n-step Bootstrapping

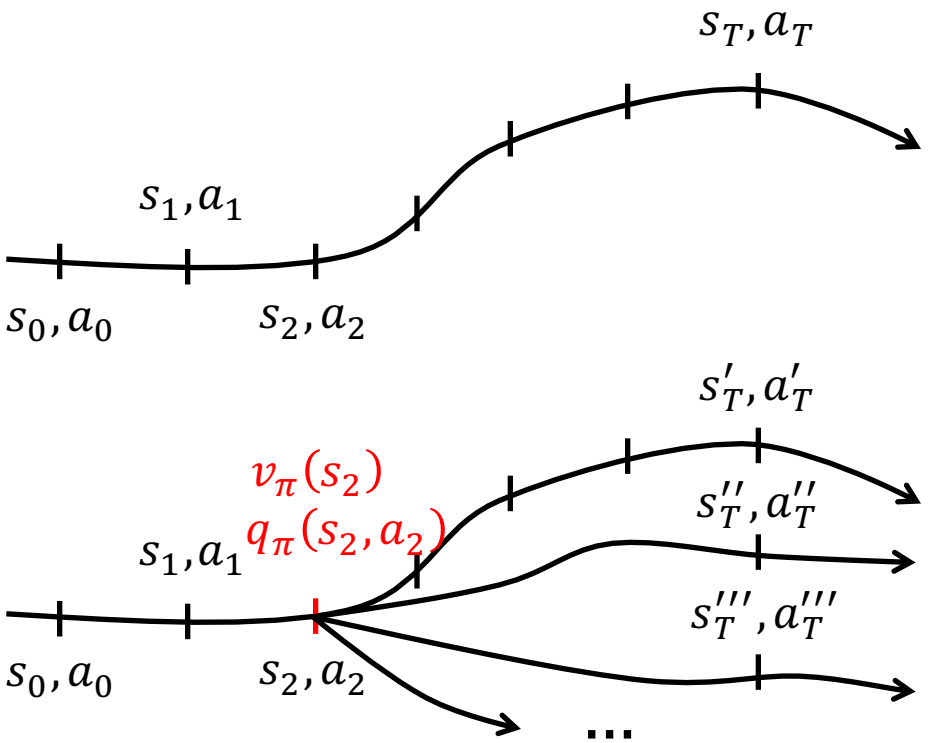
Haonan Yu

SVAIL RL reading group

05/10/2018

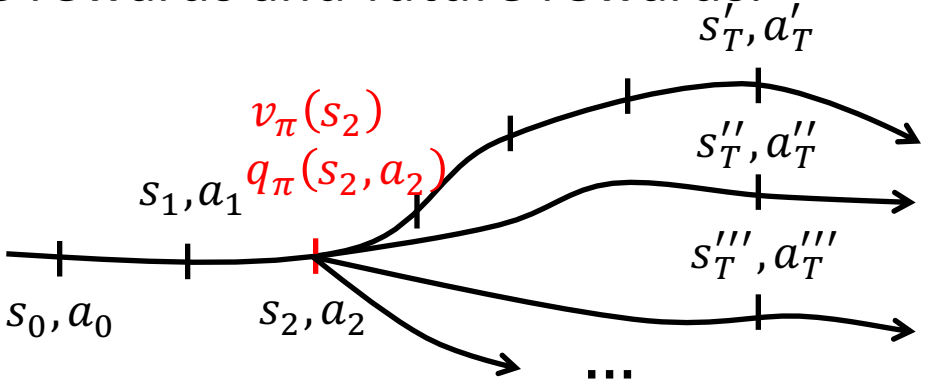
Background – MDP

- Markov Decision Process (MDP)
  - Environment state: s_t, some representation summarizing the environment
  - Agent action: a_t, taken based on the current environment state
  - Reward: r_t, a numerical reward of taking a_t at s_t (assume deterministic)
  - State transition: $P(s_{t+1}|s_t, a_t)$
  - Policy: $\pi(a_t|s_t)$
- A trajectory assuming an infinite horizon
  - $s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_T, a_T, r_{T+1}, \ldots$
- Value function (given π)
  - $v_\pi(s_t) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t)$
  - $q_\pi(s_t, a_t) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t, a_t)$
  - π is stochastic



Background – Why value function

- Value function (given π)
  - $v_\pi(s_t) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t)$
  - $q_\pi(s_t, a_t) = \mathbf{E}_\pi(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t, a_t)$
- Because the policy π is stochastic, we care about how an action will affect the future in an **expected** way, not in the best way nor in the worst way. This is always true in the RL objective.
- A value function tells us how well (in expectation) the current state (and action) will result in a future given the current policy π.
- A value is not greedy: it focuses on both immediate rewards and future rewards.
- This is called **prediction**.



