CS 295A/395D: Artificial Intelligence

Reinforcement Learning

Prof. Emma Tosch

29 April 2022

The University of Vermont

Logistics

- Last topical lecture
    - Exam next Friday
    - New assignment out later today
    - Last programming assignment out later today
- Monday: Guest lecture (Eric Atkinson) on belief programming
    - Eric has also worked on programming constructs for sequential decision making
    - Papers to be linked on the website
- Course evaluations

If you do choose to do them, please answer the following:

1. D … is class?
2. D

If yo … ppy to have not contributed to your misery.

If yo … you do me a solid and write something like:

I am **Prof. or Dr.**
Please do not refer to me as Ms. or Mrs.
(yes, I have been called Mrs. in a course eval)

While Prof. Tosch's knowledge of the area is adequate, the course was perhaps a bit too formal. I suspect both she and the students would be much happier if her future course assignments were actually in her active research areas.

<u>I also always enjoy weird commentary:</u>

Overall, Emma Tosch is a new instructor trying to find her way as a UMass graduate at the lax pothead school that is UVM.

Recall: MDPs

An MDP is defined by the model $\langle S, A, T, R, \gamma \rangle$ such that:

$S = \{s_1, \ldots, s_n\}$ is the set of *n* states

$A = \{\, a_1, \ldots, a_m\}$ is the set of *m* actions (assume wlog we can take every action in every state)

T is a representation of the transition probability into a state given an action and a current state (i.e., $P(s \mid s', a)$, possibly represented by a $(\mathrm{m} * n) \times n$ matrix of transition probabilities such that each row represents some $v_i = \langle s^1, a^i \rangle$ and

$$p_{ij} = P\big(X_t = s_j \mid X_{t-1} = v_i(0), A = v_i(1) \big)$$

$R : S \times A \times S \rightarrow \mathfrak{R}$ is the reward function, which can be defined in terms of the current state, action, next state, or even as a probabilistic map – it is whatever you need it to be

Recall: Value Functions

A *value function* $V^\pi : S \to \Re$ is a utility function *specific to a given policy* that assigns real numbers to each state in S.

This function is designed to be the expected "return":

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t E[R_t \mid S_0 = s] = \sum_{t=0}^{\infty} \gamma^t \sum_{s' \in S} \sum_{a \in A} R(s,a,s') P(s' \mid s,a) \pi(a \mid s)$$

$$= \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} \sum_{t=0}^{\infty} \gamma^t P(s' \mid s,a) R(s,a,s') = \sum_{a \in A} \pi(a \mid s) \sum_{s' \in S} P(s' \mid s,a)[R(s,a,s') + \gamma V^\pi(s')]$$

Action-Value Functions

An *action-value function* $Q^{\pi}: S \times A \rightarrow \Re$ is a utility function *specific to a given policy* that assigns real numbers to each state-action pair in S and A such that...

We get the the expected "return" for following the input action a in the first step and π o/w.

$$Q^{\pi}(s,a) = \sum_{t=0}^{\infty} \gamma^t E[R_t \mid S_0 = s, A_0 = a] = \sum_{s\prime \in S} P(s' \mid s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Recall: Computing the value function

```
Initialize V_0^π(s) ≔ 0 for all s
For t until convergence:
    For each state s_from:
        For each state s_to:
```

$$V^{\pi}_{t+1}(s_{from}) \coloneqq \sum_{s_{to}} \sum_{a} P(s_{to} \mid s_{from}, a)\pi(a \mid s_{from})[\ R(s_{from}, a, s_{to}) + \gamma\, V_t(s_{to})\ \ ]$$

Decision theory: Analogous to evaluating the expected utility *given* a set of actions.

How to find the best set of actions?

