Higher-Level Application of Adaptive Dynamic Programming/Reinforcement Learning – a Next Phase for Controls and System Identification?

George G. Lendaris,
Systems Science Graduate Program, Portland State University
Portland, OR, USA
lendaris@sysc.pdx.edu

Abstract—In previous work it was shown that Adaptive-Critic-type Approximate Dynamic Programming could be applied in a “higher-level” way to create autonomous agents capable of using experience to discern context and select optimal, context-dependent control policies. Early experiments with this approach were based on full a priori knowledge of the system being monitored. The experiments reported in this paper, using small neural networks representing families of mappings, were designed to explore what happens when knowledge of the system is less precise. Results of these experiments show that agents trained with this approach perform well when subject to even large amounts of noise or when employing (slightly) imperfect models. The results also suggest that aspects of this method of context discernment are consistent with our intuition about human learning. The insights gained from these explorations can be used to guide further efforts for developing this approach into a general methodology for solving arbitrary identification and control problems.

Keywords-context; system identification; autonomous control; Adaptive Dynamic Programming; adaptive critic; reinforcement learning

I. INTRODUCTION

For this keynote talk at the 2011 Symposium on Adaptive Dynamic Programming and Reinforcement Learning, I am assuming a basic familiarity with the Adaptive Dynamic Programming (ADP) ideas and methodology, and with the generic way ADP has been applied to the design of (near) optimal controllers (cf. [1][2]). At its base, ADP is a combination of the Adaptive Critic (AC) type of Reinforcement Learning and Dynamic Programming (DP) ideas. Importantly, whereas DP calculates the control via the optimal Value Function, the AC concept utilizes an approximation of the optimal Value Function to design the controller. The combination became known as Approximate Dynamic Programming, and more recently, Adaptive Dynamic Programming. Considerable advances continue to be made in the theoretical and application aspects of ADP, as exemplified by the material presented at the current Symposium. A good recent review of advances pertaining to control is given in [3].

The approach here adopts a different focus on how to apply ADP, motivated by the desire in the controls field to accomplish human-like control capability, and by the observation that 1) humans have the ability to make use of experience while selecting their control actions in changing situations, and 2) their process speeds up and has enhanced effectiveness as more experience is gained. The latter is in stark contrast to current technological implementations which slow down as more knowledge is stored. Guided by the objective of achieving enhanced speed and effectiveness as more experience is gained, the approach here employs ADP in a manner that shifts its focus “up a level”, away from designing individual (optimal) controllers to that of developing algorithms that efficiently and effectively select designs from a library of existing controller solutions, where this library/repository of existing controller designs is taken as a proxy for the notion of “experience” in anthropomorphic terms.

The process of designing a control policy by definition takes into account i) the type of plant to be controlled, ii) the environment in which it operates, and iii) the specified optimality/performance criteria. For the purposes of the present work, these three components taken together are said to comprise context from the perspective of the controller. ADP has been fruitfully employed in recent decades to design (near) optimal control policies for a variety of contexts.

We comment that methods in the controls field known as Adaptive Control and Learning Control focus on redesigning the controller as changes in context (typically in the plant and/or its environment) occur, whereas Experience-Based Control introduced in [4] and described here entails selecting a previously-designed controller from an experience repository, appropriate to the current situation.

The approach taken here in effect presupposes an autonomous agent that has previously encountered a variety of distinct contexts for a given problem domain and for each has gone through a process of developing an optimal control strategy (called ‘control policy’ in the Dynamic Programming literature). It is assumed that the agent has a way of saving these optimal control policies, each indexed by the specific context for which it is optimal. It is next assumed that in the interest of efficiency, rather than designing a control strategy anew each time a previously-experienced context is encountered, the agent saves each control policy as it is developed into a kind of library/repository and then selects a control policy from this repository each time it discerns that context has changed. For this approach to be useful, the process of select-

This work was supported in part by the NSF Grant no. ECS-0301022.
ing the appropriate previously-designed control policy is itself to be optimized, defined in terms of speed of selection (efficiency) and effectiveness of the selected control policy.

To accomplish the design of such an agent, it would be useful to come up with learning methods to:
1) Accumulate a set of “relevant” control policies for an engineering task (as is done by the engineer for Adaptive Control).
2) Create a representation schema that facilitates accessing the policies.
3) Create an algorithm that efficiently and effectively selects among these policies during online operation of the system, in response to changes in its context.

The Adaptive-Critic-type of ADP is employed in the present work to develop such an optimal selection strategy – to select an optimal controller from the repository in an optimal way, in contrast to (simply) adjusting the existing controller’s design. Note that the two uses of the word “optimal” here refer to two distinctly different optimality criteria: the first relates to the control task, and the second relates to the process of selecting the corresponding optimal controller [4][5].

