On-Line Error Detection of Annotated Corpus Using Modular Neural Networks

Qing Ma[1], Bao-Liang Lu[2], Masaki Murata[1], Michnori Ichikawa[2], and Hitoshi Isahara[1]

[1] Communications Research Laboratory, Kyoto 619-0289, Japan
[2] RIKEN Brain Science Institute, Wako 351-0198, Japan

Abstract. This paper proposes an on-line error detecting method for a manually annotated corpus using min-max modular (M^3) neural networks. The basic idea of the method is to use guaranteed convergence of the M^3 network to detect errors in learning data. To confirm the effectiveness of the method, a preliminary computer experiment was performed on a small Japanese corpus containing 217 sentences. The results show that the method can not only detect errors within a corpus, but may also discover some kinds of knowledge or rules useful for natural language processing.

1 Introduction

To enable machines to deal with considerably varied natural language texts, it is next to impossible to pre-code all knowledge necessary for them. One solution to this problem is to compile the knowledge needed by the system directly from corpora: very large databases for natural language texts to which several kinds of tags, such as part of speech (POS) and syntactic dependency, have been added instead of one that consists only of plain texts. Corpora have been used successfully in constructing various fundamental natural language processing systems including a morphological analyzer and parser, which can be widely applied in many areas of information processing including preprocessing for speech synthesis, post-processing for OCR and speech recognition, machine translation, and information retrieval and text summarization. Manually annotating very large corpora (the Penn Treebank, e.g., consists of over 4.5 million words of American English with 135 POSs.), however, is a very complex and costly endeavor.

For this purpose, many automatic POS tagging systems using various machine learning techniques have been proposed(e.g., [1,2]). In our previous work, we have developed a neuro and rule-based hybrid tagger, which reached a practical level in terms of tagging accuracy that requires less training data compared to the other methods [3]. To further improve our systems' tagging accuracy, we can use two approaches; one is to increase the amount of training data and the other is to improve the quality of the corpus to be used for training. However, since in the first approach multilayer perceptrons were used in our tagger, it will suffer from an unconvergent problem. To overcome this inherent drawback, we have adopted a min-max modular (M^3) neural network [4] that can solve large and complex problems by decomposing them into many small and simple subproblems [5]. For the second approach, a POS-error-detecting technique is needed, which is the main issue of this paper.

Words are often ambiguous in terms of their POSs, which have to be disambiguated (tagged) using context of the sentence. POS tagging, however, no matter using manual or automatic methods, usually involves errors. The POSs in manually annotated corpora can basically have three kinds of errors: simple-mistake type (e.g., POS "Verb" is inputed as "Varb"), incorrect-knowledge type (e.g., word *fly* is always tagged as "Verb"), and inconsistent-type (e.g., word *like* in sentence "Time flies like an arrow" is correctly tagged as "Preposition", but in the sentence "The one like him is welcome" it is tagged as "Verb"). The simple-mistake type can be easily detected by only referring to an electronic dictionary. The incorrect-knowledge type, however, is hardly possible to detect using automatic methods. If we consider tagging words with correct POSs as classification or input-output mapping problems of mapping words under the context of POSs, then the inconsistent-type errors can be considered as sets of data with the same input but different outputs (classes), which can be dealt with using the statistical methods proposed so far or the neural-network method proposed in this paper. The work that has been done so far to develop a detection technique with statistical approaches [6,7] was for off-line use, that is, the detection must be performed ahead of the learning. Off-line detection, however, is expensive for very large corpora because detection must be performed word by word through the whole corpus with no preprocessing to first focus on a few blocks of words or sentences that are certain to include errors, as can be done by the proposed method.

Since the M^3 network consists of modules dealing with very simple and small subproblems, the modules can be constructed with very simple multilayer perceptrons by only either using very few or no hidden units. This means that such modules will basically not suffer from unconvergent problems. In other words, if a module cannot converge, we may basically consider that the module is trying to learn the data, including the inconsistent-type errors. This type of error within an annotated corpus may therefore be detected on-line, in the sense that the detection is performed while learning, that is, by picking out the uncoverged modules and then determining the inconsistent data from the data sets being learned. Since the unconverged modules compared to the converged ones will be very limited when using a corpus of high quality, and the data set each module learns is very small, this on-line error-detecting method will be extremely cost-effective for very large corpora. By using such an on-line error detection method, the quality of the corpus while it is being learned can be improved immediately by a light manual intervention, and the new data can be immediately used to re-train the unconverged modules.

