Reinforcement Learning for CPS Safety Engineering

Sam Green, Çetin Kaya Koç, Jieliang Luo
University of California, Santa Barbara
Motivations
Safety-critical duties desired by CPS?

- Autonomous vehicle control: UAV, passenger vehicles, delivery trucks
- Automatically responding to, or preventing, damage
- Industrial robot control for use around humans
- Large process automation
  - E.g., optimization of factory
Reinforcement Learning
Machine Learning

- Supervised
- Unsupervised
- Reinforcement
Introduction to RL

• A computational approach to **learning from interaction**
  • Established in the 1980s
  • Objective is to take actions to maximize a reward (or minimize a cost)
  • Seen as a path toward Artificial General Intelligence

• RL is at the intersection between
  • Psychology
  • Control Theory
  • Computer Science/AI

• Resurgence with advent of deep learning methods
# Advances in RL since 2015

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>8 days on GPU</td>
<td>121.9%</td>
<td>47.5%</td>
</tr>
<tr>
<td>Gorila</td>
<td>4 days, 100 machines</td>
<td>215.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>D-DQN</td>
<td>8 days on GPU</td>
<td>332.9%</td>
<td>110.9%</td>
</tr>
<tr>
<td>Dueling D-DQN</td>
<td>8 days on GPU</td>
<td>343.8%</td>
<td>117.1%</td>
</tr>
<tr>
<td>Prioritized DQN</td>
<td>8 days on GPU</td>
<td>463.6%</td>
<td>127.6%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>1 day on CPU</td>
<td>344.1%</td>
<td>68.2%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>4 days on CPU</td>
<td>496.8%</td>
<td>116.6%</td>
</tr>
<tr>
<td>A3C, LSTM</td>
<td>4 days on CPU</td>
<td>623.0%</td>
<td>112.6%</td>
</tr>
</tbody>
</table>

*Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric.*

Terminology

- **Agent** – The thing we are learning to control
- **Environment** – All the factors affecting the agent
- **Action** – Performed by agent in an attempt to affect change on the environment
- **Reward** – Returned by the environment to the agent after the agent makes an action. Used to help the agent learn.
  - AKA the negative cost
Markov Decision Process

• What RL solves
• Environments where agent’s decisions are only dependent on present
  • An object in flight
  • Self-driving car
  • Manufacturing process
  • Robot control
• It’s not that the past doesn’t matter, but the laws of physics guarantee certain things, e.g. momentum
• Methods also exist to solve approximate MDP
Example: Student Markov Chain

Start here at the beginning of each episode

[http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf]
RL for CPS Safety Engineering

• Interdisciplinary natures makes RL interesting for CPS engineering
  • AI, ML (Math, Statistics)
  • Mechanics design and simulation (ME, Physics, CS)
  • Programming and implementation (CS, EE)
Mountain Car Example
Canonical example: Mountain Car

- Agent is an underpowered car with 3 actions:
  - Backward, Neutral, Forward
- Reward := -1 per timestep
  - Implicit goal := Reach the flag as fast as possible
- State := x-pos and velocity

Model-Free Control via Policy-Based RL

- A simple physics model determines the behavior of car
  - Captures position of the car on the hill
  - Captures effect of limited engine power
- Using a physics model simplifies approach
  - Use an efficient traditional controller
- But in many scenarios the model is not available or too complex
  - Amazon package delivery drone
- Solve mountain car using sophisticated method as toy example
  - Directly train a neural network-based policy
RL Terminology and Notation

• $S_t$ – State of the environment at time $t$
  • x-axis position and velocity
• $A_t$ – Action taken by agent at time $t$
  • Backward, Neutral, Forward
• $\pi$ – The policy function; returns the next action to take. Stochastic in this example
• $\theta$ – A parameter vector for the policy; i.e. the weights learned in a neural network

Putting everything together:
$$A_{t+1} \sim \pi_{\theta}(A_t, S_t) = P(A_t \mid S_t, \theta)$$
The policy $\pi_\theta$

- $\pi_\theta$ is often approximated
- Deep neural networks are powerful for approximation
- We will use gradient ascent to optimize the DNN
The policy function $\pi_\theta$, approximated by NN

- **State information at time $t$:**
  - Position and Velocity
- **Action options at time $t$:**
  - Forward acceleration
  - Neutral
  - Backward acceleration
Reward function

- At every time step take an action
  - Forward, neutral, backward
  - Each action has a reward of -1
  - Train agent to reach the flag in minimum time steps
Example: Markov Reward Process

[http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/MDP.pdf]

Start here at the beginning of each episode
How to train the NN?

- Small networks can be effectively trained with genetic algorithms.
- Genetic algorithms work poorly with large networks (parameter space is too large).
- Gradient-ascent optimization works with large parameter space.
Monte-Carlo Policy Gradient (REINFORCE)

• Find DNN parameter vector $\theta$ such that $\pi_\theta$ maximizes the reward

• For every episode, until flag is reached
  • Get state information (position & velocity) from environment
  • Feed NN with state information
  • NN will output a probability for (F)orward, (N)eutral, and (B)ackward
  • Randomly select action F, N, and B (using the above probabilities)
  • Store the state information and action taken

• Once flag is reached
  • Assign the most reward to the last action ... least reward to the first action
  • Update $\theta$ s.t. actions made at the end are more probable

[http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html]
Monte-Carlo Policy Gradient

• Method leverages methods created for supervised learning
  • Inputs := the state information (position, velocity)
  • Predictions := forward, neutral, or backward action taken
  • Labels ("ground truth") := After the episode was over, assign most value to the last actions. Assign least value to the first actions

• Run many episodes, after each episode finishes (flag is reached) strengthen the network such that the last moves become more probable

[http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html]
Gradient-ascent

• Gradient algorithms find a local extremum
• At end of each episode, adjust each parameter in $\theta$ s.t. actions made near the end are strengthened
• How much and in which direction to move each parameter is determined by the backpropagation method
Caveats

• Deep RL is usually slow to learn

• Transferring knowledge from one problem to another is difficult

• Reward function can be complex
Safety and Security Considerations
Adversarial Examples In The Physical World
Safety and Security Considerations

• DNNs are black-box models
  • Possible to give an input which causes DNN to provide wild output

• Efforts to mitigate this limitation
  • E.g. Constrained Policy Optimization
Constrained Policy Optimization

- School-book RL specifies only the reward function
  - Problem: when an agent is learning, it may try anything
  - Potentially unsafe when training is in physical environment
- Constraints can be added to the objective function

[Achiam et al. “Constrained Policy Optimization”, 2017]
Current Efforts
Developing RL for Quadcopter Control

• Good case study for complex autonomous CPS
  • Collision avoidance
  • Target tracking
  • Package delivery

• Using open source firmware and hardware
Using Microsoft AirSim for 1\textsuperscript{st}-order learning

Conclusions

• RL is a generalizable method to tackle many CPS decision making problems
  • High-capacity models can make sophisticated decisions

• Good approach for CPS education, because of interdisciplinary nature

• Open problems when using black-box functions for safety applications
Questions?