A key precursor to the act of selecting a control policy is that the agent must be able to discern when the context changes (e.g., of the plant itself, the environment, the optimality/performance criterion, or any combination thereof). Accordingly, the agent is to be endowed with the capability to monitor the controller’s context, and when a change in context is discerned (in any of its components), the agent selects a controller from the experience repository that is optimal for the currently discerned context. Among our desires for such an agent is to provide it with the capability to “interpolate” controller designs – that is, when a context is discerned that is near one for which an optimal controller has been previously designed, then the agent is able to come up with a controller based on the design(s) in the repository that correspond to the nearby context(s).

There are clearly multi-level aspects to such a decision environment for the agent. From the perspective of how the ADP is employed, that is, what it is optimizing, the present research focus is deemed the first level up from the “usual” one. See [4] for further elaboration of this aspect.

A. Conceptual Configuration of Experience-Based Control Process

Fig. 1 provides a conceptual layout of the Experience-Based (EB) control idea as it might be performed when the various aspects of the EB method are worked out. The reader is directed to the upper left corner of Fig. 1 at the place labeled ‘Starting Condition’; this is intended to represent the situation where a controller/plant configuration is functioning as expected in some operating environment. Moving to the right, the Agent monitors the situation to become aware of changes that may occur. When a change is noted, the Agent goes into a context discernment mode to figure out what changes have occurred; in Fig. 1 this action is labeled Perform SysID (hence specializes the figure to a control setting with changing plant parameters). This stage yields an updated plant model, and the Agent procedes to the EB Controller-Selection task. Following this, the Agent runs a simulation with the new controller and plant models, and assesses performance according to the performance measure (Criterion Function—CF). [As a side note, the author has been informed by colleagues that human’s appear to perform ‘rehearsing’ in such situations; this motivates the ‘simulation’ stage.] If all is OK, the new controller design is uploaded to the controller box in the upper left corner of the figure; if not, the context discernment stage is entered again.

![Conceptual layout of Experience-Based Control (EBC) process.](image-url)

Two early versions of the EBC process have previously been demonstrated: 1) steering a car on a dry road and then encountering an ice patch (see [20][21] in [6] of this paper); and 2) flying a hypersonic aircraft and then encountering a sudden shift in the center-of-gravity (e.g.) (see [18][23] in [6] of this paper). In both cases, it was reasoned an experienced human operator, upon noticing changed vehicle behavior (context) would invoke a “higher level” process to acquire information, say by sensing the vehicle’s response to small perturbations to selected control inputs to update the operator’s knowledge of Environment and/or Plant parameter values (a SysID process), to assist in deciding what to do next. The latter could include modifying the CF being employed (e.g., emphasize safety vs. timely arrival), and then selecting appropriate control actions. Guided by this reasoning, we (the human designers) developed a proxy measurement that provided data about the car/road interaction (tire slip angle) in one case (environment data), and a proxy measurement for location of c.g. in the other case (plant dynamics data) – i.e., ‘context variables’. Important here is that during the Adaptive-Critic design process, the context variable was included as an auxiliary input to the controller, in addition to the usual plant state variables (cf. bottom half of Fig. 2). In both applications, the controller learned to use a change in the value of the context variable to select a different controller instantiation – i.e., the context discernment was (externally) provided to the NN controller as a value of the context variable, and the NN controller in essence developed its own local repository of controllers selectable via the context input (cf. Fig. 2). What distinguishes these precursor results from the current research focus is the fact that the context variables (and their respective sensors) were crafted by the control engineer, whereas it is desired to take the human out of that part of the design loop. While crafting the context variables employed in the above examples, we consciously took into account how a human operator of each type of vehicle might acquire the additional context data. We
did this “manually”, as at that time it had not occurred to us to define a “next level up” task for the ADP method to perform, a process we now know entails crafting a new and different CF, one appropriate for the new level.

In the present paper, focus is on the SysID part of the Context Discernment process, with a fixed environment and fixed performance criterion. We note that the interpolation idea relates also to the task of system identification—discerning plant status—where the agent has accumulated a collection of models that it now uses as its experience repository for efficient and effective systems identification.

Space limitations preclude very much review of prior work. To summarize, however, journal papers [4][5] contain the underlying ideas and rationale for the proposed approach to development of autonomous agents capable of performing context discernment and context-dependent control. These references include results of early experiments and cite the fact that those early experiments were based on full a priori knowledge of the structure of the plant/model. Future explorations were suggested wherein the process be subject to less-precise knowledge of the plant/model structure, that the process be subject to noise, and that adequate representation schemes are developed for indexing and accessing items from the library/repository, both for systems identification and for controller selection. This written version of the talk primarily contains descriptions of results from recent experiments. More detailed description of these recent experiments is contained in a new conference paper [6].

