RECORD, Volume 26, No. 2*

San Diego Spring Meeting
June 22–23, 2000

Session 26PD
Hot Technologies

Track: Computer Science

Moderator: MARK D.J. EVANS
Panelists: FRED A. WATKINS†
JACEK ZURADA‡
Recorder: MARK D.J. EVANS

Summary: The panel covers:

- *The basic concepts underlying fuzzy logic*
- *Current applications to risk classification and pricing*
- *Possible future uses of fuzzy logic*
- *The basic concepts underlying neural networks*
- *Current applications of neural networks in life insurance*
- *Possible future uses of neural networks*

Mr. Mark D.J. Evans: The first speaker will be Jacek Zurada, who will be talking about neural networks. Our second speaker will be Fred Watkins, who will be talking about fuzzy logic.

Jacek Zurada is the professor of electrical engineering at the University of Louisville, where he holds the position of Fife Alumni Professor. He received his Ph.D. in electrical engineering from the Gdansk University of Technology in Poland. He is currently editor-in-chief of the *IEEE Transactions of Neural Networks*. He is a fellow of the Institute of Electrical and Electronics Engineers (IEEE), and he has also written books on neural networks.

Dr. Jacek Zurada: I am very thankful to have the opportunity to talk about neural networks to the community of actuaries. I have heard much about actuaries, and I have always had a high opinion of them. I will give you a perspective of neural networks. I'll discuss how they can be used in a very versatile way, and I will try to steer away from mathematics.

Many of you have heard about neural networks. Neural networks are parallel computing machines that are organized in cells called neurons, which are densely

*Copyright © 2001, Society of Actuaries

†Dr. Watkins, not a member of the sponsoring organizations, is with HyperLogic Corporation in Escondido, CA.

‡Dr. Zurada, not a member of the sponsoring organizations, is S.T. Fife Alumni Professor, IEEE Fellow of the University of Louisville in Louisville, KY.

Note: The charts referred to in the text can be found at the end of the manuscript.

interconnected. Each of these cells performs relatively simple operations—color product computation and nonlinear squashing, nonlinear mapping of the scalar product into a real variable. A real variable between –1 and 1 is the output of a neuron. We are mapping the space of real numbers through one single neuron; however, dimensionality of the input vector can be mapping into an open interval from –1 to 1. However, out of this rather naïve concept comes tremendous opportunity for building a family of models, identifying stationary processes, identifying time discrete processes and time series, computing probabilities and clustering, and many other things. I am going to go through these things gradually and show a few applications. Some of them are not very relevant for actuaries, but some of them are, especially for an expert system.

There are no algorithms behind neural networks except for an algorithm for learning based on data that represents a certain problem. One might think that neural networks, as we know them today, are very biologically oriented but, in fact, they are not. This is a class of mathematical models that builds on statistics, mathematics, and a little bit on neural physiology. It is probably more on statistics than anything else. They can provide very efficient solutions of the same or better quality as other very specialized techniques for signal processing, for expert system building, and so on. They are widely applicable especially in pattern recognition.

We typically don't use neuron networks in the form of specialized hardware or microelectronic chips, because they can be very expensive. They are available, but overly costly. We would rather simulate them on general-purpose computers. There's a lot of software out there in both the business field and the public domain that supports neural network learning. How do neural networks differ from computers? For neural network development, we need a set of representative examples followed by training. The computation takes a form of parallel collective processing and the whole knowledge in the neural network is stored in weights that are very imprecise. So there is no specialized algorithm, no need for programming, and no need for sequential computation.

For us engineers, it is very essential that neural networks handle very difficult tasks, like building an autonomous driver. That was done was by Pomeroy. This neural network collects visual images from the camera that is sitting on top of a van, and this information is processed and issues a signal for the driver to turn left, right, or drive straight. There's one example. Another example that also merges the problem of control and vision is balancing a pole. A neural network is very simple in fact. We can issue a balancing force that would make the pole stand vertically, and the whole system would be in balance by merging visual fields that can be captured from the camera. Once neural networks are able to solve such problems, which are rather difficult, combining vision, recognition, and control, they can solve simpler problems like handling data and making sense out of them.

Why do we have interest in using neural networks for data analysis? We use them in order to develop profiles and ratings scores to reveal hidden information such as risk and fraud and to build if-then rules from certain acquired knowledge. We are often interested in getting an explanation of the reason for the decision behind certain computations. We support them with if-then rules, which can be also

retrieved from neural networks. Among many application areas are medicine, repairs, maintenance, diagnostics, agriculture, chemical analysis, and pattern recognition. A large sector of applications is in the financial area in banking. I don't have the direct knowledge about it, but I have seen reports about financial management using neural networks for risk evaluation, investment planning, and portfolio management. It has also been helpful in the loans and insurance business for credit assessment, loan evaluation, and evaluation of profitability of insurance. Towards the end of my presentation, I'm going to get closer to how to approach certain problems that might be of special interest to this audience.

What do neural networks do in laymen's terms? They can associate a certain object with an input. One very important function or mission of neural networks is to classify input as belonging to one of the classes that are memorized and stored within it. Here a binary vector is usually available at the output of a neural network, which indicates a class number and recognition of classification when the input is distorted. One very important function of a neural network is to compute output vector *O* based on input vector *X*. It is essentially a computer of a vector variable based on an input vector variable. The beauty of that very basic model is that these *Xs* and *Os*, (entries of these vectors can be either real or binary numbers) can be uncertain or missing information. Neural networks are, to a degree, free-wheeling models, that are very nonrigorous about the form of these entries of input vectors, both as input and output. That's why we can put anything out here on the input and output. Vector *O* consists of "*M*" components and computes, for instance, classification or the posterior probabilities that we are dealing with in class *C* when *X* is present at the input, which is already a very nice task of performing a classification.

With multivariable approximations, we have "M" approximations of "N" variable functions. We develop them based upon examples in a situation where the model O of X does not exist. I grabbed one or two books at this meeting, and I found out that some authors were trying to find out formulas or a power series to model certain economical phenomenon and insurance phenomenon. The basic product of a neural network is that you do not need any analytical facility or any analytical description of a model. You need data that represents input and output vectors to develop the model, and this model will actually be described with no formula but it will have certain algebraic formulas. These can be used to compute O1, O2, through OM. If we enrich this model into Xs, which are spread in time, like X1 is at the time 1 week, and X2 is at the time 2 weeks, we can model time sequences as well in that general arrangement. So we can build predictors or models of input/output sequences, which is essentially solving difference equations. Neural networks can cluster information, compress data, and can finally extract features, which is a very important and very unique task. It allows us to see some patterns in the big cloud of data.

Where does the computing power of neural networks come from? As I mentioned, learning is responsible for storing the information in the network and capturing the relationship that is present in data. Let's discuss the computation of output vector *O* in response to input vector *X*. We call it regal but regal is a very trivial phase.

