Approximate ILP Rules by Backpropagation Neural Network: A Result on Thai Character Recognition

Boonserm Kijsirikul and Sukree Sinthupinyo

Department of Computer Engineering, Chulalongkorn University,
Phayathai Rd., Phathumwan, Bangkok, 10330, Thailand
Email: Boonserm@cp.eng.chula.ac.th

Abstract. This paper presents an application of Inductive Logic Programming (ILP) and Backpropagation Neural Network (BNN) to the problem of Thai character recognition. In such a learning problem, there exist several different classes of examples; there are 77 different Thai characters. Using examples constructed from character images, ILP learns 77 rules each of which defines each character. However, some unseen character images, especially the noisy images, may not exactly match any learned rule, i.e., they may not be covered by any rule. Therefore, a method for approximating the rule that best matches the unseen data is needed. Here we employ BNN for finding such rules. Experimental results on noisy data show that the accuracy of rules learned by ILP without the help of BNN is comparable to other methods. Furthermore, combining BNN with ILP yields the significant improvement and surpasses the other methods tested in our experiment.

1 Introduction

Inductive Logic Programming (ILP) has been successfully applied to real-world tasks, such as drug design[11], traffic problem detection [4], etc. This paper presents an application of ILP to the task of Thai printed character recognition. Although this task has been widely researched for many years and there are some commercial products of Thai character recognition software available, the accuracies are not yet as high as those of English. This is due to the fact that Thai characters are comparatively more complex and some character is similar to others. Various approaches have been proposed to Thai character recognition such as the method of comparing the head of characters [6], backpropagation neural network [8,15], the method of combining fuzzy logic and syntactic method [13], the method of using cavity features [12], etc.

The reason for choosing ILP in the task of Thai printed character recognition is that ILP is able to employ domain-specific background knowledge that makes a better generalization performance on data. However, some problems arise when ILP is applied to our task where there are several classes of examples, i.e., 77 different Thai characters. Most ILP systems work with two classes of examples (positive and negative), and construct a set of rules for the positive class. Any example not covered by the rules is classified as negative. If we want to employ these systems to learn a multi-class concept, we could do this by first constructing a set of rules for the first class with its examples as positive and the other examples as negative, then

constructing the sets of rules for other classes by the same process. The learned rules are then used to classify future data, and the rule that covers or exactly matches the data can be selected as the output. One major problem of this method is that some test data, especially noisy images in our task, may not be covered by any rule. Thus the method is unable to determine the correct rule. Dzeroski et al. [5] solved this problem by assigning the majority class recorded from training data to the test data that is not exactly matched against any rule.

We approach this problem directly by proposing a method to approximate the rule that provides the best match with the data. Here, we employ BNN for the approximation of ILP rules. First we tried to approximate the rule by using the number of *nonmatching* literals (literals whose truth values are false) and the number of *matching* literals (literals whose truth values are true) as the training input vector to the BNN. We found that the method is able to find the approximate rule that gives a higher accuracy than that of ILP alone. However, our first method did not take into account the significance or the weight of each literal. We then redesigned the structure of BNN to consider the weight of each literal. In our second method, instead of the numbers of nonmatching and matching literals, we use the truth values of all literals in the rules as the input vector, and design a new structure of BNN that can give a weight to each literal separately. Experimental result shows that the recognition accuracy of the new structure is further improved, and is higher than those of the other methods tested in our experiment.

2 Feature Extraction

In the literature of Thai printed character recognition, the method of combining fuzzy logic and syntactic (FLS) reported very high accuracy and was shown to be one of the most successful methods [13]. Therefore, in the following experiment FLS will be used to compare with our methods, and for the comparison purpose the feature extraction in FLS is employed.

After a character image is preprocessed by noise reduction and thinning processing, its features are extracted. Basically, the extracted feature is represented as a *primitive vector list*. A primitive vector list is composed of primitive vectors each of which represents the type of lines or circles in the original image. Therefore, the list represents the structure of lines and circles, and shows how these lines and circles are connected to form the character. Each primitive vector is defined to be one of 13 types shown in Figure 1.

The primitive vectors of type 0 to 7 represent lines and those of type 8 to 12 represent circles. The primitive vector of type 0 is the line whose angle is between 1 to 45 degrees, and the angle of the type 1 is between 46 to 90 degrees, and so on. The primitive vector of type 8 is the circle that does not connect to any line. The primitive vector of type 9 is the circle that connects to a line at the quadrant 1, etc.

