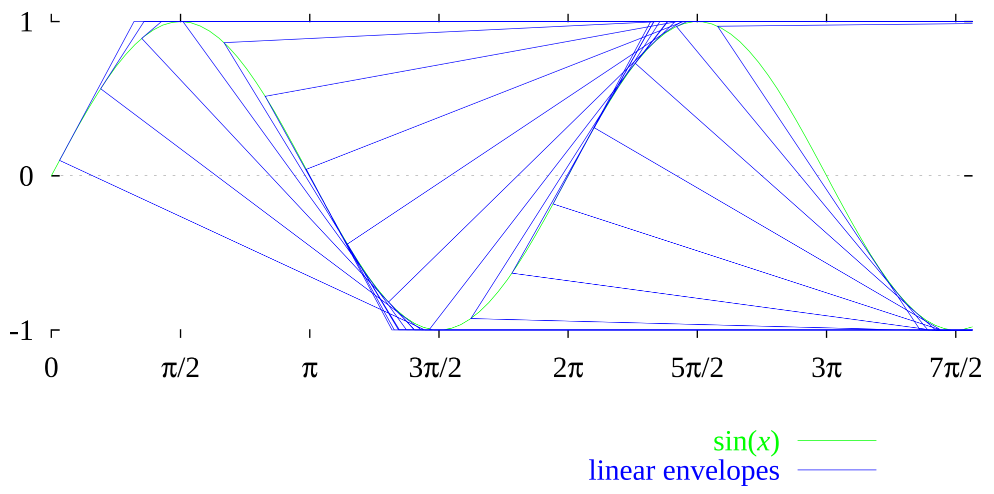


Figure 4.14: Linear Envelopes of $f(x) = \sin(x)$

4.9.2 Sine Function

In calculation of the linear envelopes of a sine function, it is useful to use information of tangent lines from the start point of the linear envelope. However, there is no easy formula for calculating the tangent lines. We used the following approximation for the gradient of one of the tangent lines:

$$\gamma(x_0) = \begin{cases} \cos(\theta) & (\theta < -\frac{\pi}{2}) \\ \sin(\alpha(\cos(\beta(\theta + \frac{\pi}{2})) - 1)) & (\theta \geq -\frac{\pi}{2}) \end{cases} \quad (\theta = x_0 + 2n\pi,\ -\pi \leq \theta \leq \pi), \tag{4.2}$$

where $\alpha = 1.311$ and $\beta = 0.375867$. The gradient of the other tangent line can be calculated from $-\gamma(t_0 + \pi)$.

As shown in Figure 4.15, the approximation always gives a greater gradient than the actual gradient of a tangent line. A linear envelope calculated from the approximation therefore always contains a linear envelope calculated from the

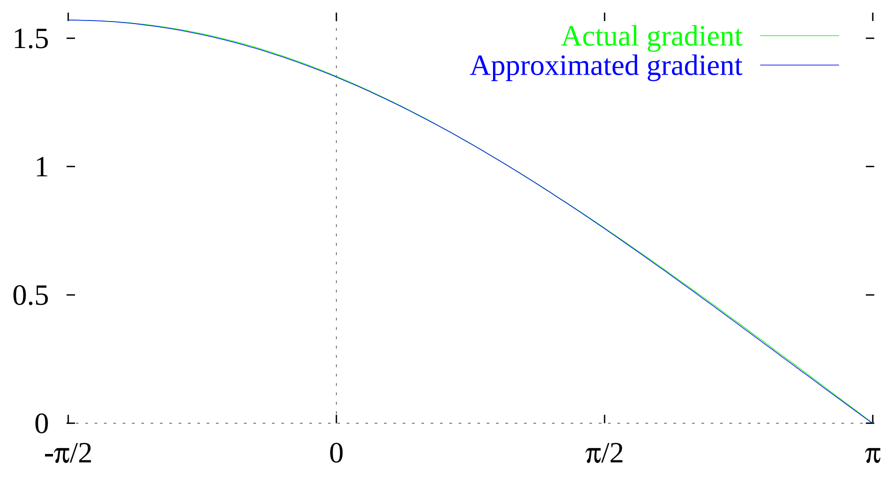


Figure 4.15: Comparison of actual gradient and approximated gradients $\gamma(x_0)$

actual tangent lines. Figure 4.14 illustrates linear envelopes for $f(x) = sin(x)$. In a formula, the linear envelope is given below.

$$\mathcal{L}(sin, x_0) = \begin{cases} y \leq 1 \\ y \leq -\gamma(x_0 + \pi)(x - x_0) + x_0 \\ y \geq \gamma(x_0)(x - x_0) + x_0 \\ y \geq -1 \end{cases} \tag{4.3}$$

4.9.3 Pulse Response Function

In the Spike-Response model, the following function is often used as a response to a pulse.

$$s(x) = w \cdot (\exp(-ax) - \exp(-bx)) \quad (0 < a < b) \tag{4.4}$$

We can apply linear transformation for the function in the following canonical form.

$$S(x) = (\exp(-zx) - \exp(-z\phi x)) \quad (\phi = \frac{b}{a} > 1, z = \frac{2\log\phi}{\phi - 1}) \tag{4.5}$$

Here, z is a normalization factor that fixes the inflection point of the function to $x = 1$. Beyond the point ($x > 1$), the function is monotonically decreasing, so the linear envelope described in Section 4.9.1is applicable. This function reaches the peak at $x = \frac{1}{2}$, and the value at the peak is $S(\frac{1}{2}) = \exp(-\frac{1}{2}zx) - \exp(-\frac{1}{2}z\phi x)$.

We also use the approximation to estimate the gradient of the tangent line:

$$\xi(x_0) = (-z\exp(-z) + z\phi\exp(-z\phi))(1 - (1 - x_0)^{\psi}), \tag{4.6}$$

where $\psi = 0.3(\log_{10}\phi)^2 + 2.45$. Although this approximation is not as good as the approximation of sine function, the function always gives a larger gradient than that of the actual tangent.

This approximation gives the linear envelope as in follows:

$$\mathcal{L}(S, x_0) = \begin{cases} y \le S(\frac{1}{2}) \\ y \le S(x_0) + \begin{cases} \left.\frac{dS}{dx}\right|_{x=x_0}(x - x_0) & (x_0 < \frac{1}{2}) \\ 0 & (x_0 \ge \frac{1}{2}) \end{cases} \\ y \ge S(x_0) + \begin{cases} \xi(x_0)(x - x_0) & (x_0 < 1) \\ \left.\frac{dS}{dx}\right|_{x=x_0}(x - x_0) & (x_0 \ge 1) \end{cases} \\ y \ge 0 \end{cases} \tag{4.7}$$

Figures 4.16 and 4.17 illustrate linear envelopes of Equation (4.7) for $\phi = 10$ and $\phi = 2$. Figures 4.18, 4.19, 4.20, and 4.21 show the actual tangent, estimated tangent, estimation error, and actual and estimated tangents at $\phi = 10$.

