The Connectionist Scientist Game:

Rule Extraction and Refinement in a Neural Network

Clayton McMillan, Michael C. Mozer, & Paul Smolensky

CU-CS-530-91 May 1991

Department of Computer Science and
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430 USA

(303) 492-4103
(303) 492-2844 Fax
mozer@cs.colorado.edu
To appear in: ***Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*****, Hillsdale, NJ: Erlbaum. 1991.**

The Connectionist Scientist Game: Rule Extraction and Refinement in a Neural Network

Clayton McMillan, Michael C. Mozer, & Paul Smolensky

Department of Computer Science and
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430

Abstract

Scientific induction involves an iterative process of hypothesis formulation, testing, and refinement. People in ordinary life appear to undertake a similar process in explaining their world. We believe that it is instructive to study rule induction in connectionist systems from a similar perspective. We propose an approach, called the Connectionist Scientist Game, in which symbolic condition-action rules are extracted from the learned connection strengths in a network, thereby forming explicit hypotheses about a domain. The hypotheses are tested by injecting the rules back into the network and continuing the training process. This extraction-injection process continues until the resulting rule base adequately characterizes the domain. By exploiting constraints inherent in the domain of symbolic string-to-string mappings, we show how a connectionist architecture called RuleNet can induce explicit, symbolic condition-action rules from examples. RuleNet's performance is far superior to that of a variety of alternative architectures we've examined. RuleNet is capable of handling domains having both symbolic and subsymbolic components, and thus shows greater potential than purely symbolic learning algorithms. The formal string manipulation task performed by RuleNet can be viewed as an abstraction of several interesting cognitive models in the connectionist literature, including case role assignment and the mapping of orthography to phonology.

Introduction

A cognitive behavior may be characterized in terms of a transformation from an initial cognitive state to a target cognitive state. There are many ways of specifying this transformation. At one extreme, an enumeration of input/output pairs provides a description of the transformation, but such a list is not particularly satisfying or concise because it does not explicitly capture regularities of a domain. An alternative is a set of symbolic condition-action rules that provides an algorithm for performing the transformation. The notion of explicit rules has played an important role in cognitive modeling because rules are a descriptive, readily comprehensible, and powerful representational language, and because people appear to exhibit rule-governed behavior when performing higher cognitive tasks. However, rule-based characterizations are often brittle and incomplete, and mechanisms for learning condition-action rules from scratch have not been extensively studied.

To model the process by which people acquire rules, it is instructive to examine theory construction in scientific domains. Scientists approach a domain first by observation, and then, armed with initial intuitions, formulate explicit hypotheses concerning regularities in the domain. Such hypotheses can be tested by experimentation, allowing for an iterative refinement of the hypotheses until eventually the hypotheses are consistent with the observations. We call this process of iterative hypothesis formulation and refinement the *Scientist Game.* We conjecture that rule acquisition by people in ordinary life involves a very similar process, one in which people play the role of the scientist and the hypotheses take the form of explicit rules in a given domain.

This paper proposes a modification of the scientist game, called the *Connectionist Scientist Game*, as a technique for hypothesizing and refining a set of rules through induction in a connectionist network. The Connectionist Scientist Game is an iterative process that involves first training a network on a set of input/output examples, which corresponds to the scientist developing intuitions about a domain. After a certain amount of exposure to the domain, symbolic rules are extracted from the connection strengths in the network, thereby forming explicit hypotheses about the domain. The hypotheses are tested by injecting the rules back into the network and continuing the training process. This extraction-injection process continues until the resulting rule base adequately characterizes the domain.

Although the view held by many neural net researchers is that explicit rules are unnecessary (e.g., Rumelhart

This research was supported by NSF Presidential Young Investigator award IRI-9058450, grant 90-21 from the James S. McDonnell Foundation, and DEC external research grant 1250 to MM; NSF grants IRI-8609599 and ECE-8617947 to PS; by a grant from the Sloan Foundation's computational neuroscience program to PS; and by the Optical Connectionist Machine Program of the NSF Engineering Research Center for Optoelectronic Computing Systems at the University of Colorado at Boulder.

and McClelland, 1986), the Connectionist Scientist Game suggests that rules can be exploited in connectionist networks in several important ways: 1) as a means for better understanding the inner workings of a network (McMillan & Smolensky, 1988), 2) as a technique for increasing learning speed, 3) as a means of constraining – and thereby improving – generalization, and 4) as a way of bridging the gap between sub-conceptual and conceptual levels of cognitive processing (Smolensky, 1988).

