

Discerning Context Through Reinforcement Learning

Abstract

We present a *method* for using reinforcement learning (RL) to construct an artificial agent capable of applying learned knowledge in a contextually appropriate way. The method addresses a need, as observed from the perspective of the authors, to expand upon existing *machine learning algorithms* (MLAs) such that the knowledge gained through solving individual learning problems can be reused and generalized to provide solutions to a much larger set of novel problems. The method operates on any existing MLA that uses parameter settings to encode knowledge (e.g. neural networks). In the proposed method, the stream of inputs and feedback used by the MLA is treated as a context to the optimal parameters as determined by the MLA. Thus, each set of parameters has a context in which they are most appropriate. The RL aspect of the method builds an algorithm which extracts features from any context and translates these features into optimal “guesses” for the parameter settings. As a result, features in novel contexts can be used as a priori knowledge for determining appropriate parameters. Since our method applies reinforcement learning to spaces of context we propose the term Contextual Reinforcement Learning (CRL) in reference to the work presented here.

Background

We present here a *method* for using reinforcement learning to construct an artificial agent capable of applying learned knowledge in a contextually appropriate way. The method addresses a need, as observed from the perspective of the authors, to expand upon existing *machine learning algorithms* (MLAs) such that the knowledge gained through solving individual learning problems can be reused and generalized to provide solutions to a much larger set of novel problems. The method operates on any existing MLA that uses parameter settings as the primary device for encoding knowledge; for example, consider backpropagation applied to the weights of a multilayer perceptron. In the proposed method, the stream of inputs and feedback used by the MLA is treated as a *context* to the optimal parameters as determined by the MLA. In essence, the stream of inputs and feedback are characteristic of the learning problem being presented to the agent whose solution is some particular set of parameters. Thus each set of parameters has a context in

which they are most appropriate. The reinforcement learning aspect of the method seeks to build an algorithm which works across contexts and extracts those context features which are most informative about the optimal parameter settings. Thus for novel contexts, the proposed method works to extract context features and translate them into an “educated guess” of the appropriate parameter settings. We find, as will be described, that these educated guesses are often near optimal for novel contexts. As a result, the optimal parameter settings are determined much faster than they would have been if only using the MLA. Since our method applies reinforcement learning primarily to spaces of context we propose, and hereafter use, the term Contextual Reinforcement Learning (CRL) in reference to the work presented here.

Context, Learning Problems and MLAs

Many, if not most, machine learning algorithms have three common components: a learning problem, a parameterized solution structure and a method for determining those parameters. As a simple example, already mentioned, consider the multilayer perceptron (MLP) used to solve a supervised learning problem. Supervised learning implies that the problem definition is one of approximating some mapping. The parameterized solution is the MLP structure and the parameters are the adjustable weights and biases. The method for determining those parameters is to calculate the partial derivative of the squared error of the MLP with respect to each weight and use gradient descent to translate this derivative into an update of each weight. Assuming all works well, for a given mapping the process of gradient descent will result in a set of parameters that will bring the squared error to near zero and thus the mapping will be approximated and the learning problem solved.

Now consider the case where two distinct learning problems are presented to the MLA at differing times. Continuing with the example of the MLP consider having an MLP approximate two different mappings. For clarity, define two mapping, $M1$ and $M2$, that operate over the same domain but have differing outputs. Consider the two continuous mappings presented in figure 1. An MLP with one input, one output and two hidden layers with four

nodes each could be trained to approximate either of these mappings. Figure 2 shows the squared error curve of the MLP being trained to approximate the first mapping ($M1$), then the second mapping ($M2$) and then finally again the first mapping ($M1$). Notice that the squared error curve for the second presentation of the first mapping looks almost the same as during the first presentation. So even though the MLP has been presented with the first mapping before, it holds on to no memory of having solved that learning problem and, as a result, it has to be solved anew.

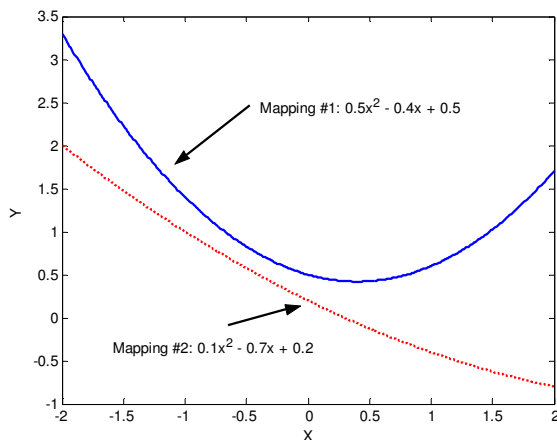


Figure 1 Simple quadratic functions are used to construct two different mapping which define two unique learning problems for an MLP.