Generalized policy iteration

```
Initialize V_0^π(s) := 0 for all s
For t until convergence:
    For each state s_from:
        For each state s_to:
            Update estimate of value function
        Compute Q values for s_from
        Update policy to select action with max Q value
```

- Naïve approach:
    - Do complete sweep, compute value functions
    - Do another sweep, update policy
- Idea: update policy and value function together, using Q values

High-level

Supervised Learning

- Input: n x 2 matrix of pairs (x, y).
- Assumption: x and k are correlated.
- Output: a function f: X -> Y such that f predicts y for input x
- Examples: classification, regression

Unsupervised learning

- Input: n x 1 vector of data (x) and k parameters
- Assumption: x and k are independent given problem space
- Output: a function f: X -> Y such that f predicts y for input x
- Examples: clustering (|Y| = k), ranking

Reinforcement learning

- Input: environment (maybe MDP?), k parameters, objective (goal)
- Assumption: can collect (s, r) from the environment
- Output: a function π: S -> A that selects actions given state

High-level

Supervised Learning

- Input: n x 2 matrix of pairs (x, y).
- Assumption: x and k are correlated.
- Output: a function f: X -> Y such that f predicts y for
- Examples: classification, regression

Supervised & Unsupervised Learning are problem-agnostic

Unsupervised learning

- Input: n x 1 vector of data (x) and k parameters
- Assumption: x and k are independent given proble
- Output: a function f: X -> Y such that f predicts y for inpu
- Examples: clustering (| Y | = k), ranking

Reinforcement learning

- Input: environment (maybe MDP?), k parameters, objective oal)
- Assumption: can collect (s, r) from the environment
- Output: a function π: S -> A that selects actions given state

High-level

<u>Supervised Learning</u>

- Input: n x 2 matrix of pairs (x, y).
- Assumption: x and k are correlated.
- Output: a function f: X -> Y such that f predicts y for
- Examples: classification, regression

<u>Unsupervised learning</u>

- Input: n x 1 vector of data (x) and k parameters
- Assumption: x and k are independent given proble
- Output: a function f: X -> Y such that f predicts y for inpu
- Examples: clustering (|Y| = k), ranking

<u>Reinforcement learning</u>

- Input: environment (maybe MDP?), k parameters, objective oal)
- Assumption: can collect (s, r) from the environment
- Output: a function π: S -> A that selects actions given state

Encoding the problem varies by domain, but core techniques do not.

High-level

<u>Supervised Learning</u>

- Input: n x 2 matrix of pairs (x, y).
- Assumption: x and k are correlated.
- Output: a function f: X -> Y such that f predicts y for
- Examples: classification, regression

