Solving Unknown MDPs

Andrew Perrault
perrault.17@osu.edu
CSE 5539

Recap: MDPs

We need a notion of environment to allow agents to learn from experience

→ the MDP, which has what components?

Designing the best MDP for a task can be challenging
- Ideally: dense, immediate rewards, but still specifying the goal not the behavior
- Find invariances in optimal policy to minimize search space

Recap: MDP structure

We made extensive use of the **value function,** i.e., the expected discounted return when following π from s:

$$V_\pi(s) = \mathbb{E}_{s_0=s,a_0,r_1,\ldots}\left[\sum_{t=0}^{\infty}\gamma^t r_{t+1}\right]$$

We will see several other examples of “value functions” today

Recap: MDP key results

To prove structural results, we combined recurrences with fixed point theory

Key facts about MDPs:

1. Any policy has a unique value function, which satisfies unrolling
2. The optimal value function is unique and satisfies the Bellman equation
3. Optimal policy needs to only put probability on “optimal actions”
4. Finding the optimal policy in a known MDP is a convex problem → efficient algorithms exist

Why do we care?

Want to solve unknown MDPs, useful to know “structure”
- E.g., tendency to collapse stochastic into deterministic policies

Value function**s** are very important across RL algorithms

Solution methods for known MDPs become planning methods for model-based RL

Meta-strategies

We like the “transformers recipe” as a way to hold intelligence

In MDPs, intelligence is contained in:

- The policy function $\pi : S \to \Delta A$
- The value functions:

1. **Value function** (or *state-value function*) $V_\pi : S \to \mathbb{R}$
2. **Q-function** (or *action-value function*) $Q_\pi : S \times A \to \mathbb{R}$. Intuition: how good is a particular state-action pair?
3. **Advantage** $A : S \times A \to \mathbb{R}$, which is $Q_\pi(s, a) - V_\pi(s)$. Intuition: how good is a state-action pair relative to the current policy for that state?

- Environment/world model, $\hat{P} : S \times A \to S$ and $R : S \times A \times S \to \Delta\mathbb{R}$

The Q-function

$$Q_\pi : S \times A \to \mathbb{R}.\ Q_\pi(s,a) = \mathbb{E}_{s_0=s, a_0=a, r_1, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}\right]$$

Q: how good is action a from state s? Really good question for control

It can also be unrolled: $Q_\pi(s,a) = \sum_{s'} P(s' \mid s,a)\left[\mathbb{E}[R_a(s,s')] + \sum_{a'} \pi(a' \mid s') Q_\pi(a' \mid s')\right]$

And it has a Bellman equation: $Q_\pi(s,a) = \sum_{s'} P(s' \mid s,a)\left[\mathbb{E}[R_a(s,s')] + \max_{a'} \pi(a' \mid s') Q_\pi(a' \mid s')\right]$

Both satisfy Brouwer's & Banach's conditions as well → uniqueness, etc.

The advantage A

$A_\pi : S \times A \to \mathbb{R}. A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$

A nicer training signal

- Q_π and V_π can have high average magnitude, advantage always has mean 0

Unique since Q and V are unique

Which is best for learning in unknown MDPs?

- The policy function $\pi : S \to \Delta A$
- The value function $V_\pi : S \to \mathbb{R}$
- The Q-function $Q_\pi : S \times A \to \mathbb{R}$
- The advantage function $A : S \times A \to \mathbb{R}$, $A = Q_\pi(s, a) - V_\pi(s)$
- Environment/world model, $\hat{P} : S \times A \to S$ and $\hat{R} : S \times A \times S \to \Delta\mathbb{R}$

Policy “reading”

Policy “reading”: convert learned function into a policy we can execute

Model-free: We can’t read a policy from V_π unless we also have $\hat{P}(s' \mid s, a)$
- Need transition function to convert state values into best action

Infinite actions:
- Use π
- Discretize

Environment model: only needed for policy reading if V_π is the only other thing we learn

Note: *can still be worth learning something even if it is not needed for policy reading*

Memory/compute considerations

The more functions we learn, the more compute and memory we consume

If expensive to interact with environment, easier to justify more intensive computation