II. SYSTEM IDENTIFICATION PRELIMINARIES

A high-level representation of the system identification process is depicted in Fig. 3. The plant that is being controlled is monitored to determine whether its input/output characteristics have changed, and if so, system identification (SysID) is undertaken to determine what it has changed to. In Fig.1, the SysID activity is depicted as a Self-Adjustable Model. From the perspective of the controller (not shown in Fig. 1), the plant status is one of the components of the context in which the controller is operating; it is therefore important to monitor changes in the plant, which entails a system identification (SysID) process, here called (part of) Context Discernment.

There are two key components of the Self-Adjustable Model of Fig. 3, shown in Fig. 4. There is a component “box” with inputs and outputs that performs a mapping (function) from its inputs to the outputs. This box contains a Parameterized Model that performs the mapping and has an additional input \( p \) (called a ‘selector input’) that sets values of the Model’s parameters, with the effect of specifying the particular input-output mapping being implemented by the box. In addition, there is a box whose purpose is to provide the ‘selector input’ \( p \) to the Parameterized Model and is called the Context Discerner (CD). Its role here is to discern the current input/output characteristics of the plant being controlled and to select the mapping within the Parameterized Model box that represents those characteristics. The CD is trained via ADP to perform its discerning task. Details of this are given in [7], and a more general description of ADP operation in [8].

III. OUTLINE OF CURRENT EXPLORATIONS

The objective of the work described in this section has been to expand prior work via exploring effects of noise on the Context Discernment process, and in addition, the effect of employing a (slightly) inaccurate parameterized model to represent the plant being controlled.

To provide for an orderly exploration, the Plant was set up as a feedforward neural network (NN) of a given connection pattern, and the Parameterized Model (the Model) was set up with a NN of identical structure and weight assignments. For each set of experiments a different instantiation of weight assignments was made with the biases employed as adjustable parameters (selector inputs). Conceptually, to each instantiation of weight values there corresponds a family of mappings that can be performed by the NNs, where different members of the family are instantiated by different bias values. Thus, the task of the Context Discerner box of Fig. 4 is to determine the
correct value of bias $p$ so it can select the correct model from the Parameterized Model box.

A number of different experiments were run via different instantiations of weight values, thus providing different families of mappings to explore. Two example families are given in Figs. 5 and 6. The Plant for these experiments had two inputs and one output. This choice was made to facilitate the representation employed in Figs. 5 & 6, namely, the bottom plane representing the two inputs ($x_1$ and $x_2$), and the vertical direction representing the NN’s output ($y$). (Visualization of the context discernment process has been very useful for understanding it at this stage of exploration.) Each surface depicted in the two figures corresponds to a different value of the bias. Both figures are based on results from experiments discussed in the following sections. Some of the qualitative differences are easy to notice in these figures.

![Figure 5. Representation of a family of mappings implemented by one instantiation of weight values and different bias values. The three indicated surfaces correspond to three different bias values. This family of mappings is characterized as “high volume, high tilt.” This characterization and the trajectories on the surfaces are described in Section IV.](image)

![Figure 6. Representation of a family of mappings implemented by an instantiation of weight values different from those instantiated for the mappings in Fig. 5. This family of mappings is characterized as “low volume, low tilt.” This characterization and the trajectories on the surfaces are described in Section IV.](image)

Referring to Figs. 5 & 6, given inputs $x_1$ and $x_2$ and an initial (random) bias vector $p$ to the Model, the Model generates an output $\hat{y}$ (the black square in both figures). At the same time, the Plant is providing the target output ($y^*$) for a given input-vector value (the Plant is instantiating a member of the family of mappings associated with its current weight values, determined by its current bias setting). The target output for the given input-vector value is a point on this surface. When the error measure between the Model and Plant outputs exceeds a specified threshold, the CD updates its guess of the bias setting based on three observations: this error-measure value, the CD’s last guess for the bias settings (context), and the Plant’s inputs. Within a few time steps the error-measure value becomes close enough to zero (according to some threshold value) that the context discernment process stops, and the CD is considered to have identified the Plant. After a few more time steps in the experiments reported in the next sections, the Plant is changed again, i.e., new bias settings are instantiated. The CD, having not yet received any information to signal that the context has changed, does not update the bias settings and so the Model output is (momentarily) incorrect; however, at the next time step the CD receives an error-measure value which triggers the context discernment process once again, and within a few steps the CD successfully identifies the new Plant.

