ADAPTIVE MORPHOLOGICAL ANALYSIS WITH A SMALL TAGGED CORPUS
最小限のタグ付きコーパスを用いた適応性の高い形態素解析

by

Jun'ichi KAZAMA
風間 淳一

A Master Thesis
修士論文

Submitted to
the Graduate School of
the University of Tokyo
on January 30, 2001
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Information Science

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

ABSTRACT

This thesis describes a method for constructing a probabilistic Japanese morphological analyzer with a small amount of tagged corpus. Probabilistic models have been applied to natural language processing successfully due to the existence of large manually tagged corpus to estimate their parameters. However, for the real world texts such as ones from the Internet, it is hard to prepare sufficiently large tagged corpus for all text and task type pairs. We solve this dilemma by using a small amount of tagged corpus and a large raw corpus. The idea is, if the hidden states of the probabilistic model such as hidden Markov models (HMMs) trained by unsupervised learning from a large raw corpus compactly contain sufficient information to determine tags, we can easily learn the mapping between states and tags by viewing a few occurrences of each state in a small tagged corpus.

In our method, character-based HMMs are employed to prevent the unknown word problem, which is serious in Japanese because there is no word segmentation markers such as spaces. Given raw texts from a target domain, we train a character-based HMM by the Baum-Welch algorithm, or a Gibbs sampling version of the Baum-Welch algorithm when the number of states is large. Then, our algorithm learns a tag emission model from a small tagged corpus by using the maximum entropy (ME) method. The learned tag emission model emits a tag appropriate for each character by viewing the Viterbi sequence of the HMM's hidden states. Experimental results are encouraging, showing that our method performs better than existing methods when only 10–500 tagged sentences are available.

論文要旨

本論文では, 最小限のタグ付きコーパスから, 日本語の確率的形態素解析器を構成する方法について述べる. これまで, 確率モデルは自然言語処理に応用され, 大きな成功を収めてきたが, その成功は, モデルのパラメータを学習するために人手でタグ付けされた, 大量のコーパスの存在に依存している. インターネット上の文書などの実世界の文書を処理することを考えると, 全ての文書タイプとタスクの組について, 十分な量のタグ付きコーパスを用意することは困難である. 我々は, この問題を少量のタグ付きコーパスと大量のタグ無しコーパスを使うことによって解決する. もし, 隠れマルコフモデル (HMM) など, 大量のタグ無しコーパスから学習した確率モデルの隠れ状態がタスクのタグを決定するのに十分な情報を持つようであれば, 少量のタグ付きコーパスにおいて, 各状態の出現を数回見るだけで, 隠れ状態とタグの対応を容易に学習できる可能性が高い.

我々の方法では, スペースなどの単語区切りを示すものが存在しない日本語において深刻な問題である未知語問題に対応するため, 文字ベースの HMM を用いる. これを, 対象領域のタグ付けされていない生の文書から, Baum-Welch アルゴリズム, あるいは, 状態数が大きいときには, Gibbs サンプリングを使った Baum-Welch アルゴリズムを用いて学習させる. その後, タグ出力モデルを, 最大エントロピー法などを用いて, 少量のタグ付きコーパスから学習する. 学習されたタグ出力モデルは, HMM の隠れ状態のビタビ列を見て, 各文字に最適なタグを出力する. 実験から, 10〜500 文のタグ付きコーパスしか利用できないときには, 我々の方法が既存の方法よりも高い精度を示すことが確認された.

Acknowledgements

First of all, I would like to thank my supervisor, Prof. Jun'ichi Tsujii, for his invaluable advice and encouragement. This thesis could not have been completed without his introduction to this exciting NLP research. I also owe much thanks to Dr. Kentaro Torisawa for his invaluable suggestions during this research. He also gave me continuous instruction and encouragement during the writing of this thesis. I thank Mr. Takaki Makino and Mr. Yusuke Miyao for reading and correcting my drafts. I also thank Dr. Edson T. Miyamoto for checking English of my thesis. And I thank Mr. Takashi Ninomiya for his continuous encouragement and tender care. Finally, I thank all members of Tsujii laboratory for their helpful suggestions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we propose a novel stochastic method for Japanese morphological analysis. Our method consists of two parts, a tag-independent character-based hidden Markov model (HMM) and a tag emission model which uses the maximum entropy method.

Morphological analysis identifies *morphemes* such as words and affixes in a sentence, and assigns them a tag such as *Noun*, *Verb*, and *Adjective*, which is called a part-of-speech (POS) tag. We refer to these processes as word segmentation and POS tagging. Probabilistic models have been applied to morphological analysis successfully since they can accurately extract statistical tendencies between the input surface data and the hidden POS tags from a large corpus where POS tags are *annotated* or *tagged* by human annotators. In order for these probabilistic models to be successfully applied, a corpus must be annotated by human annotators with the tag set which other NLP systems require as input. Moreover, the annonated corpus must be of the same type of which we are targeting.

We are interested in applying probabilistic methods to real-world texts from the Internet. However, there are few tagged corpora for this type of texts, and the required tag set varies depending on the situation. Therefore, the demands for an *adaptive* method that requires only a small amount of tagged corpus are increasing. We define two types of adaptabilities according to the above two requirements for tagged corpus – domain

adaptability and tag adaptability. Domain adaptability refers to the fitting of the model's parameters to the target text type. On the other hand, tag adaptability indicates the method's ability to handle any tag set when required by another natural language processing system which comes after morphological analysis. The final goal of this study is to achieve these two adaptabilities at the same time by using only a small amount of corpus.

In this thesis, domain adaptability is addressed by unsupervised learning of a character-based tag-independent HMM from a large raw corpus of the domain using the well-known Baum-Welch algorithm.

Arguments justifying the values of untagged raw corpus for achieving domain adaptability follow those in [9, 26]. Namely, although an untagged corpus alone is insufficient to yield better-than-random tagging because the relation between the hidden states of the HMM and tags is not known, untagged corpora contain information about the distribution over features, in our case, character cooccurrence. If we use this information together with a sample of tagged data, we will increase the tagging accuracy.

For tag adaptability, we propose the following approach.

- Divide parameter estimation into the estimation of a character-based tag-independent HMM and the estimation of a relation between an HMM's hidden states and the part-of-speech tags.
- Employ an HMM which has a sufficiently large state space to contain information required for morphological analysis with any tag set.

After the HMM is trained so that it can capture the statistical characteristics of the domain, the *tag emission model*, which relates the hidden states of the trained HMM and the part-of-speech tags, is estimated using a small amount of tagged corpus, utilizing the trained HMM. The estimation of this relation is based on the cooccurrence of the tags and the states in the Viterbi path which the trained HMM generates for a

sentence in a small tagged corpus. In this thesis, we propose a simple estimation based on frequency counts and a more complex estimation based on the maximum entropy framework. When we have to output another tag set, the same HMM can be used and the only thing we have to do is to re-train a tag emission model using a small tagged corpus in the given tag set.

Brants [8] proposed the use of two different tag sets in part-of-speech tagging, one is the internal tag set for the underlying HMM and the other is the external tag set for its output. The internal tag set is more fine-grained than the external tag set, and all information contained in the internal tag set is used for tagging. Using a large HMM as a tag-independent HMM is the extension of this idea to unsupervised learning of a tag-independent HMM. To enable training of such HMMs with large state space, we apply Gibbs sampling to the standard Baum-Welch algorithm.

The reason for employing character-based HMM, which emits each character in a sentence as output symbol, is to prevent the *unknown word problem*. Most existing morphological analysis methods use manually tailored word dictionaries. In word-based methods, all the possible word candidates in a sentence are retrieved from a word dictionary, then rules or probabilistic models select the most probable word sequence. However, it is impossible to build a dictionary which contains all possible words. If the identification of a word fails because of unknown words, the accuracy of POS tagging greatly declines accordingly. On the other hand, character-based methods are said to be robust against the unknown word problem because they do not require word dictionaries, as they define probabilistic models on characters. In the domain we are targeting such as the earlier, the rate of unknown words is high in addition to the lack of appropriate tagged corpora. Thus character-based methods are more suitable for our purpose. In addition, character-based models are useful for both types of adaptabilities, since domain adaptability often requires the solution of the unknown word problem and the

tag set adaptability may require the adaptation to a tag set in which differences at the word segmentation level cannot be solved by the word-based methods. We also propose to augment the HMM's output symbols with character type information to improve the accuracy of Japanese morphological analysis.

In the experiments, we show that our method outperforms other character *n*-gram based methods in word segmentation when the amount of available tagged corpus is less than $1,000$ sentences.

Structure of the Thesis

In Chapter 2, we describe the background topics of this thesis. We first describe the Expectation-Maximization (EM) algorithm briefly. Next, we introduce hidden Markov models. Finally, we describe the maximum entropy framework. In Chapter 3, we describe the division of model parameter estimation and the actual morphological analysis using it. In Chapter 4, we describe the training of a large HMM using Gibbs sampling. In Chapter 5, we report a series of experiments and analyze the results to confirm the effectiveness of our approach. In Chapter 6, we conclude this thesis and discuss future work.

Chapter 2

Background

In this chapter, we describe the background topics of this thesis. We first describe the Expectation-Maximization (EM) algorithm briefly. This is the basis of the Baum-Welch algorithm which we use in the unsupervised learning of a character-based HMM. Next, we introduce hidden Markov models and their basic algorithms used in our approach for morphological analysis. Then, we describe the maximum entropy framework which is used in the estimation of tag emission model. Finally we describe existing unsupervised approaches to morphological analysis.

2.1 The EM Algorithm

The Expectation-Maximization (EM) algorithm is a method for computing the maximum likelihood (ML) estimates from incomplete data iteratively [11].

Let $\mathbf{Y}$ be a random vector corresponding to observed data $\mathbf{y}$, which has a probability density function $p(\mathbf{y}; \mathbf{\Psi})$ where $\mathbf{\Psi} = (\psi_1, \cdots, \psi_d)$ is a vector of model's parameters. And let $\mathbf{X}$ be a random vector corresponding to complete data $\mathbf{x}$, which include observed data and hidden unobservable data with a probability density function $p_c(\mathbf{x}; \mathbf{\Psi})$. For example, in hidden Markov models, $\mathbf{y}$ is the observed sequence, and $\mathbf{x}$ is an observed sequence and a state sequence.

The aim of the maximum likelihood estimation is to find the parameters $\hat{\mathbf{\Psi}}$ that maximize the likelihood,

$$L(\mathbf{\Psi}) = p(\mathbf{y}; \mathbf{\Psi}),$$

or equivalently the log likelihood,

$$\log L(\mathbf{\Psi}) = \log p(\mathbf{y}; \mathbf{\Psi}).$$

We define $Q(\mathbf{\Psi}, \bar{\mathbf{\Psi}})$ as

$$Q(\mathbf{\Psi}, \bar{\mathbf{\Psi}}) = E_{\mathbf{\Psi}} \left[\log p_c(\mathbf{X}; \bar{\mathbf{\Psi}}) | \mathbf{y} \right].$$

Here, $E_{\mathbf{\Psi}}[\bullet]$ denotes the expectation with respect to the distribution over the range of $\mathbf{X}$ given by

$$k(\mathbf{x}|\mathbf{y}; \mathbf{\Psi}) = \frac{p_c(\mathbf{x}; \mathbf{\Psi})}{p(\mathbf{y}; \mathbf{\Psi})}.$$

That is,

$$Q(\mathbf{\Psi}, \bar{\mathbf{\Psi}}) = \int_{\mathcal{X}(\mathbf{y})} k(\mathbf{x}|\mathbf{y}; \mathbf{\Psi}) \log p_c(\mathbf{x}; \bar{\mathbf{\Psi}}) d\mathbf{x},$$

where $\mathcal{X}(\mathbf{y})$ denotes all $\mathbf{x}$ that yield $\mathbf{y}$.

Let $\mathbf{\Psi}^{(0)}$ be the some initial value for $\mathbf{\Psi}$. Then, the EM algorithm repeatedly alternates the following two steps.

On the kth iteration,

E-Step: Calculate $Q(\mathbf{\Psi}^{(k)}, \mathbf{\Psi})$

M-Step: Choose $\mathbf{\Psi}^{(k+1)}$ so that

$$\mathbf{\Psi}^{(k+1)} = \underset{\mathbf{\Psi}}{\operatorname{argmax}}\, Q(\mathbf{\Psi}^{(k)}, \mathbf{\Psi}),$$

that is,

$$Q(\mathbf{\Psi}^{(k)}, \mathbf{\Psi}^{(k+1)}) \geq Q(\mathbf{\Psi}^{(k)}, \mathbf{\Psi}) \qquad \text{for all } \mathbf{\Psi}.$$

Dempster et al. [11] showed that the log likelihood $\log L(\mathbf{\Psi})$ does not decrease after an EM iteration, i.e.,

$$\log L(\mathbf{\Psi}^{(k+1)}) \geq \log L(\mathbf{\Psi}^{(k)}).$$

Therefore, for most models, the algorithm will monotonically converges to a local maximum of likelihood by iteratively applying these two steps.

The most important advantage is the simplicity of its implementation. An intuitive interpretation of the EM algorithm is that predicting values for unobservable variables depending on current parameters and observed variables, i.e., $k(\mathbf{x}|\mathbf{y}; \mathbf{\Psi}^{(k)})$ and considering that values as complete data, the computation of the maximum likelihood estimates for complete data is performed. For most problems, the computation of maximum likelihood estimates for complete data is very easy – in some cases, it falls into the calculation of relative frequencies. Thus, the implementation of the EM algorithm tends to be easy.

There are some criticisms for the EM algorithm. The convergence of the EM algorithm is slow in some cases. It is not a global optimization algorithm. For the first problem, some methods are proposed to speed-up the convergence [17, 25, 20]. If the application suffers from the second problem, i.e., local optima, we may need the methods such as [33] which utilize annealing.

2.2 Hidden Markov Models

In this section we describe hidden Markov models and basic algorithms for them. Because of their simplicity and efficiency, hidden Markov models are used to model a time series data in various applications in natural language processing such as speech

recognition [19, 29], part-of-speech tagging [10, 21], named-entity extraction [5], and information extraction [13].

First, we introduce building blocks of HMMs and two algorithms, the forward-backward algorithm and the Viterbi algorithm. Next, we describe the Baum-Welch algorithm, which is used to estimate parameters of HMMs from incomplete (untagged) date. Finally, we note some issues to be cautious of when we implement Baum-Welch algorithm in the following section.

