
Learning with Holes: Pitfalls with Simulation in Learning Robot Control

Dean F. Hougen

HOUGEN@OU.EDU

Robotic Intelligence and Machine Learning Laboratory, School of Computer Science, University of Oklahoma, Norman, OK 73019 USA

Abstract

Machine learning and simulation are important tools for creating intelligent robotic systems for autonomous space applications. However, there are potential pitfalls to using these two tools in conjunction. This paper presents three case studies that expose some of these pitfalls and suggests an approach to avoiding them.

1. Introduction

In recent years, machine learning has become increasingly important as a method by which autonomous robots can increase their capabilities, often significantly outperforming the best hand-coded approaches to the same task (e.g., Stone, 2000). The use of simulation is necessary if we are to apply machine learning to robot control for autonomous space applications. This is because the environments in which our robots will ultimately be deployed are (quite obviously) remote.

This brings to the fore a false dichotomy often made by roboticists: You are either working in simulation or on a “real robot.” However, while the use of physical robot hardware may allow us to advance our research in ways that would be difficult or impossible to achieve using computer simulations alone, moving a physical robot around in a test environment that differs in some way from the final deployment environment *is* a simulation. It is just not a *computer* simulation. This means that *all* machine learning for autonomous space applications that takes place before deployment involves learning in simulation.

Unfortunately, there are several pitfalls present in the combination of machine learning and simulation that are not there in either of these components alone. Section 2 presents three case studies in which these

pitfalls have manifested themselves. Each case study includes discussion of learned vs. hand-coded control policies with an emphasis on why a hand-coded approach would likely have avoided these pitfalls. The case studies are followed, in Section 3 with both discussion of non-solutions to these problems and a suggestion for a possible meliorating strategy.

2. Problems with Simulation: Three Case Studies

Beyond the well-known problems of simulation in robotics research in general, there are several additional pitfalls possible when simulation is combined with machine learning for robot control, as the following three case studies make clear. In each case, there was a “hole” in the simulation environment. That is, there was a problem with the simulation environment provided to the learning system that allowed the learning system to learn an inappropriate control policy. Moreover, in each case it is likely that a hand-coded control policy tested in this simulation environment would have avoided this pitfall.

2.1. Case One: A Hole in the Code

The cart-and-pole task is a classic one in both dynamics (e.g., Cannon, Jr., 1967) and machine learning (e.g., Widrow & Smith, 1964). A pole is hinged to a cart that moves on a track of finite length. The objective is to keep the pole upright by moving the cart while simultaneously preventing it from reaching either end of the track. The dynamics task is solved by hand-crafting a set of equations that specify the impulse that should be given to the cart based on its current position and velocity and the pole’s angle and angular velocity. The machine learning task is solved by creating a system that can learn this relationship.

Nearly all machine-learning researchers who have pub-

lished on this topic have used computer simulations of carts and poles and, when I first studied this task, so did I (Hougen, 1993). I chose to implement my simulation using the physical parameters given by Barto, Sutton, and Anderson (1983). Unfortunately, unknown to me at the time, I introduced an error into the physics equations for the pole as it fell. Moreover, I had research using this simulation accepted for publication and was in the process of writing the full paper before I discovered the error.

That I did discover the error is, of course, why I am able to include this case study in the present paper. If I had not found this error, then my 1993 paper would simply have been published with flawed results, joining an unknown number of other machine learning papers also published with flawed results because of errors that were never caught in simulations of various systems. The instructive question here is: How did I catch this error?

The equally instructive answer is: By happenstance. While I was writing the full paper, I made screen captures of the GUI that I had built to help visualize the system as it learned. As I did so I noticed something peculiar – lasting only a fraction of a second – about the movement of the simulated cart and pole on the screen. I ran it several more times until I saw another momentary peculiarity. I pulled up my code for the physics of the cart and pole and went through it carefully, reviewing every character – comparing it with what I thought it should be. That is when I found the error.

The reason I think this answer is instructive is that it contrasts so nicely with how we might catch the same error in simulation code for testing a hand-coded controller. If both the controller and the *plant* (the system to be controlled) are hand-coded, then an error in one may cause a significant mismatch between the two, leading to the controller failing to control the plant. With a learned controller, however, if the learning system can learn to control the given plant, it will, regardless of whether that plant is the one we intended to provide. It will only fail to control the plant if it is inherently more difficult for the system to learn control for the plant provided.

