

# The Petri Net Threshold Learning Unit

Copyright © 2002, Andrew Seely

Nova Southeastern University  
seelyand@nova.edu

**Abstract.** This paper introduces the Petri Net Threshold Learning Unit (PNTLU), a neural network threshold learning unit (TLU) that is implemented entirely with basic Petri Nets. No modification to the Petri Net structures is required; all TLU processing is accomplished through manipulation of the Petri Net arc connectivity between places and transitions that are created to serve as thresholds and weights. The complexity and growth of the PNTLU is discussed and several special cases are defined. The PNTLU is compared to the similar technologies of the Fuzzy Petri Net and Fuzzy Neural Petri Net in terms of problem domain, implementation, and complexity.

## 1. Introduction

The Threshold Learning Unit (TLU) is a relatively simple idea but is useful only in a neural network setting. Petri Nets have the advantage of being generalist tools; a Petri Net can be used to model a system, to do simple processing, to show relationships between concurrent systems, and to demonstrate parallel processing interactions [1][2]. While TLU networks are theoretically parallel in nature, i.e. two TLUs in the same neural network can operate independently of each other [3], Petri Net transitions are just as parallel but without the added complexity of  $W$  and  $\theta$  attributes. Both the Petri Net and the TLU can easily work with Boolean values, but the Petri Net is less able to handle real or integer numbers. This paper presents a method for implementing the relatively special-purpose TLU with a relatively general-purpose Petri Net.

### 1.1 Petri Net Terms

Petri Nets have been thoroughly defined in the literature, such as in [1] and [2]. The basic Petri Nets used in this paper conform to the Petri Net behavior described in [2]. For purposes of this paper, a Petri Net is defined as the five-tuple  $\{ P, T, I, O, M \}$ , where  $P$  is the set of places;  $T$  is the set of transitions;  $I$  is the set of places from which arcs are directed to each transition  $t$  in  $T$ ;  $O$  is the set of places to which arcs are directed from each transition  $t$  in  $T$ ;  $M$  is the quantity of tokens present in each place  $p$  in  $P$ .

Common conventions are followed when graphically depicting Petri Nets in this paper: A circle for a place, a dot for a token, a bar for a transition, and an arrow for an arc. Places are numbered (1..n) and transitions are numbered (1..m). Tokens and arcs are not numbered as their attributes are dependent on the places and transitions of the net.

## 1.2 TLU Terms

The TLU employed by this paper is similar to the TLU defined by [3] or the Adaline defined by [4], and is essentially a sub-case or predecessor of neural network structures like the Perceptron or Radial Basis Function processing elements. [4] This paper uses some typical TLU nomenclature:  $X_{1..n}$  is the set of  $n$  input values;  $W_{1..n}$  is the set of weight values;  $\theta$  is the TLU threshold. The TLU described here also includes a separate element called a *concentrator*, equivalent to the summation unit that is commonly bundled with the threshold portion of the TLU. In all respects, the TLU in this paper conforms to the TLU behavior defined in [3].

## 2. Building a TLU with Petri Nets

The Petri Net Threshold Learning Unit (PNTLU) is a Petri Net that simulates the features of a TLU by varying the number of places and transitions and the degree of interconnectivity between them. Adjusting an input weight or  $\theta$  is done by reorganizing the Petri Net arcs to accommodate the changing quantities of places and transitions. It is useful to describe the PNTLU in terms of three 'layers': The Input Layer contains the 'input' places and 'weight' transitions; the Processing Layer contains 'weight' places and 'threshold' transitions; the Output Layer contains the 'output' place.

Figure 1 shows an example of a PNTLU. This PNTLU's attributes may be compared to those of a TLU:

- The TLU input set  $X_{1..n}$ ,  $n = 3 \approx$  the Petri Net marking  $M(p_1, p_2, p_3)$
- The TLU weight set  $W_{1..n} = \{ w_1, w_2, w_3 \} \approx$  the number of Petri Net output arcs for each transition in  $O(t_1, t_2, t_3) = \{ 3, 1, 2 \}$
- The TLU  $\theta$  is 3  $\approx$  the Petri Net configuration at its Processing Layer

These attributes are artificially assigned to the Petri Net and only have meaning in this context; in all respects, Figure 1 is a pure Petri Net. The following discussion demonstrates the functionality of a PNTLU, using Figure 1 for reference.