In addition to the primitive vectors, other features are also extracted such as the starting and ending zone of a primitive vector, the level, and the ratio of the width and the height of characters, the list of zones containing junctions of lines, the list of zones

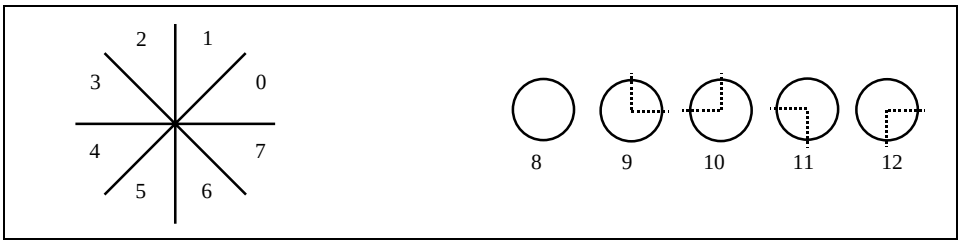

Fig. 1. Primitive Vectors

containing the '∧' angles, and the list of zones containing '∨' angles. The level indicates the character position. Thai characters are written in 3 levels, i.e., level 1, level 2 and level 3. For example, in Figure 2, the word "กุ้ง(shrimp)" consists of four characters in the three levels; ' ้ ',' ุ ', 'ก' and 'ง' are in the level 1, the level 2, the level 3 and the level 3, respectively. The '∧' and '∨' angles are helpful in distinguishing between a pair of similar characters except the angle; one of them contains such an angle and the other does not, such as ('ข','ฃ'), ('ค','ฅ'), etc.

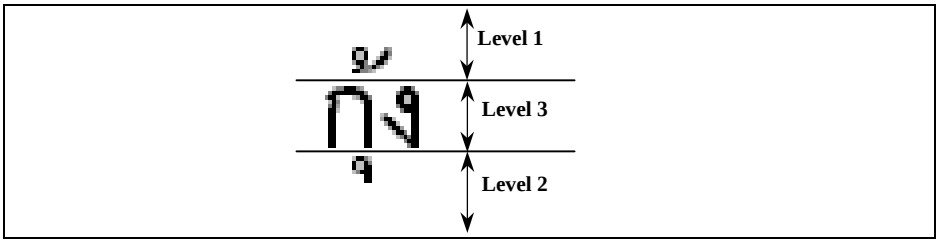

Fig. 2. Three levels of Thai writing system

An instance of the extracted features of the character image 'ก' is shown in Figure 3. This character image contains 8 primitive vectors whose numbers are indicated by 0, 1…7 in the figure. For instance, the vector no. 0 is of type 1, contains the end point (the point not connected to another which is indicated by -1), has the starting zone in zone 3 and the ending zone in zone 2. The image contains '∧' angles in zone 1.

3 Applying Progol to Thai Character Recognition

The ILP system used in our experiment is Progol [10]. The inputs to Progol are positive examples, negative examples and background knowledge. The output is a set of rules defining the positive examples. The following subsection explains the inputs and output from Progol.

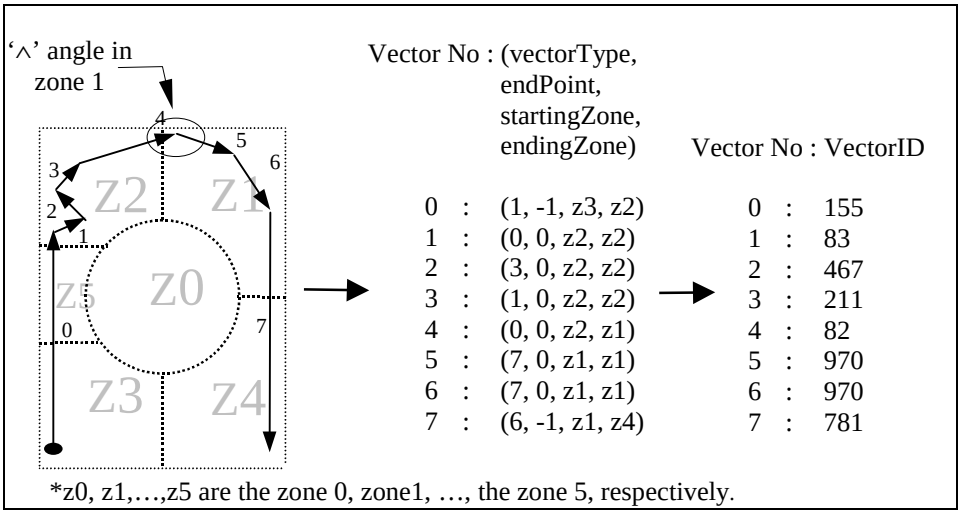

Fig. 3. An instance of features extracted from a character image.

