Exploration

Hao Su

(slides prepared in tandem with Quan Vuong, Tongzhou Mu and Zhiao Huang)

Spring, 2021

Some contents are based on ***Bandit Algorithms*** *from Dr. Tor Lattimore and Prof. Csaba Szepesvári, and* ***COMPM050/COMPGI13*** *taught at UCL by Prof. David Silver.*

Agenda

- Intrinsic Rewards (continued)
- Structural Environment Modeling
- Debugging Tips

click to jump to the section.

Intrinsic Rewards (continued)

Novelty-driven Exploration

- Takeaway: novelty bonus works well when increasing novelty aligns with making progress in the task
  - Novelty in a game setting means going to the next level in the game
  - When will this break down?

As an aside: Distribution Distance Measure

- Given two distributions Q and P over the same space.
- The cross entropy is:

$$H(Q, P) = -\mathrm{E}_{x \sim Q}[\log P(x)]$$

- The Kullback–Leibler divergence (relative entropy) is:

$$D_{\mathrm{KL}}(Q \| P) = -\sum_{x} Q(x) \log\left(\frac{P(x)}{Q(x)}\right) = -\mathrm{E}_{x \sim Q}\left[\log\left(\frac{P(x)}{Q(x)}\right)\right]$$

- Their relationship:

$$H(Q, P) = H(Q) + D_{\mathrm{KL}}(Q \| P)$$

- Widely used in ML.

As an aside: Evidence Lower Bound

- In statistical inference, we have 2 sets of variables:
  - Unobserved variables $\mathbf{Z}$
  - Observed variables $\mathbf{X}$ that is a function of $\mathbf{Z}$
- The inference problem is to estimate the value of $\mathbf{Z}$ given $\mathbf{X}$.
- The Bayesian inference problem is to estimate the so-called posterior $P(\mathbf{Z} \mid \mathbf{X})$ given a prior $P(\mathbf{Z})$ and the observed data $\mathbf{X}$.
- Computing the posterior $P(\mathbf{Z} \mid \mathbf{X})$ exactly is often intractable.
- Variational inference offers a tractable approximation to the posterior $P(\mathbf{Z} \mid \mathbf{X})$.

As an aside: Evidence Lower Bound

- In variational inference, we learn a distribution Q over $\mathbf{Z}$ to approximate $P(\mathbf{Z} \mid \mathbf{X})$.
- To learn Q, we maximize the objective function:

$$L(\mathbf{X}) = H(Q) - H(Q; P(\mathbf{X}, \mathbf{Z}))$$

- where:
  - $H(Q)$ is the entropy of Q
  - $H(Q; P(\mathbf{X}, \mathbf{Z}))$ is the cross entropy
- The objective is called the variational lower bound or evidence lower bound (abbreviated as ELBO).
- Very influential idea in ML.

As an aside: Justifying the Evidence Lower Bound

- It can be shown that:

$$\log P(\mathbf{X}) - D_{\mathrm{KL}}(Q \| P) = L(\mathbf{X})$$

- Thus, maximizing the evidence lower bound $L(\mathbf{X})$ is equivalent to minimizing the KL from Q to the true posterior P.
- Some good properties:
  - $L(\mathbf{X})$ and its gradients wrt parameters of Q can be obtained for specific choice of Q, such as a diagonal Gaussian.
  - $L(\mathbf{X})$ is maximized when $D_{\mathrm{KL}}(Q \| P)$ is 0
- Evidence Lower Bound is thus both well-motivated and easily optimizable.

Curiosity-driven Exploration through Forward Prediction

- The RL agent can model the MDP transition function with a model $p\left(s_{t+1} \mid s_t, a_t; \theta\right)$ where θ belongs to a particular parameter space, i.e. $\theta \in \Theta$.
- Curiosity-driven exploration can be seen as taking action that allows the agent to learn more about the MDP transition function.
- Let $\xi_t = \{s_1, a_1, \ldots, s_t\}$ be the history of interaction until timestep t.
- Actions that maximize the reduction in uncertainty about the transition function can be viewed as maximizing the reduction in entropy over the transition function parameter space:

$$H\left(\Theta \mid \xi_t\right) - H\left(\Theta \mid S_{t+1}, \xi_t, a_t\right)$$

- We now derive an estimate of this reduction.

VIME: Variational Information Maximizing Exploration. Houthooft et al.

Curiosity-driven Exploration through Forward Prediction

- It can be shown that:

$$H\left(\Theta \mid \xi_t\right) - H\left(\Theta \mid S_{t+1}, \xi_t, a_t\right) = \mathbb{E}_{s_{t+1} \sim \mathcal{P}\left(\cdot \mid \xi_t, a_t\right)}\left[D_{\mathrm{KL}}\left[p\left(\theta \mid \xi_t, a_t, s_{t+1}\right) \| p\left(\theta \mid \xi_t\right)\right]\right]$$

  - where $\mathcal{P}\left(\cdot \mid \xi_t, a_t\right)$ is the true transition function of the MDP.
- We can therefore estimate the reduction in entropy with a single sample of s_{t+1}:

$$H\left(\Theta \mid \xi_t\right) - H\left(\Theta \mid S_{t+1}, \xi_t, a_t\right) \approx D_{\mathrm{KL}}\left[p\left(\theta \mid \xi_t, a_t, s_{t+1}\right) \| p\left(\theta \mid \xi_t\right)\right]$$

- We can thus promote curiosity-driven exploration by modifying the reward as follows:

$$r'\left(s_t, a_t, s_{t+1}\right) = r\left(s_t, a_t\right) + \eta D_{\mathrm{KL}}\left[p\left(\theta \mid \xi_t, a_t, s_{t+1}\right) \| p\left(\theta \mid \xi_t\right)\right]$$

  - where η is a hyper-parameter.