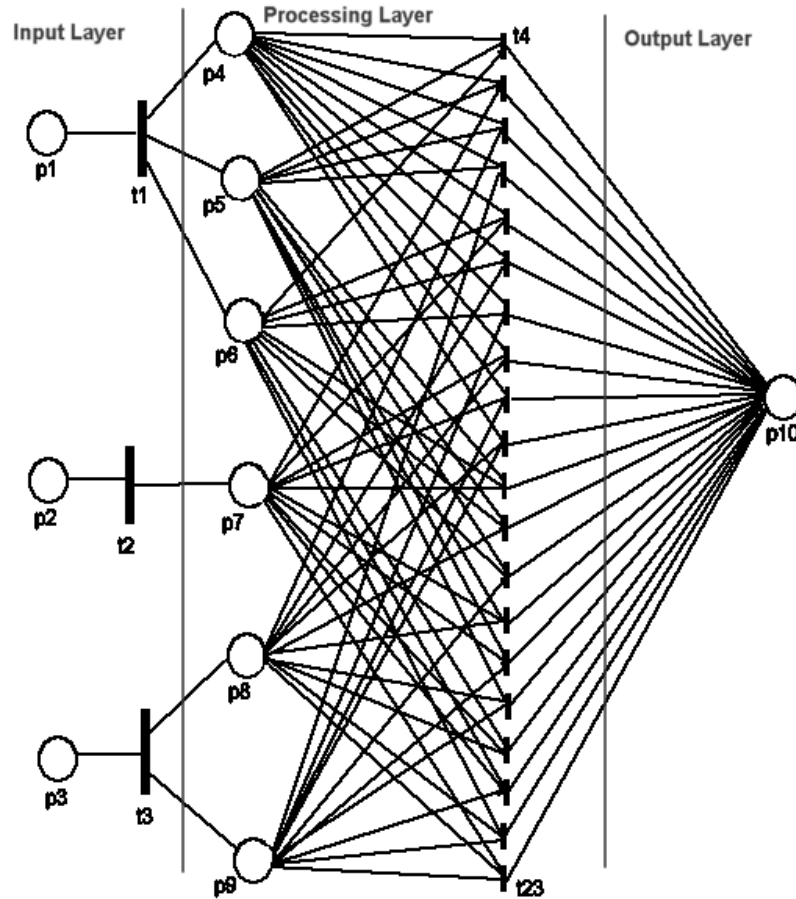


Fig. 1. Petri Net Threshold Learning Unit with  $W = \{ 3, 1, 2 \}$ ,  $\theta = 3$

The PNTLU logically functions in the same manner as the TLU. The TLU input set  $X = \{ 1, 0, 1 \}$  is represented in the PNTLU by initial state markings  $P_{1-10} = \{ 1, 0, 1, 0, 0, 0, 0, 0, 0, 0 \}$ . In this example, a token present in  $p_{1-3}$  is equivalent to a *true* input value in  $X_{1-3}$ . Places  $p_{1-3}$  are designated as the input and  $p_{10}$  is the output and may be designated  $p_{\text{output}}$  for clarity.

The set  $W$  of the TLU is represented by the relationship between  $t_{1-3}$  and  $p_{4-9}$ . The presence of an input token on any of  $p_{1-3}$  will activate the transition that satisfies the value  $I(t_n) = p_n$ . This transition in turn places tokens in its output places. The number of places connected to the output of each transition at this layer is analogous to the weight value for that input. Adjusting the weight associated with an input  $p_n$  is accomplished by increasing or decreasing the number of places found in the set  $O(t_n)$ .