2.2.1 Hidden Markov Models

Let $X_t \in \{s_1, s_2, \cdots, s_N\}$ denote a discrete random variable that takes N possible values at time t. A series of random variables $\{X_t\}$ is said to be the nth order Markov process when X_t depends only on the previous n random variables (*Markov property*). Formally,

$$P(X_t|X_1^{t-1}) = P(X_t|X_{t-n}, \cdots, X_{t-1}).$$

In general, an nth order Markov process can be transformed to first order Markov process by introducing new random variable $Y_t = \{X_t, \cdots, X_{t+n-1}\}$. Thus, we assume first order Markov process in the following explanation. The Markov process is also said to be *stationary* if $P(X_t|X_{t-1})$ does not depend on the time:

$$P(X_t = s_i|X_{t-1} = s_j) = p(s_i|s_j).$$

A stationary Markov process can be specified by $N \times 1$ *initial probabilities* π and $N \times N$ *transition probabilities* $A\{a_{ij}\}$:

$$\begin{aligned} \pi_i &= P(X_t = s_i), \\ a_{ij} &= P(X_t = s_i|X_{t-1} = s_j). \end{aligned}$$

An hidden Markov model (HMM) is a stochastic process whose hidden states $Q_t \in \{q_1, q_2, \cdots, q_N\}$ form a stationary Markov process and emit observations $O_t \in \{o_1, o_2, \cdots, o_M\}$

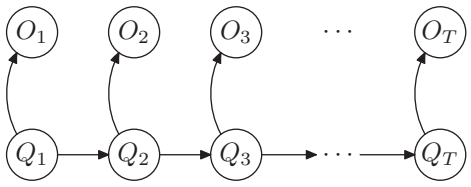


Figure 2.1: Probabilistic structure of an hidden Markov model (state emission model)

stochastically. Throughout this thesis we use state emission type HMMs where each state emits one observation symbol with probability $P(O_t = o_i|Q_t = q_j)$ (Figure 2.1). In addition to initial probabilities and transition probabilities, we need $N \times M$ *emission probabilities* $B\{b_{ij}\}$ to specify an HMM. Then an HMM can be specified by

$$\begin{aligned} \pi_i &= P(Q_t = q_i), \\ a_{ij} &= P(Q_t = q_j|Q_{t-1} = q_i), \\ b_{ij} &= P(O_t = o_j|Q_t = q_i). \end{aligned}$$

We denote this whole set of parameters as $\lambda = \langle \pi, A, B \rangle$. Then the probability that the HMM specified by λ generates the observation o_1^T (We write a sequence $\{o_i, o_{i+1}, \cdots, o_{j-1}, o_j\}$ as o_i^j) with the state transitions q_1^T, i.e., the joint probability $P(o_1^T, q_1^T|\lambda)$ is calculated as

$$P(o_1^T, q_1^T|\lambda) = \pi(q_1)b(q_1, o_1) \prod_{t=1}^{T-1} \{a(q_t, q_{t+1})b(q_{t+1}, o_{t+1})\}.$$

By summing this probability over all possible state transitions naively, the probability that the HMM specified by λ generates the observation o_1^T , i.e., the *likelihood* $P(o_1^T|\lambda)$ can be calculated as

$$P(o_1^T|\lambda) = \sum_{q_1^T} P(o_1^T, q_1^T|\lambda).$$

However this calculation is clearly intractable even for HMMs with small N. Fortunately, there is an efficient dynamic programming algorithm which utilizes Markov property and calculates $P(o_1^T|\lambda)$ in time $O(TN^2)$. We describe this algorithm – the forward-backward algorithm – in the next section.

2.2.2 Forward-Backward Algorithm

We define *forward probabilities* and *backward probabilities* as follows.

Forward Probability $\alpha_t(i)$**:** The probability that the model λ generates the observations o_1^t and is in state q_i at time t, i.e., $P(o_1^t, Q_t = q_i|\lambda)$.

Backward Probability $\beta_t(i)$**:** The probability that the model λ generates the rest of the observations o_{t+1}^T starting from state q_i at time t, i.e., $P(o_{t+1}^T|Q_t = q_i, \lambda)$.

The forward and backward probabilities are used extensively in applications of HMMs. For instance, the forward and backward probabilities can be used to calculate the likelihood $P(o_1^T|\lambda)$, which is one of the measurements for model goodness. Also, the forward and backward probabilities are used at the first step of the Baum-Welch algorithm which estimates model parameters from incomplete data. This extensive use of the forward and backward probabilities is due to the efficient algorithm – the forward-backward algorithm. The forward-backward algorithm calculates these probabilities by the following

procedures.

Forward Procedure:

Initialization:

$$\alpha_1(i) = \pi_i b_i(o_1), \qquad 1 \le i \le N.$$

Recursion:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a(q_i, q_j)\right] b(q_j, o_{t+1}), \qquad 1 \le t \le T-1, 1 \le j \le N.$$

Backward Procedure:

Initialization:

$$\beta_T(i) = 1, \qquad 1 \le i \le N.$$

Recursion:

$$\beta_t(i) = \sum_{j=1}^{N} a(q_i, q_j) b(q_j, o_{t+1}) \beta_{t+1}(j), \qquad T-1 \ge t \ge 1, 1 \le i \le N.$$

The forward-backward algorithm has the time complexity of $O(TN^2)$. After the calculation of the forward and backward probabilities finished, we can calculate $P(o_1^T|\lambda)$ by summing up the forward probabilities at time T as

$$P(o_1^T|\lambda) = \sum_{i=1}^{N} \alpha_T(i),$$

or summing up the backward probabilities at time 1 as

$$P(o_1^T|\lambda) = \sum_{i=1}^{N} \beta_1(i) \pi_i b(q_i, o_1).$$

More generally, $P(o_1^T|\lambda)$ can be calculated as

$$\begin{aligned} P(o_1^T|\lambda) &= \sum_i \sum_j \alpha_t(i) a(q_i, q_j) b(q_j, o_{t+1}) \beta_{t+1}(j) \quad \forall t \quad 1 \le t \le T-1 \\ &= \sum_i \alpha_t(i) \beta_t(i) \qquad\qquad\qquad\qquad\qquad\qquad \forall t \quad 1 \le t \le T-1. \end{aligned}$$

2.2.3 Viterbi Algorithm

We are also interested in finding the most probable state transitions given the observations. The Viterbi algorithm computes the most probable state transitions through a procedure similar to forward procedure in the forward-backward algorithm [34]. The difference is that at each time step the Viterbi algorithm memorize the best probability and the previous state that yields it for each state. Backtracking the stored previous states recovers the most probable state transitions. Before we describe the Viterbi algorithm we define *best probabilities* and *best previous states* as follows.

Best Probabilities $\delta_t(i)$**:** The highest probability of being in state i at time t among all state transitions, i.e., $\max_{q_1,q_2,\cdots,q_T} P(Q_t = q_i, o_1^T|\lambda)$.

Best Previous States $\psi_t(i)$**:** The state at time $t-1$ of the state transition that yields $\delta_t(i)$.

Now, we describe the Viterbi algorithm:

Initialization:

$$\delta_1(i) = \pi_i b(q_i, o_1), \qquad 1 \le i \le N$$

$$\psi_1(i) = 0 \qquad 1 \le i \le N.$$

Recursion:

$$\delta_t(j) = \max_{1\le i\le N} \left[\delta_{t-1}(i)a(q_i, q_j)\right] b(q_j, o_t), \qquad 2 \le t \le T, 1 \le j \le N$$

$$\psi_t(i) = \operatorname*{argmax}_{1\le i\le N} \delta_{t-1}(i)a(q_i, q_j), \qquad 2 \le t \le T, 1 \le j \le N.$$

Termination:

$$\hat{p} = \max_{1\le i\le N} \delta_T(i),$$

$$\hat{q_T} = \operatorname*{argmax}_{1\le i\le N} \delta_T(i).$$

Backtracking:

$$\hat{q_t} = \psi_{t+1}(\hat{q_{t+1}}), \qquad T-1 \ge t \ge 1.$$

Although the Viterbi algorithm has the same time complexity of $O(TN^2)$ as the forward-backward algorithm, we can compute the most probable transitions by add operations of floating point numbers since if we hold probabilities as logarithm, we obtain

$$\log \delta_t(j) = \max_{1 \le i \le N} \left[\log \delta_{t-1}(i) + \log a(q_i, q_j) \right] + \log b(q_j, o_t).$$

2.2.4 Baum-Welch Algorithm

In the context of HMM, the EM algorithm had been known as the Baum-Welch algorithm [3] before Dempster et al. formulated the EM algorithm generally. The reestimation formulas in the Baum-Welch algorithm can be easily derived from the definition of $Q(\mathbf{\Psi}, \bar{\mathbf{\Psi}})$ for an HMM by using the Lagrange multiplier method. We show only the result here.

To describe the reestimation formulas, we first define $\gamma_t(i)$, the probability of being state i at time t, given the model λ and the observations o_1^T, i.e.,

$$\gamma_t(i) = P(Q_t = q_i | o_1^T, \lambda).$$

By using Bayes' rule, we can write $\gamma_t(i)$ as follows using forward and backward probabilities.

$$\begin{aligned} \gamma_t(i) &= \frac{P(Q_t = q_i, o_1^T | \lambda)}{P(o_1^T | \lambda)} \\ &= \frac{\alpha_t(i)\beta_t(i)}{P(o_1^T | \lambda)}. \end{aligned}$$

We next define $\xi_t(i, j)$, the probability of being in state q_i at time t and in state q_j at time $t + 1$, given the model λ and the observations o_1^T, i.e.,

$$\xi_t(i, j) = P(Q_t = q_i, Q_{t+1} = q_j | o_1^T, \lambda).$$

By the same argument for $\gamma_t(i)$, we can write $\xi_t(i, j)$ as

$$\begin{aligned}\xi_t(i, j) &= \frac{P(Q_t = q_i, Q_{t+1} = q_j, o_1^T|\lambda)}{P(o_1^T|\lambda)} \\ &= \frac{\alpha_t(i)a(q_i, q_j)b(q_j, o_{t+1})\beta_{t+1}(j)}{P(o_1^T|\lambda)}.\end{aligned}$$

By summing $\gamma_t(i)$ and $\xi_t(i, j)$ over the time t except for $t = T$, we obtain two quantities which can be interpreted as follows.

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from state } q_i$$

$$\sum_{t=1}^{T-1} \xi_t(i, j) = \text{expected number of transitions from state } q_i \text{ to state } q_j.$$

Using these $\gamma_t(i)$ and $\xi_t(i, j)$, the reestimation formulas of the Baum-Welch algorithm are written as

$$\begin{aligned}\bar{\pi_i} &= \text{expected number of transitions from } q_i \text{ at time 1} \\ &= \gamma_1(i) \\ &= \frac{\alpha_1(i)\beta_1(i)}{\sum_{i=1}^N \alpha_T(i)} \\ \bar{a_{ij}} &= \frac{\text{expected number of transitions from } q_i \text{ to } q_j}{\text{expected number of transitions from } q_i} \\ &= \frac{\sum_{t=1}^{T-1} \xi_t(i, j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \\ &= \frac{\sum_{t=1}^{T-1} \alpha_t(i)a(q_i, q_j)b(q_j, o_{t+1})\beta_{t+1}(j)}{\sum_{t=1}^{T-1} \alpha_t(i)\beta_t(i)} \\ \bar{b_{ij}} &= \frac{\text{expected number of emissions of symbol } o_j \text{ in state } q_i}{\text{expected number of times being in state } q_i} \\ &= \frac{\sum_{t=1}^{T} \delta(O_t, o_j)\gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \\ &= \frac{\sum_{t=1}^{T} \delta(O_t, o_j)\alpha_t(i)\beta_t(i)}{\sum_{t=1}^{T} \alpha_t(i)\beta_t(i)},\end{aligned}$$

where δ is Kronecker's delta, i.e.,

$$\delta(O_t, o_j) = \begin{cases} 1 & O_t = o_j \\ 0 & \text{otherwise.} \end{cases}$$

In the Baum-Welch algorithm, the calculation of the forward and backward probabilities, γ and ξ corresponds to the E-Step of the EM algorithm, and this reestimation of the parameters corresponds to the M-Step. Most of the time of one iteration is consumed at the calculation of the forward and backward probabilities. Therefore, the Baum-Welch algorithm also has the time complexity of $O(TN^2)$.

2.2.5 Implementation Issues

Scaling

As described in Section 2.2.2, the calculation of the forward and backward probabilities involves a number of multiplications of small numbers in the range of $[0, 1]$, which can easily lead to underflow to zero. To prevent this, some *scaling* techniques must be applied [19, 29, 10]. By scaling the probabilities at each time step by some scaling factor to make them fall into a constant range, we can prevent underflow. The technique described here [10] scales the forward probabilities $\alpha_t(i)$ to $\hat{\alpha}_t(i)$ so that

$$\sum_{i=1}^{N} \hat{\alpha}_t(i) = 1 \quad \text{for all } t.$$

The modified forward-backward algorithm is as follows.

Forward Procedure:

Initialization:

$$\tilde{\alpha}_1(i) = \alpha_1(i) \qquad 1 \le i \le N$$

$$c_1 = \left[\sum_{i=1}^{N} \tilde{\alpha}_1(i)\right]^{-1}$$

Recursion: $1 \le t \le T-1$

$$\hat{\alpha}_t(i) = c_t \tilde{\alpha}_t(i) \qquad 1 \le i \le N$$

$$\tilde{\alpha}_{t+1}(j) = \left[\sum_{i=1}^{N} \hat{\alpha}_t(i) a_{ij}\right] b_j(o_{t+1}) \qquad 1 \le j \le N$$

$$c_{t+1} = \left[\sum_{i=1}^{N} \tilde{\alpha}_{t+1}(i)\right]^{-1}.$$

Backward Procedure:

Initialization:

$$\hat{\beta}_T(i) = \beta_T(i) = 1 \qquad 1 \le i \le N$$

$$\tilde{\beta}_T(i) = \hat{\beta_T}(i) \qquad 1 \le i \le N$$

Recursion: $T-1 \ge t \ge 1$

$$\hat{\beta}_t(i) = \sum_{j=1}^{N} a_{ij} b_j(o_{t+1}) \tilde{\beta}_{t+1}(j) \qquad 1 \le i \le N.$$

$$\tilde{\beta}_t(i) = c_t \hat{\beta}_t(i) \qquad 1 \le i \le N$$

Note that $\hat{\alpha}_t(i) = C_1^t \alpha_t(i)$ and $\hat{\beta}_t(i) = \beta_t(i) C_{t+1}^T$ where

$$C_i^j = \prod_{t=i}^{j} c_t.$$

$P(o_1^T|\lambda)$ can not be calculated directly since it would be out of the machine precision (that is the reason why we introduced the scaling). By using scaling factors c_t calculated

in the modified forward-backward algorithm, $\log P(o_1^T|\lambda)$ can be calculated as follows.

$$\begin{aligned}\log P(o_1^T|\lambda) &= \log \sum_{i=1}^{N} \alpha_T(i) \\ &= \log \frac{1}{C_1^T} \underbrace{\sum_{i=1}^{N} \hat{\alpha}_T(i)}_{=1} \\ &= -\sum_{t=1}^{T} \log c_t.\end{aligned}$$

When the scaling is used at the forward-backward probability calculation of the Baum-Welch algorithm, the reestimation formulas need to be modified. However, the modification is simple as follows, just replacing $\alpha_t(i)$ with $\hat{\alpha}_t(i)$, and $\beta_t(i)$ with $\hat{\beta}_t(i)$, except for the addition of the term c_{t+1} in the reestimation formula for a_{ij}.

$$\begin{aligned}\bar{\pi_i} &= \frac{\hat{\alpha}_1(i)\hat{\beta}_1(i)}{\sum_{i=1}^{N} \hat{\alpha}_T(i)} \\ &= \hat{\alpha}_1(i)\hat{\beta}_1(i) \\ \bar{a_{ij}} &= \frac{\sum_{t=1}^{T-1} \hat{\alpha}_t(i)a(q_i, q_j)b(q_j, o_{t+1})\hat{\beta}_{t+1}(j)c_{t+1}}{\sum_{t=1}^{T-1} \hat{\alpha}_t(i)\hat{\beta}_t(i)} \\ \bar{b_{ij}} &= \frac{\sum_{t=1}^{T} \delta(O_t, o_j)\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{t=1}^{T} \hat{\alpha}_t(i)\hat{\beta}_t(i)}.\end{aligned}$$

Multiple Observations

When we apply the Baum-Welch algorithm to the training of a character-based HMM in our approach, we consider each kth sentence $S_k = o^{(k)}{}_1^{T_k}$ in the texts is independently generated by the HMM. Then the probability to be maximized is as follows.

$$P(Text|\lambda) = \prod_{k=1}^{K} P(S_k|\lambda) \tag{2.1}$$

$$= \prod_{k=1}^{K} P(o^{(k)}{}_1^{T_k}|\lambda). \tag{2.2}$$

This situation is called *multiple observations*. The modification to the single observation Baum-Welch algorithm is straightforward. In the numerator and the denominator of the

reestimation formulas, we sum the statistics for each sentence as

$$\bar{\pi_i} = \frac{\sum_{k=1}^{K} \gamma_1^{(k)}(i)}{K}$$
$$\bar{a_{ij}} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \xi_t^{(k)}(i,j)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k-1} \gamma_t^{(k)}(i)}$$
$$\bar{b_{ij}} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \delta(O_t^{(k)}, o_j) \alpha_t^{(k)}(i) \beta_t^{(k)}(i)}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \alpha_t^{(k)}(i) \beta_t^{(k)}(i)}.$$

We can easily introduce the scaling to the multiple observation Baum-Welch algorithm in the same way we introduced it to the single observation Baum-Welch algorithm.