Curiosity-driven Exploration through Forward Prediction

$$r'\left(s_t, a_t, s_{t+1}\right) = r\left(s_t, a_t\right) + \eta D_{\mathrm{KL}}\left[p\left(\theta \mid \xi_t, a_t, s_{t+1}\right) \| p\left(\theta \mid \xi_t\right)\right]$$

- But $D_{\mathrm{KL}}\left[p\left(\theta \mid \xi_t, a_t, s_{t+1}\right) \| p\left(\theta \mid \xi_t\right)\right]$ is intractable still:
  - Both $p\left(\theta \mid \xi_t, a_t, s_{t+1}\right)$ and $p\left(\theta \mid \xi_t\right)$ are posterior given different datasets
  - Such posterior is often intractable as we have discussed
- In the language of statistical inference:
  - θ plays the role of the unobserved variables $\mathbf{Z}$
  - ξ_t or $\{\xi_t, a_t, s_{t+1}\}$ plays the role of the observed variables $\mathbf{X}$

$\Longrightarrow$ Use variational inference to learn an approximation to the posteriors.

Curiosity-driven Exploration through Forward Prediction

- We abstract the observed data as $\mathcal{D}$.
- To approximate the posterior $p(\theta \mid \mathcal{D})$, we learn a function $q(\theta; \phi) \approx p(\theta \mid \mathcal{D})$.
- $q(\theta; \phi)$ plays the role of Q and has parameters ϕ.
- To learn ϕ, we maximize the objective with gradient ascent:

$$L[q(\theta; \phi), \mathcal{D}] = \mathbb{E}_{\theta \sim q(\cdot; \phi)}[\log p(\mathcal{D} \mid \theta)] - D_{\mathrm{KL}}[q(\theta; \phi) \| p(\theta)]$$

  - Interpretation?
  - Do not worry about the alphabet soup.
  - It is the variational lower bound you have seen before.
- $p(\theta)$ is a prior over the parameter space of the learned transition function.
  - Often chosen to be uniform or unit diagonal Gaussian

Curiosity-driven Exploration through Forward Prediction

- The intractable curiosity-driven reward:

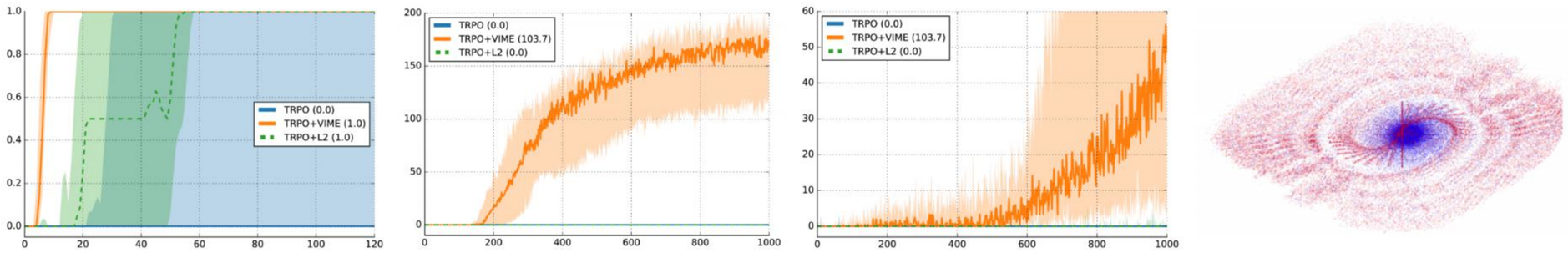
$$r'\left(s_t, a_t, s_{t+1}\right) = r\left(s_t, a_t\right) + \eta D_{\mathrm{KL}}\left[p\left(\theta \mid \xi_t, a_t, s_{t+1}\right) \| p\left(\theta \mid \xi_t\right)\right]$$

- can now be approximated with:

$$r'\left(s_t, a_t, s_{t+1}\right) = r\left(s_t, a_t\right) + \eta D_{\mathrm{KL}}\left[q\left(\theta; \phi_{t+1}\right) \| q\left(\theta; \phi_t\right)\right]$$

- where:
  - $q\left(\theta; \phi_t\right) \approx p\left(\theta \mid \xi_t\right)$
  - $q\left(\theta; \phi_{t+1}\right) \approx p\left(\theta \mid \xi_t, a_t, s_{t+1}\right)$
- We can plug this reward function into any RL algorithm.

Performance



(a) MountainCar (b) CartPoleSwingup (c) HalfCheetah (d) state space

Figure 1: (a,b,c) TRPO+VIME versus TRPO on tasks with sparse rewards; (d) comparison of TRPO+VIME (red) and TRPO (blue) on MountainCar: visited states until convergence

- TRPO: Blue curve.
- TRPO + curiosity-driven exploration reward: Orange curve.
- Rightmost figure shows the extensive exploration behavior when the curiosity-driven exploration reward is added.

Forward Dynamics vs. Inverse Dynamics Prediction

- Learning a forward model means approximating the transition function of the MDP:

$$p(s_{t+1}|s_t, a_t; \theta_F)$$

  - θ_F is the parameter of the learned transition function
- Learning an inverse model means predicting which action transitions state s_t to state s_{t+1}:

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$$

  - θ_I is the parameter of the learned inverse model

When can the inverse model help?