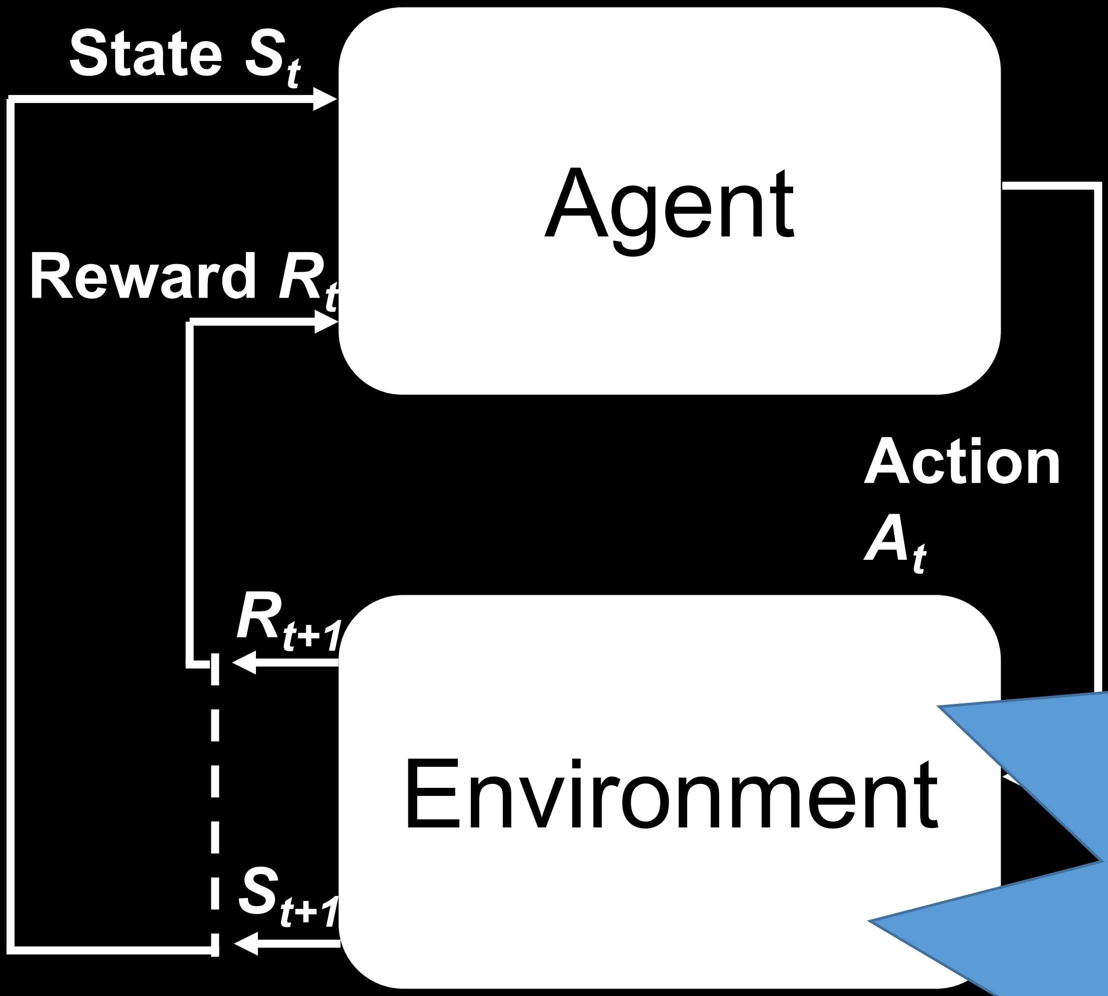
<u>Unsupervised learning</u>

- Input: n x 1 vector of data (x) and k parameters
- Assumption: x and k are independent given proble
- Output: a function f: X -> Y such that f predicts y for inpu
- Examples: clustering (|Y| = k), ranking

<u>Reinforcement learning</u>

- Input: environment (maybe MDP?), k parameters, objective oal)
- Assumption: can collect (s, r) from the environment
- Output: a function π: S -> A that selects actions given state

RL: deeply tied to the shape/properties of the environment
(bridges AI and ML)



No values functions!
(Recall: RL =/= MDPs)

Range of learning approaches

Value-Based Methods

- Closely associated with MDP formalism
- Requires explicitly representing states
- Exact computation with full observability is expensive!
- Alternative: function approximation

Function approximation: the ML side of RL

How to use fewer states?

- Some states are very similar.
- States are defined by their observable (and unobservable) features
- Select a subset of features and group states together
- Some environments: there exists a (compact) representation

Encoding domain knowledge – you are making the problem space easier!

ANNUAL REVIEWS

Annual Review of Control, Robotics, and Autonomous Systems

A Tour of Reinforcement Learning: The View from Continuous Control

Benjamin Recht

Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California 94720, USA; email: brecht@berkeley.edu

Annu. Rev. Control Robot. Auton. Syst. 2019. 2:253–79

First published as a Review in Advance on December 14, 2018

The *Annual Review of Control, Robotics, and Autonomous Systems* is online at control.annualreviews.org

https://doi.org/10.1146/annurev-control-053018-023825

Keywords

reinforcement learning, control theory, machine learning, optimization

Abstract

This article surveys reinforcement learning from the perspective of optimization and control, with a focus on continuous control applications. It reviews the general formulation, terminology, and typical experimental implementations of reinforcement learning as well as competing solution paradigms. In order to compare the relative merits of various techniques, it presents a case study of the linear quadratic regulator (LQR) with unknown dynamics, perhaps the simplest and best-studied problem in optimal control. It also describes how merging techniques from learning theory and control can provide nonasymptotic characterizations of LQR performance and shows that these characterizations tend to match experimental behavior. In turn, when revisiting more complex applications, many of the observed phenomena in LQR persist. In particular, theory and experiment demonstrate the role and importance of models and the cost of generality in reinforcement learning algorithms. The article concludes with a discussion of some of the challenges in designing learning systems that safely and reliably interact with complex and uncertain environments and how tools from reinforcement learning and control might be combined to approach these challenges.

