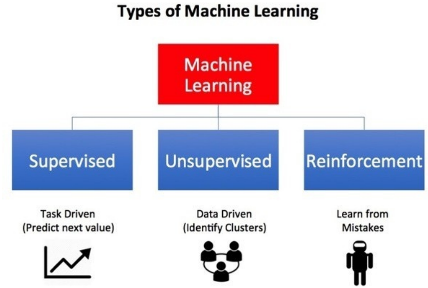
Reinforcement Learning and Control

Luis Pimentel

ECE 4803 Mathematical Foundations of Data Science

November 24, 2020

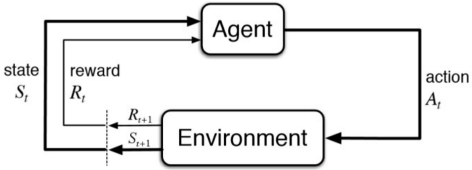
Reinforcement Learning: Machine Learning Perspective



Goal of Reinforcement Learning

Learn to how behave in an unknown and uncertain enviroment using data from previous behavior.

Definitions



- Agent: system that performs actions affecting its state
- Environment: the world in which the agent performs
- State S_t: particular condition of an agent at a specific time
- Action A_t: activity performed by the agent
- Policy π_t: state to action mapping (what you are trying to learn!)
- Reward R_t: feedback from the environment as a result of action

Optimization Problem Formulation

- Let $\boldsymbol{\tau}$ represent a state-action trajectory s.t. $\boldsymbol{\tau} = [\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \dots; \boldsymbol{s}_{N-1}, \boldsymbol{a}_{N-1}, \boldsymbol{s}_N]$
- Let $f(\boldsymbol{s}_t, \boldsymbol{a}_t, e_t)$ represent the dynamics of the system with Gaussian noise e_t.

$$\underset{\pi}{\text{maximize}} \quad \mathbb{E}\Big[R(\boldsymbol{\tau})\Big] \qquad\qquad R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{s}_t, \boldsymbol{a}_t, t)$$

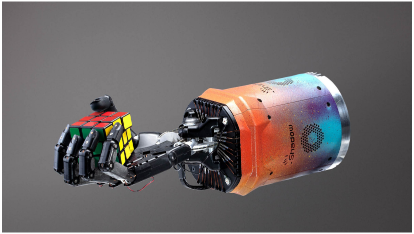
subject to

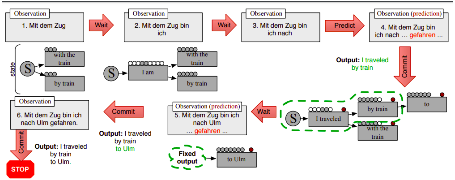
$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, e_t)$$

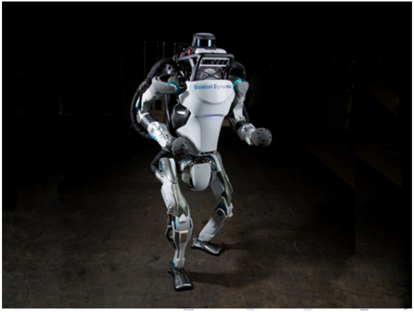
$$\boldsymbol{a}_t = \pi(\boldsymbol{\tau})$$

- Our goal is to optimize our policy such that we maximize our reward.
- We set up our reward function such that
  - "good" action leads to a "good" state, thus a positive reward
  - "bad" action, which leads to a "bad" state, thus a negative reward

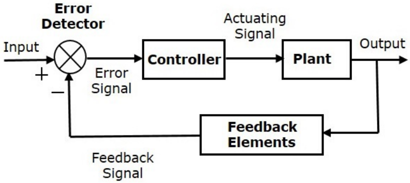
Reinforcement learning applications:







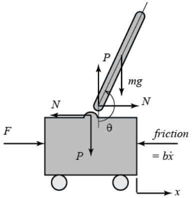
Control Systems



$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)$$
$$\boldsymbol{y}(t) = \boldsymbol{C}\boldsymbol{x}(t)$$

- $\dot{\boldsymbol{x}}$: differential equation describing how the state changes
- $\boldsymbol{x}$: state of the sytem
- $\boldsymbol{u}$: input to the system
- $\boldsymbol{y}$: output of the sytem
- $\boldsymbol{A}$,$\boldsymbol{B}$,$\boldsymbol{C}$: state and control transition matrix, state observation matrix

Example: Inverted Cart-Pendulum Linearized Model



$$(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x}$$
$$(M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} = u$$

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mnl^2} \end{bmatrix} u$$

Reinforcement Learning and Control

What happens when we don't have a model??

Can we still control a system without understanding the detailed mathematics and physics behind it?

We can reformulate the control problem as a Reinforcement Learning problem and learn the system input policy.