3.1 Example Representation & Background Knowledge

An example to Progol is a set of extracted features explained in the previous section. It is the same as one used by FLS, except that it is converted to the logical representation in the following form:

char(A,B,C,D,E,F)

where *char* is a character, *A* is the level of the character, *B* is the ratio of the width and the height, *C* is the list of numbers each of which represents a primitive vector, whether or not the primitive vector contains the end point, the starting zone and the ending zone of the vector, *D* is the list of zones that contain junctions of lines, *E* is the list of the zones that contain the ‘∨’ angles, and *F* is the list of the zones that contain the ‘∧’ angles. For ease of writing definition of background knowledge, we map a tuple *“(vectorType, endPoint, startingZone, endingZone)”* into a number *“VectorID”*, and used it in *C*. Figure 4 shows some examples of the character ‘ก’.

ก(3,0.78,[155,83,467,211,82,970,970,781],[],[],[z1]).

ก(3,0.76,[155,339,339,211,83,83,978,970,781],[],[],[z2]).

ก(3,0.72,[155,467,467,211,211,83,82,970,842,845,805,19],[z2],[z2],[z1]).

ก(3,0.75,[155,467,211,83,83,978,842,842,714,653,19],[z2],[z2],[z2]).

ก(3,0.75,[155,467,211,83,82,970,845,677,19],[z2],[z2],[z1]).

ก(3,0.78,[155,467,211,83,82,970,970,781,83,979,19],[z2],[z2,z2],[z2,z1]).

Fig. 4. Examples of character ‘ก’

The Thai character set consists of 77 different characters (‘ก’, ‘ข’, ‘ฃ’,…, ‘ฮ’). We choose two types of fonts, i.e., Cordia and Eucrosia, each of which contains 7

different sizes (20,22,24,28,32,36 and 48 points). Therefore, the number of examples is 77 x 2 x 7 = 1078.

We have constructed background knowledge that describes our knowledge about the domain of Thai character recognition. The appropriate background knowledge will help Progol produce more accurate rules. For example, if we believe that the zone of the head of the character can be used to discriminate between the characters, we will add a predicate, such as `head_zone(ListofPrimitiveVector,InZone)` that examines this characteristic. The background knowledge used in our experiment contains 55 predicates. Some of them are shown in Figure 5. Note that some background predicates are complicated and defined by a recursive program. This is one reason of using Progol in our task.

```
head_zone(A,B) :- head(A,C), startZone(C,B).
head_primitive(A,B) :- head(A,C), primitive(C,B).
count_primitive_type4([],0).
count_primitive_type4([A|B],C) :-
    primitive(A,4), count_primitive_type4(B,D), C is D+1.
count_primitive_type4([A|B],C) :-
    not primitive(A,4), count_primitive_type4(B,C).
∨_angle_at_head(A) :- member(z2,A).
endpoint_primitive(A,B) :-
    member(C,A),endpoint(C,-1),  primitive(C,B).
circle_at_endpoint_in_zone(A,B) :-
    member(C,A), isCircleEndpoint(C), startZone(C,B).
count_circle_at_endpoint([],0).
count_circle_at_endpoint([A|B],C) :-
    isCircleEndpoint(A),count_circle_at_endpoint(B,D), C is D+1.
count_circle_at_endpoint([A|B],C) :-
    not isCircleEndpoint(A),count_circle_at_endpoint(B,D).
count_startZone5([],0).
count_startZone5([A|B],C) :-
    startZone(A,5), count_startZone5(B,D), C is D+1.
count_startZone5([A|B],C) :-
    not startZone(A,5), count_startZone5(B,C).
right_line(A) :-
    member(B,A), endPoint(B,-1),endZone(B,1), primitive(B,1).
right_line(A) :-
    member(B,A), endPoint(B,-1),endZone(B,1), primitive(B,2).
member_zone(A,B,C) :- member(E,A), startZone(E,B), endZone(E,C).
head_primitive_type9or10(A) :- head_primitive(A,9).
head_primitive_type9or10(A) :- head_primitive(A,10).
head_primitive_type10or11(A) :- head_primitive(A,10).
head_primitive_type10or11(A) :- head_primitive(A,11).
```

Fig. 5. Some of background knowledge used in the experiment.