IV. EXPERIMENTS

A. Experimental Design

In crafting the experimental design, a variety of considerations are important and comprise at least the following: the overall system structure (including that of the Model and the Context Discerner), the inputs provided to the context discerner, whether or not the available observations of the system are noisy, the training and testing syllabi, and measures of performance. In this endeavor we are guided by our understanding and intuition about how humans learn and develop experience. While the experimental results that follow are based on a relatively simple formulation of the system identification component of the context discernment problem, formulation of the experiments was guided by the associated desire to have this methodology eventually applicable to arbitrary control problems, of which system identification is the first component.

A first step is to decide what types of plants/systems to focus on. Given that certain neural network paradigms are universal function approximators, these were deemed to provide a good starting point. Although the context discernment methodology is intended to be able to identify any type of input-output system, neural network models are used in the present experiments because they provide known plant structures on which models can be based. A model may be capable of representing a system perfectly, but generally it only need be good enough to provide the information needed to perform the task at hand. In humans, it is believed that models are continuously improved as learning progresses, up to a degree required by the problem domain. For example, initial driving lessons may be given on an isolated driving course to provide the new driver a safe environment in which to develop a starting model, which is then continuously improved as the student driver moves out of the driving course and encounters “real world” driving conditions. The actual model that is developed depends on whether the driving is to be performed in a rural area, a congested metropolitan area, or a race track.
The models employed here had structures either identical or nearly identical to the plants being identified.

Mathematically, a feedforward neural network provides a mapping from input to output. Once the type of element is selected and the NN’s structure is fixed and the weights and biases are specified, the mapping performed by the NN is determined. The mapping can be changed by changing any of these aspects. For example, varying the biases can change the mapping performed by the NN while maintaining the same structure and weights. For each instantiation of the vector of the bias values there corresponds a particular mapping that is performed by the given NN structure. This is illustrated by the multiple surfaces (mappings) in Figs. 5 and 6. The surfaces depicted there represent three distinct mappings from the family of mappings the given NN whose weights are specified is capable of performing; the particular instantiations were determined by three different bias-vector values.

The NN employed for the experiments reported here comprised two hidden nodes, and specification of their biases is given via a 2-dim. bias vector. Each such bias-vector value implements a different plant, and hence a different context from the perspective of the agent whose task is to select an appropriate controller as changes in plant I/O characteristics (changes in context) are detected. For simplicity in construction, the two hidden nodes were assigned identical bias values.

With this particular formulation of the Plant, the SysID task (context discernment) is to determine the values of the biases used in the current plant instantiation that is generating the input/output mapping. In the ideal case the context-discriming agent will select the correct bias settings from the experience repository and do so in an efficient manner. In the present line of research, the agent is designed (trained) via the ADP method to implement a (near-optimal) selection strategy.

One of the key challenges in setting up this context discernment process is representation. How can the different context components be represented to create a natural “indexing” schema that the agent can use to effect the selection of context, and further, one that includes the notion of “nearness” mentioned earlier? Another challenge in developing this methodology is the fact that, in practice, whether or not the agent can learn to optimally select a mapping from the repository (represented here as a family of mappings) is affected not only by the particular maps but also by the training process, in particular, the “training syllabus” employed. When a good selection performance is not achieved, it is not always clear whether this failure was due to the difficulty of the problem or to incorrect or sub-optimal ADP training parameter values.

To explore such issues, it would be useful to develop a method for characterizing a family of mappings. There no doubt exists a large variety of possible representations and measures that might be useful for the present purposes. In our explorations, two simple measures and corresponding representation approaches emerged. The measures are here called “volume” and “tilt.” A vector space is defined via the Plant’s inputs, outputs, and bias settings. Outputs are plotted in this space for all possible values of inputs and bias settings. The portion of the space that is circumscribed by the resulting plot is said to have a “volume” and a “tilt” in that vector space. The volume measure characterizes how much of this space is occupied by the family of mappings; the tilt measure characterizes how the family of mapping is oriented within this space. (Refer to Figs 5 and 6 for examples.) Both these measures are crude, and the intent here is not to suggest that these measures are universal or applicable to all problems, rather to characterize the specific problems explored here in such a way that we can gain insight into the performance of the context discerner and how we might adjust the training syllabus to improve performance. Clearly, for other families of mappings, other characterizations and measures may be more useful.

That context (plant characteristics) can change without having a significant effect on the output is an important consideration for performance. This brings forth the idea that it is not so much an issue of the Plant characteristics changing, per se, but rather an issue of magnitude of divergence between the Plant’s output and the Model’s output (recall that the agent employs the Model to select the controller). The Plant characteristics have to have changed “enough” to appreciably affect the output mapping to be of interest. To this end, three measures of performance were developed for the present experiments: i) operation within threshold, ii) root mean square (RMS) error, and iii) recovery speed. Operation within threshold represents the percentage of time that the current model generates an output within some tolerance—an arbitrary, acceptable threshold. The RMS error is simply the square root of the squared errors averaged over the entire test. The recovery speed is the amount of time (number of time steps) it takes for the context discerner to recognize that the context has changed, select a new Model, and get it to operate within threshold once again.

