

Foundations of machine learning

Markov decision problems and reinforcement learning

Maximilian Kasy

Department of Economics, University of Oxford

Hilary term 2027

Outline

- Markov decision problems: Goal-oriented interaction with an environment.
- Three approaches: Learn the model, the value function, or the policy.
- Dynamic programming: Transition probabilities are known.
- Learning action value functions: Sarsa and Q-learning, with deep neural nets for approximation, and eligibility traces.
- Policy gradient methods: Learning the policy directly.

Takeaways for this part of class

- Markov decision problems provide a general model of goal-oriented interaction with an environment.
- In reinforcement learning, transition probabilities are unknown.
- Value-based methods estimate action value functions, in tabular form, or using approximation (e.g., neural nets).
- They can be on-policy or off-policy.
- Policy gradient methods parametrize the policy, and perform stochastic gradient ascent on expected rewards.

Introduction

- The framework is familiar from macro and structural micro, where it is solved using dynamic programming.
- Big difference in reinforcement learning:
Transition probabilities are unknown, and need to be learned.
- Like bandit problems, with the addition of changing states.
- Early successes: Game play (Backgammon, Chess, Go, ...), using self-play.
- More recently: Post-training of large language models.
 - RL from human feedback (RLHF): Rewards from a model of human ratings.
 - RL with verifiable rewards (RLVR): Rewards from checking answers.

Markov decision problems

Expected updates - dynamic programming

Sample updates

Approximation

Eligibility traces

Policy gradient methods

Markov decision problems

- Time periods $t = 1, 2, \dots$
- States $S_t \in \mathcal{S}$ (This is the part that's new relative to bandits!)
- Actions $A_t \in \mathcal{A}(S_t)$
- Rewards R_{t+1}
- Dynamics (transition probabilities):

$$P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a, S_{t-1}, A_{t-1}, \dots) = p(s', r | s, a).$$

- The distribution depends only on the current state and action.
- It is constant over time.
- We will allow for continuous states and actions later.

Policy function, value function, action value function

- Objective: Discounted stream of rewards, $\sum_{t \geq 0} \gamma^t R_t$.
- Expected future discounted reward at time t , given the state $S_t = s$:
Value function,

$$V_t(s) = E \left[\sum_{t' \geq t} \gamma^{t'-t} R_{t'} | S_t = s \right].$$

- Expected future discounted reward at time t , given the state $S_t = s$ **and** action $A_t = a$:
Action value function,

$$Q_t(a, s) = E \left[\sum_{t' \geq t} \gamma^{t'-t} R_{t'} | S_t = s, A_t = a \right].$$

Bellman equation

- Consider a policy $\pi(a|s)$, giving the probability of choosing a in state s . This gives us all transition probabilities, and we can write expected discounted returns recursively

$$Q_{\pi}(a, s) = (\mathcal{B}_{\pi} Q_{\pi})(a, s) = \sum_{s', r} p(s', r | s, a) \left(r + \gamma \cdot \sum_{a'} \pi(a' | s') Q_{\pi}(a', s') \right).$$

- Suppose alternatively that future actions are chosen optimally. We can again write expected discounted returns recursively

$$Q_*(a, s) = (\mathcal{B}_* Q_*)(a, s) = \sum_{s', r} p(s', r | s, a) \left(r + \gamma \cdot \max_{a'} Q_*(a', s') \right).$$

Existence and uniqueness of solutions

- The operators $\mathcal{B}_\pi$ and $\mathcal{B}_*$ define contraction mappings on the space of action value functions. (As long as $\gamma < 1$.)
- By Banach's fixed point theorem, unique solutions exist.
- The difference between assuming a given policy π , or considering optimal actions $\arg\max_a Q(a, s)$, is the dividing line between **on policy** and **off policy** methods in reinforcement learning.

Three approaches to reinforcement learning

What should we learn from observed tuples $\langle s, a, r, s' \rangle$?

1. **The model:** Estimate the transition probabilities $p(s', r | s, a)$, then solve the Bellman equation by dynamic programming (plug-in).
 - Familiar from structural econometrics.
 - Infeasible when the state space is large.
2. **The value function:** Estimate $Q_\pi(a, s)$ or $Q_*(a, s)$ directly, without estimating p . Choose actions (approximately) as $\operatorname{argmax}_a Q(a, s)$.
 - Sarsa, Q-learning, $TD(\lambda)$.
3. **The policy:** Parametrize $\pi_\theta(a|s)$, and directly maximize expected rewards in θ .
 - Policy gradient methods.
 - Cf. empirical welfare maximization in treatment choice.
 - Natural for large action spaces – e.g., language models, where the policy is the model.

Actor-critic methods combine 2 and 3.

