

Neural Representation of Polygon Layers

José L. Silván (jlsilvan@txstate.edu)

Department of Geography, Texas State University-San Marcos, TX 78666

June 15, 2005

Abstract

A feed-forward network-like representation of polygon layers is introduced in this paper. Unlike vector and raster data models (two of the most popular data models in GIS), the proposed model is meant to be implemented on a machine which has computational units similar to those found in the biological brain. Though the model has been developed for multiple point-in-polygon tests, application to the overlay of multiple polygon layers is also envisaged. The proposed model has the structure of a feed-forward neural network composed of linear threshold functions which not only encodes a layer of disjoint regions onto a number of network parameters, but also provides a computational model of the space. The algorithm to efficiently compute the network parameters is formally described and illustrated with some examples. The potential and limitation of the proposed representation are highlighted in a discussion section at the end of the paper. In particular, it is stressed that this representation looks promising when huge amount of tests are required and, especially, if parallel processing is allowed.

1 Introduction

In order to process information about the real world on computers, a proper representation of information in a form which the computers can handle is required. During this process, one needs to decide what is the most relevant information concerning a particular problem and then how to organize it to be stored on a computer for further processing. In doing this we are dealing with data models (Wise, 2002). Geographical Information Systems

(GIS) are used to perform a number of fundamental spatial analysis operations, which can use any number of analytical processes with a limited number of data models. Two of the most popular data models supported by any GIS are vector and raster representations. In a vector-based system, overlay operations are much more complex than in a raster-based system because, in the former, the topological data is stored as points, lines and/or polygons, which requires more complex geometrical operations to derive the intersected polygons, and the necessary creation of new nodes (points) and arcs (lines), with their combined attribute values.

Most complex geometrical operations generally require intensive use of the point-in-polygon test. The point-in-polygon test is a natural problem in the field of computational geometry and it is often encountered in several computer graphics problems as well as in GIS overlay operations between point and polygon layers. Haines (1994a) provides a thorough comparative treatment of existing point-in-polygon algorithms. Within vector-based model, two main definitions can be found in the literature (Sedgewick, 1988; Dovkin et al., 1988; Haines, 1994a). The first one is the parity rule, in which a line is drawn from the testing point to some other point that is guaranteed to lie outside the polygon. If this line crosses the edges of the polygon an odd number of times, the point is inside, otherwise it is outside. This rule is turned into an algorithm that loops over the edges of the polygon, decides for each edge whether it crosses the line or not, and counts the crossings. The various implementations of this strategy (Bourke, 1987; Franklin, 1994; Haines, 1994b) differ in the way the intersection between the line and an edge are computed and how this rather computationally expensive procedure can be avoided for edges that are known not to cross the line. Of course, the time complexity of the problem is always $O(n)$ for arbitrary n -gons. The complexity can only be reduced for special polygons, such as convex polygons, for which an $O(\log n)$ algorithm can be built (Sunday, 2004). The second approach commonly used is based on the winding number, which is the number of times the polygon winds around the testing point. The point is outside only when this winding number is equal to 0; otherwise, the point is inside (Sunday, 2004). For self-intersecting polygons the winding number provides a more general way for defining the interior of a polygon. A comparison of several implementations of the winding number strategy for arbitrary polygons is provided by Hormann (2001).

Unlike vector-based approaches the constructive solid geometry (CSG) defines polygons by means of Boolean operators (union and intersection) of the half planes defined by the edges of the polygon (Dovkin et al., 1988). The main idea underlying this problem is to come up with a Boolean formula that might be used to test several points. The representation has been used in a practical solution for raytracing in computer graphics where many points must be tested against a given polygon (Walker, 1999). In that work, empirical evidence that a point-in-polygon algorithm using an edge sequence graph derived from a CSG representation is faster on most non-convex polygons than all the vector-based methods presented by Haines (1994a). The CSG representation can be readily linked to a neural network since any Boolean function can be computed with a two-layer feed-forward threshold network (Antony, 2003). This is of prime importance since a great number of neuro-computational models have been proposed in the literature for varied applications, and techniques for parallel neurocomputing have been developed as well during the last decade. Unfortunately, practical methods for determining the optimal network in terms of the minimum number of computational units required are rare in the literature. Reducing the size of the network might not be of major concern for some applications; however, for the point-in-polygon problem in GIS, time complexity is very important given the current trend of increasing size in data sets.

These previous results, the fact that neuroprocessor technology is widely available and in continuous development and that trends in growing data volume and increasing sophistication of analysis have been taking place in the geographical arena (Armstrong, 2000), have triggered the author's interest in seeking for an alternate representation suitable for set of contiguous polygons as those used in GIS to represent partitions of the geographic space. In the discussion for justification of data models in GIS, it is sometimes stated that vector-based systems developed as a "marriage of traditional cartographic imagination and computational capacity of early computers" (Schuurman, 1999). In this sense, computer models have determined the physical and, up to some extent, the conceptual representation of real world data. Although the application-driven approach has also permeated into the discussion for defining more appropriate spatial data models at the conceptual level (Peuquet, 1988), a technology-driven argument is made here: as new computational models and technologies become available, the interface between conceptual

