FUZZY LOGIC AND GENETIC ALGORITHM FOR OPTIMISING THE APPROXIMATE MATCH OF RULES BASED ON BACKPROPAGATION NEURAL NETWORKS

Jun Srisutapan[1,2] and *Boonserm Kijsirikul*[2]

[1]King Mongkut's University of Technology Thonburi, Prachauthit Rd., Thung-Kharu, Bangkok 10140, Thailand
srisutapan_j@thai.com

[2]Chulalongkorn University, Phayathai Rd., Phathumwan, Bangkok, 10330, Thailand
boonserm.k@chula.ac.th

ABSTRACT

This paper presents an application of Fuzzy Logic(FL) and Genetic Algorithm(GA) for improving the approximate match of first-order Inductive Logic Programming(ILP) rules that is based on Backpropagation Neural Networks(BNN). With the help of FL, the evaluation of the truth values of logic programs is more problem-sophisticated, before these values are sent to the BNN for learning or for recognising. We employ GA to find the best fuzzy sets. Experimental results on a Thai-OCR domain show that the our method gives the best recognition accuracy of 85.95% compared to 82.31% recognition accuracy of the previous method.

1. INTRODUCTION

For the last ten years, many techniques have been developed to improve classification accuracy of Thai-Optical Character Recognition (Thai-OCR) which is not reasonably high enough, according to the complexity nature of Thai characters plus the high similarity between some characters, e.g., the characters ฎ, ฏ, ท or ฑ.

Supanwansa has proposed an interesting method that employed Inductive Logic Programming (ILP) which is a kind of Machine Learning technique to generate a rule set for recognising characters[4]. ILP uses application-specific background knowledge, positive instances and negative instances, and outputs the rule set for characterising the concept of the positive instances. However, for some noisy or unseen character instance, there can be no rule that exactly characterises the character. In this case, we say that this character is *ILP-unrecognisable*. Kijsirikul et al. solved this ILP-unrecognisable problem by sending the binary truth values of literals of the rule set to a BNN [2]. After the BNN is applied, every character is recognisable, and thus, the more correctly recognised characters are obtained.

There is some disadvantage we observed in the line of the above work, i.e., the truth values used in this work are evaluated using classical binary logic. These values are sharp possibility values of either 0 or 1. The value 1 is used when a given character exactly has some property; otherwise, the value 0 is used, which is unsuitable. We believe that the more suitable truth values should increase the classification accuracy of the BNN. This paper will examine this hypothesis.

The rest of this paper is organised as follows. Section 2 gives an overview of ILP&BNN methods. Section 3 briefly discusses the concept of FL. Section 4 gives the concept of GA. Section 5 describes the experiments. The conclusion of this paper will be given in Section 6.

2. TRADITIONAL METHOD

This section gives an overview of the traditional method of combining ILP with BNN.

Training the BNN For convenience, suppose that there are only three Thai characters, i.e., 'Kai'('ก'), 'Khai'('ข') and 'Tahan'('ท'). After ILP is applied, we obtain a *rule set* that is used to recognise future characters. Figure 1 shows an example of a rule set generated by ILP.

There are three rules in this rule set, each of which defines the concept of character 'Kai', 'Khai' or 'Tahan', respectively. The predicates `HeadZone(I, 3)` and `HeadPrimitive(I, 1)` are characteristics of character 'Kai'. `HeadZone(I, 3)` characterises that the head zone of character 'Kai' must be of type 3 (at the bottom left). `HeadPrimitive(I, 1)` characterises that the head of character 'Kai' must be of type 1 (in octant 2). Such predicates are called *background knowledge* that must be provided to the ILP. The details of background knowledge are described in [4].

```
Kai(I):-     HeadZone(I, 3),
             HeadPrimitive(I, 1).

Khai(I):-    HeadZone(I, 2),
             HeadPrimitive(I, 10),
             EndPointZone(I, 4),
             CountEndPoints(I, 2).

Tahan(I):-   not TopRightTail(I),
             HeadZone(I, 2),
             HeadPrimitive(I, 12),
             EndPointPrimitive(I, 5),
             CountEndPoints(I, 3).
```

Figure 1: An example of a rule set.