We describe an architecture called RuleNet, which, based on a characterization of the task domain, plays the Connectionist Scientist Game. Although connectionist networks are often conceptualized as embodying *implicit* rules, the RuleNet architecture embodies rules explicitly. In the following sections we describe the domain towards which this work is currently directed, the task and network architecture, simulations that demonstrate the potential for this approach, and finally, future directions of the research leading towards more general and complex domains.

Domain

We are interested in intrinsically rule-based domains that map input strings of n symbols to output strings of n symbols. Rule-based means that the mapping function can be completely characterized by explicit, mutually exclusive condition-action rules. A *condition* is a feature or combination of features necessarily present in each input in order for a given rule to apply, and an *action* describes the mapping to be performed on each input if the rule does apply. For example, a typical condition might be that the symbol A must be present at the beginning of the input string. The string ABCD meets this condition, while the string BCDA does not. A typical action might be to switch the first two symbols in the input, replace the third symbol with the symbol A, and copy the remainder of the input exactly to the output string, e.g., ABCD → BAAD.

In these simulations we allow three types of conditions: 1) a *simple* condition, which states that symbol s must be present in slot x of the input string in order for a rule to apply, 2) a *conjunction* of two simple conditions, and 3) a *disjunction* of two simple conditions. The action performed by a rule is to produce an output string in which the value of each slot is either a constant or a function of a particular input slot, with the additional constraint that each input slot maps to at most one output slot. In the present work, this function of the input slots is the identity function, and a constant is any symbol in the output alphabet.

The number of possible rules in a given domain is dependent upon the length of the strings, n, and the size k of the input/output alphabets. The number of possible rules is:

$$2\left(\left((k+1)^n - \frac{nk}{2}\right)\sum_{i=0}^{n} k^i \left(\frac{n!}{i!}\right)\right)^2$$

which is an exponential function of n.

An example of such a rule for strings of length three over the input/output alphabet {A, B, C, D, E, F, G, H} is:

if ($input_1 = A \wedge input_3 =$ G) **then**
($output_1 = input_3$, $output_2 = input_2$, $output_3 =$ B)

where $input_\alpha$ denotes slot α of the input string, and $output_\beta$ denotes slot β of the output string. As a shorthand for this rule, we write [∧ A_G → 32B], where the square brackets indicate this is a rule, the "∧" denotes a conjunctive condition, and the "_" denotes a *wildcard* symbol. A disjunction is denoted by "∨".

This formal string manipulation task can be viewed as an abstraction of several interesting cognitive models in the connectionist literature. For example Miikkulainen and Dyer (1988) have built a case role assignment network that maps syntactic constituents of a sentence into their semantic roles. This amounts to copying the representation of say, a sentence subject, such as *boy*, to a role slot in the output, such as *agent*. NETtalk (Sejnowski and Rosenberg, 1987) and McClelland and Rumelhart's past tense model (1986) are two further examples of problems that might be viewed as string mapping problems, although implementing them in the framework described above requires slight modifications to the original approach.

Task

The task RuleNet must perform is to induce a compact set of rules that accurately characterize a set of training examples. Note that any corpus of examples has at least one set of rules that correctly characterizes the mapping function, namely the set containing one rule per example. However, this set of rules is trivial in that it is no more interesting than an enumeration of the patterns themselves. We consider the appropriate set of rules to be the minimal set that completely describes the training corpus. In this work, the training set for a given simulation is generated using a set of prespecified rules as templates for creating valid examples. Input strings that meet the conditions of several rules in the template set are excluded. The rules are over strings of length four and an alphabet of eight symbols, {A, B, C, D, E, F, G, H}. For example, the following rules may be used to generate these exemplars:

[∨ _EG_ →30E2]:	GEAF→FGEA, BECE→EBEC
[∧ _BF_ →3G12]:	EBFF→FGBF, DBFG→GGBF
[_F__ →30B1]:	GFAB→BGBF,CFHD→DCBF

The patterns above are similar to those that form the training corpus used in the simulations described later. In general there is no guarantee that the underlying rule base represents the minimal set that describes the training corpus. However, given that in our experiments the number of rules is very small relative to the number of examples, it is unlikely that a more compact set of rules exists that describes the training set. Therefore, in our simulations the target number of rules to be induced is the same as the number used to generate the training corpus.

There are several traditional, symbolic systems, e.g., COBWEB (Fisher, 1987), that induce rules for classifying inputs based upon training examples. The power of these systems lies in their ability to build complex conditions that recognize relevant features of an input. It seems likely that, given the correct representation, a system such as COBWEB could learn rules that would *classify* patterns in our domain. However, it is unclear how such a system could also learn the action associated with each class. Classifier systems (Booker, Goldberg, & Holland, 1989) learn both conditions and actions, but the actions are limited to passing an input feature through unchanged, or writing a fixed feature to the output. Although classifiers could easily be made to represent the symbolic strings of our domain, there is no obvious way to map a symbol in slot α of the input to slot β of the output.

Architecture

We propose a neural network architecture, called RuleNet, based on the work of Jacobs, Jordan, Nowlan, and Hinton (1991), that can implement string manipulation rules of the type outlined above. As shown in Figure 1, RuleNet has three layers of units: an input layer, an output layer, and an intermediate layer of *condition units*. There are m rules represented in the network and one condition unit per rule. The input layer activates the condition units, which, to first approximation, participate in a winner-take-all competition. The winning condition unit enables one set of connections from the input layer to the output layer through a set of multiplicative or gating connections. The input units then pass their activity through the enabled set of weights to activate the output units.

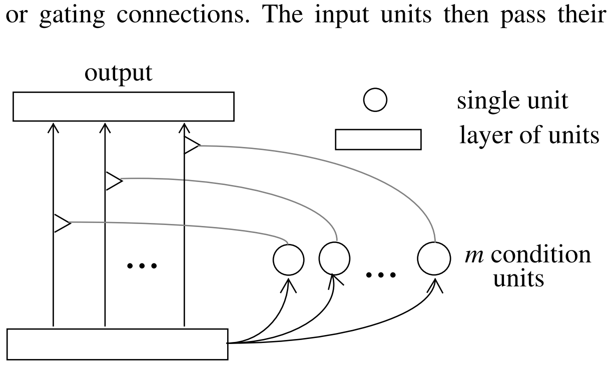

Figure 1
The RuleNet architecture.

The condition and action parts of a rule are implemented in different subnets. In the *condition subnet*, the net input to each condition unit i is computed,

$$net_i = 1/(1+e^{-\mathbf{c}_i^{\mathrm{T}}\mathbf{x}})$$

where $\mathbf{x}$ is the input vector and $\mathbf{c}_i$ is the incoming weight vector to condition unit i. The activity of condition unit i, p_i, is then determined by a normalization:

$$p_i = \frac{net_i}{\sum_j net_j}$$

The normalization enforces a competition among condition units. The activation of condition unit i represents the probability that rule i applies to the current input. In the *action subnet* there are m weight matrices $\mathbf{A}_i$, one per rule. A set of multiplicative connections between each condition unit i and $\mathbf{A}_i$ determines to what extent $\mathbf{A}_i$ will contribute activation to the output vector $\mathbf{y}$, calculated as follows:

$$\mathbf{y} = \sum_i^m p_i \mathbf{A}_i \mathbf{x}$$

Ideally, one condition unit is fully activated by a given input.