3.2 The Output

We first train Progol to induce the rule for character 'ก' by 14 examples of 'ก' as positive examples and the rest (1,064 examples) as negative examples. The training process is then repeated for constructing the rules of all characters. The output of Progol is a set of 77 rules that are used to classify the future character images. Each rule defines the characteristics of each character. Note that we did not force Progol to learn one rule per class, but the output having one rule per class is just that obtained by Progol.

Figure 6 shows some of the learned rules. For instance, the first rule in Figure 6 defines the character 'ก'; an input character image will be recognized as the character 'ก' if that character is in level 3, has the head in zone 3, the head of character is the primitive vector of type 1, and the number of primitive vector of type 4 is equal to 0. These rules will be used to compare with the features of a future character image. The rule that exactly matches the image is selected as the output. However, in the case of noisy image or unseen data, a character image may not exactly match any rule in the rule set. In the next section, we will describe the methods of using BNN to approximately match the rule with the character image.

```
ก(3,A,B,C,D,E)    :-  head_zone(B,3), head_primitive(B,1),
                      count_primitive_type4(B,0).
ข(3,A,B,C,D,E)    :-  not ∨_angle_at_head(D), head_zone(B,2),
                      endpoint_primitive(B,1),
                      circle_at_endpoint_in_zone(B,2),
                      count_circle_at_endpoint(B,1),
                      right_line(B), member_zone(B,4,1),
                      head_primitive_type9or10(B),
                      count_startZone5(B,0).
ฃ(3,A,B,C,D,E)    :-  head_zone(B,2), head_primitive(B,10),
                      endpoint_zone(B,4),
                      begin_and_endzone(B,2,1),
                      member_zone(B,2,1), member_zone(B,4,1),
                      have_member(B,7,2,2),
                      head_primitive_type9or10(B),
                      head_primitive_type10or11(B).
ค(3,A,B,[z3],C,D) :-  not ∧_angle([z3]), not ∨_angle(C),
                      head_zone(B,0), endpoint_primitive(B,6).
```

Fig. 6. Some Rules learned by Progol.

4 Approximate ILP rules by BNN

Backpropagation neural network (BNN) [14] is widely applied to various recognition tasks. In this paper, BNNs are employed to select the rule that closely matches with the input image. The following subsections explain two types of BNNs, i.e., (1) BNN that uses *the number of nonmatching literals and the number of matching literals* of the rules as the input vector, and (2) BNN that uses *the truth values of literals* of the rules as the input vector.

4.1 Using Numbers of Nonmatching and Matching Literals (1st BNN)

There can be many ways to approximate the rule that best matches the data. A simple choice is based on the assumption that the best rule should be the rule that contains a small number of nonmatching literals but a large number of matching literals. We then first build a simple BNN to test this idea.

The structure of the first BNN is composed of three layers: (1) the input layer consisting of 154 neurons, (2) the hidden layer consisting of 154 neurons, and (3) the output neurons having 77 neurons. All neurons receive real values. The hidden and output neurons use the sigmoid function. The links from the input neurons to the hidden neurons, and from the hidden neurons to the output neurons are fully connected, as shown in Figure 7.

As described above, the training set in our experiment consists of 77 different characters (‘ก’, ‘ข’, ‘ฃ’, ‘ค’,…,‘ฮ’). Each character has the corresponding rule produced by Progol. In the training process of BNN, for each training example, we evaluate the number of nonmatching and number of matching literals by comparing the example with all rules in the rule set. Therefore, 154 numbers from all 77 rules in the rule set are used as the training input vector for BNN. The hidden layer consists of 154 neurons, simply the same as the number of the input neurons. The output layer is composed of 77 neurons, each of which represents a character. When the BNN is trained by an example of the character ‘ก’, the first output neuron that corresponds to the character is activated to 1 and the other neurons are set to 0. In the recognition process, the neuron with the maximum activation will be turned on.