A character X is represented by the information I_X, *e.g.,* I_X = (3, 0.78, [78, 82, 83], [], [], [1]). X will be recognised as 'Kai' if when all occurrences of `I` in the rule set are substituted with I_X, the truth values of all literals in rule 'Kai' are 1. Logically, a given character Y which is represented by I_Y, will not be recognised as 'Khai' if when all occurrences of `I` in the rule set are substituted with I_Y, the truth value of some literal in rule Khai is 0.

In order to train the BNN to learn a character of font 'Kai', e.g., *TrainKai.* Firstly, we substitute all occurrences of `I` in the rule set with $I_{TrainKai}$ and then evaluate the truth value of every literal in the rule set. Suppose such the truth values are shown on the right hand side of each literal in Figure 2 below.

```
HeadZone(I_TrainKai, 3)                  →   1
HeadPrimitive(I_TrainKai, 1)             →   1

HeadZone(I_TrainKai, 2)                  →   0
HeadPrimitive(I_TrainKai, 10)            →   0
EndPointZone(I_TrainKai, 4)              →   1
CountEndPoints(I_TrainKai, 2)            →   0

not TopRightTail(I_TrainKai)             →   1
HeadZone(I_TrainKai, 2)                  →   0
HeadPrimitive(I_TrainKai, 12)            →   0
EndPointPrimitive(I_TrainKai, 5)         →   0
CountEndPoints(I_TrainKai, 3)            →   0
```

Figure 2: The truth values of all literals when $I_{TrainKai}$ is substituted into the rule set.

We send the vector `[1 1 0 0 1 0 1 0 0 0 0]`^{T} for training the BNN for *TrainKai* (Figure 3). We repeat this step for all training characters. The links of the BNN are fully connected from the hidden layer to the output layer. The number of nodes in the hidden layer equals to the number of rules, which is three. All input nodes corresponding to predicates in the same rule are connected to one hidden node that represents that rule.

Recognising characters For a given noisy ILP-unrecognisable character of font 'Khai', e.g., *NoisyKhai* which is represented by $I_{NoisyKhai}$, as shown in Figure 4. Analogous to the previous example, we send the vector `[1 0 0 1 0 0 1 1 0 0 0]`^{T} to the BNN. In this case, the prediction from the BNN (the best matching character to *NoisyKhai*) is 'Khai', which is correct.

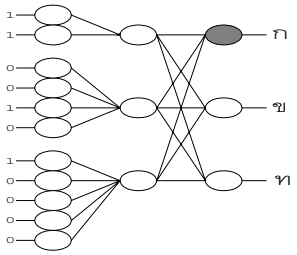

Figure 3: Training the BNN with *TrainKai*.

```
HeadZone(I_NoisyKhai, 3)                 →   1
HeadPrimitive(I_NoisyKhai, 1)            →   0

HeadZone(I_NoisyKhai, 2)                 →   0
HeadPrimitive(I_NoisyKhai, 10)           →   1
EndPointZone(I_NoisyKhai, 4)             →   0
CountEndPoints(I_NoisyKhai, 2)           →   0

not TopRightTail(I_NoisyKhai)            →   1
HeadZone(I_NoisyKhai, 2)                 →   1
HeadPrimitive(I_NoisyKhai, 12)           →   0
EndPointPrimitive(I_NoisyKhai, 5)        →   0
CountEndPoints(I_NoisyKhai, 3)           →   0
```

Figure 4: *NoisyKhai* is not ILP-recognisable.

However, there is some weakpoint of this method as mentioned earlier. Let us consider the predicate `CountEndPoints(I, 3)` which counts the number of the end points in the image of the character. The truth value of this predicate will be set to the value 1 if `I` contains exactly three end points; otherwise, it will be set to the value 0. Suppose `I` contains two end points, so the truth value 0 is assigned, which is not suitable. We believe that the more suitable truth values being sent to the BNN should increase the recognition accuracy. This is the hypothesis of our method. Here we employ FL to find such the values, which will be introduced in the next section.

