Reinforcement Learning and the Frame Problem

Roberto Santiago[1], George G. Lendaris[2]
NW Computational Intelligence Laboratory, Portland State University, Portland, OR 97201
[1]Systems Science, robes@pdx.edu
[2]Systems Science and Electrical & Computer Engineering, lendaris@sysc.pdx.edu

Abstract: The Frame Problem, originally proposed within AI, has grown to be a fundamental stumbling block for building intelligent agents and modeling the mind. The source of the frame problem stems from the nature of symbolic processing. Unfortunately, connectionist approaches have long been criticized as having weaker representational capabilities than symbolic systems so have not been considered by many. The equivalence between the representational power of symbolic systems and connectionist architectures is redressed through neural manifolds, and reveals an associated frame problem. Working within the construct of neural manifolds, the frame problem is solved through the use of contextual reinforcement learning, a new paradigm recently proposed.

I. Background of the Frame Problem

The frame problem is one of the major stumbling blocks for realistically modeling cognition and for developing intelligent systems. The key issues revolve around the use of symbolic processing systems, a) for modeling phenomena associated with mind and brain, and b) as building blocks for the development of intelligent systems. While it originally arose as a significant challenge in classic artificial intelligence, the frame problem is now also discussed in the fields of cognitive science and philosophy of mind.

Without tracing details of its historical evolution, reinvention and multi-disciplinary variants, the basic issues leading to the frame problem can be summarized in the following way: a) If knowledge is to be represented in a symbolic manner, then as the amount of knowledge needed by an agent increases, so does the number of symbols needed to represent all that knowledge; b) as the number of needed symbols increases, the agent's efficiency decreases. Thus, a fundamental problem arises when we want to construct an intelligent agent that has even a modicum of common sense knowledge, for, as the corpus of common sense knowledge grows, the agent slows down. This yields a situation that is not only inconvenient, but unrealistic in comparison with the large volume of common sense knowledge that human beings efficiently use every day.

The original description of the frame problem, which was actually an attempt at its solution, focused on the management of fluents, a construct from the situational calculus [4]. A fluent is a state variable that describes the environment of an agent and whose value changes temporally. An example of a fluent would be *president (USA)*, which in 2000 equals W. J. Clinton and in 2001 equals G.W. Bush. In the situational calculus, all temporally changing states are described with fluents. For real world application, the number of fluents is quite large. The *frame* of the famous frame problem refers to an attempt to limit the number of axioms (rules) for determining which fluents do and do not change in any particular situation. An agent is said to always be operating within some particular frame, and every frame has associated with it axioms (*frame axioms*) for determining the changed and unchanged fluents. The idea is that the number of frame axioms per frame should be small in order to minimize processing cost. An unfortunate side effect of this constraint is that as the desired upper limit of frame axioms per frame decreases, the number of frames increases. The subsequent proliferation of frames brings with it the challenge of setting up rules for determining when an agent is to be in a particular frame. In essence, it trades one frame problem, the one focused on managing fluents, for another frame problem, the one focused on managing frame transitions.

The different variants and versions of the frame problem (such as the representation problem, the persistence problem, the prediction problem, etc.) all have the same basic structure. Namely, when the enumeration of knowledge into symbolic statements (rules, axioms, predicates, etc.) becomes large, some method is invoked to limit the number of symbolic statements considered at any one time. The varieties of solutions proposed so far have, unfortunately, fallen short. Either the proposed solution solves one frame problem by generating another, as in the example above, or the solutions are so limited in scope and application they fail to be a solution to the general frame problem.

II. Neural Networks, Systematicity and the Frame Problem

It has been said that artificial neural networks (NNs) are immune to the frame problem, based on the fact that NNs encode their knowledge into a set of weights of fixed cardinality, with the implication that there is always only a fixed cost to recall that knowledge, regardless of how much knowledge is encoded onto those weights. On the other hand, it is also argued that while NNs may avoid the frame problem in this way, they do so at the cost of reduced representational power compared to that available in symbolic systems. This latter argument is based on the notions of systematicity, brought forth by Fodor and Pylyshyn. Their cri-

This work was partially supported by NSF Grant ECS-0301022, and IEEE Walter Karplus Summer Grant, 2004.

tique, as we shall argue, hinges upon a limited understanding of connectionist architectures; and, moreover, that connectionist architectures do provide the representational power of symbolic systems. Alas, this representational power comes at the cost of incurring a frame problem; but, as we shall also argue, this frame problem comes in a form that is solvable.