Further, even if the mismatch between controller and plant is not significant enough to cause a dramatic control failure, the discrepancy is still likely to be readily detectable. This is because we can *predict* how the controller and plant should interact; we have detailed equations describing both and can use these to make calculations regarding system behavior. With the cart and pole system, for example, we can predict the min-

imum and maximum value that will be seen for any of the state variables (e.g., cart position), their minimum and maximum rates of change, their periodicity, etc. If the observed values do not match the predicted values, we know there is a problem. With a learned controller, however, we cannot predict beforehand how it will behave because we don't know ahead of time what control policy it will learn. After all, if there were only one policy that will be most effective, and we know what that policy is, then we would be foolish not to hand code it. However, if we don't know what this policy is (or even if there is only one such policy), then we cannot predict how the controller and plant will interact once that policy is discovered.

2.2. Case Two: A Hole in the Model

Many researchers studying machine learning for robots are today using robotic soccer as a testbed (e.g., Stone, 2000) and I am no exception. In particular, I have begun to apply my machine learning ideas within the popular RoboCup Soccer Server simulation environment (Chen et al., 2002). Case two revolves around the collision model provided by the system (Chen et al., 2002, p. 35):

If at the end of the simulation cycle two objects overlap then the objects are moved back until they do not overlap. Then the velocities are multiplied by -0.1. Note that it is possible for the ball to go through a player as long as the ball and the player never overlap at the end of the cycle.

The fact that the objects only collide if they overlap at the *end* of a simulation cycle, resulting in the possibility that a ball may move *through* a player, is quite difficult to justify.

To give a concrete example of this problem, imagine two opposing players standing a short distance apart, facing one another, with the ball directly in front of one of the players. Given the collision model in place, the player with the ball can use a policy of kicking the ball directly at the opponent and allowing the ball to pass through.

There is a problem with creating a player that follows this policy if you are planning to implement your policies on physical robots at some point: The policy will fail. In the physical world, the ball will not pass through the opponent but will, instead, bounce off or be intercepted. This will lead to an unexpected next state, at least from the perspective of the simulated experience.

If we want to avoid this policy with our hand-coded agents, we can simply tell them not to follow it. Fur-

ther, it is unlikely that someone hand-coding robot soccer agents would even need to know of this hole in the simulation in order to avoid accidentally exploiting it. Instead, he or she would know that solid objects don't just pass through one another and develop a policy appropriate to that.

If we want to avoid this policy with our learning systems, however, we will have a more difficult time. Our learning systems do not know to design policies that work in the "real world" – the only world they know is the simulation environment in which they are placed. What our systems know, from experience, is what works. They will try out possibilities that never occurred to their inventors. In many cases, that is their strength (e.g., Thompson & Layzell, 1999). When learning in a simulation with holes, it is their weakness.

2.3. Case Three: A Hole in the Distribution

While the first two cases involve simulation environments that didn't match up with the worlds they were intended to model, the third case involves a simulation environment that adequately matched the world but a use of that environment that did not. Specifically, the problem was that the distribution of training samples differed between those provided to the implementation that learned using the computer simulation and the one using the physical robot.

In this case, the learning system was to learn a policy for the trailer-backing task. In this task, the objective is to back a truck and trailer rig to a goal by steering the front wheels of the truck. We implemented two copies of a learning system for this task and tested one using a computer simulation and the other using a "mini-robot" truck and trailer rig roughly two feet in length. Each system learned independently but their learning rates and overall success rates – the standard measures for reinforcement learning systems – were quite similar, leading us to initially believe that they had learned similar quality solutions to the task, despite the different media in which they had been embedded. We were interested in knowing if the solutions were of similar quality, rather than knowing if the solutions were identical, because there were a great many possible solutions of approximately equal quality and we did not care which one was learned.

Observations of the physical robot led us to question whether the solutions learned were, in fact, of similar quality, despite what our numeric data told us. Additional details have been published previously (Hougen et al., 1999), however, suffice it to say that the data were misleading and that the system that learned us-

ing the physical robot learned a quantitatively poorer solution to the task than the one that learned using the computer simulation. This was due to the fact that the system using the physical robot used a set of initial hitch and goal angle states that failed to sample as thoroughly from larger (and more difficult) angles as did the one using the computer simulation.

