Networks of Relations for Representation, Learning, and Generalization

Matthew Cook and Jehoshua Bruck
California Institute of Technology
Mail Stop 136-93
Pasadena, California, 91125 USA
{cook,bruck}@paradise.caltech.edu

Abstract— **We propose representing knowledge as a network of relations. Each relation relates only a few continuous or discrete variables, so that any overall relationship among the many variables treated by the network winds up being distributed throughout the network. Each relation encodes which combinations of values correspond to past experience for the variables related by the relation. Variables may or may not correspond to understandable aspects of the situation being modeled by the network. A distributed calculational process can be used to access the information stored in such a network, allowing the network to function as an associative memory. This process in its simplest form is purely inhibitory, narrowing down the space of possibilities as much as possible given the data to be matched. In contrast with methods that always retrieve a best fit for all variables, this method can return values for inferred variables while leaving non-inferable variables in an unknown or partially-known state. In contrast with belief propagation methods, this method can be proven to converge quickly and uniformly for any network topology, allowing networks to be as interconnected as the relationships warrant, with no independence assumptions required. The generalization properties of such a memory are aligned with the network's relational representation of how the various aspects of the modeled situation are related.**

I. Introduction

Cortical neural structures such as gain fields [2] appear to implement relationships between small numbers of variables, such as the three-way relation between two successive joint angles and the resulting composite angle, important for an animal using its body. Given any two of the values, this three-way relationship can be used to deduce the third value.

This relation can be thought of as a trivial associative memory which has learned a pattern of triples $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$, and if 2/3 of a pattern is supplied to it, it will complete the pattern. Similarly, if another relation relates variables $\langle \alpha_3, \alpha_4, \alpha_5 \rangle$, then the two relations can communicate the value of α_3 to each other, and if α_1, α_2, and α_4 are supplied then the relations together can calculate both α_3 and α_5. Again, this can be thought of as an associative memory which, if given any three variables which are independent (i.e. the third is not computable by one of the relations from the first two), can find the values of the remaining two variables.

The relations described above act on three variables, with each variable in the cortical implementation effectively represented by a neuronal bundle projecting both into and out of each relational mapping area where it is used. Presumably there are many such relations in the brain, each similarly acting on a small number of variables, and it seems reasonable to guess that these relations probably form a large interconnected network. Rather than speculate too much on how the brain works, here we simply take these ideas as our inspiration for developing an abstract model of relational networks and investigating how they can be used to solve interesting problems.

We define the basic model in Section II, explore some natural modifications and extensions in Section III, and apply the model to the task of learning to ride a bicycle in Section IV.

II. Exclusion Networks

We define an *Exclusion Network* as a set of variables v_i and a set of relations r_j, with each relation relating a tuple of variables $v_{k \in t_j}$. For this section, we will assume the variables to be discrete, taking values from a finite set. In a *Logical Exclusion Network*, which is what we will first focus on, each relation is a logical relation, meaning that any particular tuple of values is either allowed or disallowed by the relation.

For example, if each variable v_i takes values in the range $1, 2, \ldots m_i$, then a relation relating variables v_2, v_5, and v_8 could be represented by a three dimensional $m_2 \times m_5 \times m_8$ array of zeros and ones, with the ones indicating which value triples are acceptable, so a one in position $(2, 1, 4)$ in the array would mean that $\langle v_2, v_5, v_8 \rangle = \langle 2, 1, 4 \rangle$ is allowed by the relation, while a zero in position $(2, 2, 6)$ in the array would mean that $\langle v_2, v_5, v_8 \rangle = \langle 2, 2, 6 \rangle$ is not allowed by the relation. In this way, the relation specifies exactly which tuples of values the related variables are allowed to have, or in other words, how they are related.

The training of such a network consists simply of letting it record the examples it sees. For example, a one in position $(2, 1, 4)$ of the array for a relation on variables v_2, v_5, and v_8 means that the network has seen some situation in which $\langle v_2, v_5, v_8 \rangle = \langle 2, 1, 4 \rangle$.

Of course, this training method only works for relations on variables which are directly observable or which can be inferred using relations which are not being trained. Training of relations on unobservable variables (conceptual modelling variables) will be discussed in Section IV-C.

