On-Line System Identification Using Context Discernment

Lars Holmstrom[1], Roberto Santiago[2], George G. Lendaris[3]
NW Computational Intelligence Laboratory, Portland State University
Portland, OR 97201
1,2: Graduate Students, Systems Science Ph.D. Program [1]larsh@pdx.edu, [2]robes@pdx.edu
3. Professor of Systems Science and Electrical & Computer Engineering, lendaris@sysc.pdx.edu

Abstract: Mathematical models are often used in system identification applications. The dynamics of most systems, however, change over time and the sources of these changes cannot always be directly determined or measured. To maintain model accuracy, it is desirable to design system identifiers that can adapt to these dynamical shifts. We use reinforcement learning to train an agent to recognize dynamical changes in a modeled system and to estimate new parameter values for the model. The subsequent actions of this agent are characterized as "moving" the parameterized model on an optimal trajectory in model parameter space. It is found that this method is capable of quickly and accurately discerning the correct parameter values.

I. Introduction

System identification is central to many applications (e.g., controls, system analysis, predictive modeling), and central to system identification is the development of models. Parametric models are typically used, wherein the model's structure is crafted to blend requirements associated with aspects of the system that are of interest to the user of the model, and those associated with *a priori* knowledge of the system's underlying "structure". Coefficients associated with the various model components are called parameters; distinct combinations of parameter values typically correspond to a different system. To illustrate, consider a dynamic system where the system's behavior can be written as a series of differential equations (systems governed by physical laws are ideal candidates for this type of modeling). For such a system, the model structure is typically determined by the physical relationships of the components of the system, and the model parameters often correspond to physical attributes of the system (e.g. moments of inertia, masses, friction, and spring constants). Such models are not limited to physical systems, however. This type of modeling is also common in economics, psychology, the social sciences, and more.

After defining the structure of one's model, values for each of the parameters must be determined to yield model behavior that closely matches that of the system being modeled. Some parameters may be easy to directly measure, like the length of a visible component of a mechanical system. Others may be more difficult to directly measure, such as the average effect of disposable income on physical health in a socio-economic model. Still others may not be directly measurable at all, such as the friction from a specific bearing in an assembled piece of machinery.

Gradient descent is a method commonly used to determine values for model parameters that are not directly accessible in the system being modeled. This method defines a space whose dimensions correspond to the various parameters, plus an extra dimension to represent some criterion function (CF), for example, the squared difference between model output and corresponding system output. This space is searched for the parameter values that minimize the CF, where the search employs the local gradient to determine the direction and size of the next "step" in the parameter space. This process is iterated over a broad range of the system's possible behaviors, with the goal of converging on the parameter values that, on average, best fit the behavior of the model to that of the system.

While effective, gradient descent methods are unfortunately plagued with a number of problems, a well-known one being the potentially long time it may take for the process to converge. In addition, there is no guarantee in the gradient descent method that the path taken through parameter space during the search (estimation) process is a direct one, nor that the path taken will avoid estimates that could be problematic for application related calculations and/or decisions that are based on these estimates.

One approach used to address both of these issues is to perform the model parameter estimation process off-line, in a non-critical setting, and subsequently employ the completed model in the application. Unfortunately, there is no guarantee that this model will continue to be the best fit model as time progresses, as properties of real world systems typically change over time. In some cases, these changes are the result of fundamental changes to the interactions present among the components of the system, and to reflect such changes, the underlying structure of the model must be changed. A more common scenario, however, is that the types of interactions remain intact but the magnitude of these interactions change. For example, in a mechanical system springs wear out and friction increases. In such cases, the model can be adapted by re-estimating (finding new values for) the model parameters that correspond to this type of change in the system.

This work was partially supported by NSF Grant ECS-0301022

In an on-line application where accurate modeling is time critical, however, gradient descent is typically not a viable option for re-fitting the model; the process simply takes too long. Further, there is no guarantee that the search for better parameter values will not result in worse models along the way. Finally, it may not be possible in an on-line setting to iterate the gradient descent search over a wide enough range of system behaviors, and the process may thus yield a sub-optimal model.