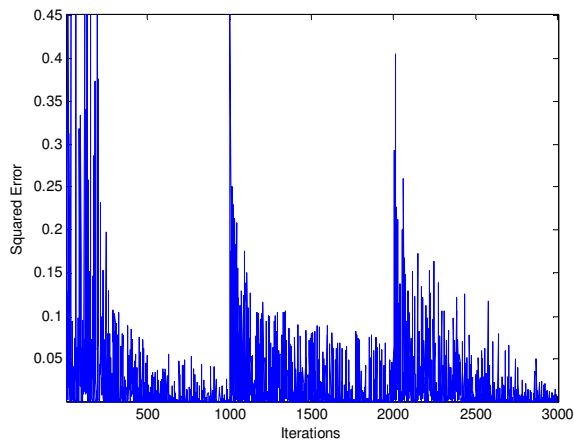


Figure 2 The squared error curve from training the MLP on the first learning problem ($M1$) for 1000 iterations, then the second learning problem ($M2$) for 1000 iterations and then again the first learning problem ($M1$). Almost no memory from the first presentation of $M1$ is retained.

The first motivation for the development of CRL was the desire for knowledge of previously solved learning problems to be made available to a learning agent so as to

avoid treating repeated presentation of the same learning problem *de novo*. In the example provided it would be desirable that knowledge from the first presentation of $M1$ could be reused during the second presentation. This immediately raises the question of knowledge representation, though. Specifically, we asked: How would knowledge from a solved learning problem be stored and, more importantly, how would it be recalled in an efficient manner? MLAs, as stated before, store their knowledge within parameters but since they only have the capacity to use one set of parameters at a time something else is needed. In particular, we hypothesized that the stream of inputs and feedback could provide a source of features which could then be used in an associative way to store and retrieve parameter sets for the MLA. In terms of the MLP example, we would look for features in the stream of random numbers between -2 and 2 that serve as input and the stream of squared errors that serve as feedback. By way of naming convention, we refer to the streams of inputs and feedback that lead to the production of a particular set of parameters by the MLA as a *context* and, logically, the features of the streams as *context features*.

The second motivation for CRL comes from asking the following question: If contexts have features then does there exist a generalizable relationship between context features and appropriate parameters? This second motivation is more complex than the first and begins to form, in our opinion, some basic questions about building perceptual mechanisms and memory for an artificial agent. More specifically, it is to say that when an agent is solving a single learning problem, not only is it solving the problem but it should also be looking for characteristics of the problem that make it unique as well as similar to other learning problems. Therefore, the presentation of a learning problem is immediately subject to the extraction of features and the association of prior knowledge which then play a role in how the agent solves the problem. But, this level of description seems awfully similar to the functional role of perception and memory. These thoughts form the motivation for the method described here and form the basis for the discussion at the end of this paper.

Discerning Context

Several models for extracting features from context were considered in light of the question: How do we determine which features in the context are most informative if we do not know what these features are ahead of time? It seemed that relying on *a priori* feature definitions would be a form of “cheating” in that the effective use of context features would then be more a function of engineering rather than learning by the agent. For clarity, we make distinct systems that take into account context as being context-sensitive; but, in contrast, those systems that are context-sensitive and also determine the appropriate context features without *apriori* information as being *context discerning*. Unfortunately, to date the closest set of

research which seems to address context discernment is the work being done on mixture of expert systems; but, the majority of this research relies on significant use of *a priori* defined context features.