3. FUZZY LOGIC

Classical logic, which is sometimes not sophisticated to many real-world problems, e.g., the problem of counting something we just mentioned above. *Fuzzy Logic* (FL), is a logic system for reasoning that are approximate rather than exact. The fundamental unit of FL is the *Fuzzy Set* (FS). FS and FL are developed by Zadeh [5, 6].

Fuzzy set Definition *Given the universal set X, in order to define a fuzzy set A on X, we define a membership function A : X → [0, 1] that maps elements x of X into real numbers in [0, 1]. A(x) is interpreted as the*

degree to which x belongs to the fuzzy set A. We sometimes write fuzzy set A as {(x, A(x))| x ∈ X}. □

The following are some fuzzy sets used in our experiments.

(i) Many predicates in our background knowledge are used for counting some objects. We then naturally construct the fuzzy set *near-*x_0 = { (x, *near-*$x_0(x)$ = $\frac{1}{1+k(x-x_0)^2}$) | $x \in \Re$ } to characterises the degree to which an element $x \in \Re$ is belong to the real number x_0. The shape of the fuzzy set is controlled by $k \in \Re^+$. The predicates use this fuzzy set with their own k. Note that when $k \to \infty$, this fuzzy set becomes crisp and this is allowable in our experiment as in some situation, a crisp set may be suffice.

(ii) In our background knowledge, each character image is composed of *primitive vectors*(*or vectors*). There are two kinds of vectors, i.e., line vectors and circle vectors. (see Figure 5).

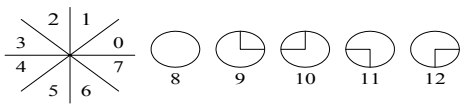

Figure 5: Primitive vectors.

The vectors of type 0 to type 7 are used to represent lines. The vectors of type 8 to type 12 are used to represent circles. The type-0 vectors are the lines whose angles lie in octant 1 and so on. The type-8 vector is the circle that does not connect to any line. The type-9 vector is the circle that connects to a line at quadrant 1 and so on.

Next, the similarity on given two vectors are defined in the fuzzy set *VectorsRelation* = {((*i*, *j*), *VR*(*i*, *j*)) | (*i*, *j*)∈ {0, ..., 12}^{2}}. *VR*(*i*, *j*) equals 0 if *i* and *j* are not the same kind; otherwise, it is defined in Figure 6 (the blank cells indicate the value 0). k_1 and k_2 are parameters which should be appropriately determined by GA.

VR(*i*, *j*)	0	1	2	3	4	5	6	7
0	1	k_1						k_1
1	k_1	1	k_1					
2		k_1	1	k_1				
3			k_1	1	k_1			
4				k_1	1	k_1		
5					k_1	1	k_1	
6						k_1	1	k_1
7	k_1						k_1	1

VR(*i*, *j*)	8	9	10	11	12
8	1	k_2	k_2	k_2	k_2
9	k_2	1	k_2		k_2
10	k_2	k_2	1	k_2	
11	k_2		k_2	1	k_2
12	k_2	k_2		k_2	1

Figure 6: *VectorsRelation*(*i*, *j*).

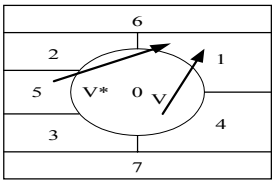

Figure 7: Primitive vector zones.

(iii) *Primitive vector zones* (or *zones*) are used to address the position of vectors. We address the position of a vector using its *starting zone* and its *ending zone* (see Figure 7). There are eight zones, i.e., zone 0, ... , zone 7. Next, the similarity on zones are defined by the fuzzy set *ZoneRelation* = {((*i*, *j*), *ZR*(*i*, *j*)) | (*i*, *j*) ∈ {0, ..., 7}^{2}}. Where *ZR*(*i*, *j*) is defined as in Figure 8 (the blank cells indicate the value 0).