Figure 7 illustrates the input and output vectors of an example of character ‘ก’, First, the numbers of nonmatching and matching literals of the rule of character ‘ก’ are counted, that in our experiment are 0 and 4, respectively. Then the example is examined with the rule of the character ‘ข’, and numbers of nonmatching and matching literals are 5 and 5. After the example is checked with all 77 rules, the numbers of nonmatching and matching literals are then fed into the neuron network as an input vector of the character ‘ก’. In the training mode, the first neuron in the output layer that corresponds to character ‘ก’ is set to 1. The training process is then repeated for the other examples of character ‘ก’ as well as the examples of the other characters.

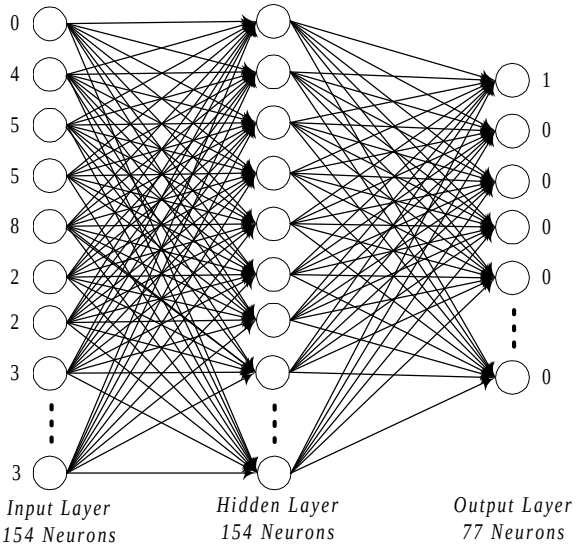

Fig. 7. The structure of the first BNN and the input & output vectors of character 'ก'

4.2 Using Truth Values of Literals (2nd BNN)

The disadvantage of the first BNN is that it takes each literal in a rule with equal significance. Thus it is unable to give important literals higher weights than others. We argue that literals in a rule have different levels of significance in classifying the characters. To capture this idea, instead of the numbers of nonmatching and matching literals, we design the structure of our second BNN by using the truth values of all literals for constructing the input vector. In the rules obtained in our experiment, no literal introduces a new variable which makes it easy to determine its truth value (true or false).

In this type of BNN, the value of matching literal is set to 1 (true) whereas the value of nonmatching literal is set to 0 (false) for an input neuron. Each hidden neuron represents each rule in the rule set. All input neurons corresponding to literals in the same rule are linked to one hidden neuron that represents the rule. Therefore the number of the hidden neurons is equal to the number of all rules; in our experiment, the number is 77, since only one rule per class is obtained by Progol. Nevertheless, if some class is defined by more than one rule, the additional hidden neurons together with their corresponding input neurons must be included, and this can be directly handled by the proposed BNN without any change.

Figure 8 shows the structure of our second BNN, and the input and output vectors for training the character 'ก'. When training the network with an example of character 'ก', the example is examined with all literals in all 77 rules. First, the truth values of four literals in the rule of the character 'ก' in Figure 6 are evaluated to 1, 1, 1 and 1.

Then the example is examined with the rule of the character 'ข', and the values of the literals are 1, 1, 0, 1, 0, 0, 1, 0, 0 and 1. After the image is checked with all 77 rules, these numbers are then fed into the neural network as the input vector for the character 'ก'. In the training mode, the first neuron in the output layer that corresponds to the character 'ก' is set to 1. This training process is then repeated for the other examples of the character 'ก' as well as those of the other characters. In the recognition mode, the neuron with the highest-valued output will be taken as the prediction.

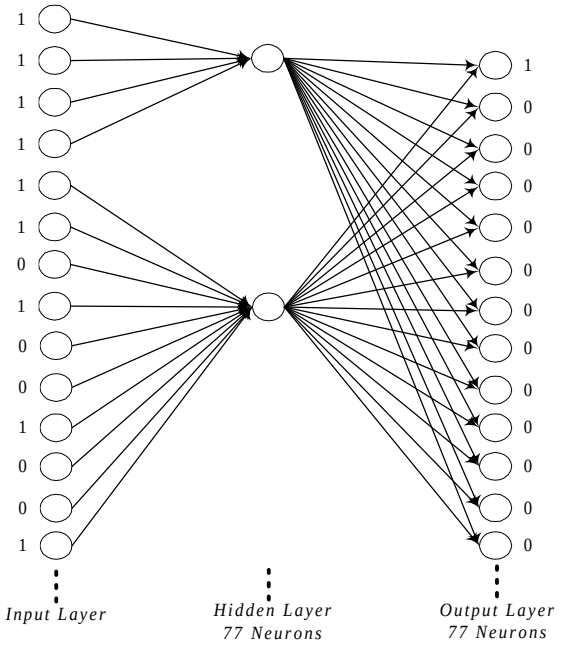

Fig. 8. The structure of the second BNN and the input & output vectors of character 'ก'.

