Advanced Econometrics 2, Hilary term 2020
Reinforcement learning

Maximilian Kasy

Department of Economics, Oxford University

Agenda

- Markov decision problems: Goal oriented interactions with an environment.
- Expected updates – dynamic programming.
  Familiar from economics. Requires complete of knowledge transition probabilities.
- Sample updates: Transition probabilities are unknown.
  - On policy: Sarsa.
  - Off policy: Q-learning.
- Approximation: When state and action spaces are complex.
  - On policy: Semi-gradient Sarsa.
  - Off policy: Semi-gradient Q-learning.
  - Deep reinforcement learning.
  - Eligibility traces and $TD(\lambda)$.

Takeaways for this part of class

- Markov decision problems provide a general model of goal-oriented interaction with an environment.
- Reinforcement learning considers Markov decision problems where transition probabilities are unknown.
- A leading approach is based on estimating action-value functions.
- If state and action spaces are small, this can be done in tabular form, otherwise approximation (e.g., using neural nets) is required.
- We will distinguish between on-policy and off-policy learning.

Introduction

- Many interesting problems can be modeled as Markov decision problems.
- Biggest successes in game play (Backgammon, Chess, Go, Atari games,...), where lots of data can be generated by self-play.
- Basic framework is familiar from macro / structural micro, where it is solved using dynamic programming / value function iteration.
- Big difference in reinforcement learning:
  Transition probabilities are not known, and need to be learned from data.
- This makes the setting similar to bandit problems, with the addition of changing states.
- We will discuss several approaches based on estimating action-value functions.

Markov decision problems

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

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

  - 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

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

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

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

Existence and uniequeness of solutions

- The operators $\mathscr{B}_\pi$ and $\mathscr{B}_*$ 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 $\operatorname{argmax}_a Q(a, s)$, is the dividing line between on policy and off policy methods in reinforcement learning.

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 $(\mathscr{B}_\pi Q_\pi)(a, s)$, or
  - $Q_*(a, s)$ by $(\mathscr{B}_* Q_*)(a, s)$.
- Decision problems with terminal states: Can solve in one sweep of backward induction.
- Otherwise: Value function iteration until convergence – replace repeatedly.

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

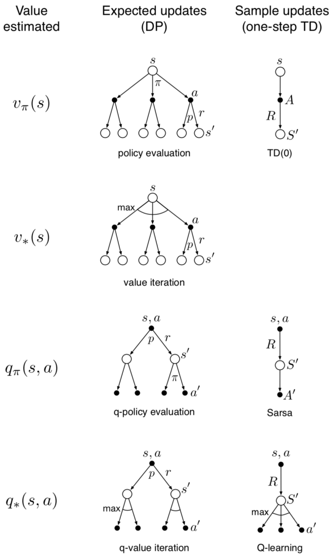
$$\langle s, a, r, s' \rangle.$$

- 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 off all possible values (a, s).
- Then: “approximate” methods for richer spaces of (a, s), e.g., deep neural nets.



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

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

- 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

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

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

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

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:

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

$$Y_t^\lambda = (1-\lambda) \sum_{k=1}^{\infty} \lambda^k \cdot Y_t^k.$$

- 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.

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.*