- The inverse model ignores factors of variation in the data that do not affect the agent.
  - If information in the states s_t, s_{t+1} contains distractor that is not relevant to predicting a_t, the inverse model will ignore it
- For example, there are three categories of information:
  - Things that can be controlled by the agent
  - Things that the agent cannot control but that can affect the agent (e.g. a vehicle driven by another agent)
  - Things out of the agent's control and not affecting the agent (e.g. moving leaves)

Curiosity-driven Exploration by Self-supervised Prediction. Pathak et al.

Training the Inverse Model

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$$

- For MDP with discrete action space:
  - parameterize the output of g with a softmax
  - maximize the log probability of the ground truth action a_t given s_t, s_{t+1}
- Training data, consisting of tuples of (s_t, a_t, s_{t+1}), can be collected during RL training process.
- This is just a supervised learning problem.

Inverse Model Architecture

- The inverse model g consists of two sub-modules:
  - state feature encoder $\phi(s)$
  - inverse model in feature space:
    - input: feature encoding of the current state $\phi(s_t)$ and the next state $\phi(s_{t+1})$
    - output: the action a_t
- The inverse model no longer has to predict pixel values directly if the states are image.
  - Predicting pixel values are generally quite hard
- $\phi(s)$ encodes the state s into a feature space that extracts information that is only relevant to the agent.

Intrinsic Rewards using Forward Prediction in Feature Space

- $\phi(s)$ provides a good feature space.
- We train a forward dynamics model f in the feature space of ϕ:

$$\hat{\phi}\left(s_{t+1}\right) = f\left(\phi\left(s_t\right), a_t; \theta_F\right)$$

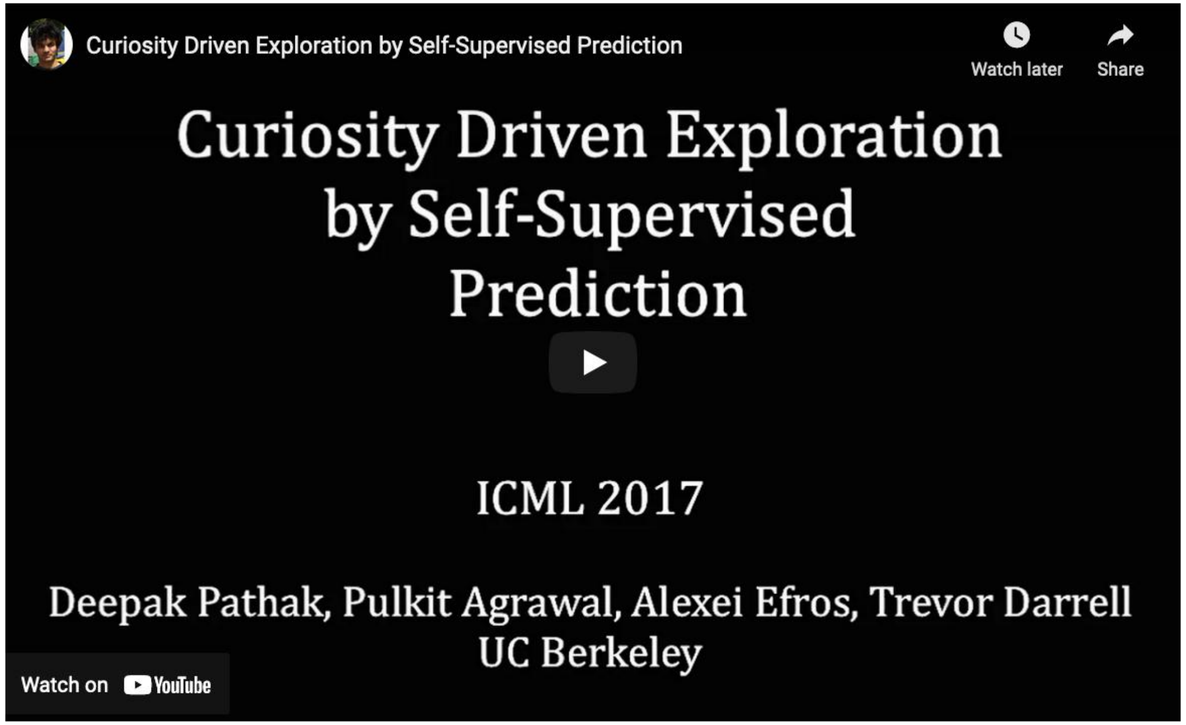
- f is trained to minimize MSE in feature space:

$$\frac{1}{2}\left\|\hat{\phi}\left(s_{t+1}\right) - \phi\left(s_{t+1}\right)\right\|_2^2$$

- Prediction error by f in feature space is used as the intrinsic reward.
- Add the intrinsic reward to the MDP reward to form a new reward function:

$$r'(s_t, a_t, s_{t+1}) = r(s_t, a_t) + \frac{\eta}{2}\left\|\hat{\phi}\left(s_{t+1}\right) - \phi\left(s_{t+1}\right)\right\|_2^2$$

Performance



- Mute video before playing.
- Video shows exploratory behavior without any MDP reward.

Principle of Occam's Razor

- Given observed data, there maybe infinitely many models that explain the data.
- How do we choose a specific model?

  For each accepted explanation of a phenomenon, there may be an extremely large, perhaps even incomprehensible, number of possible and more complex alternatives. Since failing explanations can always be burdened with ad hoc hypotheses to prevent them from being falsified, simpler theories are preferable to more complex ones because they tend to be more testable.

https://en.wikipedia.org/wiki/Occam%27s_razor

Principle of Maximum Entropy

Take precisely stated prior data or testable information about a probability distribution function. Consider the set of all trial probability distributions that would encode the prior data. According to this principle, the distribution with maximal information entropy is the best choice.