Some researchers have addressed this issue by pursuing the development of system identifiers that learn to adapt to new dynamic behaviors of the system being modeled. This has generated interest in recurrent neural networks which can rapidly recall previously learned dynamic behaviors without needing to be retrained in order to do so. These networks can then adapt to novel situations by generalizing among these previously learned behaviors. It was recently shown in [3] that a standard recurrent multilayer Perceptron (RMLP) could be trained using back propagation through time with extended Kalman filtering to create a network that performs rapid system identification. The latter task is accomplished by the trained network without needing to modify any of its weights. These networks are known under the rubric of Fixed Weight Neural Networks (FWNNs). More recently, it was shown in [4] that the weight values developed in some FWNNs effectively define two distinct sub-NNs: one serves as a parameterized model, referred to as a multifunction network, and the other serves to set this model's parameters, referred to as a context discerner. That result has since been extended to a more general method, called context discerning multifunction networks (CDMN), and has been generalized to a theory of Contextual Reinforcement Learning [5]. The research reported here is an application of the CDMN method.

DEFINITION: For the present purposes, each different instantiation of dynamics by the system is called a different *context*. The agent's ability to detect shifts in the system's dynamics is called *contextual awareness*. Once the context shift has been detected, the agent's ability to recognize the new context (e.g., by determining a new model whose behaviors match those of the system – by re-estimating model parameters in our current scenario) is called *context discernment*. Once the discernment is accomplished, the agent may use this information as needed, e.g., to directly select an optimal controller that corresponds to the newly discerned system dynamics. We propose calling a controller with such discernment capability embedded into it a *contextually aware* controller.

II. The Experiment

For a demonstration of this context discernment methodology, we use a version of the benchmark pole-cart system. We assume that the length and mass attributes of the pole may change (perhaps drastically) at various points in time, but cannot be directly measured for the purpose of fitting a model of the system. Through application of reinforcement learning, we train an agent to detect the effects of changes to the length and mass of the pole by comparing deviations in the state trajectories between the available pole-cart model and the actual pole-cart system. Using only these observations, this agent is then able to output an adjustment to our current values of these parameters in an iterative process that ultimately converges upon the actual values of the pole length and mass. In the present paper, we discuss only the parameter discernment process, not the process of designing a contextually aware controller, for the pole-cart system. The latter process will be the subject of a forthcoming paper.

III. Methods

Fig. 1 indicates the basic architecture of the context discerner used in the experiments for the present work. The variables subscripted with an 'A' represent the *Actual* pole-cart system being controlled; those subscripted with a 'D' represent *Discerned* values resulting from the context discernment process. At time t, the pole-cart system is in state $R_A(t)$. The current estimated values of the mass and length of the pole in the pole-cart system (the discerned context) constitute the vector $C_D(t)$, and the actual values constitute the vector $C_A(t)$. [Note: the training process never requires explicit knowledge of $C_A(t)$.] The current state information $R_A(t)$ and a control $u(t)$ are applied to the model (resulting in a transition to state $R_D(t+1)$) and to the pole-cart system (resulting in a transition to state $R_A(t+1)$). $D(t)$ corresponds to the difference between $R_D(t+1)$ and $R_A(t+1)$. If $D(t)$ is non-zero, this indicates that our current model is incorrect and needs to be adjusted. The ensemble of data comprising $R_A(t)$, $u(t)$, $D(t)$, and $C_D(t)$ are presented to a Context Discerning Network (CDN). The CDN then provides an adjustment, $\Delta C_D(t)$, to $C_D(t)$ such that $C_D(t+1) = C_D(t) + \Delta C_D(t)$. The goal of the CDN is to iterate the above process and to provide a sequence of $\Delta C_D(t)$s such that $C_D(t)$ ultimately con-

