CHAPTER 10

The Logic of Activation Functions

R. J. WILLIAMS

The notion of logical computation, in some form or other, seems to provide a convenient language for describing the operation of many of the networks we seek to understand. Digital computers are built out of such constituents as AND and OR gates. Feature-detecting neurons in biological sensory systems are often idealized as signaling the presence or absence of their preferred features by becoming highly active or inactive, respectively. It seems a relatively simple extension of this concept to allow the activity of units in the network to range over some interval rather than over just two values; in this case the activity of a unit is regarded as signaling its *degree of confidence* that its preferred feature is present, rather than just the presence or absence of this feature. There are several ways one might attempt to formalize this degree-of-confidence notion. For example, if the activation values range over the closed unit interval [0,1], one might treat such an activation value as a conditional probability; alternatively, it might be viewed as a measure of truth in some unit-interval-valued logic, such as fuzzy logic (Zadeh, 1965).

There is at least one alternative to the notion of activation as degree of confidence which sometimes provides a convenient language for discussing the role of, for example, neural feature detectors in sensory systems. In this view, the activation of a unit encodes (within finite limits) the *amount* of its preferred feature present. This rival view seems advantageous particularly when the computation performed is described in the language of linear systems or linear signal processing;

examples of this are the concepts of spatial filtering and spatial Fourier analysis in the visual system and the concept of correlational processing in matrix models of associative memory (Kohonen, 1977). Chapter 9 describes the relevant mathematics for this approach, that of *linear algebra*.

This chapter explores some ideas motivated by the first of these two views of a PDP unit's computation (i.e., as some generalization of the notion of a Boolean function), but the approach is implicitly based on a very liberal interpretation of what this means. Essentially, the only structure assumed for the set of confidence values is that it be a totally ordered set with a Boolean interpretation of its endpoints. While a fully developed mathematical theory along these lines would deal with those properties that are invariant under any transformations preserving this structure, the ideas presented here do not go this far.

The specific program to be embarked upon here is probably best described as an exploratory interweaving of several threads, all related to these notions of logical computation and their potential applicability to the study of activation functions. First, the point of view is taken that any function whatsoever is a candidate for being an activation function. From this perspective, the traditional linear and thresholded linear activation functions may be viewed as very isolated examples from a much larger range of possibilities. Next, several ways to shrink this vast space of possibilities are suggested. One way proposed here is the imposition of a constraint based on the requirement that the notion of excitatory or inhibitory input be meaningful. Another way is the introduction of an equivalence relation on activation functions based on invariance under transformations preserving the logical and ordinal structure. Finally, an investigation is carried out to determine just where certain familiar types of activation functions, built out of the more traditional ingredients such as additive, subtractive, and multiplicative interactions among input values and weights, fit into this scheme. As a by-product of this development, some elementary results concerning implementation of Boolean functions via real-valued functions are also obtained.

This last aspect is closely related to what is historically one of the oldest formal approaches to the theory of neural computation, in which neurons are treated as Boolean devices. This approach was pioneered by McCulloch and Pitts (1943); an introductory overview of this whole subject can be found in the text by Glorioso and Colón Osorio (1980). An important influence on much of the work done in this area has been the perceptron research of Rosenblatt (1962; Minsky & Papert, 1969).

In what follows, several simplifying assumptions will be made. The first is that the range of values over which each input to a unit may vary is the same as the range of values over which the output of the unit (its *activation*) may vary. Another is that time may be ignored as a variable. The activation function of a unit will be taken to be a function that computes the output of the unit (at a fixed but unspecified time) as a function of its inputs (at a presumably slightly earlier but unspecified time). Thus, given a unit with n inputs whose activation values range over the set $\mathbf{A}$, the activation function α for this unit is just a function from $\mathbf{A}^n$ (the set of ordered n-tuples of elements of $\mathbf{A}$) to $\mathbf{A}$, denoted $\alpha : \mathbf{A}^n \rightarrow \mathbf{A}$.

In order to avoid cluttering the presentation, detailed proofs of the results have been omitted; in their place are short sketches indicating the key steps. A more rigorous and abstract formulation of the basic concepts introduced here, along with detailed proofs of the results, may be found in Williams (1983).

EXAMPLES OF ACTIVATION RULES

The following are some examples of activation functions from which models have been constructed.

Example 1. $\mathbf{A} = \{0,1\}$ (the two-point set), $\alpha = f \circ g$, where g is linear into $\mathbb{R}$ and $f : \mathbb{R} \rightarrow \mathbf{A}$ is a thresholding function. (The operator $\circ$ between two functions here denotes composition in a right-to-left manner.) A unit using this activation function is called a *threshold logic unit* or a *linear threshold unit* and is the basis of the simple perceptron (Rosenblatt, 1962; Minsky & Papert, 1969).

Example 2. $\mathbf{A} = \mathbb{R}$, α linear (Kohonen, 1977).

Example 3. $\mathbf{A} = \mathbf{I}$ (the closed unit interval $[0,1]$), $\alpha = f \circ g$, where g is linear into $\mathbb{R}$ and f is nondecreasing into $\mathbf{I}$. This is a commonly used variant of Example 1. Let us call this a *quasi-linear activation function*. The function f is sometimes called a *squashing function* for obvious reasons.

Example 4. $\mathbf{A} = \mathbf{I}$, $\alpha = f \circ g$, where f is nondecreasing into $\mathbf{I}$ and g is a multilinear function into $\mathbb{R}$ of the form

$$g(x_1, \ldots, x_n) = x_1x_2 + x_3x_4 + \cdots + x_{n-1}x_n$$

(where n is assumed to be even). Such an activation function is suggested by Hinton (1981b). Note that this is similar to Example 3 except that the coefficients have now become explicit inputs. This type of activation function will be called a *gating activation function* because the odd-numbered inputs gate the even-numbered ones (and vice-versa).

Example 5. $\mathbf{A} = \mathbf{I}$, $\alpha = f \circ g$, where f is nondecreasing into $\mathbf{I}$ and g is an arbitrary multilinear function into $\mathbb{R}$. That is, g is of the form

$$g(x_1, \ldots, x_n) = \sum_{S_j \in \mathbf{P}} w_j \prod_{i \in S_j} x_i \ ,$$

where $\mathbf{P}$ is the power set (i.e., set of subsets) of $\{1, \ldots, n\}$. Such an activation function is called a *sigma-pi activation function*, with the coefficients w_j being called *weights*. (We might also call this a *quasi-multilinear activation function* to emphasize its relationship to Example 3.) Note that Examples 3 and 4 are just special cases of this activation function.

THE MAIN CONCEPTS

Henceforth in this chapter the set of activation values will be assumed to be the closed unit interval [0,1], denoted $\mathbf{I}$. An *activation function* is then simply a function $\alpha : \mathbf{I}^n \rightarrow \mathbf{I}$. It will be convenient to identify $0 \in \mathbf{I}$ with the Boolean value *false* and $1 \in \mathbf{I}$ with the Boolean value *true*.