When a network has been trained, it can be used to reason about hypothetical or partially-observable situations.

The reasoning method is simple, and is the source of the name "Exclusion Network". It works by indicating, for each possible value of each variable, whether that value is currently considered to be plausible for the current situation. To start with, everything is considered to be plausible. Then, for any observable variable, the values which it can be seen not to have (that is, all values except the observed one) are excluded, by marking those values as not plausible. Then the following asynchronous distributed process occurs: Each relation, knowing the possible tuples for its variables, can exclude those tuples which contain excluded values for any of the variables, since those tuples are not plausible tuples. This effectively yields a reduced relation, namely the set of tuples that still appear to be plausible. Any value not appearing in any of the reduced relation's tuples is then marked as excluded for that variable (if it was already so marked then there is no effect).

For example, if a relation relating v_3 and v_6 allows the tuples $\langle v_3, v_6\rangle \in \{\langle 1,1\rangle, \langle 1,2\rangle, \langle 2,2\rangle\}$, and then the value 2 is excluded for variable v_6, then this relation will exclude those tuples in which $v_6 = 2$, leaving just the tuple $\langle 1,1\rangle$. Since no remaining tuples allow $v_3 = 2$, the value 2 becomes excluded for variable v_3. This information is then transmitted to the other relations involving v_3, which might allow them in turn to exclude yet other values of other variables.

Theorem 1: The updating process described above always converges to a stable state.

Proof: Since the variables are discrete, they only have a limited number of values. Each step of the process can only exclude more values. Once a value has been excluded, it remains excluded for the duration of the process. Thus the process must end at a stable state, where no relations can exclude any more values. ■

Theorem 2: The updating process described above converges quickly.

Proof: If each newly excluded value is considered as one step of progress, then the maximum number of steps that can occur is $\sum_i m_i$, which is less than the descriptional size of the network itself. ■

This maximum number of steps only occurs if everything is excluded (which is indeed a stable state), which only happens if the supplied partial input was already inconsistent with the network's experience.

One may still ask whether the stable state that is reached is necessarily unique, that is, whether it is independent of the order in which the various relations exclude the various variable values. The answer is that the stable state is indeed unique, meaning that asynchronous implementations do not need to worry about any subtle effects arising from the asynchrony. The uniqueness of the stable state is due to the monotonicity of the exclusion procedure.

Theorem 3: The order in which relations update their information has no effect on the final stable state reached.

Proof: If we define a state of the network as a list of all variable values which have not yet been excluded, then we get a partial ordering on states where $s_i \preceq s_j$ means that state s_i is a subset of state s_j. It is straightforward to prove that if $s_i \preceq s_j$, then any variable value excludable at the next step by s_j is also excludable or already excluded by s_i. Therefore if there is a stable state s_s reachable from the current state s_c, then since $s_s \preceq s_c$, we know that it is impossible to exclude any variable value not excluded by s_s. Therefore the next state $s_{c'}$ reached by the asynchronous distributed process will also satisfy $s_s \preceq s_{c'}$, and by induction the eventual stable state $s_{s'}$ reached by the process must also satisfy $s_s \preceq s_{s'}$. So any stable state reachable from the current state must be $\preceq$ any other, and in particular $s_{s'} \preceq s_s$, so we must have $s_s = s_{s'}$, indicating that the stable state reachable from the current state is unique. ■

This section has defined the structure and behavior of Logical Exclusion Networks, as well as proving some basic properties of the resulting dynamics, but it is still useful to understand in intuitive terms what the network is and isn't calculating. Given partial information about a situation, the network is doing its best to deduce what it can about the missing information, based on the relationships it understands, assuming the relationships observed in past situations apply to the current situation as well. If it is given partial information corresponding to a previously seen situation, it may or may not be able to fully deduce the missing information corresponding to that learned situation.

A simple reason for being unable to fully deduce the missing information is that perhaps two (or more) learned situations both match the variable information supplied, while differing in other variables. In this case the network has no way to know which of the memories should be retrieved, and will not retrieve any values which are specific to only one (or only some) of the memories. For example, a network that "has seen it all", i.e. has been exposed to every possible combination of variable values, will never be able to deduce anything, since as far as it is concerned, all combinations are possible, and knowing some variable values never proves anything about any other variable values.