In the example from Figure 1:

$I(t_1) = \{ p_1 \}$  and  $O(t_1) = \{ p_4, p_5, p_6 \}$ . In TLU terms,  $x_1 \approx M(p_1)$ ,  $w_1 = 3$ .  
 $I(t_2) = \{ p_2 \}$  and  $O(t_2) = \{ p_7 \}$ . In TLU terms,  $x_2 \approx M(p_2)$ ,  $w_2 = 1$ .  
 $I(t_3) = \{ p_3 \}$  and  $O(t_3) = \{ p_8, p_9 \}$ . In TLU terms,  $x_3 \approx M(p_3)$ ,  $w_3 = 2$ .

In Figure 1, the Processing Layer places  $p_{49}$  are interconnected to the Processing Layer transitions  $t_{23}$ . The number of transitions at this layer and the degree of interconnectivity are determined by the threshold  $\theta$ . In this example,  $\theta = 3$ . Each Processing Layer transition  $t$  has  $\theta = 3$  places in its  $I(t)$  input set. The degree of interconnectivity is determined by the number of mathematical combinations of size  $\theta$  that are possible from the number of places in the Processing Layer. The number of required Processing Layer places may be found by summing the number of output arcs from the Input Layer transitions, as shown in Formula 1.

$$\Sigma W = \sum_{i=1}^q O(t_i). \quad (1)$$

The number of Processing Layer transitions  $R$  is defined by the combinations formula, as shown in formula 2, where  $p$  is the number of Input Layer transitions and  $q$  is  $\theta$ .

$$R = C(p, q) = \frac{p!}{q!(p-q)!} \cdot [5] \quad (2)$$

For the example in Figure 1,  $R = C(\sum_{i=1}^3 O(t_i), \theta) = C(6, 3) = 20$  Processing Layer transitions. For each transition in  $R$  there is one arc:  $O(p_{\text{output}})$ .

### 3. PNTLU by Example

The logical flow of the PNTLU is as follows. Boolean input values are received from the environment or from another PNTLU in the form of tokens (*true*) or no tokens (*false*). A particular input is given an integer weight of 1 for no change or greater than 1 to increase that input's influence on the PNTLU output in proportion to the other inputs; weights are represented by the number of places in the output  $O(t)$  set of the input's transition. Since every transition in  $R$  is connected to  $\theta$  Processing Layer places and since  $R$  includes every possible combination of  $\theta$  Processing Layer places, any occurrence of  $\theta$  tokens in the Processing Layer places will cause at least one Processing Layer transition to fire. Since every Processing Layer transition has

only  $O(p_{\text{output}})$ , the output place  $p_{\text{output}}$  will receive a token representing a PNTLU output of *true* whenever any transition in  $R$  fires.

Consider an example of a TLU with  $X_{1..3} = \{ 1, 0, 1 \}$ ,  $W_{1..3} = \{ 3, 1, 2 \}$ , and  $\theta = 3$ . In Petri Net terminology,  $X$  is equivalent to the marking set  $M = \{ 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, \}$  in the PNTLU from Figure 1. This  $X$  causes  $t_1$  and  $t_3$  to be activated. When  $t_1$  and  $t_3$  fire, the tokens from  $p_1$  and  $p_3$  are removed and new tokens are placed in all places found in  $O(t_1, t_3) = \{ p_4, p_5, p_6, p_8, p_9 \}$ . Five places in the Processing Layer have tokens, which is equivalent to a TLU having a value of 5 at the concentrator. Since the interconnectivity of the Processing Layer transitions is designed such that any combination of three or more Input Layer places with tokens will cause a Processing Layer activation, on the next firing at least one of the Processing Layer transitions  $t_{2-3}$  will fire. A consequence of this firing is that other activated transitions at this layer may be disabled, but this does not have an effect on the overall truth-value of the PNTLU since only one firing is required to put a token in the output place. If  $\theta$  in this example were to be increased to 6, none of the transitions in the Processing Layer would have sufficient tokens to allow activation.