Based on these measures, different approaches can be used to determine whether or not to select a new context. Having set forth the idea that for human-like control the agent must “detect large enough changes” before selecting a new policy, context was updated here only when the difference between Model and Plant outputs was outside a set threshold.

We again drew upon our intuition and knowledge of human learning to inform design of the “training syllabus.” In general, it is not possible to train for all possible contexts; nevertheless, the goal is to create agents that generalize well, that is, perform well in contexts never before encountered. Humans can often perform near-optimally in new contexts by drawing upon experience from similar contexts.

Two main methods for exploring different mappings/surfaces during training of the Context Discerner were employed: random and organized. In the random case, some subset of contexts (bias values) are selected randomly; each is instantiated as the Plant to be modeled and is employed for some number of training iterations. Each bias value may be selected multiple times during the training process. In the organized case, the contexts (bias values) are selected methodically and purposefully. Within this organized case, two possibilities exist: either a defined sequence of different contexts is explored, or after each is explored for a designated number of training iterations the next context/surface (bias value) to be
explored is based on the previous performance. In terms of human learning, one could imagine an instructor employing a strategy of presenting a pupil with a pre-defined set of exemplars or one of presenting “easy” exemplars first and then increasing and/or decreasing the difficulty over time depending on the pupil’s performance. Other organized approaches are also possible. Both random and organized approaches to training were used in the experiments discussed below. As with any NN training algorithm, the learning rate, momentum, and other learning parameters must also be considered.

The experiments reported here were run on the software platform called ONPAK (Ordered Network Package), a neural network package developed at the Northwest Computational Laboratory (NWCIL) at Portland State University. The ONPAK Matlab code set and additional scripts were used to create the necessary components and implement the context discernment process described in [9]-[11]. Multilayer Perceptrons filled the Plant, Model, CD, and Critic roles. The actual architecture used for all system identification experiments is shown in Fig. 7. The Plant and Model both have two inputs, \(x_1\) and \(x_2\), and a single output node, plus a single selector input \(p^*\) or \(p\) used to tune the biases of the two hidden nodes. Both the Plant and Model have identical, random, fixed weights for all connections except those on the connections from the selector input to the hidden nodes, which are fixed at a value of 1. The Model is created by making an exact copy of the Plant. The CD and Critic each have a single hidden node, and the connection weights are adjusted during the Adaptive-Critic-type ADP training process. The Critic trains the CD and itself based on the Critic’s outputs calculated from the input, discerned context, an output error measure, and the CD output. This architecture was used for all experiments. A bias node (not shown) with a constant input of 1 is also connected to the output node in all four networks. In the Plant and Model the bias weight is random but fixed; in the CD and Critic the bias weight is adjusted until training is completed.

A number of preliminary experiments were run to confirm some general principles about training prior to setting out the final experiments. In particular, early experiments focused on finding reasonable values for the learning rate and number of training epochs. It is generally acknowledged in the NN literature that, for a given problem, there exists a “sweet spot” at which the best results are consistently obtained. For these preliminary experiments, learning rates of 0.01, 0.05, and 0.25 were used, with training epochs of 500, 2,500 and 10,000, along with variable amounts of bias settings, \(x\)-vector (Plant input) presentations, and measurement noise.

Results of these first experiments demonstrated that training with a learning rate of 0.05 for 2,500 epochs consistently produced good results for a wide range of two-input, one-output Plants with varying amounts of noise. While comparable results were sometimes obtained with lower and higher learning rates, more often results were worse. With a learning rate of 0.05 for 10,000 epochs, slightly superior results were sometimes obtained, but not often enough to warrant the much higher computational expense. This is not to imply that a learning rate of 0.05 for 2,500 epochs is the ideal general configuration for the context discernment methodology; just that it worked well for the present experiments.

For all experiments, the learning rate was 0.05 and the number of training epochs was 2,500. For each experiment, either 10 or 50 bias settings \(p^* \in [-1,1]\) were used during training. In experiments with the 10 bias settings, each setting was held constant in the Plant for either 10 or 50 learning iterations (input-output presentations); for experiments with the 50 bias settings, each bias setting was instantiated for only 10 consecutive learning iterations at a time. In other words, a bias setting was selected, either randomly or according to some organized training scheme, and instantiated in the Plant for a specified number of learning iterations, then another bias setting was selected (with replacement) and presented for the same number of iterations, and so on. At each iteration, a random input pair \((x_1, x_2)\) was used to generate an output from the Plant based on its current bias setting, and that same input pair was also presented to the Model. The CD received the same inputs, and in addition, the current discerned context and the absolute error between the Plant and Model outputs. The Plant always received noise-free inputs, but for the experiments with noise both the Model and CD received the inputs with noise added (to represent measurement noise); in some cases the CD also received error-measure values with noise added. The noise distribution had a mean of 0, a stan-
edard deviation of 0.33, and was multiplied by a factor of 0.10, 0.25, or 0.50 before being added to the input or error values. Because the input and output values were scaled to be between -1 and 1, this noise distribution could significantly alter the data received by the CD.