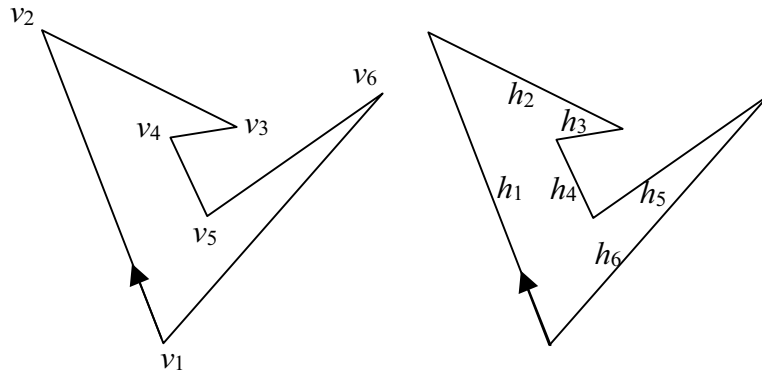


Figure 1: Vertex representation vs. Half-space representation

frameworks and computational models need to be reconciled in order to exploit these new capabilities.

The central issue with the proposed approach is how to train the network, i.e., the strategy to adjust the network parameters. Two of the most widely used learning models are the perceptron (Rosenblatt, 1962) and WINNOW (Littlestone, 1989) algorithms. Unfortunately, whether or not Boolean formulas can be efficiently learned using this kind of methods has remained one of the most challenging open problems in the field of computational learning theory since Valiant (1984) introduced the probability approximately correct (PAC) model. A more reliable and efficient solution to determine the network parameters for representing polygon layers is developed in this paper. The research focused on developing a new design strategy of the network structure that is claimed to be more efficient in the sense that the least number of processing units (neurons) are used in representing the interior of a set of contiguous polygons. The network can then be simulated to perform the overlay operation among point and polygon layers. It is emphasized that more complex set operations may be readily implemented using this model and that further generalizations are allowed as well. Because the model has been developed at computational level, the issue of the attribute space required when modelling geographical entities is not addressed here.

2 From vertices to half-spaces

The translation of polygon representation from the vector model to the half-space model is described in this section. The process can be summarized as follows. Let $v_i = (x_i, y_i)$, for $i = 0, \dots, n$, with $v_n = v_0$, denote the vertices of a given polygon (see for example Figure 1). A line containing the two consecutive vertices v_i and v_{i+1} can be written as $\alpha_i x + \beta_i y = \rho_i$ with $\alpha_i^2 + \beta_i^2 = 1$. The later constraint is imposed to maintain the lowest number of degree of freedoms, which can be defined by the angle θ_i formed with the line and the x -axis (therefore, $\alpha_i = \cos(\theta_i)$ and $\beta_i = \sin(\theta_i)$) and the distance ρ_i from the line to the origin. Each of such lines defines the boundaries of two half-planes so that, when orienting the edges clockwise about the polygon, the half plane to the right of the edge is considered inside the polygon, and the one to the left outside. The relative position of a point with respect to the oriented lines is determined by the half-plane indicator variables

$$h_i = \begin{cases} 1, & \text{if } \alpha_i x + \beta_i y \geq \rho_i \\ 0, & \text{if } \alpha_i x + \beta_i y < \rho_i \end{cases} \quad (1)$$

for $i = 1, \dots, n$. A number of intersection and union operations of such half-planes, performed in the correct order, defines the polygon, and the half-planes are said to support the polygon. This representation is commonly referred to as the constructive solid geometry (CSG) representation in computer graphics (Dovkin et al., 1988; Walker, 1999).

Dovkin et al. (1988) showed that any simple polygon, i.e., non-self-intersecting, allows a monotone-formula representation in which each literal corresponding to a side of the polygon appears exactly once. They also gave an efficient algorithm to construct the CSG representation for simple polygons. The resulting representation is in either the form of half-plane representations or as a direct binary tree whose nodes contain Boolean operators. An overview of the algorithm is provided here, and interested readers are referred to the original for more details.

Let us assume that the edges are oriented clockwise and define the half-plane to the right of an edge to be the one that supports the polygon. The space covered by the polygon is then described by a combination of Boolean AND (\wedge) an OR (\vee) operators acting on the half-plane supporting the polygon. If two edges meet at an acute angle, the interior of the polygon is described by an AND of the adjacent half-planes, whilst an

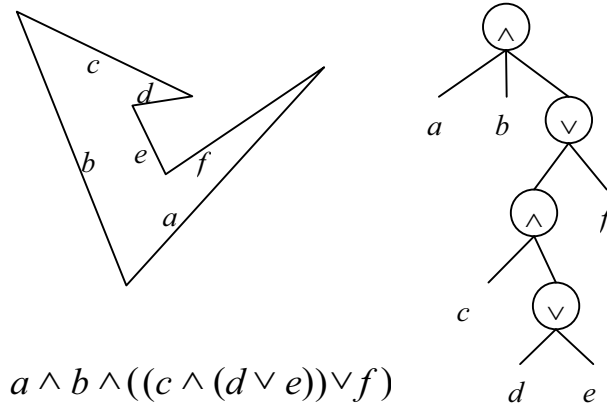


Figure 2: CSG representation of a polygon by its monotone formula and equivalent binary tree

obtuse angle require and OR operation of these half-planes. In this way, the sequence of operations that describe the interior of the polygon can be built up, but the correct order in which these operations must be applied (term grouping) has to be determined.