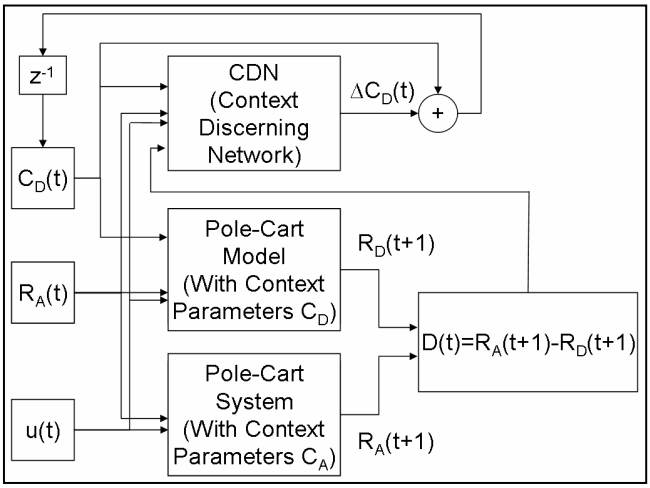

Figure 1: Basic Architecture for Contextual Discernment

verges to $C_A(t)$. If the structure of our model is chosen correctly, the behavior of our model will accurately match the behavior of the pole-cart system when $C_D(t) = C_A(t)$. When this occurs, we say we have discerned the values of $C_A(t)$.

The state vector $R_A(t)$ of the pole-cart system comprises the horizontal displacement of the cart from a specified reference point ($x(t)$), the horizontal velocity of the cart ($dx(t)/dt$), the angular displacement of the pole from the upright position ($\theta(t)$), and the angular velocity of the pole ($d\theta(t)/dt$). The control signal, $u(t)$, is the magnitude of the force applied to the cart parallel to the track. Positive control signals indicate a force to the right and negative control signals indicate a force to the left. The context vector contains parameters corresponding to the mass and length of the pole (cf. Section IX).

This architecture provides a framework in which the key components of context discernment as defined above are present. It handles contextual awareness by comparing $R_D(t+1)$ and $R_A(t+1)$ at each step of the process. This, in effect, is a measure of how our model's state trajectory deviates from that of the actual pole-cart system. It handles context discernment by the iterative corrections to C_D by the CDN. What remains, however, is a description of how to train the CDN to perform this task.

IV. Reinforcement Learning

The method we use for training the CDN is based on Dual Heuristic Programming (DHP) [2], one of the adaptive critic versions of reinforcement learning. Reinforcement learning [6] is a paradigm in which an agent learns a policy – a mapping of state information to actions – that maximizes a reward or minimizes a punishment over time. One family of algorithms that belong to this paradigm learns approximately optimal policies through application of the underlying definitions and recursion equation of Dynamic Programming [1]. When the agent applies an action $u(t)$ (this may be a control signal or a decision) and transitions a plant from state $R(t-1)$ to state $R(t)$, a reward (or punishment) $U(t)$ occurs. In Dynamic Programming, the latter is called the primary utility, and a secondary utility function, $J(R(t))$, or more simply $J(t)$, is defined as:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k) \qquad (1)$$

The value $J(t)$ is often called the "cost to go", and refers to the cost that will be accumulated in going from the state at current time t to the end point of the plant's trajectory in state space, under influence of the current control policy. The γ term in the above equation ($0<\gamma\leq 1$) allows discounting of future costs.

A. Adaptive Critic Methods

The methods known as Adaptive Critics belong to the Reinforcement Learning class. In Adaptive Critics, there is a *critic agent* whose purpose is to output an estimate of $J(t)$, given current state and/or action information. The critic is adaptive in the sense that it learns to better approximate the J surface ("plotted" via an extra dimension for J in state space) as the iterative, two-loop, learning process proceeds. This J surface information is used to aid the process of developing an optimal policy within the *action* agent (controller).