After training, the Context Discerners were tested on a subset of five bias settings used during training to see if they had been adequately trained. The ability of the CD to select the correct model via its context parameter was then measured using two generalization tests: 1) performance relative to a specified sequence of five discrete context changes and 2) performance relative to a slowly varying context (a sine wave). These two tests are here referred to as “fast” and “slow” tests. Performance was evaluated in terms of the percentage of time the Model operated within threshold, the average error, and the speed of recovery. The threshold was set at a squared error of 0.01 for all experiments.

The experiments were designed with the intent that results would be able to provide insights useful for development of the context discernment methodology. So while the results can be used to evaluate the performance of the methodology for the present system identification tasks, they also provide guidance for how to better apply it and what general approaches might be used to improve its performance. In the discussion that follows, we focus on these broader methodological issues.

B. Experimental Results

1) General Observations

The initial runs comprised 1,350 experiments with 50 different Plant “sets” (50 unique sets of weight assignments) and served to explore questions related to training schemes and noisy inputs. Context discerners were successfully trained to perform system identification for the 50 different Plant sets under a variety of conditions. Additional experiments, performed after reviewing results of the initial experiments, showed good performance could also be achieved when noise was introduced into the error-measure calculations and when (in the present exploration, only slightly) imperfect models were used for context discernment.

Training performance was a very good predictor of generalization. Performance on both generalization tests (the fast and slow tests) was similar to training performance for all measures (operation within threshold, RMS error, and recovery speed); if training performance was poor (as it was in a few cases), generalization was also poor, and when training was successful, generalization was as well. Results on the fast test were usually slightly better than on the slow test. This was likely due to the nature of the tests themselves: the fast tests included only five context changes, while the slow test used a continuously varying context (bias values); due to what may be called quantization (via the threshold employed for the error measure) the context discernment lags behind. Specific details regarding performance are discussed below.

In terms of average performance (both operation within threshold and RMS error) and the number of performances exceeding 90% within threshold, use of the 10-50 scheme for training achieved the best-performing context discerners. Based on these results it appears that having more consecutive presentations for each bias setting during training (equivalent to more sample points on each selected surface, if visualized as in Figs. 5 and 6) typically improved performance. In both the 10-10 and the 10-50 schemes, each specific mapping is sampled, on average, for 250 of the 2,500 total training iterations. The only difference between these two training schemes is how many input/output exemplars are presented before a new bias setting is selected. Using an insufficient number of exemplars can be characterized as “under-training”—the context discerner does not learn enough about the specific mapping before the trainer moves it onto another one. Results for the 50-10 scheme were very similar to the results for the 10-10 scheme. This suggests that given limited training time (in this case just 2,500 total training iterations), increasing the number of consecutive training presentations may be more fruitful than increasing the number of different bias settings.

On the other hand, examples of the worst performances also occurred with the 10-50 training scheme. This suggests that in some cases the Context Discerner was “over-trained” using an inadequate set of training examples. As with many other machine learning methods, just what constitutes a good set of exemplars and how long and how often they should be presented is unclear. A few supplemental experiments with organized presentation of non-random, purposely selected bias settings presented in either particular or random orders generated much worse results than the main experiments with randomly generated bias settings presented in random order. Choosing a “good” set of exemplars is a non-trivial task—even for the present seemingly straightforward, low-dimensional system-identification task. Future research must necessarily explore this aspect of the methodology.

Performance is also affected by characteristics of the plants themselves, here characterized as volume and tilt. Plots of test performance with respect to volume and tilt are shown in Fig. 8 for the (a) 10-10 and (b) 10-50 training schemes. (As noted above, results for the 50-10 scheme were very similar to those for the 10-10 scheme; in the interest of space, a plot for these is omitted.) Circle size represents the percentage of operation outside threshold based on a squared error of 0.01 between the Plant and Model outputs; smaller circles represent better performance, larger circles worse performance.

In general, performance decreased as volume and tilt increased, regardless of whether measurement noise was present or not. For “low-volume” problems changes in context within that volume yield plants whose output values are very similar and thus lower corresponding error-measure values and operation outside threshold; for “high-volume” problems changes in context yield plants whose output values vary more and thus increase error-measure values and operation outside threshold. For plants with very low volumes and very low tilts, operation within threshold was (or was very close to) 100%—indicated by the very small circles in the lower left corner of both plots in Fig. 8—even with noisy inputs.

