PAPER

Part of Speech Tagging with Min-Max Neural Networks

Qing MA[†], **Bao-Liang LU**[††], *Members*, **Hitoshi ISAHARA**[†], *and* **Michinori ICHIKAWA**[††],

SUMMARY A parts of speech (POS) tagging system using neural networks has been developed by Ma et al. This system can tag unlearned data with a much higher accuracy than that of the Hidden Markov Model (HMM), which is the most popular method of POS tagging. It does so by learning a small Thai corpus of the order of ten-thousand words that are ambiguous as to their POS. However, the three-layer perceptron used in the system has slow convergence and low learning accuracy even on such a small amount of data. It is therefore difficult to improve accuracy by incrementing the iteration of learning or by increasing the amount of learning data. To solve this problem, the tagging system of this paper makes use of the min-max modular (M^3) neural network of Lu et al. This new system learns faster and has a higher learning accuracy compared with the old one, by decomposing large, complicated POS tagging problems into many smaller, easier problems. Learning accuracy can be improved by using the same learning data and larger data sets can be learned, which results in a much higher tagging accuracy.

key words: POS tagging, Thai corpus, M^3 neural network, parallel learning, overlearning

1. Introduction

Identifying correct part of speech for words that are ambiguous in context, i.e., parts of speech (POS) tagging, is fundamental to natural language processing. This technique can be applied to the preprocessing of speech synthesis, post processing of OCR and speech recognition, and information retrieval. Although many POS tagging systems have been proposed [1]–[13], these systems need a large amount of learning data, a large corpus whose data has been tagged (for example, on order of a million learning data, for English) to obtain a high enough accuracy. However, for most languages, which are not well studied as English or Japanese in regard to natural language processing, the corpora are still in the development stage. To avoid this problem, we have developed a system that can almost reach a practical level in terms of tagging accuracy [i.e., the accuracy on tagging unlearned data (the data not used in learning)] by using a small amount of data for learning [14]–[17]. It uses neural networks that can be considered to be more robust against the data sparseness problem compared with statistical methods such as HMM. It has been shown to be more accurate in POS tagging than HMM, when a small Thai corpus of the order of ten-thousand words that are ambiguous as to their POSs is used for learning.

However, because the three-layer perceptron used in the POS tagging system suffers from a convergence problem, it is not easy for the system to reach a learning accuracy (i.e., the accuracy on tagging the learning data) higher than 96% even with learning data of the order of ten-thousand words, and the learning takes a considerable amount of time. In addition, overlearning (overfitting) may occur if the system tries to improve learning accuracy by increasing learning time, so there is no guarantee that the tagging accuracy will be improved. Moreover, the perceptron learning may not converge with more learning data. Therefore, it is difficult to improve tagging accuracy by increasing learning iterations or the amount of learning data.

To solve the problems mentioned above, we adopt the Min-Max modular (M^3) neural networks of Lu et al. [18]–[20]. The M^3 network is free of convergence problems and can learn POS tagging problems faster and more accurately compared with our previous system, because it splits a large-scale problem into several easy and small subproblems. Because the M^3 network is free of convergence problems, it becomes possible to significantly decrease the number of units in the middle layer of each module, which results in a low probability of overlearning when raising the learning accuracy. This improves tagging accuracy by either raising the learning accuracy with the same learning data or with learning a larger amount of data.

This paper is organized as follows. First, we formalize the problem of POS tagging and describe the problem decomposition and solution combination, the two intrinsic operations of the M^3 network. Next, we describe the POS tagging method of the M^3 network. Finally, we describe the experimental results and show that a POS tagging system superior to other systems in terms of both its learning and generalization ability.

[†]The authors are with Keihanna Human Info-Communication Research Center, Communications Research Laboratory, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-0289 Japan

[††]The authors are with RIKEN Brain Science Institute, 2–1 Hirosawa, Wako, 351–0198 Japan

2. POS Tagging Problems

Suppose there is a lexicon:

$$V = (w^1, w^2, \cdots, w^v), \tag{1}$$

where the POSs that can be assigned to each word are listed, and suppose there is a set of POSs:

$$\Gamma = (\tau^1, \tau^2, \cdots, \tau^\gamma), \tag{2}$$

where v is the number of registered words and γ is the number of POS. The POS tagging problem is to find a string of POSs $T = \tau_1\tau_2\cdots\tau_s (\tau_i \in \Gamma,\ i = 1, \cdots, s)$ by the following procedure φ when a sentence $W = w_1 w_2 \cdots w_s\ (w_i \in V,\ i = 1, \cdots, s)$ is given.

$$\varphi : W^t \to \tau_t,\ t = 1, \cdots, s \tag{3}$$

Here, t is the index of the target word (the word to be tagged) and W^t is the word sequence which is centered on the target word w_t and has l and r words to the left and right of the target, respectively. That is,

$$W^t = w_{t-l} \cdots w_t \cdots w_{t+r}, \tag{4}$$

where $t - l \geqq 1$, $t + r \leqq s$. POS tagging can therefore be regarded as a classification problem by replacing the POS with a class and can be handled by using neural networks.

3. How to Solve Problems with M^3 Networks

The essential idea of solving problems with M^3 networks is how to decompose a complicated problem into simpler subproblems and how to combine the solutions of the subproblems, which are solved by independent modules, into a complete solution. In this section, we describe the fundamental aspects of the problem decompostion and solution combination. For the details, see [18]–[20].