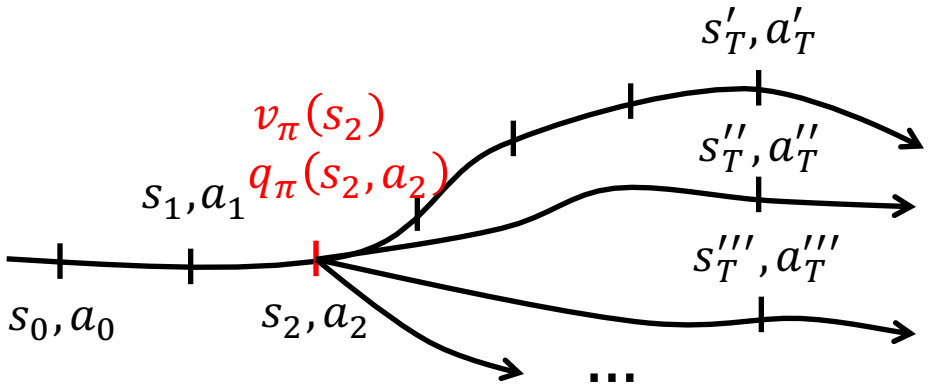
Background – Improve the policy

- Usually we start with a random policy which is not good enough
- We take many trajectories under the current policy and compute the values
- During this exploration, we found new information from the values
- We update the current policy using the new information by biasing the policy towards good actions.
  - Policy gradient: update the action probability weighted by a value
  - Q-learning: update the probability by taking ε-greedy of $q_\pi(s_t, a_t)$
- This is called **control**.

- So the real problem is how to compute the values.

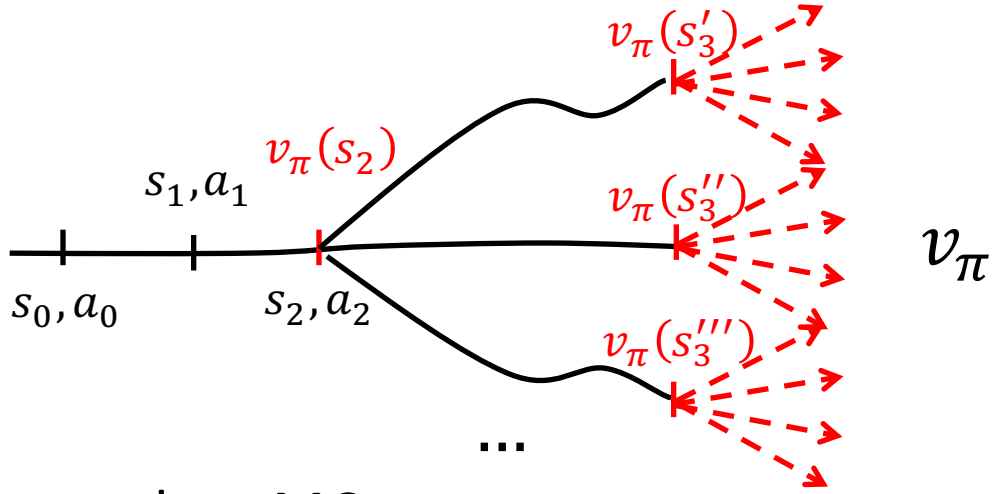
Background – Monte Carlo (MC)