The real trick in neural networks is in organizing the learning. As complicated as learning can be, it is philosophically very simple because the learning can take one of two forms. In one form, it can be a supervised learning with a teacher present who knows the answers that a network is supposed to produce. If the answers that the teacher knows do not agree with the network response, the teacher issues a modification signal for this adaptive neural network, and this process in a feedback loop of learning goes through many steps and many cycles. In a pervasive form, the teacher punishes and rewards the network by issuing weight corrections that usually go in a negative radian descent through stochastic approximation methods. I don't want to go into it. It's the subject of a graduate course in engineering or maybe statistics. A little bit more evasive learning takes place in an unsupervised model because, in that case, the teacher is absent. There are data at the input and something that can be seen as a feedback. When new data are put in, they can learn from their previous occurrences and previous histories through the learning rules that are embedded in the network. The weights can adjust themselves and arrange themselves in order to detect classes or capture principal component values present in the data.

The heart of any neural network is a very simple arrangement. It's a scalar product computer that multiplies input vector *X* times real coefficients *Ws*. We are computing the value NET as *X* times transpose times *W.* That scalar product, called by three letters, NET, is either processed through a signum function, so plus or minus one is stripped out of the scalar product only, or a soft activation function is present in the neuron, which maps again in that open interval, the whole space of *n*-dimensional real numbers. So In this *n*-dimensional real number, each value is real, and we are getting on the output only an open interval –1 to 1. There is a big processing power in that.

Once we arrange neurons in layers, and I show neurons generally as circles. That is a single layer feet-forward network that computes *O* based upon *X*. There is a linear computation and a linear computation mixed together, and all of them are analog. Although we can model it in a digital way, it's not a big deal. The single layer architecture is very rigid and doesn't allow us to do many things, so we employ two-layer architectures. We have a so-called hidden layer and an output layer. This network is trained by an error back-propagation algorithm, and it's capable of these complicated approximations non-linear relationships, classification, and predictions. For instance, for classification purposes, we would train this network for binary values of the vector *O*, which manifests itself as a membership to a class (one or two or three).

For function approximation, we have *K* different functions to be approximated because each output produces one function. We would have to have these values normalized because a neuron responds to values between plus or minus one. But this network is just purely instantaneous, and it computes output based upon input. At the same time, it is not able to model recurrences or a time series, but if we insert delays, we can model any time sequences in a pretty accurate way. How do we model them? We first have to know the sequences, or inputs and outputs, and then perform multi-step training by submitting data for which we know outputs.

Then the weights are being trained into that system so that we can identify this discrete time model. There are a few architectures that do that.

There are a lot of mathematics behind the learning of these architectures. I would like to go over them. Essentially, learning is adjusting a single weight. There are typically hundreds of weights, if not thousands of weights, around the network. So depending on what we try to learn, how we try to learn, we implement the delta-*W* operation; delta *W* is computing the increment of the weight vector of the i^{th} neuron, as a product of three components. One is the input vector, learning signal, and a constant. I don't want to go into details. The analytical justifications and theory for learning is mathematically well-founded. The learning has one prevailing concept behind it: delta-*W* the weight increment of a single neuron is always proportional to the input vector. Delta-*W* is a vector, so each weight increment is a product of constant times another number. That is a real number, times the input vector. That's one paramount feature of neural network learning.

There are many learning rules. Depending upon whether it's supervised or unsupervised learning, we use either perceptron, hepion, winner take all, and the delta learning, or Arabic propagation learning, which is generalized delta learning. I don't want to go into it, but there is one complete learning algorithm. It will be given, I believe, at the end. It's a complete learning algorithm for Arabic propagation learning. It is the most famous, the most needed, the most used algorithm. It refers to this architecture, and it continues through two subsequent pages. I am highlighting this algorithm because this is probably the only example of an algorithm that is put in a box and ready to program for a moderately skilled programmer.

You can develop a training algorithm for this entire very rich network by following these steps and implementing the Arabic propagation training algorithm. Again, you would probably not like to do it. My graduate students have to do it to pass the course. Once you have written this algorithm, you have an understanding of how this network learns and how it works. There are many packages, that can do it for you, and they are already pre-programmed.

I would like to discuss the application issues because they are very attractive. First, one can easily show that the neural network behaves in a way similar to the computation of an integral. The problem is that neural network produces this approximation. There is this staircase to learning and not by hand or other numerical integrations. So you can teach the network to produce that wavy curve by imposing an architecture on the network and producing a few examples of pairs. That all amounts to producing training pairs. This is, of course, a single variable function, but if we have *n*-dimensional real numbers into *N* functions, then we have to have pairs of vectors for multivariable inputs and examples of functions.

We have done a very large project that actually modeled a multidimensional semi-conductor manufacturing facility, which cannot be described by any mathematical equation. They simply do not exist. No one even dares to find out what the output is with respect to the input because you cannot model this thing.

There is a great deal of measurement data that allow us to develop a good model for the purpose. I want to show an example of a classifier and go into expert systems, which can be very interesting to actuaries. Classification is a very popular engineering application and very useful. You can use pixel maps as input and classify whether the character is one or *B* or *A* and build class files that work completely free of mistakes. By shoving inputs and outputs repetitively to a network, you can develop 100% correct classification on at least printed characters. For handwriting, it's not as perfect because handwriting is not always very clean.

Here is an expert system story. I think it's a big success story, and I would count that probably some of you either have been thinking about something similar or maybe you are working on neural network aspects of the expert system already. We'll discuss a medical expert system that I have described in my book. The expert system takes symptoms, processes them through a neural network in a trained way, and produces one of the *K* diseases at the output. We are able to differentiate among *K* diseases. In a simple example, one has taken 200 patients, and has developed a very good expert system in Scotland for distinguishing among four back pain diseases. Back pain diseases are very difficult to diagnose. The authors have claimed a great success because neural networks that are shown here have outperformed the fuzzy logic expert system, as well as teams of doctors that had a diagnosis rate of only 70%. These networks were developed based on symptoms and post-mortem diagnoses. Post-mortem diagnoses were used because one had to know for sure whether the disease was 1 or 2 or *K*. We had to deal with four possible diagnoses because there are four diseases that amount to low back pain.

Another project that was developed in San Diego by Bachst, was for imminent or incoming heart attack diagnoses for patients that arrive at the emergency room with heart pain. He took 20 symptoms, the history, examination results, and EKG findings. He listed them, trained the neural network on several hundred patients, and has produced an expert system that performs better than doctors, once again.

I wanted to extrapolate what would come out if I were working in the actuarial area. I would repeat that neural network, which provides a very versatile, multivariable modeling, again with no restrictions on Xs and Os. We can use real or binary numbers or certain values that we doubt. The network at the output can compute risk evaluation, scores, and rating, or quality assessment or return value or lifespan, or whatever economic factor or financial factors we invent. However, it only makes sense to try to model this if we don't know any analytical formula, or if they do not exist. It is best to have sufficient historical records available that would be able to produce input/output pairs.