2 The M^3 Network

This section describes in brief the key question for M^3 Network solving classification problems: i.e., how a large and complex K-class problem is decomposed into many smaller and simpler two-class problems, which are individually solved by using modules independent of each other, and how they are integrated to obtain the final solution (for details see [4]).

2.1 Problem Decomposition

Suppose T is a set of learning data for a K-class problem; i.e.,

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

where $X_l \in R^n$ is an input vector, $Y_l \in R^K$ is a desired output, and L is the number of learning data. In general, any K-class problem can be decomposed into $\binom{K}{2}$ two-class problems:

$$T_{ij} = \{(X_l^{(i)}, 1-\epsilon)\}_{l=1}^{L_i} \cup \{(X_l^{(j)}, \epsilon)\}_{l=1}^{L_j}, \quad i = 1, \cdots, K, \ j = i+1, \cdots, K \tag{2}$$

where ϵ is a small positive number, and $X_l^{(i)}$ and $X_l^{(j)}$ are input vectors belonging to C_i and C_j.

In the $\binom{K}{2}$ two-class problems, those that are still too complex can be further decomposed. First, each larger set of input vectors, e.g., $X_l^{(i)}$ [see Eq. (2)], belonging to each class is divided by a random method into N_i $(1 \le N_i \le L_i)$ subsets χ_{ij}; i.e.,

$$\chi_{ij} = \{X_l^{(ij)}\}_{l=1}^{L_i^{(j)}}, \quad j = 1, \cdots, N_i, \tag{3}$$

where $L_i^{(j)}$ is the number of input vectors in subset χ_{ij}. If by using such subsets, the two-class problem defined by Eq. (2) can be decomposed into $N_i \times N_j$ smaller and simpler problems as follows.

$$T_{ij}^{(u,v)} = \{(X_l^{(iu)}, 1-\epsilon)\}_{l=1}^{L_i^{(u)}} \cup \{(X_l^{(jv)}, \epsilon)\}_{l=1}^{L_j^{(v)}}, u = 1, \cdots, N_i, \ \ v = 1, \cdots, N_j, \tag{4}$$

where $X_l^{(iu)} \in \chi_{iu}$ and $X_l^{(jv)} \in \chi_{jv}$ belong to C_i and C_j, respectively.

Thus, if all the two-class problems defined by Eq. (2) are further decomposed into those defined by Eq. (4), the original K-class problem is decomposed into $\sum_{i=1}^{K} \sum_{j=i+1}^{K} N_i \times N_j$ two-class problems. If the learning data set only includes two different elements, i.e., $L_i^{(u)} = 1$ and $L_j^{(v)} = 1$, then the two-class problem defined by Eq. (4) is obviously a linearly separable problem.

2.2 Module Integration

After learning the decomposed subproblems using the individual modules, they next have to be integrated to give a final solution for the original problem. This section focuses on how the modules are integrated. To read why such an integration can solve problems, see Ref. [4].

For integration, three units called MIN, MAX, and INV are used. Here, denotations M_{ij} and $\mathrm{M}_{ij}^{(u,v)}$ are used to indicate the modules for learning subproblems T_{ij} [Eq. (2)] and $T_{ij}^{(u,v)}$ [Eq. (4)], respectively. In the case that K-class problem

T [Eq. (1)] is solved by decomposing into $\binom{K}{2}$ two-class problems T_{ij} [Eq. (2)], first the following combination is performed with the MIN unit that selects the minimum value from the multiple inputs:

$$\mathrm{MIN}_i = \min(\mathrm{M}_{i1}, \cdots, \mathrm{M}_{ij}, \cdots, \mathrm{M}_{iK}), \quad i = 1, \cdots, K \ (i \neq j) \tag{5}$$

where for convenience, the denotation for the MIN unit is used to express its output and the denotations for the modules are used to express their outputs. Thus, the final solution is obtained from these K outputs of the MIN units:

$$C = \arg\max_i \{\mathrm{MIN}_i\}, \quad i = 1, \cdots, K, \tag{6}$$

where C is the class that the input data belongs to. In the case that two-class problem T_{ij} is further decomposed into $T_{ij}^{(u,v)}$ [Eq. (4)], the module $\mathrm{M}_{ij}^{(u,v)}$ learning $T_{ij}^{(u,v)}$ is combined first with the MIN unit:

$$\mathrm{MIN}_{ij}^{(u)} = \min(\mathrm{M}_{ij}^{(u1)}, \cdots, \mathrm{M}_{ij}^{(uN_j)}), \quad u = 1, \cdots, N_i, \tag{7}$$

and then module M_{ij} is formed by using the MAX unit which selects the maximum value from the multiple inputs:

$$\mathrm{M}_{ij} = \max(\mathrm{MIN}_{ij}^{(1)}, \mathrm{MIN}_{ij}^{(2)}, \cdots, \mathrm{MIN}_{ij}^{(N_i)}). \tag{8}$$

The module M_{ij} formed in such a way is then integrated into Eq. (5). Since the two-class problem T_{ij} is the same as T_{ji}, M_{ji} is constructed by M_{ij} and an INV unit which reverses the input value.

3 Error Detection Using M^3 Network

Since the error detection is performed on-line during the learning of a POS tagging problem, to describe how error detection can be performed we have to first describe what is the POS tagging problem, and how the POS tagging problem is decomposed and learned by M^3 networks.

3.1 POS Tagging Problem

Suppose there is a lexicon $V = \{w^1, w^2, \cdots, w^v\}$, where the POSs that can be served by each word are listed, and there is a set of POSs, $\Gamma = \{\tau^1, \tau^2, \cdots, \tau^\gamma\}$. The POS tagging problem is thus to find a string of POSs $T = \tau_1\tau_2\cdots\tau_s$ ($\tau_i \in \Gamma$, $i = 1, \cdots, s$) by following procedure φ when sentence $W = w_1 w_2 \cdots w_s$ ($w_i \in V$, $i = 1, \cdots, s$) is given.

$$\varphi : W^p \to \tau_p, \tag{9}$$

where p is the position of the word to be tagged in the corpus, and W^p is a word sequence centered on the target word w_p with (l, r) left and right words:

$$W^p = w_{p-l} \cdots w_p \cdots w_{p+r}, \tag{10}$$

where $p - l \geq s_s$, $p + r \leq s_s + s$, s_s is the position of the first word of the sentence. Tagging can thus be regarded as a classification or mapping problem by replacing the POS with class and can therefore be handled by using supervised neural networks trained with an annotated corpus.

3.2 Decomposition of POS Tagging Problem

The Kyoto University Text Corpus [8] is composed of 19,956 Japanese sentences that include a total of 487,691 words with 30,674 distinct ones. More than half the total words are ambiguous in terms of the 175 kinds of POSs used in the corpus. As a preliminary study to see whether the M^3 Network can detect errors in the on-line mode while learning POS tagging problem, this paper selected only 217 Japanese sentences, each of which had at least one error. These sentences include a total of 6,816 words with 2,410 distinct ones and 97 kinds of POS tags. By regarding POS as a class, the POS tagging problem in this case is therefore a 97-class classification problem.

According to Sec. 2.1, this 97-class problem is first uniquely decomposed into $\binom{K}{2} = 4,656$ two-class problems. Since some of these problems are still too large, they are further decomposed in the random method described in Sec. 2.1. As a result, two-class problem $T_{1,2}$, for example, is decomposed into eight subproblems, but problem $T_{5,10}$ is not further decomposed. In such a way, the original 97-class problem has been decomposed into total of 23,231 smaller two-class problems.

3.3 M^3 Network for Learning POS Tagging Problem

The M^3 Network that learns the POS tagging problem described in the previous section is constructed by integrating modules, as shown in Fig. 1(a). The individual module M_{ij} is further constructed as shown in Fig. 1(b) if the corresponding problem T_{ij} is further decomposed. In the example shown in Fig. 1(b), since problem $T_{7,26}$ is further decomposed into $N_7 \times N_{26} = 25 \times 10 = 250$ subproblems, $M_{7,26}$ is constructed by 250 modules $M_{7,26}^{(u,v)}$ ($u = 1, \cdots, 25$, $v = 1, \cdots, 10$). And, M_{ji} ($j > i$) is constructed by M_{ij} and the INV unit.