Term grouping (or parenthesization) starts by dividing the polygon into two bi-infinite chains by splitting it at its left- and right-most vertices and extending the edges incident upon these vertices to infinite. The convex hull of a bi-infinite chain is determined and the extremal vertex in the direction opposite the vector sum of the two semi-infinite edges of the chain is found. The chain is then split at this vertex, and the incident edges extended to infinity, forming two smaller chains; the process proceeds recursively until all the chains contain only one edge. Whenever a split occurs, the half planes corresponding to the edges in a chain are grouped within parentheses in the Boolean formula, which result in a nested formula consisting of unambiguously ordered binary operations. Dovkin et al. (1988) showed that this procedure is practical in the sense that $O(n \log n)$ operations are needed for an n -edge polygon.

The binary tree representation is finally obtained from the Boolean formula: a half-plane becomes a leaf node, while the binary operator becomes the root of a tree whose children are half-planes or subtrees formed by subformulae. The binary tree obtained can be consolidated into n -ary form by merging adjacent Boolean operator nodes of the same type. For example, in the Boolean formula of Figure 2, the parenthesis enclosing $a \vee b$ was removed. As a result, the root of the tree has three children. The n -ary form is used here to compute the neural network representation derived in the following sections.

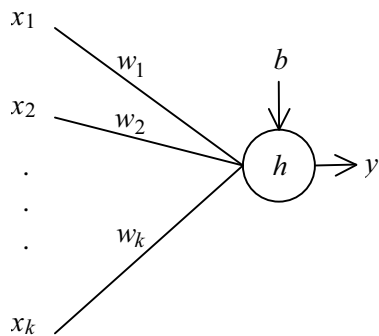


Figure 3: Linear threshold unit

3 Neural networks

The building blocks of feed-forward networks are computation units (or neurons). In isolation, a neuron has some number, k , of inputs, and is capable of taking on a number of states, each described by a vector $w = (w_1, w_2, \dots, w_k)$ of k real number called synaptic weights and a real number b called the bias or activation level. Generally, when in the state described by (w, b) , and on receiving input $x = (x_1, x_2, \dots, x_k)$, the computation unit produces as output an activation $h(x; w, b)$.

3.1 Linear threshold units

For a linear threshold unit (see Figure 3 for a graphic representation), the activation function takes the particularly simple form $h(x \cdot w - b)$, where \cdot denotes the scalar product of vectors and

$$h(s) = \begin{cases} 1, & \text{if } s \geq 0 \\ 0, & \text{if } s < 0 \end{cases} \quad (2)$$

Thus, the output is 1 precisely when $w_1x_1 + w_2x_2 + \dots + w_kx_k \geq b$, and 0 otherwise. If the inputs to the threshold unit are restricted to $\{0, 1\}^k$, then the Boolean function implemented is known as a (Boolean) threshold function.

Single linear threshold units have very limited computational abilities, but when interconnected as a feed-forward network, they become more useful for several computational tasks such as recognition, classification or Boolean function mapping.

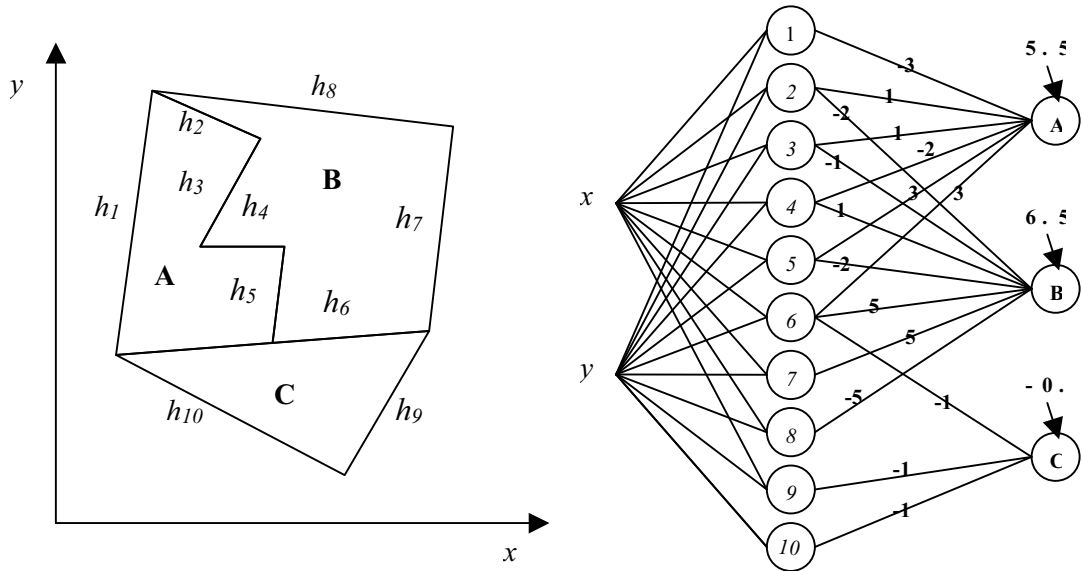


Figure 4: Example of network representation of a simple map

3.2 Computational model of maps

In the proposed model a layer of linear threshold units is used to compute the half-spaces defined by the edges of contiguous polygons. Subsequent layers must take the output of the first layer to perform the Boolean mapping as determined by the CSG representation. In order to illustrate this idea, an example is now introduced.