A more subtle reason for being unable to fully deduce the missing information is that the relations in the network may not be able to represent some relations existing in the data. For example, suppose v_6, v_7, and v_8 are observed to take on all possible combinations of values except $\langle v_6, v_7, v_8\rangle = \langle 1,1,1\rangle$. If the fact that $\langle v_6, v_7, v_8\rangle$ is never equal to $\langle 1,1,1\rangle$ is not correlated with any other variable values, then it is only representable by a relation that relates v_6, v_7, and v_8 (possibly among other variables). If the network does not have a relation relating v_6, v_7, and v_8, then it cannot "observe" this fact about its world, and it will not be aware of this relationship. In particular, if it is provided the partial information that $v_7 = 1$ and $v_8 = 1$, it will be unable to exclude the possibility that $v_6 = 1$.

However, this is not all bad, as it is exactly such "interference" between correct memories which leads to the ability to generalize to similar situations. The generalization is based on the general knowledge embodied in the learned relationships. The network does not learn anything specific about what it

sees, beyond how groups of variables are related for those groups whose relationship it is studying. This allows the network to generalize by assuming future situations will match the relationships it has studied, while not assuming anything more than that.

III. More General Exclusion Networks

A. Tallying Exclusion Networks

While in some cases the rigid logical nature of Logical Exclusion Networks is exactly what is desired, in other cases one might want a less strictly logical behavior. For example, if on one peculiar occasion some rare combination of variables was somehow observed, but on the vast majority of occasions nothing like it was observed, then it would be nice if a network could base its inferences when possible on the majority of what it has seen, rather than always giving equal weight to the one erratic case.

Such a modification turns out to be implementable with a simple enhancement to the Logical Exclusion Network. Instead of each relation simply recording whether or not it has seen some particular combination of values, it records how many times it has seen that combination of values. Thus, the logical relation has been replaced with a tallying relation. A *Tallying Exclusion Network* is the same as a Logical Exclusion Network except that the array of values for each relation can now contain any non-negative integers, rather than just zeros and ones. For each relation, this array is simply the discrete joint distribution (histogram) of the related variables, as observed and recorded by the network.

When a Tallying Exclusion Network is being used to reason about a partially described situation, the exclusion process must also be generalized to operate on non-negative integers. Now, instead of each value for a variable being marked as simply plausible or implausible, each value gets marked with an integer "cap" which indicates a bound on the number of previously seen situations (having the variable values corresponding to the location in the array) that might correspond in all aspects to the current partially-specified situation.[1] The exclusion process can start in the same way, with each relation excluding those tuples (reducing their tally to zero) which do not match the partial information on some supplied variable. Then, a relation can look at all the tuples it has where, say, $v_4 = 5$, and see that the largest tally for any of these tuples is, say, 37 occasions. Based on this, it can reduce the cap for $v_4 = 5$ to 37. Then, when another relation receives this information, that "the current situation, whatever it may be, fully matches at most 37 previous occasions if variable v_4 has the value 5," it can look at every tuple it contains in which $v_4 = 5$, and if any of them have tallies higher than 37, it can reduce such tallies down to 37. This reduction of tallies may lead to other totals being reduced, leading again to propagation of information throughout the network.

In this way, a Tallying Exclusion Network can yield information not only about whether a particular variable value is consistent with the given information, but in some sense *how* consistent it is, so if there was a single outlying observation that dirtied the observational data, it will not have much of an effect on the output, and as more observations are accumulated over time, any individual outliers will have a vanishingly small relative influence on the numbers.

The proofs of convergence from Section II also apply to the Tallying Exclusion Network. Note that any Tallying Exclusion Network can be mapped into a Logical Exclusion Network by simply mapping positive numbers to 1 and 0 to 0, so a Tallying Exclusion Network always knows exactly what the corresponding Logical Exclusion Network's state would be. To make the proofs apply without modification, a more complicated mapping is to enlarge the set of possible values for variable v_i from $\{1, 2, \ldots m_i\}$ to $\{(a, b) \mid a \in \{1 \ldots m_i\}, b \in \mathbb{Z}^+\}$, so that the tally is treated as part of the value. Although the set of possible values appears to become infinite, only a finite number of them are plausible at any time, and the values can now be formally treated as a Logical Exclusion Network with no loss of information.