This problematic fact was obscured by the way in which learning was done: As is standard in reinforcement-learning research, we did not have separate learning and testing sets. Instead, we let the system perform the task and recorded its success rate while it did so. This makes sense in the context of this type of learning because reinforcement-learning systems are supposed to improve their performance while carrying out the very task that they are learning. Unfortunately, because the distribution of initial states that was being tested over was the same as that being used for learning (they were, in fact, the very same states) the systems were able to produce similar numeric data while learning dissimilar solutions. This was only discovered because I watched the physical robot as it moved around and saw it failing on initial states at which I "knew" (strongly suspected) it should be able to succeed. It wasn't the number of times it succeeded or failed (which is what our numeric data reflected), it was the situations in which it failed.

Again it is instructive to compare this to the situation faced when designing a hand-coded controller for this system, rather than using machine learning. In the hand-coded instance, we would typically specify the boundaries of the control regions within which the solution should work. We can then test the system with initial states that fall both within and without those boundaries, paying particular care to sample extensively along them. If the system fails within the control boundaries that we have deduced, we know that there is a problem. This is a much more rigorous and trustworthy method than relying on the suspicions of an observer to whom things just look wrong.

3. Patching the Holes

Before I attempt to offer a suggestion as to how to address these difficult problems in using simulation for machine learning for autonomous space applications, let me discount several possible alternatives to the one I will put forward.

3.1. Non-Solutions

The solution to these problems is not to abandon the idea of using machine learning to determine robot poli-

cies. As mentioned in the introduction, learned solutions may significantly outperform the best hand-coded solutions on the same task. Giving up on machine learning means giving up the possibility of using these better solutions. Moreover, abandoning machine learning is not necessarily a path to more reliable systems – besides creating faster, smaller, or cheaper solutions, it is very possible to use machine learning methods to develop systems that are more robust, reliable, or fault-tolerant than systems designed using conventional methods (e.g., Thompson, 1997).

Nor is constraining what the system is allowed to learn the solution. While some people might be reassured by limiting a learning system to merely “tweaking” parameters in a conventional approach to a task, rather than letting the system learn its own (perhaps radical) approach, this, like abandoning machine learning altogether, is to give up many of the potentially best solutions (Thompson & Layzell, 1999).

Switching from simulation to learning “in the real world” is also not an option. While the use of physical robots in research has allowed us to learn much about robot control that would not have been learned using computer simulations alone, we should not fall into the trap of believing that experiments with physical robots are necessarily more accurate than computer simulations. For example, because of significant differences between the Earth and Mars (e.g., lower gravity and thinner atmosphere on Mars), a physical rover moving around at the Mars Yard at JPL or here in Oklahoma at the Mars Analog Field Site should be considered a simulation just as much as a computer simulation of a rover on Mars (where many of these differences can be included). Again, for space applications, we’ll need to learn in simulations of one sort or another.

Of course, ignoring these problems is not the solution, either. Better performance, in terms of the speed or efficiency with which a task is accomplished, means little for expensive propositions such as autonomous space applications if confidence in the system must be sacrificed to achieve it.

3.2. Approach

I wish I could say that I have *solutions* to these problems. However, the best I can offer is an *approach*. This approach has three parts: analysis, prediction, and testing.

While the problems highlighted in these three case studies came from three different kinds of holes, I believe that all of them could have been systematically detected through this approach. If, for example, a

final, learned controller for the cart-and-pole system had been analyzed mathematically to understand its functioning, we could have predicted how it would behave given a set of initial conditions, then compared its actual behavior to those predictions, just as we could do with a hand-coded controller.

Of course, this is easier said than done. The analysis of learned solutions is in its infancy but is beginning (e.g., Thompson & Layzell, 1999). In our case, this involves directly analyzing control policies learned by the robot. This analysis could use methods normally applied to the construction and analysis of hand-coded solutions, such as hybrid dynamic control (Fierro & Lewis, 1997). Alternately or in conjunction, this could involve the use of learning frameworks that are more transparent, such as learning Bayesian networks (Heckerman et al., 1995) as opposed to notoriously opaque methods, such as standard feedforward artificial neural networks (such as backprop, quickprop, or rprop nets).

Finally, for testing, the use of multiple test environments is desirable. A physical rover at the Mars Analog Field Site may be a simulation but it is a very different simulation than the computer simulation and each may reveal problems in the other.

3.3. Approach in Practice

To implement this approach, we must first determine the failure modes that may result from the pairing of machine learning with simulation. Three different failure modes were presented in the three case studies:

Simulation Code Failure A mismatch between the code and the mathematical model of the environment that the code was meant to capture.