Certain behaviors of the PNTLU  $\theta$ , the quantity of  $R$ , and the quantity of weight places  $\Sigma W$  can be observed:

1. Normal PNTLU behavior requires the relationship  $0 < \theta \leq \Sigma W$ .
2. If  $\theta \leq 0$ , the PNTLU output place is *dead* [2]. Conceptually, however, this state should imply that the PNTLU is always *true*.
3. If  $\theta > \Sigma W$ , the PNTLU output place is *unreachable* [2]. Conceptually and practically, this state implies that the PNTLU is always *false*.
4. If  $\Sigma W = \theta$ , then  $R$  consists of a single transition with  $I(t) = \{ W \}$ .
5. If  $\theta = 1$ , then  $R$  consists of  $\Sigma W$  transitions, each with  $I(t_i) = \{ p_i \}$ .
6. If  $1 < \theta < \Sigma W$ , then the number of transitions in  $R$  is defined by Formula 2:  $C(\Sigma W, \theta)$ .

For purposes of this paper, conditions 2 and 3 are undefined.

#### 4. Training the PNTLU

*Training* a TLU is accomplished by adjusting its weights and/or  $\theta$  [3]. Conceptually, this process is exactly the same for the PNTLU. In the PNTLU, changing  $\Sigma W$  or  $\theta$  in the governing formula  $C(\Sigma W, \theta)$  will have the effect of increasing weights or the threshold for the overall PNTLU, respectively. Changing weights for a single input is a matter of changing the  $O(t)$  set for that input transition to properly reflect the input's proportion to the new  $\Sigma W$  value.

Modification of  $\Sigma W$  or  $\theta$  requires a "re-wiring" of the PNTLU between the Processing Layer places and the output place. This may be accomplished by the following algorithm:

1. Modify  $\theta$ ,  $\Sigma W$  to bring the PNTLU behavior into tolerance
2. Recalculate  $O(t)$  for all transitions at the Input Layer
3. Remove all transitions at the Processing Layer
4. Remove all places at the Processing Layer

5. Create  $\Sigma W$  places at the Processing Layer and connect them to satisfy the new  $O(t)$
6. Create  $R = C(\Sigma W, \theta)$  transitions and set  $O(t)$  to be  $p_{\text{output}}$  for each transition  $t$
7. Derive the set of combinations of size  $\theta$  that can be made from the set  $W$ ; for each combination, set the combination members to  $I(t)$  and move to the next  $t$  in the new Processing Layer transition set  $R$ .

## 5. PNTLU Special Cases

In contrast to a TLU, the basic PNTLU described in Section 2 is not able to handle negative input or negative weights, nor is it able to deal with real numbers. The domain of a simple Petri Net is the set of positive integers and the input/output domain of a PNTLU is the set of Boolean values  $\{ 0, 1 \}$ . This makes the PNTLU less functional than a TLU in some neural network settings, although as the previous section demonstrates, both the PNTLU and the TLU function equivalently when dealing with a Boolean input/output domain.

### 5.1 PNTLU with Negative Inputs and Weights

The PNTLU could be modified to allow negative weights or negative input values, creating an input range of  $\{ -1, 0, 1 \}$ , but this modification would increase the interconnectivity complexity of the PNTLU considerably. Allowing negative weights or inputs would require new 'subtraction' layers which would function in a similar manner to the Processing Layer, except that the Subtraction Layer output would be to a 'trash can' place [1] [6] instead of to an output place. Negative-weight or negative-input Subtraction Layer transitions would serve to remove tokens from the processing layer as they were placed there from the Input Layer.

