MC and TD Learning

Whats New?

- We want to estimate the value function without having access to the dynamics model
- We want to interact with the environment and collect experiences
- We average sampled returns to calculate the expectations (earlier we used the known probability distribution)

Prediction Problem vs Control Problem

- **Prediction problem** : Policy evaluation, calculating the value function of a given policy pi (eg. Monte Carlo Prediction)
- **Control problem** : General policy iteration (GPI), getting the optimal policy (eg. Monte Carlo Control)

Monte Carlo Methods

- All episodes must terminate (learning can only happen from complete trajectories)
- Only defined for episodic tasks
- Basic idea : Vpi(s) = mean return

Monte Carlo Prediction / Monte Carlo Policy Evaluation

- Goal : Learn Vpi(s) using episodes under policy pi
- Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-1} R_T$$

- Value function using the expected return

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

- We use the empirical mean return to estimate the expected return

Every Visit vs First Visit MC

- **Every visit** : while averaging, consider returns for every time state s is visited in a trajectory
- **First visit** : while averaging, consider returns only for the first time strate s is visited in a trajectory

Monte Carlo Control

- Monte Carlo Prediction / Monte Carlo Policy Evaluation
- Policy Improvement Step : gredify with respect to the value or action value function

Exploration Problem

- Dilemma : Policies need to act sub optimally in order to explore all actions

- Solutions :
  - **Exploring Starts** : every state action pair has non-zero probability of being the start
  - **Epsilon Soft Policies** : Policy never becomes deterministic, epsilon is always non-zero
  - **Off Policy** : Use different policy to collect experiences (behavior policy) and then update the target policy
    - Importance Sampling : Weight returns by the ratio of probabilities of the same trajectory under the two policies

Temporal Difference (TD) Learning

- General equation:

$$V(S_t) \leftarrow V(S_t) + \alpha\,(G_t - V(S_t))$$

- TD(0) / One Step TD :

$$V(S_t) \leftarrow V(S_t) + \alpha\,(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

TD Target : $R_{t+1} + \gamma V(S_{t+1})$

TD Error : $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$

TD Prediction / TD Policy Evaluation

- Goal : Find Vpi(s) using episodes from policy pi
- TD(0) uses:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

Advantages of TD Learning

- Like MC, does not need to know the model dynamics
- Bootstrap : learning a guess from a guess
- Does not need to end for the end of an episodes (some episodes can be really long!)
- Requires lesser memory and computation
- In practice, converges faster than MC methods

Bias Variance Trade-Off between MC and TD

- MC has zero bias, high variance
- TD has low variance, some bias

SARSA and Q-Learning

- SARSA is On-Policy TD Control
- Q-Learning is Off-Policy TD Control

MCTS

Online Planning

- Concept of ‘mental unrolling’
- Unroll the model of the environment forward in time to select the right action sequences to achieve your goal

- Since we have the model, why not learn the value function of every state directly?
- Because there can be extremely large number of possible states

Limitation : Cannot Exhaustively Search

- Curse of dimensionality : not possible to search the entire tree

- Solution :
  - **Reduce Depth** by truncating the search stree at a state s and then use the approximate value of s
  - **Reduce Breadth** by sampling actions from a policy pi(a|s) which is probability distribution of actions given state s

Internal and External Policies

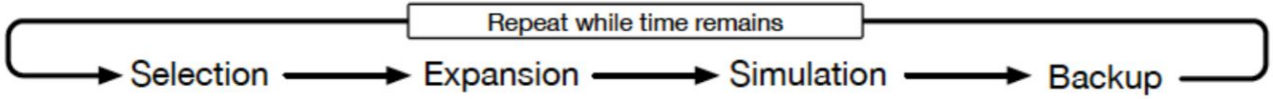
- **Internal Policy** : Keep track of action values for the root and nodes internal to the tree (which is being expanded), and use these to improve the simulation policy over time
- **External Policy** : We donot have Q estimates, therefore we use a random policy