Decomposing training data into independent multiple sequences will make it easy to apply some techniques for faster training such as incremental estimation [17] and parallelization. In addition, in the case where word-based HMMs are trained, we might not need to perform the scaling since each observation is not long.

2.3 Maximum Entropy Framework

In this section, we describe the maximum entropy framework. Recently, the maximum entropy framework has been successfully applied to many natural language processing tasks such as machine translation, part-of-speech tagging, and named entity recognition [4, 30, 6]. One of the most important advantages of the maximum entropy framework is that it can combine diverse forms of cues, known as *features*, in a principled manner, and does not assume any condition on their distributions such as the independence of each feature. Therefore, we can automatically obtain the same effect as backoff or smoothing by introducing a bag of overlapping features at ease.

The maximum entropy method estimates a probability distribution $p(x, y) \in \mathcal{P}$ given training data $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$. In our estimation of a tag emission model, x corresponds to a tag and y corresponds to the contexts that the tag emission model can access to determine output the tag at each time step. To focus on a certain part of the

contexts, we define a binary-valued function, called a *feature function*, and denote a set of feature functions as

$$\mathcal{F} = \{ f_i : (x, y) \mapsto \{0, 1\}, i \in \{1, 2, \cdots, n\}\}.$$

As mentioned above, these feature functions do not need to be independent of each other. For example, in a tag emission model which we describe in Chapter 3.3, feature functions look like as follows.

$$f_1(x, y) = \begin{cases} 1 & x = t_5 \wedge \text{current-state}(y) = q_3 \\ 0 & \text{otherwise} \end{cases}$$

$$f_2(x, y) = \begin{cases} 1 & x = t_5 \wedge \text{current-state}(y) = q_3 \wedge \text{previous-state}(y) = q_1 \\ 0 & \text{otherwise} \end{cases}$$

We define an *empirical distribution* as

$$\tilde{p}(x, y) = \frac{Count(x, y)}{N},$$

where $Count(x, y)$ counts the number of times that (x, y) appeared in the training data and N is the traing data size. Then, we denote the empirical expectation of f_i as

$$E_{\tilde{p}}[f_i] = \sum_{x,y} \tilde{p}(x, y) f_i(x, y).$$

Similarly, the expectation of f_i with respect to the distribution $p(x, y)$ is denoted as

$$E_p[f_i] = \sum_{x,y} p(x, y) f_i(x, y).$$

The maximum entropy method requires that $E_p[f_i] = E_{\tilde{p}}[f_i]$, i.e.,

$$\sum_{x,y} p(x, y) f_i(x, y) = \sum_{x,y} \tilde{p}(x, y) f_i(x, y).$$

This requirement is called a *constraint equation*, or a *constraint*.

We define the subset of $\mathcal{P}$, which consists of distributions which satisfy the constraint equations, as follows.

$$C = \left\{ p \in \mathcal{P} \mid E_p[f_i] = E_{\tilde{p}}[f_i] \quad i \in \{1, 2, \cdots, n\} \right\}.$$

The maximum entropy method selects the distribution which is most uniform among the models in C. The uniformity is measured by the entropy defined as

$$H(p) = -\sum_{x,y} p(x,y) \log p(x,y).$$

Then, the maximum entropy method selects the model p^* with the maximum entropy:

$$p^* = \underset{p \in C}{\operatorname{argmax}}\, H(p)$$

It can be proved that that there is always a unique model p^*, and p^* is in the form:

$$p_\Lambda(x,y) = \frac{1}{Z_\Lambda} \exp\left(\sum_i \lambda_i f_i(x,y)\right),$$

where λ_i is the weight for the feature f_i and Z_Λ is a normalization constant such that

$$Z_\Lambda = \sum_{x,y} \exp\left(\sum_i \lambda_i f_i(x,y)\right).$$

The values for λ_i can be estimated by numerical methods such as the Improved Iterative Scaling (IIS) algorithm [28].

2.4 Related Work

Cutting et al. [10] applied a word-based hidden Markov model and its unsupervised learning, the Baum-Welch algorithm, to English POS tagging to achieve the same goal as our domain adaptability. They used word dictionary to reduce the time complexity of the training and achieved state-of-the-art accuracy for English POS tagging. Merialdo [21] also applied the Baum-Welch algorithm to improve English POS tagging which uses word-based hidden Markov models. However, they concluded that the Baum-Welch algorithm does not improve the accuracy when we have sufficiently large tagged corpus and can calculate good initial parameters for the Baum-Welch algorithm. We believe that this is because the state-tag relation assumed at the estimation of initial parameters changes as the Baum-Welch algorithm proceeds.

In Japanese morphological analysis, it is harder to apply Baum-Welch type algorithm in word-based approach because of the ambiguity in word boundaries. Yamamoto [35] used word-based HMM and the Baum-Welch type unsupervised learning, utilizing existing rule-based morphological analyzer, JUMAN[36], to generate the word network.

Character-based methods are proposed mainly for word segmentation in the languages such as Japanese and Chinese where there is no word segmentation marker [37, 27, 22, 31].

Chapter 3

Two-step Model for Morphological Analysis

In this chapter, we describe our approach to morphological analysis which consists of two parts, a tag-independent character-based HMM and a tag emission model.

3.1 Overview

Our goal is to achieve domain adaptability and tag adaptability defined as follows.

Domain adaptability The fitting of the probabilistic model's parameters to the target text type.

Tag adaptability The method's ability to handle any tag set when required by another NLP system which comes after morphological analysis.

To achieve domain adaptability and tag adaptability at the same time, we divide parameter estimation into the estimation of a character-based tag-independent HMM and the estimation of a tag emission model. Domain adaptability is achieved by unsupervised learning of a character-based tag-independent HMM. On the other hand, tag adaptability is achieved by this division of parameter estimation and by the employment of an HMM

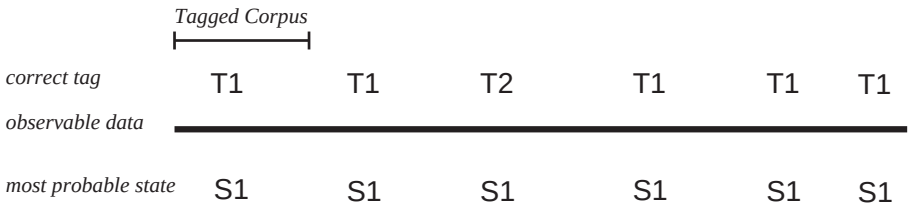


Figure 3.1: How the state-tag relation can be estimated from a small corpus

with a sufficiently large state space to contain information required for morphological analysis with any tag set.

An intuitive explanation for why unsupervised learning of a character-based HMM has an effect on making the amount of required tagged corpus smaller is as follows. HMMs have been successfully applied to morphological analysis, combined with the Baum-Welch algorithm. We expect that, trained by the Baum-Welch algorithm from a raw corpus large enough to capture the statistical tendencies, a character-based HMM becomes to yield a certain fixed Viterbi state sequence for the character sequences which must be equally tagged. In other words, we expect that the hidden states classify characters according to their corresponding tags. When the number of states is less than the number of output characters, it also can be said that the hidden states perform *smoothing* of output characters. The ideal situation is illustrated in Figure 3.1. The figure shows the case where the trained model is in state $S1$ for 5 times as the most probable state for input data, and for 4 times out of 5 times the correct tag the system must output is $T1$. When we relate $S1$ to $T1$ by viewing one cooccurrence in the tagged corpus, the system will achieve the accuracy (precision) of 0.8 (= $4/5$) with respect to state $S1$ and tag $T1$.

Our method for Japanese morphological analysis then consists of two estimation phases and two runtime phases:

Estimation:

1. Unsupervised estimation of a character-based HMM by the Baum-Welch algorithm using an untagged corpus.

2. Supervised estimation of a tag emission model using a tagged corpus and the trained character-based HMM.

Runtime:

1. Find the Viterbi path given a sentence.

2. Emit the most probable tags using the tag emission model.

In Section 3.2, we describe the unsupervised training of a character-based HMM by the Baum-Welch algorithm. In Section 3.3, we describe tag emission models and their estimation. In Section 3.4, we describe the runtime tagging procedures.

Although we want to train an HMM with a large state space in addition to the state-tag separation to achieve tag adaptability, the Baum-Welch algorithm is not applicable because it has the time complexity of $O(TN^2)$. To enable training of an HMM with a large state space, we apply Gibbs sampling to the standard Baum-Welch algorithm. We leave the description of this Gibbs sampling version of the Baum-Welch algorithm for the next Chapter.

3.2 Training of character-based HMM

To achieve tag adaptability, we have divided parameter estimation into the estimation of a character-based tag-independent HMM and the estimation of a relation between HMM's hidden states and the part-of-speech tags. In this section, we describe the issues of the first estimation, i.e., the estimation of a character-based tag-independent HMM.

The training of a character-based HMM is performed by the Baum-Welch algorithm using an untagged raw corpus. Because our method does not assume prior relation between hidden states and tags, we can train an HMM with arbitrary number of states. On

the other hand, since we cannot obtain tag-specific initial parameters, we start training from such as random parameters. In the experiments, we usually trained each HMM starting from random initial parameters. More uniform initial parameters might be desirable in terms of preventing overfitting. However, it seems in our case that uniformity leads to slower convergence and has not significant changes in the final accuracy of the task. In the experiments, we train HMMs using two raw training data, a small ($< 10,000$ sentences) one and a large ($> 100,000$ sentences) one, and discuss the required amount of the raw corpus to train a HMM.

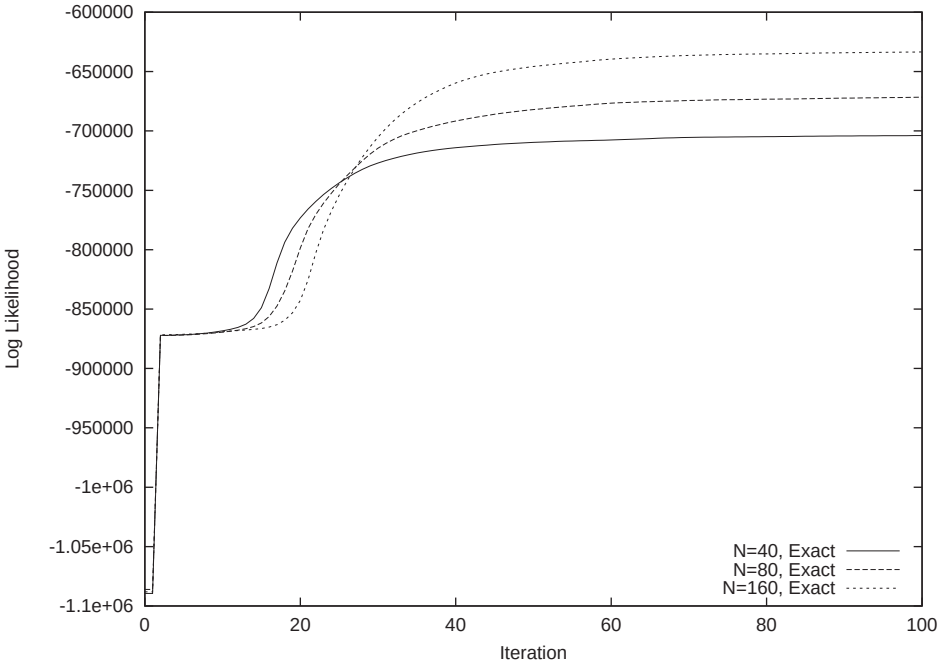
3.2.1 Reducing Output Symbols

Since a Japanese character is usually represented in 2 bytes, which require 65536 array indexes, it is not efficient to use character code itself as the output symbol. To reduce the size of the output symbol set, i.e. M, we consider the most frequent characters as distinct output symbols and other characters as the symbol *UNK*, which stands for unknown characters. We varied the number of distinct symbols in the following experiments to see the effect of the output symbol reduction.

3.2.2 Stopping Criterion

In the Baum-Welch algorithm, the likelihood is exactly obtained in the forward-backward probability calculation. Therefore, we can use the change in the likelihood as a stopping criterion. In our experiments, however, we force the training to stop at the 100th iteration, rather than check convergence at each iteration. This is because the likelihood slightly increases even at the iteration such as 100 or 200, and the amount of this increase cannot be distinguished from the amount of the slight increase in the initial stage of the training. Figure 3.2 shows the training curves of the Baum-Welch algorithm applied to Part 1 of the Kyoto corpus ($M = 501$) described in Chapter 5.

Figure 3.2: Training curve of the Baum-Welch algorithm



3.2.3 Augmenting Output Symbols with Character Types

Here, we describe the augmentation of the HMM's output symbols with character type information. In Japanese, there are several distinct character types such as *kanji*, *hiragana*, *katakana*, *alphabet*, *number*, and *special*. Examples of these character types are shown in Table 3.1. Each character type has its own role, as hiraganas are often used to write functional words or verb conjugation suffixes, katakanas are used for the words from foreign languages. Therefore, the changes in character types are significant clues to find the word boundary. To incorporate this significant clues into the character-based HMMs, we tried augmenting the symbols which the HMMs output. For each output symbol o_i, we assign one of two new output symbols, represented as $\langle o_i, 0 \rangle$ and $\langle o_i, 1 \rangle$, where 0 and 1 are determined by the change in character type which is defined

Kanji	日 木 川 走 上 人 動 来 行 犬 海 山 言 ···
Hiragana	あ い う え お か き く け こ さ し す ···
Katakana	ア イ ウ エ オ カ キ ク ケ コ サ シ ス ···
Alphabet	A B C D E F G H I J K L M ···
Number	１ ２ ３ ○ 一 二 三 十 千 万 億 兆 京 ···
Special	。 、 ？ ！ （ ） 「 」 ％ ＝ ・ ● ◇ ···

Table 3.1: Character types in Japanese

as follows.

$$\text{type-change}(o_t) = \begin{cases} 1 & \text{if char-type}(o_t) \neq \text{char-type}(o_{t+1}) \\ 0 & \textit{otherwise}, \end{cases}$$

where, char-type(o_{T+1}) is defined as the type which is not equal to any character type. As a result, the size of the new symbol set becomes $M \times 2$.

For example, suppose that we have a symbol set $\{0, 1, \cdots, 99\}$ and a set of character types $\{C(\textit{kanji}), H(\textit{hiragana}), K(\textit{katakana}), A(\textit{alphabet}), N(\textit{number}), S(\textit{special})\}$. We assign a new symbol code $o_t^{new} \in \{0, 1, \cdots, 199\}$ to each symbol o_t in a sentence. In this case, o_t^{new} is calculated as

$$o_t^{new} = \text{type-change}(o_t) \times 100 + o_t.$$

The following illustrates this assignment process.

	1	人	の	男	が	ガ	ン	ジ	ス	川	を	渡	っ	て	い	る	。
Symbol code	50	2	11	5	13	27	21	22	28	6	17	6	15	19	12	18	61
Character type	N	C	H	C	H	K	K	K	K	C	H	C	H	H	H	H	S
Change in type	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0	1	1
New code	150	102	111	105	113	27	21	22	128	106	117	106	15	19	12	118	161

We consider the sentence as a sequence of these new codes when we train or use character-based HMM. We found that this symbol augmentation significantly improve the accuracy of word segmentation. Details of the effect of this symbol augmentation are shown in the experiments chapter.

3.3 Tag Emission Models

This section describes the details of tag emission models. Tag emission models are estimated using a tagged corpus and a character-based HMM estimated from an untagged corpus. For each character, a tag emission model determines which tag is the most probable tag in the task's tag set, $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_{|\mathcal{T}|}\}$, by viewing the most probable state transitions for the character sequence, and some other contexts. We investigated two types of emission models. The first is a simple frequency-based model, and the second is a more complicated model based on the maximum entropy framework.