B. Dual Heuristic Programming

Dual Heuristic Programming (DHP) is an Adaptive Critic method. A key distinction of DHP from other Adaptive Critic methods is that the critic approximates the *gradient* of the J surface rather than the J surface itself. This gradient is referred to as $\lambda(t)$. The basic architecture of the DHP design consists of five components: an actor (action agent), a critic, a plant, a plant model, and a utility function.

The actor component provides a mapping of the state $R(t)$ to the action $u(t)$. The plant and plant model provide a mapping of the state $R(t)$ and action $u(t)$ to the state $R(t+1)$. The critic component provides a mapping of the state $R(t)$ (and sometimes $u(t)$) to the value $\lambda(t)$. The utility component provides a mapping of $R(t)$ (and sometimes $u(t)$) to $U(t)$.

During training, these components work in conjunction to update the actor and critic components. The actor component is updated with the objective of maximizing (or minimizing) $J(t)$. The critic is updated with the objective of approximating $\lambda^*(t)$, corresponding to the gradient of $J^*(t)$, which in turn corresponds to the optimal actor design.

V. Training the CDN

The method we use to train the CDN borrows heavily from applications of Reinforcement Learning in the domain of controller design. As mentioned above, the goal of Reinforcement Learning in controller design is to learn a control policy (a mapping of plant state $R(t)$ to control $u(t)$) that minimizes the utility function, $J(t)$ – cf. Eqn. (1). By redefining a few variables, we can re-purpose this framework and apply this same methodology for the task of context discernment. Let $C_D(t)$ be the state of the plant that is to be "controlled" at time t, and the corresponding *control* at time t to be $\Delta C_D(t)$. With these definitions, $R_{plant}(t) = C_D(t)$ and $R_{plant}(t+1) = C_D(t) + \Delta C_D(t)$. Next, define the primary utility $U(t)$ to be $D(t)^2$; the squared error between $R_A(t+1)$ and $R_D(t+1)$. Looking back at Figure 1 and considering this new framework, the CDN plays the role of controller, and the summing node that follows it plays the role of plant (a dynamics-free plant).

VI. Discernment

The goal of Reinforcement Learning in this application is then to design an agent (the CDN) that implements a policy (mapping of $C_D(t)$ to $\Delta C_D(t)$) that minimizes the summed values of $D(t)^2$ over time (cf. Eqn (1)) as the model parameters are adjusted. Not only does the CDN learn a policy that enables it to discern the current system context (that is, determining a new model whose behaviors match those of the