Upper Confidence Bound

$$A_t = \text{argmax}_a \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$$

- Probability of choosing an action:
  - Increases with a node's value : Qt(a) (exploitation)
  - Decreases with number of visits : Nt(a) (exploration)

The 4 Steps



- Selection :
  - For nodes we have seen before
  - Pick actions according to UCB
- Expansion :
  - Used when we reach the frontier
  - Add one node per rollout
- Simulation :
  - Used beyond the frontier
  - Pick actions randomly
- Back-Propagation :
  - After reaching a terminal node
  - Update values and visits for selected and expanded states

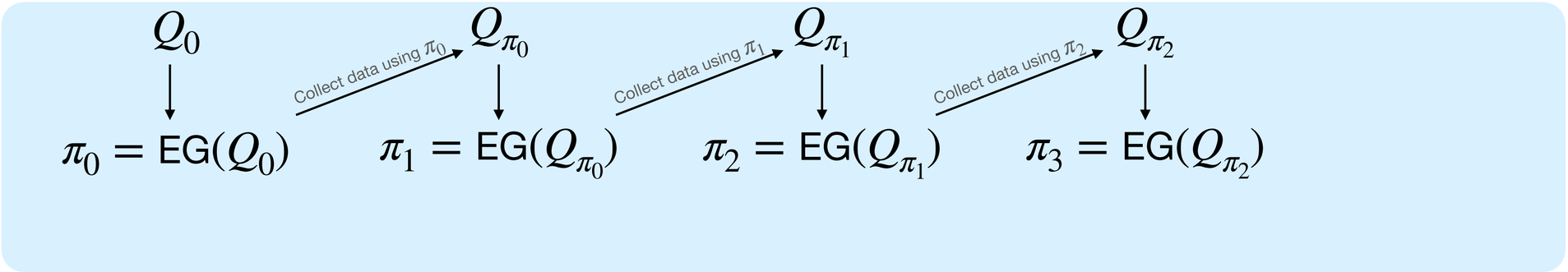
Review: REINFORCE, RwB, A2C

Recitation 7 10-403

Conor Igoe — April 2 2021

REINFORCE, RwB, A2C

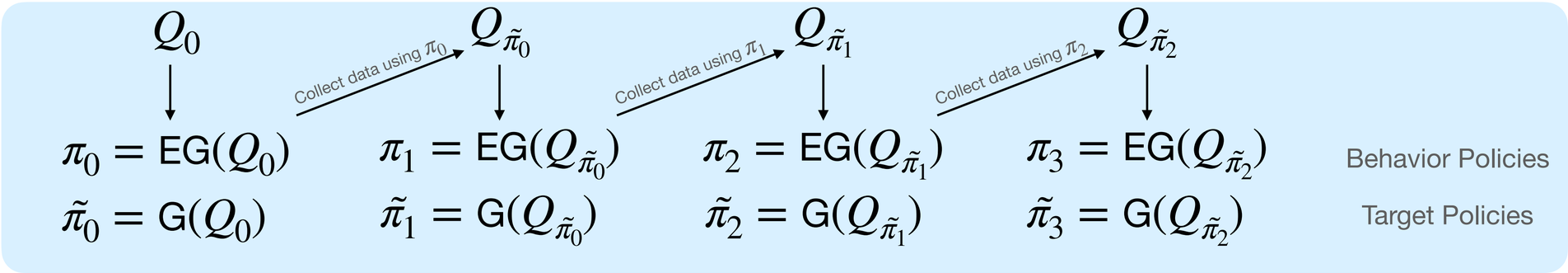
Overview

- Shift in approach: most of the course up until REINFORCE focused on action-value methods for RL:
  - 

  - Different version based on: replacement frequency, Q function estimation (e.g. with or w/o function approximation, with or w/o bootstrapping), behaviour & target policy construction
  - Some example algorithms include tabular SARSA, tabular Q-Learning, semi-gradient N-step SARSA
  - **The important point is that the policy is always defined in terms of some sort of greedification of a q function**