B. Continuous Exclusion Networks

Another way in which we would like to generalize the Logical Exclusion Network is by allowing continuous variables. This turns out to require almost no modification of the framework at all. Merely allowing infinite sets of variable values and infinite sets of tuples for the relations turns out to take care of this quite nicely. Of course, a continuous (infinite) set of training experiences would be needed in this case to populate the data in the relations. The resulting formalism can be called a *Continuous Exclusion Network*.

However, while the proof of a unique limit for convergence remains unscathed by this modification, the proof of "quick" convergence falls apart, and indeed, it is easy to construct examples which converge at any desired rate, meaning that there is no rate of convergence which can be guaranteed. However, in practice, such pathological examples do not seem to be the norm, and convergence does not tend to present a practical problem.

With continuous variables, a dimensional analysis is possible for certain kinds of state spaces. If the state space is a k-dimensional manifold in the n-dimensional space of possibilities (the space defined by n continuous variables), then any relation relating k or fewer of these variables is unlikely to be able to capture the nature of the manifold, whereas relations relating at least $k + 1$ of the variables are going to be much more successful at modeling the nature of the

[1] Alternatively, it could indicate the maximum number of times the variable might have had that value in *similar* situations (corresponding only in the specified aspects), from among the learned situations. In this case, addition would be used (instead of the maximum) when calculating new caps. Such a network still has the properties of unique and swift convergence, but the overall deductive performance of a nontrivial network using this method becomes relatively poor due to the accumulated degradation of signal to noise resulting from repeated summations as information flows through the network. Here we will not analyze such "Tally-Sum" networks, but only present "Tally-Max" networks. (Although we will not discuss them here, there do exist some arguments in favor of Tally-Sum networks, and they are closer to belief propagation networks than Tally-Max networks are.)

manifold.[2] If k is large, then one would like to avoid having such large relations, and in this case one can use auxiliary variables to distribute the calculation so that a network of relations on three variables each can suffice. In Section IV-C we will explore automated ways of creating such auxiliary variables.

C. Continuous Tallies

Can we merge Continuous Exclusion Networks with Tallying Exclusion Networks? Strictly speaking, there is no theoretical difficulty with allowing a continuous range of variable values, with an integer tally for each of the uncountably infinite possible combinations of values. However, this does not correspond to the sort of data we have in practice.[3] In practice, we would like to make the tallies be continuous as well, so each relation stores the joint probability distribution on its variables. This is the continuous limit of storing a discrete histogram (with all entries expressed as fractions of the whole) as the number of training samples goes to infinity and the granularity of the discrete variables becomes infinitely fine.

The resulting calculations are equivalent to traditional Bayesian networks, but with the *minimum* operation taking the place of multiplication, and *maximum* taking the place of addition. *Minimum* replaces multiplication because we are manipulating upper bounds, not probabilities, and we do not make any assumptions about independence. *Maximum* replaces addition because of the way we have defined what the bounds are on.[4]

D. Multivariate Exclusion Networks

One final extension to Exclusion Networks as so far described is to let relations communicate to each other not only regarding single variables, but also about joint distributions of variables. In general, two relations should be able to communicate regarding the joint distribution of all the variables they have in common.

If we consider such a joint distribution of variables as a single distribution on a new composite variable whose single value uniquely indicates the values of all the variables in the joint distribution, then the formalism becomes identical to the previous formalism. However, in a typical implementation, being able to pass joint distributions between relations generally constitutes a positive and not too difficult improvement.

IV. Learning to Ride a Bicycle

A. The Basic Network

If an Exclusion Network is going to ride a bicycle, it needs to have inputs from sensors on the bicycle, as well as actuators to steer, lean, pedal, or use the brakes.