Consider the simple map composed of three regions shown in Figure 4. A total of ten edges are defined in terms of the half-space representation. Notice that redundancy due to shared edges can be readily avoided within this representation. Unlike vertex-list representation, a strict order of half-space indicator variables h_i is not mandatory. Instead, one needs to keep track of the half space each variable supports, i.e., where it takes the value of 1. The convention of placing the variable on the side of the half space supported by it has been used in this example. Thus, h_1 does not support polygon **A** but its complement (h'_1) does, h_4 supports polygon **B** and its complement (h'_4) supports **A**, and so forth. The neural network depicted in Figure 4 takes as input a point (x, y) and produces three binary outputs. Each processing unit is a linear threshold defined by its corresponding parameters. The units in the first layer, labelled as 1, 2, ..., corresponds to the half-space indicator variables as defined by the edges (Equation 1), i.e., the weights and bias parameter for this layer of neurons can be determined from the vertex-list as described in Section 2. Using the defined convention, a point lying in the right-hand side of edge h_1

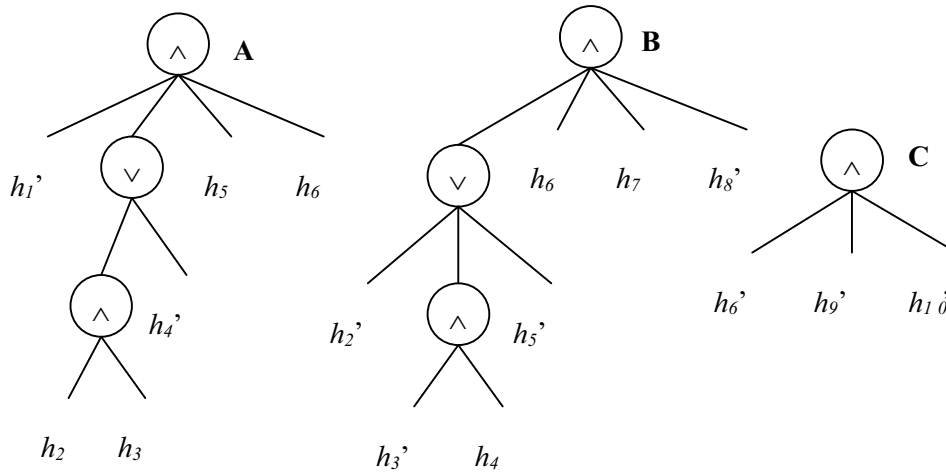


Figure 5: Tree representation for the three polygons of Figure 4

causes neuron 1 to produce an output 0, while a point lying in the left-hand side of the same edge will cause that neuron to produce an output 1. The output layer defines the membership function for each region. Thus, if the input point lies in the region defined by polygon **B**, the corresponding neuron becomes active (output 1) whereas the neurons **A** and **C** neurons remain inactive (output 0). Determining weights and bias parameters for the output layer can be far more difficult than for the first (hidden) layer. In general, the mechanism by which these parameters are determined is referred to as training.

During a typical training process a network with initial parameters is presented with some input-output exemplar pairs, if the input does not fit the output then the weights are systematically adjusted and the process is repeated with a reasonable number of examples. The training process is repeated several times till the network is capable of reproducing all the examples with a tolerable accuracy. The major issues with this procedure are how to select the appropriate training sample and what is the number of required units that guarantee convergence. In general, the approaches to address those issues are rather computationally expensive. As an alternative, a new method based on the tree derived from the CSG representation was developed. The training strategy is described in the next section.

Table 1: Boolean threshold training using the tree **A** of Figure 5. (\diamond denotes unassigned values)

BO	k	h'_1	h_2	h_3	h'_4	h_5	h_6	s^{min}	ϕ	τ	s^{max}
\wedge	1	\diamond	1	1	\diamond	\diamond	\diamond	0	1	2	2
\vee	2	\diamond	1	1	-2	\diamond	\diamond	-2	-1	0	2
\wedge	3	-3	1	1	-2	3	3	-5	5	6	8
$w =$		-3	1	1	-2	3	3	$b =$		5.5	

Table 2: Boolean threshold training using the tree **B** of Figure 5. (\diamond denotes unassigned values)

BO	k	h'_2	h'_3	h_4	h'_5	h_6	h_7	h'_8	s^{min}	ϕ	τ	s^{max}
\wedge	1	\diamond	-1	1	\diamond	\diamond	\diamond	\diamond	-1	0	1	1
\vee	2	-2	-1	1	-2	\diamond	\diamond	\diamond	-5	-4	-3	1
\wedge	3	-2	-1	1	-2	5	5	-5	-10	6	7	11
$w =$		-2	-1	1	-2	5	5	-5	$b =$		6.5	