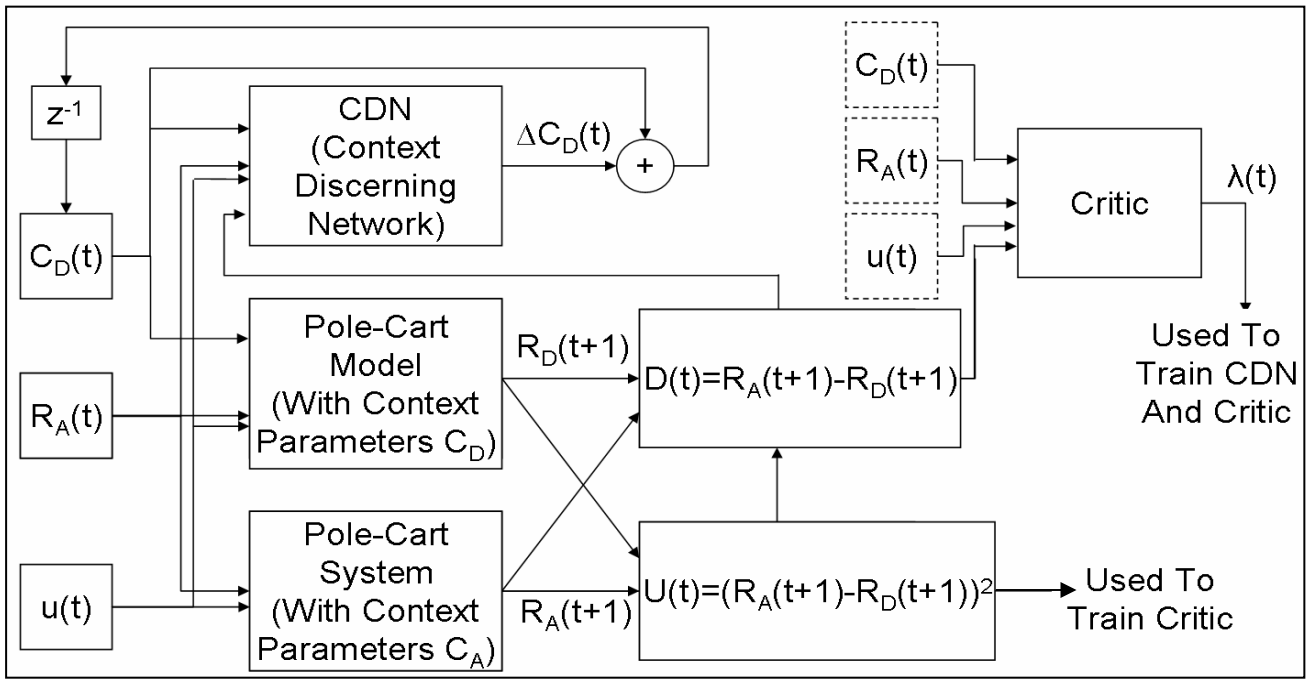

Figure 2: Architecture For Training CDN

system), but it does so in a way that meets our defined optimality criteria.

Fig. 2 shows the architecture used for training the CDN. In a typical application of DHP, the following error signals are used for training the controller and critic components during each iteration of the training process:

$$Error_{Critic} = \left(\frac{\partial U(t)}{\partial R(t)} + \gamma \left(\hat{\lambda}(t+1) \left(\frac{\partial R(t+1)}{\partial R(t)} + \frac{\partial R(t+1)}{\partial u(t)} \frac{\partial u(t)}{\partial R(t)} \right) \right) - \hat{\lambda}(t) \right)^2 \quad \textbf{(2)}$$

$$Error_{Controller} = \frac{\partial R(t+1)}{\partial u(t)} \hat{\lambda}(t+1) \quad \textbf{(3)}$$

With the substitutions $R(t) \rightarrow C_D(t)$, $u(t) \rightarrow \Delta C_D(t)$, and $C_D(t+1) = C_D(t) + \Delta C_D(t)$, these training signals simplify to:

$$Error_{Critic} = \left(\frac{\partial U(t)}{\partial C_D(t)} + \gamma \left(\hat{\lambda}(t+1) \left(I + \frac{\partial \Delta C_D(t)}{\partial C_D(t)} \right) \right) - \hat{\lambda}(t) \right)^2 \quad \textbf{(4)}$$

$$Error_{Controller} = \hat{\lambda}(t+1) \quad \textbf{(5)}$$

$\frac{\partial \Delta C_D(t)}{\partial C_D(t)}$ is obtained by back propagating $\Delta C_D(t)$) through the CDN. $\hat{\lambda}(t)$ and $\hat{\lambda}(t+1)$ are estimated values for the present and future gradients of the secondary utility function $J(t)$ and are output by the critic. $\frac{\partial U(t)}{\partial C_D(t)}$ is calculated using the model Jacobian with respect to the current discerned context variables.

VII. Training Parameters

For the training process, both the controller and critic networks have feed forward neural network architectures with 7 inputs, 1 hidden layer with 12 nodes, and 2 outputs. Hyperbolic tangent activation functions are used in all cases.

For each training epoch, the mass and length of the pole (the components of $C_A(t)$) were randomly set in the pole-cart system. The range for the mass was [0.02, 1.5] kg and the range for the length was [0.02, 1.5] m. This range was ample to provide a wide range of dynamic behavior from the system.