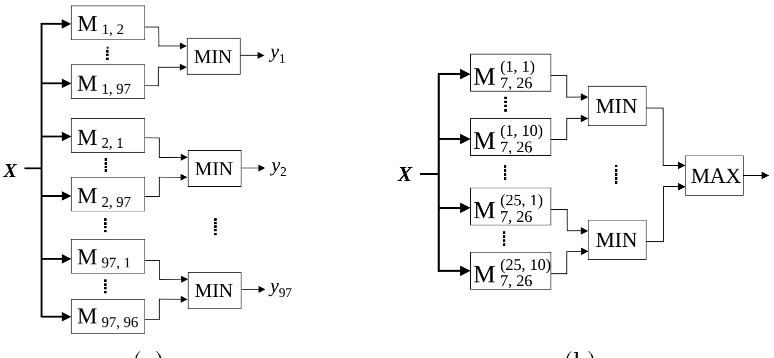

Fig. 1. The M^3 Network: (a) the entire construction and (b) a close-up module $M_{7,26}$.

Input vector X (X_l [Eq. (1)] etc.) in the learning phase is constructed from word sequence W^p [Eq. (10)]:

$$X = (\boldsymbol{x_{p-l}}, \cdots, \boldsymbol{x_p}, \cdots, \boldsymbol{x_{p+r}}). \tag{11}$$

Element $\boldsymbol{x_p}$ is a binary-coded vector with ω dimensions

$$\boldsymbol{x_p} = (e_{w1}, \cdots, e_{w\omega}) \tag{12}$$

for encoding the target word. Element $\boldsymbol{x_t}$ $(t \neq p)$ for each contextual word is a binary-coded vector with τ dimensions

$$\boldsymbol{x_t} = (e_{\tau 1}, \cdots, e_{\tau\tau}) \tag{13}$$

for encoding the POS that has been tagged to the word.[1] The desired output is a binary-coded vector with τ dimensions

$$Y = (y_1, y_2, \cdots, y_\tau) \tag{14}$$

for encoding the POS that the target word should be tagged.

3.4 Detection of POS Errors within Corpus

Since each individual module in the M^3 network only needs to learn a very small and simple two-class problem, it can be constructed, for example, with a very simple multilayer perceptron by only using either none or a few hidden units. Thus, individual module basically does not suffer from an unconvergent problem as long as the learning data is correct. In other words, if a module does not converge, it may be considered that it is learning a data set, $T_M = (X_l, Y_l)_{l=1}^{L_M}$, that includes some inconsistent data: i.e., there is at least one pair of data, (X_i, Y_i) and (X_j, Y_j) in the data set, such that

$$X_i = X_j, Y_i \neq Y_j \quad (i \neq j). \tag{15}$$

Here, T_M is used for denoting T_{ij} [Eq. (2)] or $T_{ij}^{(u,v)}$ [Eq. (4)].

Thus, this type of error within an annotated corpus that is being learned may be detected "on-line" by only picking out the unconverged modules and then determining if the data are in conflict with each other; i.e., a set of pairs, (X_i, Y_i) and (X_j, Y_j), that satisfy Eq. (15) from the data set the module is learning with a simple program. Since the unconverged modules compared to the converged ones will be very limited when an high-quality annotated corpus is used and the data set each module learns is very small, this on-line error-detective method has an extremely good cost performance, which will increase with the size of a corpus. By using such an effective on-line error detection method, the quality of the corpus while it is being learned can be improved by a light manual intervention and the new data can be immediately used to re-train the unconverged modules.

[1] This coding is a simplified version of the original coding method actually used in POS tagging system [5]. However, the detective mechanism that determines the patterns that are in conflict with each other will be unchanged.

4 Experimental Results

The data described in Sec. 3.2 was used in our experiment. Because there are 30,674 distinct words and 175 kinds of POSs in the whole corpus, the dimensions of the binary-coded vectors for word and POS, ω and τ, were set at 16 and 8, respectively. The length (l, r) of a word sequence given to M^3 network was set at (2,2). The number of units in the input layers of all modules was therefore $[(l + r) \times \tau] + [1 \times \omega] = 48$ and all modules were basically constructed using three-layer perceptrons whose input-hidden-output layers had 48-2-1 units, respectively. Modules will stop learning as one round when the average-squared error reaches an objective value, 0.05, or the iterations reached 5,000 epochs. For the modules that could not reach the objective error, relearning of up to five rounds was performed by adding hidden layers of two units each in each new round until reaching the objective error.