REINFORCE, RwB, A2C

Overview

- Shift in approach: most of the course up until REINFORCE focused on action-value methods for RL:
  - 

  - Different version based on: replacement frequency, Q function estimation (e.g. with or w/o function approximation, with or w/o bootstrapping), behaviour & target policy construction
  - Some example algorithms include tabular SARSA, tabular Q-Learning, semi-gradient N-step SARSA
  - **The important point is that the policy is always defined in terms of some sort of greedification of a q function**

REINFORCE, RwB, A2C, N-step A2C

Overview

- π_θ vs Q_θ

- Advantages:
  - trivial to consider continuous action spaces
  - gives us an opportunity to inject domain knowledge into the problem

REINFORCE, RwB, A2C, N-step A2C

Overview

Towards Generalization and Simplicity in Continuous Control

Aravind Rajeswaran* **Kendall Lowrey*** **Emanuel Todorov** **Sham Kakade**

University of Washington Seattle

{ aravraj, klowrey, todorov, sham } @ cs.washington.edu

Abstract

This work shows that policies with simple linear and RBF parameterizations can be trained to solve a variety of widely studied continuous control tasks, including the OpenAI gym benchmarks. The performance of these trained policies are competitive with state of the art results, obtained with more elaborate parameterizations such as fully connected neural networks. Furthermore, the standard training and testing scenarios for these tasks are shown to be very limited and prone to over-fitting, thus giving rise to only trajectory-centric policies. Training with a diverse initial state distribution induces more global policies with better generalization. This allows for interactive control scenarios where the system recovers from large on-line perturbations; as shown in the supplementary video.

REINFORCE, RwB, A2C, N-step A2C

Overview

- Formal Setup
- Policy Gradient Theorem Recap
- Obtaining Estimators from the Policy Gradient Theorem
- REINFORCE
- REINFORCE with Baseline (RwB)
- (N-step) Advantage Actor Critic (A2C)

REINFORCE, RwB, A2C, N-step A2C

Formal Setup

- Finite episodic MDP
  - $|\mathcal{S}| < \infty$, $|\mathcal{A}| < \infty$, $|\mathcal{R}| < \infty$, $\max(\mathcal{R}) < \infty$, $\min(\mathcal{R}) > -\infty$
  - For every initial condition $s_0 \in \mathcal{S}$, no matter what sequence of actions taken, guaranteed to reach terminal state $s \in \mathcal{S}_{\text{term}} \subset \mathcal{S}$ in finitely many timesteps
- "Nice" stochastic policy class Π parameterised by θ
  - $\pi_\theta(a \mid s)$ differentiable in θ for every (s, a)
  - All policies in Π output non-zero probabilities for all actions in all states

REINFORCE, RwB, A2C, N-step A2C

Formal Setup

- Objective function:

$$J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ \sum_{t=0}^{T} \gamma^t R_t \right\}, \gamma \in (0,1)$$

- $\mathbb{P}_{\pi_\theta}$ is a probability distribution over trajectories $\tau = S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  - $S_0 \sim \rho_0(\cdot)$
  - $A_t \sim \pi_\theta(\cdot \mid S_t) \quad \forall t \geq 0$
  - $(S_{t+1}, R_{t+1}) \sim p(\cdot, \cdot \mid S_t, A_t) \quad \forall t \geq 0$
  - T is random (depends on whether $S_t \in \mathcal{S}_{\text{term}}$, but guaranteed to be finite)

REINFORCE, RwB, A2C, N-step A2C

Policy Gradient Theorem Recap

- Policy Gradient Theorem (version 1)
  - $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ G_0 \sum_{t=0}^{T-1} \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right) \right\}$$
    - $$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$
      - We derived this in class; relies on the log ratio trick and decomposition of $\mathbb{P}_{\pi_\theta}$ into policy and dynamics components.