For each training step of each epoch, the pole-cart system and the model were placed in a randomly selected point in state space ($R_A(t)$) and a random control ($u(t)$) applied. The ranges for the random selection were: $x(t)$=[-1,1] m, $dx(t)/dt$=[-1,1] m/s, $\theta(t)$=[-π/4, π/4] rads, $d\theta(t)/dt$=[-1,1] rads/s, and u(t)=[-5,5] N. The pole-cart system then transitions to $R_A(t+1)$ and the model to $R_D(t+1)$. The difference between these resulting states is then used by the CDN for the calculation of $\Delta C_D(t)$. The error signals for the critic network (Eqn. 4) and the CDN (Eqn. 5) are then calculated and both networks' weights are adjusted. For the training of the CDN discussed below, 10,000 epochs of 25 training steps each were performed.

VIII. Results Using the Trained CDN

Figs. 3 and 4 show the test results of the trained CDN after the learning process has been completed. The mass and/or length of the pole in the pole-cart system ($C_A(t)$) were

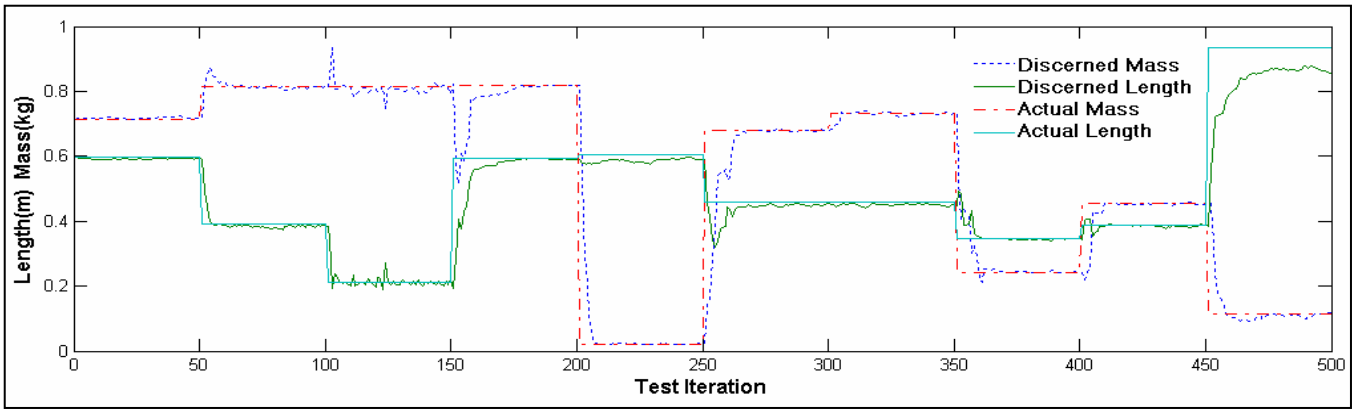

Figure 3: Demonstration of Context Discernment

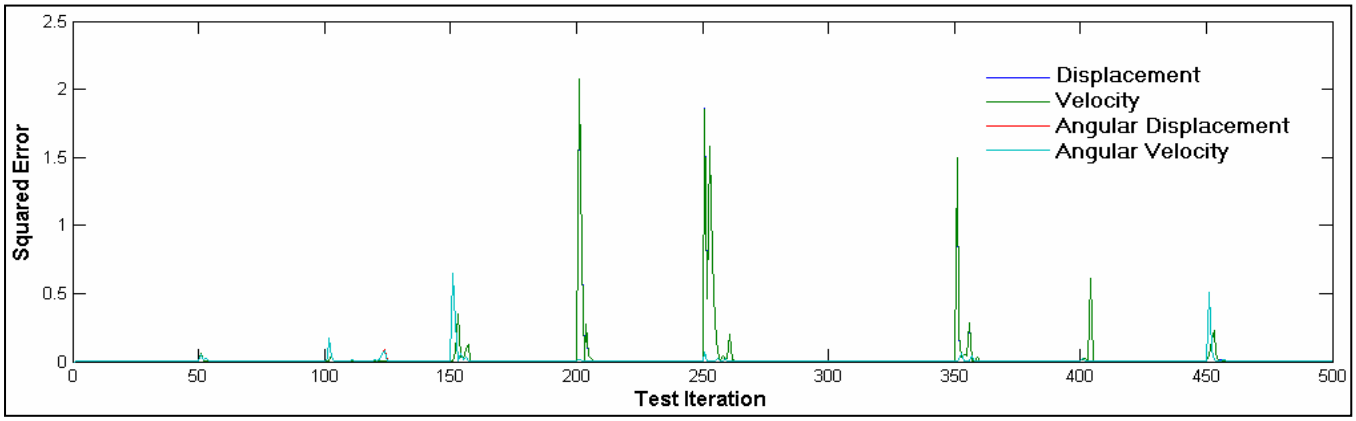

Figure 4: Error between Outputs of System and Model

randomly reset after every 50 iterations. The flat lines in Fig. 3 correspond to these "actual" parameters. Based on deviations between $R_A(t+1)$ and $R_D(t+1)$ at each step of the testing process, the CDN produces a correction to $C_D(t)$. The non-flat lines in Fig. 3 correspond to these discerned parameters. As can be seen, the values of $C_D(t)$ converge, to a high degree, on the values of $C_A(t)$. In some cases, this convergence happens in as few as 5 iterations. Considering the 0.02s sampling time for our process, the CDN is able to discern the mass and length of the pole in the pole-cart system in as little as 0.1s.

Fig. 4 shows the squared error between $R_A(t+1)$ and $R_D(t+1)$ at each step of this test. As can be seen, there is a jump in the error between the actual and expected state trajectories every 50 steps when new values for the mass and length of the pole are instantiated. This error quickly drops to near zero as the context discernment process refits the model to the pole-cart system.

IX. Simplifications and Assumptions