4 Learning the map

Learning complex Boolean functions has been recognized to be far more complicated than solving ordinary pattern recognition problems (Kwek, 1996). Conventional training algorithms such as Rossmblat's perceptron (Rosenblatt, 1962) and the Littlestone's WINNOW (Littlestone, 1989) algorithms suffer from a long convergence time and often fail in finding solutions to such Boolean mapping problems. Furthermore, finding the minimum number of neuron units required for the mapping is not an easy task. Antony (2003) presents a thorough discussion on the application of artificial neural networks to Boolean functions mapping. He provides a proof that any Boolean function on $\{0, 1\}^n$ can be computed with a two-layer threshold networks with 2^n hidden units and gives the notion of the universal network. A universal network for Boolean functions is a threshold network which is capable of computing every Boolean function of n variables. A natural question is what is the universal network with fewest threshold units. In particular it can be shown that any universal network (regardless of how many layers it has) must have at least $\Omega(2^{n/2}/\sqrt{n})$ threshold units. Moreover, any two-layer universal networks for Boolean

functions must have at least $\Omega(2^n/n^2)$ units (Antony, 2003). From the point of view of polygon representation, only a small fraction of the total number of Boolean function represents valid polygons in the two-dimensional space. The reason is that of the 2^n primitive AND terms one can form on n literals (with complementation allowed), only $\Theta(n^2)$ are non-zero and, therefore, denote non-empty regions of the plane (Dovkin et al., 1988). Therefore, instead of determining a universal network for any Boolean function one can construct a network for a particular polygon based on its n -ary tree representation.

4.1 The training algorithm

In the n -ary tree representation, we call *literal node* a leaf node that represents a half-space indicator variable, *operator node* a node that represents the Boolean operation of its children (operator or literal nodes); an *ending node* is an operator node which does not have operator nodes as children, and a *chain* is a sequence of linked nodes each of which has at most one operator node as a child. Monotonous formulae have the property that alternate AND and OR operator nodes comprise any chain of its associate n -ary tree expansion.

Let n be the number of operator nodes acting on the literals h_1, h_2, \dots, h_m of a chain C , and let b , and w_1, w_2, \dots, w_m , be the bias and weight parameters of a linear threshold T in which its i -th input is set to $h_i \in C$. The following proposition can be made on C and T .

Proposition 1 *Any Boolean function $f : \{0, 1\}^m$ which has a tree representation in the form of a chain C can be implemented with a single linear threshold T .*

Proof A constructive proof will consist in determining b and w_i , for $i = 1, \dots, m$, such that $T = f$ for all $h \in \{0, 1\}^m$. Let us define $s = w_1h_1 + w_2h_2 + \dots + w_mh_m$; an input point h is said to lie in the false region of f , if $s < b$, whereas it is said to lie in the true region of f , if $s \geq b$. The function f is evaluated by traversing the chain from the bottommost operator node ($k = 1$) to the uppermost operator node ($k = n$), the computation of b and w can be performed by following the same route. Let assume that the bottommost node is an AND operator. In this case there is only one input combination that produces a true value: when all the non-complemented operands are true (1s) and all complemented operands are false (0s). Now, weight with 0 all the literals that are not children of the node and weight the children with 1 if they are non-complemented and -1 if they are complemented.

Clearly, for the true point the variable s will equals the number of non-complemented literals in the node, say p , whereas for any other combination, s can take at most the value $p - 1$. Therefore, the threshold value that separates the true and false points must be in between $p - 1$ and p , say $p - 0.5$. At this point, the reader is invited to perform a similar analysis assuming that the bottommost node is an OR gate and prove that a threshold $0.5 - q$ separates the true and false points in this case, where q is the number of complemented literals.

Before continuing with the upper nodes it is convenient to introduce some parameters. Let s_k^{min} and s_k^{max} be the minimum and maximum values of s respectively provided that all the weights for literals under the subtree at node k has been already assigned. Similarly, let ϕ_k be the maximum of s in the false region and τ_k the minimum of s in the true region. Notice that a partial implementation of f at node k is possible only if ϕ_k is less than τ_k since that is the only case in which we can define a threshold b such that $\phi_k < b < \tau_k$. As shown before, for $k = 1$, $s_1^{min} = -q$, $s_1^{max} = p$, and $\phi_1 = p - 1$ and $\tau_1 = p$ for an AND gate, whereas $\phi_1 = -q$ and $\tau_1 = 1 - q$ for an OR gate. In either case $\tau_1 = \phi_1 + 1$. The constraint $\tau_k = \phi_k + 1$ can be imposed for arbitrary k to ensure that $\phi_k < \tau_k$.

Let us now consider a generic node k in the chain. Set all the weights for the literals which are children of the node k to $-a_k$ if the literals are complemented and to a_k if not. The problem is now reduced to choose parameters a_k which always guarantee the linear separability of true and false points of f . The minimum contribution to s at level k can be $-q_k a_k$ where q_k is the number of negated literals and the maximum contribution to s can be $p_k a_k$ where p_k is the number of positive literals at the node. Therefore, we can update the minimum and maximum values of s as

$$\begin{aligned} s_k^{min} &= s_{k-1}^{min} - q_k a_k \\ s_k^{max} &= s_{k-1}^{max} + p_k a_k \end{aligned} \tag{3}$$