Although this architecture is based on the work of Jacobs et al., it is independently motivated in our work by the demands of the task domain. The Jacobs architecture divides up the input space into disjoint regions and allocates a different *local expert* network to each subspace. A *gating network* determines the probability that expert α will produce the correct output for a given input vector $\mathbf{x}$. During learning, weights are adjusted using back propagation to increase the probability that expert α produces the correct output and decrease the probability that other experts produce any output at all for $\mathbf{x}$. In effect, each network competes to be the sole network responsible for input $\mathbf{x}$. RuleNet has essentially the same structure as the Jacobs network, where the action substructure of RuleNet serves as the local experts and the condition substructure serves as the gating network. However, their goal—to minimize crosstalk between logically independent subtasks—is quite different than ours.

Input/output strings, such as AEG, are encoded as binary vectors. We use a local representation of each symbol and concatenate them together. The representation of the ith symbol in an alphabet of k symbols is a subvector of length k in which the ith bit is 1 and the remaining bits are 0. With the eight symbol alphabet {A, B, C, D, E, F, G, H}, then, the representation of AEG is composed of three subvectors of length eight concatenated together: (10000000 00001000 00000010).

To precisely implement rules of the type discussed in the previous section, it is necessary to impose some restrictions on the values assigned to the weights in $\mathbf{c}_i$ and $\mathbf{A}_i$. In $\mathbf{c}_i$, the first k weights detect the appropriate symbol in the first slot of the input string, the next k weights detect the symbol in the second slot, and so on, up to the length of the string. Since this is a local representation, one bit, at most, in each k-bit subvector should be nonzero. For example, using the eight-symbol alphabet, the vector $\mathbf{c}_i$ that detects the simple condition $input_1$ = A is: (10000000 00000000 00000000 0). Condition unit i will be activated only if there is an A in the first slot of the input string. The final weight in $\mathbf{c}_i$ is the condition bias, θ_i, which is required to detect a conjunctive condition. If the condition is a conjunction, as in [$\wedge$ A_G $\rightarrow$ 021], θ_i must be negative to compensate for the effect of one input; that is, the net input will be positive only if both symbols are present. If the condition is simple or a disjunction, θ_i should be zero to allow the net input to be positive if any input line is active. To encourage the condition weights to develop such a structure, it is necessary to initialize all weights in $\mathbf{c}_i$ except θ_i to non-negative values, and then set a lower bound of zero on those weights during training.

Similarly, if we wish the actions carried out by the network to correspond to the string manipulations allowed by our rule domain, it is necessary to impose some restrictions on the values assigned to the weights in $\mathbf{A}_i$. An action matrix $\mathbf{A}_i$ has an $n \times n$ block form, where n is the length of input/output strings. Each block is a $k \times k$ submatrix, and must be either the identity matrix or the zero matrix. The block at block-row α, block-column β of $\mathbf{A}_i$ copies $input_\alpha$ to $output_\beta$ if it is the identity matrix, or does nothing otherwise. An additional constraint that only one block may be nonzero in block-row α or block-column β of $\mathbf{A}_i$ ensures that there is a unique mapping from $input_\alpha$ to $output_\beta$. If $output_\beta$ is to be a constant, then block-column β must be all zero except for the action bias weights in block-column β. Further, because the output is a local representation, at most one bias in block-column β should be nonzero.

To ensure that during learning every block approaches the identity or zero matrix, we constrain the off-diagonal terms to be zero and constrain weight changes within a block to be equal for all weights on the diagonal, thus limiting the degrees of freedom to one parameter within each block. By imposing these restrictions on weight changes, the hope is that the resulting $\mathbf{c}_i$-$\mathbf{A}_i$ pairs are close enough to the block structure described above that it will be easy to extract a symbolic description of the mapping.

The constraints described above, however, do not guarantee that learning will produce weights that correspond exactly to symbolic rules. However, using a process we call *projection*, it is possible to transform the $\mathbf{c}_i$ and $\mathbf{A}_i$ weights such that the resulting network can be interpreted as a set of symbolic rules.