REINFORCE, RwB, A2C, N-step A2C

Policy Gradient Theorem Recap

- Policy Gradient Theorem (version 2)
- $$\nabla_\theta J(\theta) = \frac{1}{1-\gamma}\mathbb{E}_{(S,A)\sim\mu_{\pi_\theta}}\left\{ q_{\pi_\theta}(S,A)\,\nabla_\theta \log\left(\pi_\theta(A \mid S)\right)\right\}$$
  - Didn't derive in class but often shows up in literature so good to know (not on quiz)
  - μ_{π_θ} is the discounted state-action "visitation" distribution under policy π_θ

REINFORCE, RwB, A2C, N-step A2C

Policy Gradient Theorem Recap

- Policy Gradient Theorem (version 2)
  - $$\nabla_\theta J(\theta) = \frac{1}{1-\gamma}\mathbb{E}_{(S,A)\sim\mu_{\pi_\theta}}\left\{ q_{\pi_\theta}(S,A)\,\nabla_\theta \log\left(\pi_\theta(A\,|\,S)\right)\right\}$$
    - $\mu_{\pi_\theta}(s,a) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}(S_t = s, A_t = a)$
      - Draw a timestep $M \sim$ Geometric$(1-\gamma)$
      - Then deploy π_θ for M timesteps and observe $S_0, A_0, R_1, \ldots, S_{M-1}, A_{M-1}$
      - (S_{M-1}, A_{M-1}) will be distributed according to μ_{π_θ}

REINFORCE, RwB, A2C, N-step A2C

Policy Gradient Theorem Recap

- Policy Gradient Theorem (version 2)
  - $$\nabla_\theta J(\theta) = \frac{1}{1-\gamma}\mathbb{E}_{(S,A)\sim\mu_{\pi_\theta}}\left\{ q_{\pi_\theta}(S,A)\,\nabla_\theta \log\left(\pi_\theta(A \mid S)\right)\right\}$$
    - $$\mu_{\pi_\theta}(s,a) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t \mathbb{P}(S_t = s, A_t = a)$$
      - See Sham Kadade's RL Theory Notes for derivation (Theorem 9.4)

REINFORCE, RwB, A2C, N-step A2C

Obtaining Estimators from the Policy Gradient Theorem

- Policy Gradient Theorem gives us a starting point to build an estimators:
  - E.g. using version 1:
    - $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ G_0 \sum_{t=0}^{T} \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right) \right\}$$
    - $$\hat{g} = G_0 \sum_{t=0}^{T-1} \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right)$$ (Take a single sample from the RV in the Policy Gradient Theorem)

REINFORCE, RwB, A2C, N-step A2C

Obtaining Estimators from the Policy Gradient Theorem

- Policy Gradient Theorem gives us a starting point to build estimator:
  - E.g. using version 1:
    - $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ G_0 \sum_{t=0}^{T} \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right) \right\}$$
    - $$\hat{g} = \sum_{t=0}^{T-1} G_t \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right)$$

(Inject temporal structure; this is the REINFORCE policy gradient estimate; note that it is basically a single sample estimator of the Expectation from the PG theorem, so expect very high variance!)

REINFORCE, RwB, A2C, N-step A2C

REINFORCE

1: **procedure** REINFORCE
2: *Start with policy network* π_θ
3: **repeat for** E **training episodes:**
4: *Generate an episode* $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ *following* $\pi_\theta(\cdot)$
5: **for** $t = 0, 1, \ldots, T$:
6: $G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
7: $\hat{g} = \sum_{t=0}^{T-1} G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$
8: $\theta \leftarrow \theta + \alpha \hat{g}$
9: **end procedure**

REINFORCE, RwB, A2C, N-step A2C

RwB

- Subtracting a state dependent baseline doesn't affect bias

$$\hat{g} = \sum_{t=0}^{T-1} \left(G_t - b(S_t)\right) \nabla_\theta \log\left(\pi_\theta(A_t \mid S_t)\right)$$