For an AND operator we use again the fact that there is only one case giving a true value in which the sub-chain (node $k - 1$) evaluates to true with a contribution to s of at least τ_{k-1} , and all positive literals are true and negated literals are false at level k , which gives a contribution $p_k a_k$. The minimum value of s in the true region is then updated at level k as

$$\tau_k = \tau_{k-1} + p_k a_k \tag{4}$$

For the maximization of s in the false region we have two possibilities. If the sub-chain evaluates to false, then its maximum contribution to s is ϕ_{k-1} . In this case, there is no one combination input to give a true output and therefore the maximum contribution is $p_k a_k$. Therefore, the maximum value that can be achieved is $\phi_{k-1} + p_k a_k$ in this case. On the other hand, if the sub-chain evaluates to true, its contribution can be at most s_{k-1}^{max} and the largest contribution the literals can make, while still evaluating the function to false, is bounded by $(p_k - 1)a_k$ giving a total of $(p_k - 1)a_k + s_{k-1}^{max}$. However, from the constraint $\tau_k = \phi_k + 1$ and Equation 4, we have

$$\phi_k = \phi_{k-1} + p_k a_k \quad (5)$$

Therefore, in order to guarantee the separability of true and false points it is required that $\phi_{k-1} + p_k a_k \geq (p_k - 1)a_k + s_{k-1}^{max}$. This constraint is met only if

$$a_k \geq s_{k-1}^{max} - \phi_{k-1} \quad (6)$$

A similar analysis for the OR operator leads to the following updating equations

$$\tau_k = \tau_{k-1} - n_k a_k \quad (7)$$

$$\phi_k = \phi_{k-1} - n_k a_k$$

with the constraint

$$a_k \geq \tau_{k-1} - s_{k-1}^{min} \quad (8)$$

Therefore, it is always possible to set up the weights so that the true and false points of f are separated by the linear threshold T . \square

In practice, we can use the smallest integer value for a_k that satisfies Equations 6 and 8 for AND and OR operators respectively. The entire procedure can be summarized in the following three steps:

Step 1 Initialization:

- (a) initialize $a = 1$ and $s_{min} = s_{max} = 0$;
- (b) if node 1 represents an AND operation, initialize $\phi = -1$ and $\tau = 0$;
- (c) if node 1 represents an OR operation, initialize $\phi = 0$ and $\tau = 1$.

Step 2 Loop for $k = 1, \dots, n$:

- (a) determine the number of complimented q and non complimented p literals at node k ;
- (b) set all the weights for literals at node k to a for non-complemented and to $-a$ for complemented literals;
- (c) increase s_{max} by pa and decrease s_{min} by qa ;
- (d) if node k represents an AND operation, increase ϕ and τ by pa , and update $a = s_{max} - \phi$;
- (e) if node k represents an OR operation, decrease ϕ and τ by qa , and update $a = \tau - s_{min}$.

Step 3 Set bias parameter: $b = (\phi + \tau)/2$.

Table 1 and Table 2 give the runs of the algorithm with the trees **A** and **B** of Figure 5 respectively. The first column shows the Boolean operator (BO) for each iteration. Subsequent columns shows the value of weights determined at each iteration as well as values of auxiliary parameters.

The question of how to generalize the above approach to implement more complex Boolean functions so that it can be useful to represent complex polygons remains unanswered. This issue is addressed in the following Proposition.

Proposition 2 *The number of linear threshold units required to compute the Boolean formula equals the number of ending nodes of its tree representation.*

Proof Take an ending node and determine the largest chain from it toward the root of the tree. Create a threshold unit for that chain, which is always possible from Proposition 1. Replace the chain by an auxiliary literal node which takes the value of the threshold unit output. Repeat the process until the tree is reduced to the root node. Because each chain built in this way starts with an ending node in the tree, the total number of units created equals the number of ending nodes in the original tree. \square

In order to illustrate the procedure outlined in Proposition 2, consider the polygon of Figure 6 whose tree representation is given in Figure 7. Four ending nodes are identified in the tree; therefore, four threshold units are required to compute the Boolean function.