Comparison of the results using the 10-10 scheme with those of the 10-50 scheme provides additional insights into the relationship between training and the problem characteristics. Evident from comparison of the two-schemes is that perform-
ance on low-volume problems can be improved by increasing the number of consecutive $x$-vector presentations for each instantiated mapping. Because the individual mappings are so closely spaced, more refined training—e.g., more example points from each surface—is needed, otherwise it is difficult (if not impossible) for the CD to distinguish between very close mappings, i.e., under-training may be a greater concern for low-volume problems. The specifics of a particular problem will likely dictate how fine the resolution in the context discernment process is needed to generate adequate performance, and by which metric that performance will be evaluated.

Performance and noise is represented by the sets of three concentric circles, each set representing tests on the same family of mappings with different amounts of noise (either 0, 0.25, or 0.50) added to the Plant’s input values seen by the Model and CD. In most cases, the operation outside threshold increased as noise increased. This relationship also reinforces the likelihood that the four very poor performances observed when using the 10-50 scheme (Fig. 8b) were due, at least in part, to over-training with an inadequate set of bias settings since in all four cases better performance was observed with higher amounts of noise.

Interestingly, test performance did not seem to be significantly affected by whether or not the $x$-vector (Plant input) values provided to the Model and CD were noisy during training. Inspection of the weights for the trained CDs revealed that the weights on the connections between the input nodes and hidden node were close to zero and several orders of magnitude smaller than the weights for the current-context and error-measure inputs to the CD. In other words, in most of these initial experiments (with or without noisy inputs), the CDs had learned to adjust the bias settings primarily via the information from the current context and error measure.

To investigate these results further, additional runs with noisy error-measure values were made, and we observed that adding noise to the error measure had a greater negative effect on performance than did adding noise to the Plant-input observations. With 0.25 noise added to the input observations, operation within threshold was still above 90% two-thirds to three-quarters of the time, yet not one CD was able to achieve 90% operation within threshold when the same amount of noise was added to the error-measure values. With just 0.10 noise added to the error-measure values, 90% operation within threshold was achieved more than half the time.

### TABLE I. SUMMARY OF PERFORMANCE (OPERATION WITHIN THRESHOLD) FOR VARIOUS EXPERIMENTS USING THE 10-50 TRAINING SCHEME.

<table>
<thead>
<tr>
<th>experiments</th>
<th>fast</th>
<th>test</th>
<th>slow</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>89.9</td>
<td>88.5</td>
<td>86.9</td>
</tr>
<tr>
<td>Train noise-free, test noise-free.</td>
<td>96.7</td>
<td>92.6</td>
<td>89.9</td>
</tr>
<tr>
<td>Train w/ noisy input, test noise-free.</td>
<td>93.8</td>
<td>91.2</td>
<td>88.3</td>
</tr>
<tr>
<td>Train noise-free, test w/ noisy input.</td>
<td>87.1</td>
<td>86.9</td>
<td>86.2</td>
</tr>
<tr>
<td>Train w/ noisy input, test w/ noisy input.</td>
<td>87.6</td>
<td>86.1</td>
<td>85.0</td>
</tr>
<tr>
<td>Train w/ low noisy output, test w/ low noisy output.</td>
<td>90.1</td>
<td>88.3</td>
<td>85.9</td>
</tr>
<tr>
<td>Train w/ high noisy output, test w/ high noisy output.</td>
<td>64.9</td>
<td>63.1</td>
<td>63.9</td>
</tr>
<tr>
<td>Imperfect model: train noise-free, test noise-free.</td>
<td>87.1</td>
<td>83.6</td>
<td>80.8</td>
</tr>
</tbody>
</table>

After the fact, these results were not surprising since results of the prior experiments showed that the error measure provided more useful information to the CD than the input values did. However, when CDs were trained and tested with 0.25 noisy error measures, operation within threshold improved measurably. This improvement was primarily observed for the higher level of error noise, and adding noise to the inputs re-
duced this improvement. Examination of the trained CDs showed that, at least in some cases, the weights connecting the Plant-input nodes to the hidden layer were often an order of magnitude higher than those in CDs that had been trained without noisy error measures. Based on these results, we offer the conjecture that when the CD was trained with high levels of error noise it learned to extract additional information from the input data, information that was not necessary when the error measure was only slightly noisy or noise-free.