Interestingly, work within the field of recurrent neural networks (RNN), sometimes referred to as dynamical recurrent systems, provided the best insights. In experiments performed by Prokhorov [2], a RNN was trained to approximate a large set of two-dimensional quadratic mappings generated through the parameterized equation $ax_1^2 + bx_2^2 + cx_1x_2 + dx_1 + ex_2 + f$ where a through f can take on any value between -1 and 1. The task is very similar to the learning task shown in figure 2. The most outstanding feature is that the RNN was capable retaining memory of mappings it had learned in the past and was able to recall that information. Moreover, the RNN was trained on 100 instances of quadratic mappings and was able to generalize its information such that it accurately approximated mappings it had never seen without need for additional training of the RNN weights. Further analysis of these RNNs by Santiago [3] revealed that the RNN had learned to discern context and associate optimal parameter settings based on the discerned context. The work presented here was inspired directly by the work of Prokhorov and analysis from Santiago.

The critical insight from this RNN work was that context discernment happens incrementally and is reflexive in nature. Specifically, it was not the case that the RNN simply took several observations from the input and feedback streams and then generated the appropriate parameter settings. Instead, it appears that every observation from the context (every new piece of input and feedback data) was used to update a “guess” of the appropriate parameter settings. Those parameter settings were used by the rest of the RNN to generate the output for the next step and then to receive a new piece of feedback. Interestingly, the next parameter guess was not only based on the new feedback but also on the old parameter guess. In essence, it seemed that each parameter guess served to setup a “test”. That is, the feedback generated by a new input and a parameter guess provided the critical information for the next guess. Figure 3 provides a diagrammatic description of this process. The authors recognize that it can be difficult to clearly understand this incremental and reflexive context discernment process. Indeed, Santiago reports taking months to finally untangle the analysis of the RNN.

Building from the insight that context discernment happens both incrementally and reflexively we now frame the problem of context discernment as one of optimal decision making. That is, it seems that context discernment seems to be answer to the following question: Given my current guess for the appropriate parameter settings, a new inputs and the feedback generated by applying the current guess and new input, what is the next best guess of the

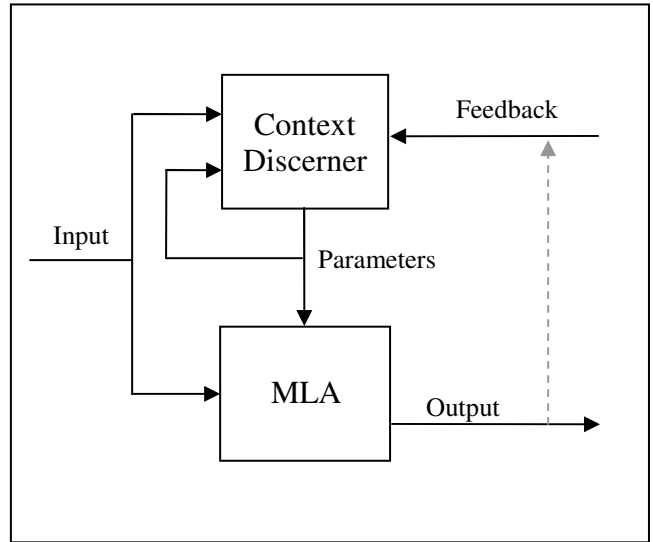


Figure 3 The basic process flow of context discernment is shown. The context discerner takes context information (the inputs and feedback) along with its current parameter guess and generates a new guess of the appropriate parameters.

appropriate parameters? At first glance this seems different from extracting features from the context. To see how it is not, think of the current parameter guess as a manifestation of the perceived context. Thus if the new input and feedback from the actual context conform with the perceived context then we guess the same parameter setting. But, if the new input and feedback do not conform to the perceived context then the way in which it does not conform provides information for generating a new guess.

To see how this would work let us return to the MLP example and imagine the MLP as the MLA in figure 3. For clarity let us define a few things mathematically. Let $in(t)$, $p(t)$, $out(t)$ and $e(t)$ denote the input, parameter guess, output and feedback at time t . the MLP as a function

$$N(in(t), p(t)) = out(t) \quad (1)$$

of an input and parameters and producing an output. We define the feedback as a function

$$e(t) = out(t) - desired(t) \quad (2)$$

where $desired(t)$ is determined by either the mapping $M1$ or $M2$. Normally, we would use a backpropagation algorithm to transform the feedback into a new parameter guess but in the framework of context discernment we define a context discernment function

$$CD(in(t), p(t), e(t)) = p(t+1) \quad (3)$$

The context discerner (CD) takes the place of the standard backpropagation algorithm and generates each new parameter guess. When the feedback $e(t)$ is zero or near zero we would expect that the CD would guess the same parameters or nearly the same parameters. But, if the feedback is far from zero then the CD would use the current input and the feedback to generate a new guess.