3.3.1 Frequency-based Model

UNIMAP

UNIMAP is the most simple tag emission model that emits a tag which is determined only by a state at the character position in a Viterbi path. That is each state is mapped to a unique tag. We estimate the most probable tag for each state q_i by counting the cooccurrence of the state and the tag in a training tagged corpus. We denote a training tagged corpus as

$$\left\{\langle o^{(1)}, \tau^{(1)}\rangle, \langle o^{(2)}, \tau^{(2)}\rangle, \cdots, \langle o^{(K)}, \tau^{(K)}\rangle\right\},$$

where $\tau^{(k)}$ is a tag sequence annotated to for a kth sentence $o^{(k)}$. We also denote the Viterbi path for the kth sentence as $v^{(k)}$. By using these notations, we can write the estimation of the state-to-tag mapping, $unimap(q_i)$, as follows.

$$\begin{aligned} unimap(q_i) &= \operatorname*{argmax}_{\tau_j \in \mathcal{T}} P(\tau_j | q_i) \\ &= \operatorname*{argmax}_{\tau_j \in \mathcal{T}} \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \delta(q_i, v_t^{(k)}) \delta(\tau_j, \tau_t^{(k)})}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \delta(q_i, v_t^{(k)})} \end{aligned}$$

Since the estimation of the UNIMAP model relies on the conditional probability of only one state, statistically reliable estimation can be achieved with a small amount

of corpus. Therefore, for our final goal of minimizing the amount of tagged corpora needed to construct a probabilistic model, the most ideal case is that each hidden state of the trained HMM deterministically corresponds to a certain tag. In this case, we can estimate that relation from a very small tagged corpus where each state occurs only few times.

However, there is no guarantee that an HMM is trained as such, because we decided to have no constraint which leads to such training, in order to keep the training independent of a task or a tag set as far as possible. Therefore, we propose another model to incorporate multiple cues to help the model determining appropriate tags by viewing other contexts rather than one state in a Viterbi path.

BIMAP

In natural language processing, using higher order *n*-grams such as *bi*gram and *tri*gram is a straightforward and well-known way to incorporate multiple cues. In the same way, we can extend a UNIMAP model to determine a tag at time t by viewing states at time $t-1$ and time t. We call this model BIMAP. Then the mapping $bimap(q_i, q_j)$ becomes

$$\begin{aligned} bimap(q_i, q_j) &= \underset{\tau_j \in \mathcal{T}}{\operatorname{argmax}} P(\tau_j | q_{i_1}, q_{i_2}) \\ &= \underset{\tau_j \in \mathcal{T}}{\operatorname{argmax}} \frac{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \delta(q_{i_1}, v_{t-1}^{(k)}) \delta(q_{i_2}, v_t^{(k)}) \delta(\tau_j, \tau_t^{(k)})}{\sum_{k=1}^{K} \sum_{t=1}^{T_k} \delta(q_{i_1}, v_{t-1}^{(k)}) \delta(q_{i_2}, v_t^{(k)})}. \end{aligned}$$

3.3.2 Incorporating Multiple Cues in the Maximum Entropy Framework

Using many cues to determine tags as in the BIMAP model increases the data sparseness. That is, more tagged corpora are needed for statistically reliable estimation, and therefore it seems to conflict with our aim of minimizing the amount of tagged corpora needed to construct a probabilistic model. However, by employing the maximum entropy framework to combine various cues, we can balance the two demands, the same

performance as the UNIMAP model when the available tagged corpus is small, and more accurate model when a large tagged corpus is available. We can see in the experiments that the maximum entropy method increase the accuracy even when only a small tagged corpus is available.

Features for a Maximum Entropy Tag Emission Model

In this thesis, we use the following four types of features for the maximum entropy tag emission model.

Unigram Features:

$$v_t = q_i \wedge \tau_t = \tau_j \qquad \forall q_i, \tau_j \text{ (always used)} \tag{3.1}$$

$$v_{t-b} = q_i \wedge \tau_t = \tau_j \qquad \forall q_i, \tau_j \quad 1 \leq b \leq UB \tag{3.2}$$

$$v_{t+f} = q_i \wedge \tau_t = \tau_j \qquad \forall q_i, \tau_j \quad 1 \leq f \leq UF \tag{3.3}$$

Bigram Features:

$$v_{t-1} = q_{i_1} \wedge v_t = q_{i_2} \wedge \tau_t = \tau_j \qquad \forall q_{i_1}, q_{i_2}, \tau_j \tag{3.4}$$

$$v_{t-b-1} = q_{i_1} \wedge v_{t-b} = q_{i_2} \wedge \tau_t = \tau_j \qquad \forall q_{i_1}, q_{i_2}, \tau_j \quad 1 \leq b \leq BB \tag{3.5}$$

$$v_{t+f-1} = q_{i_1} \wedge v_{t+f} = q_{i_2} \wedge \tau_t = \tau_j \qquad \forall q_{i_1}, q_{i_2}, \tau_j \quad 1 \leq f \leq BF \tag{3.6}$$

Trigram Features:

$$v_{t-2} = q_{i_1} \wedge v_{t-1} = q_{i_2} \wedge v_t = q_{i_3} \wedge \tau_t = \tau_j \qquad \forall q_{i_1}, q_{i_2}, q_{i_3}, \tau_j \tag{3.7}$$

Character Features:

$$o_t = o_i \wedge \tau_t = \tau_j \qquad \forall o_i, \tau_j \tag{3.8}$$

where t denotes the current position. UB, UF, BB, and BF are constants which control the scope of the features. The feature set 3.1 is equivalent to the UNIMAP model. The

feature set 3.2 and 3.3 slide this one-to-one relation backward and forward respectively to take the effect of distant states into account. The feature set 3.4 corresponds to the BIMAP model, which views two consecutive states in the Viterbi path. The feature set 3.5 and 3.6 slide their scope in the same way as the unigram features. The trigram feature set 3.7 views three consecutive states. The character feature set views the output symbol at that position. Figure 3.3, 3.4, and 3.5 illustrate how these feature types access the contexts.

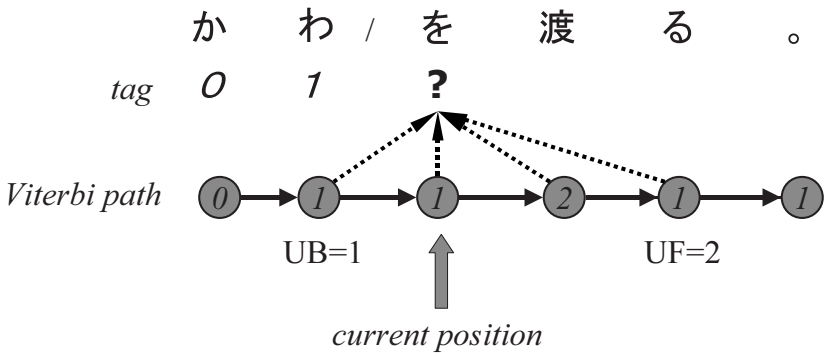


Figure 3.3: Unigram Features

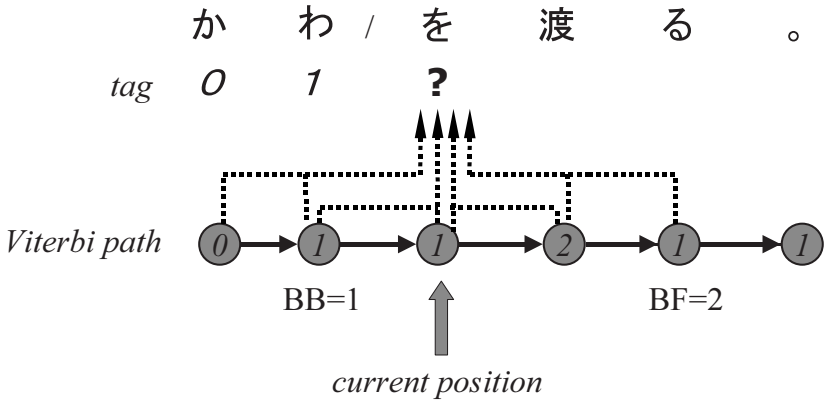


Figure 3.4: Bigram Features

To construct a maximum entropy tag emission model, we first define the feature

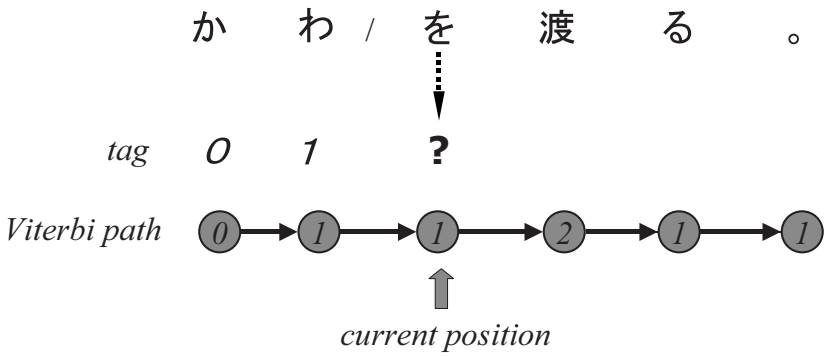


Figure 3.5: Character Features

set. Next, we calculate the empirical distributions of the features, using a tagged corpus and the Viterbi paths for the sentences in that corpus. Then, maximum entropy method estimates the weights for the features according to these empirical distributions. In this study, we used for this estimation the ChoiceMaker Maximum Entropy Estimator [7], which implements the IIS algorithm.

For the maximum entropy tag emission model, the most probable tag at position t is determined as

$$\tau_t^* = \underset{\tau_j \in \mathcal{T}}{\operatorname{argmax}} \frac{1}{Z_\Lambda} \exp\left(\sum_i \lambda_i f_i(t_j, \langle \{\cdots, v_{t-1}, v_t, v_{t+1}, \cdots\}, o_t \rangle) \right).$$

3.4 Tagging Procedure

In this section, we describe the runtime tagging procedure in our two-step model approach. The tagging procedure consists of the Viterbi search using the character-based HMM and the tag emission using the tag emission model.

3.4.1 Character-based tagging

Here, we first describe how to perform word level tagging with a character-based method. In a character-based method, we can perform word segmentation by assigning tag t_{end},

which is for instance represented as"1", to the last character of each word, and tag t_{other} represented as "0" to the other characters. Similarly, morphological analysis (word segmentation + part-of-speech tagging) can be performed by assigning a tag such as *end-of-Y* to the last character of a word which has part-of-speech tag Y. Precisely, we employ the tag schemes which is similar to those employed in [37, 22]. When our task is only the word segmentation, we use the following tag scheme:

$$\begin{cases} t_{end} & \text{for the last character of a word} \\ t_{other} & \text{for other characters.} \end{cases}$$

When our task is the segmentation and the part-of-speech tagging, we use the following tag scheme:

$$\begin{cases} t_{end_i} & \text{for the last character of a word which has POS } p_i \\ t_{other} & \text{for other characters.} \end{cases}$$

3.4.2 The Viterbi Search

Most supervised methods that employ an HMM for morphological analysis consider that an hidden state relates to an unique part-of-speech tag deterministically. Then the parameters of an HMM is estimated using a tagged corpus. To tag a sentence, they deterministically output the most probable tag sequence according to the most probable state sequence which can be computed efficiently by the Viterbi algorithm. Figure 3.6 illustrates the segmentation of a Japanese sentence "かわを渡る。(*kawa-wo wataru* .)" performed by a character-based HMM whose hidden states correspond to the segmentation tags uniquely. The emphasized state sequence in the figure is the Viterbi path.

In our approach, we first find the most probable state sequence by the Viterbi algorithm as illustrated in Figure 3.7 to tag a sentence by using a trained character-based HMM. This is the same step with a state-tag coupled method. We have mentioned that in the initial stages of the training we can not exploit the sparseness because the emission

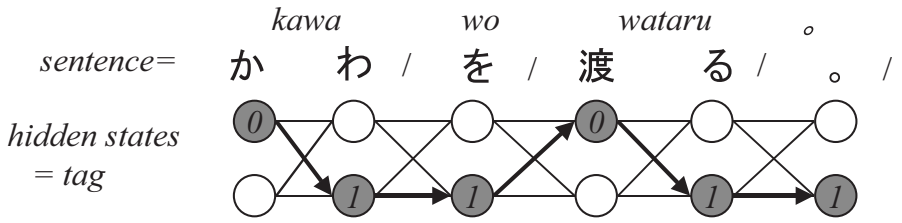


Figure 3.6: Word segmentation by an HMM whose states correspond to segmentation tags

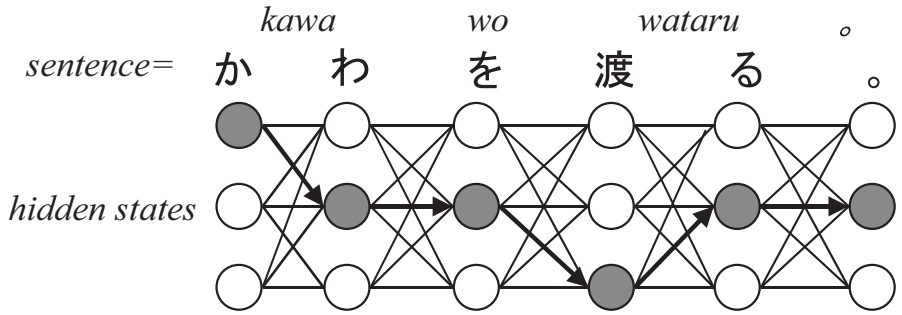


Figure 3.7: Find the Viterbi path given a sentence

probability matrix, i.e., B, is dense. Fortunately, after the training, most of the emission probabilities are close to zero. Figure 3.8 shows the rate of non-zero (> 0.00001) probabilities in B during the Baum-Welch training using the training data with $4,577,956$ characters. Therefore, we can use the Viterbi algorithm which exploits the sparseness to find the Viterbi path. We store for each symbol o_j all possible states, i.e., $\left\{q_i \in \{q_1, \cdots, q_N\} \mid B(q_i, o_j) > \text{threshold}\right\}$, where threshold is set to some small value, for example 0.00001. Because a path which has q_i such that $B(q_i, o_t) \approx 0$ can not be the Viterbi path, we can restrict the max and argmax operation in the Viterbi algorithm to the possible state combinations. Figure 3.4.2 illustrates this algorithm.

After the Viterbi search, we must determine the most probable tag sequence because there is no deterministic mapping between states and tags. For this purpose, we estimate a *tag emission model* beforehand by using a tagged corpus and the trained HMM.