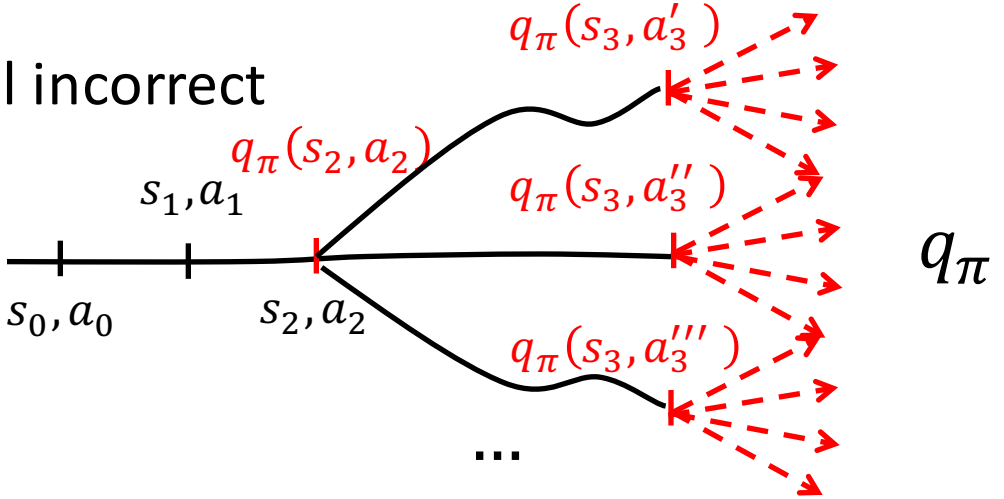
- To estimate the value $v_\pi(s_t)$, perform $K \geq 1$ rollouts at s_t till the episode ends. Then we compute the (discounted) accumulated reward for each rollout and take the average over the rollouts as the value.
- Advantage: Simple
- Disadvantage: High variance (small K), time consuming (large K)



Background – Temporal difference (TD)

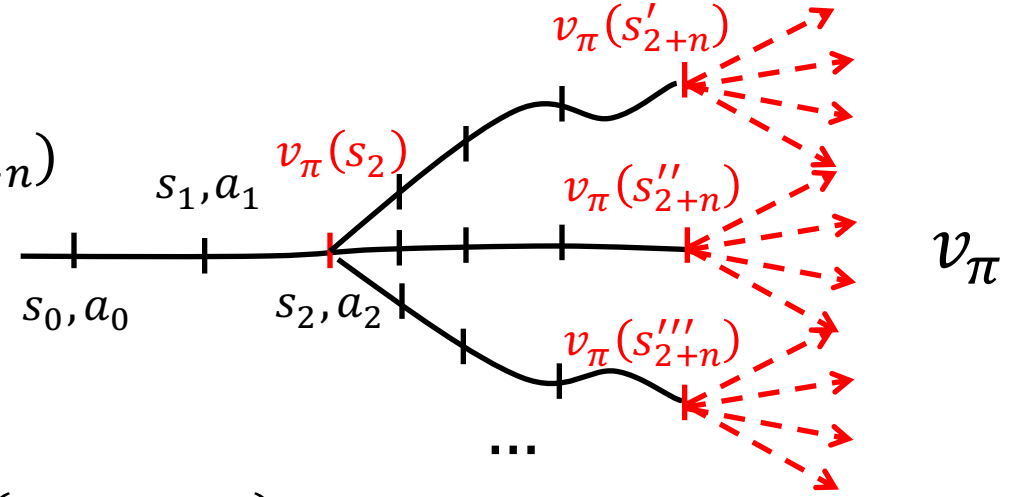
- Idea: bootstrapping
  - $v_\pi(s_t) \leftarrow r_{t+1} + \gamma v_\pi(s_{t+1})$
  - $q_\pi(s_t, a_t) \leftarrow r_{t+1} + \gamma q_\pi(s_{t+1}, a_{t+1})$
- Similar to Dynamic Programming
- We have to roll out only one step
- Use the current estimate value of the next state to replace MC
- Advantage: Low variance, time efficient
- Disadvantage: In the beginning, the values are all incorrect