So the challenge is to construct the CD function. It is best to think about this function as a policy. It is a decision policy which takes the current parameter guess and context observations and generates a new guess. Since each guess is being made on instantaneous information we want to decide how to use that instantaneous information optimally. Practically speaking, no single context observation is completely informative about the appropriate parameters. Thus, we will need to take a set of decisions (parameter guesses) that result in determining the appropriate parameters after a small number of observations. Constructing an optimal set of decisions is exactly the type of challenge that reinforcement learning solves well. Thus we use reinforcement learning to construct CD.

We will now shift our focus to our choice of reinforcement learning algorithm. While space does not permit all the details the critical aspects of the reinforcement algorithm used and how it was applied to constructing a CD will be provided. A detailed technical report is currently in preparation and MATLAB source code is available from the primary author upon request.

Reinforcement Learning

The term reinforcement learning within the machine intelligence field has come to represent a broad set of approaches to solving the temporal credit assignment problem. A recent publication [1] has brought together many of the research fronts in reinforcement learning. In general, reinforcement learning approaches fall into two broad family, model dependent and model independent forms. Q-learning is perhaps the most commonly known model independent form of reinforcement learning. It has been our experience that model independent forms of reinforcement learning are not as powerful as model dependent forms when it comes to learning problems where the desire is for the action network to control some non-linear plant (e.g. cart-pole system, bioreactor systems, aircraft, etc.). For this reason we chose a model dependent form of adaptive critic known as Dual Heuristics Programming (DHP) proposed by Werbos [4]. Space limits prevent a detailed discussion of the DHP algorithm. A detailed discussion along with helpful implementation tips can be found in [1]. The important details needed to apply DHP for CRL will be provided, though. Please note, in the discussion to follow some basic knowledge of reinforcement learning and dynamic programming is

assumed of the reader. For simplicity, the discussion focuses only on the case for deterministic systems.

Much of reinforcement learning can be seen through the framework of approximate dynamic programming. Dynamic programming is built upon the idea of defining what it means for a solution to be optimal. Starting with some basic definitions the concept of optimal can be clearly defined. First let S be a state space, s be a point in the state space and $S \supseteq \mathfrak{R}^m$. Let A be an action space, a be a point in the action space and $A \supseteq \mathfrak{R}^n$. Define $U(s, a) \rightarrow \mathfrak{R}$ as a utility evaluation of any state-action tuple; when U is positive it indicates an instantaneous reward signal and when negative an instantaneous penalty signal. Define, in discrete time, $D(s(t), a(t)) = s(t+1)$, the dynamics governing movement through the state space. Finally, let $\pi : S \rightarrow A$ be a mapping from the state space to the action space and let Π be the set of all policies that can be defined between the state space S and the action space A . With these basic definitions in hand we can define the fundamental concept of dynamic program, the value function, also known as “cost-to-go” in the case where the utility only provides a measure of penalty or “reward-to-come” when the utility function only provides a measure of reward. For a given policy $\pi \in \Pi$ the value function, denoted V , is defined as

$$V_{\pi}(s_0) = \sum_{t=0}^{\infty} \gamma^t U(s(t), a(t)) \quad \forall s(0) = s_0 \in S \quad (4)$$

where s_0 is any starting state within the state space. The basic notion here is that the value function provides a discounted sum of utility-to-come given the current policy. So the value function can be thought of as a way of evaluating the performance of a policy. Notice that the value function is completely dependent upon the policy. Optimality is defined in terms of an optimal policy, denoted π^* , which satisfies

$$V_{\pi^*}(s_0) = \max_{\pi \in \Pi} \sum_{k=0}^{\infty} \gamma^k U(s(t+k), a(t+k)) \quad (5)$$