Now we introduce a key concept of this chapter by considering the extension of the familiar notion of a monotonic function to the multidimensional case in two different ways. In order to get a feeling for the precise definitions to be given below, first consider what it means for an input to a unit to have an excitatory influence on the output of that unit. Such an input must have the property that an increase in its value must result in an increase in the output of the unit, as long as all other inputs are held constant. Furthermore, this should be true regardless of the values of the other inputs. A similar property should hold for an inhibitory input, where the output of the unit must decrease as the value of the input is increased in this case. This is the basic idea behind the notion of *uniform monotonicity*, as defined below. The weaker notion of *monotonicity-in-context* corresponds to the situation in which an input may be sometimes excitatory and sometimes inhibitory, depending on the values taken on by the other inputs.

Now we make these concepts rigorous. Let $\alpha : \mathbf{I}^n \rightarrow \mathbf{I}$. Pick one of the coordinates, say the kth, and fix all coordinates but this one, which is allowed to vary. This defines a function of a single variable which is parameterized by the remaining coordinates. Such a function is called a *section* of the original function α along the kth coordinate. Note that there is one such section along the kth coordinate for each possible combination of values for the remaining $n-1$ coordinates. Now make the following definitions:[1]

1. α is *monotonic-in-context along the* kth *coordinate* if all its sections along the kth coordinate are monotonic.

2. α is *uniformly monotonic in the* kth *coordinate* if all sections along the kth coordinate are monotonic and have the same sense (i.e., all are nondecreasing or all are nonincreasing).

3. α is *monotonic-in-context* if it is monotonic-in-context along all its coordinates.

4. α is *uniformly monotonic* if it is uniformly monotonic along all its coordinates.

One special case of a uniformly monotonic function is a *uniformly nondecreasing* function, which has the property that all its sections along all coordinates are nondecreasing. This special case will be used later.

Note that if α is uniformly monotonic then it is monotonic-in-context, but the converse need not be true, unless α is a function of a single variable, in which case both definitions collapse onto the usual notion of monotonicity. The key distinction between uniformly monotonic and monotonic-in-context is that the sense of monotonicity of the sections of α along the kth coordinate must be fixed for each k in order for α to be uniformly monotonic.

It is important to emphasize the significance of these monotonicity concepts for activation functions. An activation function is uniformly monotonic if and only if each input may be classified as solely excitatory or solely inhibitory, independently of the values actually taken on by any other inputs. Thus the usual sense of excitatory or inhibitory input to a unit is meaningful exactly when the unit's activation function is uniformly monotonic. If a unit's activation function is monotonic-in-context, then it may not be possible to categorize its inputs as solely excitatory or solely inhibitory, but the following may be a useful conceptualization of such a unit's operation: Certain inputs to the unit are

[1] The reader should be warned that the names introduced here for these concepts are not standard; these terms were chosen because it was felt that they helped to clarify the important distinctions being made in the current context.

used to set the context for the computation of its output as a function of the remaining inputs, and each input in this latter group has purely excitatory or purely inhibitory effect on the unit's output in this particular context. Whether this turns out to be a useful way to view the monotonic-in-context activation function and its possible role in activation models will not be explored here. The main reason for introducing the concept is simply that it appears to be the strongest variant on monotonicity satisfied by any activation function capable of computing an arbitrary Boolean function (such as the multilinear and sigma-pi activation functions, as will be seen later).

In order to capture the notion of an activation function being simply an extension of a Boolean function, define an activation function $\alpha : \mathbf{I}^n \rightarrow \mathbf{I}$ to be *Boolean-like* if $\alpha(x_1, \ldots, x_n) = 0$ or 1 whenever all the x_i are 0 or 1. In other words, an activation function is Boolean-like if and only if it can be viewed as a Boolean function when restricted to the vertices of $\mathbf{I}^n$. It is also useful to say that such an activation function *realizes* the Boolean function obtained by restricting to vertices.

In order to capture the notion of two activation functions agreeing for Boolean input values, define two activation functions $\alpha_1, \alpha_2 : \mathbf{I}^n \rightarrow \mathbf{I}$ to be *vertex-equivalent* if $\alpha_1(x_1, \ldots, x_n) = \alpha_2(x_1, \ldots, x_n)$ whenever all the x_i are 0 or 1. In other words, two activation functions are vertex-equivalent if and only if they agree on the vertices of $\mathbf{I}^n$. It is clear that vertex-equivalence is indeed an equivalence relation.

The reason for introducing this notion is the suggestion that there may be a certain interchangeability between different activation functions that are vertex-equivalent, in that the logic of a unit's computation might be considered to reside solely in what it does when all input lines are set to their extreme values (corresponding to *true* or *false*). If two vertex-equivalent activation functions are additionally monotonic-in-context and continuous, then an even stronger case can be made for their interchangeability in certain models, but these ideas will not be pursued here.

ILLUSTRATION OF THESE CONCEPTS

A number of examples of activation functions $\alpha : \mathbf{I}^2 \rightarrow \mathbf{I}$ will now be presented to clarify the definitions given in the previous section. The figure corresponding to each example consists of three different graphical representations for that particular function: (a) a three-dimensional plot of $\alpha(x_1, x_2)$ versus (x_1, x_2); (b) a contour plot showing at which points (x_1, x_2) certain values of $\alpha(x_1, x_2)$ are attained; and (c) various sections of α along x_1 superimposed on a single two-dimensional plot. The activation function being displayed in each figure is defined in the caption of that figure.

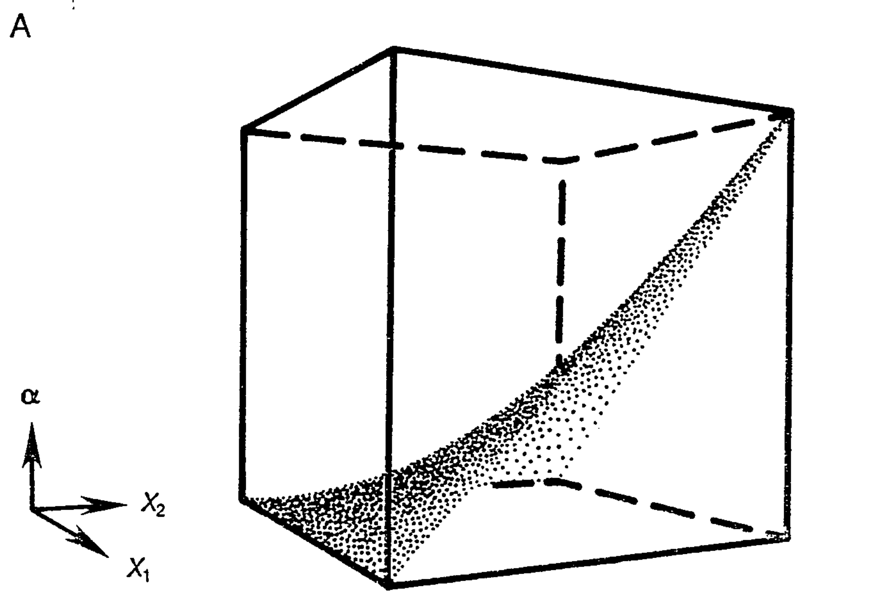
Figures 1, 2, and 3 show three different activation functions that realize the Boolean AND function at the vertices, while Figures 4, 5, and 6 show three different activation functions realizing the Boolean OR function at the vertices. These functions are all Boolean-like and uniformly monotonic.