3.1 Problem Decomposition

In this section, we describe how a complicated and large classification problem with K classes (which we call a K-class problem) can be decomposed into small two-class problems. Let T denote the learning set of the K-class problem:

$$T = \{(X_l, Y_l)\}_{l=1}^{L}, \tag{5}$$

where $X_l \in R^n$ is an input vector, $Y_l \in R^K$ is the desired output vector, and L is the number of learning data.

Generally, a K-class problem can be decomposed into K two-class problems. The learning set of each two-class problem is as follows:

$$T_i = \{(X_l, y_l^{(i)})\}_{l=1}^{L} \tag{6}$$

where $i = 1, \cdots, K$, and $y_l^{(i)} \in R^1$ is the desired output defined as:

$$y_l^{(i)} = \begin{cases} 1 - \epsilon & \text{if } X_l \text{ belongs to the class } C_i \\ \epsilon & \text{if } X_l \text{ belongs to the class } \bar{C}_i, \end{cases} \tag{7}$$

where ϵ is a small positive constant and $\bar{C}_i$ denotes all the classes other than C_i. Equation (6) corresponds to the learning set of the problem where the data belonging to C_i are discriminated from the other data. Later, we will use such a learning set to define an actual classification problem.

In general, the two-class problems defined by Eq. (6) are still large and complicated, so they should be further split into the problems of discriminating the data belonging to C_i from the data belonging to each of the other classes. First, all input vectors X_l $(l = 1, \cdots, L)$ in learning set T are split into subsets according to the class labels. As a result, K subsets of input vectors

$$\chi_i = \{X_l^{(i)}\}_{l=1}^{L_i}, \quad i = 1, \cdots, K \tag{8}$$

are obtained. Here, L_i is the number of input vectors in the i-th subset, so $\sum_{i=1}^{K} L_i = L$. A two-class problem defined by Eq. (6) can then be split into $K - 1$ smaller two-class problems:

$$T_{ij} = \{(X_l^{(i)}, 1 - \epsilon)\}_{l=1}^{L_i} \cup \{(X_l^{(j)}, \epsilon)\}_{l=1}^{L_j}, \tag{9}$$

where $j = 1, \cdots, K\ (j \neq i)$, $X_l^{(i)} \in \chi_i$ and $X_l^{(j)} \in \chi_j$ are input vectors belonging to C_i and C_j, respectively. Therefore, the two-class problem defined by Eq. (9) is to discriminate C_i from C_j. The larger the number of classes K is, the smaller the problem defined by Eq. (9) will be, compared with the two-class problem defined by Eq. (6). In this way, the K-class problem defined by Eq. (5) can be easily split into relatively small $K \times (K - 1)$ two-class problems as defined by Eq. (9).

Even in these $K \times (K - 1)$ two-class problems, however, there will still be complicated problems. These complicated problems, as shown below, can be further split into smaller and simpler problems.

First, suppose the input set χ_i defined in (8) can be split into N_i $(1 \leqq N_i \leqq L_i)$ subsets:

$$\chi_{ij} = \{X_l^{(ij)}\}_{l=1}^{L_i^{(j)}}, \quad j = 1, \cdots, N_i, \tag{10}$$

where $L_i^{(j)}$ is the number of input vectors in the subset χ_{ij} and $\cup_{j=1}^{N_i} \chi_{ij} = \chi_i$ holds. This partition is not unique, so this partition is done randomly. Using such input sets, a two-class problem defined in (9) can be split into $N_i \times N_j$ problems:

$$T_{ij}^{(u,v)} = \{(X_l^{(iu)}, 1-\epsilon)\}_{l=1}^{L_i^{(u)}} \cup \{(X_l^{(jv)}, \epsilon)\}_{l=1}^{L_j^{(v)}}, \tag{11}$$

where $u = 1, \cdots, N_i$, $v = 1, \cdots, N_j$, $X_l^{(iu)} \in \chi_{iu}$ and $X_l^{(jv)} \in \chi_{jv}$ are input vectors belonging to C_i and C_j, respectively.

Therefore, if all the two-class problems defined in (9) are split into problems in (11), the K-class problem is descomposed into $\sum_{i=1}^{K}\sum_{j=1\ (j\neq i)}^{K} N_i \times N_j$ two-class problems. If the splitting is performed such that a learning set has only two different elements (i.e., $L_i^{(u)} = 1$ and $L_j^{(v)} = 1$), Eq. (11) can be expressed as:

$$T_{ij}^{(u,v)} = \{(X_l^{(iu)}, 1-\epsilon) \cup (X_l^{(jv)}, \epsilon)\}. \quad (12)$$

This is clearly a linearly separable problem.

3.2 Solution combination

After the modules learn the split problems, the modules must be integrated to solve the original problem. In this section, we describe how the modules are merged. For the question as to why the network formed by such a merging can solve the original problem, see [19], [20].