Markov decision problems

Expected updates - dynamic programming

Sample updates

Approximation

Eligibility traces

Policy gradient methods

Expected updates - dynamic programming

- Suppose we know the transition probabilities $p(s', r|s, a)$.
- Then we can in principle just solve for the action value functions and optimal policies.
- This is typically assumed in macro, IO models.
- Solutions: Dynamic programming.
Iteratively replace
 - $Q_\pi(a, s)$ by $(\mathcal{B}_\pi Q_\pi)(a, s)$, or
 - $Q_*(a, s)$ by $(\mathcal{B}_* Q_*)(a, s)$.
- Decision problems with terminal states: Can solve in one sweep of backward induction.
- Otherwise: Value function iteration until convergence – replace repeatedly.

Markov decision problems

Expected updates - dynamic programming

Sample updates

Approximation

Eligibility traces

Policy gradient methods

Sample updates

- In practically interesting settings, agents (human or AI) typically don't know the transition probabilities $p(s', r|s, a)$.
- This is where reinforcement learning comes in.
Learning from observation while acting in an environment.
- Observations come in the form of tuples

$$\langle s, a, r, s' \rangle.$$

- Based on a sequence of such tuples, we want to learn Q_π or Q_* .

Classification of one-step reinforcement learning methods

1. Known vs. unknown transition probabilities.
2. Value function vs. action value function.
3. On policy vs. off policy.
 - We will discuss Sarsa and Q-learning.
 - Both: unknown transition probabilities and action value functions.
 - First: “tabular” methods, where we keep track of all possible values (a,s) .
 - Then: “approximate” methods for richer spaces of (a,s) , e.g., deep neural nets.

One-step updates

Value
estimated

Expected updates
(DP)

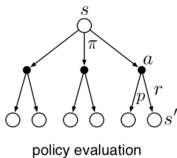
Sample updates
(one-step TD)

Value
estimated

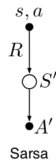
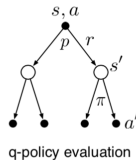
Expected updates
(DP)

Sample updates
(one-step TD)

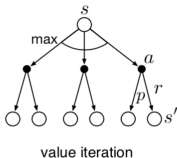
$$v_{\pi}(s)$$



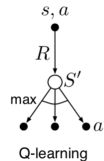
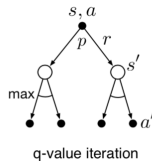
$$q_{\pi}(s, a)$$



$$v_{*}(s)$$



$$q_{*}(s, a)$$



Backup diagrams from Sutton and Barto (2018).

Sarsa

- On policy learning of action value functions.
- Recall Bellman equation

$$Q_{\pi}(a, s) = \sum_{s', r} p(s', r | s, a) \left(r + \gamma \cdot \sum_{a'} \pi(a' | s') Q_{\pi}(a', s') \right).$$

- Sarsa estimates expectations by sample averages.
- After each observation $\langle s, a, r, s', a' \rangle$, replace the estimated $Q_{\pi}(a, s)$ by

$$Q_{\pi}(a, s) + \alpha \cdot (r + \gamma \cdot Q_{\pi}(a', s') - Q_{\pi}(a, s)).$$

- α is the step size / speed of learning / rate of forgetting.

Sarsa as stochastic (semi-)gradient descent

- Think of $Q_\pi(a, s)$ as prediction for $Y = r + \gamma \cdot Q_\pi(a', s')$.

- Quadratic prediction error:

$$(Y - Q_\pi(a, s))^2.$$

- Gradient for minimization of prediction error for current observation w.r.t. $Q_\pi(a, s)$:

$$-(Y - Q_\pi(a, s)).$$

- Sarsa is thus a variant of stochastic gradient descent.
- Variant: Data are generated by actions where π is chosen as the optimal policy for the current estimate of Q_π .
- Reasonable method, but convergence guarantees are tricky.

Q-learning

- Similar to Sarsa, but **off policy**.
- Like Sarsa, estimate expectation over $p(s', r|s, a)$ by sample averages.
- Rather than the observed next action a' consider the optimal action **$\operatorname{argmax}_{a'} Q_*(a', s')$** .
- After each observation $\langle s, a, r, s' \rangle$, replace the estimated $Q_*(a, s)$ by

$$Q_*(a, s) + \alpha \cdot \left(r + \gamma \cdot \max_{a'} Q_*(a', s') - Q_*(a, s) \right).$$

Markov decision problems

Expected updates - dynamic programming

Sample updates

Approximation

Eligibility traces

Policy gradient methods

Approximation

- So far, we have implicitly assumed that there is a small, finite number of states s and actions a , so that we can store $Q(a, s)$ in tabular form.
- In practically interesting cases, this is not feasible.
- Instead assume parametric functional form for $Q(a, s; \theta)$.
- In particular: Deep neural nets!
- Assume differentiability with gradient $\nabla_{\theta} Q(a, s; \theta)$.