- Since the distribution with the maximum entropy is the one that makes the fewest assumptions about the true distribution of data, the principle of maximum entropy can be seen as an application of Occam's razor.

https://en.wikipedia.org/wiki/Principle_of_maximum_entropy

Entropy Regularized RL

- Let $\rho_\pi(\mathbf{s}_t)$ and $\rho_\pi(\mathbf{s}_t, \mathbf{a}_t)$ denote the state and state-action marginal distribution induced by a policy $\pi(\mathbf{a}_t \mid \mathbf{s}_t)$.
- The RL objective is:

$$\sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

- Maximum entropy RL optimizes for a different objective:

$$\sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}\left(\pi(\cdot \mid \mathbf{s}_t)\right) \right]$$

  - α is a hyper-parameter, often called the temperature parameter
  - $\mathcal{H}\left(\pi(\cdot \mid \mathbf{s}_t)\right)$ is the entropy of the policy given a state

Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy. Brian D. Ziebart PhD Thesis.

Entropy Regularization

- Maximum entropy RL optimizes for the objective:

$$\sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right) \right]$$

  - α is a hyper-parameter, often called the temperature parameter
  - $\mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right)$ is the entropy of the policy given a state
- This objective leads to more robust and better exploration behavior:
  - Encourage exploratory action around the current mean of the policy (vs. uniformly random in ϵ-greedy)
  - Assign equal density mass to equally good action
- But α is hard to tune.
  - Good values for α depend on the environment and change over the course of training in one environment.

Automatic Tuning of the Temperature

- Maximum entropy RL optimizes for the objective:

$$\sum_t \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right) \right]$$

- Soft Actor Critic (SAC) instead solves the constrained optimization problem:

$$\max_{\pi_{0:T}} \mathbb{E}_{\rho_\pi} \left[\sum_{t=0}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \quad \text{s.t. } \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[-\log(\pi_t\left(\mathbf{a}_t \mid \mathbf{s}_t\right)) \right] \geq \overline{\mathcal{H}} \quad \forall t$$

  - $\overline{\mathcal{H}}$ is the desired minimum averaged entropy (lower bound)
  - The entropy can vary across different states
  - Only the averaged entropy is constrained
  - Solving the optimization problem automatically tunes α
  - α becomes the Lagrange multipler of the optimization problem

Soft Actor-Critic Algorithms and Applications. Haarnoja et al.

SAC Objective Function

- SAC maintains value function Q_θ and policy π_ϕ.
- Solving the constrained optimization problem leads to the following objective functions for the value function, policy and α:

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) - \left(r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p}\left[V_{\bar{\theta}}\left(\mathbf{s}_{t+1}\right) \right] \right) \right)^2 \right]$$

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[\alpha \log\left(\pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right) \right) - Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) \right] \right]$$

$$J(\alpha) = \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[-\alpha \log \pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right) - \alpha \overline{\mathcal{H}} \right]$$

- $\mathcal{D}$ is the replay buffer.
- Sample-based estimates of the gradients of the three objective functions can be obtained (not shown).
  - Perform a few gradient steps for each objective function before collecting new data

SAC Pseudocode

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

SAC also maintains two value functions with parameters θ_1, θ_2 to implement double Q-learning.

SAC on Real Robot

Video shows SAC trains the Minitaur robot to walk in 2 hours of training.

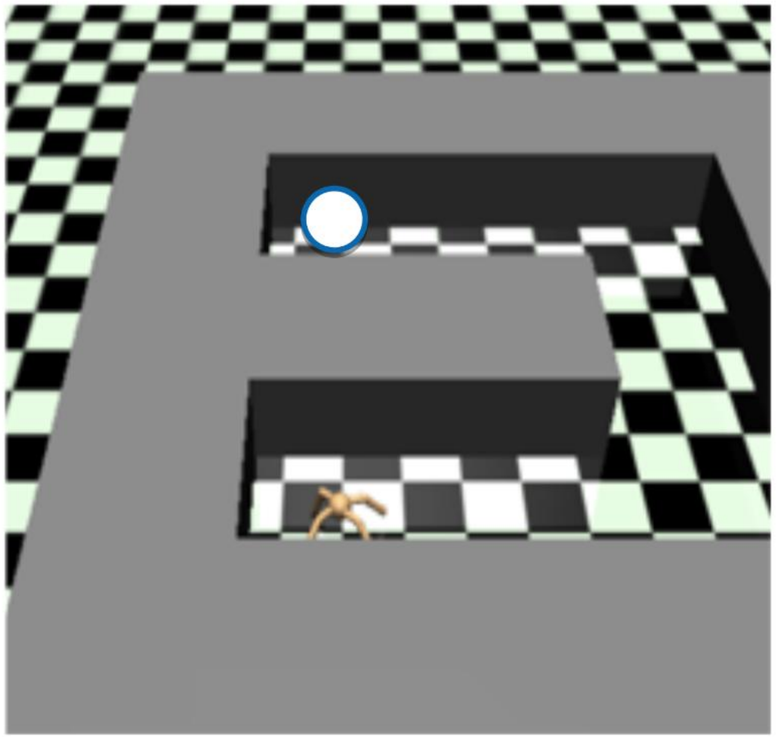
Structural Environment Modeling

Structural Environment Modeling

- We will use two case studies to show more structured environment model can improve exploration behavior:
  - Learnt high-level environment model + Search to explore frontier states
  - Environment model + Monte Carlo sampling + Search to focus computation and explore states immediately relevant