The systematicity analysis of the representational capability of NNs was based on the then prevalent static, feed-forward architectures in the NN literature. Unfortunately, the focus on such architectures yielded misleading conclusions. Specifically, in their systematicity argument, Fodor and Pylyshyn [1] not only focus on feed-forward architectures but also invoke an interpretation that nodes in a feed-forward NN correspond to concepts, or at least to some set of symbols as would be used for symbolic representation (hereafter referred to simply as a concept). This method for knowledge representation is commonly referred to today as a semantic network and, we argue, is one of the most limiting uses of neural networks. None the less, in consideration of only 'semantic network' type usage of neural networks, Fodor and Pylyshyn correctly draw the conclusion that the connections between nodes are only interpretable as causal relationships, a crippling representational limitation. But, this is not the only issue with feed-forward semantic networks

The diagram in Fig. 1 shows a prototypical example considered in the systematicity argument. There are three nodes labeled *a priori* with the concepts Mary, John, and MaryLovesJohn. Assuming the network in Fig. 1 operates from left to right, the network architecture represents a causal relationship from the nodes labeled Mary and John to the node labeled MaryLovesJohn. *If* this were a connectionist architecture, the job of the network would be to learn the strength of the causal relationship. Notice that this network only has the ability to encode MaryLovesJohn and not JohnLovesMary without an additional node being defined. As such, the type of compactness and expressiveness one gets by *systematically* (hence, systematicity) recombining symbols is not to be found in this type of connectionist architecture with concept-labeled nodes.

While it is historically true that similar representational schemes appear in the works of well known connectionist researchers (e.g., [6]), they by far do not represent the breadth of knowledge representation approaches with NNs, a fact more understood today. In fact, recurrent connectionist architectures can *learn and develop their own representational schemes*. Of recent, Turing computability has been proven for discrete recurrent neural networks and super-Turing computability has been shown of analog recurrent neural networks. This means that a recurrent neural network is best thought of as a program which opens up a myriad of representational schemes. If a recurrent neural network can be thought of as a program then, of course, the training of a recurrent neural network can be thought of as constructing a program. Programs within the realm of Turing computability are certainly not limited to only causal representation of relationships between variables. Thus in consideration of recurrent connectionist architectures we overcome the first major criticism of Fodor and Pylyshin.

We agree that at the time, the 'semantic network' representational schemes posited were likely necessary, not only for early empirical work and comparison with symbolic systems, but also because little else was on offer for applying NNs to the problems and challenges of artificial intelligence and cognitive science. With the benefit of insights about recurrent connectionist architectures, we are now able to redress the above systematicity argument against connectionism. While we could again use the Turing computability argument for recurrent connectionist architectures to counter the main systematicity criticism, we believe a deeper analysis is needed. Specifically, redressing the systematicity criticism reveals a frame problem similar to 'managing frame transitions' discussed in Section I. However, finding the frame problem embedded within a connectionist architecture also reveals a clear solution path through a version of Reinforcement Learning. Thus we devote the next two sections to addressing systematicity and returning to the frame problem.

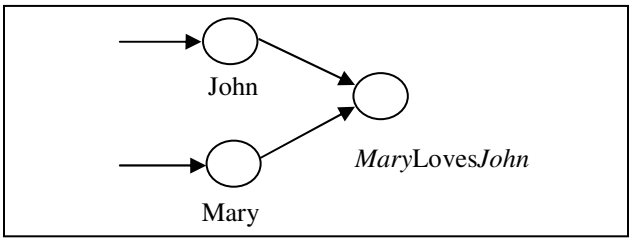

Figure 1. Basic Semantic Net representation.