Figure 7 shows a realization of the Boolean XOR (*exclusive or*) function. The activation function depicted is Boolean-like and monotonic-in-context, but not uniformly monotonic. In fact, no realization of XOR can be uniformly monotonic. Figure 8 shows an activation function that is uniformly monotonic but not Boolean-like. Its restriction to vertices thus does not have a straightforward Boolean interpretation; this activation function might be viewed as a unit-interval confidence measure based on the number of active inputs. Finally, Figure 9 shows a rather pathological example of an activation function. It is Boolean-like and vertex-equivalent to the constant function 1, but intuition suggests that any unit in a PDP network which performs such a computation will behave very differently from one which puts out the constant value 1. This essential difference in behavior is formalized here in terms of the observation that such an activation function fails to be monotonic-in-context while the constant function 1 is uniformly monotonic.

SOME RESULTS

Before stating the main results, it will be helpful to define two functions, the first of which maps vertices of $\mathbf{I}^n$ to Boolean expressions in formal variables $X_1, \ldots, X_n$, and the second of which maps such Boolean expressions to real algebraic expressions in formal variables $x_1, \ldots, x_n$. In our notation for Boolean expressions we will use "+" to denote disjunction and juxtaposition to denote conjunction but it will always be clear from the context whether Boolean or real operations are intended.

The mapping from vertices to Boolean expressions is defined by assigning to a vertex $(v_1, \ldots, v_n)$ the conjunction in which each X_i appears once, with the negation operator applied to X_i if and only if $v_i = 0$. For example, applying this function to the vertex (0,1,1,0) of $\mathbf{I}^4$ yields the expression $\overline{X}_1 X_2 X_3 \overline{X}_4$.



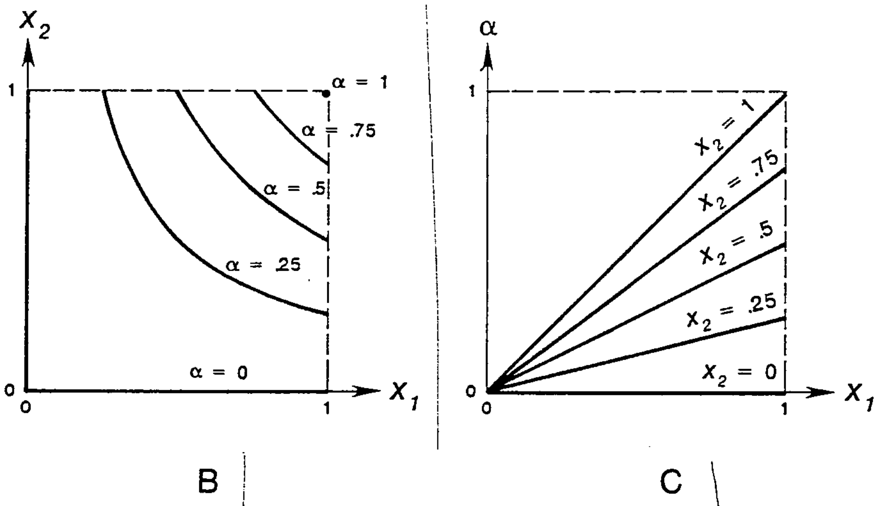


FIGURE 1. $\alpha(x_1,x_2) = x_1x_2$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that each section along x_1 is a linear function with nonnegative slope; by symmetry the same is true of each section along x_2. Thus this function is uniformly nondecreasing.

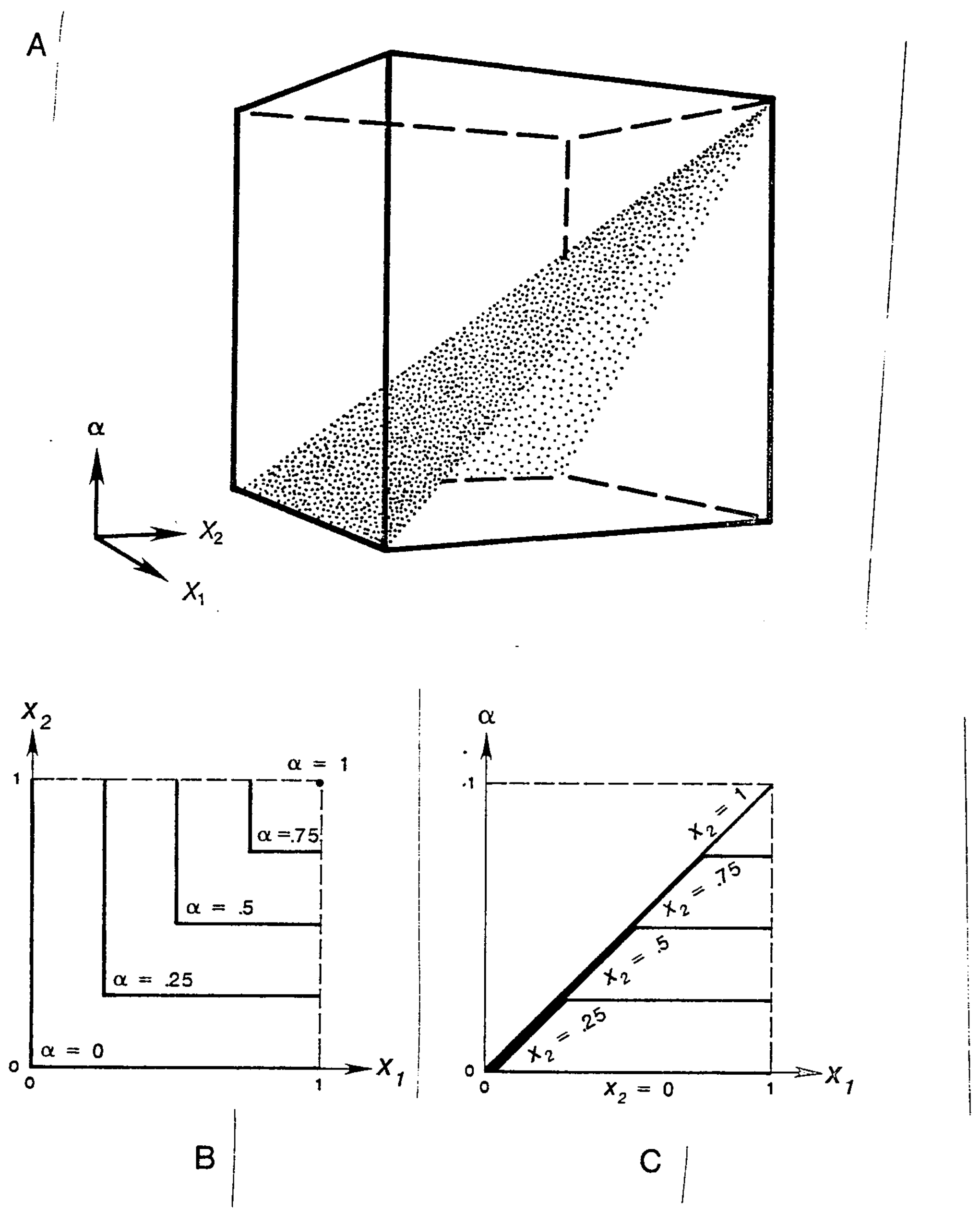


FIGURE 2. $\alpha(x_1,x_2) = \min(x_1,x_2)$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that the three-dimensional plot of this function consists of two planar surfaces. Evidently, each section along x_1 is a nondecreasing function; by symmetry the same is true of each section along x_2. Thus this function is uniformly nondecreasing.