5 Experimental Results

The experiment was run to test our methods. The training character images in our experiment are printed by the laser printer with the resolution 300 dpi and scanned into the computer by the scanner with the same resolution. These images consists of 77 different characters ('ก', 'ข', 'ฃ', 'ค',…,'ฮ'), 2 fonts (Cordia and Eucrosia), and 7 sizes (20, 22, 24, 28, 32, 36 and 48 points). These images are then fed into noise reduction and thinning processing for finding the structure of the images. Next, the

important features are extracted for constructing the training examples. These examples are used to train FLS and our methods. For testing, the noise was added into an original image by copying the image by a photocopy machine twice; darker and lighter ones. The total number of test data was 2x2x7x7=2156. The experimental results on the test data are shown in Table 1. We did not report the accuracies of the methods on training set since they are almost the same and nearly 100%. We also include in Table 1 the result of using BNN alone with the same training examples.

Table 1. Accuracies of FLS, Progol, BNN and Progol & BNN on noisy images.

Font & Size	No. character	FLS	BNN	Progol	Progol& 1st BNN	Progol & 2nd BNN
C20*	154	86.54	80.52	80.52	93.47	94.77
C22	154	88.90	84.21	89.61	95.42	98.04
C24	154	89.20	90.91	87.01	93.54	94.77
C28	154	92.89	79.22	88.31	94.81	96.10
C32	154	88.23	85.71	88.31	95.42	98.04
C36	154	91.58	86.84	89.61	97.37	98.68
C48	154	87.95	72.73	85.06	92.84	93.46
E20*	154	68.19	81.82	68.18	75.99	78.57
E22	154	86.89	82.89	88.31	92.86	94.77
E24	154	87.86	93.51	83.77	90.92	92.86
E28	154	86.89	88.31	83.77	91.56	95.45
E32	154	93.51	92.21	89.61	95.46	95.45
E36	154	92.20	92.11	88.31	95.45	96.05
E48	154	85.90	68.42	79.22	90.76	92.72
Average	2156	87.62	84.25	84.97	92.55	94.26

* C and E are fonts Cordia and Eucrosia, respectively.

The results reveal that the accuracy of rules produced by Progol is comparable to BNN, and is lower than the accuracy of FLS. Since Progol can correctly recognize the input data only when the data is perfectly matched with the correct rule, it is not able to recognize some input data that is matched partly with the correct rule and partly with others. This is the reason for lower accuracy of Progol. On the other hand, FLS always chooses the most similar stored pattern as the matching character, even if there is no stored pattern that exactly matches the input data. After approximate match was performed by using the numbers of nonmatching and matching literals, the accuracy is improved to 92.55%. However, in the case of character that is similar to others, the first BNN may misclassify the character. For instance, in our experiment we found the case that the first BNN misclassified a character image 'ก' as the character image 'ถ'. In this case, the second BNN that uses the truth values of literals correctly classified this character image. As shown in Table 1, the second BNN yields significant improvement by achieving 94.26% accuracy.

6 Related Works and Limitations

There are some ILP systems that are able to learn multi-class concepts represented in first-order rules, such as RTL [1], ICL[7], MULT_ICN[9]. Nevertheless, it is still possible that some unseen data may not exactly match the rules produced by these systems. We believe that our method for approximating rules can also be applied to these systems. TILDE [2] is a first-order extension of decision tree learning algorithm and it will assigns some class to a given unseen data. Therefore, TILDE can be applied to our task as well. Further study is required to compare our method and TILDE that is one of our ongoing work. KBANN [16] and FONN [3] are the systems that employ initial rules and training examples for learning neural networks. These systems showed that the performance is better than methods which learn purely from examples. FONN translates first-order rules possibly including literals with new variables into neural network. Its primary goal is to refine numerical literals of the rules. It first finds only substitutions for new variables that satisfy non-numerical literals and uses these substitutions for refining the numerical literals [3]. One limitation of our method is that the network deals only with rules which introduce no new variable. Though the rules with no new variable are sufficient for our current task, to be applied to other tasks where rules introduce new variables, the method should be further studied. The method like that of FONN may be helpful. In our task, we consider the method of approximation of ILP rules when there is no rule that exactly matches unseen data. Though it did not occur in our current task, a related problem when multiple rules fire will be studied in the near future.