Projection of $\mathbf{c}_i$ involves setting non-essential weights to zero, setting essential weights to 1, and setting θ_i to zero if there is only one essential weight or if a disjunction is indicated, -1 otherwise. Because of the constraint that only one unit per slot in the input can be on, it is possible to subtract a value ε_α from slot α of $\mathbf{c}_i$ and add that value to θ_i without affecting the net input to condition unit i. This ε_α can be thought of as an irrelevant component of each weight in slot α, a by-product of learning. We estimate ε_α with a least squares procedure and use it to adjust $\mathbf{c}_i$. The resulting $\mathbf{c}_i$ is compared to prototype models of simple, disjunctive, and conjunctive conditions, and the closest model is taken as the projected condition vector.

Projection on a matrix $\mathbf{A}_i$ requires finding the largest block diagonal in $\mathbf{A}_i$, located say, at block-row α and block-column β, setting it to 1, and setting all other blocks in block-row α and block-column β to zero. The process is repeated until n blocks have been found (one per symbol in the input/output strings). Action bias strengths in a block-column are summed and treated as any other block during this process. The biases themselves are projected by setting the maximum bias in each slot to 1, and setting all other biases to zero. All biases in a block-column β with a non-zero identity block are also set to zero.

Simulations

Having described the projection technique for extracting explicit rules from RuleNet, we turn to simulations of the network. The simulations allow RuleNet to play the Connectionist Scientist Game by iteratively extracting rules and injecting them back into the network. To illustrate how this extraction-injection cycle improves RuleNet's performance, we compare learning performance using this technique with learning using several simpler techniques:

1) Technique J: Start with a fixed set of r rules, random initial weights, and minimize the error function described by Jacobs et al.

2) Technique JC: Learn as in technique J, but constrain weights in $\mathbf{A}_i$ to a single parameter on block diagonals as described above.

3) Technique JCA: Learn as in technique JC, but start with one rule, learn for m epochs, then add a new rule with random initial weights, and repeat until r-1 rules have been added or perfect performance achieved.

4) Technique JCAP (the Connectionist Scientist Game): Learn as in technique JCA, but before adding a new rule, *project* weights in $\mathbf{c}_i$ and $\mathbf{A}_i$ to conform exactly to the closest valid rule.

In all simulations, the maximum number of rules allowed was 10. In simulations using techniques JCA and JCAP, a new rule was added after every 500 epochs. All simulations ran for 5000 epochs and used a learning rate of .009 on both $\mathbf{A}_i$ weights and $\mathbf{c}_i$ weights. We tested four different rule bases, consisting of eight, three, three, and five rules (see Figure 2 for the rule templates) using strings of length four over the alphabet {A, B, C, D, E, F, G, H}. For each rule base, a training set was formed by generating fifteen random instances of each rule template. Figure 2 shows the error curve for the four simulations using learning techniques J, JC, JCA, and JCAP. Each curve is an average over five runs with different initial weights. The spikes in the curves for techniques JCA and JCAP reflect the additional error when RuleNet is allowed to make use of a new rule. In all four cases the error for techniques J and JC is monotonically decreasing; however, JC arrives at a consistently small error, while the pure Jacobs network, technique J, does not. There is a similar distinction between techniques JCA and JCAP: JCAP arrives at a reasonable solution, while JCA does not. In terms of learning speed and convergence, techniques JC and JCAP are clearly preferable.

The question is, can a given learning technique be relied upon to discover a valid set of rules? Table 1 shows the average number of valid rules extracted from RuleNet after 5000 epochs and the percentage of the training set mapped correctly to the output string in each case. The critical measure of performance should take both percentage of patterns and number of valid rules into account. Recall that our criterion for an appropriate set of rules was to find at most q rules, where q is the size of the set used to generate the training corpus, such that those rules completely describe every pattern in the training set. Learning technique JCAP is certainly the most effective judging by this criterion. In fact, technique JACP induces exactly the target number of rules in every simulation, mapping all the patterns correctly. JC maps patterns correctly, but at the expense of the number of rules. In this domain at least, it appears that RuleNet using technique JC makes use of approximately as many rules as one allows it. Technique JCA comes closer to inducing the target number of rules, but at the expense of consistent accuracy. Finally, technique J leaves quite a bit to be desired in both the number of valid rules extracted and the percentage of patterns covered.