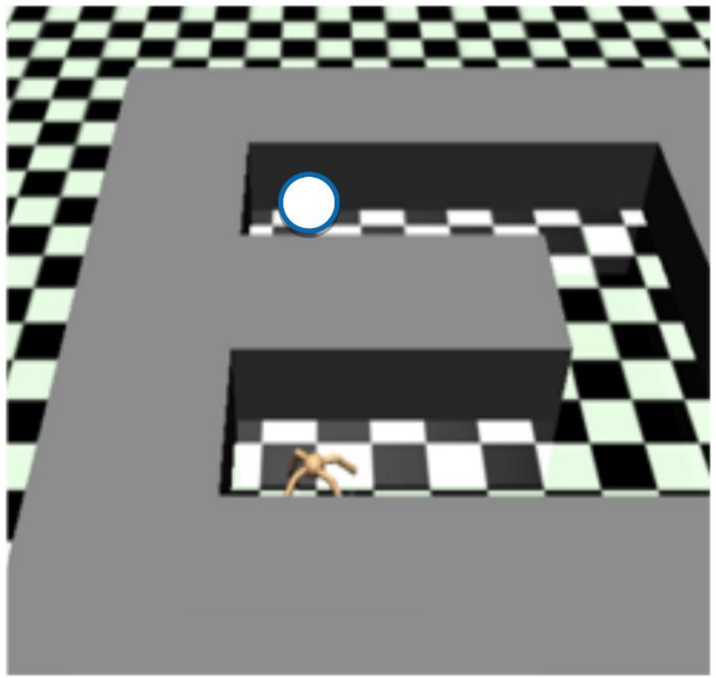
Search to Explore Frontier States

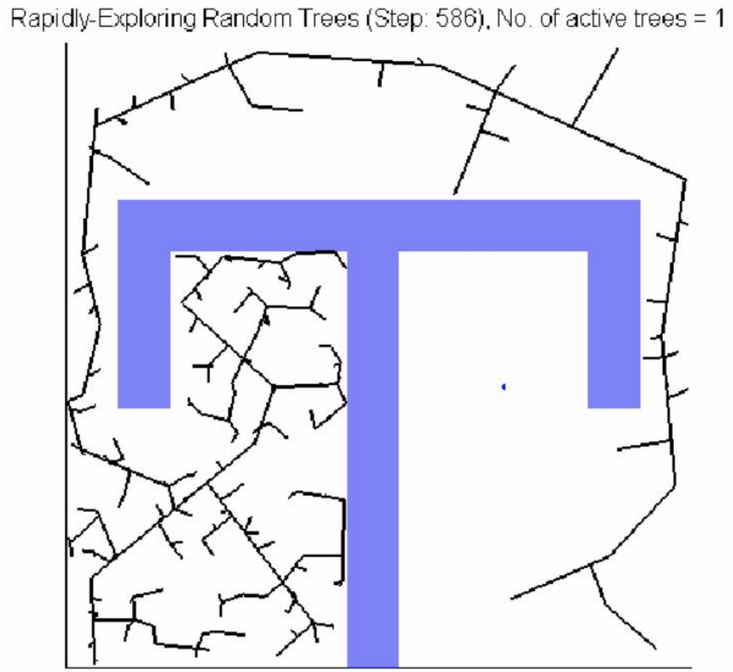
The task: an ant robot tries to reach a specific goal.



Motion Planning

If we have the transition function in closed form and environment geometry, solve it by planning algorithms.



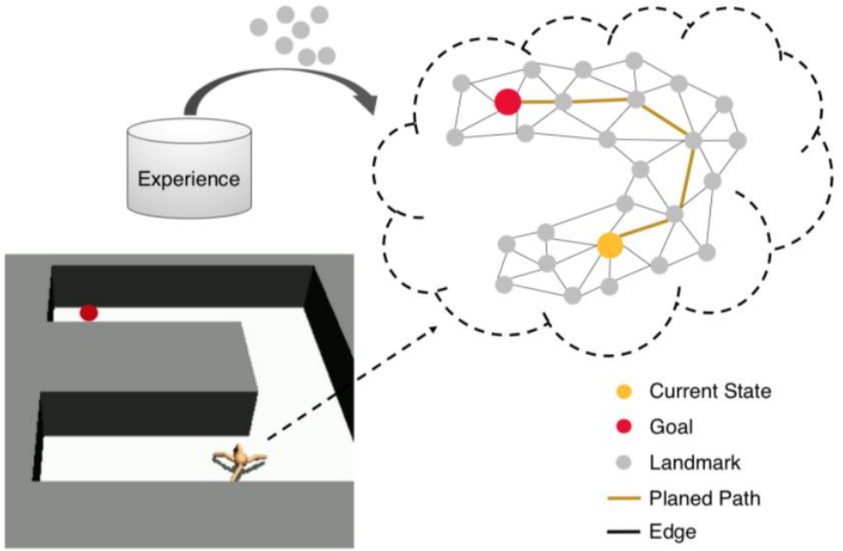


Challenges in the RL Setting

- No prior physics model => need to learn complex low-level controllers for the high DOF ant.
- No geometry model of the environment => Can not build a map.

Approach

- **Reinforcement Learning** for low-level control policy to reach nearby states.
- Mapping state space with **landmarks** from experience.
- **Search** the map to find paths to frontier states and goal states.