Cold/warm-start mixing

Suppose we have access to a strong start for one function, e.g.,

- an expert policy for π
- an expert value function V_π

Combining an expert estimate with a blank estimate for another function is risky

- Can forget expert knowledge
- Or do hard computation to “translate” expert estimate

Policy smoothness

The policy implied by the **value functions** (V, Q, A) is deterministic
- There is a single optimal action for every state (unless there is a tie)

If the winning action changes ($\arg\max_a$), the policy snaps from the old action

to the new one

π can be made to change smoothly

Monte Carlo vs. Temporal Difference

Temporal difference (TD) updates can allow for faster learning
- Don’t have to wait for episode to end
- Especially useful if rewards are present on many timesteps

If we want to do TD, have to have something to bootstrap **off of** (Q or V)

Off- vs. on-policy learning

Off-policy data: data not from the current policy. Common sources:
- Data from an old policy
- Observational data

Value-based (Q, V, A) approaches can more easily leverage off-policy data than policy-based approaches

Model-based vs. model-free

If we learn a good environment model, less interactions are needed
- Good if a model is easy to learn and/or interactions are expensive

We still need a method to actually solve the model
- Can be anything: planning (MPC, MCTS) or model-free (e.g., model is just used to generate synthetic data)

RL terminology

If you learn and use $P(s' \mid s, a)$: **model-based (otherwise model-free)**
- Examples: Dyna-Q, Dreamer, AlphaZero, MuZero

Anything that learns π: **policy gradient** (policy = π)
- Examples: REINFORCE, PPO, GRPO, soft actor-critic (SAC), DDPG, TD3

If you learn π and Q or V: **actor-critic** (actor = π, critic = Q or V)
- Examples: PPO, SAC, DDPG, TD3 (REINFORCE and GRPO do not have critics)

If you learn Q only: **Q-learning**
- Examples: Deep Q-network (DQN), C51

If you do *not* learn π: **value-based** (policy is represented implicitly)
- Examples: Deep Q-network (DQN), C51

Questions?

Value-based methods for unknown MDPs

Common structure:

0. Choose what function you want to estimate (Q or V) and make sure you can read the implicit policy
1. Run the implicit policy + some exploration (because the implicit policy is deterministic) to collect trajectories
2. Update the function estimates
- Because the policy is implicit, this also updates the policy
- Some choices for update: Monte Carlo, TD unrolling, TD Bellman
3. If model-based: update the model $\hat{P}(s' \mid s, a)$ and $\hat{R}(s, a, s')$
4. If model-based: use model to do further updates (e.g., simulate trajectories and update value functions based on them)

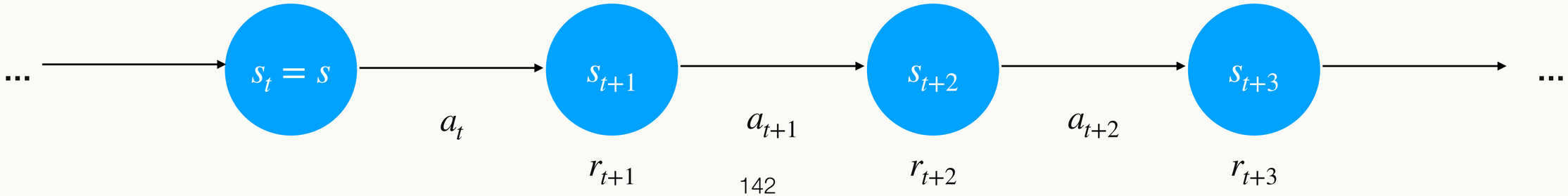
Update: MC, TD unrolling, TD Bellman

Using Q as an example, compare the "targets" (what the update "aims" for)

MC target: $Q(s_t, a_t)$ is updated to target $G(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} \ldots$

TD unrolling: $Q(s_t, a_t)$ is updated to target $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$

TD Bellman: $Q(s_t, a_t)$ is updated to target $r_{t+1} + \max_{a'} \gamma Q(s_{t+1}, a')$



Example: dynamic programming vs. Q-learning

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Online value-based estimation

So far, we have only seen *batch* value-based estimation, i.e., we re-estimate each function **only** using data from that batch

In online estimation with a neural network, each target becomes a training data point
- We rely on the neural network to naturally forget the old data as the new data comes in!