ZR(*i*, *j*)	0	1	2	3	4	5	6	7
0	1	k_3	k_3	k_3	k_3	k_3		
1	k_3	1	k_3		k_3		k_3	
2	k_3	k_3	1			k_3	k_3	
3	k_3			1	k_3	k_3		k_3
4	k_3	k_3		k_3	1			k_3
5	k_3		k_3	k_3		1		
6		k_3	k_3				1	
7				k_3	k_3			1

Figure 8: *ZoneRelation*(*i*, *j*).

Figure 9: Five levels of Thai writing system.

(iv) In Figure 9, Thai characters are written in five levels, that are, level 1 to level 5. For example, the level of each character in the string "ปฏิกูล" (pollution) are 2, 4, 1, 3, 5 and 3, respectively. The reason we need a fuzzy set here is that, there are a lot of Thai characters that look very similar but they are in different levels, e.g., ป(2) and บ(3); ฟ(2) and พ(3). We then define a fuzzy set *LevelRelation* = { ((*i*, *j*), *LR*(*i*, *j*)) | (*i*, *j*) ∈ {1, 2, 3, 4, 5}^{2}} to express the matter of degree these levels are related as in Figure 10.

LR(*i*, *j*)	1	2	3	4	5
1	1				
2		1	k_4		
3		k_4	1	k_4	
4			k_4	1	
5					1

Figure 10: *LevelRelation*(*i*, *j*).

The rest of fuzzy sets in our experiment are used to compare two real numbers wheter they are greater, or less than each other. After all necessary fuzzy sets are defined,

we interpret the meaning of a logic program in the background knowledge into the first-order logic framework, and apply FL to them.

Fuzzy logic In FL, the truth value of a given fuzzy proposition (substituted predicate) can have an intermediate truth-value in [0, 1]. We symbolised the procedure of evaluating the truth value of our background knowledge as in function T described in Figure 11. In Figure 11, "A" is some concept such as high, low. A is the fuzzy set that is used to characterise concept "A". c, i, u and L are any fuzzy complement, fuzzy intersection, fuzzy union and a finite set of objects, respectively. $\exists$ is existentially fuzzy quantifier, $\forall$ is universally fuzzy quantifier. Q is fuzzy number. $A(v)$ is some predicate involving variable v. $A(v \leftarrow x)$ is the proposition which is obtained by substituting all occurrences of variable v with x, in A. Φ and Γ are any fuzzy propositions. This paper employs the standard fuzzy operations for c, i, and u. For more detail about FS, or FL, see [3].

function $T(p$: fuzzy proposition) : [0, 1];
{
  case p **of** {
    "x is A": $T := A(x)$;
    "$\neg\Phi$": $T := c(T(\Phi))$;
    "$\Phi \wedge \Gamma$": $T := i(T(\Phi), T(\Gamma))$;
    "$\Phi \vee \Gamma$": $T := u(T(\Phi), T(\Gamma))$;
    "$\exists v \in L{:}A(v)$": $T := \max_{x \in L}\left[T\left(A(v \leftarrow x)\right)\right]$;
    "$\forall v \in L{:}A(v)$": $T := \min_{x \in L}\left[T\left(A(v \leftarrow x)\right)\right]$;
    "$Qv \in L{:}A(v)$": $T := Q\left(\sum_{x \in L} T\left(A(v \leftarrow x)\right)\right)$;
  }
}

Figure 11: Truth value function.