We use three units, MIN, MAX, and INV for the integration of modules. Here, let M_{ij} and $\mathrm{M}_{ij}^{(u,v)}$ denote the modules which learn the learning set T_{ij} [Eq. (9)] and $T_{ij}^{(u,v)}$ [Eq. (11)], respectively. Let M_{ij} denote the (i,j)-th element of the $K \times K$ matrix $\mathcal{M}$, and let $\mathrm{M}_{ij}^{(u,v)}$ denote the (u,v)-th element of the $N_i \times N_j$ matrix $\mathcal{M}_{ij}$, which corresponds to the element (i,j) of $\mathcal{M}$.

When solving the K-class problem T [Eq. (5)] by splitting it into $K \times (K-1)$ two-class problems T_{ij} [Eq. (9)], the modules in each row of matrix $\mathcal{M}$ are first merged using the MIN unit, which selects the minimum value from multiple inputs:

$$\mathrm{MIN}_i = \min(\mathrm{M}_{i1}, \cdots, \mathrm{M}_{ij}, \cdots, \mathrm{M}_{iK}), \quad i = 1, \cdots, K\ (i \neq j), \quad (13)$$

where, for simplicity, the symbols of the MIN unit and the modules express their outputs. Next, using the outputs of the K MIN units, the final solution can be obtained as

$$C = \arg\max_i \{\mathrm{MIN}_i\}, \quad i = 1, \cdots, K, \quad (14)$$

where C is the class to which the input data belongs.

If there exists a large two-class problem T_{ij}, it is further split into smaller problems $T_{ij}^{(u,v)}$. In this case, the modules $\mathrm{M}_{ij}^{(u,v)}$ that learn $T_{ij}^{(u,v)}$ are merged as follows. First, N_j modules in each row of matrix $\mathcal{M}_{ij}$ are merged by the MIN unit:

$$\mathrm{MIN}_{ij}^{(u)} = \min(\mathrm{M}_{ij}^{(u1)}, \cdots, \mathrm{M}_{ij}^{(uN_j)}), \quad u = 1, \cdots, N_i. \quad (15)$$

Next, the N_i MIN units are merged into the module M_{ij} by the MAX unit, which selects the maximum value from multiple inputs:

$$\mathrm{M}_{ij} = \max(\mathrm{MIN}_{ij}^{(1)}, \mathrm{MIN}_{ij}^{(2)}, \cdots, \mathrm{MIN}_{ij}^{(N_i)}). \quad (16)$$

The merged module M_{ij} in this way is integrated into Eq. (13).

From the viewpoint of classification, the two-class problems T_{ij} and T_{ji} are identical. The only difference between them is the desired output of the modules. That is, if the desired output of the former problem is $1-\epsilon$ [refer to Eq. (7)], then that of the latter is ϵ. Therefore, M_{ji} can be constructed using M_{ij} and the INV unit, which convert the input values.

4. POS Tagging with M^3 Network

4.1 POS tagging of Thai

Unlike Japanese or English, Thai words have no inflection. That is, the lexical form is fixed even when the word is in a different position or performs different roles in a sentence. Moreover, it often happens that one word serves multiple POSs, such as verb and preposition, or verb and adverb. These facts imply that POS tagging of Thai is more difficult than in Japanese or English (for more details, see [21]). On the other hand, the ambiguity rates (the rate of words ambiguous in POSs in the corpus being used) are 35% for English (Brown Corpus [22]), 56% for Japanese (Kyodai Corpus [23]), and 42% for Chinese (Tsinghua University corpus[24]), respectively, but only 19% for Thai (the corpus used in this paper†). It is not clear whether this figure (19%) reflects the actual characteristics of Thai, because the size of the corpus is small. Regardless, the tagging accuracy shown in this paper is measured only for the words that are ambiguous in POS, so the result will not be influenced by the ambiguity rate.

4.2 Decomposition of the POS tagging problem

The Thai corpus consists of 10,452 sentences. We treat a randomly selected 8,322 sentences of the corpus as the learning data and the remaining 2,130 sentences as the testing data. The learning data contains 124,331 words, of which 22,311 words are ambiguous in POS. The testing data contains 34,544 words, of which 6,717 words are ambiguous in POS. We use only ambiguous words for the learning of the network. Although 47 kinds of POSs are defined in Thai [21], [25], only 38 of them appear in the corpus. Therefore, the POS tagging problem here can

†The Thai corpus, ORCHID (Open linguistic Resources CHanelled toward InterDisciplinary research), used in this paper is the outcome of the collaboration project [21] of the Communications Research Laboratory (CRL) and Thai National Electronics and Computer technology Center (NETEC) NECTEC from 1996.

Table 1 Distribution of data belonging to each class.

Class	Number of data		Class	Number of data	
	learning	testing		learning	testing
C_1	3,041	962	C_{20}	4	0
C_2	72	13	C_{21}	76	17
C_3	1,444	385	C_{22}	4	7
C_4	2,400	701	C_{23}	9	2
C_5	1,582	399	C_{24}	57	15
C_6	3,008	1,011	C_{25}	32	11
C_7	12	0	C_{26}	90	34
C_8	3,197	1,008	C_{27}	30	5
C_9	1,537	475	C_{28}	6	1
C_{10}	481	176	C_{29}	88	23
C_{11}	705	233	C_{30}	2	1
C_{12}	787	226	C_{31}	177	58
C_{13}	601	108	C_{32}	6	0
C_{14}	124	38	C_{33}	8	0
C_{15}	906	328	C_{34}	17	1
C_{16}	90	30	C_{35}	20	3
C_{17}	213	49	C_{36}	2	0
C_{18}	875	214	C_{37}	131	37
C_{19}	476	145	C_{38}	1	0

Table 2 Number of subsets of learning data in each class.