There is one aspect behind this particular mode of computation. Once the network computes O values, We sometimes need the reasons for the answer so we can justify to our boss, why we have to do this or that. That is another topic. I have been working on so-called rule extraction from these knowledge-based expert systems. In order to do the rule extraction, you have to evaluate what the mappings are here. Make them a little bit coarser and a little bit cruder, and then

evaluate the if-then rule. Based upon the accuracy that is required, we can produce 3, 5, 15, or 20 rules that are logic rules in terms of human language.

I can give you one example of a logic rule behind an expert system that is from a very bizarre area. My colleague in Japan got me into it. He developed a classifier of poisonous mushrooms. The story is not very trivial because you want to see which mushroom is poisonous or edible. You have a database of different samples of mushrooms. You can see the mushroom, how it smells, how big it is, and so on. You would enter these attributes and get a very rich connection within that neural network. We need to simplify it, trim down the neurons, and trim the weights to get some explanations. After he developed that expert system, it turned out that the logic rule for an edible mushroom is merging odor and spore print color. This is the logic value for an edible mushroom. There are only 24 errors produced if you use this rule on a big population of mushrooms. There are 24 errors. No one can be perfect, but this was pretty impressive.

Imagine we have 16 animals. Suppose we have a topographical or component map of animals. How was this map produced? Each animal has about 12 attributes, including, the number of legs, whether it has feathers, what it eats, how big it is, and whether it is domesticated. So we list these 12 attributes and compress the closeness of animals into a sheet of paper. As you can see, the duck and goose are related but they were originally characterized in 12-dimensional space. The duck and goose are related. The horse and zebra and cow are related. These are grass eaters. Predators are in another area. There is one more example that I wanted to add—compression of multidimensional data into very pervasive 2-D space. Out of 10,000 or 20,000 dimensional relationships, you can dump something on a piece of paper and convince your boss that Mary is similar to Joe.

I am running a test in my class. I have 30 students typically in a class, and I ask each student to characterize himself/herself with respects to 30 aspects. So they rate themselves from 0 to 10. Then I develop a similarity map on the plane, so you can see that Mary is similar to Joe, Jim is similar to John, but let's say Steve and Raymond are very dissimilar. Steve drinks beer and Raymond drinks only milk. So it's a compression of data into the extreme. You cannot simplify it more. It gives very nice results as well.

The best business use of neural networks is to access good data records that you might have. Don't attempt to build a model because models do not exist in many situations. Identify input and output variables. Choose a suitable neural network model. This is not very hard to do because that one, two-layer model is pretty flexible. Split data into training, testing and validation sets. That already relates to the technique of good training. Train neural networks toward the validated input/output process modeling on specific data or modeling specific data. If needed, extract logic rules that are behind certain problems that would provide justification as to why the decision has to be such and such, and not the other. Order form factor analysis. When new evidence comes, you need to re-train the network and update the models, so you can start from scratch and re-train the network.

Mr. Evans: Our next speaker is Fred Watkins. He runs HyperLogic Corporation in Escondido, California. He has a bachelor's degree in mathematics from Louisiana State University (LSU), a master's degree in mathematics from LSU and a master's degree in computer science and engineering from Tulane. He also has a Ph.D. in electrical engineering from the University of California at Irvine. His firm works primarily in the area of fuzzy logic, primarily doing consulting. He also provides some software tools in that area.

Dr. Fred A. Watkins: The name of the game for me is fuzzy logic, even if it doesn't necessarily do as well as Dr. Zurada's neural networks. We might square off one day, you know. What is fuzzy logic? It's the business of deciding what follows from what when the what's are vague or ambiguous. That is the key that distinguishes the kind of logic that this is from the standard Aristotelian or Boolean logic. So here's an example of fuzzy logic. There's light rain outside, whatever that means. Rain here is always breezy, whatever that means. From that, you can conclude that if it's raining outside lightly, then it's probably breezy outside as well, at least to some degree. As soon as we try to quantify all this stuff , we get into a mire. If we try to be too rigorous in our logic, that is if we try to be too Aristotelian, we're liable to get gummed up in a hurry. Nonetheless, everybody seems to be able to understand what it means, and they agree that there's a certain amount of truth, plausibly at least, in this fuzzy modus ponens.

It is about logic in the face of ambiguous concepts. We normally, to make it manageable, translate the logic into truth values. Once we have truth values, that is to say numbers, then we can ask our computing machinery to deal with those numbers and save us the trouble of dealing with them ourselves. It gives a certain amount of insulation. Years ago, when the Gulf War was going on, Dan Rather had a fellow on his show. At that time, Saddam was burning all the oil wells. It was a big mess over there. Dan Rather asked this fellow, "Well, what about the ecology of the region?" The guy said, "It's ruined. The place will never be the same. It will be one hundred years or more before it recovers." Six months later, of course, Red Adair and his boys cleaned up the place, and Dan brought the fellow back. Dan said, "You told me six months ago that the place was going to be a mess forever, and it's not. What is the excuse that you can give us?" The guy said, and this is a classic response, "I told you what the model said." This is a good reason not to use models or a good reason to use models because, if you have one, you can always hide behind it when it fails. If it succeeds, then it's you, but if it fails, then it's the model. That's real good.

Let's get back to the subject. Here are some grains of truth. The kind of thing that we want to do is we want to associate numbers between 0 and 1 with degrees of truth. This is not necessarily the same as believability or abstract knowledge or anything else. It's just truth. The undefined term. We like to say, If it's 1, then whatever we're talking about, if that's the truth value, then that thing is absolutely right or absolutely certain. Absolutely means always was, always will be, now and forever, with no exceptions, no how, no way, anytime, anywhere. True. That's what it means to be true in the Boolean sense. This is a serious thing. I can't stress it enough. What happens is, sometimes people say "Oh yes, that's true."

You might say, "Well, what about X?" They say, "Oh, well, yeah, forget that." That's not true. Truth is absolute.

Now that we have our truth reduced to numbers, we have to find a way of combining truths because we want to combine the concepts to which they refer. Here are the rules. This is how it's done. For every truth, or every statement that you make, . all the truth values sit between 0 and 1. Sometimes you want to say, " I have this concept and that concept. They each have a truth value. I'd like to know what's the truth value of the thing that you get when you say this and that." Table 1 is a mapping of a syntactical construct, *A* and *B*, and we want to be able to take the truth of *A* and *B*. Here's how you do it. You use the min, the smaller one. Conversely, when you have *A* or *B*, one uses the max. Of course, there's also that word *not*. We give the truth of not *A*, 1 minus the truth of *A*.

TABLE 1
FUZZY LOGIC OPERATIONS

Domain:	$\forall A \quad 0 \le T_A \le 1$
AND:	$T_{A \wedge B} = \min(T_A, T_B)$
OR:	$T_{A \vee B} = \max(T_A, T_B)$
NOT:	$T_{\neg A} = 1 - T_A$
IMPLIES:	$T_{A \Rightarrow B} = \min(1, 1 - T_A + T_B)$

In short, fuzzy logic and probability are different. We also see the word, *implies,* which is very important. *A* implies *B*. How do you know the truth of the implication, and not the result or the implication itself. What is the truth of *A* implies *B* if you know the truth of *A* and you know the truth of *B*. There is at least one way of saying what it is. Actually, there are a lot of ways to say what it is. We just write that down, and we are done with it. Tell it to the computer and let it do its thing.