There are 55 predicates in the background knowledge. They need seven fuzzy sets parametrised by the variables $x[0]$, ..., $x[12]$ to define such k_1, k_2, ... we mentioned earlier. We will refer the tuple ($x[0]$, ..., $x[12]$) as *fuzzy configuration*, *FCFG*. $x[0]$, ..., $x[5] \in [0, 1]$ are used to define matrix fuzzy sets, if $x[i] = 0$, all FSs corresponding to $x[i]$ will become crisp. $x[6]$, ..., $x[12] \in [0, \infty]$ are used to control the shape of the *near*-x_0's, and if $x[i] = 0$, the predicate corresponding to the $x[i]$ will also become crisp. Therefore, the FCFG (0, ..., 0) will cause the entire system become crisp. By some manual coarse searching, we have found that some FCFGs can improve the recognition accuracy of the BNN. In order to optimise this FCFG automatically, we decide to employ GA because the solution FCFG can be represented as a finite-length binary string, and GA can be applied effectively.

4. GENETIC ALGORITHM

Genetic Algorithms (GA) [1] is widely known as a robust probabilistic algorithm used for optimising an encodable solutions. GA locates the optimal solution using a procedure similar to those in natural selection. Each solution is called a *genome*. A set of genomes is called a *population*. GA evaluates each genome to measure its *fitness*. In the initial generation, GA randomly creates a population using a random number seed. In the next generation, GA randomly picks some genomes with probabilities according to their fitness to generate offsprings and modifies them by genetic operators. This procedure is then repeated until the terminating condition is satisfied, such as the maximum number of generation is encountered.

Genetic configuration setting The genetic operators used in this experiment are *cloning*, *two-points crossover* and *mutation*. The cloning is a duplication from parent to children. In the two-point crossover, two genomes are picked as parents, two random points are picked to cut each genome into three pieces, then these pieces are swapped and recombined. The mutation is actually bit flipping. The *fitness* of a genome we used is the linear scaling of the *objective function*. The objective function ϕ measures the recognition accuracy returned from the BNN as below.

function $\phi(g$: genome) : **accuracy in %**;
{
- decode g into FCFG = ($x[0]$, ..., $x[12]$);
- **if** (FCFG is invalid) **then**
  $\phi := 0\%$;

else {
- generate traning/test data set for the BNN using FCFG, run the BNN;
- ϕ := the accuracy returned from the BNN;

}
}

Figure 12: The objective function.

The objective function will reject any invalid genome that produces a FCFG with some $x[i \in \{0, ..., 5\}] > 1$, and then assigns 0% recognition accuracy for such genome. Figure 13 shows such the genome while Figure 14 shows the genetic configuration setting.

```
X[0]=3.28125e-07;          X[0]=0;
X[1]=140;                  X[1]=0;
X[2]=0;                    X[2]=0;
.                          .
.                          .
X[10]=1;                   X[10]=1.5;
X[11]=0;                   X[11]=1;
X[12]=0;                   X[12]=781250;
[ ac = 0 ]                 [ ac =  71.30 ]
```

Figure 13: Invalid(left) and valid(right) FCFG genome with its recognition accuracy.

Number of Genomes	64
Length of the String	156
Replacement Percentage	50%
Crossover Percentage	100%
Crossover Method	Two-point
Mutation Percentage	10%
Max. Number of Generations	256
Random number seed	191

Figure 14: Genetic configuration setting.

Encoding scheme We design each 12-bits floating point $x[i]$ as $mantissa[i] \times 10^{exponent[i]}$, $i \in \{0, ..., 12\}$. Therefore the total length of the encoded string is 13×12 = 156 bits. Figure 15 shows the encoding scheme.

$x[0].b[0]$	...	$x[0].b[11]$	...	...	$x[12].b[11]$

Figure 15: Encoding scheme.

We decode $mantissa[i]$ as $b[6] \times (\{1 + b[0] \times 1 + b[1] \times 2 + ... + b[5] \times 32\} / 64)$ and decode $exponent[i]$ as $b[10] \times (-1)^{b[11]} \times (b[7] \times 1 + ... + b[9] \times 4)$. We normalise $mantissa[i]$ by 64 to range them in [0, 1]. From our experiments, normalised mantissa always gives a better results than unnormalised one. Hence, the set of possible values of each $x[i]$ is $\{0\} \cup \{1.56 \times 10^{-9}, ..., 10^{7}\}$ which is wide enough. The bit $b[6]$ is used to reset $mantissa[i]$ to 0 while $b[10]$ is used to reset $exponent[i]$.