In short, the condition in equation 5 states that the best policy is the one that maximizes the sum of utility-to-come from all points in the state space. Using the Bellman recursion we can rewrite (5) in the following form

$$V_{\pi^*}(s_0) = U(s_0, a_0) + V_{\pi^*}(s_1) \quad (6)$$

where $a_0 = \pi^*(s_0)$ and $s_1 = D(s_0, a_0)$. In order to see the connection with Q-learning the following two insights are helpful: a) that the optimal policy will always choose the action that maximizes the right most term of (6) and b) the action is a function of the policy. These two insights along with using the letter Q instead of V allow us to

rewrite (6) without reference to a policy and recover the statement of optimality from Q learning:

$$Q^*(s_0, a) = U(s_0, a) + \max_{a' \in A} Q^*(s_1, a') \quad (7)$$

The Q-learning optimality equation is an example of the so-called value iteration methods to approximate dynamic programming. The value iteration methods focus on the development of the value function of the optimal policy. The actual optimal policy is then implied from the value function. For Q-learning, the value function of the optimal policy is approximated by first “guessing” the value function and through exploration of the combined state-action space, the guess is updated through the method of temporal differences. When completed, in theory, the policy is determined at any point in the state-action space by choosing the action which maximizes the last term of the right hand side of equation(). Value iteration methods, especially Q-learning, can often require a significant exploration of the state-action space and in cases where the

Alternatively, one can take an approach where the policy is incrementally improved based on information from the value function as in equation (5) or (6). These methods are known as policy-action iterative methods or hybrid methods; we will use with the latter term. The majority of so called actor-critic methods fall into the class of hybrid methods. Hybrid methods work in the following manner:

1. Start with a “guess” of the optimal policy
2. Approximate the value function of the current policy
3. Use information from the value function to update the policy
4. Lather, rinse, repeat (go back to step #2)

Such methods, it has been our experience, work well under two conditions, the method used to approximate the value function is differentiable and there is available a differentiable model of the dynamics (D). In short, the derivative of the value function informs us how the value function would change if we were in a slightly different state. The derivative of the dynamics tells us how to change our action in order to get into that slightly different state. Thus if we can change our action in a way that increases the value function then we use that information to change the policy.

DHP is a form of actor critic algorithm. Its key difference from all other actor critic algorithms is the way in which it estimates the value function (step 2 above). Werbos proposes the insight that almost all actor critic methods require that the value function be differentiated, as discussed. Thus to improve accuracy it is best to directly estimate the derivative of the value function as opposed differentiating an estimate of the value function. Again, space does not permit a detailed discussion which can be

found in [1]. It has been our experience that this form of actor critic algorithm provides superior and more consistent results in comparison to other algorithms.

For the purposes of CRL, the CD is the policy we are looking to create. The state is a combination of the context and the current guess (i.e. $s(t)=[p(t), in(t), e(t)]$). The action space is the space of parameters, $a(t)=p(t)$. The utility function is the feedback stream $U(s(t), a(t))=e(t)$. The dynamics in CRL are governed by the MLA; in the case of the MLP example it is the MLP structure. Given a current state and an action the dynamics tell us what state we will be in next. Since the action output from the CD is a parameter guess its only real impact on the next state is the feedback. Again, a detailed technical report is forthcoming with the details of the exact application of DHP to creating a CD.

Experiment

Using DHP we constructed a CD which approximated a large family of mappings. We defined an MLP structure as the MLA. We first tried to have the CD set all the parameters of the MLP. Instead of using a family of quadratic mappings to define our learning problems we took a somewhat ambitious approach. We used a second MLP which had the same structure as the MLA. For clarity we will refer to the MLP used for the MLA as N and the MLP used to define our learning problems as N^T . Thus each learning problem was defined by a set of parameters on N^T and the feedback signal was the difference between N and N^T . It was hypothesized that the CD would be able to determine the parameters used on N^T . Unfortunately, this was a little too ambitious. We found instead that we had to limit the number of parameters we were determining through the CD.

So we again defined two MLPs. The first one for the MLA

$$N(in(t), sp, p(t)) = out(t)$$

where sp is a set of static parameters which were set at the beginning of the experiment and never changed. $p(t)$ are the parameters set by the CD. To define our learning problems we defined the second MLP

$$N^T(in(t), sp, p^*(t)) = t \arg et(t)$$

Again both MLPs had the same structure. Both MLP shared the exact same static parameters sp . $p^*(t)$ specified each learning problem. They can be thought of in the same way as the coefficients of a quadratic function. That is, each unique set specifies a unique desired mapping. The feedback was defined

$$e(t) = out(t) - t \arg et(t)$$

This experiment yielded results but we found that the ratio of static parameters to parameters set by the CD had to be rather high to work, about 8 static parameters to every one parameter set by the CD. For examples, the results discussed next used an MLP structure with four inputs, four units on the hidden layer and four outputs. On this structure the CD set the values of the biases for the four hidden units.