Another thing that's important to know is that the truth values distribute. Here is some heavy duty math, If you have *A* and then open the parentheses in your head, you have *B*1 or *B*2 or *B*3 or *B*4, all the way out to *BN*. You get the same value using the rule that I just gave you. It's the same value as *A* and each of the *B*s. Get all those truth values and take the maximum of them. That will be the same number that you get if you take the min of the truth of *A* and the truth of that whole.

That's just how it works. The two things to take from Table 2 are the "and" and the "or's. " They have a relationship to each other with respect to the truth values they distribute. This is also true of the "not."

TABLE 2
"DEMORGAN" RULES

$$A \wedge \left[\bigvee_K B_K \right] = \bigvee_K \left[A \wedge B_K \right] \quad \text{AND Distributes Over OR}$$

$$A \vee \left[\bigwedge_K B_K \right] = \bigwedge_K \left[A \vee B_K \right] \quad \text{OR Distributes Over AND}$$

$$\neg \left[\bigvee_K A_K \right] = \bigwedge_K \left[\neg A_K \right] \quad \text{NOT Distributes Over OR}$$

$$\neg \left[\bigwedge_K A_K \right] = \bigvee_K \left[\neg A_K \right] \quad \text{NOT Distributes Over AND}$$

Chart 1 is the DeMorgan rule illustrated. Let's look at the union of the *A* and the *B*s. The common parts are slashed. Chart 1 is just a pictorial explanation of how the DeMorgan rules work.

Here's the heart of the matter. A proposition is properly fuzzy if it's neither absolutely true nor absolutely false. In other words, a fuzzy thing is and is not simultaneously. Fuzziness is the interplay between is and is not. Anytime you start trying to describe something by asking, Is it that? and the answer is "yes and no,", then you're really in the realm of the fuzzy. Formally, fuzzy logic denies Aristotle's primary principle of Boolean type precepts, the law of the excluded middle, and the law of noncontradiction. The law of excluded middle says that something is either true or false. There's nothing in between, hence the name. The law of non-contradiction says something cannot be true and not true at the same time. So in those cases in which you're tempted to say yes and no, that's ruled out. You are pushed aside and ignored by Boolean logic. It doesn't mean it's not there; it just means that Boolean logic does not cover it.

Mathematicians love to say "in the limit." When all the truth values that you're discussing go to the limit values 0 or 1, using the rules I have given you, then the results you get by using the fuzzy logic methodology to calculate truth values match what you get from regular Boolean logic. So the fuzzy logic people like to say, "Fuzzy logic generalizes Boolean logic." There's a little bit of a trap there that I'm not going to go into. However, in this sense of the mathematics, reducing to the Boolean case in the case where the arguments of the discussion are Boolean all works out very well. We say then that the fuzzy logic generalizes the Boolean logic. So it has that nice property. It tells you that if you have a fuzzy system, that's doing something and you push the system into a region where everything is well defined, then the system would be doing the same thing that logic would be doing for you in that case. I think that is what you'd want. It's certainly what I would want if I was building a machine like that.

To calculate the truth value of the and of two propositions, you use the min of their truth values. In fact, there are other things you could use, like t-norms. What's a t-norm? I'm not going to tell you. You will have to look it up. Suffice it to say it's a lot like min. Among all the t-norms that there are, min is the biggest. If you have a t-norm and an *A* and a *B*, and you apply the t-norm to the *A* and the *B*, and you apply the min to the *A* and the *B*, the number you get from the min will be no less than the number you got from that t-norm. So min is the largest t-norm. Conversely, max is the smallest t-conorm. Instead of max and min, you can use a t-norm and its corresponding t-conorm. I just tell you this for completeness. It's only of interest to some people who make their living writing papers for fuzzy journals. They dwell on this kind of thing. I saw one article by a famous fellow from Europe. He spent 16 pages on all these t-norms. The article was done in 10-point type, single-spaced. It's a matter of some interest to some people.

What I've told you up to now is all you need to know if you want to do fuzzy logic. Now you probably want an application. By and large, unless you're doing *A*, *B*, *A* implies *B*, and the truth of *A* or the truth of *B* is something other than 0 or 1, we're not really using fuzzy logic. We're doing fuzzy systems, and that's different.

If fuzzy systems will let us do something intelligent with information that is available to us, then so be it. We will consider using it. At least I would. So if we're going to use a system, we're going to need a way to map the things that happen in the world into truth values. If you don't have those truth values, you can't give them to the machine. If you can't give them to the machine, you can't crank out the numbers. This is a no-brainer, right?

So how do you get truth values from events that happen in the world? We have nomenclature, *T* sub *A* is the truth of *A*, whatever it is. Truth and probability can coincide. In fact, some people like to think that probability is belief. Of course, there's a problem there. If that's the truth, then without people, there's no probability. So if you don't believe it, then it's not. Anyway, it's possible that fuzziness can coincide with probability, but it doesn't have to. The primary reason for that is that fuzzy numbers need not be summable but a probabilistic number must be summable. That's the big difference. I say fuzziness and probability differ because probability is dealing with things that sum and fuzziness need not.

Have you ever heard of L1? That is a mathematician's name for all the things that integrate to a finite value. Among other things, the things that integrate are the probability densities. There is no fuzzy density. The fuzzy density could be, if you want to call it that, the line that is one, all the way across the reals. That does not integrate, and it is not summable. Therein is the difference. Everything else is the same.

Let's discuss a definition of fuzzy set. Why would we want to think about a fuzzy set? It's the case that Boolean logic, standard mathematical set theory, and lattice algebra are all the same. They are what we call isomorphic. I claimed that the circles in Chart 1 were sets or something. The reason that you can claim that a circle and a set are the same, and you can deal with them similarly is because of this isomorphism. You can say that something is a set, and you can have the

intersection of these things defined. The intersection of two sets is the and of sets and is also the conjunction of notions represented by those sets. The fact that you can do that is given by this isomorphism.

If we can deal with things in a set-wise context, so much the better for us because set-wise things are logical. Logical things can be handled by the computer. Then we can spend our time past this curtain. A fuzzy set, by definition, is any function that maps any place, which we'll call the universe of discourse, into the closed unit interval. It is any function whatsoever. Now, if you're mathematically trained, then you know some of those functions can be real nasty. The ones you see in papers written by me or anybody else will not be so nasty, but they could be. The way we define the operations on the fuzzy sets is to define them point-wise. Functions are defined at point values in the universe of discourse, so we can compute the maximum, given two functions that are in that range. That is to say, the function which is point-wise the maximum of the two, or the minimum of the two, or the one minus of any one of them. You can set up an algebra of these functions. Whoops, did I say algebra? Just a little while ago we said set algebra and so on. We're doing set theory with functions instead of actual round things on planes. This is the same game.