3) Imperfect Model

Performance was also degraded by use of an imperfect model, in this case a Model that used a linear activation function in its output node instead of the hyperbolic tangent activation function present in the Plant’s output node. For some problems, performance degraded little, but for others it degraded significantly. On average, operation within threshold was reduced by about 10%, although for some cases the reduction was much higher. However, the average increase in RMS error was only 33%, and only in a few experiments did the RMS error more than double. An example of results using an imperfect model is shown in Fig. 9. The Plant used for this experiment was identical to that used to produce Fig. 5. Comparing the two figures confirms the performance of the CD was degraded by using an imperfect model, but the CD still performs reasonably well given this limitation and the difficulty of the problem. This suggests that the CD did “as best it could.”

Regarding the last two items, when measurement noise is present or an imperfect model is employed it may be necessary to tolerate worse performance unless/until noise can be reduced or a better model can be developed.

5) Two additional sets of experiments

Two additional sets of experiments were performed—both with 50 new target Plants.

a) First set

In the first additional set of experiments, rather than using the adaptive-critic-type method described above, a Model with the same structure as the Plant (two hidden units) but with arbitrary, adjustable weights was used. During training the Model was provided with the bias setting currently instantiated in the Plant, and (plain) Backpropagation was used to update the Model weights over 2,500 iterations with ten random selected bias settings presented via the 10-50 training scheme. This method produced performance significantly superior to that achieved with the adaptive-critic-type training method for all cases. Recovery speed was zero because the Model, rather than representing a specific instantiation of the Plant, was now a general model that had learned to minimize the error measure for a variety of contexts. The trained Model could be considered to be the Self-Adjustable Model shown in Fig. 3; the mechanism for self-adjustment is embedded within the Model itself.

If superior results can be achieved with a simpler, less computationally expensive but still self-adjustable model, what is the benefit of using a Context Discerner? To train the Model via Backpropagation, it was necessary to provide the Model with the bias settings during training; for the ADP method, the Context Discerner learned to adjust the bias settings by observing the inputs and outputs of the Plant and Model and its own output without ever explicitly receiving information about what the actual Plant bias settings were. If a context parameter (or parameter vector) can be measured, standard Backpropagation training may be the most efficient method for generating context-dependent mappings. However, in many cases it may not be possible to have explicit knowledge of what constitutes context and hence what parameter(s) should be provided to the Model—a more general method is needed.

b) Second set

For the second additional set of experiments, a two-stage training method was used. First, a Model with arbitrary, adjustable weights was trained to represent the Plant using standard Backpropagation over 2,500 iterations, via two distinct instantiations of the Plant (two different bias settings, \( p^* \in \{-0.5, 0.5\} \), which were also provided to the Model). Following this first training stage, the weights in the Model were fixed. Inspection of the final weights revealed no obvious relationship between the Model and Plant weights other than the weights for the output node biases, which were nearly identical in all cases. The Model had simply learned to represent the mapping using another set of weights. Following this, a second stage was performed using ADP for an additional 2,500 iterations to train up the CD to provide selector inputs to the Model, with a set of ten randomly selected bias settings instantiated in the Plant via the 10-50 training scheme. As with the original ex-

![Figure 9](image-url)
periments, the bias settings were not provided to the Model; the Context Discerner learned to adjust the bias settings of the Model based on the input-output characteristics of the Plant and Model and its own feedback. Performance using this two-stage process was comparable to that achieved when using a perfect Model for both noise-free and noisy inputs and error-measures. Interestingly, average performance was better for the low noisy input case (0.25 noise) than for the noise-free case. The average for the noise-free case included two very poor performances: one with 0% operation within threshold and another with 30 to 40% operation within threshold. If these two cases were to be ignored, the average operation within threshold was 94.8%, just slightly better than the 0.25 noise case. The lowest operation within threshold for the 0.25 noise case was 70%. These results suggest that adding low levels of noise may be beneficial in that it may help avoid very poor performance while not significantly limiting good performance.

V. Conclusion

Adaptive-Critic-type Approximate Dynamic Programming is being applied in a “higher-level” way to create autonomous agents capable of using experience to discern context and select optimal, context-dependent control policies. The experimental results reported here show that this approach can be robust and adaptive when performing system identification tasks on small neural networks capable of representing different families of mappings. Agents trained using this approach can achieve high levels of performance when subject to even large amounts of noise and perform reasonably well when employing imperfect models. These results suggest that aspects of the proposed method of context discernment are consistent with our intuition about human learning, particularly with respect to how learning is affected by the models and measurements used to perform a task and the difficulty of the task itself. Future experiments must necessarily introduce more complex problems and less perfect knowledge representations if this approach is to be developed into a general methodology for creating agents capable of performing identification and control tasks for arbitrary control systems.

References*


* Please see [4] and [5] for extended lists of References for this material.

Keynote Talk for 2011 Symposium on Adaptive Dynamic Programming & Reinforcement Learning, April, Paris – 2/20/11