	Number of subsets		Number of subsets
N_1	10	N_9	5
N_3	5	N_{11}	2
N_4	8	N_{12}	2
N_5	5	N_{13}	2
N_6	10	N_{15}	3
N_8	10	N_{18}	3

be treated as a 38-class classification problem[††]. The class distributions of the 22,311 learning data and the 6,716 testing data are shown in Table 1.

According to section 3.1, this 38-class classification problem is first uniquely split into $K \times (K-1) = 38 \times 37$ two-class problems T_{ij} $(i, j = 1, \cdots, K,\ j \neq i)$ defined in Eq. (9). However, as shown in Table 1, while the number of learning data in the smallest two-class problem ($T_{36,38}$) is only $2 + 1 = 3$, that of the largest two-class classification problem ($T_{6,8}$) is $3,008 + 3,197 = 6,205$. Since such a problem is still too large, according to section 3.1, we randomly split the data sets of the 12 classes that contain more than 481 data (C_{10}) in Table 1, so that each subset contains 300 or less data [†]. As a result, the two-class problems $T_{i,j}$ containing these large classes are further split into $N_i \times N_j$ two-class problems $T_{ij}^{(u,v)}$ $(u = 1, \cdots, N_i,\ v = 1, \cdots, N_j)$ defined in Eq. (11). Here, N_i and N_j denote the numbers of subsets derived from the learning data in class C_i and class C_j, respectively, as in section 3.1. The numbers of subsets larger than 1 are shown in Table 2 (that the number of subsets is 1 means that the data set of the class has not been split). Therefore, for example, the two-class problem $T_{1,2}$ is split into $N_1 \times N_2 = 10 \times 1 = 10$ subproblems, and $T_{1,3}$ is split into $N_1 \times N_3 = 10 \times 5 = 50$ subproblems. On the other hand, problems like $T_{2,7}$ are not split any further. That is, $N_2 = 1$ and $N_7 = 1$ hold (note that N_2 and N_7 are not on Table 2).

As mentioned in section 3.2, two-class problems T_{ij} and T_{ji} are identical, so the 38-class POS tagging problem is split into

$$\sum_{i=1}^{38} \sum_{j=i+1}^{38} N_i \times N_j = 3,893 \tag{17}$$

smaller two-class problems. Among these, the largest two-class problem is $T_{10,19}$ in which the number of learning data is 481+476=957.

4.3 The M^3 network for POS tagging

4.3.1 Construction

The M^3 network is constructed by merging the above mentioned 3,893 modules that learned the 3,893 two-class subproblems as shown in Figure 1. Each module M_{ij} is constructed as in Figure 2, if the corresponding problem T_{ij} is split (for example, T_{13}, see Table 2). In the example in the figure, M_{13} is composed of 50 modules $\mathrm{M}_{13}^{(u,v)}$ $(u = 1, \cdots, 10,\ v = 1, \cdots, 5)$, because the problem T_{13} is split into $N_1 \times N_3 = 10 \times 5 = 50$ subproblems. M_{ji} $(j > i)$ is constructed by using M_{ij} and the INV unit.

4.3.2 Input and Output

The input vector X (or X_l Eq. [(5)] at the learning phase) of the M^3 network is constructed from a word sequence W^t [Eq. (4)] that is centered on the target word w_t and has l and r words to the left and right of w_t, respectively:

$$X = (x_{t-l}, \cdots, x_t, \cdots, x_{t+r}). \tag{18}$$

Concretely, given the position p $(p = t-l, \cdots, t+r)$ of the word w, x_p, an element of X, is a vector:

$$x_p = (e_{w1}, e_{w2}, \cdots, e_{w\gamma}), \tag{19}$$

where the dimension γ (47) of the vector is equal to the number of the kinds of POSs. The dimension of the input vector X is $\gamma \times (l + 1 + r)$. If the word w appears in the learning data, each element e_{wi} is obtained as follows:

$$e_{wi} = \mathrm{Prob}(\tau^i | w), \tag{20}$$

[††]Although only 38 kinds of POSs appear in the corpus, by taking into account the generalization ability, the input format of our system was designed to cope with all 47 POSs (refer to Section 4.3.1).

[†]The two values (481 and 300) may be determined by considering the trade-off in speed of convergence and number of modules (the number of parameters). If convergence speed is needed, these values should be decreased, or vice-versa.