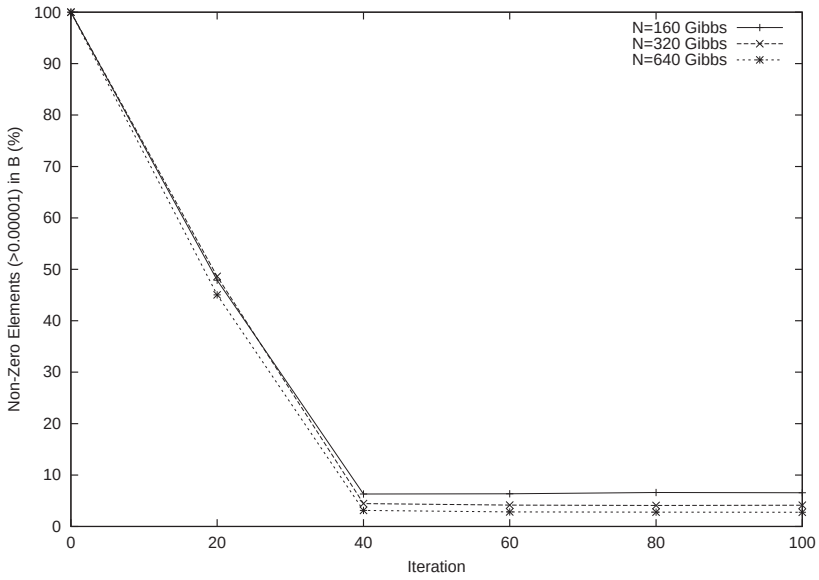


Figure 3.8: The rate of non-zero (> 0.00001) elements in B during the Baum-Welch training ($M = 4,002$, $T = 4,577,956$)

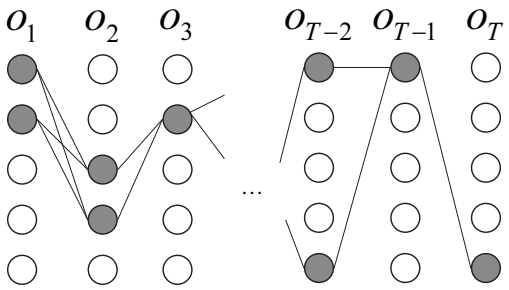


Figure 3.9: The Viterbi algorithm exploiting sparseness of B

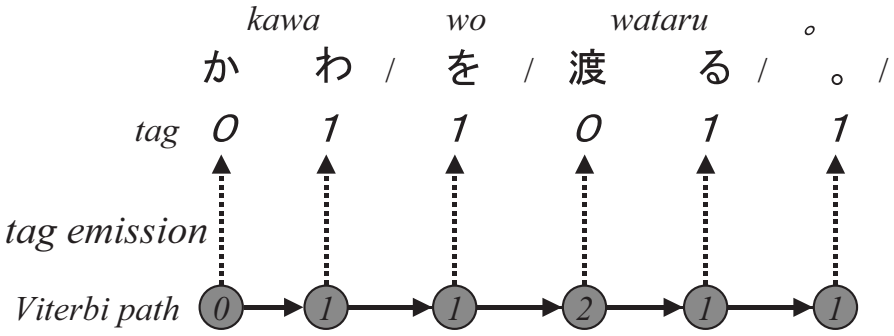


Figure 3.10: Emit tags according to the state in the Viterbi path

3.4.3 Tag Emission

The tag emission model emits tags stochastically according to the contexts such as the Viterbi path and the characters in a sentence. At each character position, the most probable tag t^* is determined by

$$t^* = \underset{t}{\operatorname{argmax}} P(t|\text{context information}).$$

The forms of $P(t|\text{context information})$ depend on the tag emission models.

Figure 3.10 illustrate this process. In this case, the tag emission model is illustrated as it can access only the state at the current character position, i.e., UNIMAP. The most probable tag for state 0 and 2 is “0”, and “1” for state 1.

Chapter 4

Training of HMM with Large State Space

In this chapter, we introduce Gibbs sampling to the standard Baum-Welch algorithm to enable the training of an HMM with a large state space.

4.1 Number of States Needed for Morphological Analysis

The number of distinct POS tags varies according to annotation schemes. For example, the EDR corpus [12] has 31 POS tags. On the other hand, the Kyodai text corpus [18] has hierarchical POS tags, 13 major POS tags at the top level, 109 minor POS tags at the next level, and for each POS which conjugates, it has conjugation types and conjugation forms. Combination of all level tags results in about $1,500$ distinct tags. When we assume a one-to-one mapping between hidden states and tags, we need at least the same number of hidden states as the number of the POS tags. Therefore, to make a model adaptive to any tag set, we require a method which can train an HMM with up to few thousands of states in a practical time.

The standard Baum-Welch algorithm can not be used to train an HMM with large state space because it has the time complexity $O(TN^2)$, where T is the size of the training

data, and N is the number of states of the HMM. In addition, since we do not have strong assumption about the structure of the HMM, the initial parameters that give the sparse probability matrix, will not be given at the beginning of the training. Therefore, the technique which reduces the time complexity of the standard Baum-Welch algorithm by exploiting the sparseness of the emission probability matrix [10] cannot be applied. As a solution of this problem, we introduce Gibbs sampling to the Baum-Welch algorithm.

4.2 Baum-Welch Algorithm via Gibbs Sampling

This section describes a variant of the Baum-Welch algorithm which utilizes Gibbs sampling in the forward-backward calculation to reduce the training time complexity [16]. Since the most of the time is consumed in the forward-backward probability calculation, we try to reduce the time complexity of this step by using some approximated calculation rather than the exact calculation at some, not so large, cost in accuracy.

4.2.1 Approximation by Sampling

Sampling is a method to generate realizations of random variables $\mathbf{X} = \{X_1, X_2, \cdots, X_n\}$ according to the probability distribution $P(\mathbf{X})$. Using a sufficient number of such samples $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(K)}\}$, we can approximate the expectation of a function $a(\mathbf{X})$ with respect to the distribution $\mathbf{X}$ as

$$\begin{aligned} E_P[a(\mathbf{X})] &= \sum_{x_1, x_2, \cdots, x_n} a(x_1, x_2, \cdots, x_n) P(X_1 = x_1, X_2 = x_2, \cdots, X_n = x_n) \\ &\approx \frac{1}{K} \sum_{k=1}^{K} a(x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)}). \end{aligned}$$

In our case, we want to obtain the samples of the state sequence drawn according to the distribution given the observations and the current model parameters, i.e., $P(Q_1, Q_2, \cdots, Q_T | o_1^T, \lambda)$.

Given a number of the samples of the state sequence for the observations, we can calculate the statistics needed to reestimate the model's parameters by counting the state transitions and symbol emissions. Then the reestimation formulas using samples become

$$\bar{\pi_i} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}} \delta(q_1^{(k)}, q_i)}{K}$$

$$\bar{a_{ij}} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}-1} \delta(q_t^{(k)}, q_i)\delta(q_{t+1}^{(k)}, q_j)}{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}-1} \delta(q_t^{(k)}, q_i)}$$

$$\bar{b_{ij}} = \frac{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}} \delta(q_t^{(k)}, q_i)\delta(o_t^{(k)}, o_j)}{\sum_{k=1}^{K} \sum_{t=1}^{T^{(k)}} \delta(q_t^{(k)}, q_i)}$$

where $\{q_1^{(k)}, q_2^{(k)}, \cdots, q_T^{(k)}\}$ is the kth sample for the random variables $\{Q_1, Q_2, \cdots, Q_T\}$ which correspond to the state sequence, and K is the total number of generated samples.

4.2.2 Gibbs Sampling

Among the various sampling methods, *Gibbs sampling* is one of the simplest methods. Gibbs sampling is a Monte Carlo Markov Chain (MCMC) sampling method. This means that each sample is generated by the Markov chain according to the previous sample. Under the suitable conditions, the distribution of the samples converges to the stationary distribution which is equal to the desired distribution from which we want to draw samples [15, 24].

Starting from some initial sample $\{x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(1)}\}$, Gibbs sampler generates the next sample repeatedly. To generate the $(k+1)$th sample from the kth sample, Gibbs sampler updates one component of the vector at a time as follows.

$$\begin{aligned}
&\text{Draw } x_1^{(k+1)} \text{ from } P(X_1|\{\mathbf{X}\}\backslash X_1) \quad \text{given} \quad x_2^{(k)}, x_3^{(k)}, \cdots, x_n^{(k)}.\\
&\text{Draw } x_2^{(k+1)} \text{ from } P(X_2|\{\mathbf{X}\}\backslash X_2) \quad \text{given} \quad x_1^{(k+1)}, x_3^{(k)}, \cdots, x_n^{(k)}.\\
&\qquad\qquad\qquad\qquad\vdots\\
&\text{Draw } x_n^{(k+1)} \text{ from } P(X_n|\{\mathbf{X}\}\backslash X_n) \quad \text{given} \quad x_1^{(k+1)}, x_2^{(k+1)}, \cdots, x_{n-1}^{(k+1)}.
\end{aligned}$$

Note that the new value for X_i is used immediately to draw the next value for X_{i+1}. After some long time steps, we can use the generated samples to calculate statistics of the desired distribution.

Gibbs sampling requires three parameters to be decided:

1. The number of samples discarded until the algorithm reaches the stationary distribution.
2. The number of samples collected after reaching the stationary distribution.
3. The number of runs of above cycle.

However there is no definite method to decide them.

4.2.3 Gibbs Sampler for the Baum-Welch Algorithm

An HMM is considered as a graphical model [14] written as in Figure 2.1. In general, each node in a graphical model is conditionally independent from other nodes given its Markov blanket, defined as the children of a node, parents, and parents of the children. Therefore, for an HMM we get

$$\begin{aligned} P(Q_t|\{\mathbf{Q}\}\backslash Q_t, O_1^T) &= P(Q_t|Q_{t-1}, Q_{t-1}, O_t) \\ &\propto P(Q_t|Q_{t-1})P(O_t|Q_t)P(Q_{t+1}|Q_t). \end{aligned}$$

The pseudo-code for generating $q_t^{(k+1)}$ becomes as Program 1. As we can see, the generation of the next value for one component of the vector costs $O(N)$, where N is the number of states of an HMM. If we generate K samples per each observation symbol to collect statistics, the total time complexity of the Baum-Welch algorithm via Gibbs sampling becomes $O(KNT)$. If we can collect reliable statistics by generating $K(\ll N)$ samples, Gibbs sampling can be used to accelerate reestimation. In our experiments, the settings such as 1 cycle of 20 discarded and 20 collected was found to be enough and thus Gibbs sampling is practical.

```
pstate = q_{t-1}^{(k+1)};
nstate = q_{t+1}^{(k)};
sym = o_t;
acc = 0.0;
r = 0.0;
for(i = 0; i < N; i++){
    acc += a(pstate, q_i) × b(q_i, sym) × a(q_i, nstate);
}
r = random number in [0, 1];
r × = acc;
acc = 0.0;
for(i = 0; i < N; i++){
    acc += a(pstate, q_i) × b(q_i, sym) × a(q_i, nstate);
    if(r ≤ acc){
        q_t^{(k+1)} = q_i;
        break;
    }
}
```

Program 1: Pseudo code for generating $q_t^{(k+1)}$

4.2.4 Timing Comparison with the standard Baum-Welch algorithm

Figure 4.1 shows the timing comparison between the Gibbs sampling version of the Baum-Welch algorithm and the standard Baum-Welch algorithm. The plot is for the training data of 402, 295 output symbols in the symbol set with size 501. The sampling strategy is 1 cycle of 20 discarded and 20 collected (1-20-20). We can see that the Gibbs sampling version significantly reduces the time complexity to the linear. In Chapter 5, we show that while there is the penalty according to the approximation by Gibbs sampling, this 1-20-20 setting actually has the benefit which exceeds the penalty when a large raw corpus is used to train an HMM.

4.2.5 Initial Sample Generation

One remaining question in the Gibbs sampling Baum-Welch algorithm is how to generate the initial sample. The simplest method is to generate the initial sample by assigning each component at random. In addition to that, we tested two initialization methods

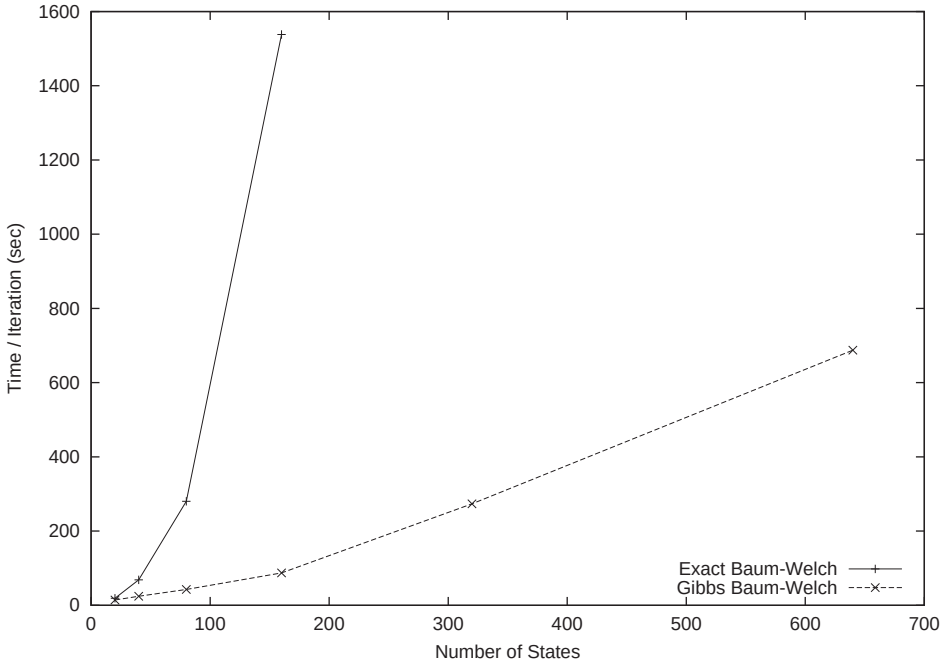


Figure 4.1: Time per iteration: Exact Baum-Welch algorithm vs. Gibbs sampling Baum-Welch algorithm

which intend to make the sampler reach the stationary distribution earlier.

The first method generates the each component of the initial sample greedily according to the conditional probability given the previous component as follows.

Generate $q_1^{(1)}$ according to $\pi(q_i) \times b(q_i, o_1)$

Given $q_t^{(1)}$ and o_{t+1}, generate $q_{t+1}^{(1)}$ according to the distribution:

$$P(Q_{t+1}|Q_t, O_t) \propto P(Q_{t+1}|Q_t)P(O_{t+1}|Q_{t+1}) = a(q_t^{(1)}, q_i) \times b(q_i, o_{t+1}).$$

We call this method as GREEDY. Note that, the cost of this generation of initial sample is less than one full generation of Gibbs sampling.

The second method, called PBEST, uses the sample that yielded the greatest joint probability, i.e., $P(o_1^T, q_1^T)$, at the last Baum-Welch iteration as the initial sample of the current iteration. This method is also not so costly since we only need to compare and store the the sample.

Figure 4.2 shows the comparison between these three initialization methods. Here, we show the goodness of the trained models measured by the (test-set) cross entropy per symbol, HS, defined as follows.

$$\begin{aligned} HS &= -\frac{1}{T} \log P(Text|\lambda) \\ &= -\frac{1}{T} \sum_{k=1}^{K} \log P({o^{(k)}}_1^{T_k}|\lambda), \end{aligned}$$

where T is the total number of symbols in the text, and K is the total number of sentences in the text. Small values indicate good models and roughly indicate good accuracy in morphological analysis. The models are trained using the text of $T = 402,295$ and $M = 501$ from newspapers with 100 Baum-Welch iterations. *Closed* means that we calculated cross entropy using the training text and *open* means that we calculated cross entropy using the text which is not used for training, in this case $T = 372,373$ (the number of sentences is same as the training text). The figures are the averaged

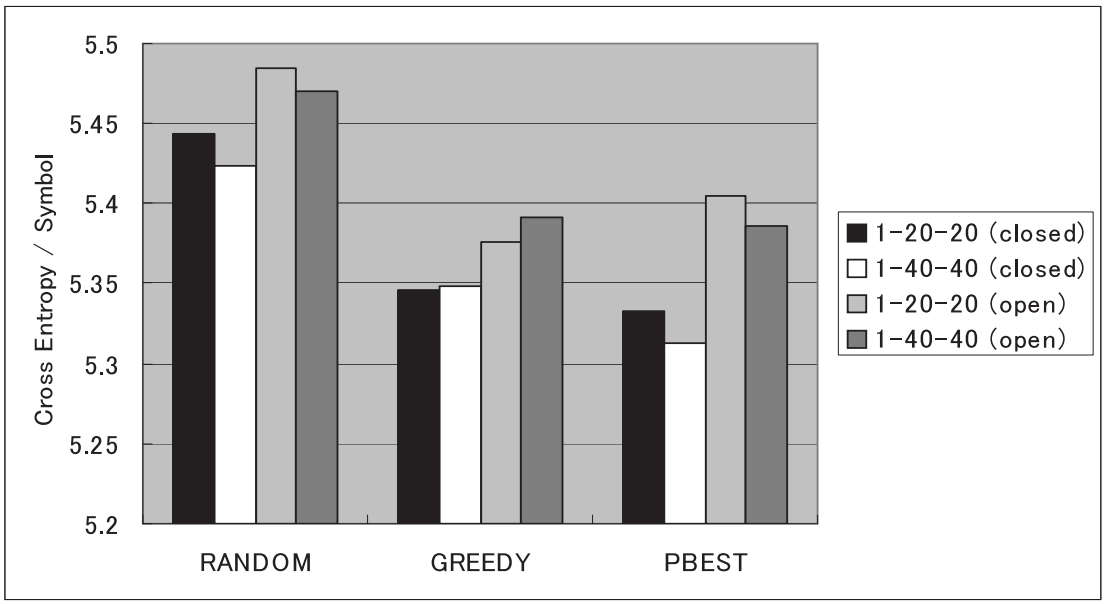


Figure 4.2: Initial sample generation methods and cross entropy per symbol of the trained HMMs

values over 4 runs for the sample generation strategies of 1 cycle of 20 discarded and 20 collected, written as 1-20-20, and 1 cycle of 40 discarded and 40 collected, written as 1-40-40, respectively. We can see that GREEDY and PBEST perform better than the simple RANDOM. In our later experiments, we use the setting of 1-20-20 and GREEDY initialization which marked the highest performance for the open test.