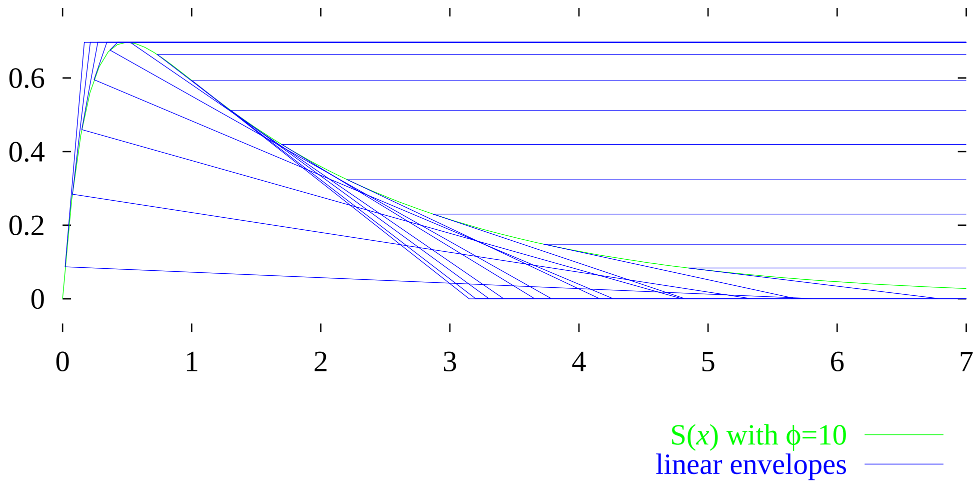


Figure 4.16: Linear envelopes of $S(x)$ at $\phi = 10$

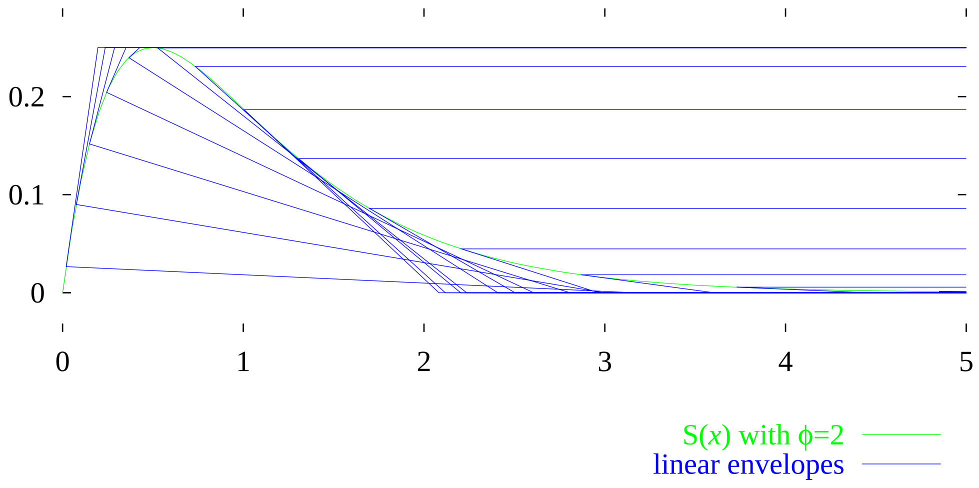


Figure 4.17: Linear envelopes of $S(x)$ at $\phi = 2$

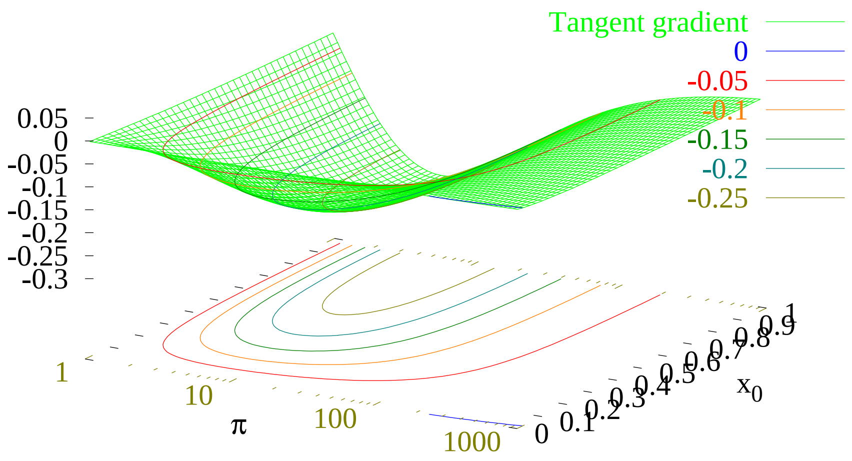


Figure 4.18: Actual tangent gradient

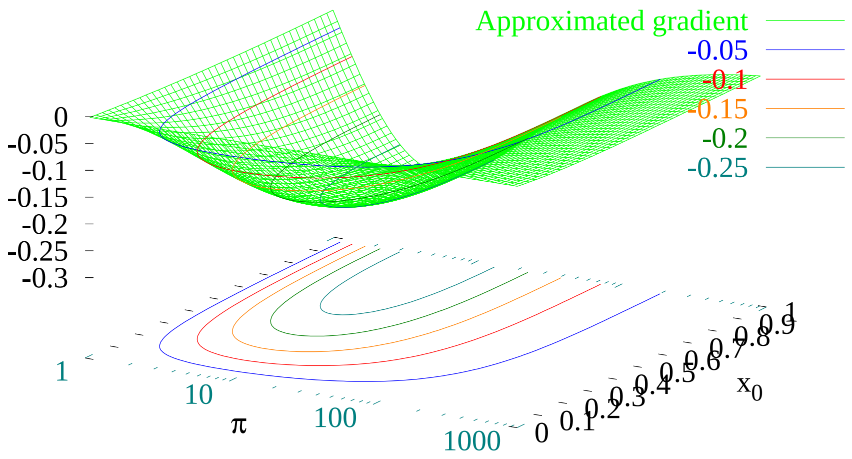


Figure 4.19: Approximated tangent gradient $\xi(x_0)$

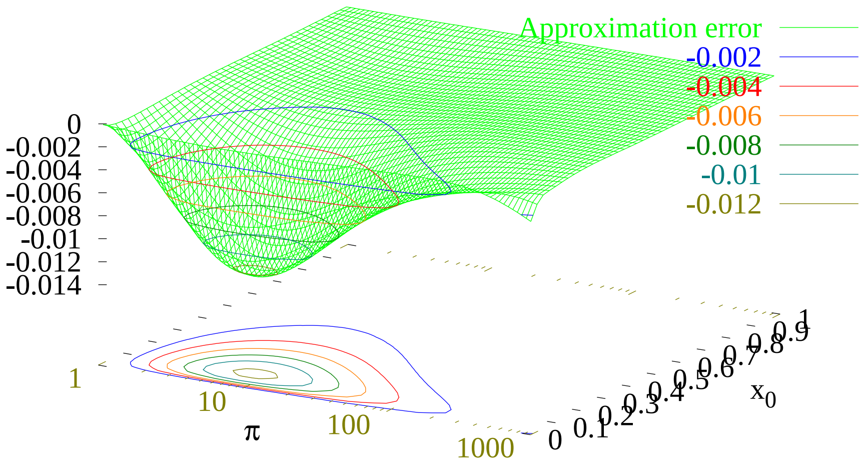


Figure 4.20: Approximation error of $\xi(x_0)$

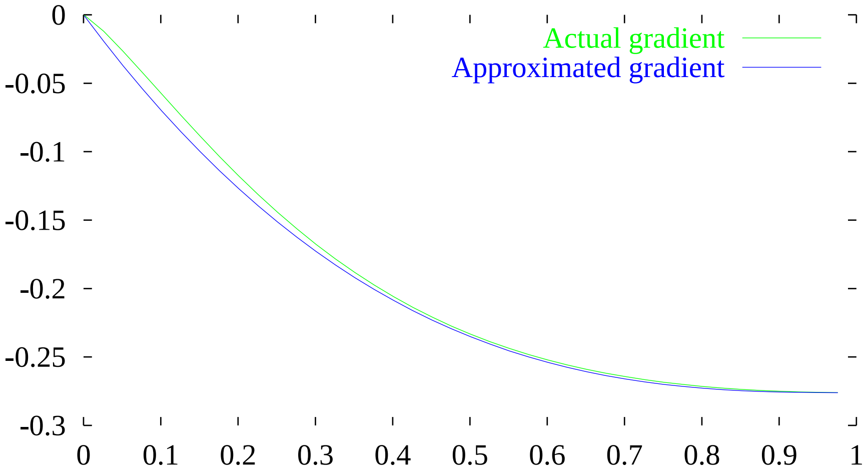


Figure 4.21: Actual and approximated tangents of $\xi(x_0)$ when $\phi = 10$

Chapter 5

Neural Network Model Towards Language Understanding

We propose a design of a dynamical neural network model that converts a simple input sentence into a set of binding representations in the temporal coding. This network model can be an important step towards a language-understanding neural network model, because binding representations are considered as a critical portion of semantic representations. The experiments on the network model show that a neural network model can be constructed within the requirements and preferences we explored in Chapter 3. Although the model is too small to discuss the linguistic problems, the following discussion reveals some important differences of the model from human language understanding, and points to further directions of research.

In Section 5.1 we describe the design of neural network model. After that, the result of experiments on the network is shown in Section 5.2, which shows the construction of binding representations from a sentence. Finally we discuss the design and future directions of this research in Section 5.3.

5.1 Design of the Network Model

In this section we describe the overall architecture of our neural network model for sentence-understanding.

Our goal is a continuous-time model of a neural network that produces binding representations of an input sentence, as described in Section 3.2. A sentence is given to the network as a sequence of words. The network changes its state as it receives each word. After the network receives the last word of a sentence, the result state is expected to keep the meaning of the sentence. Here, a meaning includes bindings in the sentence, e.g., a sentence 'John loves Mary' contains 'John=lover' and 'Mary=beloved,' where the two symbols connected by an equal sign ($X = Y$) denote a binding of the two, X and Y. As we discussed in Chapter 3, we suppose the oscillation phases is the representation of bindings. We run simulation of the network model, and test the achievement by probing the state of the network.

We designed a neural network model illustrated in Figure 5.1. The network consists of three components, a short-term memory holder, an autoassociative network, and a heteroassociative network.

In the language-understanding task, these components cooperate to construct the binding representation as demonstrated in Figure 5.2. When a sen-