III. Representation and Neural Manifolds

Methods for representation in neural systems are significantly more diverse than previously thought. One of the key architectures for enhanced representational capability is the recurrent neural network (RNN). In fact, analyzing RNNs provides a gateway to a more robust set of concepts with respect to connectionist architectures. Consider the recurrent multilayer Perceptron (RMLP) architecture (cf. [2]); in this architecture, the standard feed forward MLP is augmented with weighted connections between nodes where node output values from previous processing cycles may serve as inputs to nodes for the current cycle (a.k.a. *recurrent connections).* The recurrent connections of the RMLP enable this type of neural system to incorporate temporal information, and further, they enable the *development of representational schemes* during the process of training. The signal values that appear as inputs to the nodes via the resulting recurrent connections may be seen as a representation of the information content of data presented to the network in the past. For clarity we will refer to this representation as a *dynamic representation* to distinguish it from other representations that arise in the use of neural systems.

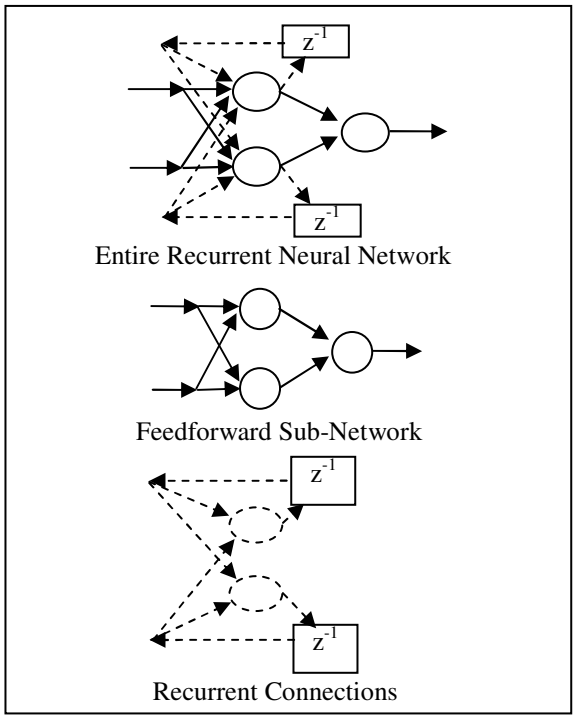

Figure 2. Components of Recurrent MLP (RMLP)

The role of these dynamic representations in RMLPs is to encode the relative use of current and past input data. To clarify, consider the RMLP structure as comprising two distinct parts: the feed forward network and the recurrent connections (see Fig. 2). The feed forward network on its own only has the ability to process new inputs to the RMLP (i.e. just the current input data), and further, during operation it is a static structure. But now consider the dynamic representation generated by the recurrent connections. Mechanically, this representation may be thought of as shifting the bias settings on each of the feed forward nodes. The shifting of the bias settings has the important qualitative effect of changing the feedforward network from a static structure to a dynamic structure. More specifically, one can think of the RMLP as a *sequence* of feed forward MLPs. This sequence of MLPs is selected from a *set* of MLPs where each MLP is exactly the same, except for its bias settings. Therefore, each MLP within the set is uniquely identified by its bias settings. During each step of operation, the recurrent connections of the RMLP serve the role of generating bias settings and, as a result, selecting an MLP from the set. Thus we recast the RMLP into two components: 1) a set of MLPs each uniquely identified by their bias settings and 2) a *network selector* (i.e. the recurrent connections which generate a set of bias settings during each step of operation).

The first component, the set of MLPs, can be thought of as a manifold. While manifolds entail deep mathematical properties, it suffices for the purposes of this paper to think of a manifold as comprising the following: 1) a set of elements, S, and 2) a coordinate system (a one-to-one mapping from S to R^n that specifies each element in S via a vector of