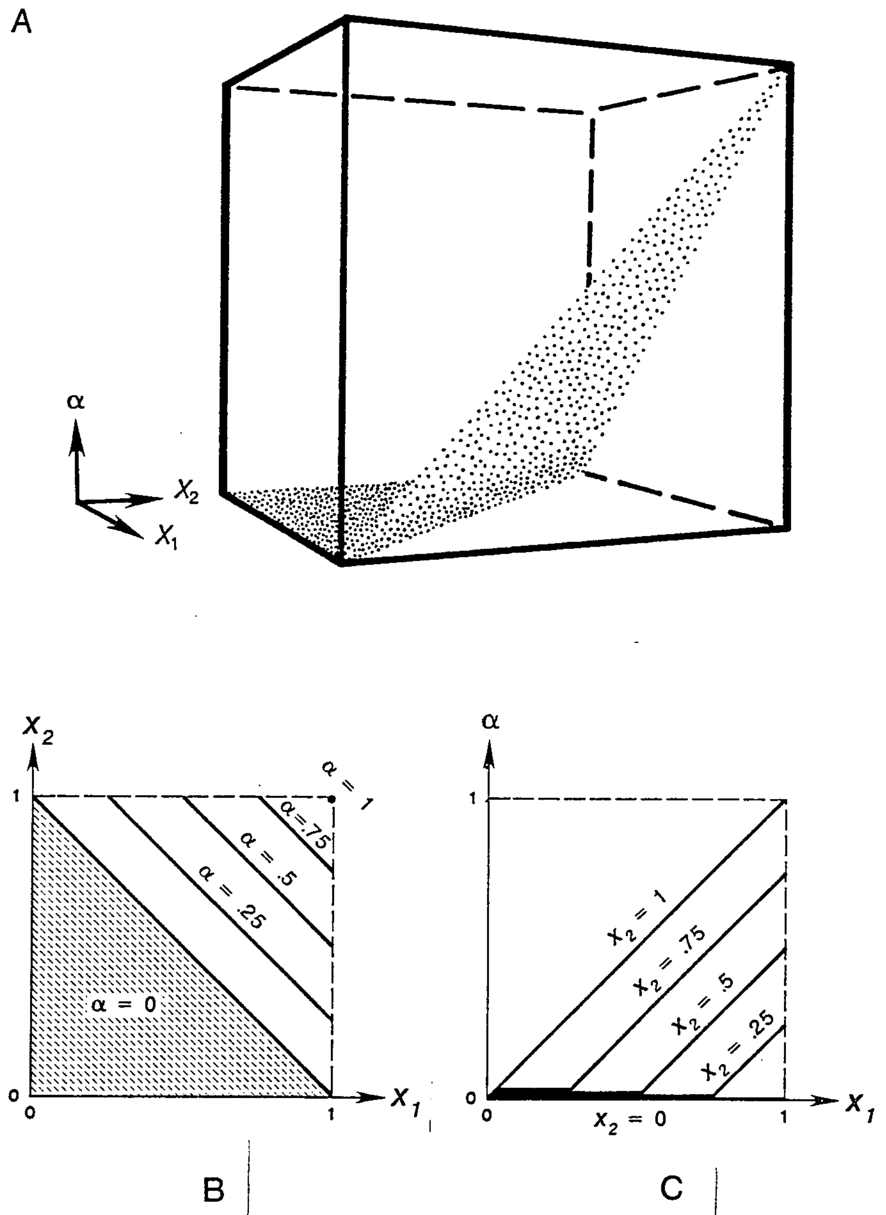


FIGURE 3. $\alpha(x_1,x_2) = \max(0, x_1+x_2-1)$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that the three-dimensional plot of this function consists of two planar surfaces. Clearly, each section along x_1 is a nondecreasing function; by symmetry the same is true of each section along x_2. Thus this function is uniformly nondecreasing.

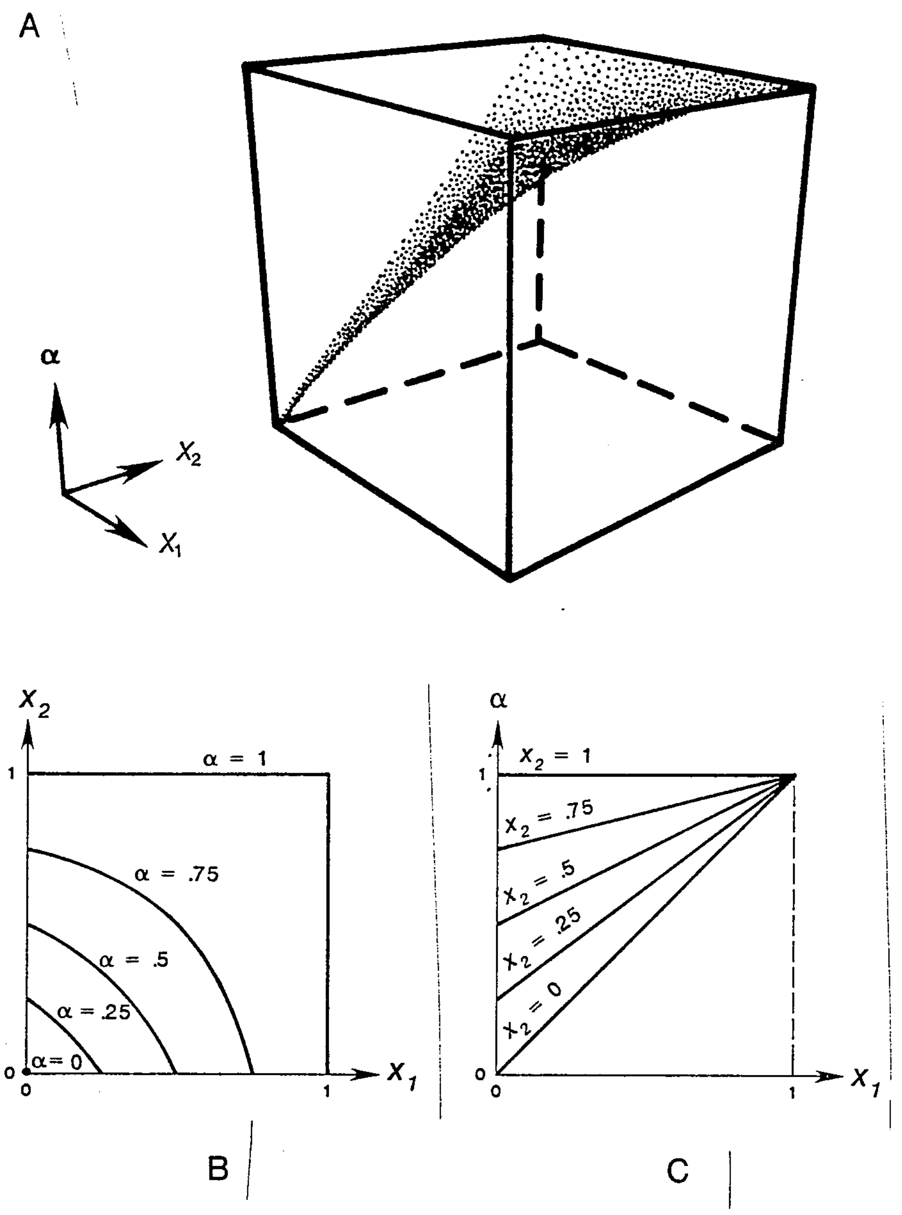


FIGURE 4. $\alpha(x_1,x_2) = x_1+x_2-x_1x_2$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that each section along x_1 is a linear function with nonnegative slope; by symmetry the same is true of each section along x_2. Thus this function is uniformly nondecreasing.