### 5.2 PNTLU with Integer Input

The PNTLU is limited to Boolean input, while the TLU allows real or integer input. Real numbers are outside of the domain of Petri Nets, but it may be possible to build a PNTLU that accepts positive integers for input. Integer input could be represented by the number of tokens placed in an input place. Since an input value is modified by multiplication with an associated weight, it follows that the PNTLU Processing Layer would need to be increased by a factor of the input quantity. Dynamic reconnecting of the arcs and places in the PNTLU's Processing Layer for every new set of inputs would accomplish this goal. Multiple-token input would be treated as a special case by essentially cloning the input's weight set by the quantity of input, but this would require addition of new Processing Layer places and reconnecting of Processing Layer transitions after input was presented to the PNTLU. For example, an input of 3 with a weight of 2 would be treated by the PNTLU, after reconnection, as three inputs of *true*, each with a weight of 2. The influence at the

Processing Layer would be the same:  $3 \bullet 2 = 6$  for the TLU compared to  $\sum_{i=1}^3 (1 \bullet 2)$   
 $= 6$  for the PNTLU.

### 5.3 PNTLU with Integer Output

The PNTLU could be modified to give an integer value as output rather than just a simple Boolean. This behavior is not consistent with TLU function, but is an interesting side effect of the Petri Net architecture. If the conflict between Processing Layer transitions were overcome, the number of tokens in the output place would represent the degree of satisfaction the PNTLU found from its input set. The non-removal of tokens from places simply implies that the motivation for their original placement remains true [6]. The PNTLU behavior could be modified to not remove tokens, which would prevent Processing Layer transitions from disabling each other but would add three new requirements to the PNTLU. First, removal of input tokens would become the responsibility of the environment; i.e. when the conditions change that caused the input to be created, the input is removed. Second, any change to the input places would require the PNTLU to re-process from input to output. Third, all activated PNTLU transitions would only fire one time per processing cycle to prevent transitions from firing an infinite number of times. These modifications would cause the output place to be filled with a number of tokens that represent the number of times  $\theta$  was reached in the processing layer. For example, if  $\theta = 3$  and there are 12 marked places in the Processing Layer then 4 Processing Layer transitions will fire, placing 4 tokens in the output place.

## 6. PNTLU Complexity

The complexity of a PNTLU is determined by the number of inputs, the threshold  $\theta$ , and the number of weights  $\Sigma W$  for all inputs. In a software context, we can say that the 'processing' of a Petri Net is done at the transition; a transition does an *if* operation for each input arc in its  $I(t)$  set. If every *if* is true for the transition, then the transition does a *remove token* operation for each arc in its  $I(t)$  and a *create token* operation for each arc in its  $O(t)$ . In the best-case scenario of no inputs, a transition makes  $\Sigma I(t)$  computations; in the worst-case, with maximum inputs, a transition makes  $2(\Sigma I(t)) + \Sigma O(t)$  computations.

At the PNTLU Input Layer, the number of inputs =  $i$  transitions each with one input arc. The number of output arcs per transition is determined by  $W(t)$  for that transition, but the total number of output arcs for all  $i$  transitions is  $\Sigma W$ . So at the Input Layer of the PNTLU the number of computations ranges from  $i$  to  $2i + \Sigma W$ .

At the PNTLU Processing Layer, the number of transitions  $R$  is determined by  $C(\Sigma W, \theta)$ , where  $W$  is the set of Input Layer transitions from the last paragraph. The number of input arcs per transition is  $\theta$ , so the total number of input arcs at the Processing Layer is  $C(\Sigma W, \theta) \bullet \theta$ . Since every transition at the Processing Layer has

a single output to the output place, the total output arcs at the Processing Layer is the number of transitions at the Processing Layer. The number of computations at this layer ranges from  $C(\Sigma W, \theta) \cdot \theta$  to  $2(C(\Sigma W, \theta) \cdot \theta) + C(\Sigma W, \theta)$ .

Combining the processing requirements for the two layers gives an upper bound for the number of computations done by the PNTLU, as shown in Formula 3.

$$2(\Sigma I(t) + \Sigma O(t) + 2(C(\Sigma W, \theta) \cdot \theta) + C(\Sigma W, \theta)). \quad (3)$$

This formula is mostly influenced by  $\Sigma W$  and  $\theta$  in the combinations portion of the equation. While it appears that the growth of this formula would be factorial, the combinations formula actually causes growth to be modeled by Pascal's Triangle [7] as  $\Sigma W$  and  $\theta$  become large.