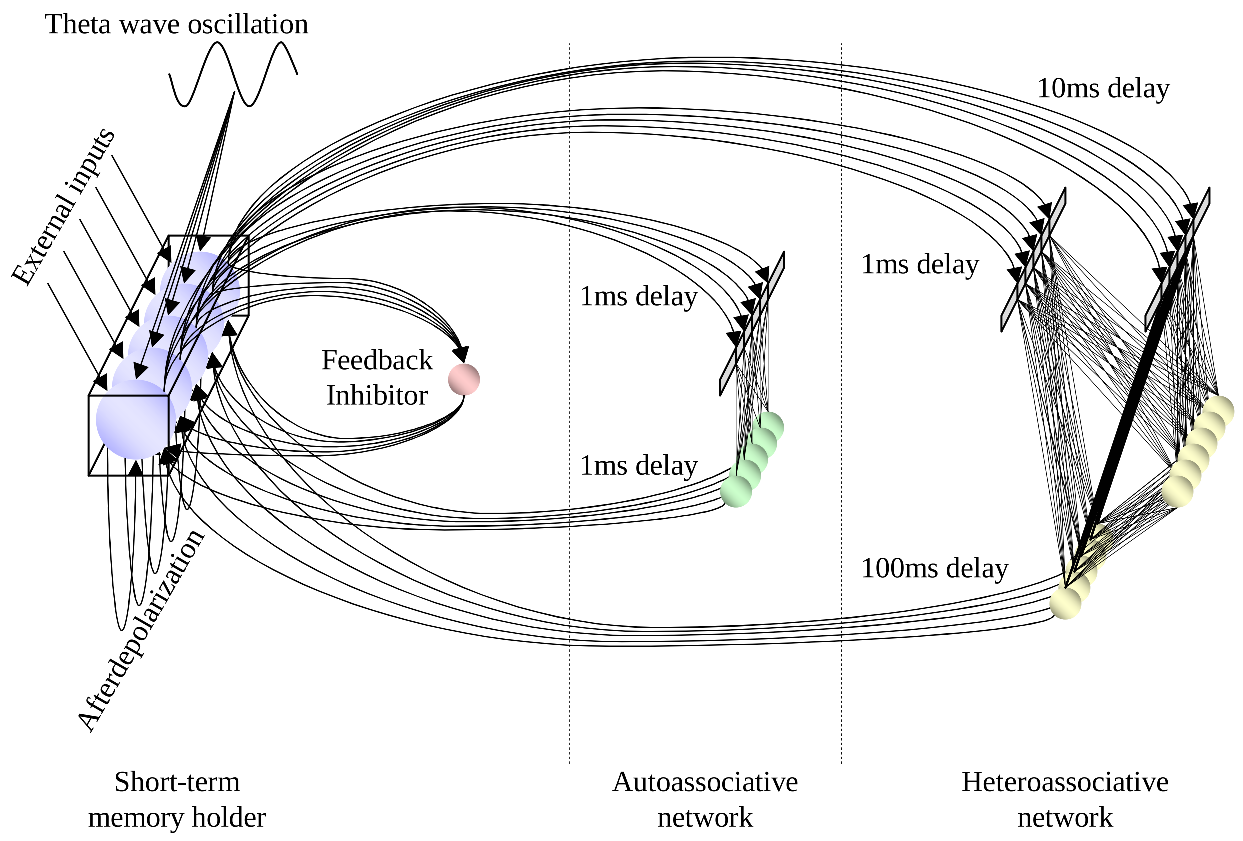


Figure 5.1: Proposed design of the neural network model

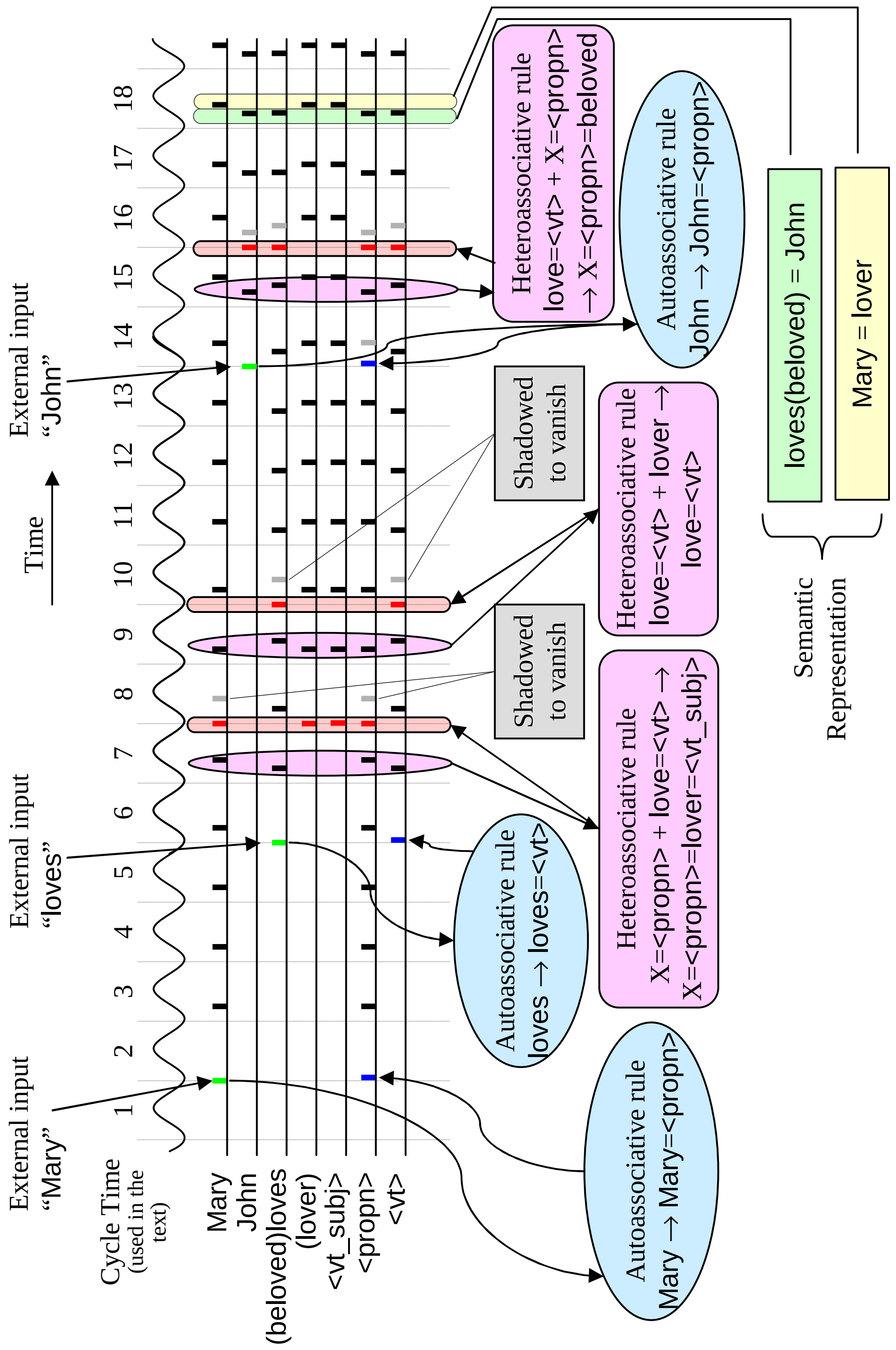

External input "Mary"
External input "loves"
External input "John"
Time
Cycle Time (used in the text)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Mary
John
(beloved)loves
(lover)
<vt_subj>
<propn>
<vt>
Autoassociative rule Mary → Mary=<propn>
Autoassociative rule loves → loves=<vt>
Heteroassociative rule X=<propn> + love=<vt> → X=<propn>=lover=<vt_subj>
Shadowed to vanish
Heteroassociative rule love=<vt> + lover → love=<vt>
Shadowed to vanish
Autoassociative rule John → John=<propn>
Heteroassociative rule love=<vt> + X=<propn> → X=<propn>=beloved
loves(beloved) = John
Mary = lover
Semantic Representation

Figure 5.2: Language understanding process in the proposed neural network model

tence is input word-by-word to the network, the short-term memory holder keeps the words in order and replays the sequence periodically. The autoassociative network recalls the corresponding lexical categories and attaches them to the words. Then, the heteroassociative network detects grammatical relations in the sequence to compose the memory items, and stores them as another item to the short-term memory holder. Binding representation is also generated by this composition. Since Lisman's memory model has a shadowing effect, which removes an old memory item at the time an overriding signal appears, an entry is recursively composed to other entries on the memory holder. Eventually the short-term memory holder holds a set of bindings, which represents the meaning of the original sentence.

5.1.1 Neuron Model

As the basic model of a neuron, we adopt a pulsed neural network model. More precisely, it is based on the Spike-Response model with external inputs, described in Section 2.2. This implies several important decisions, as described in the following.

First, we use a continuous-time model. We are interested in the construction of memory and meaning over a temporal domain in the real brain, whose behavior is dynamic in a continuous time domain. Discrete-time models, such as in SRN [8] or SHRUTI [46], limits temporal freedom, which contradicts our purpose to simulate the temporal behavior of the brain. To reconstruct the dynamics of the brain on the simulation, we choose to use a simulation based on a continuous-time model.

Second, we suppose the level of pulses of a given neuron is always constant. It is known that neurons transmit information by voltage pulses of membrane potential [14]; since all pulses of a given neuron look alike, the form of pulses does not carry any information. Continuous signal levels, such as used in sigmoidal gate neurons, are regarded as a modelization of rate coding, in which the information is conveyed by the density of pulses. However, in the viewpoint of temporal-coding, temporal difference greatly affects the meaning of same-rated oscillation; to simulate such a case, it is better to model each pulse individually, and to assign a constant activation level for the pulses.

On the memory model we represent the bindings by the synchrony of the phases of pulse firings. Although the discussion in Chapter 3 contains the representation of bindings other than the synchronous pulse firings (e.g. oscillation of specific patterns in a specific phase difference), we adopt the simplest form of the binding representation, synchrony of pulse firings, as a binding representation in our neural network model.

5.1.2 Short-term Memory Holder

The short-term memory holder is a set of memory neurons, which keep information as in a neural memory model proposed by Lisman and Idiart [29][1].

As we discussed in Section 3.4, language understanding with temporal coding requires phase arbitration, which arbitrates signal phases for a newly introduced memory item with existing memory items. Based on our claim of a global property of phase arbitration, we surveyed physiological studies to look

[1]Our memory model is slightly different from Lisman's model at some constants and response functions, but their behaviors are logically equivalent.

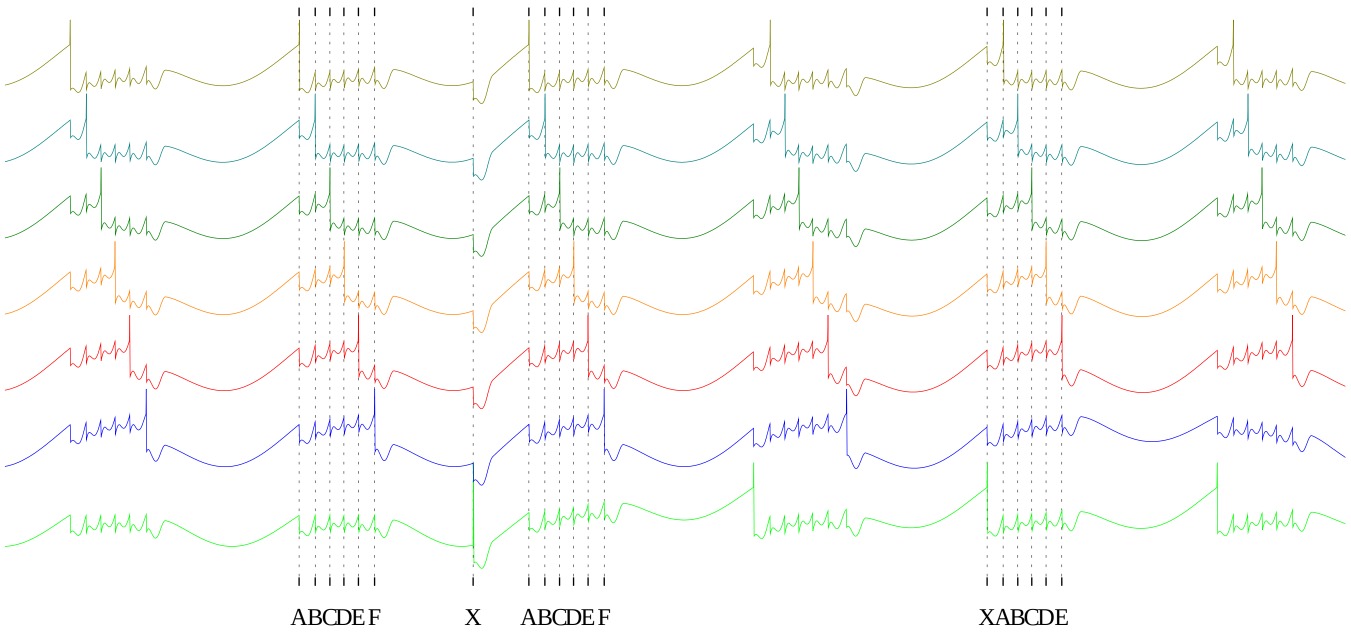


Figure 5.3: Lisman's memory model. At the left part of the figure, the sequence of six memory items, ABCDEF, is periodically replayed in the same order by the memory neurons. When a new item X is introduced, the item is prepended to the sequence. Feedback inhibition (dimples in the graph) causes the rest of the sequence to be shifted and, as a result, the last memory item F is spilled out from the memory.

for the possible implementation of phase arbitration; and we found that the phase precession phenomenon on the Theta wave oscillation [39] can account for the phase arbitration. Among several models describing phase precession, Lisman's model has an advantage over other models of its mathematical simplicity and comprehensiveness.

Figure 5.3 shows the simulation of Lisman's memory model. In this model, three types of effects, afterdepolarization (ADP), theta wave oscillation, and feedback inhibition, are combined to keep a series of activations on the memory. The ADP effect is a phenomenon that a neuron rises its membrane potential slowly after its activation. Although ADP is too brief to account for the duration of short-term memory, but it is long enough to store information between cycles of Theta–Alpha range oscillation (5–12Hz). Since Theta wave oscillation (8Hz) is also combined to the signal level of the neuron, ADP triggered in one cycle promotes firing in the next cycle, and as a consequence, ADP is refreshed each cycle and firing is maintained for many cycles.

The feedback inhibition is also introduced to partition a cycle into subcycles, on which multiple memories are stored in order. Since ADP works slowly, the most excitable cells are therefore not those that just fired, but those that fired earliest. Thus the slow increase in the ADP provides ramps of excitation that could serve as a basis for ordering multiple memory items.