Stochastic gradient descent

- Denote our prediction target for an observation $\langle s, a, r, s', a' \rangle$ by

$$Y = r + \gamma \cdot Q_{\pi}(a', s'; \theta).$$

- As before, for the on-policy case, we have the quadratic prediction error

$$(Y - Q_{\pi}(a, s; \theta))^2.$$

- Semi-gradient: Only take derivative for the $Q_{\pi}(a, s; \theta)$ part, but not for the prediction target Y :

$$-(Y - Q_{\pi}(a, s; \theta)) \cdot \nabla_{\theta} Q(a, s; \theta).$$

- Stochastic gradient descent updating step: Replace θ by

$$\theta + \alpha \cdot (Y - Q_{\pi}(a, s; \theta)) \cdot \nabla_{\theta} Q(a, s; \theta).$$

Off policy variant

- As before, can replace a' by the estimated optimal action.
- Change the prediction target to

$$Y = r + \gamma \cdot \max_{a'} Q_*(a', s'; \theta).$$

- Updating step as before, replacing θ by

$$\theta + \alpha \cdot (Y - Q_*(a, s; \theta)) \cdot \nabla_{\theta} Q_*(a, s; \theta).$$

Markov decision problems

Expected updates - dynamic programming

Sample updates

Approximation

Eligibility traces

Policy gradient methods

Multi-step updates

- All methods discussed thus far are one-step methods.
- After observing $\langle s, a, r, s', a' \rangle$, only $Q(a, s)$ is targeted for an update.
- But we could pass that new information further back in time, since

$$Q(a, s) = E \left[\sum_{t'=t}^{t+k} \gamma^{t'-t} R_{t'} + \gamma^{k+1} Q(A_{t+k+1}, S_{t+k+1}) | A_t = a, S_t = s \right].$$

- One possibility: at time $t+k+1$, update θ using the prediction target

$$Y_t^k = \sum_{t'=t}^{t+k-1} \gamma^{t'-t} R_{t'} + \gamma^k Q_\pi(A_{t+k}, S_{t+k}).$$

- k -step Sarsa: At time $t+k$, replace θ by

$$\theta + \alpha \cdot \left(Y_t^k - Q_\pi(A_t, S_t; \theta) \right) \cdot \nabla_\theta Q_\pi(A_t, S_t; \theta).$$

$TD(\lambda)$ algorithm

- Multi-step updates can result in faster learning.
- We can also weight the prediction targets for different numbers of steps, e.g. using weights λ^k :

$$Y_t^k = \sum_{t'=t}^{t+k} \gamma^{t'-t} R_{t'} + \gamma^{k+1} Q_{\pi}(A_{t+k+1}, S_{t+k+1}),$$
$$Y_t^{\lambda} = (1 - \lambda) \sum_{k=0}^{\infty} \lambda^k \cdot Y_t^k.$$

- But don't we have to wait forever before we can make an update based on Y_t^{λ} ?
- Note quite, since we can do the updating piece-wise!
- This idea leads to the so-called $TD(\lambda)$ algorithm.

Eligibility traces

- For $TD(\lambda)$, we proceed as for one-step Sarsa, using the prediction target

$$Y_t = R_t + \gamma \cdot Q_\pi(A_{t+1}, S_{t+1}; \theta).$$

- But we replace the gradient $\nabla_\theta Q_\pi(A_t, S_t; \theta)$ by a weighted average of past gradients, the so-called eligibility trace: Let $Z_0 = 0$ and

$$Z_t = \gamma\lambda \cdot Z_{t-1} + \nabla_\theta Q_\pi(A_t, S_t; \theta).$$

- Updating step: At time t replace θ by

$$\theta + \alpha \cdot (Y_t - Q_\pi(A_t, S_t; \theta)) \cdot Z_t.$$

- This exactly implements the updating by Y_t^λ in the long run.
- This is one of the most popular and practically successful reinforcement learning algorithms.

Markov decision problems

Expected updates - dynamic programming

Sample updates

Approximation

Eligibility traces

Policy gradient methods

Parametrizing the policy

- Approach 3: Rather than learning value functions, learn the policy directly.
- Parametrize the policy as $\pi_{\theta}(a|s)$, differentiable in θ .
- Leading example: Softmax (multinomial logit),

$$\pi_{\theta}(a|s) = \frac{\exp(f_{\theta}(a, s))}{\sum_{a'} \exp(f_{\theta}(a', s))},$$

where f_{θ} is linear in features, or a deep neural net.

- The policy is randomized,
since deterministic policies are not differentiable in θ .
- Example: A language model is a policy of this form.
State s : The text so far. Action a : The next token.