Environment Model Failure A mismatch between the environment model and the ultimate destination environment.

Sample Distribution Failure A mismatch between the frequencies of certain events encountered under different training regimens.

This is not meant to be an exhaustive list of possible failure modes, merely likely failure modes in many pairings of machine learning with simulation. Note that only simulation code failure is particular to computer simulations. Environment model failure could be caused by a mismatch between the mathematical model underlying a computer simulation (as shown in case two) or a mismatch between a physical analog test site and the physical environment that test site is meant to model. Finally, sample distribution failure has more to do with how a simulation environment is

used, rather than anything inherent to the environment itself, including whether it is a computer simulation or a physical environment.

Therefore, while methods for software verification from computer science might be appropriate in helping to address the simulation code failure mode, they may not help with environment model failure and will not help in sample distribution failure. Moreover, because of difficulties in applying these methods to simulations of the physical world (e.g., the mismatch between the assumption made in most model checking methods that there are a finite number of states to the system, whereas our simulations will generally be of continuous phenomena), alternatives to these methods should be found for all failure modes identified.

Analysis, then, refers not to analysis of the simulation code or even the environment model. It refers to analysis of the policy learned by the system. This requires that the policy be expressed in a format amenable to such analysis, which is often not the case. For example, many of the control policies cited previously (e.g., Barto et al., 1983; Hougen, 1993) divide the state space into discrete regions within which a different system response is specified, then cobble together a control policy by combining these independent responses. Such an overall policy is not amenable to representation using continuous equations. Moreover, some methods (e.g., Barto et al., 1983) make analysis more difficult by specifying a probabilistic control policy, rather than a deterministic one. An approach that learns a continuous control equation instead (e.g., Koza, 1992) would simplify this analysis.

Once the analysis has been performed, we can predict how the resulting system will behave. Ideally this would allow for behavioral guarantees to be made. However, in many cases such guarantees are unlikely to be possible. Nonetheless, we should be able to predict system performance under specific expected conditions. Such predictions can then be tested using all available simulation environments.

It should be noted that this approach will cannot ensure that these failure modes are not encountered. However, this approach will verify that the policy is appropriate to the same extent that a hand-coded policy could be verified, allowing us to place the same level of confidence in the policy arrived at through machine learning as any hand-coded policy.

Acknowledgments

Thanks to Mark Woehrer for bringing my attention to some of the literature cited herein and to the School of

Computer Science, the College of Engineering, and the Office of the Provost at the University of Oklahoma for supporting my participation in this workshop.

References

- Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-13*, 834–846.
- Cannon, Jr., R. H. (1967). *Dynamics of physical systems*. New York: McGraw-Hill.
- Chen, M., Foroughi, E., Heintz, F., Huang, Z., Kapetanakis, S., Kostiadis, K., Kummeneje, J., Noda, I., Obst, O., Riley, P., Steffens, T., Wang, Y., & Yin, X. (2002). *RoboCup soccer server users manual for soccer server version 7.07 and later* (Technical Report). The RoboCup Federation.
- Fierro, R., & Lewis, F. L. (1997). A framework for hybrid control design. *IEEE Transactions on Systems, Man, and Cybernetics, 27-A*, 765–773.
- Heckerman, D., Geigerr, D., & Chickering, D. M. (1995). Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning, 20*, 197–243.
- Hougen, D. F. (1993). Use of an eligibility trace to self-organize output. *Science of Artificial Neural Networks II, Proceedings SPIE* (pp. 436–447).
- Hougen, D. F., Rybski, P. E., & Gini, M. (1999). Repeatability of real world training experiments: A case study. *Autonomous Robots, 6*, 281–292.
- Koza, J. R. (1992). A genetic approach to finding a controller to back up a tractor-trailer truck. *Automatic Control Conference* (pp. 2307–2311).
- Stone, P. (2000). *Layered learning in multi-agent systems: A winning approach to robotic soccer*. MIT Press.
- Thompson, A. (1997). Evolving inherently fault-tolerant systems. *Proceedings of the Institute of Mechanical Engineers, 211*, 365–371.
- Thompson, A., & Layzell, P. (1999). Analysis of unconventional evolved electronics. *Communications of the ACM, 42*, 71–79.
- Widrow, B., & Smith, F. W. (1964). Pattern-recognizing control systems. *COINS (Computer and Information Sciences Symposium)* (pp. 288–317). Washington, DC.