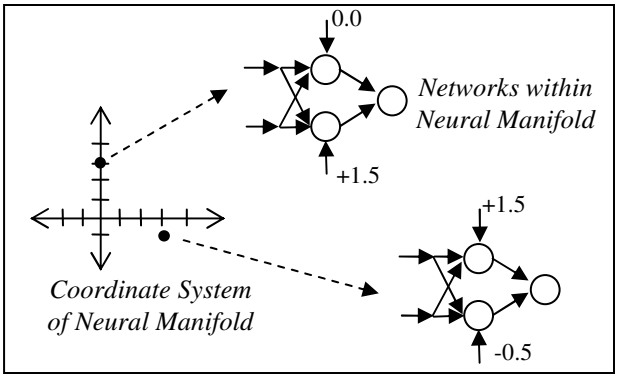

Figure 3. Example for description of Dynamic Parameters.

n real numbers, a.k.a. the *coordinates* of the element). For the RMLP the set of elements is the collection of feed-forward neural networks (NN) defined by varying the bias parameters of the hidden nodes and holding all other weights static. The coordinate of each feed-forward NN is the vector of bias settings. Thus the set can be considered a manifold of neural networks, or more simply a *neural manifold*. Fig. 3 presents a visualization aid for this concept.

The definition of neural manifold is generic enough that we may consider any type of NN architecture not just feed-forward when constructing the set of elements. What is important, though, is that we designate some set of parameters for the chosen architecture as static and some other set of parameters as defining the coordinates for each element of the set. We may refer to each group of parameters as *static parameters* and *dynamic parameters*, respectively. In the example of the RMLP, the dynamic parameters are the biases of the hidden nodes and the static parameters are all the other weights on the feed-forward network. Notice, though, that the weights of the network selector are not in either of these two parameter sets. In the RMLP example this would be the weights of the recurrent connections. Also notice that the weights of the recurrent connections can themselves be thought of as a simple linear network. We can generalize the definition of the network selector to be an NN of any architecture. What is important is that at every step of operation it chooses a network from the manifold to use for information processing, that is, it sets the dynamic parameters.

The neural manifold and the network selector concepts provide already a rich foundation for representation. In general, we may consider any NN a program, not just RNNs. So a manifold can be thought of as a library of programs. Thus, the network selector is a program for selecting programs. These concepts will be leveraged as we return to the systematicity criticism and subsequently the frame problem. It is important at this point to remark that an RMLP is a very simple example of a neural manifold and network selector and that the remainder of this paper will deal with the concepts of neural manifolds and network selectors and only refer back to the RMLP as an example to add clarity.

IV. Neural Manifold and the Systematicity Argument

Recall that the systematicity argument was directed at the expressive capabilities of single feedforward neural networks, which we agree have a limited repertoire of relationships they can express among primitive symbols. We submit, though, that through the agency of neural manifolds and network selectors the story is different. Consider a neural manifold whose elements are RNNs, per say a *recurrent neural manifold (RNM)*. Because RNNs can be thought of as programs we can indeed imagine constructing an RNM with at least one program that takes the primitives *John* and *Mary* and encodes *JohnLovesMary* and at least one program that takes the same primitives and encodes *MaryLovesJohn*. Due to space limitations we cannot go into a detailed explanation but at this point simply assert this is true. By definition, these two RNNs are exactly the same in architecture and are accessible via their respective coordinates. Thus, contrary to the conclusions of the systematicity argument, it is not necessary to specify an architecture with multiple specialized nodes; rather, the *same* RNN architecture may be used but only with differing dynamic parameter settings. So for the same node count, we are in a position to get much more representational power, and in principle, at the same level of representation as AI's systematic manipulation of symbols.

The above establishes that RNMs provide expressiveness on par with symbolic processing, and thus takes the wind out of the sails of the systematicity argument against NNs. So far, so good. What we have yet to address is a principled way to construct a useful manifold, through setting of the static parameters, and an optimal network selector, through setting of the parameters on the network selector. We know that in the case of the RMLP this is possible through the use of backpropagation through time (BPTT). By way of foreshadow, we shall see how BPTT is a form of reinforcement learning and that reinforcement learning is the general method for setting the static parameters of the manifold and the parameters of the network selector. Before this, though, we shall see that constructing an optimal network selector is equivalent to solving the frame problem.