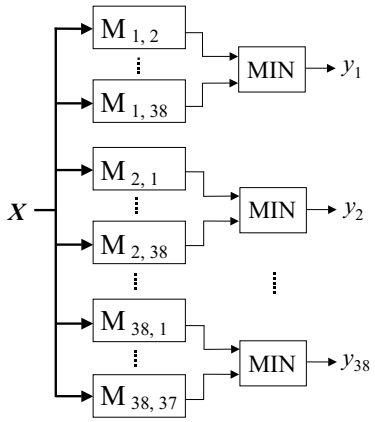

Fig. 1 Composition of the M^3 Network

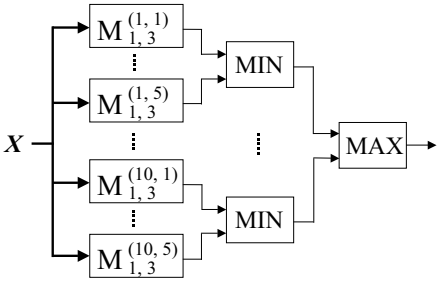

Fig. 2 Composition of module $M_{1,3}$

where $\mathrm{Prob}(\tau^i|w)$ is a prior probability of τ^i that the word w can take. It is estimated from the learning data:

$$\mathrm{Prob}(\tau^i|w) = \frac{|\tau^i, w|}{|w|}, \tag{21}$$

where $|\tau^i, w|$ is the number of times both τ^i and w appear, and $|(w)|$ is the number of times w appears in the learning data. If the word w does not appear in the learning data, each element e_{wi} is given by:

$$e_{wi} = \begin{cases} \frac{1}{\gamma_w} & \text{if } \tau^i \text{ is a candidate} \\ 0 & \text{otherwise,} \end{cases} \tag{22}$$

where γ_w is the number of kinds of POSs that the word w can take[†].

The output vector Y is defined as

$$Y = (y_1, y_2, \cdots, y_{38}), \tag{23}$$

provided that Y is decoded as

$$\tau(w_t) = \begin{cases} \tau^i & \text{if } y_i \geqq 0.5 \ \& \ y_j < 0.5 \text{ for } j \neq i \\ Unknown & \text{otherwise,} \end{cases} \tag{24}$$

where $\tau(w_t)$ is the result of tagging of the word w_t.

In the systems that have been proposed so far [16], an elasticity has been introduced to the length of inputs (l, r) to boost the accuracy of tagging. However, since these systems use gradual learning "from small networks to large networks", which fits the elasticity, the learning time of the first network used as the kernel is quite long. We do not introduce such an elasticity into M^3 network to make the learning faster. In addition, during tagging a sentence from left to right, the words on the left side of the target word have already been tagged. In the systems proposed so far, the tagging results for these preceding words are dynamically used to construct inputs. For simplicity, we do not distinguish between left and right words, and we use the method described above for constructing inputs. Moreover, in the systems proposed so far, when constructing an input x_p [Eq. (19)], the information gain (IG) computed from the learning data is used as weights. However, in our system, we do not use IG because the exact IG for each module can not be obtained for the split data due to the small number of learning data in this case.

5. Experimental Results

The data described in section 4.2 was used. The length (l, r) of the word sequence given to M^3 network was (3,3). Corresponding to this length, the number of units on the input layer is $(l+1+r) \times \gamma = 7 \times 47 = 329$, where γ denotes the number of kinds of POSs. Each module of the M^3 network is basically composed of a three-layer perceptron with $266 - 2 - 1$ units in the input, middle, and output layers, respectively. The learning for all modules is iterated through all the learning data until the error $E_{ACT}(k)$ is smaller than the objective error;

$$E_{ACT}(k) = \frac{\sum_{b=1}^{P} |D^{(b)} - O^{(b)}(k)|}{P}, \tag{25}$$

where k is the number of iterations, $D^{(b)}$ is the desired output for the b-th learning data, $O^{(b)}(k)$ is the actual output for the b-th leaning data, and P is the total number of learning data. If the error is still larger than the objective error after 2000 iterations, we regard the module as non-convergent. In this case, we perform the learning again by increasing the number of units in the middle layer to 4 or 6[††].

The previous system used for comparative experiments is composed of a three-layer perceptron with $[47 \times (l + 1 + r)]$ units in the input layer, $\frac{[47\times(l+1+r)]}{2}$

[†]The learning and testing corpora used in the experiments of the paper are those that have been manually tagged beforehand, so they have no the so-called unknown words (the words that are not in the dictionary). If we set each element e_{wi} as $\frac{1}{\gamma}$, however, our system then can also handle the case when word w is unknown.

[††]When the M^3 network is used, there is no problem of convergence in the strict sense, because complicated problems can be split into simpler problems. The convergence problem arises here because we restricted the learning times and the number of units in the middle layer.

Table 3 Tagging accuracy of M^3 Networks*

Case	Objective error	Number of units in middle layer	number of converged modules	number of unconverged modules	Accuracy (%)	
					learning	testing
1	0.005	2	3,841	52	98.4	92.6
2	0.005	2, 4	3,882	11	98.5	93.0
3	0.005	2, 4, 6	3,890	3	99.5	93.4
4	0.002	2	3,738	155	98.7	93.0
5	0.002	2, 4	3,797	96	99.3	93.1
6	0.002	2, 4, 6	3,852	41	99.8	94.2

*Accuracy was determined only for ambiguous words.

units in the middle layer, and 38 units in the output layer. The length (l, r) of a word sequence given is elastic. In the learning phase, (l, r) increases stepwise as (1,1) → (2,1) → (2,2) → (3,2) → (3,3), that is, the learning proceeds gradually from small networks to large networks. In the tagging phase, (l, r) decreases as (3,3) → (3,2) → (2,2) → (2,1) → (1,1) → (1,0) → (0,0) in compliance with necessity, provided that the number of units in middle layer is the maximum number (i.e., the one corresponding to $(l, r) = (3, 3)$). A three-layer perceptron using the maximum input length (l, r) [i.e., (3,3)] was also used for comparative experiments.