5. EXPERIMENTAL RESULTS

We run experiments to test hypothesis. We first determine by experiments the most optimised BNN-configuration, i.e., the learning rate, the momentum and the convergence condition to ensures the worth of FL.

Data set All training and test characters in our experiments are printed by a 300-dpi laser printer and scanned into the computer with the same resolution scanner. Let E and C stand for the sets of Eucrosia and Cordia fonts, respectively. Let E^{darker} and C^{darker} denote noised Eucrosia and noised Cordia fonts, which are obtained by copying the images by a photocopy machine with darker setting. In the same way, let $E^{lighter}$ and $C^{lighter}$ denote noised Eucrosia and noised Cordia fonts, which are obtained by copying the images by a photocopy machine with lighter setting. These images consist of 77 different Thai characters ('ก', ... , 'ฮ') and 7 sizes (20, 22, 24, 28, 32, 36 and 48 points). Therefore, the size of each set ($|E|$, $|C|$, $|E^{darker}|$, $|C^{darker}|$, $|E^{lighter}|$, or $|C^{lighter}|$) is equal to 77×7 = 539.

Experimental results There are three cases of experiments. We compare our proposed method with the optimised ILP&BNN [2]. All experiments use the same test data set, i.e., $E^{darker} \cup C^{darker} \cup E^{lighter} \cup C^{lighter}$ which is of size $|E^{darker} \cup C^{darker} \cup E^{lighter} \cup C^{lighter}| = 4 \times 539 = 2{,}158$.

Experiment 1 uses the training set $E \cup C$ which is of size 2×539 = 1,078. Experiment 2 uses the training set E which is of size 539. Experiment 3 uses the training set $E^{20} = \{ x \in E \mid x \text{ is a 20-point font} \}$ which is of size $|E^{20}| = 77$. All experiments use the same genetic configuration mentioned earlier. By evaluating the fitness of genomes, the genomes that produce good recognition accuracy are picked. After enough iterations of the GA, the genome that produces the best recognition accuracy is established. The recognition accuracy of the final generation on each experiments are shown in Table 1. We can say that FL affects the changing of synaptic weights, and this makes BNN more flexible for recognising imperfect chracters.

Table 1: Recognition accuracy of the final generation.

Exp.	*#train*	*#test*	*ILP& BNN*	*ILP& BNN&FST*
1	1,078	2,158	94.77%	96.13%
2	539	2,158	84.23%	86.93%
3	77	2,158	67.94%	74.80%
avg.			82.31%	85.95%

6. CONCLUSION

This paper presents an application of FL for improving the approximate match of ILP rules based on the BNN. With the help of FL, the evaluation of the truth values of logic programs is more sophisticated, before the values are fed to the BNN for learning or for recognising. GA is used to find the best fuzzy sets. Currently, the standard fuzzy operations are used, another fuzzy operations may give an additional improvement.

7. REFERENCES

[1] D.E. Goldberg, *Genetic Algorithm in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[2] B. Kijsirikul, S. Sinthupinyo, and K. Chongkasemwongse, "Approximate Match of Rules Using Backpropagation Neural Networks", *Machine Learning*, Kluwer, Volume 44, No.3, pp. 1-32, 2001.

[3] G. Klir, and B. Yuan. *Fuzzy Set and Fuzzy Logic,* Prentice-Hall, 1995.

[4] A. Supanwansa, "An Application of Inductive Logic Programming to Thai Printed Character Recognition", *Master's Thesis, Department of Computer Engineering, Chulalongkorn University*, 1997 (in Thai).

[5] L.A. Zadeh, "Fuzzy Sets". *Information and Control*, Volume 8, pp. 338-353, 1965.

[6] L.A. Zadeh, "Fuzzy Logic". *IEEE Computer*, Volume 1, pp. 83-93, 1988.