Experimental results show that 82 modules within the total of 23,231 modules did not finally converge. Of these 82 modules, 81 had exactly 97 pairs of contradictory learning data, which at first reinforced the hypothesis that the M^3 network has basically no unconvergent problems. These 97 pairs of learning data were checked by one of the authors, an NLP expert who knows both Japanese grammar and the Kyoto University Text Corpus well. As a result, we saw that of these 97 pairs of learning data, 94 pairs included true POS errors and the precision rate reached nearly 97%. Table 1 shows a pair of learning data detected from unconverged module $M_{7,26}^{(1,6)}$, a sub-module of $M_{7,26}$ shown in Fig. 1(b). The left column shows in order of sentence and word number the positions of the words that are being checked. Each word sequence shown in the right column is constructed by morphemes that are separated by the denotation ",". Each morpheme was formed by "Japanese word (English equivalent): POS". The underlined Japanese word is the target word being checked. The word sequence tagged with the symbol "*" in the head shows that the target word in this word sequence is tagged incorrectly.

We examined the remaining three pairs that were in exact conflict with each other, but were all are correct and found that they were all tagging word cases "*de* (in , at, on, ...)" functioning as either postpositional particle or copula in various contexts. The word "*de*" in Japanese, however, belongs to a very special case in which it is not sufficient to only use n-gram word and POS information to determine its POS, the grammar of the whole sentence has to be considered

Table 1. An example of detected pair of learning data

No.	Word sequences tagged by POSs
2-18	* *seijika* (politician): common-noun, *gawa* (side): nominal-suffix, <u>*no*</u> (of): case-postpositional-particle, *odate* (flattery): common-noun, *ni* (by): case-postpositional-particle
124-19	*shinario* (scenario): common-noun, *dukuri* (making) : nominal-suffix, <u>*no*</u> (of): conjunctive-postpositional-particle, *houkou* (direction): common-noun, *o* (object): case-postpositional-particle

instead. This result indicates that this method not only detected POS errors with a substantially 100% precision rate, but also discovered some kind of knowledge that is difficult to be found by us in general but is useful for natural language processing. Since each of the 217 sentences had at least one error, respectively, the recall rate was low. The main reason, however, was in the too small corpus, from which too few clues could be extracted and then used to find further conflicting data. In our future full-scale experiment using the entire corpus, the recall rate of error detection can therefore be expected to rise dramatically with an almost unchanged precision rate.

5 Conclusion

We proposed a cost-effective on-line error-detecting method for manually annotated corpus by conversely utilizing an unconvergent problem, which usually causes us distress when using neural networks, and reinforced the hypothesis that M^3 network basically has no unconvergent problems at the same time. Although the scale of the preliminary experiment was small, since the M^3 network can deal with a very large and complex classification problem, it appears that this technique may be very useful not only in further improving our POS tagging system, but also in improving the quality of various manually annotated corpora, and may discover some kinds of knowledge or rules useful for natural language processing.

References

1. Merialdo, B.: Tagging English text with a probabilistic model, *Computational Linguistics*, Vol. 20, No. 2, pp. 155-171, 1994.
2. Brill, E.: 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.
3. Ma, Q., Uchimoto, K., Murata, M., and Isahara, H: Hybrid neuro and rule-based part of speech taggers, *Proc. COLING'2000*, Saarbrücken, pp. 509-515, 2000.
4. Lu, B. L. and Ito, M.: 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.
5. Lu, B. L., Ma, Q., Isahara, H., and Ichikawa M.: Efficient part-of-speech tagging with a min-max module neural network model, to appear in *Applied Intelligence*, 2001.
6. Eskin, E.: Detecting errors within a corpus using anomaly detection, *Proc. NAACL'2000*, Seattle, pp. 148-153, 2000.
7. Murata, M, Utiyama, M., Uchimoto, K., and Ma, Q., and Isahara, H.: Corpus error detection and correction using the decision-list and example-based methods, *Proc. Information Processing Society of Japan, WGNL 136-7*, pp. 49- 56, 2000 (in Japanese).
8. Kurohashi, S. and Nagao, M: Kyoto University text corpus project, *Proc. 3rd Annual Meeting of the Association for Natural Language Processing*, pp. 115-118, 1997 (in Japanese).