  - Convenient to choose $b(s) = \hat{V}_\phi(s)$
  - Baseline can be learned using Monte Carlo methods, or semi-gradient TD methods; both are known as RwB
  - Everything else exactly the same as REINFORCE

REINFORCE, RwB, A2C, N-step A2C

(N-step) A2C

- Take a look at the following version of the true Policy Gradient (with a baseline):
- $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ \sum_{t=0}^{T-1} \left(G_t - b(S_t) \right) \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right) \right\}$$
- The true Policy Gradient involves taking an expectation over several terms. In Vanilla REINFORCE with Baseline, we really were just using a single sample from the random variable under the expectation (in blue) as our estimator
- The main idea with Actor-Critic methods is to do the same thing, but using a critic to approximately perform *part* of this expectation by relying on previous data to reduce the variability coming from G_t terms i.e. use an estimate of $\mathbb{E}\{G_t \mid S_t, A_t\}$ instead of G_t

REINFORCE, RwB, A2C, N-step A2C

(N-step) A2C

- Take a look at the following version of the true Policy Gradient (with a baseline):
- $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ \sum_{t=0}^{T-1} \left(G_t - b(S_t) \right) \nabla_\theta \log \left(\pi_\theta(A_t \,|\, S_t) \right) \right\}$$
- If we use $\hat{V}_\phi(s)$ to construct estimates of $\mathbb{E}\{G_t \,|\, S_t, A_t\}$ by bootstrapping then we call $\hat{V}_\phi$ a *critic* (and π_θ an *actor*)
  - For example:
    $$\mathbb{E}\{G_t \,|\, S_t, A_t\} \approx R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \gamma^{T-t} R_T \quad \text{(high variance but unbiased)}$$
    $$\mathbb{E}\{G_t \,|\, S_t, A_t\} \approx R_{t+1} + \gamma \hat{V}_\phi(S_{t+1}) \quad \text{(lower variance but biased)}$$
  - Bootstrapping for 1 time-step = Advantage Actor-Critic (A2C)
  - Bootstrapping for N time-steps = N-step A2C

REINFORCE, RwB, A2C, N-step A2C

(N-step) A2C

- Take a look at the following version of the true Policy Gradient (with a baseline):
- $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ \sum_{t=0}^{T-1} \left(G_t - b(S_t) \right) \nabla_\theta \log \left(\pi_\theta(A_t \mid S_t) \right) \right\}$$

- **Why do we say $R_{t+1} + \gamma \hat{V}_\phi(S_{t+1})$ has lower variance?**
  - Because it is a lower variance estimate of $\mathbb{E}\{G_t \mid S_t, A_t\}$, and $\hat{V}_\phi(s)$ has averaged over much of the randomness that previously existed in G_t
- **Why do we say using $R_{t+1} + \gamma \hat{V}_\phi(S_{t+1})$ introduces bias?**
  - Imagine using a really inexpressive network for $\hat{V}_\phi(s)$. Then our estimates for $\mathbb{E}\{G_t \mid S_t, A_t\}$ will be very poor and we can never hope to estimate the policy gradient well!

REINFORCE, RwB, A2C, N-step A2C

(N-step) A2C

- Take a look at the following version of the true Policy Gradient (with a baseline):
  - $$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\pi_\theta}} \left\{ \sum_{t=0}^{T-1} \left(G_t - b(S_t) \right) \nabla_\theta \log \left(\pi_\theta(A_t \,|\, S_t) \right) \right\}$$
- $$\hat{g} = \sum_{t=0}^{T-1} \left(R_{t+1} + \gamma \hat{V}_\phi(S_{t+1}) - \hat{V}_\phi(S_t) \right) \nabla_\theta \log \left(\pi_\theta(A_t \,|\, S_t) \right)$$
  - Note that the critic network is used in two places: baseline & to bootstrap an estimate for $\mathbb{E}\{G_t \,|\, S_t, A_t\}$

Natural PG, TRPO, PPO

A review

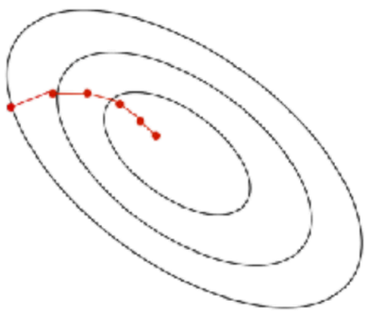
Yuning Wu, 4/2/2021

Choosing a proper **stepsize** in policy gradient is critical.