- $\boldsymbol{\tau} = [\boldsymbol{x}_0, \boldsymbol{u}_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \ldots; \boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}, \boldsymbol{x}_N]$
- We parameterize our policy by $\boldsymbol{\theta}$.

$$\underset{\boldsymbol{\theta}}{\text{maximize}} \quad \mathbb{E}\Big[R(\boldsymbol{\tau})\Big] \qquad\qquad R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{x}_t, \boldsymbol{u}_t, t)$$

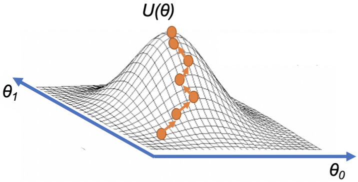
subject to

$$\boldsymbol{x}_{t+1} = \dot{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, e_t)$$

$$\boldsymbol{u}_t = \pi(\boldsymbol{\tau}, \boldsymbol{\theta})$$

Solving the optimization problem

We can solve this problem using our favorite method: gradient ascent!



$$\boxed{\underset{\boldsymbol{\theta}}{\text{maximize}} \;\; J(\boldsymbol{\theta})}$$

Algorithm 1: Gradient Ascent for Policy Parameterization

Result: $\boldsymbol{\theta}$

initialize: $\boldsymbol{\theta} = \boldsymbol{\theta}_0$, γ: learning rate

while *unconverged* **do**

$\quad \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i) = \text{compute\_policy\_gradient}(\boldsymbol{\theta}_i)$

$\quad \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$

Policy Gradients

- If we don't have a model of our system, how do we calculate the policy gradient?
  - Solution: we can use input and output data to create an approximation of the gradient.
  - Apply an input, observe the output and reward
- Estimating the Policy Gradient is a major research area in Reinforcement Learning.
- Two methods of Policy Gradient estimation:
  - Finite-Difference Methods
  - Episodic REINFORCE

Finite Difference Method

Main idea: Apply a perturbation $\delta\boldsymbol{\theta}$ to our θ parameter over many rollout trajectories to create an estimation of the Policy Gradient.

1)

$$J(\boldsymbol{\theta}+\delta\boldsymbol{\theta}_m) = J(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$

$$J(\boldsymbol{\theta}+\delta\boldsymbol{\theta}_m) - J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$

$$\Delta\hat{J}_m = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})^T \delta\boldsymbol{\theta}_m$$

2) set up a Least-Squares Regression Problem:

$$\begin{bmatrix} \Delta\hat{J}_1 \\ \Delta\hat{J}_2 \\ \dots \\ \Delta\hat{J}_m \end{bmatrix} = \begin{bmatrix} \delta\boldsymbol{\theta}_1^T \\ \delta\boldsymbol{\theta}_2^T \\ \dots \\ \delta\boldsymbol{\theta}_m^T \end{bmatrix} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\Delta\hat{\boldsymbol{J}} = \Delta\boldsymbol{\Theta}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\boxed{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left(\Delta\boldsymbol{\Theta}^T \Delta\boldsymbol{\Theta}\right)^{-1} \Delta\boldsymbol{\Theta}^T \Delta\hat{\boldsymbol{J}}}$$

Finite Difference Method Algorithm

$$\boxed{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \left(\boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta\Theta}\right)^{-1} \boldsymbol{\Delta\Theta}^T \Delta \hat{\boldsymbol{J}}}$$

Algorithm 2: Finite Difference Gradient Estimator

Result: θ

initialize: $\boldsymbol{\theta} = \boldsymbol{\theta}_0$, γ: learning rate, m: rollouts

σ^2: variance, ϵ: Gaussian noise

while *unconverged* **do**

  for *m rollouts* **do**

    $\delta\boldsymbol{\theta}_m = \sigma\epsilon$

    $\Delta\hat{J}_m = J(\boldsymbol{\theta} + \delta\boldsymbol{\theta}_m) - J(\boldsymbol{\theta})$

  $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i) = \left(\boldsymbol{\Delta\Theta}^T \boldsymbol{\Delta\Theta}\right)^{-1} \boldsymbol{\Delta\Theta}^T \Delta \hat{\boldsymbol{J}}$

  $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$

Finite Difference Method Considerations

- Very efficient for deterministic systems.
- Different configurations:
  - How to generate the perturbation $\delta\theta$
  - How to compute $\Delta\hat{J}$
  - How to choose convergence criterion
  - Tuning of other hyper-parameters
- Have been successfully implemented on robotics systems.
- Require close supervision of robotics engineer as badly generated $\delta\theta$ can lead to de-stabilization and failure in optimization.

Episodic REINFORCE

- In Episodic REINFORCE we consider path probabilities as the basis of our objective function.
- $\boldsymbol{\tau} = (\boldsymbol{x}_0, \boldsymbol{u}_0, \ldots, \boldsymbol{x}_{N-1}, \boldsymbol{u}_{N-1}, \boldsymbol{x}_N)$
- R($\boldsymbol{\tau}$) is the accumalated cost over a trajectory
- $p(\boldsymbol{\tau})$ represents the path probability of the trajectory, which using Bayesian and Markov properties can be expressed as:

$$p(\boldsymbol{\tau}) = \underbrace{p(\boldsymbol{x}_0)}_{init.dist.} \prod_{i=0}^{N} \underbrace{p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i, \boldsymbol{u}_i)}_{\text{state trans. dist.}} \underbrace{p(\boldsymbol{u}_i|\boldsymbol{x}_i; \boldsymbol{\theta})}_{\text{policy } \boldsymbol{\pi} \text{ dist.}}$$

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{x}_t, \boldsymbol{u}_t, t)$$

Optimization Formulation

$$\boxed{\underset{\boldsymbol{\theta}}{\text{maximize}} \; J(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\text{maximize}} \; \int p(\boldsymbol{\tau}) R(\boldsymbol{\tau}) \, d\boldsymbol{\tau}}$$

The expected return of the policy can be written as the sum of the expected rewards over all trajectories.