## 7. Comparing the PNTLU to Fuzzy Petri Nets and Fuzzy Neural Petri Nets

In Petri Net research, the Fuzzy Petri Net (FPN) [8] and the Fuzzy Neural Petri Net (FNPN) [6] engage problem domains similar to that of the PNTLU. The FPN is a Petri Net that has been modified to allow processing of "fuzzy" truth values. With arbitrary places designated as input and output interfaces to the environment, a FPN is able to accept tokens that represent truth values that are *true* (1), *false* (0), or some measure  $n$  in between true and false ( $0 \leq n \leq 1$ ). The output of a FPN is itself a fuzzy value that may in turn be fed to other FPNs. [8]

### 7.1 Function of the Fuzzy Petri Net

Fuzzy systems are made up of fuzzy truth-value propositions that represent the degree of surety or belief in the event or idea they represent combined with rules that determine a course of action based on the propositions present. For example, a proposition representing how hot a summer day feels could be combined with a rule for how long you must work today to make a decision to turn on the air conditioning. In the FPN, tokens are used to represent propositions to be evaluated. Rules are implemented as transitions in the FPN, which serve as limiters of the "strength of truth" that may pass through them. FPN transitions have a *valve setting* to limit the maximum value that may pass and they have a *threshold setting* that serves to limit the minimum value a fuzzy truth must have in order to pass. Thus, FPN transitions begin to fire when all their input places have tokens, but the transitions only complete the firing process after some amount of comparison processing has been accomplished [8] [9] [10].

The FPN is not intended to act as a learning tool, but rather it serves to process uncertain data and provide results between *true* and *false* that may be further interpreted by external elements of a system. The difference between an FPN and a TLU is in what changes during the lifetime of the processing element. The TLU changes its internal settings to accommodate given inputs and provide desired outputs. The problem domain of the FPN includes decision-making situations where

input and output may be variable within a range yet the decision-making process remains relatively static. The input and output domain of the FPN is the set of real numbers  $n$ , where  $0 \leq n \leq 1$ . [8]

## 7.2 Function of the Fuzzy Neural Petri Net

The FNPB extends the function of the FPN by adding additional properties to the Petri Net primitive structures. The most significant changes introduced by the FNPB are the TLU concepts of weights and thresholds. The FNPB restricts input places to each have a single arc to the input of a single transition, which then has a single output arc to a special common concentrator place. This concentrator place has a single output arc which serves as a threshold. This "threshold arc" connects to a single transition that serves to provide output to the environment via an arbitrary number of places on its output arcs. An FNPB may have multiple concentrator places that may be interconnected to various input weight transitions, and output threshold arcs may be interconnected to various output places. [6]

The FNPB allows the processing of fuzzy truth values presented as tokens in the same manner as the FPN, but the FNPB also allows a degree of TLU-style learning. The FNPB input weight transitions allow different inputs to be given ranked influence on the final outcome just as in a TLU. The output threshold acts to impose a binary order on the fuzzy truth values being processed: while the output may consist of a number of tokens each having a truth value  $n$  of  $0 \leq n \leq 1$ , if there is not a sufficient quantity of these tokens at the concentrator place the FNPB will not deliver any value greater than 0 to the environment. [6]

The FNPB functions as a TLU through supervised learning. Learning is accomplished through the comparison of the FNPB behavior to known input-output sets and the subsequent adjustment of the weight values for each input and the threshold value at the concentrator output. In this respect, the FNPB function is basically equivalent to the TLU, but with greater abstract representation power via the fuzzy truth inherent in each token and lessened neural network applicability through the limitations of the input domain. The problem domain of the FNPB is that of the FPN, but where the decision-making process itself must be able to be adapted to new situations