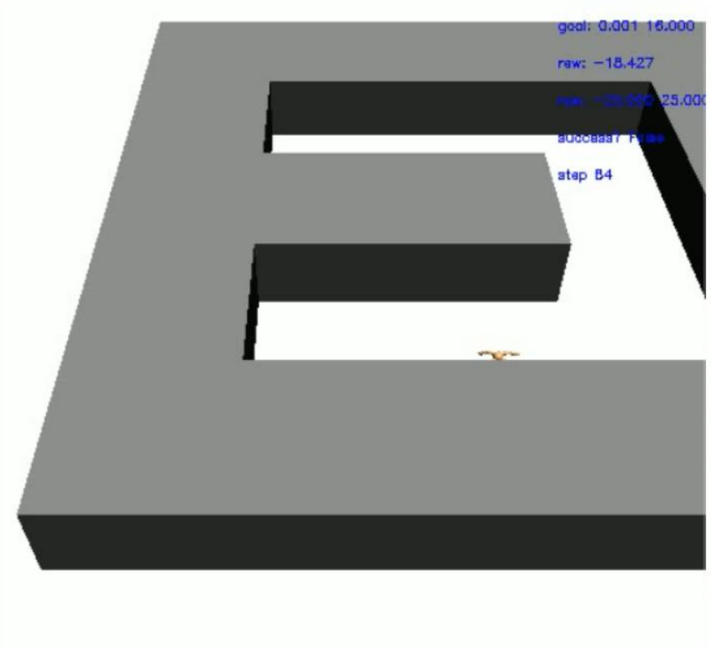
Mapping State Space using Landmarks for Universal Goal Reaching. Huang, Liu*, Su. NeuRIPS 2019.*

Reinforcement Learning for Low-level Policy

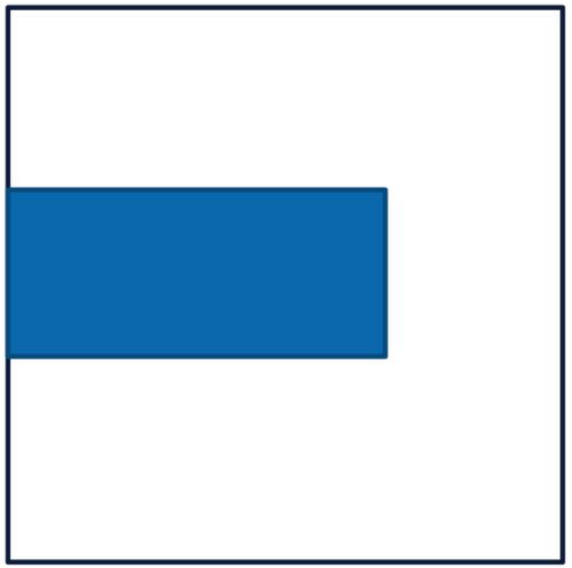
- Extend the notion of MDP with a set of goals $\mathcal{G}$ to form Universal MDP.
  - The goals $\mathcal{G}$ is often a sub-space of the state space or the state space itself.
- The reward function takes a goal as input $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to R$
- The policy is:
  - conditioned on a goal $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$
  - trained to maximize goal-condition value: $V_{g,\pi}\left(s_0\right) = E_\pi\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, a_t, g\right)\right]$
- Arbitrary collected trajectories can be repurposed to train the goal-conditioned policy and value function.
  - A very important idea and source of supervision in RL

Mapping State Space with Landmarks

- State space can be embedded into a low-dimensional manifold.



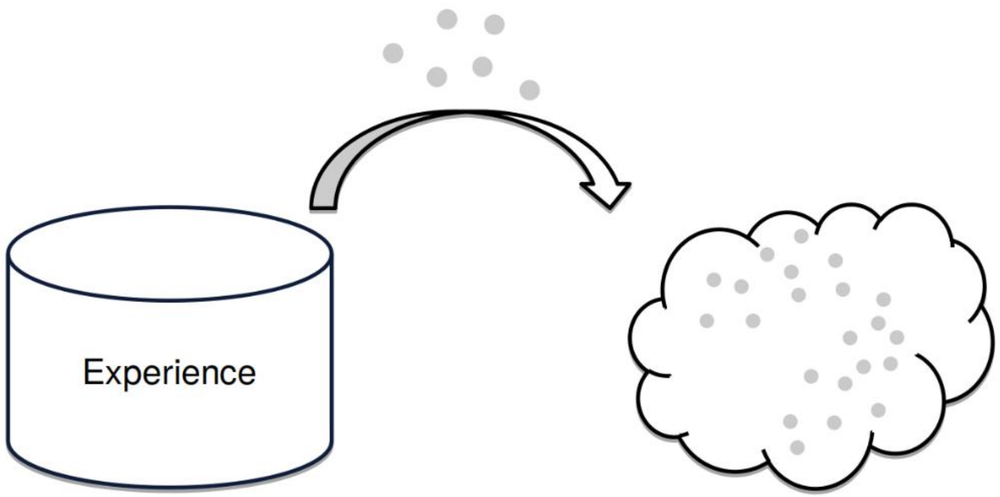
Global Structure



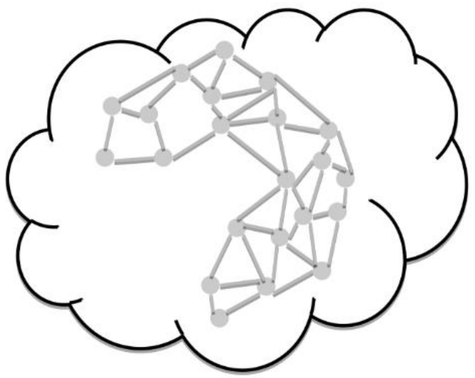
Simple 2D map

Mapping State Space with Landmarks

- Sample landmarks in state space.
- Build a graph in state space.
  - Connecting "nearby" landmarks
  - Navigating between "nearby" landmarks is a short-horizon task and solved by the RL agent