Chart 2 is a picture of a fuzzy set. The fuzzy set is Tall. Tall describes a measurement, usually in feet and inches. Let's say the numbers down at the bottom are in feet. I already said that a fuzzy set was a function from 0 to 1 and therefore the vertical axis is 0 to 1. And if you know what tall means, then you know that something that is 0 inches tall is not very tall. If someone is ten feet tall, he or she is tall. Chart 2 is a picture of how tallness increases with the measurement of somebody's height, the truth of tallness. Take the truth value of the statement "X is tall" where X is what you're measuring. X is tall is true, according to this, if you're seven feet or more. X is tall is false if it is zero. X is tall is, like say, a half, if you're at the three-and-one-half foot mark. This graph pictorially describes the meaning of the word tall, according to the person who drew it. We're moving from words on the one hand to computational units on the other.

If inflation is low and gold is low, then you ought to buy some gold. If inflation is low, then it might go up. If it goes up, then gold will be more valuable. Besides that, if it's cheap now, it might be more expensive later, so buy now. We have a hypothesis, and a dual-pronged antecedent. There is something about inflation on the one hand and something about the price of gold on the other. We have a consequence, which is what follows the word t*hen*. This is all syntactically analyzable, and so a computer, in principle, can not only do the arithmetic of the fuzzy logic part, but it could actually parse the sentence and do the whole thing from regular English. We define the truth of this rule as the truth of its antecedent. Now here you have an antecedent consisting of two parts with an "and" in between. I've already told you how to calculate the truth of two parts that have an "and" in between. You use the min or some t-norm. So the truth of this implication is this much. If you match that against the real world, you might get the truth value of the result. The consequence is the inducement for you to buy gold.

A fuzzy system will contain several fuzzy sets that describe a given quantity, whatever the quantity is. This might be tallness, and we might be trying to come up with some kind of decision rule for people based on their height. For actuaries, it's more like their age. So we'll probably put that in instead. Age can be small or large, depending on age. The degree to which you are any of those things is given by these functions. Plug in an age, get a value of smallness, largeness, and mediumness. What those functions look like is up to you. It's what it means to you, if you're going to use the system. Nonetheless, once you've committed, then you have an automatic way of saying how much is this person, low age, medium age, high age, and what kind of consequences can I infer from that?

Let's suppose that the bottom axis is age and the left axis is output. Let's suppose that that is suitability for insurance. I'm real simple-minded when it comes to that. You want to insure somebody if you think you're not going to have to pay off. We're selling life insurance now. If the age of the subject is low, they're probably not going to keel over in the next couple of days or years, and the suitability for insurability should be very high. There could be all sorts of special codicils in the policy that might invalidate that conclusion, but I'm trying to simplify.

On the other hand, as people age, then it becomes more likely that you have to pay off. Finally, I don't know this for sure, but there's probably some age beyond which nobody in his or her right mind would take the policy or sell it. Maybe you can't do that. Maybe the law doesn't allow that. Nonetheless, from a practical point of view, by the time somebody gets to be at the end of the graph, things are looking pretty dicey. You will probably not be so anxious to sell somebody a long-term life insurance policy because this is a bad time to do things like that

Recall the example in which we discussed small, medium, and large. If we knew what the truth values are for something that is state of nature in terms of small, medium, large, we could say that we're computing with words. The input is small, medium and large to some degree, and our machine will take the degree to which we're small, the degree to which we're medium, the degree to which we're large, and come up with something. You can go a little further, as Chart 3 demonstrates.

Mr. Watkins: So you can spend a lot of time reading books about how to say *very* and *somewhat* and *a little bit* and *medium lukewarm* and so on in a mathematical way. You can make this all pretty precise. Now whether you believe it is another matter, but you can do this kind of stuff. Many people have done that to an appallingly lengthy degree.

Now I'm going to tell you how to put it all together. You take the state of nature, and you measure the world. In your case, that may mean reading the form. You find some information from the world somehow, and you get numbers. You array these numbers in a vector of inputs. We could try that on a neural net, and hammer on it with a neural net. Instead, what we're going to do is weight the fuzzy sets in our fuzzy system according to the degree to which our state of nature matches each of the words that show up in those fuzzy set rules. We will, from each rule, get a fuzzy set output. Remember, if the price of gold is low and if inflation is low, then buying gold is high, and we're talking about fuzzy sets. The

output of a rule is that consequence of the rule, and it's a function. A function that always sits between 0 and 1 is a thing that can be added to other like-styled things. In the end, you get this function, which is a sum of all your rule outputs. If you want to, you can divide that by the number of rules that you put into the sum. That will leave you with a function between 0 and 1.

You can compute the centroid of that function. The centroid is where the graph of the function balances. It's the single number that contains basically all the information in the function. It uses it all.

You get an output, a value, or a number that you can apply to a machine, or to this business of yes or no decisions. If it's bigger than one-half, you say yes, if it's less than one-half, you say no, and if it's actually one- half, which would never happen, you would need to do it again. You keep doing that until you're happy with the answers or you just keep doing it in general. The thing just runs and does for you what it's supposed to be doing.

We do this kind of procedure to control temperature. We sample the temperature; we decide whether it is hot, or tepid, or cold? It's all of those to some degree. We have rules that state, if it's hot, turn up the air conditioner, if it's medium, leave the air conditioner be, and if it's cold, turn down the air conditioner. Put all those rules together, and you get a setting for the thermostat. You just keep doing that. It works. The good thing about it is, it is not computationally expensive. This stuff is fast. Here is the output of the rules from a fuzzy system. That's the place where it balances. That's the number you get out; it is the *X* value that corresponds to that.

Mathematically, you're assuming that there is a real function somewhere. You don't know what it is. You pass from the place where you started up to this place of fuzzy sets. You think of the fuzzy sets in the other place, and you pass across from fuzzy set to fuzzy set. That's the job done by the fuzzy system. Somehow you get from the fuzzy set back down to the real world. That's what the centroid does for you. Mathematically, we say we have a commutative diagram, which is for only the mathematically inclined (Chart 4).

This last little thing is for fun. It's called the fuzzy cognitive map. It has nothing to do with all that other stuff we've been talking about, but it's fun to use. It's basically a way of letting influences combine. So I'm not going to say much about these descriptions of it.

Let's take a look at the marijuana game (Chart 5). There are the narcs, the growers, and the rip-offs. These are the guys that you sometimes hear about going up into Humbolt County and being blown up by hand grenades set by the growers. They're out there to steal. There's one more player. How much money can you get for some standard unit of the goods? I've put arrows in the Chart. If the sale price is going up, then the rip-offs are going to go up because they're going to go out there and try to get some of the stuff and do some sales of their own. The more sales value there is, the more growers there will be. That's obvious supply and demand. The more growers there are, the more narcs there will be, at least as long as funding is there. Then, the more narcs there are, the less growers there'll

be, unless there's a whole bunch of bribery going on. Presumably, the more narcs there are, the more likely they are to catch those rip-offs. Anyway, you get the picture. The arrow is plus 1 if one bubble influences the other bubble, the target bubble, in the positive or aggrandizing direction and minus 1 otherwise. If there's no effect, then we just don't draw an arrow. You can take those arrows and translate them into a matrix.