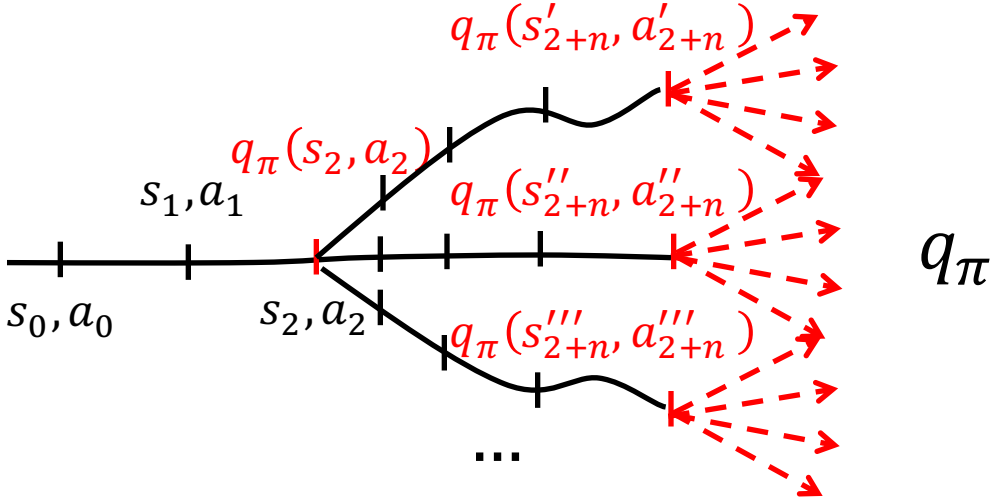


n-step TD

- Combine the advantages of MC and 1-step TD
- $v_\pi(s_t) \leftarrow r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n v_\pi(s_{t+n})$



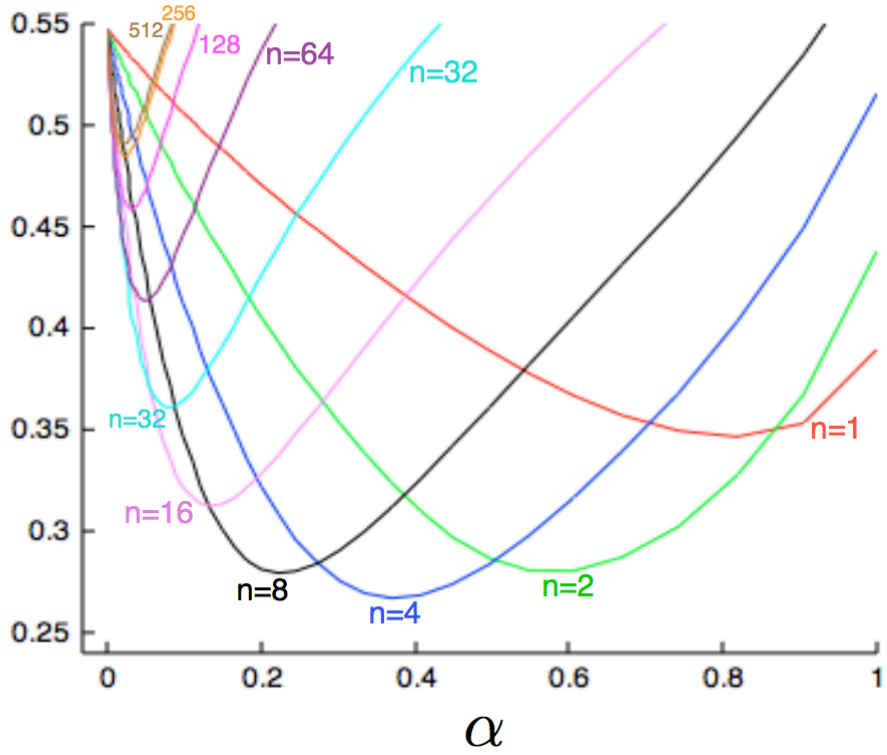
- $q_\pi(s_t, a_t) \leftarrow r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n q_\pi(s_{t+n}, a_{t+n})$



n-step TD

- Potentially, n-step TD will converge faster than 1-step TD because it bootstraps with more real information
- However, as n increases, n-step TD tends to become MC, and the variance increases, which might slow down the convergence
- A suitable value for n is crucial