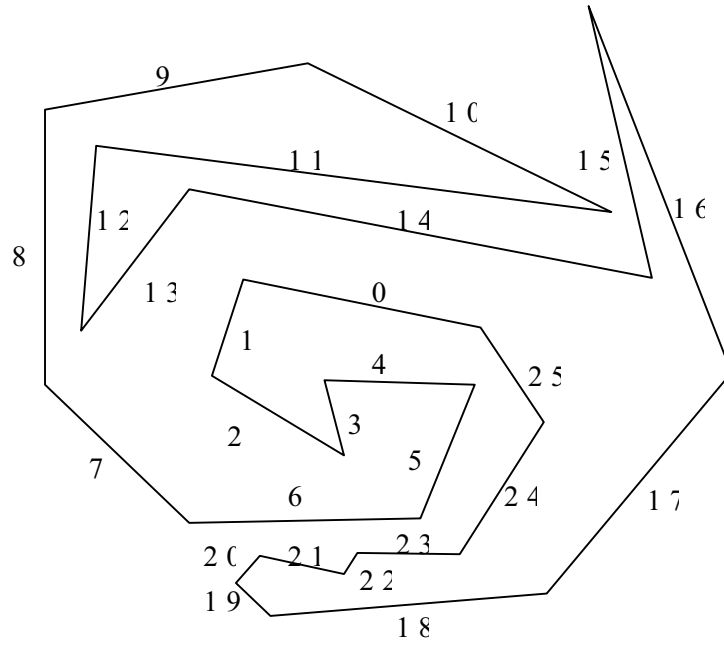


Figure 6: Sample polygon with 26 edges

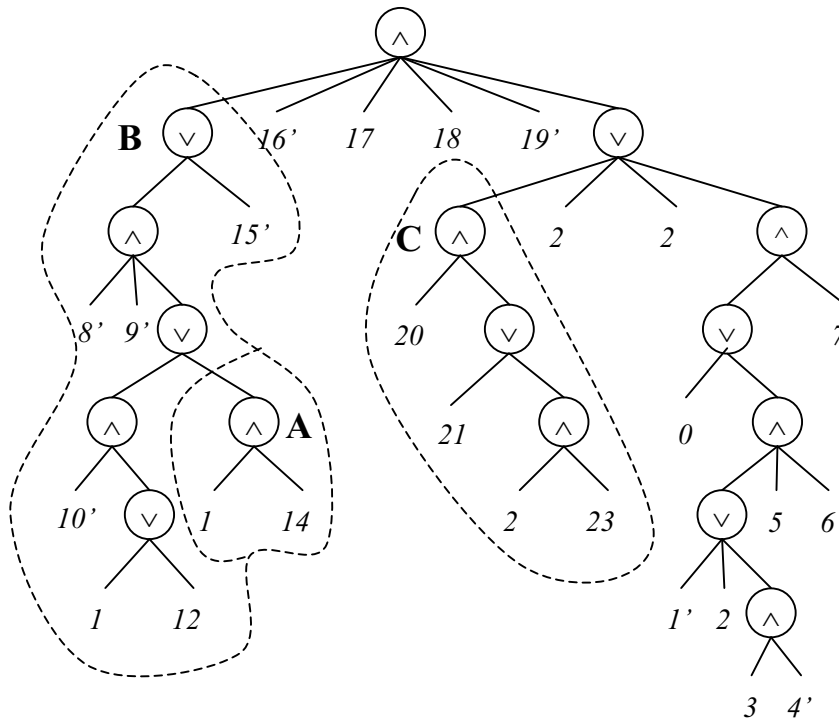


Figure 7: Tree representation for the polygon shown in Figure 6 (based on Dovkin et al., 1988)

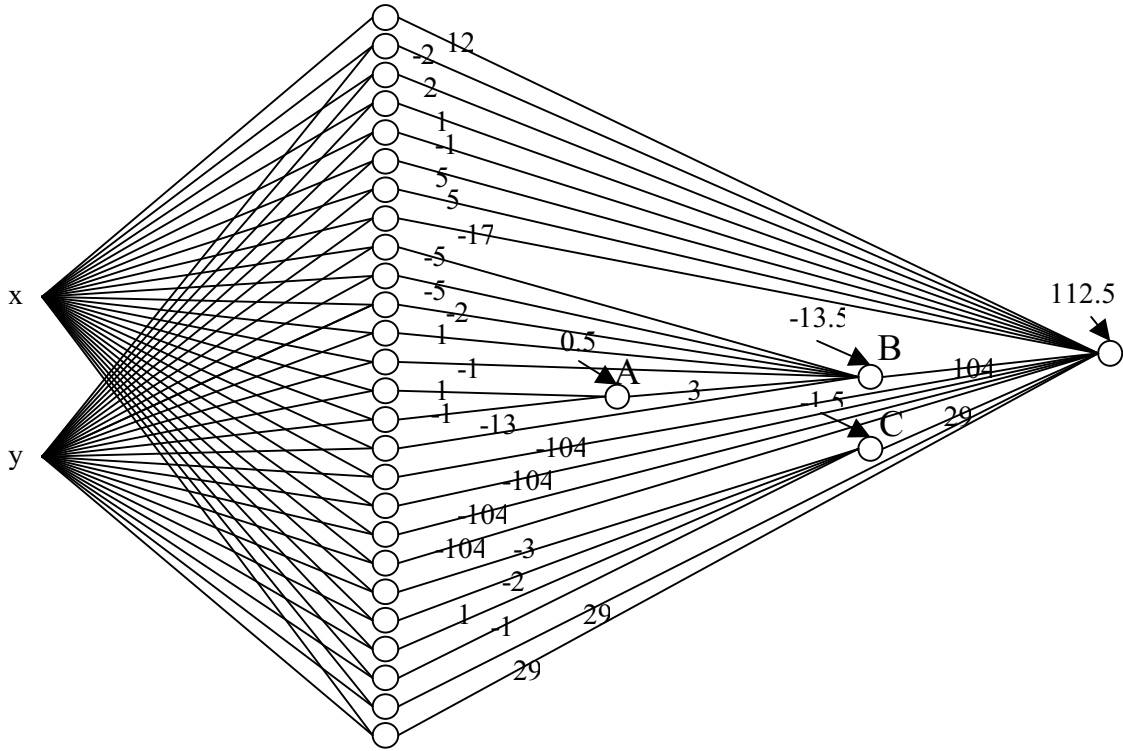


Figure 8: Network representation for polygon of Figure 6

Three of the four chains used to train the network are enclosed (dashed line) in the same figure. The auxiliary literals produced by these chains are **A**, **B** and **C**. Its network representation is depicted in Figure 8. Notice that because each chain is replaced by a literal that represents the output of the new threshold unit, the resulting network has a nested structure, i.e., the threshold unit **A** is an input to unit **B** which in turn, with **C**, are inputs to the output unit. When simulated for a set of 256 by 256 points on a square grid covering the whole polygon, the output shown in Figure 8 was obtained, where the black tone is for an output 0 (outside the polygon) and white tone for an output 1 (inside the polygon).