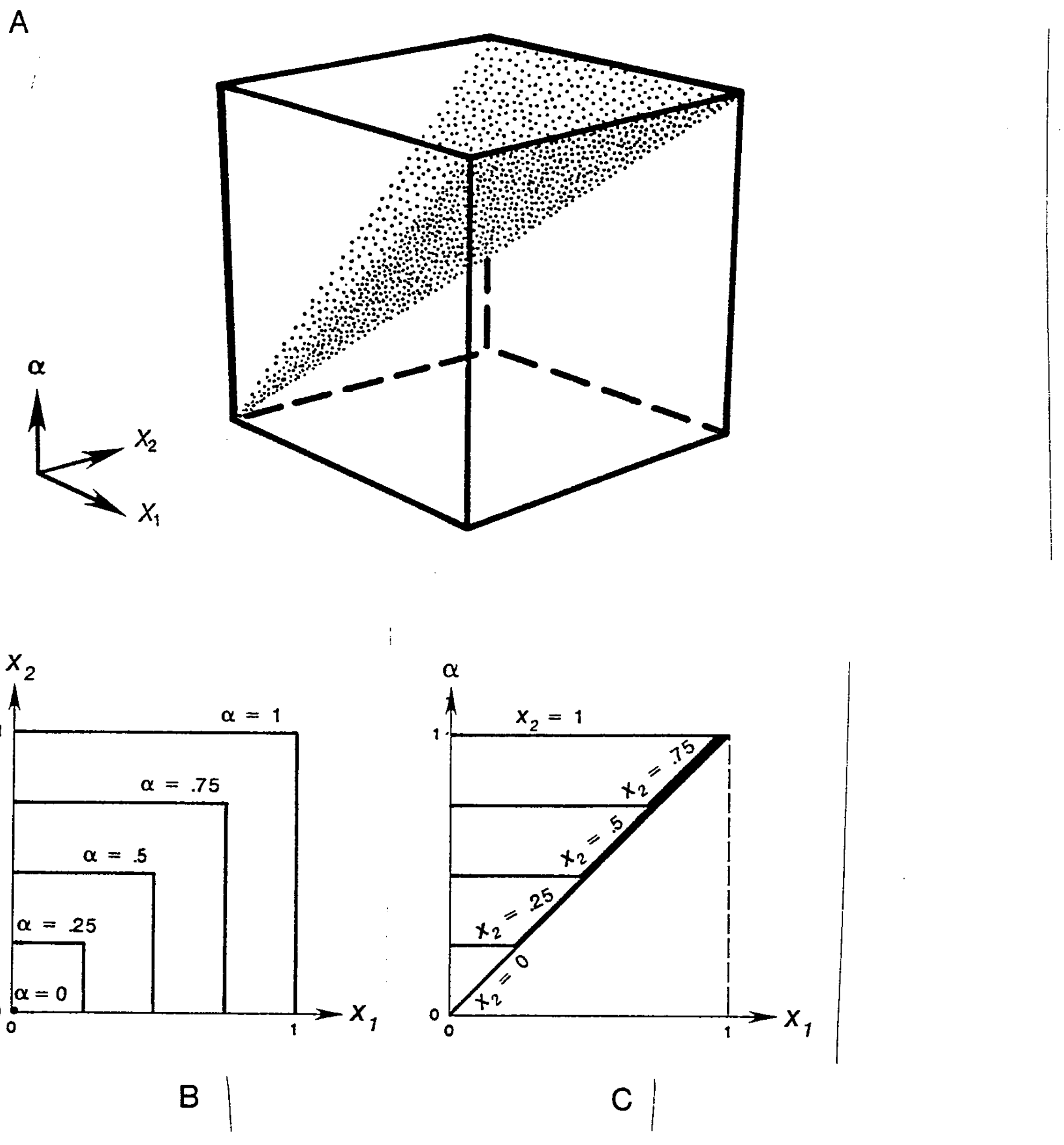


FIGURE 5. $\alpha(x_1,x_2) = \max(x_1,x_2)$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that the three-dimensional plot of this function consists of two planar surfaces. Note also that each section along x_1 is a nondecreasing function; by symmetry the same is true of each section along x_2. Thus this function is uniformly nondecreasing.

