Finding the Limits of Machine Learning in Optimization

Problem Background

Autonomous path planning is fundamental to safely integrating robots, self-driving cars and other mobile, automated systems into society. Safety is the primary requirement of such systems, but business viability imposes additional requirement such as efficiency, speed and robustness. Additionally, if robots are to improve society, they need to navigate human spaces in a way that is aesthetically and socially pleasing. Given machine learning's great success in providing algorithmic solutions to essentially human-centered problems, e.g., the board game Go, it is natural to ask whether it can also provide better path planning for robots.

A Simple Path Planning Problem

To initially test machine learning's capacity to plan robot motion, we create the following artificial problem:

- A robot is placed at a random location in the Cartesian plane with a random velocity.
- The robot's task is to reach the origin of the plane as quickly as possible.
- The robot has a maximum velocity it can achieve.
- The robot has a maximum acceleration it can apply.
    - The acceleration can be applied in any direction.
    - The acceleration can change instantly.

We attempted to solve this problem using an Actor-Critic system based on the Actor-Critic network used to solve the Lunar Lander environment of the Open AI gym. The Lunar Lander problem is to use machine learning to play the old "Lunar Lander" arcade game. It has been robustly solved using a few different methods.

The Lunar Lander has five parameters: position in the Cartesian plane, (x, y), orientation, θ, and velocity, (v_x, v_y). It has thrusters which allow it to apply force in three directions fixed to its body. Its task is to land upright on the landing pad. This set of tasks seemed similar enough to the robot planning problem for the solution to easily adapt.

When we applied the Lunar Lander's Actor-Critic network to the robot path problem, we found that we could not achieve a stable or robust solution. Most of the time, we did not reach a network which would travel to the origin and stop. We could occasionally create parameters and sequences of runs which would perform this task; however, these solutions could easily disappear if we trained any further.

The network would typically reach a solution where it commanded accelerations along one of the four diagonal directions. It would also typically move the robot to one of the axes, and then it would remain in place or slowly move away from the origin.

The Actor-Critic network depends upon the cost function used to evaluate performance. We believe our cost function rewarded getting to the origin as quickly as possible, but it may be the problem is in our cost function. The robot's progress is simulated and monitored at distinct time increments. The robot is rewarded every step. There are threecases:

- We are close enough to the target to consider ourselves finished:
    - $K \cdot N_L \cdot \left(1 - \frac{1}{2}\sqrt{\min\left(\frac{|v|}{v_{stop}}, 4\right)}\right)$
    - N_L is the number of time increments that the simulation is allowed to proceed, this ensures that the reward for finishing is greater than the accumulated reward for any other outcomes.
    - The $\left(1 - \frac{1}{2}\sqrt{\left(\min\left(\frac{|v|}{v_{stop}}, 4\right)\right)}\right)$ term penalizes the robot if it cannot reduce its velocity to zero at the origin.
- We have moved beyond a reasonable distance to the origin
    - $-K_F\, N_L$
    - This ensures the robot loses at least as many points as it could have gained from any other option if it is going away from the origin with no clear return.
- We are neither close nor far from the origin
    - $(R_N - r) \cdot \left(1 - \frac{1}{2}\sqrt{\min\left(\frac{|v|}{v_{stop}}, 4\right)}\right)$
    - This rewards you if you are within R_N, and penalizes you if you are outside that radius.
    - The velocity term rewards slowing down as you approach the origin.

Further, the network's training depends upon several parameters and the choice of nonlinear elements. These choices may have been flawed in some fashion.

Questions

Is there actually a simple solution to the basic path planning problem?

If the robot has its velocity directed parallel to the line joining it with the origin, there is a provably best control algorithm:

- Accelerate toward the origin, until you reach the maximum velocity.
    - The maximum velocity may be set by the robot's maximum velocity or by your distance to the origin.
- When you are at the correct distance to stop on the origin when you decelerate at your maximum, begin decelerating at your maximum.

In any other orientation, it seems like a good solution is to accelerate at your maximum rate toward the origin but proving this seems hard. The fact that your maximum velocity is constrained also makes this more difficult.

Is this a properly posed question?

Our system has few constraints. The Lunar Lander operates in the presence of gravity and the acceleration is constrained, at least in direction. Might it be that our problem is not very stable, and the numerical limitations of network learning and approximation cannot easily develop a robust solution?

One interesting sideline is to ask whether other classical optimization problems could be solved through a machine learning algorithm. Could a neural network solve the brachistochrone problem? The robot control problem feels like a similar variational problem.

Is there a network design and training combination which makes this a robustly solvable problem?

There are several parameters to select when creating the network. The depth and width of the network and the type of nonlinearities used seem particularly important. Further, training depends upon allowing the network to see a sufficient representation of the robot's parameter space. How do we optimize training trajectory, the learning rate and other parameters controlling how the network samples the function it is trying to approximate? What would a good network approximation of the good control law be?

How do we develop trust in machine learning for path planning?

The path planning posed seems so simple that machine learning should be able to solve it. Further, it is so simple that we can see when machine learning is performing poorly. In real path planning, the environment and the problem are significantly more complex, and the solutions are not obvious.

The most dangerous outcome is for a machine learning algorithm to provide a plausible path-planning network, which is unreliable and sensitive to input conditions. Are there elements of the simple problem and the success or failure of networks to solve it which can guide our evaluation of more complex problems?

References

The following git-project uses Deep Q-Networks (DQN) to solve the lunar-lander problem. It also has references to some papers on the subject.

https://github.com/svpino/lunar-lander

This git-project uses Actor-Critic Networks and has some references.

https://github.com/nikhilbarhate99/Actor-Critic-PyTorch

This may be a signal-to-noise problem. The simple control problem may have significant structure and may be a dynamical system. The network and its training algorithm may be insufficient to determine this structure in the presence of numerical noise and insufficient sampling. The network may be insufficient to represent the control solution. The following references provide some background on the effect of noise in training on machine learning:

"Noise Tolerance Under Risk Minimization," Naresh Manwani and P. S. Sastr, IEEE Transactions on Cybernetics, VOL. 43, NO. 3, JUNE 2013, pp. 1146-1151.

“Dealing with Noise Problem in Machine Learning Data-sets: A Systematic Review," Shivani Gupta and Atul Gupta, Procedia Computer Science 161 (2019), 466–474.

“A Survey of Machine Learning Approaches to Robotic Path-Planning", M.W. Otte, University of Colorado at Boulder

“Deep Learned Path Planning via Randomized Reward-Linked-Goals and Potential Space Applications”, T. Blum, W. Jones, and K. Yoshida, (arxiv.org)