Chapter 5

Experiments

Our final goal is to achieve domain adaptability and tag adaptability. We expect that domain adaptability with only a small tagged corpus is achieved by unsupervised learning of a character-based tag-independent HMM. On the other hand, we expect that tag adaptability is achieved by the division of parameter estimation and by the employment of an HMM with a sufficiently large state space to contain information required for morphological analysis with any tag set. To enable training of an HMM with a large state space, we have introduced Gibbs sampling to the Baum-Welch algorithm. We also proposed symbol augmentation to improve the accuracy of morphological analysis. In this chapter, we show a series of experiments conducted to see the effectiveness of our approach. In particular, we show that our approach is effective for domain adaptability in the sense that we can robustly estimate probabilistic parameters from a small amount of corpus.

In Section 5.3.2, we show the experimental results on word segmentation using the UNIMAP tag emission model and a small raw corpus. Here, we show the relation between the penalty of the Gibbs sampling and the its benefit. When the raw corpus used to train an HMM is small, the benefit of Gibbs sampling is not significant.

In Section 5.3.3, we show the experimental results on word segmentation using symbol augmentation. In this experiment, we show that symbol augmentation greatly im-

proves the accuracy of word segmentation.

In Section 5.3.4, we show the experimental results on word segmentation using an HMM trained with a large raw corpus. Here, we show that when the raw corpus used to train an HMM is sufficiently large, the benefit of a large state space enabled by Gibbs sampling exceeds the penalty according to the approximation by Gibbs sampling.

In Section 5.3.5, we show a series of feature tests for the maximum entropy tag emission model. We show that the maximum entropy tag emission model significantly improves the word segmentation accuracy when the available tagged corpus is large.

In Section 5.3.6, we vary the amount of tagged corpus used to train a tag emission model to see the effect of unsupervised learning of a character-based HMM for domain adaptability.

In Section 5.3.7, we compare the results in Section 5.3.6 with the accuracy of other character-based methods, and show that our method achieves higher accuracy when the available tagged corpus is small.

In Section 5.4, we show the experimental results on morphological analysis briefly.

5.1 Evaluation

The accuracy of morphological analysis is evaluated by using three measures, *character accuracy*, *word recall*, and *word precision*. Character accuracy is defined as

$$\textit{character accuracy} = \frac{\text{number of correctly tagged characters}}{\text{number of characters in the test corpus}}.$$

Word recall and word precision [23] are well-known measures for morphological analysis, which are defined as follows.

$$\textit{word recall} = \frac{\text{number of correctly tagged (segmented) words}}{\text{number of words in the test corpus}},$$

$$\textit{word precision} = \frac{\text{number of correctly tagged (segmented) words}}{\text{number of words the which system outputs}}.$$

Note that character accuracy is an easier measure than word recall and word precision.

	number of sentences	number of words	number of characters
Part 1 (for the closed test)	8,973	228,027	402,295
Part 2 (for the open test)	8,973	210,639	372,373

Table 5.1: Kyoto University text corpus divided into two parts

number of sentences	number of characters
112,429	4,577,956

Table 5.2: A large raw text from Mainichi shinbun

5.2 Data Set

To conduct experiments, we used the Kyoto University text corpus (the Kyoto corpus) [18]. The Kyoto corpus consists of about 20,000 annotated sentences from news articles of Mainichi shinbun, one of the major newspapers in Japan. We divided this corpus into two parts as shown in Table 5.1, one is for the training of an HMM and a tag emission model, and the other is for the open test. We also prepared a large raw text corpus for the training of an HMM, which consists of about 100,000 sentences from Mainichi shinbun [1]. The details are shown in Table 5.2.

5.3 Word Segmentation

Morphological analysis involves word segmentation and part-of-speech tagging. In this section, we show the experimental results on word segmentation.

5.3.1 Baseline Method

To see whether the training of an HMM from raw texts is really significant, we prepare a simple baseline method based on heuristics. This baseline method segments words where the character type changes. Table 5.3 shows the accuracy of this baseline method

For Part 1			For Part 2		
CA	WR	WP	CA	WR	WP
79.51	52.91	61.81	77.53	48.75	58.01

Table 5.3: Word segmentation accuracy for the Kyoto corpus by baseline method using character types. CA, WR, WP means character accuracy, word recall, and word precision respectively.

for the Kyoto corpus. At least our methods must outperform this baseline method.

5.3.2 UNIMAP Emission Model

Tables 5.4, 5.5, and 5.6 show the accuracy of word segmentation by the UNIMAP emission models. To see the effect of the number of states N, the number of symbols M, and the method of the E-Step, exact or Gibbs sampling, we varied these parameters in the experiments. We trained each HMM starting from random initial parameters using Part 1 of the Kyoto corpus as training data. Each emission model is also estimated using all of Part 1 of the Kyoto corpus. As described in Section 3.2.2, we forced the training to stop at the 100th iteration.

From Tables 5.4, 5.5, and 5.6, we can observe a tendency that the accuracy becomes higher as the number of states becomes greater. Tables 5.4, 5.5, and 5.6 also show that the Gibbs sampling E-step has some penalty in accuracy compared with the exact E-step. We display the figures in Table 5.4 as a graph in Figure 5.1 to see more clearly that with $M = 501$ the effect of the larger state space exceeds the penalty of the Gibbs sampling E-Step. However, with other settings, $M = 1001$ and $M = 4001$, we cannot observe this tendency clearly. In particular, when $M = 4001$, the accuracy of the HMM ($N = 640$) trained by the Gibbs sampling Baum-Welch extremely decreases. It should be because of the data sparseness problem.

($M = 501$)

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
40 Exact	87.21	74.38	71.91	87.92	76.13	72.01
80 Exact	86.70	73.55	69.19	87.49	75.36	70.05
160 Exact	88.65	**76.18**	73.83	89.22	**77.48**	74.57
80 Gibbs	86.14	71.37	70.04	86.14	73.02	69.33
160 Gibbs	88.13	75.15	72.88	88.32	76.86	72.75
320 Gibbs	**88.90**	75.95	**75.10**	89.65	77.41	**76.00**
640 Gibbs	88.86	74.49	74.69	**89.78**	76.84	75.85

Table 5.4: Word segmentation accuracy for the Kyoto corpus by the UNIMAP emission model ($M = 501$)

($M = 1,001$)

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
40 Exact	88.03	72.18	74.05	88.70	75.79	74.14
80 Exact	87.94	71.47	74.17	88.45	74.67	73.89
160 Exact	**89.66**	**77.85**	75.71	89.48	77.97	74.52
160 Gibbs	88.22	75.23	72.84	88.47	76.47	72.92
320 Gibbs	88.67	75.49	73.32	89.11	77.24	73.80
640 Gibbs	89.54	77.22	**75.76**	**89.65**	**78.10**	**75.44**

Table 5.5: Word segmentation accuracy for the Kyoto corpus by UNIMAP emission model ($M = 1,001$)

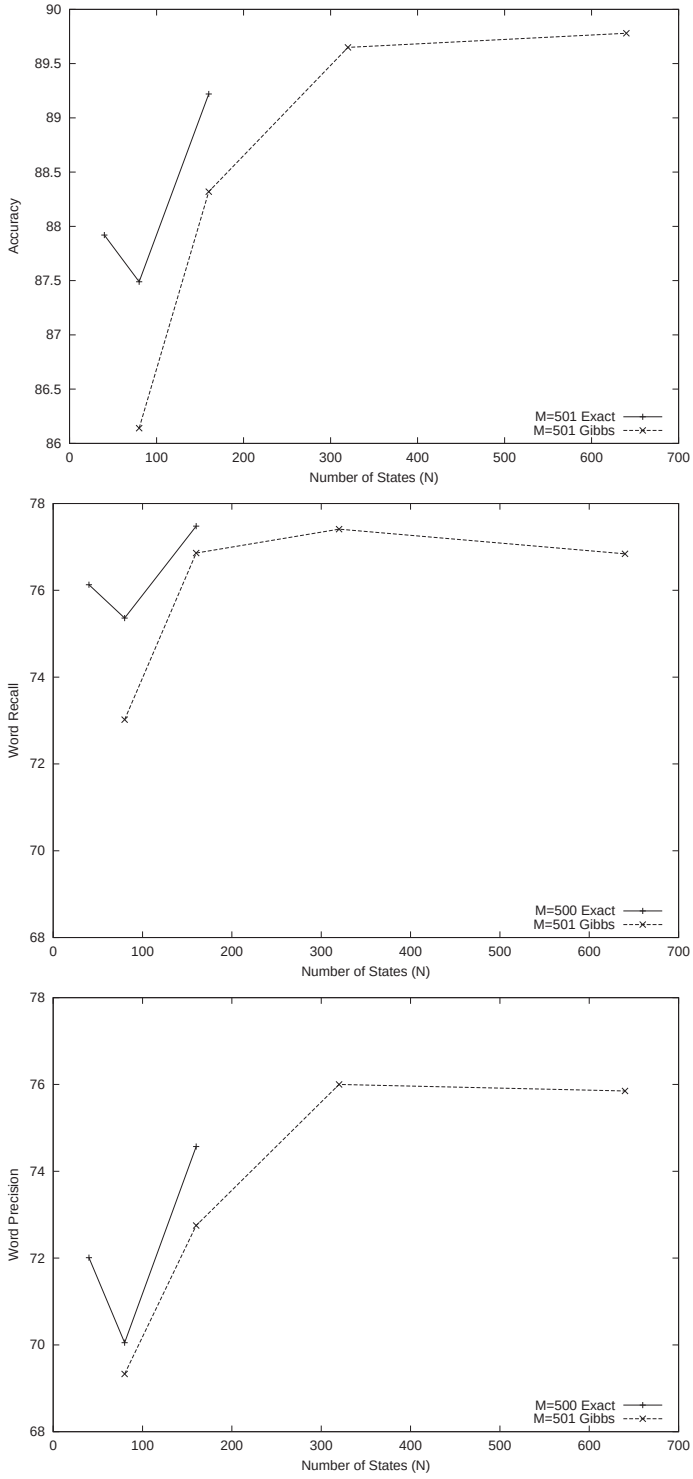


Figure 5.1: Accuracy vs. the number of states for the Kyoto corpus ($M = 501$, open test)

($M = 4,001$)

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
40 Exact	87.50	70.30	72.65	88.12	74.02	73.04
80 Exact	87.46	73.09	72.13	87.24	73.63	71.29
160 Exact	**89.95**	**78.58**	**75.95**	**89.30**	78.39	74.12
160 Gibbs	88.28	75.65	72.44	88.37	76.78	72.09
320 Gibbs	89.36	77.56	75.25	89.28	78.12	**74.27**
640 Gibbs	87.75	73.90	71.51	87.70	74.10	71.01

Table 5.6: Word segmentation accuracy for the Kyoto corpus by the UNIMAP emission model ($M = 4,001$)

5.3.3 Augmenting Output Symbols with Character Types

In this section, we show that the symbol augmentation described in Section 3.2.3 greatly improves the accuracy of word segmentation. Table 5.7 shows the performance of the HMMs trained using Part 1 of the Kyoto corpus where symbols are augmented. Compared with the performance of the HMMs trained using the Kyoto corpus encoded in 1001 symbols (Table 5.5), the augmented version marks about 2 points higher accuracy with almost the same size of the symbol set. Therefore, we focus on the models using this symbol augmentation in the following experiments.

5.3.4 Performance of HMMs trained using a Large Raw Corpus

In Section 5.3.2, we mentioned that the data sparseness problem interfere improvement of accuracy in the case of $M = 1001$ and $M = 4001$. Table 5.8 shows the accuracy of the HMMs trained using the large raw corpus described in Section 5.2, which consists of about $100,000$ sentences. In this case, the accuracy increases as N becomes greater. From this result, we conclude that the data sparseness problem can be solved by using a sufficiently large raw corpus such as the raw corpus with $100,000$ sentences. Combined

($M = 501 \times 2$)

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
40 Exact	88.86	77.16	74.47	88.27	76.90	72.80
80 Exact	88.64	77.59	72.70	87.85	76.19	70.84
160 Exact	91.59	82.09	80.88	91.12	81.75	78.92
160 Gibbs	91.24	82.43	78.86	90.87	81.85	77.69
320 Gibbs	92.26	**83.45**	82.18	91.62	**82.33**	80.23
640 Gibbs	92.21	83.15	82.10	91.77	82.02	80.73
1280 Gibbs	**92.36**	82.73	**82.52**	**91.96**	82.05	**81.18**

Table 5.7: Word segmentation accuracy for the Kyoto corpus by the UNIMAP model with symbol augmentation ($M = 501 \times 2$)

($M = 4001$)

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
40 Exact	86.04	71.54	69.56	86.35	72.49	70.06
160 Gibbs	88.95	76.51	74.35	89.17	78.02	74.08
320 Gibbs	89.96	78.43	76.18	90.35	79.25	76.98
640 Gibbs	**90.35**	**79.27**	**76.75**	**90.73**	**80.32**	**76.99**

Table 5.8: Word segmentation accuracy with HMMs trained using a large raw corpus ($M = 4001$)

with the symbol augmentation, the training using a large raw corpus marks the highest accuracy at this point as shown in Table 5.9.

We display the figures in Tables 5.8 and 5.9 as a graph in Figure 5.1 to see more clearly the effect of symbol augmentation when HMMs are trained using a large raw corpus. We can also observe, from the smooth increase in accuracy along with the number of states, that the data sparseness problem is solved by using a large raw corpus for the training. Though there is only one data point ($N = 40$) for the exact E-Step because of the lack of time (it will take approximately 18 days to train an HMM with $N = 160$

($M = 2001 \times 2$)

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
40 Exact	89.39	78.55	75.98	88.32	76.92	73.43
160 Gibbs	92.02	84.18	80.38	91.06	82.46	77.96
320 Gibbs	92.83	84.59	82.68	92.06	83.83	80.42
640 Gibbs	**93.42**	**85.47**	**84.08**	**92.93**	**84.78**	**82.32**

Table 5.9: Word segmentation accuracy with HMMs trained using a large raw corpus and symbol augmentation ($M = 2001 \times 2$)

by the exact Baum-Welch algorithm), the Gibbs sampling Baum-Welch algorithm outperforms the exact Baum-Welch algorithm at $N = 40$. To find the reason why the Gibbs algorithm outperforms the exact version, we will need further investigations.

5.3.5 Accuracy of Maximum Entropy Tag Emission Models

In this section, we show the accuracy of the maximum entropy tag emission models. As described in Chapter 3.3.2, we use unigram features, bigram features, trigram features, and character features. Table 5.10 shows the results of the experiment where we tried various feature sets. In this experiment, we used the HMMs which are written as "320 Gibbs" and "640 Gibbs" in Table 5.9. Unigram features that view only the current state are always used. UB, UF, BB, and BF in the table have the same meanings as in the definition of the features in Section 3.3.2. Bullets in the "bi", "tri" and "ch" columns indicate that we used bigram features, trigram features, character features, respectively. Note that, by definition, $UB > 0$ does not indicate that we used bigram features defined by equation 3.4 (in page 36). We estimated a maximum entropy tag emission model for each feature set by using Part 1 of the Kyoto corpus. As in the previous experiments, the accuracy was measured for Part 1 and Part 2 of the Kyoto corpus.