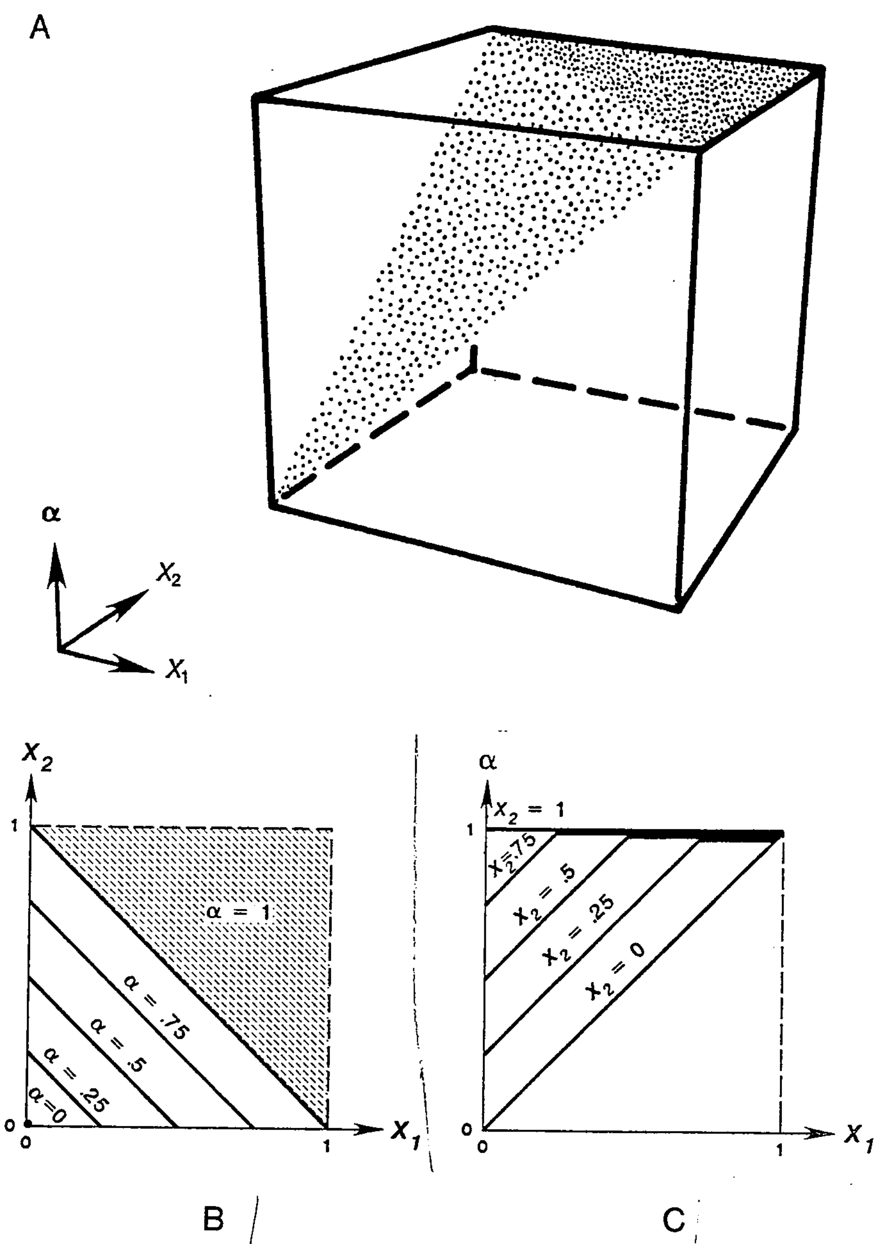
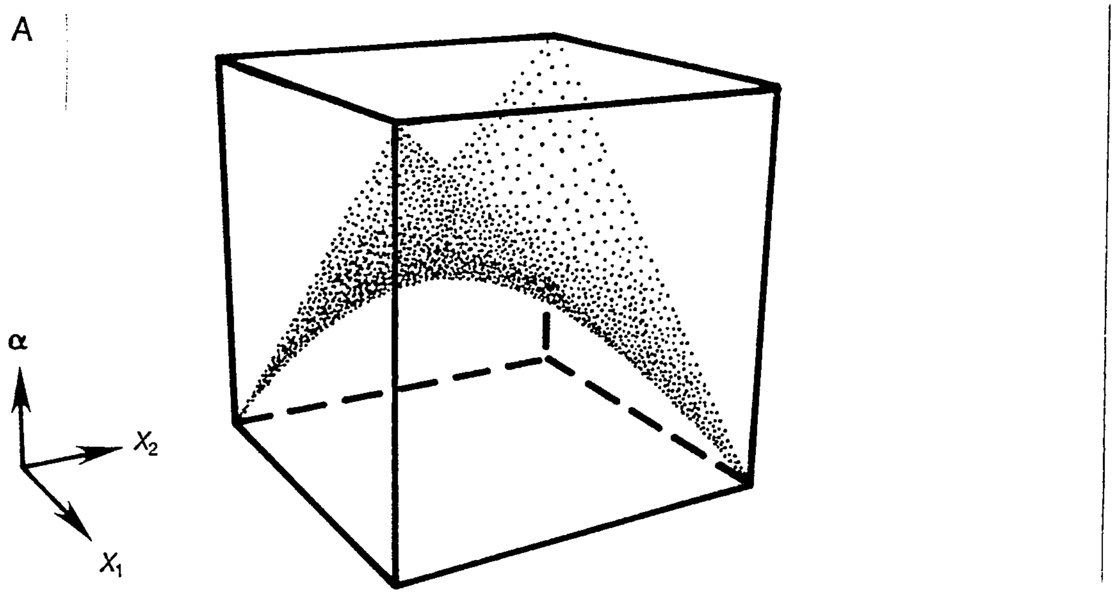


FIGURE 6. $\alpha(x_1,x_2) = \min(1,x_1+x_2)$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that the three-dimensional plot of this function consists of two planar surfaces. Evidently, each section along x_1 is a nondecreasing function; by symmetry the same is true of each section along x_2. Thus this function is uniformly nondecreasing.



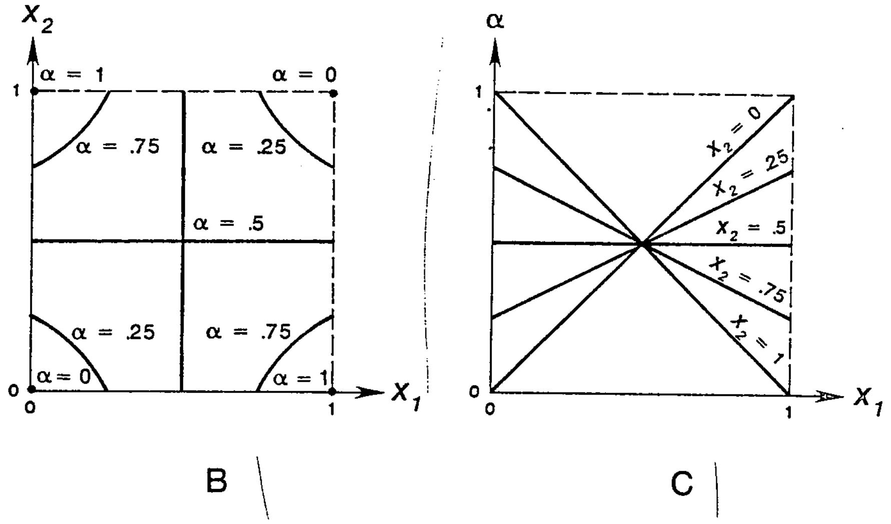


FIGURE 7. $\alpha(x_1,x_2) = x_1+x_2-2x_1x_2$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note the saddle shape of the three-dimensional plot of this function. Also note that the sections along x_1 are linear functions with slopes ranging from 1 to -1; by symmetry the same is true of the sections along x_2. Thus this function is monotonic-in-context but not uniformly monotonic.