Objective and distribution of trajectories

- Objective: Expected discounted rewards, for initial states $S_0 \sim \mu$,

$$J(\theta) = E_{\theta} \left[\sum_{t \geq 0} \gamma^t R_t \right].$$

- Trajectories: $\tau = (S_0, A_0, R_0, S_1, A_1, R_1, \dots)$, with distribution

$$P_{\theta}(\tau) = \mu(S_0) \cdot \prod_{t \geq 0} \pi_{\theta}(A_t | S_t) \cdot p(S_{t+1}, R_t | S_t, A_t).$$

- Only the policy terms depend on θ .
- Goal: Gradient ascent,

$$\theta \rightarrow \theta + \alpha \cdot \nabla_{\theta} J(\theta).$$

- But how do we get $\nabla_{\theta} J(\theta)$, without knowing p ?

The policy gradient theorem

Theorem

Let $G_t = \sum_{t' \geq t} \gamma^{t'-t} R_{t'}$ denote discounted rewards from t onward. Then

$$\begin{aligned}\nabla_{\theta} J(\theta) &= E_{\theta} \left[G_0 \cdot \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right] \\ &= E_{\theta} \left[\sum_{t \geq 0} \gamma^t \cdot Q_{\pi_{\theta}}(A_t, S_t) \cdot \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right].\end{aligned}$$

- The gradient is an expectation over trajectories generated by π_{θ} .
- It involves only the policy, not the transition probabilities p .

⇒ We can estimate it from simulated or observed trajectories.

Proof sketch

- First expression – the “likelihood ratio trick”:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{\tau} G_0(\tau) P_{\theta}(\tau) = \sum_{\tau} G_0(\tau) P_{\theta}(\tau) \nabla_{\theta} \log P_{\theta}(\tau),$$

and $\nabla_{\theta} \log P_{\theta}(\tau) = \sum_t \nabla_{\theta} \log \pi_{\theta}(A_t | S_t)$.

Cf. the score in maximum likelihood.

- Second expression: The score has mean zero given the state,

$$E_{\theta}[\nabla_{\theta} \log \pi_{\theta}(A_t | S_t) | S_t] = \sum_a \nabla_{\theta} \pi_{\theta}(a | S_t) = 0.$$

- Therefore rewards received before t drop out of the term for period t .
- Rewards from t onward have conditional expectation $E_{\theta}[G_t | S_t, A_t] = Q_{\pi_{\theta}}(A_t, S_t)$. $\square$

Monte Carlo estimation: REINFORCE

- Simulate a trajectory under π_θ , and replace θ by

$$\theta + \alpha \cdot \sum_{t \geq 0} \gamma^t \cdot G_t \cdot \nabla_\theta \log \pi_\theta(A_t | S_t).$$

- By the policy gradient theorem, the update is an unbiased estimate of $\alpha \cdot \nabla_\theta J(\theta)$.
- This is stochastic gradient ascent.
- Transition probabilities need not be known.
- Intuition: Increase the probability of actions that were followed by high rewards.
- Problem: G_t is very noisy, so the gradient estimates have high variance.

Practice problem (Policy gradient for softmax policies)

Consider the softmax policy with $f_{\theta}(a,s) = \theta \cdot \phi(a,s)$, for features ϕ .

1. Show that

$$\nabla_{\theta} \log \pi_{\theta}(a|s) = \phi(a,s) - \sum_{a'} \pi_{\theta}(a'|s) \cdot \phi(a',s).$$

2. Verify directly that $E_{\theta}[\nabla_{\theta} \log \pi_{\theta}(A_t|S_t)|S_t] = 0$.
3. Consider a bandit problem: a single state, and one period.
Write down the REINFORCE update, for $\phi(a,s) = e_a$ (one parameter per action).
Which parameters go up, and which go down, after action a yields reward R ?

Baselines and advantages

- For any function $b(\cdot)$ of the state,
 $E_\theta[b(S_t) \cdot \nabla_\theta \log \pi_\theta(A_t|S_t)] = 0$.
- We can thus replace G_t by $G_t - b(S_t)$ in REINFORCE, without introducing bias.
- A good choice is $b(s) \approx V_{\pi_\theta}(s)$.
Then $G_t - b(S_t)$ estimates the advantage

$$Q_{\pi_\theta}(A_t, S_t) - V_{\pi_\theta}(S_t).$$

- Increase the probability of actions that do better than expected.
- Actor-critic methods: Estimate V_{π_θ} using the methods discussed before.
- Language models (RLHF, RLVR) are trained using variants of this approach.
E.g., GRPO uses the average reward of several answers to the same prompt as baseline.

References

- *Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.*
- *François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., and Pineau, J. (2018). An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning, 11(3-4):219–354.*
- *Agarwal, A., Brantley, K., Kakade, S. M., Jiang, N., and Sun, W. (2026). Reinforcement learning: Theory and algorithms. <https://rltheorybook.github.io/>. Working draft, chapter 9.*