There are essentially five degrees of freedom for the state of motion of a bicycle, and so sensors are needed for five variables: The speed of the bicycle, how much the bicycle is leaning, the rate of change of the lean angle, the angle of the handlebars, and the rate of change of the angle of the handlebars.

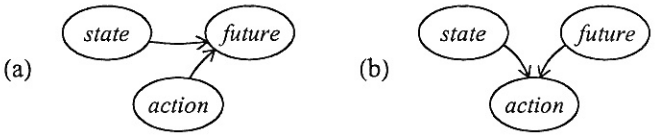

Fig. 1. Conceptual diagrams comparing the learning process and the utilization process for an associative memory control system. In (a), the learning process records how the environment tends to produce a future situation given the current state and the action being taken. In (b), the utilization process feeds in the known current state along with the desired future state to see what actions, if any, are compatible with that goal.

But what is the network supposed to learn? It needs to learn a cause and effect relationship in the non-constant context of the current state of the bicycle. The cause would be the action taken by the network at a time t, and the effect would be some summary of the resulting state of the bicycle at a later time, say $t + 1$. This summary of the resulting state need not necessarily be in terms of the five sensors giving the state of motion of the bicycle, but rather it should be in terms of parameters that are of interest for the goal-oriented processes that are likely to be trying to control the bicycle, such as the future location of the bicycle.

So we have three sets of variables: State variables, action variables, and future state variables. The state variables can be the five sensors listed above. The action variables can be one for acceleration (how much to pedal or brake), one for steering (how much to push the handlebars left or right), and possibly one for leaning (shifting some weight to the left or right), for a total of three action variables. The future state could be represented by as few as two variables, for the x and y coordinates of the location of the bicycle one second later, using a coordinate system centered and aligned with the bicycle. If the intended control system is likely to be interested in other aspects of the future (i.e. besides where the bicycle is going), then those aspects could be represented as well. So the network will have around ten variables that are inputs or outputs, connected to the outside world (or at least to a sensory or motor system that is closer to the outside world).

As the network rides the bicycle, it will constantly be recording observed relationships between old state, action, and new state. Simultaneously, the controller will use the network to help it decide how to use the actuators. It can tell the network what the current state is to get information about which future states are possible, given the current state. Then it can narrow down the future state to be one compatible with its overall goals, so as to see what sets of inputs are compatible with getting the bicycle to that future state.

A project exploring this approach is under development.

[2] These statements can be made much more precisely, but we will not go into such formalism in this paper.

[3] As they say, "In theory, there is no difference between theory and practice, but in practice, they are completely different."

[4] Note that a Tally-Sum network would use addition instead of the maximum.

B. Learning New Relationships

In previous sections, we have always assumed that the structure of the network is fixed. In this and the following section, we will examine natural ways to try to improve the operation of a network by modifying its structure.

The most obvious problem that a network might have is that it might be failing to notice a simple relationship between some variables simply because it has not been paying attention to how those variables are related, i.e. there is no relation present in the network that involves those variables.

The obvious fix is to simply add such a relation to the network. The not-so-obvious part of this is how the network is supposed to know when it is missing a simple relationship and should therefore add a new relation. This is mostly an issue for large networks, as small networks can simply include all possible small relations.

Unfortunately, there is no good general answer to this problem. Suppose that the combined parity of the following four items is the same every day for a year of your life: The day of the month, whether you have a hot or cold lunch, the number of bills you receive, and whether it is cloudy or clear in the evening. The question is, how would you ever notice that this is the case? Probably, you would never become aware of it. Even though you are perfectly aware of each of these values, you will never notice such a complex correlation between them (one which cannot be broken down into simpler dependencies among subsets of the variables) unless you are paying attention for some reason to how those values are related to each other.

A network can fare no better. The only way for it to know that there is a correlation to be noticed is by giving it a try. Theories of the brain often postulate similar strategies: The large number of neurons is taken as support for strategies where relationships are initially noticed by neurons that just happen by chance to be connected in the right way to be able to notice the relationship.