In a set of initial conditions, we might find no narcs, no rip-offs, some growers, and some profit. What is the motive? You run that vector through this matrix in the standard way, and you get a vector. If the result of a number was bigger than 1, you make it 1, or bigger than 0, you make it 1. If it's less than 0, you make it 0, and if it is 0, you leave it alone. The theory is that if you keep the value from last time, you will eventually get what we call a limit cycle. There will be some set of vectors that just keep coming up, one after the other. In fact, sometimes it's just a single vector. That's the equilibrium state of nature. If the mechanics or the dynamics of the fuzzy cognitive map are the same as the dynamics of the world that it's trying to model, then you have an accurate picture of what's going to happen. Of course, the model and reality rarely agree, and that's something you have to keep working at. That's why people have jobs in which they correct these models. This is how you do it.

Bart Kosko, who's a big gun in fuzzy logic, or Rod Taber took some situations from South Africa before the end of apartheid and made a big cognitive map out of that. They made some conclusions and they matched up remarkably well with comments from Henry Kissinger. I don't know what you think about Henry Kissinger, but everybody has to agree that the guy is smart. Moreover, his opinions about the state of nature in the world should not be taken lightly. This thing matches up with his opinion. Here's how you do it. How do you combine the knowledge of everybody in this room on some question? I could ask you to draw one of those cognitive maps for some thing. Let's say it is some complicated social problem. Should we have taxes or something? You might say you don't want taxes or you do because you want that road repaired in front of your house. You would write down all these little things that would make you decide one way or the other. You would have the little bubbles and draw all the arrows. Having done all of that I would take up the papers and I would convert them all to matrices. Wherever the labels coincided I would combine the numbers and end up in the end with a big matrix. Then I would divide by the number of people, and I would have a cognitive map. I would have the opinions of all of you in that map. I can do that in a very simple, straightforward way. Whether it works or not remains to be seen, but that's the game.

I'll finish up with four simple statements. First, fuzzy logic is the business of inference with vague data. This is the most important point. Second, fuzzy logic rejects the law of the excluded middle. That's really statement number one rephrased for the purists. Fuzzy logic operations can be arithmetized. This pertains to t-norms and all that, but just remember max and min. Finally, let us consider the practical applications. The cognitive map and the additive system that I showed you do not involve fuzzy logic. The people that make them and the people that write about them tend to call them fuzzy logic, but they are not fuzzy

logic. They were spawned by fuzzy logic but they are not fuzzy logic. If you're going to use fuzzy logic to make a decision, as an actuary, you will use a system like this in some degree.

Mr. Thomas P. Edwalds: I'm trying to get my arms around what you guys were presenting here. I thought if I gave you an example, you could tell me what your techniques would do. What if we were going to collect data from 25 insurance companies?. Let's say we have the underwriting information for everybody that they considered for insurance, including blood pressure, height and weight, cholesterol readings, and the history of cancer. We must also know whether they issued the insurance or not and even what class they put them in. There might be a whole bunch of preferred classes or whatever. We also have a history of how many of these people are still alive and how many died and what they died of. How would we use either the neural networks or the fuzzy logic to help us decide what's the proper price to charge for an individual, given a particular vector of information that an individual comes in with?

Mr. Watkins: I can comment on that. In the end, what you ask for is an input/ output relationship. There are numbers you put in, and numbers you want out. Somebody has to know what that relationship is to write down the rules that say how to get that stuff. That is presumably what you learn when you become an actuary. Somebody has to know how to do it or else somebody has to know when you're doing a good job. If somebody already knows, then mostly you'd look in my direction. If somebody merely knows when you're doing a good job, you might look to neural networks. Either way, you'll have to look at these numbers, either expressed by the rules or not. The key thing that came up in the first talk was that the neural system is model free. That was the big deal. If you don't have the model, then you really have no option but to try a neural technique. It is a statistical thing that will work in the absence of a model, and if you don't have a model, you can't use normal statistics. There's no way to do regression with one of those putative lines in there or curves or whatever. That would be how I would react to you. Do you know the reaction, or the function that you want? Is it expressed as either rules or as a description? A machine can be taught to learn what it is, or we can tell it what it is in the rules and polish the rules over time.

Dr. Zurada: Essentially, you have a lot of input data from the records, and then you have to wisely define what your outputs are. The outputs might be risk categories: very low, low, medium, high, very high, extremely high. Based on historical records, you know which candidates for certain policies have fallen into certain categories. You need to use historical data. You can build that model. Then, when you have a new candidate for a policy. You consider him an input, then you are going to compute one of the six outputs. One of the six outputs in the training process is high, and the others are low. You can have 20 outputs depending upon very tiny shades of the risk, right? Perhaps five or six outputs is what you need. So the new candidate is the new input. If the neural network is properly trained, it will produce the risk value for this particular fellow. You don't need to look at 100 input factors because you are going to get the class of this particular applicant in terms of risk. So that was the point that I was trying to make.

Mr. Evans: Let me add to that. You were talking about underwriting criteria. Let's take a possible application of fuzzy logic to say, preferred underwriting. Typically, with preferred underwriting, you have a set of rules and either you meet each individual rule or you don't. Unless you meet them all, you don't get the preferred underwriting classification. What if each of those rules or each of those characteristics, whether it be cholesterol, blood pressure, weight, whatever, was a fuzzy variable? You'd use some of this fuzzy arithmetic that Fred was talking about to come up with a decision that took into account all these fuzzy variables. At the end of the day, you would produce a risk classification.

Dr. Zurada: The subtle differentiation between the fuzzy and the neural network is that you have to have a lot of experiential training data to conceive either of the two. You don't have to have an analytical formula that is underlying both things. With a lot of historical data, you can certainly do both.

Mr. David B. Atkinson: I'm intrigued by that last question. Can you carry that a little bit farther? In our business, whether someone lives or dies is what's important. That would have a 0 or 1. It's just the opposite of fuzzy. There is no in-between state of being, like not dead and not alive at the same time. So can this then apply? Can you apply fuzzy logic to that kind of outcome?

Mr. Watkins: This will never vary. There will be four inputs in either words or number for the measurements. We're going to replace the word with numbers based on the current state of nature. Then there's going to be some computation that occurs and out of that computation will come a number. We can arrange it so that the number that comes out is something between 0 and 1, but it is always very close to 0 or very close to 1 except in the most improbable cases. what probability people like to call measure 0 event, where we just can't decide. If it turns out the machine can't decide, then you just have to live with that. You can make the probability of that event small, and then if it can decide, this decision might be 0.9 instead of 1.0, but 0.9 is the same. You're going to disambiguate in a very natural way. Even the neural net can't always do 0 and 1 if its actual outputs are graded between 0 and 1. There's always some set of circumstances that'll make it sit right on the cusp. If it's continuous, it's guaranteed by the math, and it has to be.

Mr. Atkinson: I probably expressed that incorrectly. The inputs are 0 and 1 the experiential data of whether people lived or died. Maybe that's where I'm a little confused.