To train the CD, 50 settings of p^* were randomly chosen. Every one hundred iterations the value of $p^*(t)$ was randomly changed to one of these 50 settings and then held constant. So the challenge for the CD was to set $p(t)$ equal to $p^*(t)$. 20000 iterations of training were needed to see good results. During testing 10 new setting of p^* were chosen. Each of these were presented for 100 iterations a piece.

Results

After training the results from the CD were rather impressive, in our opinion. Figure four shows the results from testing of the CD. As described, the learning task was changed every hundred iterations. Upon change, the CD was able to use the context to rapidly approximate the appropriate parameters. In figure 4 the bottom graph shows the squared error. As can be seen every hundred iterations there is a rapid jump in error but then a quick recovery. Closer inspection of results reveals that the CD only needs 10 or less observation of the context in order to determine the appropriate parameters. This is impressive in comparison to the squared error graph in figure 2. The top graph of figure 4 shows the parameter guesses. As can be seen the CD produces a series of guesses that rapidly converged to a fixed point which it "jitters" around. Upon change of learning problem, the shift in parameter guess is almost immediate. These type of results were typical of this method and experiments were performed where the CDN was responsible for setting up to 40 parameters on larger MLPs. In addition, different MLA structures such as RBF networks and fuzzy nets have been used with similar results attained.

Conclusion

CRL seems to provide a very basic model for approaching perceptual and memory systems for artificial agents. The idea of discerning context seems fundamentally important for constructing agents that not only learn but are able through learning to store and reuse knowledge with ease. The results presented here are promising and establish the ability to use reinforcement learning to construct optimal context discerning policies. It is our hope that we can expand the research here to practical problems. In particular, within the field of intelligent control, CRL poses a principled method for creating controllers that respond to changes in plant and environmental conditions. For

example, consider an airplane control system capable of compensating rapidly for damage to a wing. In the robotics field there seems a great need for control systems that allow robots to do the same task in varying environments. It is our understanding that currently the task of walking across floors of differing textures can offer significant challenges to modern bipedal and quadrupedal robots. Work has already begun to use CRL to create a contextually discerning controller for a simply toy problem which has been modified to require context discernment to be solved.

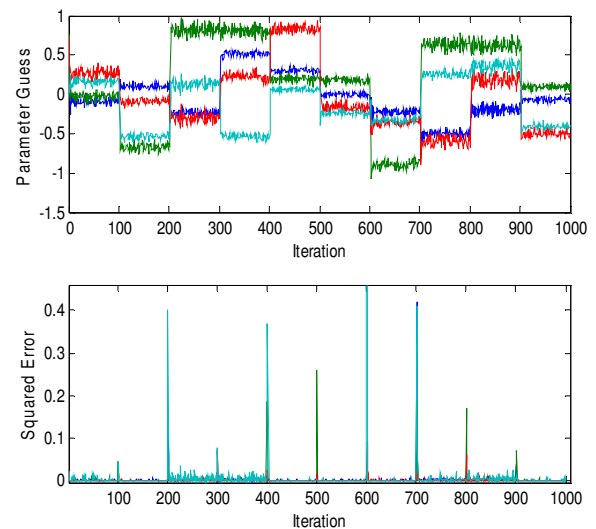


Figure 4

References

- [1]J. Si, A. Barto, W. Powell, and D. Wunsch (2004) (Eds.), *Learning and Approximate Dynamic Programming*, Wiley.
- [2]Prokhorov, D., Feldkamp, L., and I. Tyukin, "Adaptive Behavior with Fixed Weights in Recurrent Neural Networks: An Overview," *Proc. of International Joint Conference on Neural Networks (IJCNN), WCCI'02*, Honolulu, Hawaii, IEEE Press.
- [3]Santiago, R. A. & G. G. Lendaris (2004). "Context Discerning Multifunction Networks: Reformulating Fixed Weight Neural Networks." *Proc IJCNN-04*, Budapest, Hungary, IEEE Press.
- [4]Werbos, P.J. (1992), "Approximate Dynamic Programming for Real-Time Control and Neural Modeling," Chapter 13 in *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, (White & Sofge, eds), Van Nostrand Reinhold, New York, NY.