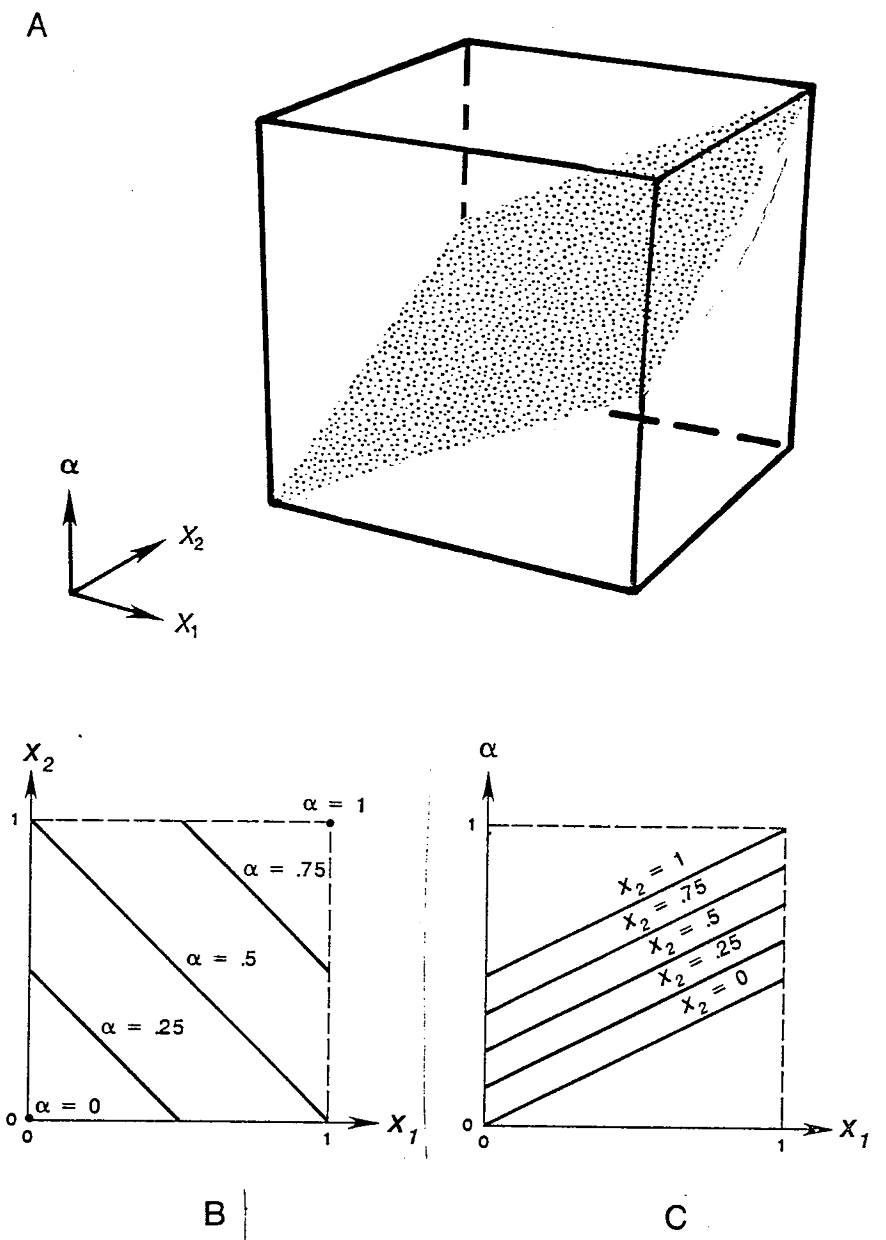


FIGURE 8. $\alpha(x_1, x_2) = \frac{1}{2}(x_1+x_2)$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that the three-dimensional plot of this function consists of a single planar surface. Each section along x_1 is a linear function with slope ½, as is each section along x_2, by symmetry. Thus this function is uniformly nondecreasing.

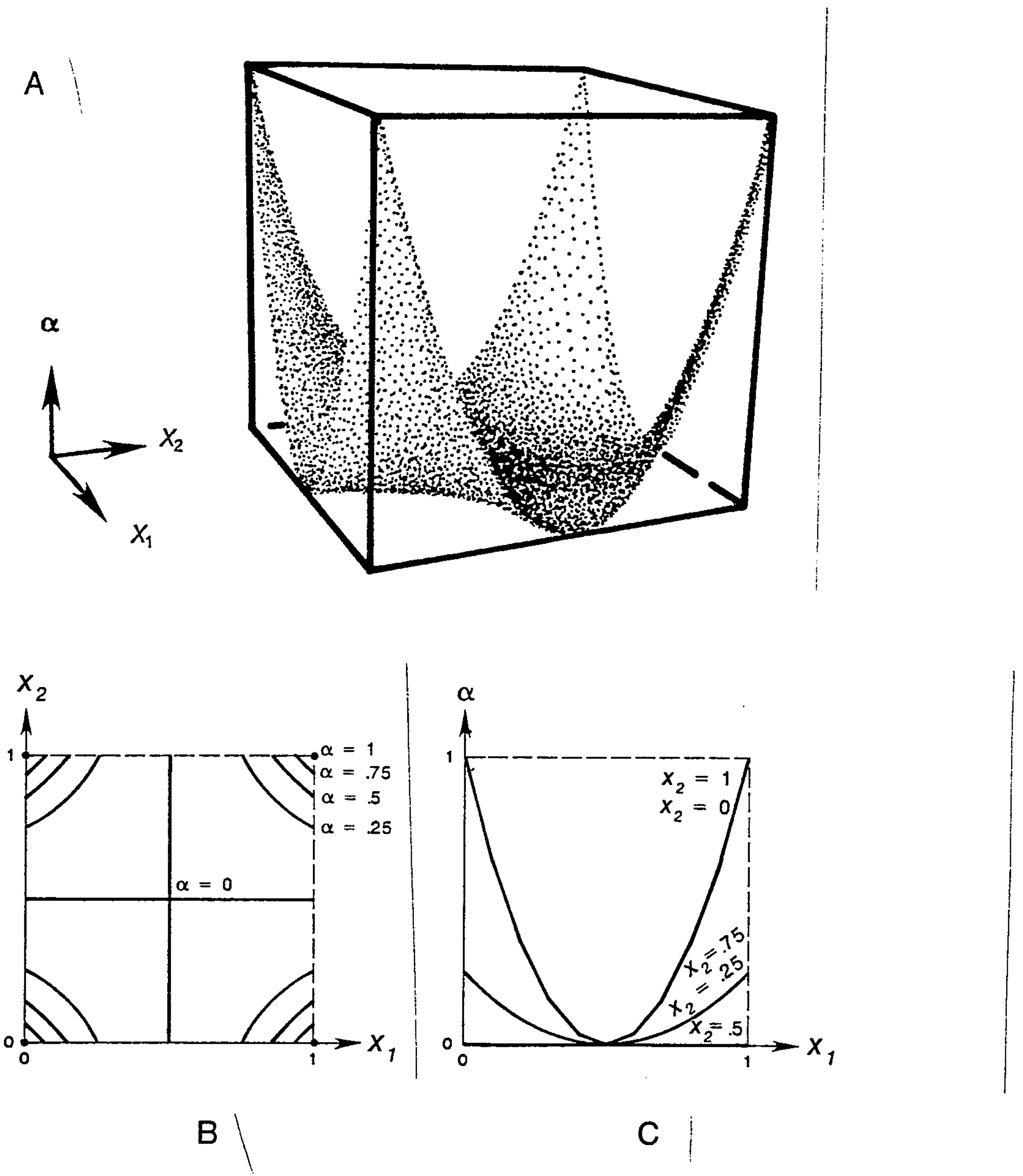


FIGURE 9. $\alpha(x_1,x_2) = (2x_1-1)^2(2x_2-1)^2$. *A*: Three-dimensional plot. The cube is bounded by the planes where each coordinate is 0 or 1. *B*: Contour plot. *C*: Some sections along x_1. Note that the sections along x_1 are parabolas of varying widths. Evidently, this function is not monotonic-in-context since, for example, when $x_2 = 0$, α first decreases and then increases as x_1 increases.

The mapping from Boolean expressions to real algebraic expressions is defined by replacing:

1. *True* by 1.
2. *False* by 0.
3. The disjunction operator by addition.
4. The conjunction operator by multiplication.
5. The negation operator by subtraction from 1.
6. X_i by x_i, for each i.

For example, applying this function to the Boolean expression $X_1\overline{X}_2+\overline{X}_1$ yields the real expression $x_1(1-x_2)+(1-x_1)$. It should be emphasized that this is a function defined only on formal expressions; two expressions that are equivalent under Boolean algebra will not, in general, be mapped to the same real algebraic expression or even equivalent real algebraic expressions. In other words, it is not a mapping from Boolean functions to real functions.

A standard result from Boolean algebra is that any Boolean function may be expressed in a certain canonical form, called the *disjunctive normal form*. A simple prescription for this form is as follows: Form a disjunction of terms, each of which is the result of applying the vertices-to-Boolean-expressions function described above to those vertices of $\mathbf{I}^n$ for which the function takes on the value *true*. For example, the disjunctive normal form for the Boolean function $\beta(X_1,X_2)=X_1+X_2$ is $X_1\overline{X}_2+\overline{X}_1X_2+X_1X_2$.

A closely related result for multilinear functions is the following:

Lemma. For any function assigning arbitrary real numbers to the vertices of $\mathbf{I}^n$ there is a unique multilinear function agreeing with the given function on these vertices.