We can see that even simple features such as forward unigram features greatly im-

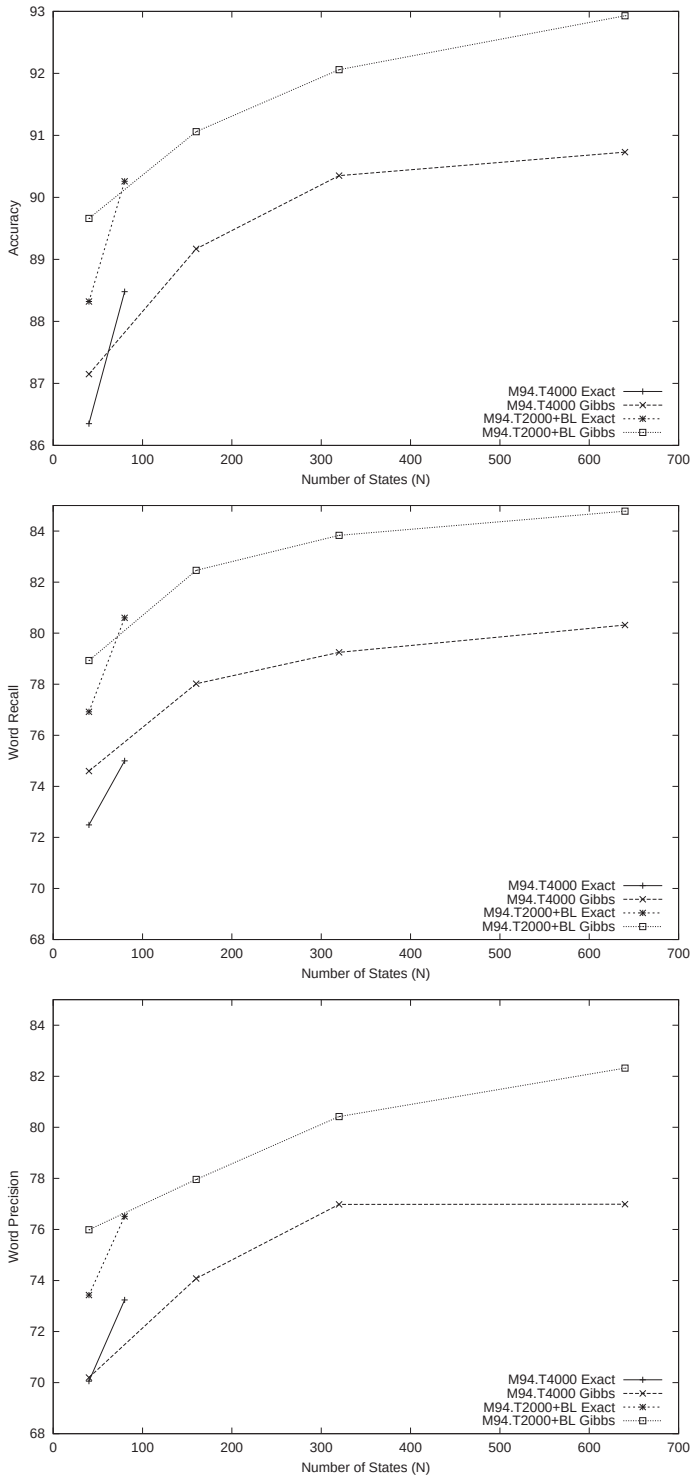


Figure 5.2: Comparison between the normal HMMs and the symbol-augmented HMMs (trained using a large raw corpus)

prove the accuracy (see the rows No. 3 – 7). Comparing the row No. 3 with No. 4, or No. 12 with No. 13, we can see that the effect of the forward features exceeds that of the backward features. The effect of the character features is also significant (see the row No. 10 and No. 18). As a result of these feature tests, we obtained the feature set in the row No. 18 and No. 21 that marks the highest accuracy of 96.67% character accuracy, 92.26% word recall, and 91.84% word precision in the open test.

5.3.6 Accuracy vs. Amount of Tagged Corpus

Figure 5.3 shows the relation between the open test accuracy of word segmentation and the amount of the tagged corpus used to train a tag emission model. In the experiments, we varied the amount of tagged corpus as “10”, “50”, “100”, “500”, “1000”, “5000”, and “all(8973)” sentences. To make clear the differences in the small amount of corpus, the logarithmic scale is used in the x-axis. The HMMs trained from a large raw corpus with symbol augmentation, which were described in Section 5.3.4 are used to conduct this experiments. The setting for the maximum entropy tag emission model, denoted as “MEMAP”, is same as the feature set in the rows No. 18 and No. 21 in Table 5.10 that marked the highest accuracy in the feature tests.

We can observe the followings from the experiments.

1. When the available tagged corpus is large, i.e., 500–10, 000 sentences, the accuracy of UNIMAP model becomes greater as the state space becomes large.

2. On the other hand, when the available tagged corpus is small, i.e., 10–100, the accuracy of UNIMAP model becomes greater as the state space becomes small.

3. The maximum entropy tag emission model greatly improves the accuracy, particularly the word precision, when the tagged corpus is large.

No.	*N*, E-Step	Features							For Part 1 (closed)			For Part 2 (open)		
		UB	UF	bi	BB	BF	tri	ch	CA	WR	WP	CA	WR	WP
1	320 Gibbs	UNIMAP							92.83	84.59	82.68	92.06	83.83	80.42
2	320 Gibbs	BIMAP							93.38	85.35	83.94	92.48	84.11	81.51
3	320 Gibbs	1	0	-	0	0	-	-	92.95	84.90	82.86	92.34	84.41	80.99
4	320 Gibbs	0	1	-	0	0	-	-	94.79	88.00	87.37	94.52	87.83	86.23
5	320 Gibbs	1	1	-	0	0	-	-	95.08	88.81	88.01	94.78	88.49	86.88
6	320 Gibbs	2	2	-	0	0	-	-	95.18	89.08	88.25	94.91	88.95	87.13
7	320 Gibbs	4	4	-	0	0	-	-	95.27	89.26	88.49	94.89	88.80	87.16
8	320 Gibbs	0	0	•	0	0	-	-	93.60	86.11	84.37	92.76	85.15	81.93
9	320 Gibbs	0	0	-	0	0	•	-	94.57	88.09	86.54	92.77	84.81	82.06
10	320 Gibbs	0	0	-	0	0	-	•	94.21	86.95	85.96	93.27	85.12	83.63
11	320 Gibbs	0	0	•	0	0	•	-	94.62	88.24	86.56	92.96	85.39	82.40
12	320 Gibbs	1	0	•	0	0	•	-	94.61	88.21	86.57	92.94	85.33	82.35
13	320 Gibbs	0	1	•	0	0	•	-	96.39	91.64	91.00	94.96	88.77	87.38
14	320 Gibbs	0	2	•	0	0	•	-	96.54	92.04	91.36	95.09	89.12	87.70
15	320 Gibbs	0	3	•	0	0	•	-	96.57	92.09	91.44	95.11	89.17	87.77
16	320 Gibbs	0	3	•	0	1	•	-	97.05	93.02	92.78	95.48	89.77	88.87
17	320 Gibbs	0	3	•	0	2	•	-	97.39	93.79	93.60	95.54	89.91	89.00
18	320 Gibbs	0	3	•	0	2	•	•	**98.26**	**95.85**	**95.75**	**96.38**	**91.64**	**91.25**
19	640 Gibbs	UNIMAP							93.42	85.47	84.08	92.93	84.78	82.32
20	640 Gibbs	2	2	-	0	0	-	-	96.09	91.30	90.45	95.85	90.88	89.56
21	640 Gibbs	0	3	•	0	2	•	•	**98.96**	**97.49**	**97.43**	**96.67**	**92.26**	**91.84**

Table 5.10: Accuracy of the maximum entropy tag emission model: Results are for HMMs trained using a large raw corpus encoded by symbol augmentation

4. In addition, the maximum entropy tag emission model also lifts up the accuracy when the tagged corpus is small.

1. and 2. mean there is a trade-off in the setting of the HMM's number of states between the accuracy with a large tagged corpus and the accuracy with a very small tagged corpus. This can be predicted to some extent from our design of tag emission models. 3. is the same as the expected effect of the maximum entropy emission model. 4. implies that we need careful smoothing or back-off techniques when the available tagged corpus is very small. The maximum entropy framework seems to have the back-off effect and lifts up the accuracy when only 10 sentences are available.

5.3.7 Comparison with Other Character-based Methods

We compared our method, which is displayed in the previous graph (Figure 5.3), with the method which is almost equivalent to the character *n*-gram based method as described in [27, 37]. In this method [32], a word boundary is represented by a special character, for instance, "$< d >$". We estimate character *bi*gram or *tri*gram from a corpus where boundaries are encoded in this way. Then for each character position in a raw sentence we assume two hidden states, "1" for word boundaries, "0" for the others. The probability of being in state j at time t, $P_j(c_1^t)$, can be calculated as

$$\begin{aligned} P_0(c_1^t) =& P_0(c_1^{t-1})p(c_t|c_{t-2}, c_{t-1}) \\ &+ P_1(c_1^{t-1})p(c_t| < d > c_{t-1}) \\ P_1(c_1^t) =& P_0(c_1^{t-1}p(< d > |c_{t-2}c_{t-1})p(c_t| < d >) \\ &+ P_0(c_1^{t-1})p(< d > | < d > c_{t-1})p(c_t| < d >). \end{aligned}$$

We used SRILM (The SRI Language Modeling Toolkit) [2] to estimate this model and to segment a sentence. The most probable state sequence can be estimated by the Viterbi algorithm.

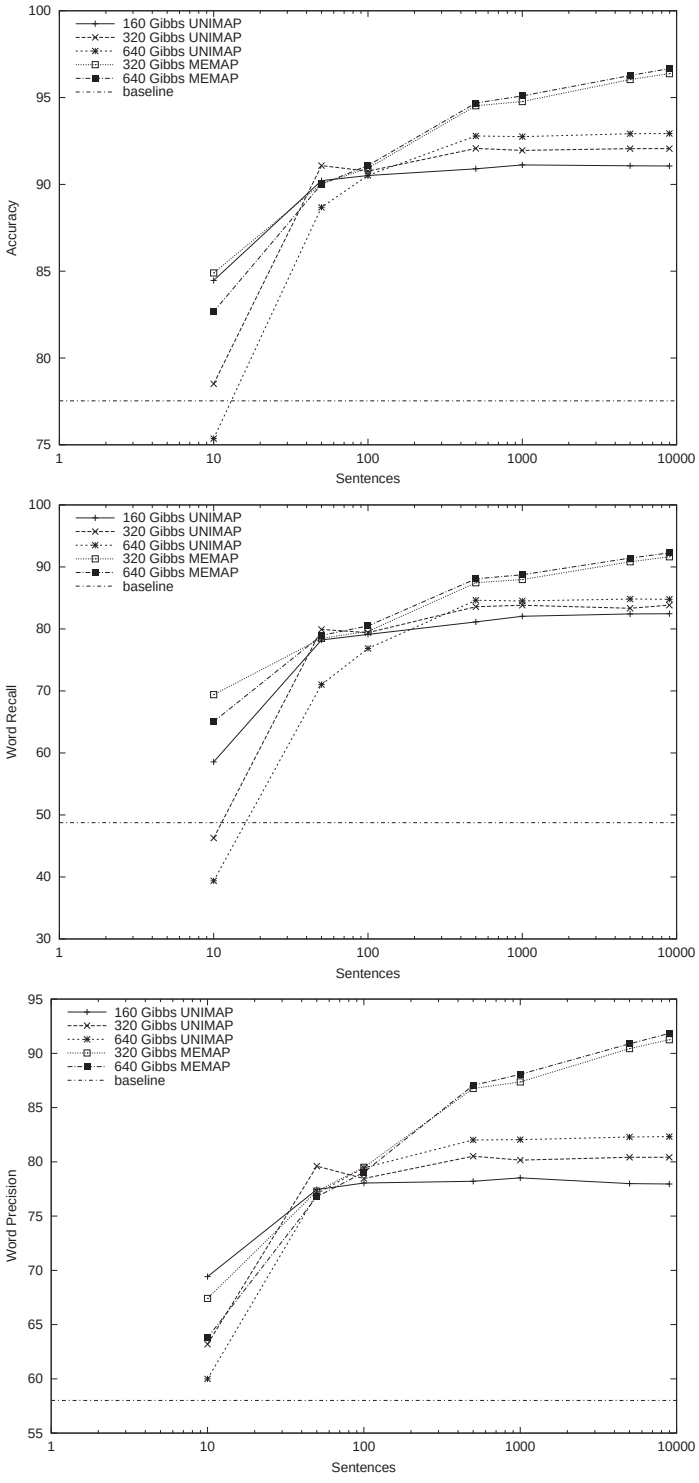


Figure 5.3: Accuracy vs. amount of tagged corpus

Figure 5.4 shows the results. We can see that the decline of the accuracy of our method is not severe compared with the character *tri*gram/*bi*gram methods, denoted as "Ext-Char Tri", "Ext-Char Bi" respectively, though the character *tri*gram method outperforms our method when a large tagged corpus (> 5000) is available. The plot denoted as "Ext-Char Tri + BL" indicates that we trained character *tri*gram using the Part 1 of the Kyoto corpus augmented by symbol augmentation. This plot is to show that symbol augmentation is not the main reason for our method's slow declining.

We also compare our method with HMM-disabled version of our method. We let the HMM return input symbols as the Viterbi sequence. Then we estimate the maximum entropy tag emission model with the setting such as $UB = 2, UF = 2$. That is, these alternatives are the maximum entropy tagging methods with only character features. Figure 5.5 show the results. "Sym ME" indicates that the model views only the current character. "Sym ME 2-2" indicates $UB = 2, UF = 2$. While we can see that the HMM-enabled versions with the maximum entropy tag emission model mark significantly higher accuracies than the HMM-disabled versions, the accuracies of "Sym ME 2-2" at 10 sentences and > 5000 sentences indicate that the character features which view characters other than the current character are also effective. Note that the HMM-enabled versions with the maximum entropy tag emission model include the character features equivalent to "Sym ME", and they show the accuracies in similar curves.