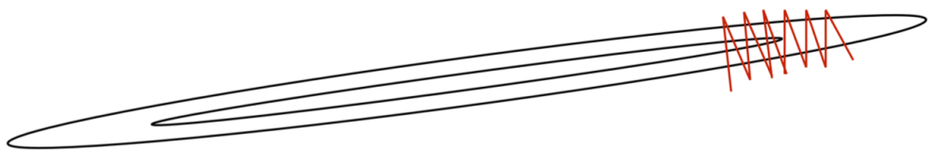
- Too big, can't recover from bad moves.
- Too small, low sample efficiency.

The spirit of natural policy gradient, is to take a **constrained** GD step.

- Euclidean distance in parameter space as constraint is unfavorable.
- The threshold is hard to pick.
- Perform badly in ill-conditioned curvature.



normal case



ill conditioned surface

Bounce around in high curvature direction. Make slow progress in low curvature direction.

Natural PG

The spirit of natural policy gradient, is to take a **constrained** GD step.

- Use KL-divergence in distribution space.
- The threshold is easy to pick.
- Solves the curvature issue.

$$\theta_{new} = \theta_{old} + d*$$
$$\theta' = \theta + d*$$
$$d* = arg \max_{KL[\pi_\theta \| \pi_{\theta+d}] \leq \epsilon} U(\theta + d)$$

Two questions.

1. What is the natural policy gradient? (Direction)
2. What is the stepsize? (Scale)

$$d* = arg\max_{d} U(\theta + d) - \lambda \left(KL \left[\pi_\theta \| \pi_{\theta+d} \right] - \epsilon \right)$$

Answer to 1st question, three ingredients.

1. Turning a constrained objective into unconstrained penalized objective.
2. Taylor expansion.
3. KL-divergence Hessian / Fisher Information matrix.

$$d^* \approx arg\max_d U(\theta_{old}) + \nabla_\theta U(\theta)|_{\theta=\theta_{old}} \cdot d - \frac{1}{2}\lambda \left(d^\top \cdot \nabla_\theta^2 KL\left[\pi_{\theta_{old}} \| \pi_\theta\right]\Big|_{\theta=\theta_{old}} \cdot d \right) + \lambda\epsilon$$

Answer to 1st question, three ingredients.

1. Turning a constrained objective into unconstrained penalized objective.
2. Taylor expansion.
    - Please refer to lecture slides for thorough expansion.
    - If you have hesitation about Taylor expansion, please come to OH.
3. KL-divergence Hessian / Fisher Information matrix.

$$d^* \approx arg\max_d U(\theta_{old}) + \nabla_\theta U(\theta)|_{\theta=\theta_{old}} \cdot d - \frac{1}{2}\lambda\left(d^\top \cdot \nabla^2_\theta KL\left[\pi_{\theta_{old}} \| \pi_\theta\right]\Big|_{\theta=\theta_{old}} \cdot d\right) + \lambda\epsilon$$

Answer to 1st question, three ingredients.

1. Turning a constrained objective into unconstrained penalized objective.
2. Taylor expansion.
3. KL-divergence Hessian / Fisher Information matrix.
    - Just plug in the definition of KL-divergence.

$$\nabla_\theta^2 KL\left[\pi_{\theta_{old}} \| \pi_\theta\right]\Big|_{\theta=\theta_{old}} = \mathbb{E}_{x\sim\pi_{\theta_{old}}}\left[\nabla_\theta \log \pi_\theta(x)\,|_{\theta=\theta_{old}} \cdot \nabla_\theta \log \pi_\theta(x)\,|_{\theta=\theta_{old}}^{\top}\right]$$

Answer to 1st question, three ingredients.