Average RMS error over 19 states and first 10 episodes



n-step TD - Implementation

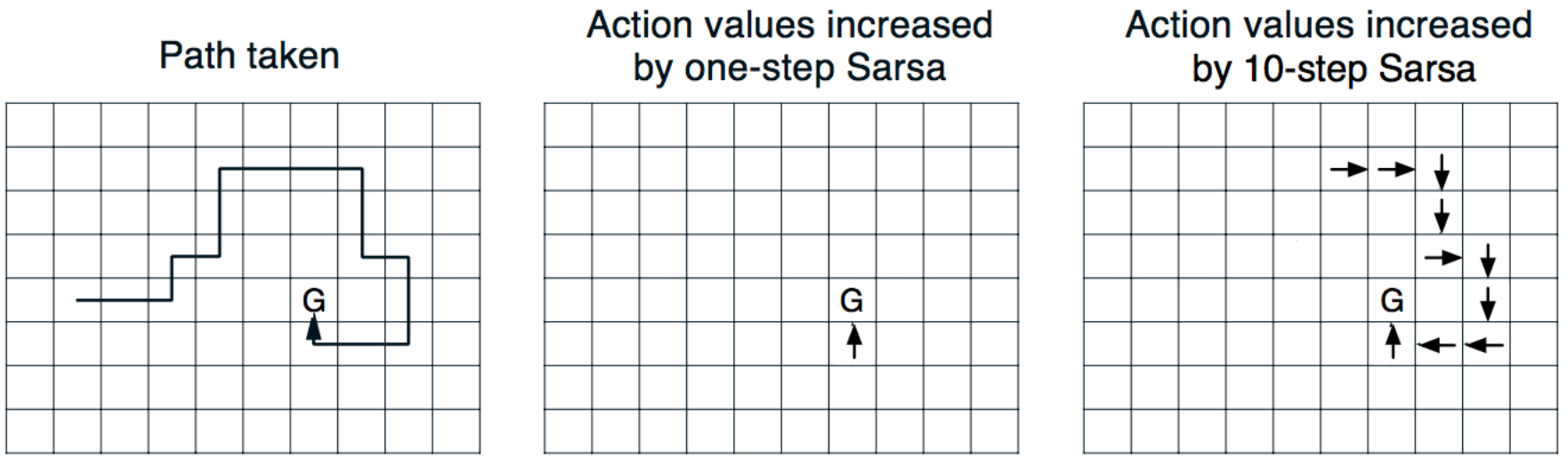
- One issue: after taking a_t at s_t, we really don't have the information $r_{t+2}, r_{t+3}, \dots, r_{t+n}, s_{t+n}$ for $n > 1$, because the future hasn't arrived yet. We only know r_{t+1} and s_{t+1}.

- So in fact we need to store the states, actions, and rewards (in a time window of size n) so that we can go back and update $v_\pi(s_{t-n})$ at time t.

- **Note** that we always use a future value to bootstrap a current value. So if you use the MSE cost and a parameterized value network, make sure to disable the gradient for the future value. Or you can explicitly set the gradient for the cost, e.g.:
  - $\frac{\partial MSE}{\partial \theta} = [v_\pi(s_t;\theta) - (r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n v_\pi(s_{t+n};\theta))] \frac{\partial v_\pi(s_t;\theta)}{\partial \theta}$

n-step for control - Sarsa

- n-step Sarsa is a simple application of n-step TD value prediction.
- Instead of assuming a given fixed policy π, we want to learn it.
- Initialize $q_\pi(s, a)$ randomly for all s and a. Set the policy always to ε-greedy of q_π.
- Whenever we update the values, we also update the policy π.