A number of simplifications and assumptions were made for the above experiments. Primarily, the "actual" pole-cart system and the model of this system use the same set of deterministic, noise-free differential equations to describe their state transitions. The behavior of each instantiation of this mathematical model may still be made distinct, however, by setting the parameters differently (including the mass and the length of the pole). There are some profound aspects in which the above simplification affects this experiment.

First, it lets us completely ignore the effects of noise and stochasticity in the context discernment process. If noise were present in our measurements or if the state transitions of our system or model were not deterministic, it would be more difficult to identify the source of deviations between the system's state trajectory and that of the system's model, particularly since we compute these deviations at each step in the present configuration. One alternative to address this issue will be to make the estimates/discernment based on longer segments of the trajectory.

Second, using the same set of equations for both the system and the model in our framework is equivalent to choosing the perfect model structure for modeling the system. This in turn allows the context discernment process to find a set of model parameters such that the model's behavior will exactly match that of the system. Furthermore, if there is a unique set of parameter values that produces these behaviors, we can exactly discern the values of these parameters by the above methodology. This will not, in general, be the case. If the structure of our model is not "perfect," then we could expect some differences between the behavior of the

model and the behavior of the system being modeled. In such scenarios, we can only hope to discern the model parameters that are "best" based on some defined criteria. Experiments along this line are planned for future work.

Finally, the use of mathematical models wherein the parameters correspond to physical attributes of the system being modeled is a special case. This allows us to give names and meaning to the discerned parameters, such as "mass of pole", "length of pole", or "wheel friction". If our model were instead, for example, a neural network, we could not typically label a discerned connection weight in the same way. We only mention this for completeness, as in our view, this does not in any way make the process of context discernment less meaningful when using connectionist models. Further discussion of this, however, is left for another paper.