Recall from Section I, the difficult issue of managing fluents for the original frame problem was solved at the cost of creating another frame problem – that of managing frames, which again was not directly solvable. In a sense, we have a similar situation here. We draw the parallel by equating the notion of frame with a point on our RNN manifold, and then posit that the task of selecting an RNN from the manifold is a (type of) frame problem. An important consequence of this view is that once we develop a solution for constructing an optimal network selector, we will have the basis for a solution to the general frame problem.

V. Neural Manifolds, Frames and the Solution

As suggested above, we posit that conceptually each point on a neural manifold corresponds to a frame. Frames, as originally proposed, are defined around a particular action. This frame action can more conventionally be thought of as a program (an *action program*). So the neural network corresponding to each point of the manifold takes the place of the action program (an *action network*, which performs the frame action).

As described earlier, each frame has associated with it frame axioms; these determine the update of the fluents. If the AI frame approach had worked well, the subsequent information processing would have rapidly resulted in a) update of fluents, b) calculation of the next action, and c) determination of the frame within which the action should be considered. In principle, this information processing set could be wrapped up into a single program (a frame program). This frame program could in principle be replaced by a neural network (a *frame network*). Indeed, the network selector plays the role of the frame network. In the case of the RMLP the recurrent connections form the network selector and thus the frame network where the frames are the feed-forward networks of the manifold. Recalling that RMLPs are trained with BPTT and, as we will show, BPTT is a form of reinforcement learning, the frame network can be constructed through reinforcement learning. We will describe this connection in further detail in the next section.

Before moving on, though, it is important to highlight some differences which exist between frames and neural manifolds. The frame problem from AI starts with the concept that all actions are explicitly programmed by an engineer, i.e., a tailored program is required for each frame. The concept of neural manifolds, on the other hand, starts by defining a large set of networks within which already exist all the programs that might be needed by the agent. Generally, the neural manifold may be thought of as comprising a large collection of "frame-relevant" programs, an idea we will explore shortly. The critical point here is that it is not necessary for all points in the manifold to correspond to *useful* programs. If the manifold contains both useful and non-useful programs, it will be the frame network's task to deal with this issue.

VI. Frame Networks and Reinforcement Learning

Conceptually, the RNN structural form derives its computational power by its ability to merge information to and from temporally disparate processing steps. A recurrent connection from one node to another takes the output value of the "from" node (that was calculated during the previous time increment) and passes it to the input of the "to" node (at the beginning of the new time interval). Once a value has been input to the first node in a structure with recurrent connections, the value will continue to influence the inputs appearing at each node in the path for all time hence. The actual contribution at any time t_k will depend on the weights of the various connections.

An interesting thing one can do with this type of computational structure is to adjust the meaning of past-present-future. The usual way of thinking about the computations

made at a node at some time t_k is to consider that all the data coming to it are based on past and present data. But, by adjusting the zero reference point on the perceiver's clock, the same computations can be treated AS IF they include data from the "future". Of course, when doing this, care needs to be taken to keep all aspects of the resulting representation synchronized.

Thus if one were training a recurrent neural network to be a policy, the computations can be set up to act as if performance information from one time step is communicated back to previous time steps. This concept is incorporated in (at least) two of the modern Reinforcement Learning procedures: the Adaptive Critics and BPTT.

The Adaptive Critic class is based on approximating Bellman's Dynamic Programming. This method defines a primary utility function U that calculates a "cost" associated with an action and subsequent change of state of the plant being controlled. Dynamic Programming is a method for designing a policy that minimizes the total cost that accrues over an entire trajectory of the plant from a starting point in its state space (this is called "cost-to-go", and is designated as J). The frame network (a.k.a. network selector) can be thought of as a policy for selecting frames (i.e. selecting networks from the manifold). The adaptive critic method uses the clock trick mentioned above for only one step into the future. The rest of the future is dealt with via a critic NN whose job it is to directly estimate the value of J (or its gradient). The policy design at each iteration of the process is based on the current estimate of J (or its gradient). As the estimate of the J value (or its gradient) is improved, so is the design of the policy.