Of course, where dependencies exist, they can often be observed by noticing that changes are correlated, and although there may be many variables we are aware of, often only a few of them are changing at any given time. So for example, if we were to receive a bill during lunch and the temperature of the lunch changed immediately, we might indeed start paying attention to relationships between bills and lunch temperature. For variables that change infrequently, it is reasonable to set up a new relation every time two or more of them change at the same time. At worst, this can result in remembering the particular situation that came up.

To counteract the effect of adding relations at random or based on "suspicious coincidence" (such as changing value simultaneously), there needs to be a way to prune relations which are not helping the network. This turns out to be much easier, as it is easy to measure the extent to which a relation is affecting the network. After all, we know when information about a variable has an effect on a relation's distribution, and in these cases we can mark the source of the new information as "helpful." Over time, relations which are not helping can be pruned to keep the overall size of the network manageable.

C. Learning New Concepts

Even more interesting than learning new relationships is the notion of learning new concepts. For a network, this corresponds to adding new variables to the network. It is meaningless to add a new variable without also adding relations relating that variable to existing variables. These relationships essentially define the meaning of the new concept. It is not at all immediately clear how the new variable should be related to the existing variables. We will approach this question by considering a simple example.

The simplest case in which a new variable would be useful is if we have four variables which have a non-factorable relationship between them (not decomposable into a combination of relations on subsets of the variables). For example, say we have four continuous variables ranging between 0 and 1, and it happens to be the case that they all take uniformly random values subject to the condition that their sum is an integer. If the network has a four-way relationship relating these variables, then that will capture this relationship perfectly, but four-way relationships can be much more expensive than three-way relationships, so we may prefer to try to represent this four-way relation in terms of two three-way relations connected by a new variable.

From the outside, knowing what the four-way relationship is, it is clear that the problem can be solved by letting the new variable be the fractional part of a partial sum of just two of the variables. This allows the four-way relation (say on a, b, c, d) to be perfectly represented by two three-way relations using the new variable e (also ranging from 0 to 1), one of which requires $a + b - e$ to be an integer, and the other of which requires $c + d + e$ to be an integer.

However, from the inside, supposing the network has no four-way relations, it is not clear how to start. Any three-way relations among a, b, c, d are simply uniform distributions yielding no information whatsoever. So far as the network can tell, there is no relationship between a, b, c, d. If you only consider three of them at a time, they appear to be completely independent. In general terms, part of the problem here is that the space of values that a, b, c, d take on is a three-dimensional subspace of the four-dimensional space of possibilities, and so projecting down to three dimensions loses the detail about the shape of the three-dimensional subspace. Let's say the network decides to try to see if a new variable e could help discover some relation among a, b, c, d. What can the network do? It can create a relation for a, b, e and it can create a relation for c, d, e (here, as in many cases, it does not matter much how a, b, c, d get paired off into relations with e), but what should these relations be to start out with? And then, how can it learn (refine) the relation over time, if it is never told what e should be for any particular experience? To answer these questions, we will start by supposing the new variable can only convey a single bit of information between the two relations.

The new Boolean variable e can at first be connected completely randomly to a, b and to c, d. After all, there is no information in the network to indicate any particular

relationship between any of a, b, c, d. So we will assume that every pair (a, b) is initially associated with exactly one of the two Boolean values for e, perhaps at random. So then how can these relationships modify themselves over time so as to make e more meaningful?

The idea is that e should be trying to find a correlation between the two places it is being used. To be able to make inferences from a finite number of samples, we will want to assume that e behaves in a "continuous" way, which for the one-bit case means that e changes values as "infrequently" as possible. This will require a sense of locality for the variables e is connected to. We will present a simple algorithm and then show that it optimizes the distinguishing ability of e. Each local area in (a, b) space needs to decide whether to associate itself with $e = 1$ or $e = 0$. To do this, it compares the incoming value of e for those times (a, b) is in the local area to the average incoming value of e. If the incoming value is $e = 1$ more often, for the times (a, b) is in the local area, than it is in general, then the local (a, b) area associates itself with the value $e = 1$. On the other hand, if the local area "receives" $e = 0$ more often than average, then it associates itself with $e = 0$.