Dr. Watkins: What do you infer from the fact that they lived or died? What happens next?

Mr. Atkinson: Let's say you put in a whole body of data, for example, 100,000 people. All these underwriting factors alluded to their cholesterol, their height and weight, their blood pressures, and so forth. I might try to draw some averages out of all of the interactions of all those variables. If you list the things underwriters look at, you'd see that there are probably 10 or 20 chief variables that are all kind of artfully put together to assess the risk.

Dr. Watkins: Then you have data. Where's the teacher? You have all the actuarial data you've collected and aside from that, you also have the Boolean outcome. By using the actuarial data, you can tell the neural net or the fuzzy logic system, if you knew the rules, what the outcome is. Is the guy alive or dead now? The answer might be good or bad.. If it turns out that it gives you a good answer most of the time, then you can use that as an aid. If it doesn't, then you either abandon it or you go and train it some more and try to get it to give better answers. In the end, that's what it is with every real system that we build, unless it's one of these things that happens to sit on some abstract mathematical threshold, and we can just crank it out. In other words, if you want something that will calculate the area of the circle, and if we suppose that the input given is precise, then the output should be pretty precise. That's the end of that. But if it's real world stuff, all that certainty goes out the window, and you'll never have a system that gives, in advance, 100% reliability. That cannot be done. The future is under no obligation to obey the past.

Mr. Evans: The application to mortality might make more sense if you expect the output of the neural network or fuzzy logic process to be a probability of mortality for a given individual rather than zero or one.

CHART 1
"DEMORGAN" RULE ILLUSTRATED

"DeMorgan" Rule Illustrated

$$A \wedge (\underset{k}{V} B_k) = \underset{k}{V} (A \wedge B_k)$$

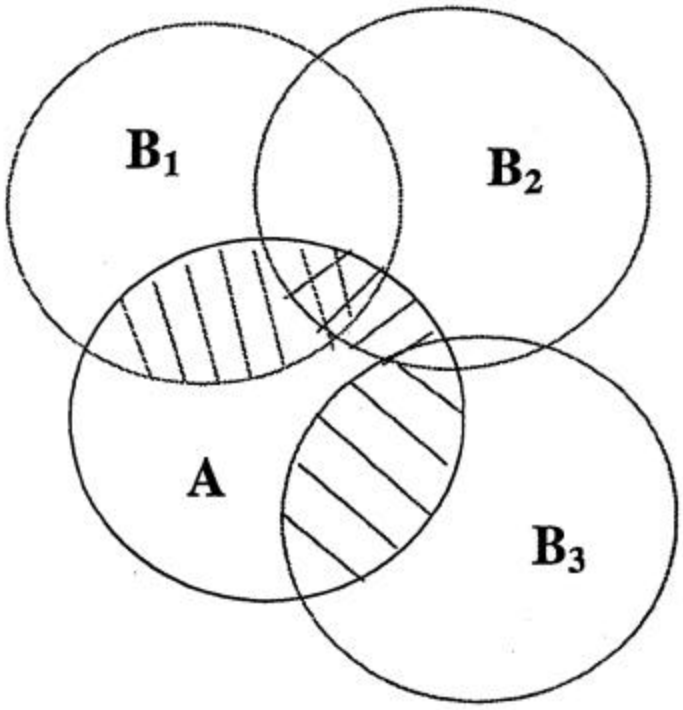


CHART 2

Truth Value from Measurement

- **Plug a measurement into a fuzzy set, get a truth value.**

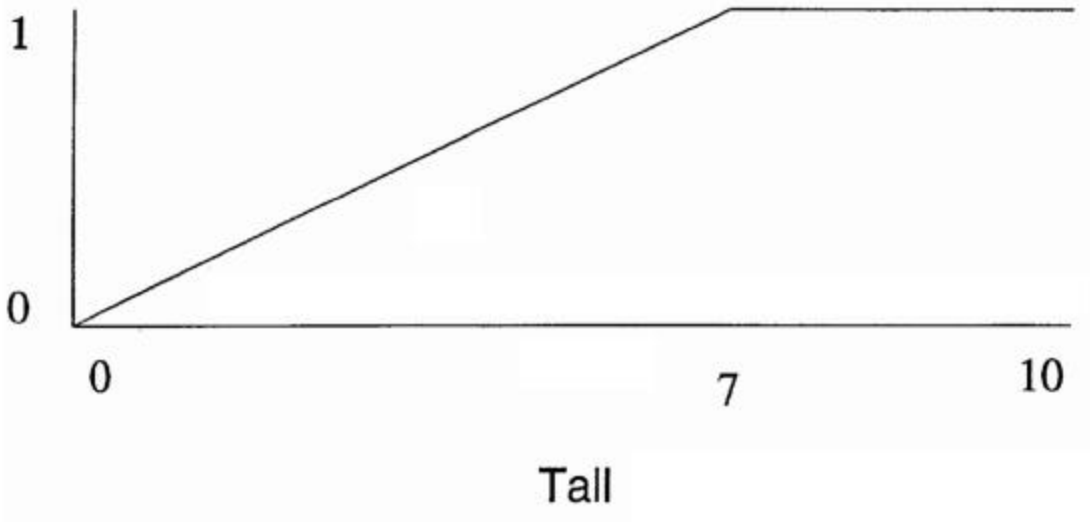
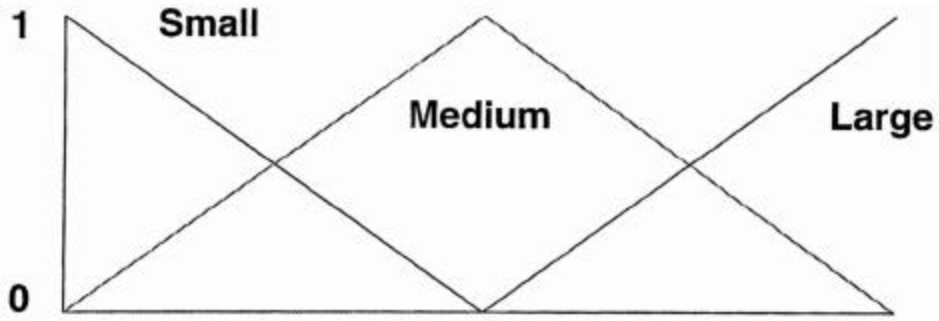