The BPTT method approaches the policy design task from the other direction, by making substantial use of the clock trick mentioned above. Instead of using a critic to estimate J, it adds up the various values of U that are in the "pipeline" during the evolution of the process, and by using the clock trick mentioned above, acts as though it is calculating J by adding up "future" values of U directly. It turns out that BPTT, in particular its truncated form, has become a popular method for training the weights of recurrent neural networks [5]. Augmentation of this algorithm with a form of extended Kalman filtering has proven extraordinarily powerful, and has been used to create RNNs which have been deployed to solve very complex tasks in consumer vehicles [5].

Getting back to the policy design task, feeding back information to previous time steps allows the output of the RNN from one time step to be adjusted so as to maximize the performance measurement for subsequent time steps. The key here is that BPTT allows the output from an RNN at one time step (say *t*) to be adjusted so as to maximize performance measurements for subsequent time steps (say t+1 through t+n). In the language of Adaptive Critics and Approximate Dynamic Programming mentioned above, the output of the RNN at time t is adjusted so as to maximize (minimize) the performance measurements for times $t+1$ through $t+n$. That is, the output of the RNN at time t is adjusted so as to maximize/minimize $\sum_{k=1}^{n} U(t+k)$. This is equivalent to the cost-to-go measure of Dynamic Programming, where $J(t) = \sum_{k=0}^{n} U(k)$.

If we denote a weight of the RNN as *w*, then BPTT calculates $\frac{\partial J(t)}{\partial w}$, hence the characterization above that BPTT implements a form of RL. We thus substantiate the claim that BPTT is a form of RL and, given all of the analysis up to this point, TL can be used to construct frame networks. In fact, in general RL can be used to construct a frame network and solve, in general the frame problem. We briefly review in the next section a new RL paradigm from which this assertion stems.

VII. Contextual Reinforcement Learning

The following discussion is based on recent work wherein the idea of "context" is used to extend application of the RL paradigm into what we have termed Contextual Reinforcement Learning (CRL). The discussion up to this point has indirectly described the major components of CRL; further details, analysis and implementation examples can be found in [3] and [7]. The major difference between the above discussion and CRL is use of the term 'frame' instead of the term 'context'. The term context refers to a larger concept than that of the term frame, in the sense that a frame focuses just on the actions of the agent whereas context includes aspects of the agent's environment as well. In a policy design setting, for example, the plant to be controlled is the context for the design, and the associated optimal policy design (actor parameters) is to be selected from the manifold. More generally, every point on the neural manifold defines a set of behaviors for the agent (e.g., optimal policy designs), and the aim of CRL is to translate all information gained from the environment into an optimal behavior selection from the neural manifold.

Fig. 4 provides for a general description of CRL, and brings together the significant concepts discussed so far. The bottom most box is the manifold we have already described. The set of networks in this manifold are indexed via their dynamic parameters, which serve as the manifold's coordinate system. In our previous RMLP example, it is the set of feed forward neural networks defined by varying the biases for the nodes receiving recurrent connections, holding all other parameters static. The coordinate system (indexing mechanism) is the domain of the dynamic parameters.

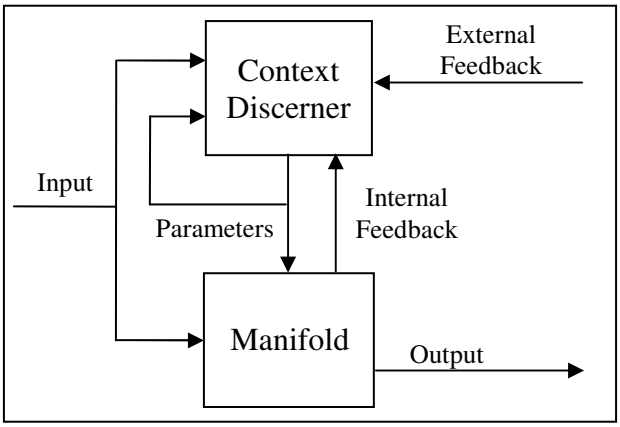

Figure 4. Basic Structure of
Contextual Reinforcement Learning

Also in Fig. 4, we see that the dynamic parameters are being set by the Context Discerner (CD). In general, we may consider the CD as implementing the network selector, that is, a 'policy' for selecting networks from the neural manifold. The path through parameter space is based on the stream of inputs and feedbacks the agent receives internally and from the environment. In our previous RMLP example, the CD is the set of recurrent connections – i.e., the CD only receives internal feedback. In the CRL experiments performed so far, the CD has been implemented as an MLP and an RMLP. In principle, the CD can be any form of machine learning algorithm.