For simplicity we adopted localist representation model; in our simulation model, we use 43 short-term-memory neurons, and assigned 30 words to 30 neurons, and subject-binding version of transitive verbs to 5 neurons, and 8 part-of-speeched to the remained neurons (See Figure 5.4).

5.1.3 Associative Networks

Two associative networks are used to provide lexical lookup and grammatical inference. One is *autoassociative* to recall a complete memorized pattern from an incomplete, noisy version of the same pattern. The other is *heteroassociative*, that is, the network produces an associated pattern from an input pattern,

Determiners	First-person pronouns	Third-person pronouns	Proper nouns	Common nouns	Verb-intransitives	Verb-transitives	Verb-transitives (subject binding)
<det>	<pn1>	<pn3>	<prop>	<noun>	<vi>	<vt>	<vt subj>
a	I	he	Mary	man	runs	likes	(liker)
the	me	him	John	lady	sleeps	has	(owner)
		she	Susan	boy	smiles	loves	(lover)
		her	Mike	girl		reads	(reader)
				book		eats	(eater)
				paper			
				mail			
				bread			
				lemon			
				banana			

Figure 5.4: Words and part-of-speeches assigned to a neuron in the simulation.

where the two patterns may differ.

This network design is inspired from the work of the sequence recall [47], which showed that two reciprocally connected recurrent networks provide more accurate sequence recall than a single heteroassociative network. Although repeated application of a heteroassociative network may seem to recall a sequence, it tends to amplify errors repeatedly. This problem can be fixed by another autoassociative network, which corrects the error in every recall step. Lisman suggests that two reciprocally connected recurrent networks, which are found in dentate and CA3 cells in hippocampal region, work in this way [28]. We thus designed our network model with the reciprocally connected heteroassociative and autoassociative networks, and assigned roles to these networks.

In our model, the autoassociative network is configured to provide a role of lexical lookups in language understanding. For example, when a memory neuron representing a word 'Mary' is activated at the beginning of the cycle 2 in Figure 5.2, the autoassociative network recalls a lexical category neuron, <noun>, as an overlaid activity to the word, Mary. This role is consistent with autoassociation if we regard a lexical lookup as recalling a complete lexicon (Mary=<noun>) from a partial information (Mary). Since the autoassociative network is connected to the memory cells with a small delay (2ms for a round trip), the completion on the memory occurs almost immediately. Note that lexical information such as <noun> is represented as a binding of the signal of entities, e.g. Mary.

On the other hand, the heteroassociative network is configured to have a role of grammatical inference. The heteroassociative network is connected to the short-term memory holder by two bundles of synapses. Different delays are assigned to the bundles (one is 10ms delay, and the other is 1ms) so that the difference (10ms − 1ms) corresponds to the duration between two memory items in the short-term memory holder. If the relation of two memory items matches a grammatical rule, the network outputs the result of the rule application. For example, at the cycle 7 in Figure 5.2, the heteroassociative network detects Mary=<noun> memory item followed by loves memory item. The heteroassociative is set up to output a binding of <noun>, lover, and anything bound to <noun> (denoted as X). As a result, the network outputs Mary=<noun>=lover, which is stored into the short-term memory holder.

We adapted the network design to the Lisman's memory model. First, we assigned a large delay (100ms) to the return connection from the heteroassociative network to the short-term memory holders. Second, the activation of the heteroassociative network is suppressed for the second and later pairs of memory items. The reasons and consequences of this adaptation are discussed in Section 5.3.2.

5.1.4 Shadowing Effect in the Short-term Memory

It is necessary to retract an item from the memory when a grammatical rule is applied to the item. For example, suppose a sequence of memory items, [love, the, girl], is stored in the memory[2] and solved by two grammar rule applications, "the + girl → NP=the=girl" and "love + NP → beloved=NP". When the first rule is applied and old entries are not removed, the result is appended as a new item for the memorized sequence, say, [love, the, girl, NP=the=girl]. From this sequence it is hard to see the two memory items love and NP=the=girl are combined by another grammar rule. It is necessary to remove the used item, namely the and girl, from the memory.

In the studies of linguistic algorithms [20], a *pop* operation (that is, 'remove the last item from the sequence') can solve this problem, since the memory forms a push-down stack. However, we want to avoid incorporating such an artificial complexity into the model. Ono's report [40] of a *shadowing effect* in the Lisman's memory model shed light to this problem. As illustrated in Figure 5.5, the shadowing effect is an inherent phenomenon in the Lisman's memory model that a memorization of a new item on a certain group of neurons causes to vanish an old memory item on the same group of neurons. This effect

[2]In our simulation model, memory items are prepended to cause the reversed sequence. For simplicity, we give this description based on an appending model.

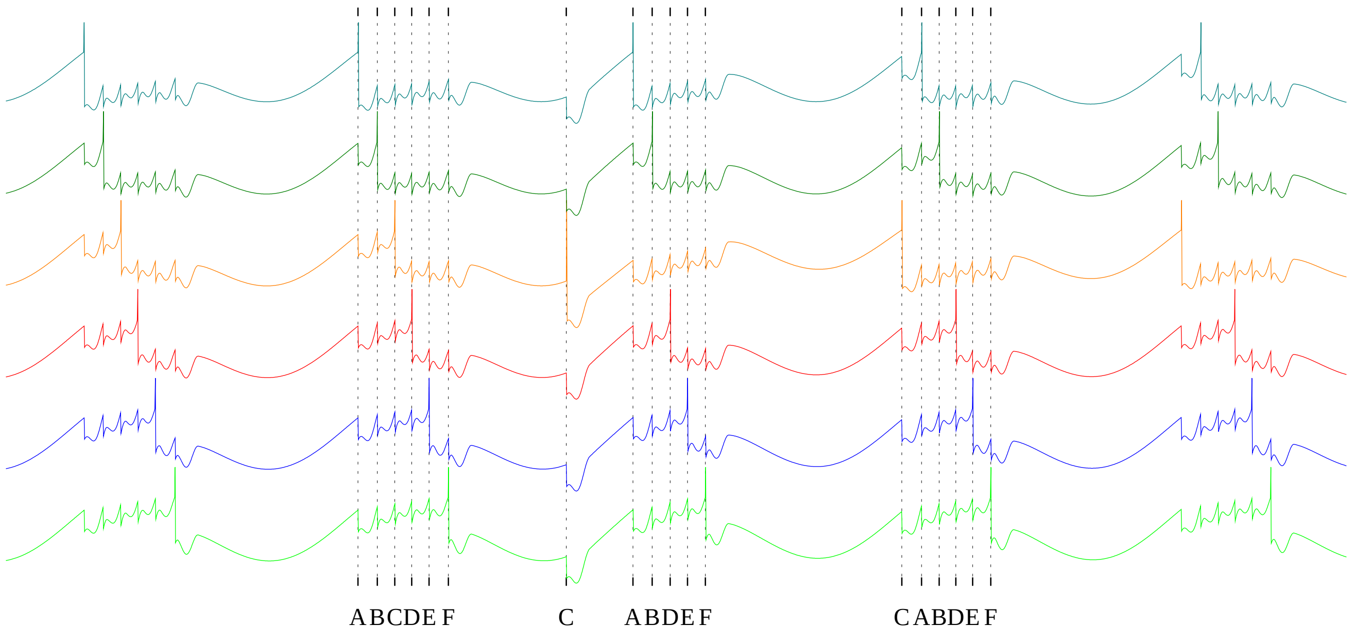


Figure 5.5: Shadowing effect of Lisman's memory model. The input of the memory item "C" shadows the existing same memory item in the sequence.

works quite differently from pop operation, but sufficiently for the recursive grammatical rule application.

We should be careful of the validity of the shadowing effect, since this is a side effect of the Lisman's memory model and we have no evidence of the effect in the brain physiology. However, usage of the shadowing effect suggests us possibility of the dynamics that organizes human language understanding.

Rule	Examples
Autoassociation	

$X = Y \rightarrow X = Y$ where $X = Y$ is a valid binding in the syntax	Mary=<propn> → Mary=<propn>
	the=<det>=boy=<noun> → the=<det>=boy=<noun>
	the=<det>=boy=<noun>=(lover)=< *vt subj* >
	→ the=<det>=boy=<noun>=(lover)=< *vt subj* >

Part-of-speech recall	
word → *word=pos*	Mary → Mary=<propn> loves → loves=<vt> (lover) → (lover)=<vt subj>

Table 5.1: Rules used in the training of the autoassociative network. A=B denotes a binding representation of A and B (synchronization of activities).

5.1.5 Training of the Associative Networks

We designed separate learning models to configure connection weights in the associative networks. Although unsupervised and integrated learning is important to prove the simplicity requirement, our first target is to empirically prove that the temporal coding and phase arbitration can formulate a language understanding mechanism. Thus we separated the learning process from the simulation: We perform separate learning to determine connection weights, and the fixed network is built into the neural network model. In the following we describe the configuration of the learning models.

The associative networks are, when isolated, just multi-layer perceptrons with a step threshold function [30]. Therefore only a good set of training data and an appropriate learning algorithm are required to determine their connection weights. In the experiments, the training data was given by enumerating

Rule	Examples
Determiner–noun binding	
$det + noun \rightarrow det{=}noun$	the=<det> + boy=<noun> $\rightarrow$ the=<det>=boy=<noun>
NP–verb(intransitive) subject binding	

$(det{=})noun + vi$
$\rightarrow det{=}noun{=}vi$
	the=<det>=boy=<noun> + sleeps=<vi>
	$\rightarrow$ the=<det>=boy=<noun>=sleeps=<vi>
	John=<propn> + runs=<vi>
	$\rightarrow$ John=<propn>=sleeps=<vi>

NP–verb(transitive) subject binding	

$(det{=})noun + vt$
$\rightarrow det{=}noun{=}vtsubj$
	a=<det>=lady=<noun> + has=<vt>
	$\rightarrow$ a=<det>=lady=<noun>=(owner)=<vt subj>
	she=<pn3> + reads=<vt>
	$\rightarrow$ she=<pn3>=(reader)=<vt subj>

verb(transitive) replacement	

$vt + (det{=})noun{=}vtsubj$
$\rightarrow vt$
	has=<vt> + a=<det>=lady=<noun>=(owner)=<vtsubj>
	$\rightarrow$ has=<vt>
	likes=<vt> + Susan=<propn>=(liker)=<vtsubj>
	$\rightarrow$ likes=<vt>

verb(transitive)–NP object binding	

$vt + (det{=})noun$
$\rightarrow det{=}noun{=}vt$
	loves=<vt> + a=<det>=man=<noun>
	$\rightarrow$ a=<det>=man=<noun>=loves=<vt>
	eats=<vt> + bread=<noun>
	$\rightarrow$ bread=<noun>=eats=<vt>

Keep silence when one is empty	
$\varepsilon + X \rightarrow \varepsilon$ $X + \varepsilon \rightarrow \varepsilon$	ε + loves=<vt> $\rightarrow \varepsilon$ loves=<vt> + $\varepsilon \rightarrow \varepsilon$
Keep silence for other occasions	

$verb + det \rightarrow \varepsilon$
$(det{=})noun{=}vtsubj + vt \rightarrow \varepsilon$
	loves=<vt> + a=<det> $\rightarrow \varepsilon$
	Susan=<propn>=(liker)=<vtsubj> + likes=<vt> $\rightarrow \varepsilon$

Table 5.2: Rules used in the training of the heteroassociative network. $A{=}B$ denotes a binding representation of A and B (synchronization of activities). A + B denotes an activity A is preceded by another activity B, that is, B has been stored in the memory just after A.