Technique JCAP is essentially an implementation of the Connectionist Scientist Game within the framework of the RuleNet architecture. The Connectionist Scientist Game involves iteratively forming a hypothesis set of rules, testing these rules on exemplars, and then refining the rules to yield better coverage of the exemplars. The game is played until a satisfactory number of exemplars

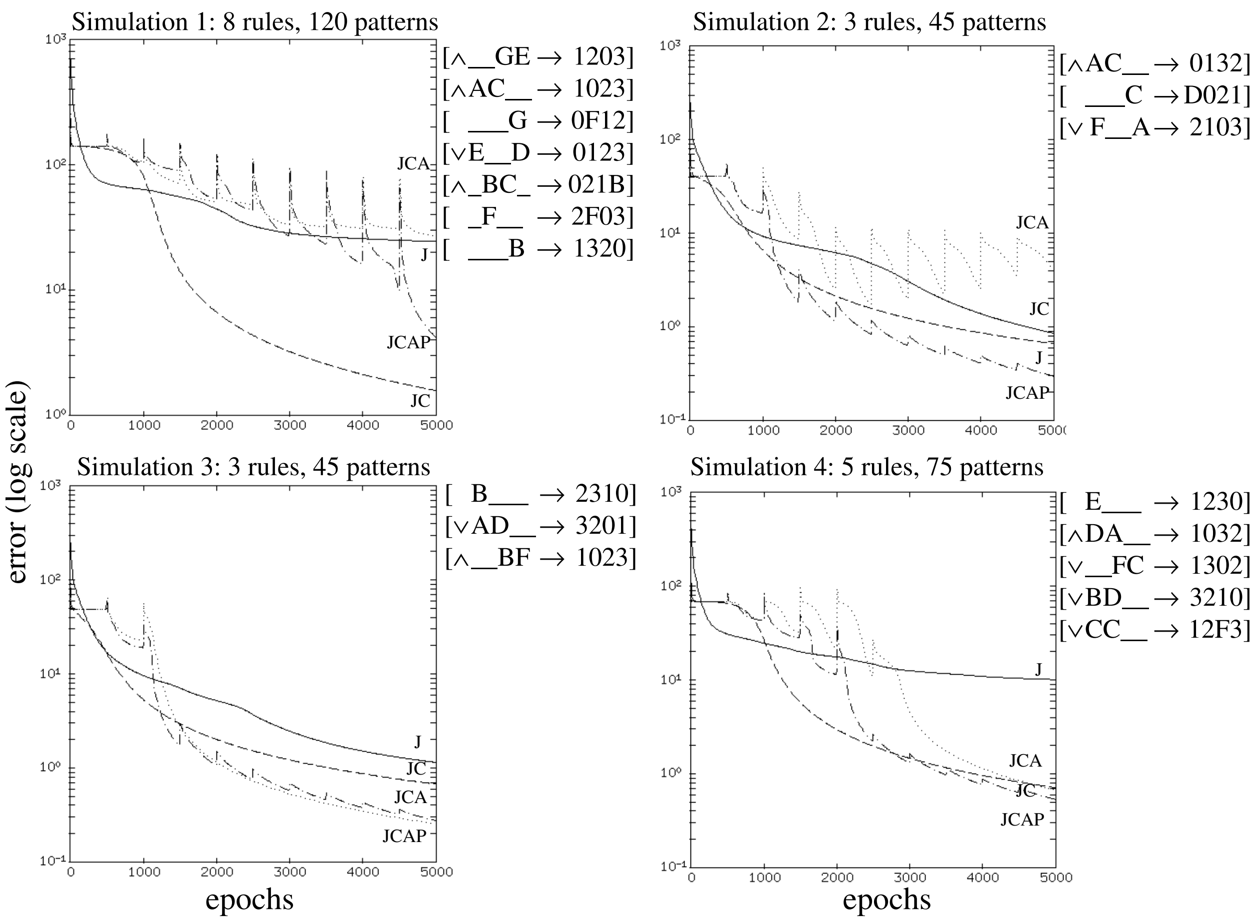

Figure 2

Average RuleNet performance on four simulations using learning techniques J, JC, JCA, and JCAP.
Key: J = Jacobs algorithm, C = constrain $\mathbf{A}_\alpha$ to diagonals, A = add new rules, P = project.