5 Discussion

A neural network representation for polygon layers was introduced in this paper. The structure might be useful in the overlay of point and polygon layers where several point need to be tested against several polygons. This network representation not only encodes the polygon layer through a set of network parameters, but also provides a computational structure for spatial operations. In this sense, the potential application envisaged is for



Figure 9: Simulated output of the network shown in Figure 8 for input points on a grid of 256 by 256 points

complex set-based operations besides point-in-polygon tests. The evaluation of a complex function in terms of set operations among several polygon layers could be carried out by means of another neural network that takes the output of the corresponding network for each layer and performs a series of Boolean operations. In this way, non-simple polygons (e.g., polygons with holes) could be also represented. Furthermore, the model could be extended to fuzzy domains by letting the activation function turn smoother. It is also interesting to notice that the network structure captures some topological relationship among the polygons so that querying about the neighbors of a given polygon might be readily determined by establishing the appropriate links in the network structure. The major limitation of the model as presented here would be its inability to represent other geometric entities apart from aerial ones, e.g., a river network could not be represented using this model.

Whether this approach can outperform the existing ones remains an issue. Recall that the number of binary operations required to evaluate the Boolean formula is in the worst case $n - 1$, where n is the number of literals, whereas the neural network may require up to 3 times that number of integer operations. Based on this rather superficial analysis, poor efficiency might be argued against the proposed approach, especially because

computational abilities of traditional computers are reduced to bit operations. However, the proposed approach is meant to be implemented on a machine that performs linear threshold as basic computational units such as an state-of-the-art neuroprocessor. In this case, the number of operations could be significantly lower than the traditional approach provided that a single computational unit implements many Boolean operations. This exemplifies the fact that data models are indeed tied to available computer capabilities. Neural processing has the additional advantage of being highly parallelizable which is of great importance when working with huge data sets.

References

- ANTONY, M., 2003, Boolean functions and artificial neural networks. Technical report LSE-CDAM-2003-01, London School of Economics and Political Science.
- ARMSTRONG, M.P., 2000, Geography and computational science. *Annals of the Association of American Geographers*, **90**, 146–156.
- BOURKE, P., 1987, Determining if a point lies on the interior of a polygon. Available online at: <http://astronomy.swin.edu.au/~pbourke/geometry/insidepoly/> (accessed 22 May 2005).
- DOVKIN, D., GUIBAS, L., HERSHBERGER, J. and SNOEYINK, J., 1988, An efficient algorithm for finding the CSG representation of simple polygon. In: *Computer Graphics (SIG-GRAPH '88 Proceedings vol. 22)*, John Dill (ed.), pp. 24–46.
- FRANKLIN, W.R., 1994, PNPOLY - Point inclusion in polygon test. Available online at: <http://www.ecse.rpi.edu/Homepages/wrf/geom/pnpoly.html> (accessed 20 May 2005).
- HAINES, E., 1994a, Point in polygon strategies. In: *Graphic Gems IV*, P. Heckbert (Ed.) (Boston, MA: Academic Press), pp. 24–46.
- HAINES, E., 1994b, CrossingsMultiplyTest. Available online at: http://www.acm.org/tog/GraphicsGems/gemsiv/ptpoly_haines/ptinpoly.c (accessed 16 Nov 2004).

- KWEK, S.S., 1996, Geometric concept learning and related topics. PhD thesis, University of Illinois, IL.
- HORMANN, K. and AGATHOS, A., 2001, The point in polygon problem for arbitrary polygons. *Computational Geometry*, **20**, 131–144.
- LITTLESTONE, N., 1989, Mistake bounds and logarithmic linear-threshold learning algorithms. PhD thesis, University of California, Santa Cruz, CA.
- PEUQUET, D.J., 1988, Representations of geographic space. *Annals of the Association of American Geographers*, **78**, no. 3. 375–394.
- ROSENBLATT, F., 1962, Principles of neurodynamics. (New York: Spartan Books).
- SCHUURMAN, M., 1999, Critical GIS: Theorizing an emerging science. *Cartographica*, **36**, no. 4.
- SEDGEWICK, R., 1988, Algorithms (2nd Edn.). *Gravitation* (New York: Addison-Wesley).
- SUNDAY, P., Fast winding number inclusion of a point in a polygon. Available online at: http://softsurfer.com/Archive/algorithm_0103/algorithm_0103.htm (accessed 18 Nov 2004).
- VALIANT, L.G., 1984, A theory of the learnable. *CACM*, **27**, no. 11, 1134–1142.
- WALKER, R.J. and SNOEYINK, J. 2003, Practical point-in-polygon tests using CSG representations of polygons. Technical report TR-99-12, University of British Columbia.
- WISE, S., 2002, GIS basics. (New York: Taylor & Francis).