Control Theory

"Control problem" – selecting a policy for a controller.

- More constrained than a typical RL problem
- Strong mathematical foundations
- Courses here in EBE (Hamid Ossareh, Mads Almassalkhi)

This paper – common simulation environments (e.g., MuJuCo) much easier than previously thought

A common occurrence in RL

Function approximation: the ML side of RL

How to use fewer states?

- Some states are very similar.
- States are defined by their observable (and unobservable) features
- Select a subset of features and group states together
- Some environments: there exists a (compact) representation
- Some features can be parameterized
- Exact grouping – Great!
- Inexact grouping…need to account for mismatch

Function approximation: the ML side of RL

Let $F_1, \ldots, F_n$ be *n* feature functions such that $F_i: S \times A \to \Re$

Then, instead of estimating

$$Q^\pi(s,a) = \sum_{t=0}^{\infty} \gamma^t E[R_t \mid S_0 = s, A_0 = a] = \sum_{s\prime \in S} P(s' \mid s, a)[R(s,a,s') + \gamma V^\pi(s')]$$

We estimate

$$Q_{\boldsymbol{w}}(\mathrm{s,a}) = \mathrm{w_0} + \mathrm{w_1 F_1(s,a)} + \mathrm{w_2 F_{2(s,a)}} + \cdots + w_n F_n(s,a)$$

Range of learning approaches

Value-Based Methods

- Closely associated with MDP formalism
- Requires explicitly representing states
- Exact computation with full observability is expensive!
- Alternative: function approximation

Policy Search

- Idea: optimize with respect to the policy *only*
    - *No need to estimate the value function!*
- Requires the policy be differentiable
- Does not require the environment be an MDP!
- Does not require the environment be *stationary*

Classical policy search algorithm

```
function REINFORCE
   initialize θ (arbitrary)
   for each trajectory of length T sampled from π_θ do
      for t=1 to T-1 do:
         θ ← θ + α∇_θ log π_θ(a_t | s_t) v_t
      done
   done
   return θ
end
```

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, v_t$$

Requires you be able to sample estimates of the value of being in state s at time t

Classical policy search algorithm

```
function REINFORCE
    initialize θ (arbitrary)
    for each trajectory of length T sampled from π_θ do
        for t=1 to T-1 do:
            θ ← θ + α∇_θ log π_θ(a_t | s_t) R_t
        done
    done
    return θ
end
```

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, R_t$$

You do this by computing the expected return *conditioned on states in this trajectory*

Classical policy search algorithm

```
function REINFORCE
    initialize θ (arbitrary)
    for each trajectory of length T sampled from π_θ do
        for t=1 to T-1 do:
            θ ← θ + α∇_θ log π_θ(a_t | s_t) R_t
        done
     done
     return θ
end
```

Key idea:

- Policy distribution is **known**
- Transition probabilities **unknown**

Range of learning approaches

Value-Based Methods

- Closely associated with MDP formalism
- Requires explicitly representing states
- Exact computation with full observability is expensive!
- Alternative: function approximation

Actor-Critic

- Policy search + value function estimation!
- “Actor” – policy
- “Critic” – value function
- Value function performs a kind of regularization
- Value function encodes domain knowledge about the environment

Policy Search

- Idea: optimize with respect to the policy *only*
- Requires the policy be differentiable
- Does not require the environment be an MDP!
- Does not require the environment be *stationary*

Problems

- MDP/table-based approaches have good theoretical underpinnings
    - Intractable for large state spaces
    - Non-toy problems require approximation
    - Need to encode lots of domain knowledge
- Policy gradient methods are performant, especially deep learning-based methods
    - Poorly understood
    - Lots of engineering to make sampling work