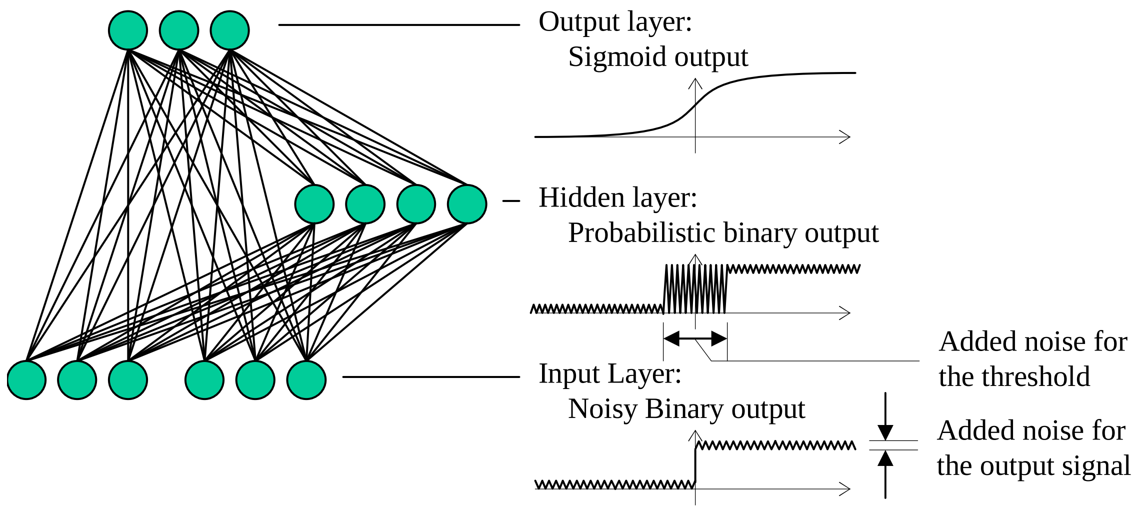


Figure 5.6: Network configuration in connection weight learning

all possible cases of lexical lookups and grammatical rule applications. Tables 5.1 and 5.2 show the training data of autoassociative and heteroassociative networks, respectively.

However, it is hard to find an appropriate learning algorithm. The neurons in the simulation communicate with pulses, and every pulse from a neuron looks alike. Therefore an output of the neurons are either presence or absence of a pulse, namely, a binary output as in the *perceptron* neuron model. No known learning algorithm with a hidden layer and binary neurons is theoretically supported.[3]

[3]The perceptron convergence procedure [38] can learn connection weights between the hidden layer and the output layer. It is known that a large number of hidden layer neurons with random connections between the input layer and the hidden layer will make the perceptron convergence procedure learnable. This can be our choice, but we didn't take this in order to minimize the simulation model.

For this problem we rely on a tentative solution: we introduce a neuron model, which outputs a binary value and propagates error as a sigmoid neuron, in the learning. Although the convergence of such a learning process is not theoretically guaranteed, we think that this is enough for our purpose, because the empirical tests showed that the algorithm can calculate the appropriate connection weights for our training set.

Another problem is that associative networks in the simulation are under influence of temporal fluctuation and trails of previous calculation, which works as a noise factor. Even if the network is correctly learned in the noiseless environment, it may not work in the actual simulation.

We solved this problem by introducing noise into the learning model. Figure 5.6 shows the learning configuration of the heteroassociative network. Small amount of random noise is added to the output of input neurons. To the hidden neuron, noise is added to both the threshold and the output. Moreover, output neurons are altered to sigmoid neurons, which allows a back-propagation algorithm to further modify the connection weight even if the inputs to the output neurons are close to the threshold.

5.2 Experiments

We implemented the simulation model as described in Section 5.1. The heteroassociative network is designed with 50 hidden neurons. The connection weights of the associative networks are calculated by the back-propagation learning algorithm, which requires 1 hour of calculation on Pentium III 500MHz with 128MB RAM. The simulation is run on PUNNETS, a discrete-event,

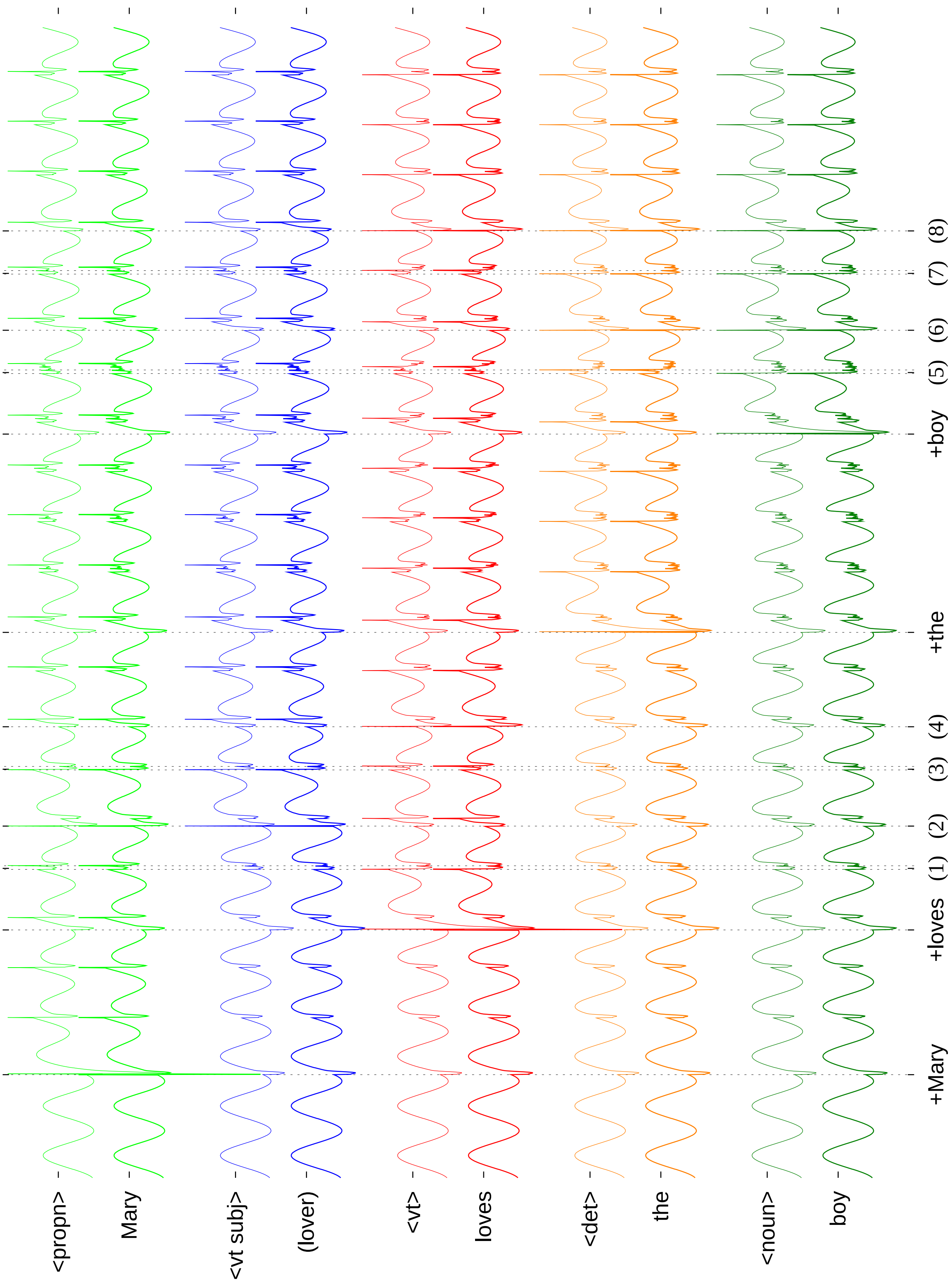


Figure 5.7: State variable transition of the language-understanding network.

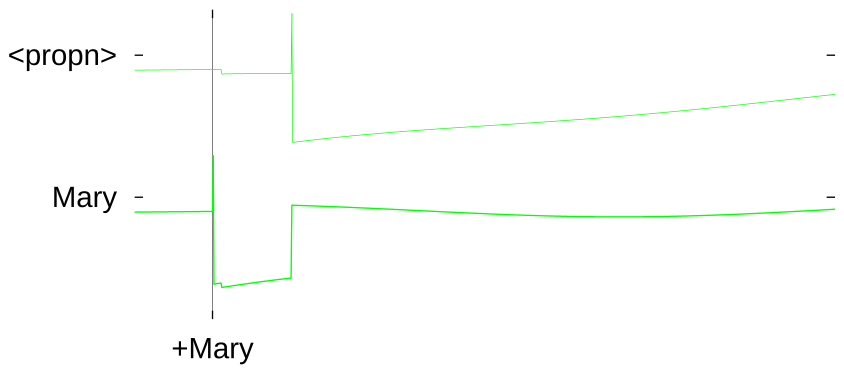


Figure 5.8: State variable transition showing the effect of the autoassociative network

continuous-time pulsed neural network simulator described in Chapter 4. In the experiments, the simulation successfully produced binding representations for 10 sentences, each consists of 3–5 words. The calculation time for the simulation is 0.3–0.8 seconds on the same machine.

Figure 5.7 shows the experiment with a sentence 'Mary loves the boy'. Each graph (Mary, <propn> etc.) shows a state variable of a short-term memory neuron. The X axis shows the progress of time in the simulation. Note that, as described in Section 5.1, all memory neurons are independent except connections by feedback inhibition of Lisman's memory model and two associative networks, although the graphs are arranged so that the graph of a word neurons is next to the graph of the corresponding part-of-speech neurons.