## 7.3 Comparison of Problem Domains

Both the FPN and the FNPB overlap with the problem domain and implementation of the PNTLU. The FPN and the FNPB are able to provide a unique output based on the effect their internal organizations have on a particular input, just as with the PNTLU. All three techniques require a stretch of the basic Petri Net concept, though the FPN and FNPB require physical modifications while the PNTLU only requires conceptual change. Each technique has a certain type of problem for which it is well suited. FPNs are good for representing and comparing data that are non-binary, but FPNs are not natively able to be 'taught'. FPN output varies based on its inputs' proximities to an established belief; while modification of belief parameters in the

form of valve and threshold settings may equate to learning, it is conceptually a different issue. FNPNS are able to represent and process non-binary data, but also incorporate the native learning ability of a TLU such that the FNPNS may not only represent how a given input relates to known 'belief' data, but it can be taught to only respond when that input is within certain parameters of quantity and input location. The PNTLU does not have any native fuzzy rule processing ability, but it is able to learn required output based on training for known inputs. The PNTLU expands the FNPNS learning ability by allowing weights from the set of positive integers vice the set  $\{-1, 0, 1\}$ . Special cases of the PNTLU may also be implemented to allow input and output as positive integers rather than Boolean values. Table 1 shows a comparison between the FPN, FNPNS, and PNTLU in terms of function and implementation.

**Table 1.** FPN, FNPNS, PNTLU Comparison

	<b>FPN</b>	<b>FNPNS</b>	<b>PNTLU</b>
<b>Type of input</b>	$0 \leq n \leq 1$	$0 \leq n \leq 1$	Boolean (Special cases allow positive integer)
<b>Type of output</b>	$0 \leq n \leq 1$	Boolean value, where true contains a fuzzy value $0 \leq n \leq 1$	Boolean (Special case allow positive integer)
<b>Supervised learning</b>	No	Yes	Yes
<b>'Fuzzy' data tokens</b>	Yes	Yes	No
<b>Requires modification to basic Petri Net structures</b>	Yes	Yes	No
<b>Output suitable for interfacing with others of its kind</b>	Yes	Yes	Yes
<b>May use any standard Petri Net software or analysis techniques</b>	No	No	Yes

## 8. Conclusions

This paper has demonstrated a method of creating a threshold learning unit using simple, unmodified Petri Nets. While this technique may lead to increased net complexity when compared to technologies like the FPN and FNPNS, the relative simplicity of the PNTLU may prove to be superior in terms of net analysis and tool reutilization when used in a neural network. The PNTLU addresses a problem domain that is purely in the realm of neural networks while conforming strictly to a

basic Petri Net definition, and so may be more appropriate than technologies like the FPN or FNPN in artificial intelligence applications where Petri Net structures are desirable.

## References

- [1] Murata, T. Petri Nets: Properties, Analysis, and Applications. Proceedings of the IEEE, Vol. 77 (4). (1989) 541-580
- [2] Peterson, J. L. Petri Nets. ACM Computing Surveys, Vol. 9 (3). (1977) 221-252
- [3] Gurney, K. An Introduction to Neural Networks. Routledge, London (1997)
- [4] Principe, J. C., Euliano, N. R., Lefebvre, W. C. Neural and Adaptive Systems. John Wiley & Sons, Inc, New York. (2000)
- [5] Balakrishnan, V. K. Introductory Discrete Mathematics. Dover Publications, New York. (1991)
- [6] Ahson, S I. Petri Net Models of Fuzzy Neural Networks. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 25 (6). (1995) 926-932
- [7] Knuth, D. E. The Art of Computer Programming, Volume 1: Fundamental Algorithms (2<sup>nd</sup> ed.) Addison-Wesley, Reading, MA (1973)
- [8] Looney, C. Fuzzy Petri Nets for Rule-Based Decisionmaking. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 18 (1). (1988) 178-183
- [9] Chen, S., Ke, J., Chang, J. Knowledge Representation Using Fuzzy Petri Nets. IEEE Transactions on Knowledge and Data Engineering, Vol. 2 (3). (1990) 311-319
- [10] Cherniaev, V. Intelligent Systems: Unified Approach to Knowledge Representation, Analysis and Implementation Based on Fuzzy Petri Nets. Proceedings of the Fourth Bar Ilan Symposium on Foundations of Artificial Intelligence. (1995) 43-50