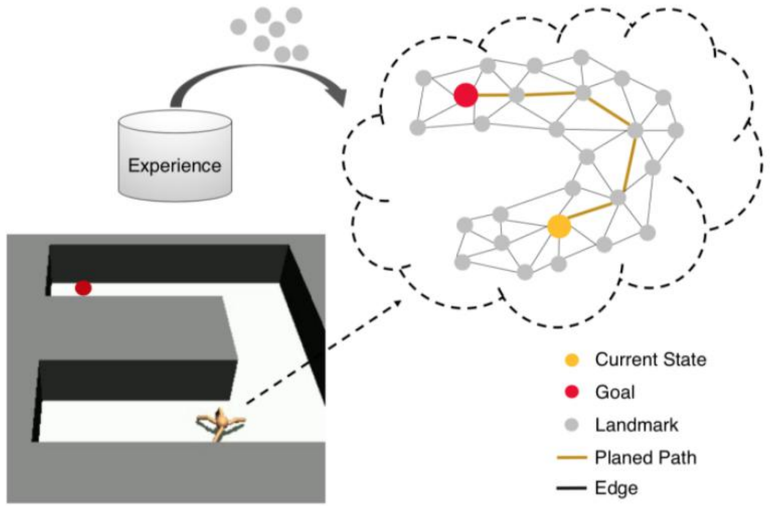
Landmark sampling to approximate state space



Build a graph connecting landmarks

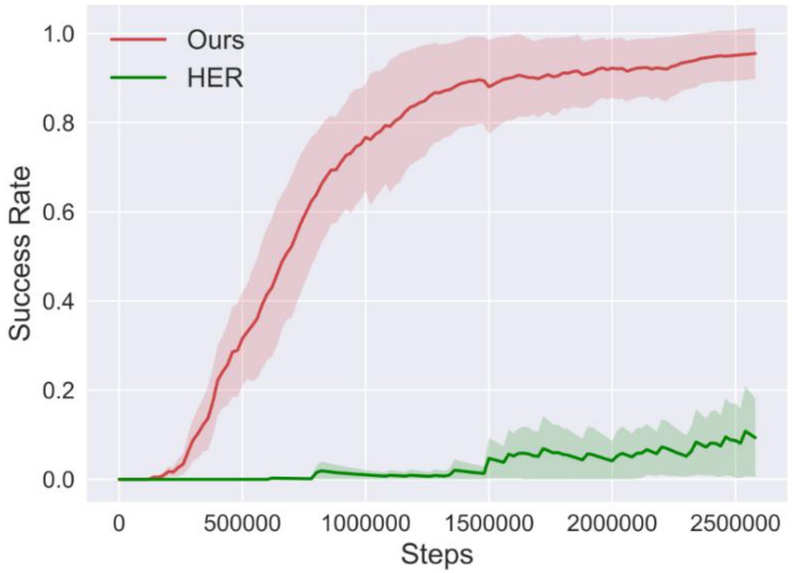
Mapping, Planning and RL

- The algorithm consists of a few high-level ideas:
  - Sample landmarks from replay buffer.
  - Build a graph whose nodes are landmarks.
  - Plan a path to the goal or frontier states (nodes at the edge of the graph).
  - Landmark traversal executed by goal-conditioned RL policy (landmark-conditioned policy in this case).



Performance

- The agent only receives a positive reward when reaching the goal.



Search to Explore Sub-MDP

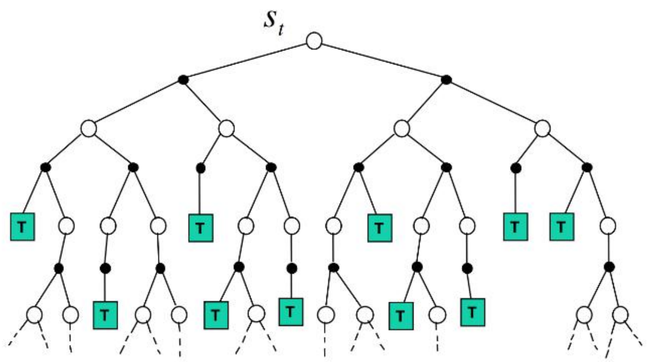
- All previous methods seek to find a good policy for every state in the state space of the MDP (global optimization).
- Solving the entire MDP is very hard for problems with large state space.
  - The observable universe contains roughly 10^{80} atoms
  - Go has about 10^{170} valid board configurations for professional board size (19×19)
- Can we focus computation on part of the state space that is more important?
  - More important can be seen as immediately relevant
    - For example: doing well in CSE291D this quarter is more important than what you will eat for lunch next year
  - With this perspective, the current state and the states that immediately follow are most important

Search to Explore Sub-MDP

- There are two main ideas, which we will explain next:
  - Forward Search
  - Sampling
- This is a very different perspective compared to previous RL methods you have seen:
  - In previous methods, to pick an action at test time, we just need to perform a forward pass through the policy
  - In this family of algorithms, we perform a lot more computation at test time

Forward Search

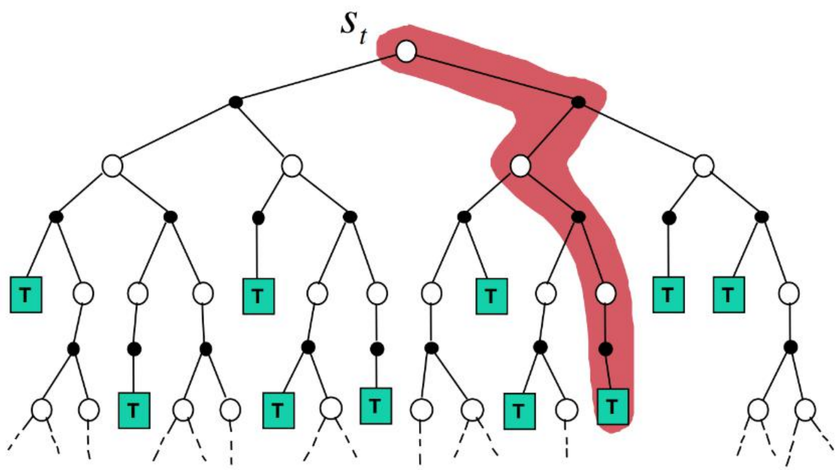
- Assume we are given the MDP state space, action space, transition and reward function.
- To compute action a_t given current state s_t, forward search focuses computation with these steps:
  - Build a search tree with the current state s_t at the root
  - Use the model of the MDP to perform lookahead
    - lookahead means running simulation using the model to compute state-action value estimates
    - computing the value estimates can be done with model-free RL algorithm (such as Q-learning)
  - Select the best action based on value estimates computed by exhaustive lookahead
- No need to solve the whole MDP, only the sub-MDP starting from the current state.