Deep Q-Network (DQN)

0. Function to estimate: Q
1. Run the implicit policy plus some random exploration
2. Update the Q function estimate with TD Bellman using MSE loss
3. Empty (no model-based steps)

“Deep” because DQNs first processed images in Atari using an AlexNet-style architecture

Value-based deep RL

ImageNet era advances in deep learning translated into rapid progress in deep RL

First DQN for Atari (2013)
- Two convolutional plus two fully connected layers
- 38 days of training experience
- Performance on par with a human (and far better than past RL)

Reportedly contributed to Google's decision to buy DeepMind in 2014

Comparison with supervised learning

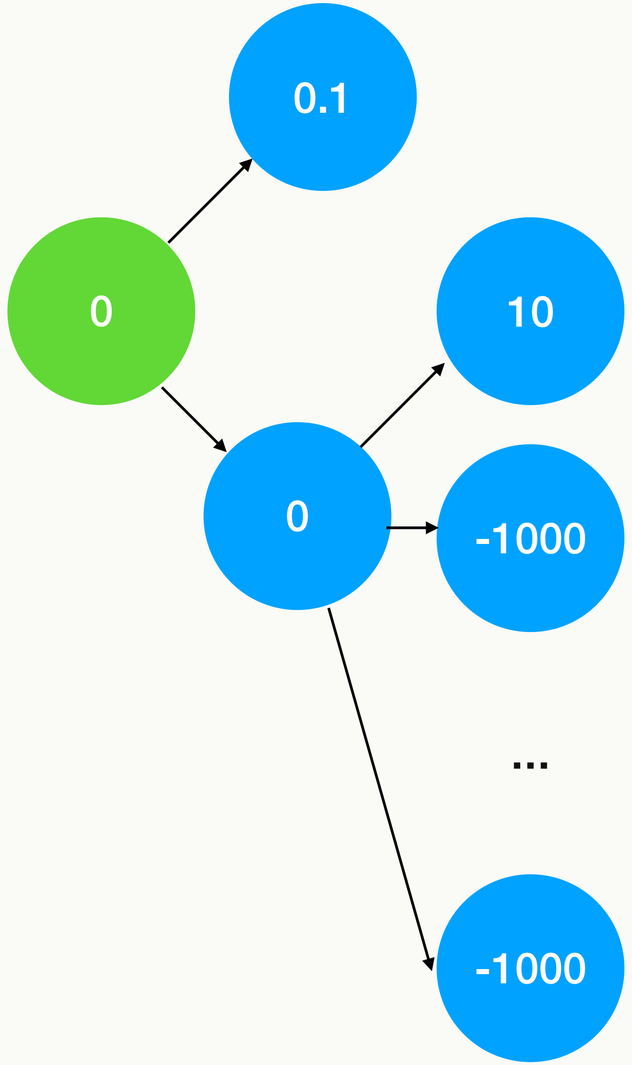
What makes training for a DQN different than say, training on ImageNet?

Exercise: exploration

1. What's the best policy in this MDP?
- Reward in each state, no discounting
- Many -1000 states in bottom right

2. How can learning go wrong?

Problem: we may prematurely conclude that {up} is the best action
- It takes us a long time to find {down, up}
- Even if do occasionally see {down, up}, it may not be enough to break our bias against {down}



More issues: scale and stationarity

Scale: the **scale of all targets will generally increase over training** as the policy gets stronger, possibly by orders of magnitude
- vs. supervised learning: label scale is fixed at start of training
- Makes it hard to set hyperparameters
- Fixes/hacks: clipping, normalization, …

Non-stationarity: the **target for the same input tends to increase over training**
- vs. supervised learning: labels do not change over training
- Want to forget previous values, but retain features/representation
- Memorization is bad (but is good in ImageNet)

More issues: estimating expectations

Hard to distinguish when the target is **genuinely changing** in the face of noise
- We want the network to “smooth out” stochasticity to learn the expectation
- vs. ImageNet, which has very low label noise
- vs. next token prediction, where labels vary but the model explicitly learns this distribution (can also be applied to RL)

Estimation biases: when we pass a random variable through a max (Bellman), the max is overestimated because it captures noise