In the simulation, the system received only four external signals, Mary, loves, the, and boy, whose timings are denoted in the figure as +Mary, +loves, +the, and +boy, respectively. In each timing of the external signals, the autoassocia-

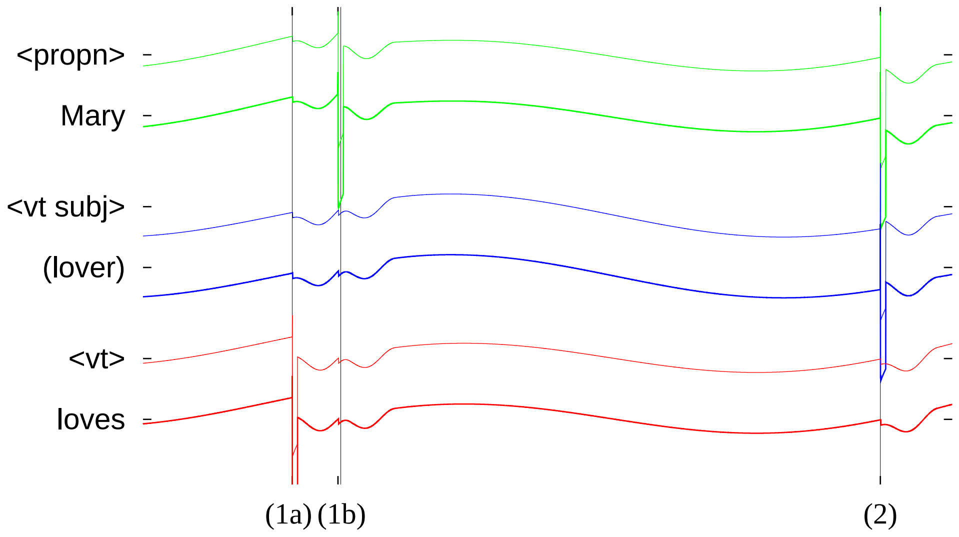


Figure 5.9: State variable transition showing the effect of the heteroassociative network

tive network is activated to cause the corresponding part-of-speech neuron (e.g. <propn> for Mary) to fire immediately (Magnified is shown in Figure 5.8). The activity of the part-of-speech neuron accompanies to the word signal throughout the simulation.

The short-term memory holder keeps memory items as the activities of neurons in a phase in the Theta oscillation. The Mary and <propn> signals keep firing synchronously at the raising of the Theta wave until the second signal (loves) is input. In the next cycle of the second signal input the phase of the first signal is shifted to the second place at the time (1).

At the same time, the temporal difference of two signals falls into the window of the heteroassociative network at the time (1). The heteroassociative network detects the two items should be combined, thus it returns the combined signals of Mary and lover at the time (2). Note that, at this point, the activation of the short-term memory becomes the semantic representation of a binding 'Mary — lover' (Magnified is shown in Figure 5.9).

At the time (3), two groups of signals, the loves=<vt> signals and Mary=<propn>=lover=<vt subj> signals, are processed by the heteroassociative network. As a result, loves signal is produced at the time (4) to change the timing of two signals.

After that, the the signal is input to the system. Since the heteroassociative network is not activated by the consecution of the signal and loves signal, it is just stored in the short-term memory. In the time between the the input and boy input in the figure, it can be seen that three groups of signals, the, loves, and Mary=lover, are periodically firing as a sequence.

When the boy signal is input, it is first prepended to the sequence. Thereafter the heteroassociative network detects the consecution of the and boy at the time (5), and the network puts a bound signal the=boy into the memory at the time (6). Then, in the next cycle (time (7)), the signal the=boy comes next to the signal loves, and the heteroassociative network is activated once again to produce a bound signal of loves and the=boy at time (8).

Eventually, the short-term memory neurons become to hold two groups of signals, Mary=lover and loves=the=boy. This is the binding representations constructed from the input sentence; two groups of signals represent the two bindings 'Mary is the lover' and 'the boy is the beloved' respectively.

5.3 Discussion

5.3.1 Continuous-time Property

We need a discussion how the continuous-time neuron model affects the simulation model. Since the continuous-time property increases the freedom of the model, it also increases the difficulty in the design of the model.

Insofar as this neural network model is concerned, we were able to construct without the continuous-time property. Although dynamics on Lisman's memory model require a continuous-time simulation model, we can emulate the dynamics on a discrete-time simulator. The associative networks are separately prepared by a learning on feed-forward networks; simulation of the network became difficult due to fluctuation caused by the continuous-time property.

Still it is significant that we have shown a way to develop a large-scale implementation on the continuous-time simulator with a complex neuron model. Continuous-time property is required for neural network models that emulates the dynamic behavior of the brain; future network models will depend to the continuous-time behavior more deeply. Since the techniques for such a simulator are absent, such a simulation tends to be restricted in an extremely small network size. We think this simulation model opened a path to the efficient simulation of more complex and large-scale dynamical neural network models.

One important application of the discrete-event simulation is the on-stage learning of the associative networks. It is said that synaptic time-dependent plasticity plays an important role on the short-term potentiation of hippocampal neurons [1]. Such a time-dependent learning rule will certainly need a simulation with high temporal precision, since the difference of a small fraction of milliseconds may change the direction of learning from potentiation to depression. Discrete-time simulation framework is unsuitable for such a simulation; it is the discrete-event simulation framework that can provide the required temporal precision efficiently.

5.3.2 Neuron Model

In this simulation model we adopted Lisman's mathematical model of the hippocampus neurons [29]. As discussed in Section 5.1.2, new memory items are prepended to the sequence in Lisman's model,

Several studies, including Lisman's later ones, suggest that a new memory item is *appended*, not prepended, to the sequence in a hippocampus region [55, 52, 28]. They pointed out that a compressed replay of a history, which is formed by appending, helps to predict future quickly. However, Lisman's memory model we used in the network design prepends a new memory item to the sequence; this behavior seems contradicting to this suggestion because the sequence formed by prepending becomes a reversal replay of a history.

While the appending, ordinarily ordered memory model seems suitable for our purpose, we used Lisman's prepending, reversely ordered memory model due to the absence of a mathematical model of pulsed neurons for appending sequences. This difference of memory order causes substantial difference of design of the neural network model. Figures 5.10 and 5.11 show the difference of behavior between ordinarily ordered and reversely ordered memory models.

First, we assigned a large delay (100ms) to the return connection to the

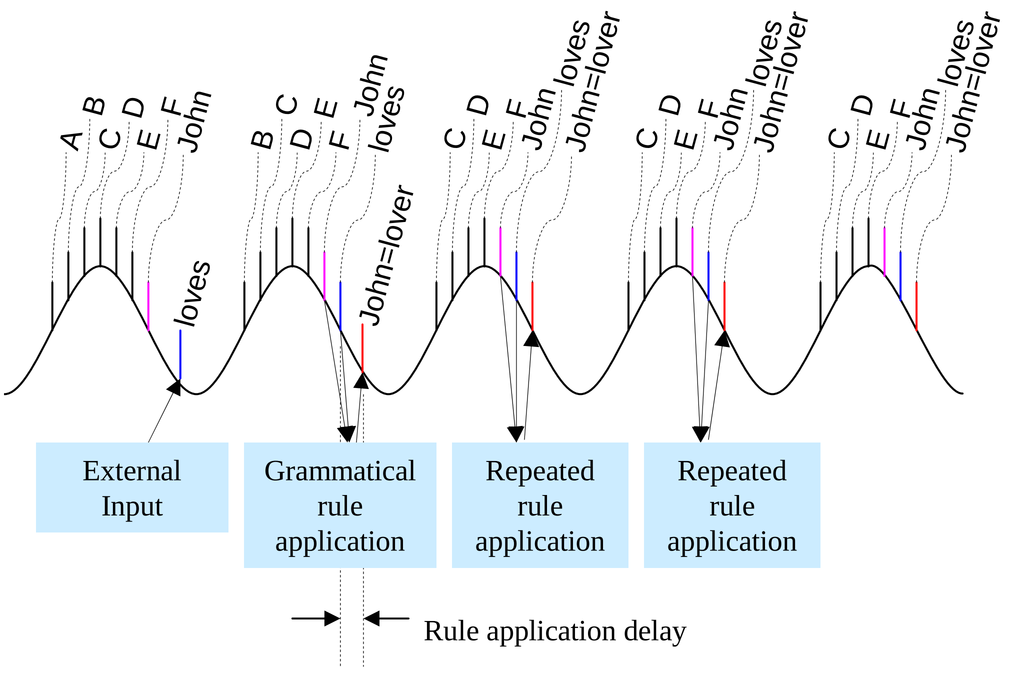


Figure 5.10: Grammatical rule application on an ordinarily ordered memory model. Delay of the rule application is as short as 1 subcycle, and multiple results are not stored because repeated rule applications overlap the previous results of application. For simplicity, the shadowing effect is not concerned in this figure.

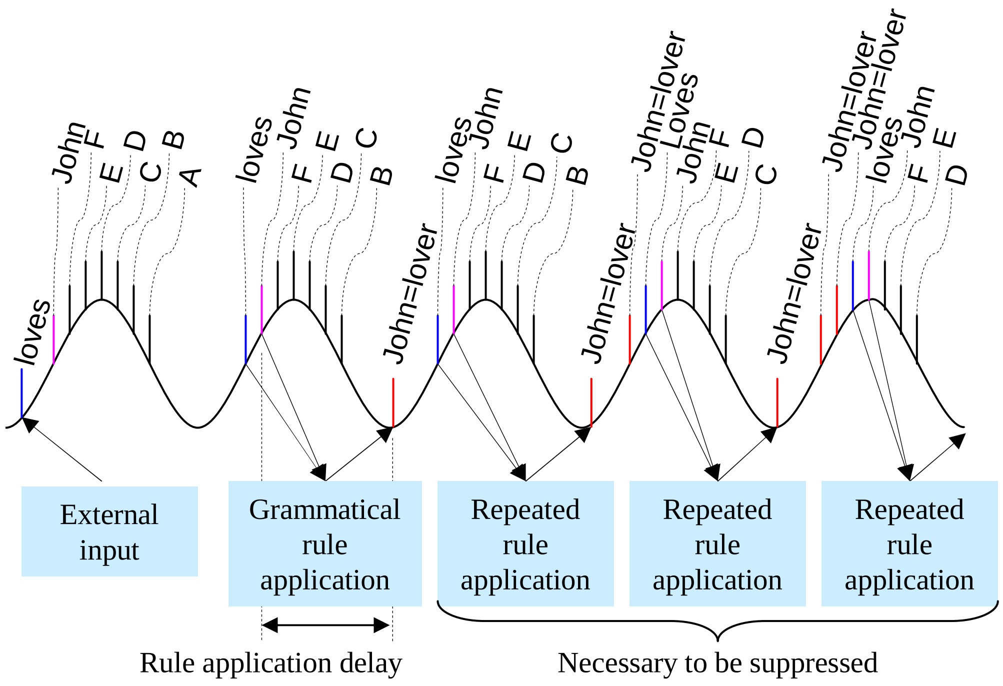

loves
John
F
E
D
C
B
A
External input
Grammatical rule application
Repeated rule application
Repeated rule application
Repeated rule application
John=lover
Rule application delay
Necessary to be suppressed

Figure 5.11: Grammatical rule application on a reversely ordered memory model. Delay of rule application have to be longer than the correctly ordered memory model. This figure shows a problem caused by repeated rule applications, which cause multiple results to be stored in the memory. In an actual system such a repeated rule application have to be suppressed, as in our simulation model. For simplicity, the shadowing effect is not concerned in this figure.

memory cells, although the delay can be as short as one subcycle in the correctly ordered memory model. This modification is necessary to store the result of rule applications at the next arriving slot of the memory.