This natural sounding algorithm can be seen to optimize (locally) $pq - rs$, where p is the fraction of the time that the two relations agree that e should be 1, q is the fraction of the time they agree e should be 0, and r and s are the fractions of the time that they disagree in the two possible ways. Of course we want to be maximizing p and q while keeping r and s to a minimum, but the reason the product pq is used instead of simply the sum is that the sum is trivially maximized by always letting e have the same value, whereas we would like the two possible values for e to be used in a balanced way. The product pq rewards a balanced use of values for e. The product rs works similarly, as an uneven split between r and s yields a more useful relation than an even split does (the $\geq$ relation is the extreme case).

If this algorithm is executed simultaneously at the two relations, then the two relations will work together to find a meaningful way to use e. This can be seen to be effective in Figure 2.

If e can transmit multiple values, the algorithm can be modified so that each local area associates itself with the incoming value which occurs most often locally relative to its global frequency. This is better than trying to use the one-bit algorithm recursively (to learn more and more bits of e), since such recursion would require the bits to be learned in stages. These methods work well when the granularity of "localness" grows finer as more incoming data is accumulated. These methods do not directly provide a "topology" for e's values, but fuzzy methods and/or keeping track of "mismatches" may be able to provide this if needed.

Once the meaning of e has stabilized, one of the two relations can be left as it is, always being used for inference (even during training), while the other (plus any additional relations the variable may get used in) can be trained and used as described in previous sections for the case where all the variables in the relation are observables.

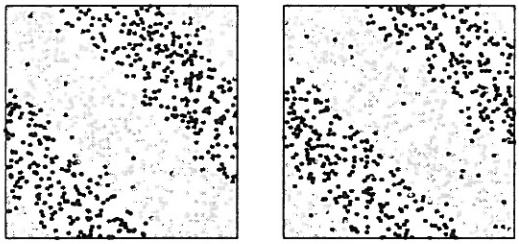

Fig. 2. The algorithm described in the text performs well when the number of local areas (shown here as points) grows at about 20% of the rate that data points (values for a, b, c, d) arrive at. The first square shows a versus b, with the color of each point indicating the associated value of e, and the second square shows the same for c versus d. Each square has 1000 points, each one representing the region for which it is the closest point. We can see how the new variable e has learned to represent one bit of information regarding the fact that $a + b + c + d$ is an integer by relating a one-bit approximation of $a + b \pmod 1$ to a one-bit approximation of $c + d \pmod 1$.

V. Conclusions

We have explored a new paradigm for constructing an associative memory, Exclusion Networks, which are based on the idea of an inhibitory, deductive style of computation being distributed throughout networks of relations. We have examined many specific approaches fitting this paradigm, and given an indication of techniques which could allow these methods to be applied to a practical situation.

The next stage of this research will be to actively apply these methods to a practical situation as described in this paper. It is to be expected that many new issues will arise in the course of this, and we view this line of research as a direction that has just begun.

On a final abstract note, the calculational process followed by Exclusion Networks can be understood as a message passing algorithm similar to the Generalized Distributive Law [1] used for belief propagation networks, but the operations used are different: Rather than a pair of operations satisfying the distributive law, $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$, Exclusion Networks use a pair of operations satisfying the absorptive law, $x \otimes (x \oplus y) = x$. As such, they do not calculate standard probabilities, but rather they allow stability and confluence to be provable properties of all networks.

References

[1] Srinivas M. Aji and Robert J. McEliece, *The Generalized Distributive Law.* IEEE Transactions on Information Theory, Vol. 46, N. 2, pp. 325–343 (March 2000).

[2] R. A. Andersen, L. H. Snyder, C. S. Li, and B. Stricanne, *Coordinate transformations in the representation of spatial information.* Curr. Opin. Neurobiol. 3, pp. 171–176 (1993).

[3] Ugo Montanari, *Networks of Constraints: Fundamental Properties and Applications to Picture Processing.* Information Sciences 7, pp. 95–132 (1974).

[4] Graeme S. Halford, William H. Wilson, and Steven Phillips, *Processing capacity defined by relational complexity: Implications for comparative, developmental, and cognitive psychology.* Cognitive Science, 13(3) pp. 295–355 (1998).

[5] Nicholas Pippenger, *Theories of Computability.* (Cambridge University Press, 1997)