Among the tasks of the (trained) CD is to "know" what combinations of inputs and feedbacks require action from the CD to pick another NN from the manifold, and further, it needs to know how to accomplish the selection in a time optimal manner. This entails knowing which changes in the environment are most important, and what they imply in terms of the NN (a.k.a. the program) to be selected. A partial demonstration of this capability is given in a companion paper at this conference [3].

VIII. Conclusion

The frame problem has been around for some four decades, and its resolution has long been overdue. It is interesting that in some sense the answer to this problem has existed in the basic form of RMLPs for some time without recognition of the fact. The majority of this paper has explored the nature of the frame problem, how it finds its analogue in connectionist architectures and, ultimately, how it might be solved, in general, through the use of reinforcement learning. Along the way, significant effort has been spent on redressing criticisms of the representational capability of neural networks. These criticisms long ago pushed away from fruitful dialogue connectionist architectures as being significant building blocks for artificial intelligence (as well as for models of mind and cognition). We believe that the research presented here takes a significant step in addressing these long standing criticisms, as well as demonstrating how some of the hardest problems of AI can now indeed be addressed through connectionism.

The work reported here represents but a small sampling of what we believe to be possible with contextual reinforcement learning and neural manifolds. Indeed, early results with CRL seem incredibly promising, but at the same time significant challenges remain. Still, the CRL architecture has the potential of providing a greater understanding of how human beings are able to use information learned in one situation across a large variety of related situations. Namely, it seems that the context discerner is a policy for looking at both, external cues (from the environment) and internal cues (from the agent state) in order to generalize the application of programs (of the neural manifold) in an optimal manner. Moreover, the neural manifold seems a powerful method for describing how memory is organized both at a cognitive and neural level. From an engineering point of view, CRL provides a new method for enabling high capacity, long term learning and memory for an artificial agent.

In closing, it is our hope that this and similar research might provide a renewed foundation for approaching again the largest challenges of AI.

References

[1] Fodor, J.A. & Z.W. Pylyshyn (1988), "Connectionism and cognitive architecture: a critical analysis," in *Connections and Symbols*, Pinker, S. and Mehler, J. (eds.).

[2] Haykin, S. (1999), *Neural Networks, A Comprehensive Foundation*, Prentice Hall.

[3] Holmstrom, L., R. Santiago, & G.G. Lendaris, 2005, "Designing an Adaptive Systems Identifier Through Reinforcement Learning," Submitted to IJCNN'2005.

[4] McCarthy, J. & P.J. Hayes (1969), "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in *Machine Intelligence 4*, ed. D.Michie and B.Meltzer, Edinburgh: Edinburgh University Press, pp. 463-502.

[5] Prokhorov, D., G. Puskorius & L. Feldkamp (2001), "Dynamical Neural Networks for Control." in *A Field Guide to Dynamical Recurrent Networks,* J. Kolen and S. Kremer (Eds.), IEEE Press.

[5] Pylyshyn, Z.W. (ed.) (1987), *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*, Norwood, NJ: Ablex.

[6] Rummelhart, D.E., J.L. McClelland, & the PDP Research Group (1986), *Parallel Distributed Processing*, Vols. 1 & 2, MIT Press.

[7] Santiago, R. A. & G. G. Lendaris (2004). "Context Discerning Multifunction Networks: Reformulating Fixed Weight Neural Networks." *Proc IJCNN-04*, Budapest, Hungary, IEEE Press.

[8] Santiago, R. (2005), "Discerning Context through Reinforcement Learning and Recurrent Neural Networks" Technical Report, NW Computational Intelligence Lab, Portland State University, www.nwcil.pdx.edu.

[9] Siegelmann, H. T.(1999), Neural networks and analog computation: beyond the Turing limit, Birkhauser Boston.