Second, a larger change is that we have to suppress the repeated rule applications. Since the heteroassociative network applies rules to a succession of memory items, the rule is applied each time of the oscillation, as shown in Figure 5.11. Although the multiple entries are removed by the shadowing effect, such a repeated application confuses sequencing of the memory items. Thus we suppressed repeated rule applications by adding a special neuron in the heteroassociative network.

Note that the suppression is supposed to be unnecessary in the ordinarily ordered model. As shown in Figure 5.10, the result of the rule application can be placed to overlap on the next entry. This comes from the nature of the memory model; since the heteroassociative network is supposed to predict a future sequence of the compressed history replay, the prediction also works in the middle of the replay. Thus the prediction delay is likely to be adjusted to the delay between two memory items, and in this case, it is no reason to suppress the repeated applications.

5.3.3 Grammaticality and Disambiguation

The current simulation model is too small to discuss the handling of grammaticality and disambiguation. However, it is beneficial to discuss these points assuming an extended scale of the model.

One point is a handling of grammaticality. This system tries to construct meaning representations from a given sequence of words, irrespective of grammaticality. For example, traditional NLP systems cannot give a structure for a sentence ‘One boat two girls three boys,’ because the sentence is not grammatical. On the other hand, this system will be able to construct three meaning representations correctly, ‘one=boat’, ‘two=girl’, and ‘three=boy’, although these are not semantically related. Considering the fact that we easily understands the ungrammatical sentence, this supports our claim that the language should not captured only through structures.

Another important point is a disambiguation problem. Disambiguation is one of the main issues on natural language processing. Although a person can see the meaning of a syntactically ambiguous sentence, it is a difficult task for current computers. It is expected that the human-like language understanding model solves disambiguation in a natural way.

From a classical viewpoint, it is convincing that ambiguities cannot be solved in this network model. Every grammatical rule is applied as soon as available, and old lexical entries are removed by the shadowing effect. This seems as a deterministic process, which prevents the network from applying another ambiguous rule, which becomes available at the time another lexical item arrives into the memory.

However, this is not the case in our simulation model, where the grammar describes relations between semantic items. As illustrated in Figure 5.12, a disambiguation process is no more than a selection of depending semantic items on memory. Note that, in the lower part of the figure, the phrase ‘on the bench’ is attached to ‘girl’ after the meaning representation of ‘a letter for the girl’ is

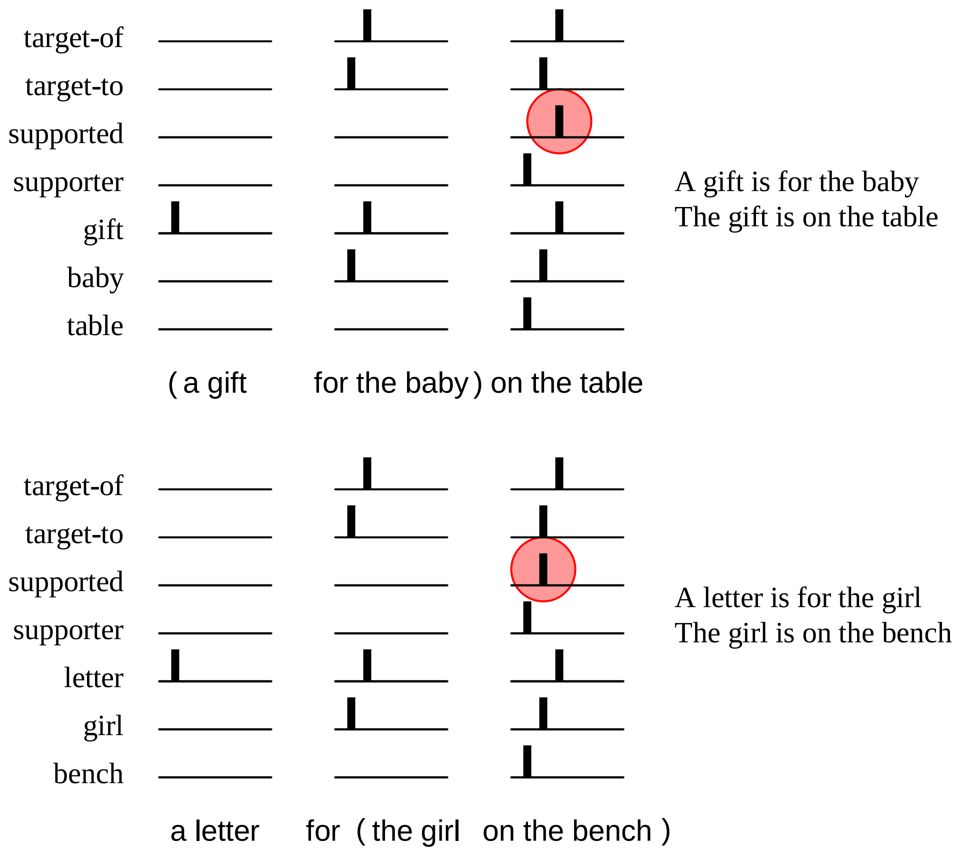


Figure 5.12: PP-attachment disambiguation in the simulation system. Since the system is free from structural constraints, disambiguation is just a selection of dependent memory items.

constructed. Such a process is contradictory to a structural view of syntax, in which 'on the bench' is attached to 'the girl' before 'the girl' and 'for' constitute a phrase. In the semantic representation, 'for the girl' is no more than 'the girl' bound with an attribute 'target-to', which allows to be attached by 'on the bench'.

It is still true that some sentence cannot be understood in a deterministic processing. To solve this problem a non-deterministic language understanding would be necessary. We believe that the connectionist representation of language understanding is a good place for future study of non-deterministic understanding process.

We do not claim that the simulation model can solve linguistic problems; our claim is that we now have a new viewpoint of the linguistic process inspired by the exploration of the brain mechanism. It is one of the major future works to pursue the linguistic implication of our simulation model.

5.3.4 Integrated Learning

One major future work is the integration of learning into the simulation model. In this simulation model, connection weights are manually given to the two associative networks. Although we utilized a variant of a back-propagation learning algorithm to determine connection weights, training are performed in the separated environment with manually prepared data set.

Integration of the learning of connection weights into the simulation model is necessary not only to discuss the acquisition of language ability but also to achieve the soundness of understanding. Although the current simulation produces signals called as a semantic representation, it is still distant from the understanding by human being, because of the lack of the grounding of the semantic representation. We think that the grounding can be achieved by learning the association of a representation to the non-verbal environmental input, as described in Section 3.2.

Since the learning process required for the association is not a supervised learning, we have to prepare an unsupervised learning rule. Synaptic time-dependent plasticity [1] is a variant of Hebb's rule with ability to learn temporal coincidence of presynaptic and postsynaptic activities without teacher signals. Since this plasticity is found in the mammal hippocampus cells, This plasticity is convincing for a learning rule in the network. However, this rule seems to require ordinary memory ordering, as described in Section 5.3.2.

5.4 Summary

We designed the neural network model, which converts a sequence of words to binding representations. The mechanism uses the short-term memory mechanism, which stores input words and binding representations in order, and grammar rules are applied on the memory items. As a result of the input of a sentence as successive words, the binding representations corresponding to the sentence is constructed on the short-term memory mechanism.

Although the simulation model is still primitive, we succeeded to show that we can construct the mechanism within the requirements and preferences of language understanding system we described in Chapter 3.Since some artificial constructions are used to build the model, some important differences of the model from human language understanding are also revealed, pointing to further directions of research.

Chapter 6

Conclusion

In this dissertation, we approached to the problem 'what is the understanding for a computer?' by constructing a neural network model of human language understanding, focusing on the binding representation of the human brain. A representation of feature bindings are included in the representation of intermediate and final products of the construction of meaning representation from a word sequence. In other words, language poses requirements to the capability of the binding representation. With an assumption that bindings are explicitly represented by some encoding in the brain, we explored the hardware implementation of the brain from the capability requirement, which the language understanding process depends on.

As a result of this study, we clarified several important requirements and preferences of the language understanding mechanism from the exploration, including utilization of the temporal complexity. We also developed several techniques, which enables us to develop a large-scale and efficient continuous-time simulator for neural network models based on the temporal coding. We finally built a simulation of a neural network model based on the requirements and techniques to validate the exploration and see the future direction of research.

The following are the requirements and preferences we clarified in the exploration: (1) If unencountered bindings are representable, representations and mechanisms have to be *additive*. (2) Since any additive static binary representation cannot keep multiple bindings simultaneously, the neural representation has to incorporate some complexity. (3) Among possible complexity sources, the temporal complexity has advantage over the intensive and spatial complexities. (4) If the temporal coding is used in the binding representation, some mechanism is required to arbitrate temporal fluctuation of input with the temporal coding. (5) The mechanism, we call *phase arbitration*, is likely to have global property, since local solution to the phase arbitration is difficult.

In the simulation techniques, we introduced a discrete-event simulation framework for efficient simulation of pulsed neural network. We focused on *delayed-firing prediction*, which had been hard for a complex neuron model in the existing simulators in the discrete-event simulation framework. We developed a *second-order incremental partitioning method* to predict delayed firings for any practical neuron model. Moreover, we introduced *gradient limit checking* to reduce the cost of the delayed-firing prediction. These techniques opened the path for a large-scale and efficient neural network simulator with high temporal precision.

We also developed a simulation of neural network model of human language understanding, based on the requirements and preferences we explored .

Although the simulation has been succeeded to show the validity of the requirements and preferences, we found several major differences from human language ability at the gaps of the requirements filled by the physiological ideas. However, this simulation model triggered a number of discussions related to linguistics, physiology, and connectionist models, which indicate the future direction of researches.