CHART 3

Resolution into Set Values

- **Partition a variable's range into descriptive sets:**



- **A measurement is SMALL, MEDIUM, or LARGE to some degree.**

CHART 4

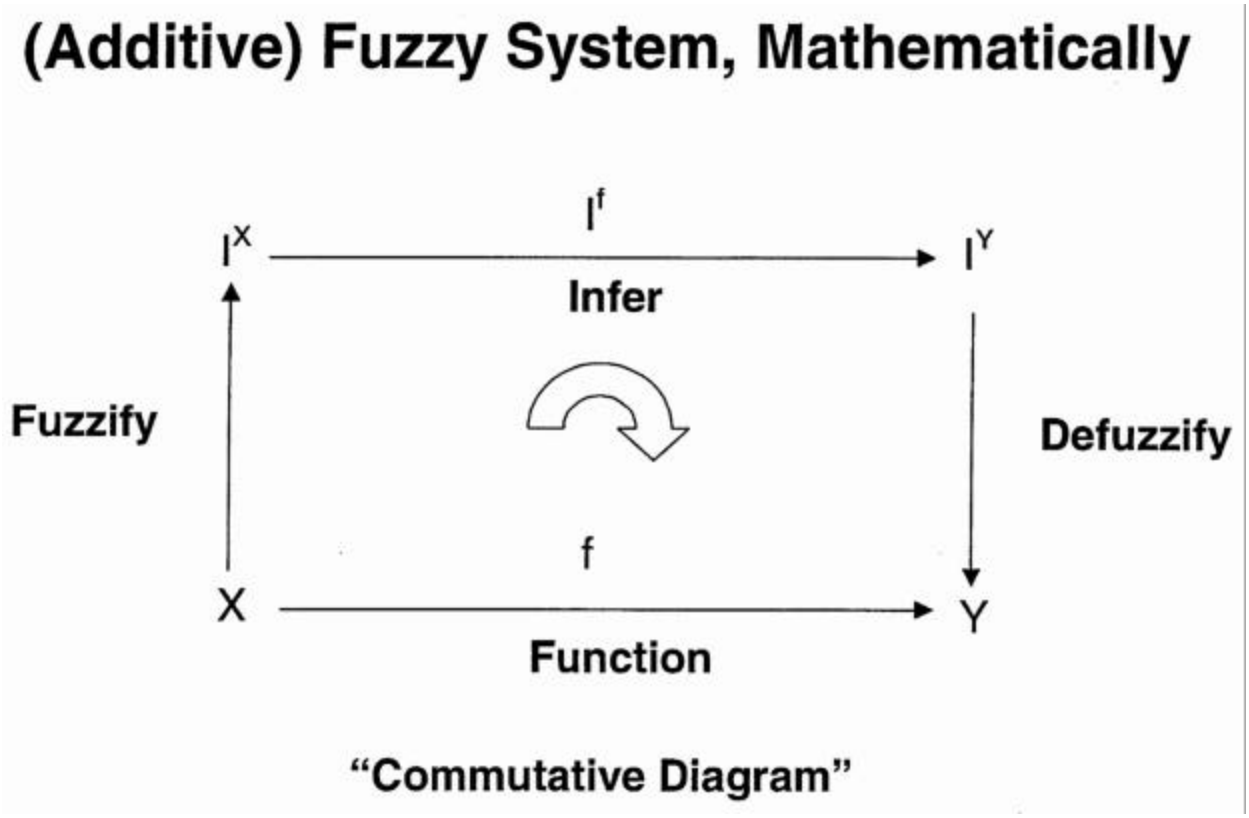
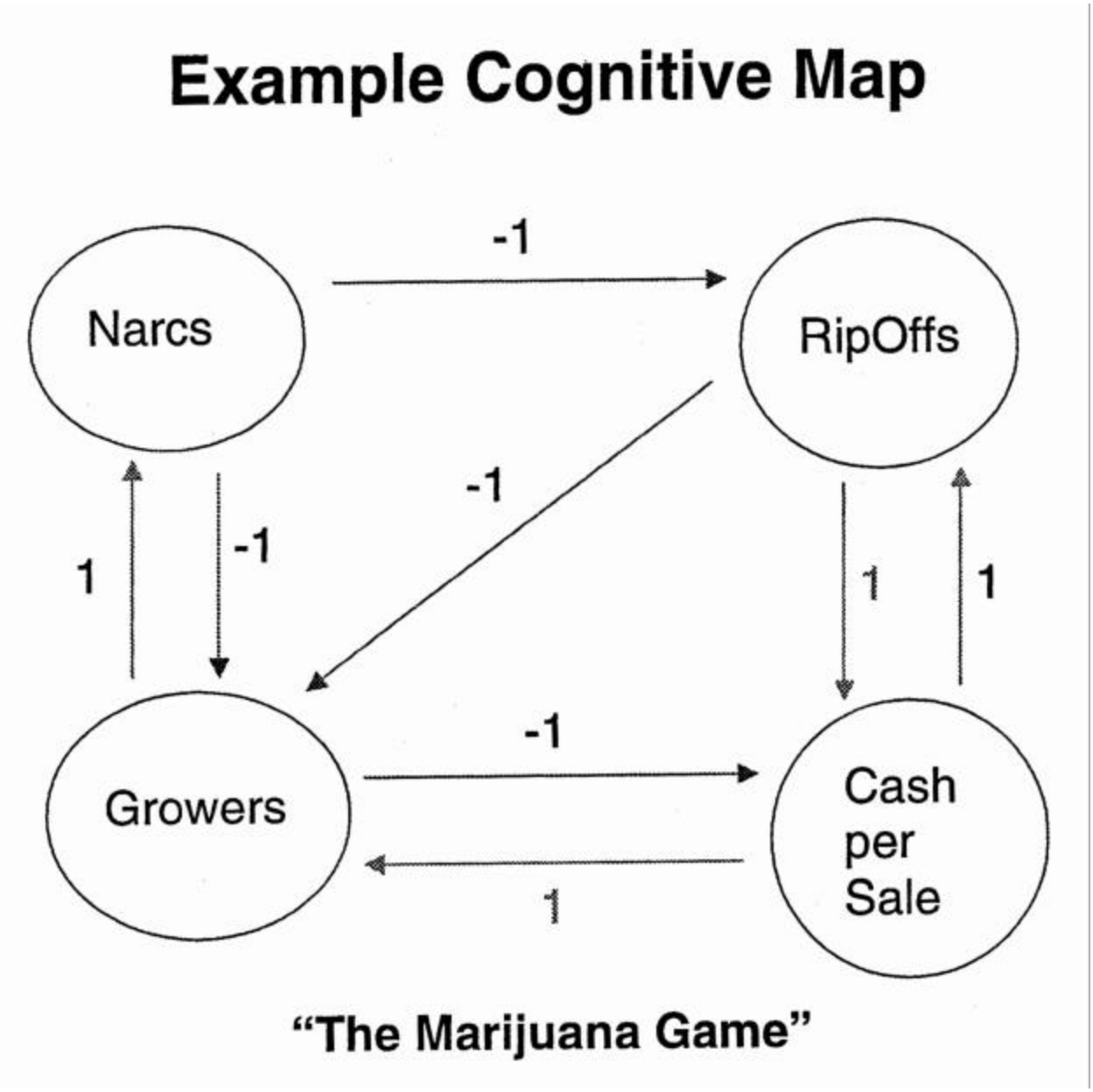

(Additive) Fuzzy System, Mathematically
I^f
I^X
I^Y
Infer
Fuzzify
Defuzzify
f
X
Y
Function
"Commutative Diagram"

CHART 5


Example Cognitive Map
Narcs
RipOffs
Growers
Cash per Sale
-1
-1
-1
1
-1
1
1
-1
1
"The Marijuana Game"