In Table 3, the POS tagging accuracies of the M^3 network are shown for six different cases. The accuracy in is measured only for the words that have POS ambiguity. In case 1, the objective error is 0.005 and the number of units in the middle layer is 2 (default). In this case, 3,841 modules converged while the remaining 52 modules did not converge. In tagging, the accuracy for the learning data (i.e., the learning accuracy) was 98.4% and the accuracy for the testing data (i.e., the tagging accuracy) was 92.6%. In case 2, for the modules that didn't converge after the learning of case 1, the number of units in the middle layer was increased to 4 and the learning was performed again. In this case, only 11 modules did not converge, and the learning accuracy was 98.5% and tagging accuracy was 93.0%. In case 3, similarly, for the modules that didn't converge after the learning of case 2, the number of units in the middle layer was increased to 6 and the learning was performed for a third time. In cases 4 to 6, the objective error was decreased to 0.002 and the learning processes of cases 1 to 3 were performed.

The results show that the M^3 network can boost learning accuracy in compliance with necessity and that tagging accuracy increases in proportion to learning accuracy. That is, the problem of overlearning did not occur in this experiment. The reason why the M^3 network is free from overlearning is thought to be that the number of units in the middle layer is quite small. On the other hand, as for the three-layer perceptron, tagging accuracy decreased as learning accuracy increased, as shown in Table 4. That is, the three-layer perceptron suffered from overlearning†.

Table 4 Tagging accuracy of three-layer perceptron*

Case	Objective error	Accuracy (%)	
		learning	testing
1	0.005	95.8	93.0
2	0.004	97.0	92.8
3	0.003	97.4	92.6

*Accuracy was determined only for ambiguous words.

Table 5 Tagging accuracy of various methods*

Method	**Accuracy(%)**
Baseline model	83.6
Hidden Markov Model	89.1
Three-layer perceptron	93.0
Elastic perceptron	94.4
M^3 network	94.2

*Accuracy was determined only for ambiguous words.

Table 5 lists the tagging accuracies of various methods. The baseline model uses only the POS frequencies for each word computed from the learning data and does not use contexts. The highest accuracies in Tables 3 and 4 are shown for the three-layer perceptron and the M^3 network. The methods using neural networks are generally better than the statistical methods, and the M^3 network's accuracy is more than 1% higher than that of the three-layer perceptron and is almost the same as the elastic perceptron.

The most important characteristic of the M^3 network is that a large, complicated problem can be split into smaller, simpler problems and these smaller problems can be learned with independent modules. That is, these modules can be learned in parallel. If the parallel computation is used, the computational time of M^3 network is dominated by the learning time of the slowest module. Ignoring the speed of the computer used in the experiment, the learning time depends on three elements: data size, the number of learning iterations, and the number of parameters (weights between neurons) to be estimated. In Table 6, these three elements are shown for the three-

†Here, the three-layer perceptron with fixed length $[(l, r) = (3, 3)]$ of input was used. The elastic perceptron was not used in comparative experiments, because it's kernel network could not converge in gradual learning when the objective error was decreased to 0.004.

Table 6 Computational complexity of different neural-network based methods

Methods	Data size	Number of learning iterations	Number of parameters
Three-layer perceptron	22,311	79	60,371
Elastic perceptron	22,311	1,055/30/24/10/14	12,620/21,244/32,078/45,120/60,371
M^3 network	957/699/699	2,000	660/1,320/1,980

layer perceptron, the elastic perceptron, and the M^3 network. For the perceptrons, the data size is the number of all learning data. On the other hand, for the M^3 network, the data size is the number of data in the largest subproblem, because the problem is learned by spliting. The learning of the M^3 network consists of three rounds, as shown in case 6 of Table 3; therefore, the largest data size of each round is shown in the table. The learning method of the elastic perceptron is a gradual learning method consisting of five steps (from smaller networks to larger networks). Therefore, the number of iterations and the corresponding parameters are shown. If we calculate the computational complexities by using "data size×number of iteration×number of parameters", then those of the three-layer perceptron, the elastic perceptron, and the M^3 network are 1.06×10^{11}, 3.57×10^{11}, and 5.88×10^{9}, respectively. That is, by using parallel computing the M^3 network should be more than sixty times faster than the other methods. We should note that although the number of units in the output layer of the three-layer perceptron and the elastic perceptron used in the experiment was 47, we used 38 as the number of units when computing computational complexity, the same as in the M^3 network, for a fair comparison. Furthermore, all three elements used for calculating the computational complexity of the M^3 network are the largest cases and the actual computational complexity therefore will be smaller because the three elements are not always present in one learning case.

Even if we do not use parallel computing, the computational time of the M^3 network is comparable to those of the previous methods. Figure 3 shows the distribution of learning iterations when 3,893 split subproblems are learned to reach an error below 0.002 (the Case 4 in Table 3). The distribution in this figure shows that most modules converge within 500 iterations and that only 4% of all the modules do not converge after learning. The computational complexities of cases 5 and 6 are therefore so small compared with case 4 that they are ignorable. Taking these facts into consideration, we can approximate the average number of learning iterations as 250, the number of parameters as 660, and the average number of learning data as the mean of the maximum and minimum values: $\frac{957+1}{2} = 479$ (refer to section 4.2). Thus, we can obtain the approximate total computational complexity: the number of modules × the number of parameters × the average number of data × the average number of iterations= $3,893 \times 660 \times 479 \times 250 = 3.08 \times 10^{11}$. This value is smaller than that of the previous methods, and even if the ignored computational complexities of cases 5 and 6 are added, the total complexity is comparable to those of the previous methods.