This function is formed in a manner generalizing the prescription given above for the disjunctive normal form: For each vertex of $\mathbf{I}^n$, form the corresponding Boolean conjunct; then apply the other function described above to turn each of these conjuncts into a real expression; finally, form the sum of these individual expressions with each one weighted by the value of the given function at the corresponding vertex. It will be convenient to dub the result the *vertex normal form* for the given function. For example, the vertex normal form for a multilinear function α of two variables is

$$\alpha(x_1,x_2)=\alpha(0,0)(1-x_1)(1-x_2)+\alpha(0,1)(1-x_1)x_2$$
$$+\alpha(1,0)x_1(1-x_2)+\alpha(1,1)x_1x_2.$$

This lemma has the following immediate consequence:

Theorem 1. Given any Boolean function, there is a unique multilinear activation function realizing it.

In contrast, not every Boolean function can be realized by a quasilinear activation function. Those Boolean functions that can be so realized are called *linearly separable.* It is easily shown that any linearly separable Boolean function is necessarily uniformly monotonic, but the converse is not true. A simple example of a function that is not linearly separable is the XOR function $\beta_1(X_1,X_2) = X_1\overline{X}_2 + \overline{X}_1X_2$. The easiest way to see that it is not linearly separable is to observe that it is not uniformly monotonic. An example of a function that is uniformly monotonic but not linearly separable is

$$\beta_2(X_1,X_2,X_3,X_4) = X_1X_2 + X_3X_4.$$

Our next result, also a consequence of the lemma, shows that the very general class of all activation functions may be represented up to vertex-equivalence by the narrower class of multilinear activation functions.

Theorem 2. Every activation function is vertex-equivalent to a unique multilinear activation function.

The next result suggests that monotonicity-in-context is enjoyed by a fairly wide variety of activation functions.

Theorem 3. Every sigma-pi activation function is monotonic-in-context.

This is an easy consequence of three facts: (a) that a multilinear function is linear in each variable when the other variables are held constant; (b) that a linear function is monotonic; and (c) that the composition of monotonic functions is monotonic.

The following result characterizes uniform monotonicity for multilinear activation functions.

Theorem 4. A multilinear activation function is uniformly monotonic if and only if its restriction to vertices is uniformly monotonic.

The key step in the proof of this result is the observation that a multilinear function may be built up inductively through linear interpolation, starting with the values at the vertices. This follows from the fact that a multilinear function is linear in each variable when the other variables are held constant. The remainder of the proof consists of verifying that each step of this inductive construction preserves uniform monotonicity. This result may be extended to the sigma-pi case as well, under certain mild restrictions, using the fact that a strictly increasing function has a monotonic inverse.

Corollary. Let $\alpha = f \circ g$ be a sigma-pi activation function, where g is multilinear and f is a squashing function. If f is strictly increasing, then α is uniformly monotonic if and only if its restriction to vertices is uniformly monotonic.

The results presented up to this point would seem to suggest that the class of multilinear activation functions provides us with sufficient power that we need not consider the more general class of sigma-pi activation functions. However, from the standpoint of uniform monotonicity, there may be some drawbacks in restricting ourselves to multilinear activation functions. One such potential drawback is that a uniformly nondecreasing multilinear activation function may have some negative weights. For example, the Boolean function $\beta(X_1,X_2) = X_1 + X_2$ corresponds, by Theorem 1, to the multilinear activation function $\alpha(x_1,x_2) = x_1 + x_2 - x_1x_2$, which requires a negative weight even though it is uniformly nondecreasing. But what if a more general sigma-pi activation function were to be used? Is there a sigma-pi realization of this same Boolean function for which all weights are nonnegative? Of course there is in this case: The sigma-pi activation function $\alpha(x_1,x_2) = \min(x_1 + x_2, 1)$ is one such realization; many others could be devised. (These two realizations of the OR function are displayed in Figures 4 and 6.) It seems reasonable to suspect that the following is true:

Conjecture. Every uniformly nondecreasing activation function is vertex-equivalent to a sigma-pi activation function with nonnegative weights.

Note that any sigma-pi activation function with nonnegative weights is certainly uniformly nondecreasing. The conjecture is that the converse is true (up to vertex equivalence). Under the assumption that the uniformly nondecreasing activation function is Boolean-like (as in the preceding example), the conjecture is indeed valid, as the following theorem shows. In fact, the conclusion may be made even stronger in this case.

Theorem 5. Every uniformly nondecreasing Boolean-like activation function is vertex-equivalent to a sigma-pi activation function whose weights are all 0 or 1.

The essential step in the proof of this result is showing that any uniformly nondecreasing Boolean function may be expressed as a disjunction of conjunctions containing no negated factors. Once such an expression is available, the desired sigma-pi activation function is obtained by converting this Boolean expression to a real expression and then composing this with the function $f(z) = \min(z,1)$.

This theorem may be generalized to cover arbitrary senses of uniform monotonicity by running any inputs for which the activation function is nonincreasing through the "inverter" $f(x) = 1-x$. Thus the general class of all uniformly monotonic Boolean-like activation functions may be represented up to vertex-equivalence by a narrower class of sigma-pi activation functions of a certain form.

It is instructive to contrast the sigma-pi activation functions which result from applying Theorems 1 and 5 to a particular uniformly monotonic activation function. Consider the Boolean function of six variables $\beta(X_1,X_2,X_3,X_4,X_5,X_6) = X_1X_2 + X_3X_4 + X_5X_6$. Theorem 1 realizes this using the vertex normal form, which, after simplification, becomes

$$\begin{aligned}\alpha_1(x_1,x_2,x_3,x_4,x_5,x_6) = {} & x_1x_2 + x_3x_4 + x_5x_6 \\ & - x_1x_2x_3x_4 - x_1x_2x_5x_6 - x_3x_4x_5x_6 \\ & + x_1x_2x_3x_4x_5x_6 .\end{aligned}$$

In contrast, Theorem 5 implies a realization of this same function by the gating activation function

$$\alpha_2(x_1,x_2,x_3,x_4,x_5,x_6) = \min(x_1x_2 + x_3x_4 + x_5x_6 , 1).$$

CONCLUSION

As suggested in the introduction, the ideas and results presented here represent an exploratory set of concepts intended to help in understanding PDP networks. There is a clear need for a general language and set of concepts for describing and understanding PDP computation, both at the local, individual unit level, as explored here, and at the level of whole networks. (In fact, the greatest need is for a means of describing and understanding the relationship between computation at these two levels.) Whether the ideas contained in this chapter can extend naturally to become a useful framework for understanding the behavior of whole networks is difficult to foresee. One way that this gap between local and global computation might be bridged is by dealing with questions of *learning* in such networks. The goal of learning is generally to cause the network to have a particular global behavior, but the learning should be implemented locally. An example of how the requirement that the network be capable of learning might interact with the ideas explored here can be found by considering the recently discovered *back-propagation* learning algorithm, described in Chapter 8. To be able to apply such a learning algorithm requires imposing the constraint on activation functions that they be differentiable, a property not satisfied by all the examples considered here. As our understanding of learning in PDP networks progresses, we may find still further restrictions useful or even necessary.

ACKNOWLEDGMENTS

This research was supported by a grant to David Zipser from the System Development Foundation. I am also grateful to James McClelland and David Zipser for their many helpful comments and suggestions.