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau}) R(\boldsymbol{\tau}) \, d\boldsymbol{\tau}$$

$$\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau})) = \frac{1}{p(\boldsymbol{\tau})} \nabla_{\boldsymbol{\theta}} p(\boldsymbol{\tau})$$

$$\int p(\boldsymbol{\tau}) \nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau})) R(\boldsymbol{\tau}) \, d\boldsymbol{\tau}$$

$$\boxed{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{p(\boldsymbol{\tau})} \left[\nabla_{\boldsymbol{\theta}} log(p(\boldsymbol{\tau})) R(\boldsymbol{\tau}) \right]}$$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{p(\boldsymbol{\tau})}\left[\nabla_{\boldsymbol{\theta}} log\left(\underbrace{p(\boldsymbol{x}_0)}_{init.dist.} \prod_{i=0}^{N} \underbrace{p(\boldsymbol{x}_{i+1}|\boldsymbol{x}_i, \boldsymbol{u}_i)}_{\text{state trans. dist.}} \underbrace{p(\boldsymbol{u}_i|\boldsymbol{x}_i; \boldsymbol{\theta})}_{\text{policy } \boldsymbol{\pi} \text{ dist.}}\right) R(\boldsymbol{\tau})\right]$$

$$\boxed{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{p(\boldsymbol{\tau})}\left[\sum_{i=0}^{N} \nabla_{\boldsymbol{\theta}} log(\underbrace{p(\boldsymbol{u}_i|\boldsymbol{x}_i; \boldsymbol{\theta})}_{\text{policy } \boldsymbol{\pi} \text{ dist.}}) R(\boldsymbol{\tau})\right]}$$

Gradient of the expected reward only depends the expected reward over a trajectory and the log probability of the parameterized control policy. The derivatives with respect to the control system do not need to be computed to estimate the gradient.

REINFORCE Algorithm

$$\boxed{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{M}\sum_{m=0}^{M}\left[\left(\sum_t \nabla_{\boldsymbol{\theta}} log\left(p(\boldsymbol{u}_{m,t}|\boldsymbol{x}_{m,t};\boldsymbol{\theta})\right)\right)\left(\sum_t r(\boldsymbol{x}_{m,t},\boldsymbol{u}_{m,t},t)\right)\right]}$$

Algorithm 3: REINFORCE Algorithm

Result: θ

initialize: $\boldsymbol{\theta} = \boldsymbol{\theta}_0$, γ: learning rate, m: rollouts

while *unconverged* **do**

  for *m rollouts* **do**

    Sample trajectories by running the policy.

  $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = E_{\boldsymbol{\tau};\boldsymbol{\theta}}$

  $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_i)$

REINFORCE Considerations

- Guaranteed to converge to the true gradient at fastest theoritical rate.
- Policy parameter variations are no longer needed eliminating computationally expensive process and potentially endangering the policy gradient estimation.
- In real world implementations a single rollout is enough to get a good estimation of the policy gradient.

Application to Control: LQR Optimal Control

Optimal Control Problem Formulation:

$$\begin{aligned}
\dot{\boldsymbol{x}} &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \\
\boldsymbol{u} &= -\boldsymbol{K}\boldsymbol{x} \\
J(\boldsymbol{x}, \boldsymbol{u}) &= \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u}
\end{aligned}$$

Control policy can be solved analytically with simple knowledge of the system dynamics and cost function:

$$\begin{aligned}
&\boldsymbol{K} = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} \\
&\boldsymbol{P}\boldsymbol{A} + \boldsymbol{A}^T\boldsymbol{P} + \boldsymbol{Q} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} = \dot{\boldsymbol{P}}
\end{aligned}$$

LQR to Reinforcement Learning

What happens when we don't have a dynamics model?

Can we use Reinforcement Learning to learn the optimal control gain K and apply the feeback controller?

We can reformulate as a Reinforcement Learning problem:

$$\underset{\theta}{\text{maximize}} \quad \mathbb{E}\Big[R(\boldsymbol{\tau})\Big]$$

$$R(\boldsymbol{\tau}) = \sum_{t=0}^{N} r(\boldsymbol{x}_t, \boldsymbol{u}_t, t)$$

$$r(\boldsymbol{x}_t, \boldsymbol{u}_t, t) = -\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} - \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u}$$

$$\boldsymbol{u} = -\boldsymbol{\theta} \boldsymbol{x}$$