		Learning Technique							
		J		JC		JCA		JCAP	
Sim.	Target # rules	rules	%	rules	%	rules	%	rules	%
1	8	2.4	17	9.5	99	6	69	8	100
2	3	1.4	10	9.5	100	3.8	95	3	100
3	3	1.4	28	9	100	3	100	3	100
4	5	3.6	34	9	100	5.2	100	5	100

Table 1
Average number of valid rules extracted and percentage of patterns covered to within an error of .5.

is covered by the rules. Figure 3 shows a template of five rules, used to generate a set of 75 exemplars, and the evolution of the hypothesis set of rules learned by RuleNet over 2500 training epochs. As in previous simulations, projection was done every 500 epochs. At the first projection the initial rule is not syntactically valid. The second projection, at 1000 epochs, results in two valid rules which correctly mapped 3% of the patterns. Subsequent iterations result in a more refined hypothesis. It is interesting to note the metamorphosis of both the set of rules and individual rules. For example, the first valid rule learned, [∧ B_H_→1230], is modified between the projection at 1000 and 1500 epochs to [∨BD__→3210]. As new rules are added, the condition and/or action components of old rules may be modified to reflect a different set of exemplars for which they must be accountable.

We have argued that learning rules can greatly enhance generalization. Using technique JCAP, in all four of the simulations reported above, ruleNet not only learned a set of rules that correctly mapped the training set, it also learned the *exact* rule templates used to generate that set. In cases where RuleNet learns the original rule templates, it can be expected to generalize perfectly to any pattern that can be generated by those templates, as was shown in Table 1.

The degree to which generalization can be enhanced is clearly illustrated in a comparison of the performance of a standard three layer back-propagation network with the performance of RuleNet using the JCAP learning technique, as summarized in Table 2. In each of the four simulations, the back-prop network had the same input and output representation as RuleNet, with 15 hidden units per rule (simulations were run with 5, 10, and 15 hidden units per rule; generalization performance was best with 15). The learning rate used was .05, with a momentum of .9. The training sets were the same as reported in the RuleNet simulations and the numbers for RuleNet are taken from Table 1. The test sets represent the complete set of patterns that can be generated with the rule templates, excluding patterns that fire more than one template rule. During the back-prop simulations, outputs were processed by setting the maximum unit in each slot to 1 and all others to zero. The *cleaned up* outputs were compared to the targets to determine which were mapped correctly. Values in Table 2 represent the average over five runs with different initial weights. RuleNet's performance on the training set is equivalent to that of the three layer back-prop net. Both learn the training set perfectly. However, on the test set, RuleNet's ability to generalize is approximately 300% to 2000% better than the standard 3-layer network.

As the results in Figure 2 and Tables 1 and 2 indicate, playing the Connectionist Scientist Game allows RuleNet to discover a set of symbolic rules that describe a set of examples, and to do so more rapidly, reliably, and with greater power to generalize than the other methods examined.

Valid Rules Learned and Percent of Training Set Covered Perfectly

Template rules	epoch 1000	epoch 1500	epoch 2000	epoch 2500
[E___ → 1230]	[∧B_H_ →1230]	[∨BD__ →3210]	[∨BD__ →3210]	[∨BD__ →3210]
[∧DA__→ 1032]	[∧__FC →1302]	[∨ __FC →1302]	[∨ __FC →1302]	[∨ __FC →1302]
[∨__FC → 1302]		[∧ CC__ →1230]	[∨ CC__ →12F3]	[∨ CC__ →12F3]
[∨BD__→ 3210]			[∧ EA__→ 1230]	[E__ → 1230]
[∨CC__ → 12F3]				[∧ DA__→ 1032]
coverage of training set	3%	40%	64%	100%

Figure 3
Evolution of RuleNet's hypothesis set of rules over 2500 training epochs
while playing the Connectionist Scientist Game.