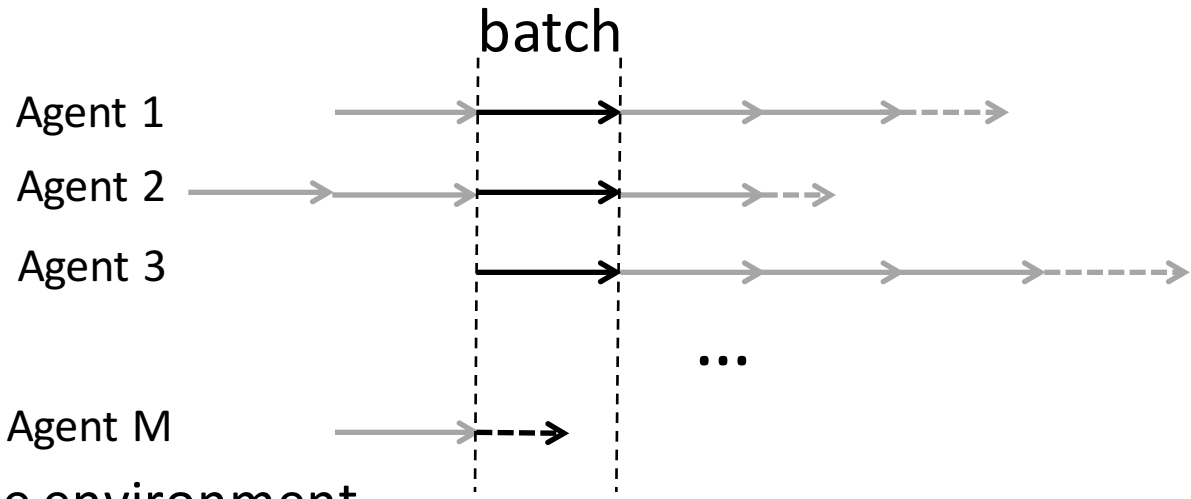
n-step for control – n-step actor critic

- Consider the policy gradient with advantage
  - $[v_\pi(s_t;\theta) - (r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n v_\pi(s_{t+n};\theta))](\underbrace{\frac{\partial v_\pi(s_t;\theta)}{\partial \theta}}_{\text{Value regression}} + \underbrace{\frac{\partial \log \pi(a_t|s_t;\theta)}{\partial \theta}}_{\text{Policy gradient}})$

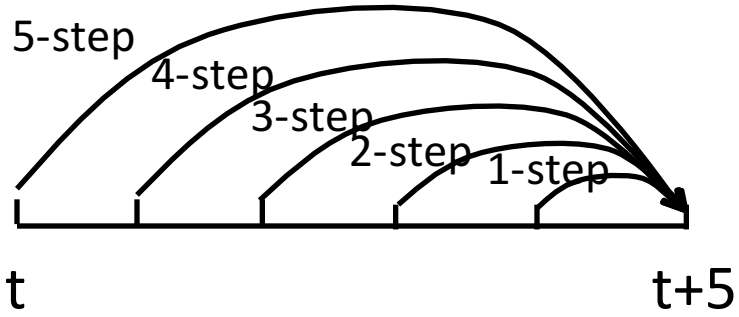
- If we disable the second term, then we only do prediction.
- Generally, we can think of the above formula doing an update of values and policies simultaneously (an extreme version of iterative update?).
- This n-step actor critic has been adopted by most works in deep reinforcement learning applications.

- A2C, A3C: two infrastructure designs on top of this, with multiple agents running in parallel to further decrease data correlation and gradient variance.

n-step for control – A2C



- A2C: synchronous **a**dvantage **a**ctor **c**ritic
- M agents running in parallel
  - every agent resides in a separate copy of the environment
  - every agent takes n steps and gets blocked
  - after all agents get blocked, a batch is collected
  - the parameters of a shared network is updated with this batch
  - the agents resume taking steps
- Question: how to compute n-step TD (easily)?
  - For each agent, we might need to maintain a length-n window so that we can compute n-step TD backward
  - In practice, an approximate solution is adopted
  - We simply compute n-step TD forward with n varying



Further reading

- Chapters (Sutton and Barto, second edition):
  - Off-policy n-step TD by importance sampling (7.3)
  - n-step tree backup algorithm (7.5)
  - A unifying algorithm: n-step $Q(\sigma)$ (7.6)

- Papers:
  - *A3C:* "Asynchronous Methods for Deep Reinforcement Learning", Mnih et al., ICML 2016.
  - *GA3C: "*Reinforcement Learning thorugh Asynchronous Advantage Actor-Critic on a GPU", Babaeizadeh et al., ICLR 2017.
  - *A2C: "*Learning to reinforcement learn", Wang et al., arXiv 2017

Questions?