1. Turning a constrained objective into unconstrained penalized objective.
2. Taylor expansion.
3. KL-divergence Hessian / Fisher Information matrix.
    - Just plug in the definition of KL-divergence.

$$\mathbb{E}_{x\sim\pi_{\theta_{old}}}\left[\nabla_\theta \log \pi_\theta(x)\,|_{\theta=\theta_{old}} \cdot \nabla_\theta \log \pi_\theta(x)\,|_{\theta=\theta_{old}}^{\top}\right] = \boldsymbol{F}$$

Answer to 1st question, three ingredients.

1. Turning a constrained objective into unconstrained penalized objective.
2. Taylor expansion.
3. KL-divergence Hessian / Fisher Information matrix.
   - Just plug in the definition of KL-divergence.
   - This is just the Fisher Information matrix! [post link]

$$d^* \approx arg\max_d U(\theta_{old}) + \nabla_\theta U(\theta)|_{\theta=\theta_{old}} \cdot d - \frac{1}{2}\lambda\left(d^\top \cdot \boldsymbol{F} \cdot d\right) + \lambda\epsilon$$

Answer to 1st question, three ingredients.

1. Turning a constrained objective into unconstrained penalized objective.
2. Taylor expansion.
3. KL-divergence Hessian / Fisher Information matrix.
    - Just plug in the definition of KL-divergence.
    - This is just the Fisher Information matrix! [post link]

$$d^* \approx arg\max_d U(\theta_{old}) + \nabla_\theta U(\theta)|_{\theta=\theta_{old}} \cdot d - \frac{1}{2}\lambda\left(d^\top \cdot \boldsymbol{F} \cdot d\right) + \lambda\epsilon$$

$$d^* \approx arg\max_d \nabla_\theta U(\theta)|_{\theta=\theta_{old}} \cdot d - \frac{1}{2}\lambda\left(d^\top \cdot \boldsymbol{F} \cdot d\right)$$

Answer to 1st question, a standard optimization problem now.

1. Take gradient.
2. Set to zero.

$$d* = \frac{2}{\lambda} \cdot \boldsymbol{F}^{-1} \nabla_\theta U(\theta)\,|_{\theta=\theta_{old}}$$

$$d* = \frac{2}{\lambda} \cdot g_N$$

Answer to 1st question, solved.

- g_N is the natural gradient we are looking for.

$$KL\left[\pi_{\theta_{old}} \| \pi_{\theta}\right] \approx \frac{1}{2}(\alpha g_N)^{\top} \boldsymbol{F} (\alpha g_N) \approx \epsilon$$

Answer to 2nd question, the stepsize α.

- Assumption: we want the KL-divergence between old and new policy to be at most ϵ.
- Use second-order Taylor expansion again.

$$\alpha = \sqrt{\frac{2\epsilon}{g_N^\top \boldsymbol{F}^{-1} g_N}}$$

Answer to 2nd question, the stepsize α, solved.

- Assumption: we want the KL-divergence between old and new policy to be at most ϵ.
- Use second-order Taylor expansion again.

Algorithm 1 Natural Policy Gradient

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **do**
Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian / Fisher Information Matrix $\hat{H}_k$

Compute Natural Policy Gradient update:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{\hat{g}_k^T \hat{H}_k^{-1} \hat{g}_k}} \hat{H}_k^{-1} \hat{g}_k$$

end for

TRPO

Natural PG (NPG) is a major ingredient of TRPO.

- NPG
- Monotonic improvement theorem. [post link]
- Line search of parameter.

- Pseudocode:

for iteration=$1, 2, \ldots$ **do**
Run policy for T timesteps or N trajectories
Estimate advantage function at all timesteps

$$\begin{aligned} &\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n \\ &\text{subject to} \quad \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta \end{aligned}$$

end for

NPG is a major ingredient of TRPO.