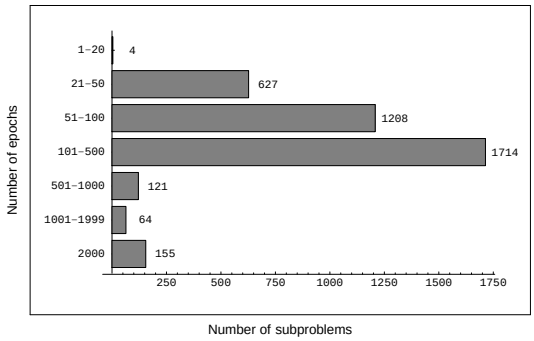

Fig. 3 Distribution of number of iterations

6. Conclusion

In this paper, we described a new POS tagging system using a modular neural network that splits a given large, complicated problem into many smaller, simple problems. This system yields almost the same tagging accuracy as the previous methods, but is 60 times faster. It does not have convergence problems and its learning accuracy can be boosted in compliance with necessity, because each module deals with only a small and simple two-class classification problem. Moreover, the number of units in the middle layer can be set to a very small value, so that overlearning does not occur. Therefore, tagging accuracy can be boosted by increasing the learning accuracy or by increasing the learning data size. In the future, we plan to apply this method to a Chinese corpus, which includes 63 kinds of POSs and more than 100 thousand words with ambiguous POSs.

References

[1] R. Garside, G. Leech, and G. Sampson, The Computational Analysis of English: A Corpus-Based Approach, London: Longman, 1987.

[2] D. Hindle, "Acquiring disambiguation rules from text," Proc. ACL'89, Vancouver BC, pp. 118-125, 1989.

[3] E. Brill, "Transformation-based error-driven learning and natural language processing: a case study in part-of-speech tagging," Computational Linguistics, Vol. 21, No. 4, pp. 543-565, 1994.

[4] K. Church, "A stochastic parts program and noun phrase parser for unrestricted text," Proc. 2nd ACL Applied NLP, Austin, Texas, pp. 136-143, 1988.

[5] S. DeRose, "Grammatical category disambiguation by statistical optimization," Computational Linguistics, Vol. 14, No. 1, pp. 31-39, 1988.

[6] D. Cutting, J. Kupiec, J. Pederson, and P. Sibun, "A practical part of speech tagger," Proc. 3rd ACL Applied NLP, Trento, Italy, pp. 133-140, 1992.

[7] E. Charnik, c. Hendrickson, N. Jacobson, and M. Perkowitz, "Equations for part-of-speech tagging," Proc. 11th National Conference on Artificial Intelligence, AAAI Press/MIT Press, Menlo Park, pp. 784-789, 1993.

[8] E. Charniak, Statistical Language Learning, The MIT Press, 1993.

[9] R. Weischedel, et al., "Coping with ambiguity and unknown words through probabilistic models," Computational Linguistics, Vol. 19, No. 2, pp. 359-382, 1993.

[10] B. Merialdo, "Tagging English text with a probabilistic model," Computational Linguistics, Vol. 20, No. 2, pp.155-171, 1994.

[11] H. Schütze and Y. Singer, "Part-of-speech tagging using a variable memory markov model," Proc. ACL'94, Las Cruces, New Mexico, pp. 181-187, 1994.

[12] M. Nakamura, K. Maruyama, T. Kawabata, and K. Shikano, "Neural network approach to word category prediction for English texts," Proc. COLING'90, Helsinki University, pp. 213-218, 1990.

[13] H. Schmid, "Part-of-speech tagging with neural networks," Proc. COLING'94, Kyoto, pp. 172-176, 1994.

[14] Q. Ma and H. Isahara, "A multi-neuro tagger using variable lengths of contexts," Proc. COLING-ACL'98, Montreal, pp. 802-806, 1998.

[15] Q. Ma and H. Isahara, "A Multi-Neuro Tagger Using Variable Lengths of Contexts," Journal of Natural Language Processing, Vol. 6, No. 1, pp. 29-42, 1999 (in Japanese).

[16] Q. Ma, K. Uchimoto, M. Murata, H. Isahara: "Automatic Part-Of-Speech Tagging System," Journal of Japanese Society of Artificial Intelligence, Vol. 14, No. 6, pp. 1116-1124, 1999 (in Japanese).

[17] Q. Ma, K. Uchimoto, M. Murata, and H. Isahara, "Hybrid neuro and rule-based part of speech taggers," Proc. COLING'2000, Saarbrücken, pp. 509-515, 2000.

[18] B. L. Lu and M. Ito, "Task decomposition based on class relations: a modular neural network architecture for pattern classification,", Biological and Artificial Computation: From Neuroscience to Technology, Lecture Notes in Computer Science, J. Mira, R. Moreno-Diaz and J. Cabestany, Eds., vol. 1240, Springer, pp. 330-339, 1997.