Bibliography

[1] Laurence F. Abbott and Sacra B. Nelson. Synaptic plasticity: taming the beast. *Nature Neuroscience*, 3:1178–1183, 2000. (cited in pages 42, 117, 125)

[2] Hassan Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction.* MIT Press, Cambridge, MA, 1991. (cited in page 41)

[3] Shun-ichi Amari and Hideo Sakata. *The brain and the neural network.* Asakura Shoten, Tokyo, 1994. In Japanese. (cited in page 14)

[4] Michael A. Arbib. *Metaphorical Brain 2, Neural Networks and Beyond.* John Wiley & Sons, Inc., 1989. (cited in page 11)

[5] James M. Bower and David Beeman. *The book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* TELOS/Springer-Verlag, New York, 1995. (cited in page 58)

[6] A. Browne and R. Sun. Connectionist variable binding. In S. Wermter and R. Sun, editors, *Hybrid Neural Systems.* Springer Verlag, Heidelberg, 2000. (cited in page 42)

[7] Noam Chomsky. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA, 1965. (cited in page 14)

[8] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–221, 1990. (cited in pages 11, 22, 31, 101)

[9] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structures. *Machine Learning*, 7:195–226, 1991. (cited in page 22)

[10] Jeffrey L. Elman. Language as a dynamical system. In R. F. Port and T. van Gelder, editors, *Mind as Motion: Explorations in the Dynamics of Cognition*, pages 195–225. MIT Press, Cambridge, MA, 1995. (cited in pages 23, 44)

[11] Jeffrey L. Elman, Elizabeth A. Bates, Mark H. Johnson, Annette Karmiloff-Smith, Domenico Parisi, and Kim Plunkett. *Rethinking Innateness: A connectionist perspective on development*. MIT Press, Cambridge, MA, 1996. (cited in page 22)

[12] Hiroshi Fujii, Hiroyuki Ito, Kazuyuki Aihara, Natsuhiro Ichinose, and Minoru Tsukada. Dynamical cell assembly hypothesis — theoretical possibility of spatio-temporal coding in the cortex. *Cognitive Science*, 9:1303–1350, 1996. (cited in pages 20, 38, 46, 47, 58)

[13] Richard M. Fujimoto. Parallel discrete event simulation: Will the field survive? *ORSA Journal on Computing*, 5(3):213–230, 1993. (cited in page 89)

[14] Wulfram Gerstner. Spiking neurons. In Wolfgang Maass and Christopher M. Bishop, editors, *Pulsed Neural Networks*, chapter 1, pages 3–53. MIT Press, Cambridge, MA, 1998. (cited in pages 19, 24, 36, 44, 59, 102)

[15] Cyprian Graßmann and Joachim K. Anlauf. Distributed, event driven simulation of spiking neural networks. In *Proceedings of the International ICSC/IFAC Symposium on Neural Computation (NC'98)*, pages 100 – 105. ICSC Academic Press, 1998. (cited in page 88)

[16] Cyprian Grassmann and Joachim K. Anlauf. Fast digital simulation of spiking neural networks and neuromorphic integration with SPIKELAB. *International Journal of Neural Systems*, 9(5):473–478, 1999. (cited in pages 58, 88)

[17] James Henderson. Connectionist syntactic parsing using temporal variable binding. *Psycholinguistic Research*, 23(5):353–379, 1994. (cited in pages 23, 31, 47)

[18] James Henderson and Peter Lane. A connectionist architecture for learning to parse. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics*, pages 531 – 537, Montreal, Canada, 1998. (cited in pages 11, 24, 47, 48)

[19] G. E. Hinton, James L. McClelland, and David E. Rumelhart. Distributed representations. In D. E. Rumelhart, J. L. McClelland, and PDP Research Group, editors, *Foundation*, number 1 in Parallel Distributed Processing:

Explorations in the Microstructure of Cognition, chapter 3, pages 77–109. MIT Press, Cambridge, MA, 1986. (cited in pages 36, 38, 52)

[20] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979. (cited in page 107)

[21] D. W. Jones. An empirical comparison of priority queue and event set implementations. *Communications of the ACM*, 29:300–311, 1986. (cited in page 82)

[22] Marcel A. Just and Patricia A. Carpenter. A capacity theory of comprehension: Individual differences in working memory. *Psychological Review*, 99:122–149, 1992. (cited in pages 12, 55)

[23] Hiroshi Kanayama, Kentaro Torisawa, Yutaka Mitsuishi, and Jun-ichi Tsujii. Statistical dependency analysis with an HPSG-based japanese grammar. In *Proceedings of 5th Natural Language Processing Pacific Rim Symposium*, pages 138–143, Beijing, China, November 1999. (cited in page 11)

[24] Tadao Kasami. An efficient recognition and syntax algorithm for context-free languages. Technical Report AFCRL-65-758, Air Force Cambridge Research Lab., Bedford, Mass., 1965. (cited in page 14)

[25] Uchimoto Kiyotaka, Satoshi Sekine, and Hitoshi Isahara. Japanese dependency structure analysis based on maximum entropy models. *Journal of the Information Processing Society of Japan*, 40(9):3397–3407, 1999. (In Japanese). (cited in page 11)

[26] Anthony Kroch and Aravind K. Joshi. Linguistic relevance of tree adjoining grammars. Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, April 1985. (cited in page 14)

[27] Yi-Bing Lin and Paul A. Fishwick. Asynchronous parallel discrete event simulation. *IEEE Transactions on Systems, Man and Cybernatics*, 26(4), 1996. (cited in page 89)

[28] John E. Lisman. Relating hippocampal circuitry to function: Recall of memory sequences by reciprocal dentate-CA3 interactions. *Neuron*, 22:233 – 242, 1999. (cited in pages 105, 118)

[29] John E. Lisman and Marco A. P. Idiart. Storage of 7 ± 2 short-term memories in oscillatory subcycles. *Science*, 267:1512 – 1515, 1995. (cited in pages 55, 56, 102, 118)

[30] Wolfgang Maass. Computing with spiking neurons. In Wolfgang Maass and Christopher M. Bishop, editors, *Pulsed Neural Networks*, chapter 2, pages 55–85. MIT Press, Cambridge, MA, 1998. (cited in pages 20, 109)

[31] Takaki Makino. A native-code compiler for a unification-based programming language with typed feature structures. Master's thesis, University of Tokyo, Tokyo, Japan, 1999. (cited in page 10)

[32] Takaki Makino. Discrete-event network simulation of arbitrary spike-response model. in submission to Neural Computing and Applications, 2002. (cited in page 82)

[33] Takaki Makino, Kazuyuki Aihara, and Jun-ichi Tsujii. Towards sentence understanding: Phase arbitration in temporal-coding memory mechanism. In *The Second Workshop on Natural Language Processing and Neural Networks (NLPNN'2001)*, pages 46 – 52, Tokyo, Japan, 2001. (cited in pages 58, 65)

[34] Takaki Makino, Kentaro Torisawa, and Jun-Ichi Tsujii. LiLFeS — practical programming language for typed feature structures. In *Proceedings of the 4th Natural Language Processing Pacific Rim Symposium*, Phuket, Thailand, 1997. (cited in page 10)

[35] Takaki Makino, Minoru Yoshida, Kentaro Torisawa, and Jun ichi Tsujii. LiLFeS — towards a practical HPSG parser. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics*, pages 807 – 11, Montreal, Canada, 1998. (cited in page 10)

[36] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* W. H. Freeman, San Francisco, 1982. (cited in pages 12, 13, 14)

[37] Maurizio Mattia and Paolo Del Giudice. Efficient event-driven simulation of large networks of spiking neurons and dynamical synapses. *Neural Computation*, 12:2305 – 2329, 2000. (cited in pages 58, 88)

[38] Marvin Minsky and Seymour Papert. *Perceptrons; an introduction to computational geometry.* MIT Press, Cambridge, MA, 1969. (cited in pages 20, 111)

[39] John O'Keefe and Michael Recce. Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330, 1993. (cited in pages 11, 55, 103)

[40] Yasuhiro Ono. Research on neural circuit network model holding short-term memory. Graduation thesis, Department of Mathematical Engineering and Information Physics, University of Tokyo, 2000. In Japanese. (cited in pages 56, 107)

[41] Randall C. O'Reilly and Yuko Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain.* MIT Press, Cambridge, MA, 2000. (cited in page 58)

[42] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press and CSLI Publications, Stanford, 1994. (cited in page 14)

[43] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, 1988. (cited in pages 67, 76)

[44] Paul Rodriguez, Janet Wiles, and Jeffrey L. Elman. A recurrent neural network that learns to count. *Connection Science*, 11(1):5–40, 1999. (cited in page 36)

[45] David E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representation by error propagation. In D. E. Rumelhart, J. L. McClelland, and PDP Research Group, editors, *Foundation*, number 1 in Parallel Distributed Processing: Explorations in the Microstructure of Cognition,

chapter 8, pages 318–362. MIT Press, Cambridge, MA, 1986. (cited in pages 17, 20)

[46] Lokendra Shastri and Venkat Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioural and Brain Sciences*, 16(3):417–494, 1993. (cited in pages 20, 47, 48, 101)

[47] H. Sompolinsky and I. Kanter. Temporal association in asymmetric neural networks. *Physical Review Letters*, 57:2861–2864, 1986. (cited in page 105)

[48] Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun ichi Tsujii. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG), 2000. (cited in page 14)

[49] Kentaro Torisawa and Jun'ichi Tsujii. Computing phrasal-signs in HPSG prior to parsing. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 949–955, 1996. (cited in page 14)

[50] Anne Treisman. Feature binding, attention and object perception. *Philosophical Transactions: Biological Sciences*, 353(1373), August 1998. (cited in page 38)

[51] Christoph von der Malsburg. The correlation theory of brain functions. Internal Report 81-2, Dept. of Neurobiology, Max-Planck Institute for Biophysical Chemistry, 3400 Gottingen, FRG, 1981. (cited in page 47)

[52] Hiroaki Wagatsuma and Yoko Yamaguchi. A neural network model self-organizing a cognitive map using theta phase precession. In *Proceedings of*

1999 IEEE International Conference on Systems, Man, and Cybernetics (SMC'99), Tokyo, Japan, October 1999. (cited in page 118)

[53] Lloyd Watts. Event-driven simulation of networks of spiking neurons. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in neural information processing systems*, volume 6, pages 927 − 934. Morgan Kaufmann Publishers, 1994. (cited in pages 58, 88)

[54] P. Werbos. Backpropagation through time: What it does and how to do it, 1990. (cited in page 24)

[55] Yoko Yamaguchi and Bruce L. McNaughton. Nonlinear dynamics generating theta phase precession in hippocampal closed circuit and generation of episodic memory. In *Proceedings of The Fifth International Conference on Neural Information Processing*, Kitakyushu, Japan, October 1998. (cited in page 118)

[56] Masaaki Yamanashi. *Principles of Cognitive Linguistics.* Kuroshio Shuppan, 2000. In Japanese. (cited in page 11)

[57] Minoru Yoshida, Takashi Ninomiya, Kentaro Torisawa, Takaki Makino, and Jun-ichi Tsujii. Efficient FB-LTAG parser and its parallelization. In *Proceedings of the 4th Conference of the Pacific Association for Computational Linguistics*, pages 90–103, Waterloo, Canada, August 1999. (cited in page 14)

Index