Sampling-based Forward Search

- Naive forward search performs exhaustive lookahead.
- Sampling-based forward search focuses computations by sampling to perform the simulation:
  - Sampling actions from the current policy and transitions from the MDP model
  - Apply model-free RL to estimate state-action values from the simulated episodes

  $\Longrightarrow$Only perform lookahead on actions and transitions that occur with high probability.
  - Case study: Monte-Carlo Tree Search



Simple Monte-Carlo Search

- Given the state space, action space, transition and reward function of MDP $\mathcal{M}$.
- Pick a **fixed** default policy π.
- Given the current state s_t, perform rollouts using the model of MDP and π:
  - For each valid action a, simulate K episodes of experience

$$\left\{s_t, a, R_{t+1}^k, S_{t+1}^k, A_{t+1}^k, \ldots, S_T^k\right\}_{k=1}^K \sim \mathcal{M}, \pi$$

  - Estimate the value of each action by Monte-Carlo estimation

$$\hat{Q}\left(s_t, a\right) = \frac{1}{K} \sum_{k=1}^{K} G_k$$

    - where G_k is the return of the k-th episode.
- To pick the real action taken in the MDP, pick action with the highest estimated value:

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, \hat{Q}\left(s_t, a\right)$$

Limitations of Simple Monte-Carlo Search

- Note that across different episodes, the default policy π is fixed and does not improve.
  - The quality of the action does not improve across different episodes
- We only estimate action values for the current state s_t.
- We now introduce tree as a data structure to store value estimates and improve the policy.

Monte-Carlo Tree Search

- Given the state space, action space, transition and reward function of MDP $\mathcal{M}$.
- Pick a **fixed** default policy π.
- Given the current state s_t, perform rollouts for K simulated episodes:
  - Initialize empty search tree
  - Each episode starts from the current state s_t
  - For each episode, to pick action for a state s:
    - If s is not in the search tree, use the default policy to pick action
    - If s is in the search tree, pick action that maximize current value estimate $\hat{Q}(s, a)$
  - After each episode:
    - Estimate the value for all state-action pairs visitted in the episode using Monte-Carlo estimation
    - Add all new state-action pairs and their value estimates to the search tree
    - Update the value estimates of existing state-action pairs in the search tree
- To pick the real action taken in the MDP from state s_t, pick action with highest estimated value.

Monte-Carlo Tree Search

- In MCTS, there are thus 2 policies, so-called tree policy and the default policy:

	When to use?	**Action selection method?**	**Improve across epsisodes?**
Tree Policy	When a state s is in the search tree	Select action with highest estimated state-action value	Yes, since the state-action value estimates are updated and improved across episodes
Default Policy	When a state s is not in the search tree	Manually defined prior to any simulation, e.g. uniformly random	No

- To improve exploration by the tree policy, we can use ϵ-greedy or bandit algorithms.
- Note that the search tree is discarded after picking action a_t and not re-used when computing action for s_{t+1}.
- The value estimates are also stored in a table lookup (not feasible for large state space).
- Using a neural network to store the value estimates will solve these two issues (AlphaGo).

Performance

Game	Date	Black	White	Result	Moves
1	9 March 2016	Lee Sedol	**AlphaGo**	Lee Sedol resigned	186 Game 1
2	10 March 2016	**AlphaGo**	Lee Sedol	Lee Sedol resigned	211 Game 2
3	12 March 2016	Lee Sedol	**AlphaGo**	Lee Sedol resigned	176 Game 3
4	13 March 2016	AlphaGo	**Lee Sedol**	AlphaGo resigned	180 Game 4
5	15 March 2016	Lee Sedol[note 1]	**AlphaGo**	Lee Sedol resigned	280 Game 5
Result: **AlphaGo 4 – 1 Lee Sedol**					

Source: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol.

Debugging Tips

Monte-Carlo Estimates to Sanity-check Trained Components

- All components in a RL algorithm are estimated.
- For environments implemented in simulation, we can obtain Monte-Carlo estimates and compare them against the trained components to sanity-check
  - use Monte-Carlo because of its unbiasedness property
  - the Monte-Carlo estimates will converge to the true value given enough samples
- To save time, parallelize the sampling to obtain Monte-Carlo estimates while the policy is training
  - a popular Python library for RL parallelism is Ray

Setting Seeds

- The performance of RL algorithms have high variance, especially for long horizon problem with sparse reward.
- Set the random seeds for every possible library (numpy, python math, pytorch cpu and gpu, etc).
- Ensure your experiment produces the same exact number, even if stopped, re-loaded and re-run.
- To evalute the performance of RL algorithms, average performance across multiple seeds:
  - Do not set your seed like: for i in range(5): set_seed(i)
  - Having linear correlations between seeds can correlate the output [1]

[1] https://github.com/openai/gym/blob/a5a6ae6bc0a5cfc0ff1ce9be723d59593c165022/gym/utils/seeding.py#L20

End