[19] B. L. Lu and M. Ito, "Task decomposition and module combination based on class relations: a modular neural network for pattern classification," IEEE Trans. Neural Networks, Vol. 10, No. 5, pp. 1244-1256, 1999.

[20] B. L. Lu and M. Ichikawa, "Emergnec of learning: an approach to coping with NP-complete problems in learning," Proc. of IJCNN'2000, Como, Italy, Vol. IV, pp. 159-164, 2000.

[21] V. Sornlertlamvanich, N. Takahashi, and H. Isahara, "Building a Thai Part-Of-Speech Tagged Corpus (ORCHID)," The Journal of the Acoustical Society of Japan (E), Vol.20, No.3, pp 189-198, 1999.

[22] J. Kupiec, "Robust part-of-speech tagging using a hidden Makov model," Computer Speech and Language, Vol. 6, pp. 225-242, 1992

[23] M. Murata, M. Uchiyama, K. Uchimoto, Q. Ma, and H. Isahara, "Corpus Error Detection and Correction Using the Decision-List and Example-Based Methods," Information Processing Society of Japan, WGNL 2000-NL-136, pp. 49-56C2000 (in Japanese).

[24] B. L. Lu, Q. Ma, M. Ichikawa, and H. Isahara, "Massively parallel learning of part-of-speech disambiguation," RIKEN Review, no. 30, pp. 40-49, 2000.

[25] T. Charoenporn, V. Sornlertlamvanich, and H. Isahara, "Building a large Thai text corpus - part of speech tagged corpus: ORCHID," Proc. NLPRS'97, Phuket, Thailand, pp. 509-512, 1997.

[26] W. Daelemans and A. Van den Bosch, "Generalisation performance of backpropagation learning on a syllabification task," In M. Drossaers & A. Nijholt (Eds.), TWLT3: Connectionism and Natural Language Processing. Enschede: Twente University, pp. 27-38, 1992.

[27] J. Quinlan, C4.5: Programs for Machine Learning, San Mateo, CA: Morgan Kaufmann, 1993.

Qing Ma received his B.S. degree in electrical engineering from Beijing University of Aeronautics and Astronautics, China, in 1983, and the M.S. and Dr. Eng. degree in computer science from University of Tsukuba, Japan, in 1987 and 1990 respectively. He worked in Ono Sokki Co., Ltd., Japan from 1990 to 1993. In 1993, he joined the Communications Research Laboratory, Ministry of Posts and Telecommunications, Japan. He is now working at the Communications Research Laboratory, Independent Administrative Institution, Japan, as a Senior Researcher. His research interests include neural networks, knowledge representation, and natural language processing.

Dr. Ma is a member of the Japanese Neural Network Society, the Association of Natural Language Processing, and the Institute of Electronics, Information and Communication Engineers of Japan.

Bao-Liang Lu received his B.S. degree in instrument and control engineering from Qingdao Institute of Chemical Technology, Qingdao, China, in 1982, the M. S. degree in computer science and engineering from Northwestern Polytechnical University, Xi'an, China, in 1989, and the Ph. D degree in electrical engineering from Kyoto University, Kyoto, Japan, in 1994.

From 1982 to 1986, he was with the Qingdao Institute of Chemical Technology. From April 1994 to March 1999, he was a Frontier Researcher at the Bio-Mimetic Control Research Center, the Institute of Physical and Chemical Research (RIKEN), Japan. Currently, he is a Research Scientist at the Brain Science Institute, RIKEN. His research interests include brain-like computers, neural computation, machine learning theory, pattern recognition, and natural language processing.

Dr. Lu is a member of the Japanese Neural Network Society, the Institute of Electronics, Information and Communication Engineers of Japan, and a senior member of the IEEE.

Hitoshi Isahara received his B.E., M.E. degrees in electrical engineering from Kyoto University, Japan, in 1978 and 1980 respectively, and Dr. Eng. degree in electrical engineering from Kyoto University, Japan, in 1995. He worked in Electrotecnical Laboratory, Ministry of International Trade and Industry, Japan, from 1980 to 1995. In 1995, he moved to the Communications Research Laboratory, Ministry of Posts and Telecommunications, Japan, where he was the Chief of the Intelligent Processing Section. He is now working at the Communications Research Laboratory, Independent Administrative Institution, Japan, as the leader of Computational Linguistics Group. His research interests include natural language processing, lexical semantics, and machine translation.

Dr. Isahara is a member of the Association for Natural Language Processing, Information Processing Society of Japan, Japanese Society for Artificial Intelligence, and Japanese Cognitive Science Society.

Michinori Ichikawa received his Doctor of Engineering degree in applied physics from University of Tsukuba, Japan, in 1986.

From 1986 to 1997, he was with the Electrotechnical Laboratory, where he worked on the mechanism of membrane excitation, optical recording methods for neural activities, and brain-like computers. In 1997, he joined the Brain Science Institute, the Institute of Physical and Chemical Research (RIKEN), as the team leader of the Laboratory for Brain-Operative Device. His research interests include neurophysiology, brain-like computers, robots, and neural computation.

Dr. Ichikawa is a member of the Japanese Neural Network Society, the Society for Neuroscience, and the IEEE.