X. Context Discernment as a Trajectory in Model Parameter Space

A. Adaptive Critic Applied to Design of Controller Agent

In the (now "classical") use of adaptive critics to design a controller, during training of a controller agent, search takes place in its parameter space (for NNs, parameters = weights), where CF is the Bellman J function. The Utility measure ("cost") used to calculate J is a function of the plant state variables (sometimes also of the control variables) and some stipulated "desired" values for each.

After the training/design process converges, the resulting controller agent provides actions (control inputs to plant) that cause the plant to take a **trajectory in *state space*** with minimum cost-to-go at each state. To accomplish this task, the controller in a sense has embedded within it de facto knowledge of the system's dynamics and physical properties.

B. Adaptive Critic applied to design of Context Discerner Agent

In the current application of adaptive critics, during training of the context discerner agent, search again takes place in its parameter space with the Bellman J function as CF. The Utility measure used to calculate J is a function of the difference between the plant's state values and the state values of the model. [If the stipulated "desired" values mentioned above are provided by a model, as in a model-reference control system, then these two statements are parallel ones.]

After the training/design process converges, the resulting context discerner agent provides actions (adjustments to model parameter values) that cause the parametric model to take a **trajectory in *model parameter* space** with minimum cost-to-go at each state (note: due to the dynamics-free nature of the summing node in the CDN, this trajectory may be discontinuous – see section V for more details). To accomplish this task, the context discerner has de facto internalized knowledge of associations between parameter values and plant/model trajectory deviations.

XI. Future Experiments

The high quality performance of the CDN demonstrated in Figs. 3 & 4 is based on a training and test protocol that provided information to the CDN from a broad sampling of the pole-cart's state space (via the random excitations). However, in a preliminary experiment that embedded the CDN in a control environment wherein only a small part of state space was "visited" during operation (after training), the CDN's performance degraded substantially. This observation suggests a difficulty akin to what is known in the control literature as persistence of excitation.

A future exploration is being planned to address this issue. A training protocol is to be developed that incorporates samples of operational sequences of states (trajectories) obtained from a pole-cart that is functioning with a reasonably good controller. This will reduce the volume of state space that we will train over, but we expect better results when performing context discernment over continuous state trajectories. More importantly, the use of state trajectory segments will allow us to use a recurrent neural network to good effect for the CDN. This should buy us two things. One, it is compelling to think that a "memory" of the history of the current state trajectory and $C_D(t)$ will be helpful in determining the next $\Delta C_D(t)$. Second, by switching to a recurrent NN, an alternative training algorithm (Back Propagation of Utility Through Time) may be used that will likely provide a more efficient training process than the DHP method currently used, and hopefully allow for faster training, with results that are at least as good as the ones obtained with DHP.

References

[1] Bellman, R.E., (1957), *Dynamic Programming*, Princeton University Press.

[2] Lendaris, G.G. & J.C. Neidhoefer, (2004) "Guidance in the Use of Adaptive Critics for Control", Ch. 4 in *Handbook of Learning and Approximate Dynamic Programming*, Si, Barto, Powell, & Wunsch, Eds., Wiley, & IEEE Press.

[3] Prokhorov, D., L. Feldkamp, & I. Tyukin, (2002) "Adaptive Behavior with Fixed Weights in Recurrent Neural Networks: An Overview", *Proc. Of International Joint Conference on Neural Networks (IJCNN), WCCI'02*, Honolulu, Hawaii, May, pp. 2018-2022.

[4] Santiago, R.A. & G.G. Lendaris (2004). "Context Discerning Multifunction Networks: Reformulating fixed weight neural networks", *Proceedings of the International Joint Conference on Neural Networks* (IJCNN'04), Budapest, Hungary, IEEE Press.

[5] Santiago, R. (2005), "Context Discernment Through Reinforcement Learning And Recurrent Networks", Technical Report, NW Computational Intelligence Lab, Portland State University, http://www.nwcil.pdx.edu

[6] Sutton, S. & A. Barto, (1998), *Reinforcement Learning: An Introduction*, MIT Press