	Simulation 1		Simulation 2		Simulation 3		Simulation 4	
	% of patterns correctly mapped							
Architecture	train	test	train	test	train	test	train	test
RuleNet (JCAP)	100	100	100	100	100	100	100	100
3-layer back-prop	100	27	100	7	100	14	100	35
# of patterns in set	120	1635	45	1380	45	1380	75	1995

Table 2
Generalization performance of RuleNet compared to a standard 3-layer back-prop network with 15 hidden units per rule (120, 45, 45, and 75 hidden units for simulations 1-4, respectively).

Conclusion

We have proposed an iterative discovery process for connectionist networks called the Connectionist Scientist Game, and have explored an architecture, RuleNet, that plays the Connectionist Scientist Game. RuleNet is able to learn a set of explicit, symbolic rules *from scratch*. These rules are useful because they accurately and concisely characterize the domain from which training examples are drawn.

Although the end product of RuleNet is a set of rules, RuleNet benefits from intermediate stages of learning in which it is allowed to construct hypotheses that do not correspond exactly to rules. That is, being a neural network, RuleNet can represent a broader range of input/output mappings than those permitted by symbolic rule-based descriptions. We conjecture that RuleNet's exploration in this subsymbolic hypothesis space gives it an additional degree of flexibility that facilitates learning, even if the end product does not require this flexibility. It is for this reason that we feel that connectionist learning techniques offer great promise in domains even where the objective is to discover symbolic representations.

Our future work will concentrate on how to expand this architecture to more challenging and complex domains. One sense in which the symbol mapping domain we have considered is too simplistic is that no type/token distinction is made. The rules we have defined are formulated directly in terms of input symbols. However, much of the power of rules lies in their ability to apply to a general symbol type, rather than just to symbol tokens. For example, rather than inducing rules that simply enumerate a collection of tokens, such as **if** (*subject* = boy) **then** (perform action *x*), **if** (*subject* = man) **then** (perform action *x*), **if** (*subject* = woman) **then** (perform action *x*), it would be more useful to induce a single rule that covers a type of condition, e.g., **if** (*subject* = human) **then** (perform action *x*). This task requires that RuleNet learn not only the conditions and actions that form a rule, but also a language in which to redescribe the input symbols.

We are currently extending RuleNet to handle this additional level of complexity. With this extension, RuleNet will learn to both classify input tokens (e.g., classifying instances such as boy, man, and woman into the category human) and perform transformations of the input that are based on the induced categories. This involves subsymbolic processing—classification—as well as symbolic processing—executing condition-action rules. A unified framework that allows both types of processing should be a significant step toward an explanation of complex cognitive behaviors.

References

Booker, L.B., Goldberg, D.E., and Holland, J.H. 1989. Classifier Systems and Genetic Algorithms, *Artificial Intelligence* 40:235-282.

Fisher, D.H. 1987. Knowledge Acquisition via Incremental Concept Clustering. *Machine Learning* 2:139-172.

Jacobs, R., Jordan, M., Nowlan, S., Hinton, G. 1991. Adaptive Mixtures of Local Experts. *Neural Computation*, *3*:79-87.

McMillan, C. & Smolensky, P. 1988. Analyzing a Connectionist Network as a System of Soft Rules. In Proceedings of the 10th Conference of the Cognitive Science Society, 62-68. Hilsdale, NJ: Laurence Erlbaum and Associates.

Miikkulainen, R. & Dyer, M. 1988. Encoding Input/output Representations in Connectionist Cognitive Systems. In Proceedings of the 1988 Connectionist Summer School.

Rumelhart, D., & McClelland, J., 1986. On Learning the Past Tense of English Verbs. In J.L. McClelland, D.E. Rumelhart, & the PDP Research Group, *Parallel Distributed Processing: Explorations in the microstructure of cognition.Vol. 2: Psychological and biological models*, 216-271. Cambridge, MA: MIT Press/Bradford Books.

Sejnowski, T. J. & Rosenberg, C. R. (1987). Parallel Networks that Learn to Pronounce English Text, *Complex Systems*, 1: 145-168.

Smolensky, P. (1988). On the Proper Treatment of Connectionism. T*he Behavioral and Brain Sciences*. 11 (1).