5.4 Accuracy of Morphological Analysis

In this section, we show the experimental results on morphological analysis, i.e., word segmentation and POS tagging. The experiments were conducted using the Kyoto corpus. We also used the HMMs trained using a large raw corpus, i.e., "320 Gibbs" in Table 5.9. In the experiments, we used only the major 13 POS tags. That is, the accuracy is for the tagging of the major POS tags. We trained a UNIMAP tag emission model and a

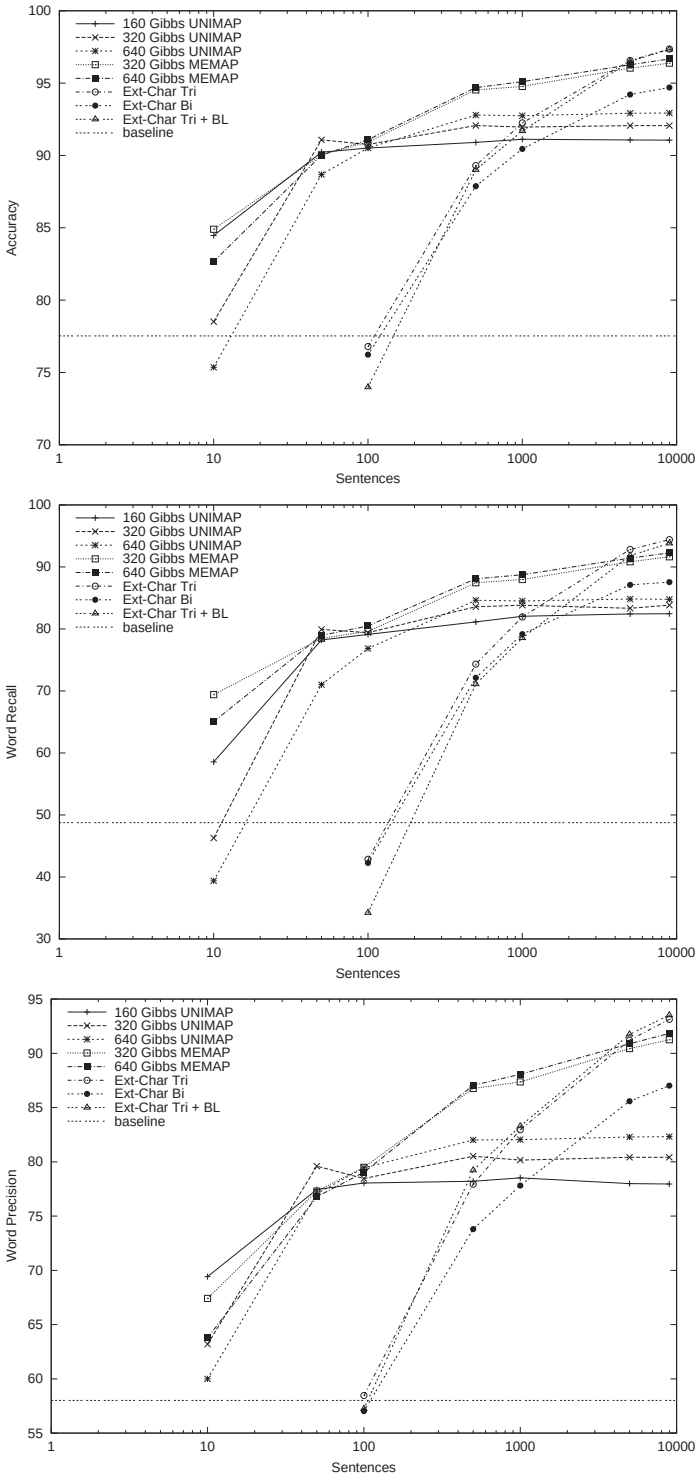


Figure 5.4: Comparison with character trigram and bigram methods

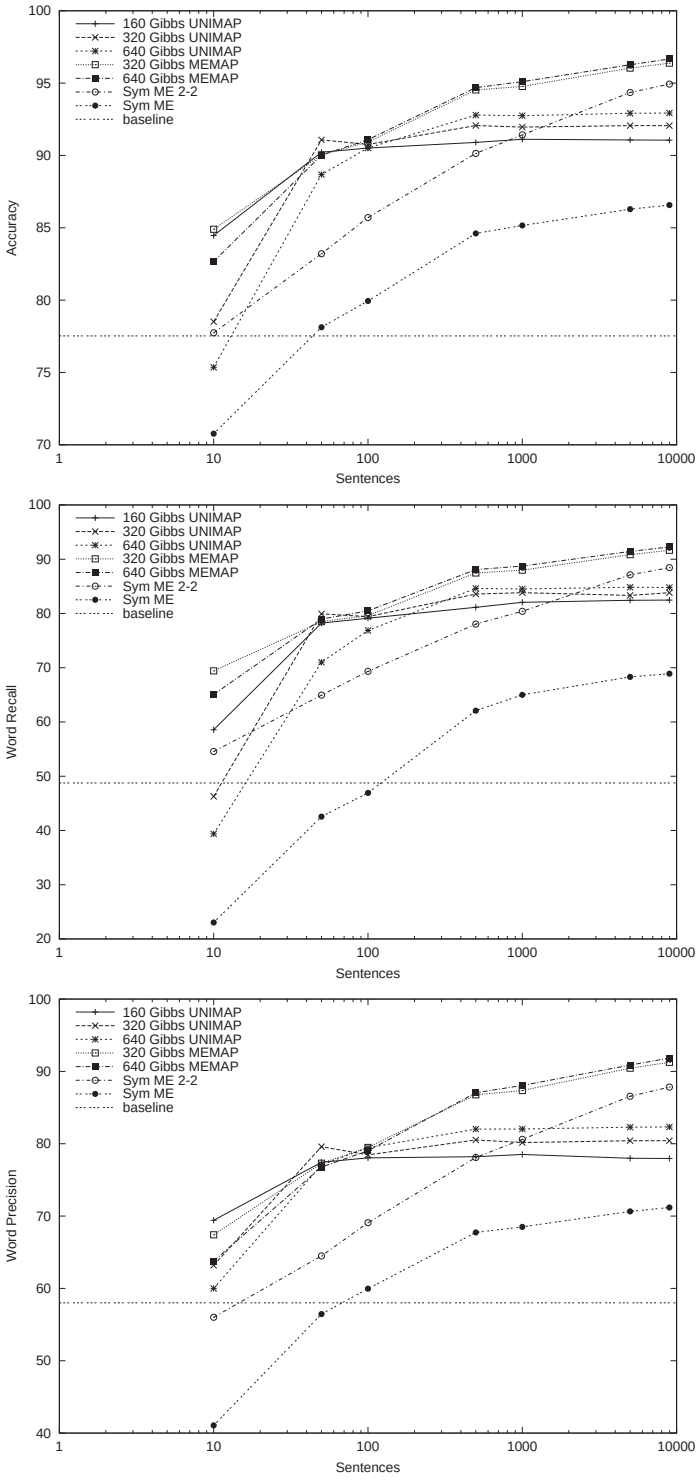


Figure 5.5: Comparison with the HMM-disabled model

N, E-Step	For Part 1 (closed)			For Part 2 (open)		
	CA	WR	WP	CA	WR	WP
320 Gibbs UNIMAP	86.26	76.01	75.19	84.86	74.63	72.75
320 Gibbs MEMAP	97.57	95.05	95.01	93.42	87.60	87.64

Table 5.11: Accuracy of morphological analysis

maximum entropy tag emission model for this tag set. The feature set for the maximum entropy model is same as the row No. 18 in Table 5.10. Table 5.11 shows the results. Although the accuracy of morphological analysis is clearly lower than that of word segmentation, there are the same tendencies with the accuracy of word segmentation.

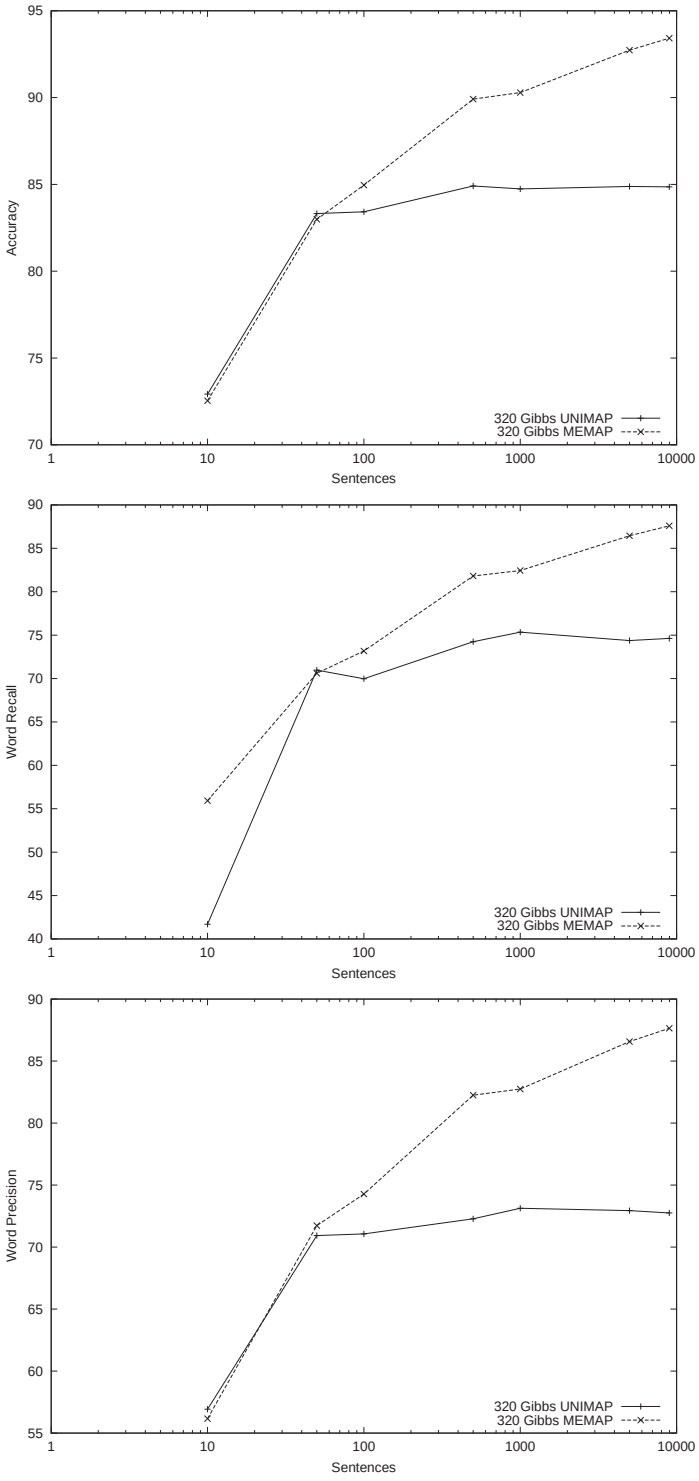


Figure 5.6: Accuracy of morphological analysis (13 major POS)

Chapter 6

Conclusion and Future Work

To achieve adaptive morphological analysis, we proposed a novel stochastic method for Japanese morphological analysis, which consists of a character-based tag-independent HMM and a tag emission model based on the maximum entropy framework. For domain adaptability, we used unsupervised learning of a character-based HMM from a large raw corpus of the domain. For tag adaptability, we proposed a division of the model into a tag-independent HMM and a tag emission model which relates HMM's hidden states to the tags. We also proposed the use of an HMM with a large state space as a tag-independent HMM, and enabled it by applying Gibbs sampling to the Baum-Welch algorithm.

The results of the experiments which compared our method with other character-based method showed that unsupervised learning using the Baum-Welch algorithm greatly improves the accuracy when the available tagged corpus is small. The experimental results also showed that the symbol augmentation with character type and the maximum entropy tag emission model greatly improve the word segmentation accuracy. We also found that while Gibbs sampling Baum-Welch algorithm has the penalty in accuracy compared with the exact Baum-Welch algorithm, the benefit of a large state space enabled by Gibbs sampling can exceed the penalty when a large raw corpus is used to train a character-based HMM. However, there is a trade-off with respect to the

number of states when the available tagged corpus is very small.

This thesis proved that the unsupervised learning for character-based HMMs using the Baum-Welch algorithm is useful for domain adaptability. However, the accuracy achieved by our method is not yet practical. To achieve more practical adaptive morphological analysis, we are planning the following directions of the future research.

- Combination with word-based methods.
- More sophisticated symbol augmentation model or symbol emission model.
- More features in the maximum entropy framework.

Although word-based methods rely on word dictionaries, and therefore on tag sets, there is an obvious advantage that they reduce the ambiguities about word boundaries significantly. Since the maximum entropy framework proved to be robust and successful in the experiments, we should improve the accuracy of morphological analysis without loss of adaptability by introducing the features from word dictionaries in a tag-independent way, and introducing more features from output symbols.

In addition, to achieve more robust estimation with very small tagged corpus, we must explore more appropriate structure of the probabilistic model which is closer to the true distribution. We also require faster algorithm for parameter estimation, since the learning with a large raw corpus becomes very time consuming even though we use Gibbs sampling.

References

[1] CD-毎日新聞'94. http://www.nichigai.co.jp.

[2] SRILM – The SRI Language Modeling Toolkit. Available via http://www.speech.sri.com/projects/srilm/.

[3] L. E. Baum and J. A. Eagon. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model of ecology. *Bull. Amer. Math. Soc.*, 73:360–363, 1967.

[4] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[5] D. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pages 194–201, 1997.

[6] Andrew Borthwick. A Ph.D. dissertation, submitted to Computer Science Department, New York University, 1999.

[7] Andrew Borthwick. ChoiceMaker Maximum Entropy Estimator, 1999. ChoiceMaker Technologies, Inc. Email: borthwic@cs.nyu.edu for information.

[8] Thorsten Brants. Internal and external tagsets in part-of-speech tagging. In *Proc. Eurospeech '97*, pages 2787–2790, Rhodes, Greece, September 1997.

[9] V. Castelli and T. M. Cover. On the exponential value of labeled samples. *Pattern Recognition Letters*, 16(1):105–111.

[10] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Language Processing*, pages 133–140, 1992.

[11] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Statist. Soc. B*, 39:185–197, 1977.

[12] EDR. EDR (Japan Electronic Dictionary Research Institute, Ltd.) electronic dictionary version 1.5 technical guide. Second edition is available via `http://www.iijnet.or.jp/edr/E_TG.htm`.

[13] D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence: Workshop on Machine Learning for Information Extraction*, pages 31–36, 1999.

[14] Brendan J. Frey. *Graphical Models for Machine Learning and Digital Communication*. The MIT Press, 1998.

[15] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (6):721–741, 1984.

[16] Zoubin Ghahramani and Michael I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[17] Yoshihiko Gotoh, Michael M. Hochberg, and Harvey F. Silverman. Efficient training algorithms for HMMs using incremental estimation. *IEEE Transactions on Speech and Audio Processing*, 6(6):539—548, November 1998.

[18] Sadao Kurohashi and Makoto Nagao. Kyoto University text corpus project. In *3rd Annual Meeting of Natural Language Processing*, pages 115–118, 1997. (in Japanese).

[19] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. An introduction to the application of the theory of probabilistic functions of a Markov process to automatic speech recognition. *The Bell System Technical Journal*, 62(4):1035–1074, 1983.

[20] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Wiley & Sons, Inc, 1997.

[21] Bernard Merialdo. Tagging English text with a probablistic model. *Computational Linguistics*, 20(2):155–171, 1994.

[22] Kenji Nagamatsu and Hidehiko Tanaka. A stochastic morphological analysis for Japanese employing character *n*-gram and *k*-nn method. In *NLPRS'97*, pages 23–28, 1997.

[23] Masaaki Nagata. A stochastic Japanese morphological analyzer using a forward-DP backward-A^* N-best search algorithm. In *Proceedings of the 15th International Conference on Computational Linguistics*, pages 201–207, 1994.

[24] Radford M. Neal. Probabilistic inference using Markov Chain Monte Carlo methods. Technical Report CRG-TR-93-1, September 1993.

[25] Radford M. Neal and Geoffrey E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, pages 205–225, 1998.

[26] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39:103–134, 2000.

[27] H. Oda and K. Kita. A character-based Japanese word segmenter using a PPM*-based language model. In *Proceedings of the 18th International Conference on Computer Processing of Oriented Languages (ICCPOL'99)*, pages 527–532, 1999.

[28] S. Pietra, V. Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[29] Lawrence. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.

[30] Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142. Association for Computational Linguistics, Somerset, New Jersey, 1996.

[31] Richard Sproad, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404, 1996.

[32] A. Stolcke and E. Shriberg. Automatic linguistic segmentation of conversational speech. In *Proc. ICSLP '96*, volume 2, pages 1005–1008, Philadelphia, PA, 1996.

[33] N. Udea and R. Nakano. Deterministic annealing EM algorithm. *Neural Networks*, 11:271–282, 1998.

[34] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13:260–267, 1967.

[35] 山本幹雄. Untagged-corpus を用いた形態素解析用 HMM パラメータの一推定法. In **言語処理学会 第 *2* 回年次大会発表論文集**, pages 61–64, 1996.

[36] 黒橋 禎夫 and 長尾 真. 日本語形態素解析システム JUMAN version 3.5, 1998.

[37] 小田 裕樹, 森 信介, and 北 研二. 文字クラスモデルに基づく日本語単語分割. **自然言語処理**, 6, 1999.