7 Conclusions

We have proposed an improved method that combines ILP and BNN for the task of Thai printed character recognition. The experimental results show that our method gives a significant improvement on previous methods. The improved results come from the combination of ILP and BNN. ILP produces rules that accurately classify the training data, and BNN makes the rule more flexible for approximately matching with unseen or noisy data. Moreover, the results also demonstrate that to approximate the rule, each predicate in the rule should be weighted unequally; this is accomplished by using the truth values of literals as the input vector and separately assigning a weight to each literal.

Acknowledgements. We would like to thank Decha Rattanatarn for providing us the code of FLS, Apinya Supanwansa for helping us construct the background knowledge used in the work, and Apinetr Unakul for comments on the paper.

Reference

[1] C.Baroglio and M.Botta. Multiple Predicate Learning with RTL. *Topic in Artificial Intelligence, LNAI 992,* 1995.

[2] H.Blockeel and L.De Raedt. Experiments with Top-Down Induction of Logical Decision Trees. *Technical Report CW247,* Dept. of Computer Science, K.U.Leuven, 1997.
[3] M.Botta, A.Giordana and R.Piola. FONN: Combining First Order Logic with Connectionist Learning. *Proceedings of the 14th International Conference on Machine Learning*, 1997.
[4] S.Dzeroski, N.Jacobs, M.Molina, C.Moure, S.Muggleton and W.V.Laer. Detecting Traffic Problems with ILP. *Proceedings of the 8th International Workshop on Inductive Logic Programming*, 1998.
[5] S.Dzeroski, S.Schulze-Kremer, K.R.Heidtke, K.Siems and D.Wettschereck. Applying ILP to Diterpene Structure Elucidation from ^{13}C NMR Spectra. *Proceedings of 6th International Workshop on Inductive Logic Programming*, 1996.
[6] S.Eupiboon. Thai Handwritten Character Recognition by Considering the Character Head. *Master Thesis, KMIT*, Thailand, 1988 (in Thai).
[7] W.V.Laer, S.Dzeroski and L.D.Raedt. Multi-Class Problems and Discretization in ICL Extended Abstract. *Proceedings of the Mlnet Familiarization Workshop on Data Mining with Inductive Logic Programming*, pp.53-60, 1996.
[8] C.Lursinsap. Speeding Up Handwritten Character Neural Learning by Surface Projection. *Proceedings of the 2nd Symposium on Natural Language Processing*, pp. 198-205, 1995.
[9] L.Martin and C.Vrain. MULT_ICN: An Empirical Multiple Predicate Learner. *Proceedings of the 5th International Workshop on Inductive Logic Programming*, 1995.
[10] S.Muggleton. Inverse Entailment and PROGOL. *New Generation Computing*, 3:245-286, 1995.
[11] S.Muggleton, D.Page and A.Srinivasan. An Initial Experiment into Stereochemistry-Based Drug Design Using Inductive Logic Programming. *Proceedings of the 6th International Workshop on Inductive Logic Programming*, pp.25-40, 1996.
[12] P.Phokharatkul and C.Kimpan. Recognition of Handprinted Thai Characters Using the Cavity Features of Character Based on Neural Network. *Proceedings of the 1998 IEEE Asia-Pacific Conference on Circuits and Systems*, pp. 149-152. 1998.
[13] D. Rattanatarn and S. Jitapunkul. Thai Printed Characters Recognition Using the Fuzzy Logic Technique and Syntactic Method. *Proceedings of the 18th Conference of Electrical Engineering*, 1995 (in Thai)
[14] D.E.Rumelhart, G.E.Hinton and R.J.Williams. Learning internal representations by error propagation. In D. E. Rumelhart, & J. L. McClelland (Eds.), *Parallel distributed processing (Vol 1)*, Cambridge, MA:MIT Press, 1986.
[15] T.Tanprasert and C.Tanprasert, Variable Simulated Light Sensitive Model for Handwritten Thai Digit Recognition. *Proceedings of the 2nd Symposium on Natural Language Processing*, pp. 173-179, 1995.
[16] G.G.Towell and J.W.Shavlik. Knowledge-Based Artificial Neural Networks. Artificial Intelligence, 70 (4):119-116, 1994.