- NPG
- Monotonic improvement theorem. [post link]
- Line search of parameter.

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates) — NPG
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

  Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$
  Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\epsilon}{x_k^T \hat{H}_k x_k}} x_k$
  Perform backtracking line search with exponential decay to obtain final update — Line search

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for

Issues with TRPO.

- Calculation of fisher information matrix (second order) is expensive!
- Is there a cheaper first order method that could achieve similar performance?

PPO

Two types of PPO.

- Adaptive KL-penalty
- Clipped objective

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \hat{A}_t - \beta KL \left[\pi_{\theta_{old}} \| \pi_\theta \right] \right]$$

Objective

$$d = \hat{\mathbb{E}}_t \left[KL \left[\pi_{\theta_{old}} \| \pi_\theta \right] \right]$$

- If $d < d_{target}/1.5, \beta \leftarrow \beta/2$
- If $d < d_{target} \times 1.5, \beta \leftarrow \beta \times 2$

Need to update β to enforce KL constraint

Two types of PPO.

- Adaptive KL-penalty
- Clipped objective

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \hat{A}_t, clip \left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right]$$

Objective

The clipping version vs. the KL penalty version.

- Clipping version is easier to implement.
- Generally speaking, clipping is working almost as well as KL penalty version in practice.
- Strongly recommend reading the original PPO [paper].

DQN

Q-LEARNING

- $Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{k} \mid s,a\right]$
- $Q^{\pi}(s,a) = \mathbb{E}_{s',a',r}[R(s,a) + \gamma Q^{\pi}(s',a') \mid s,a]$
- $Q^{*}(s,a) = \mathbb{E}_{s',r}\left[R(s,a) + \max_{a'} Q^{*}(s',a') \mid s,a\right]$

- $\hat{Q}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} \hat{Q}(s',a')$
- $\hat{Q}\,(s,a) \leftarrow \hat{Q}(s,a) + \alpha[r(s,a) + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a)]$

Q-LEARNING: OFF POLICY

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using ~~policy derived from~~ Q (e.g., ε-greedy) Exploration
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$ Exploitation
    $S \leftarrow S'$
  until S is terminal

Q-LEARNING -> DQN

Pros

- Same principal but use function approximation
- Large scale problems, huge (Continuous) state space
- More flexible data(e.g. Images, signals)
- Stability boost: Replay buffer & Target network
- Efficiency boost: Output array of Qs

Issues

- Limited to discrete action space
- Bad for large action space

DQN IMPLEMENTATION

Algorithm 4 DQN

procedure DQN

Initialize network Q_ω *and* Q_{target} *as a clone of* Q_ω

Initialize replay buffer R *and burn in with trajectories followed by random policy*

repeat for E **training episodes:**

Initialise S_0

for $t = 0, 1, \ldots, T-1$:

$$a_t = \begin{cases} \arg\max_a Q_\omega(s_t, a) & \text{with probability } 1-\epsilon \\ \text{Random action} & \text{otherwise} \end{cases}$$

Take a_t *and observe* r_t, s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) *in* R

Sample minibatch of (s_i, a_i, r_i, s_{i+1})*with size* N *from* R

$$y_i = \begin{cases} r_i & s_{i+1} \text{ is terminal} \\ r_i + \gamma \max_a Q_{\text{target}}(s_i, a) & \text{otherwise} \end{cases}$$

$L(\omega) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - Q_\omega(s_i, a_i))^2$

Update Q_ω *using* $\texttt{Adam}\,(\nabla_\omega L(\omega))$

Replace Q_{target} *with current* Q_θ *if* $t \mod 50 = 0$

end procedure

Loss reduced from 2D data

ONLY update on ONE ACTION, NO CHANGE to other Qs

MODEL-BASED RL

IDEA

- Learns an environment/dynamics model/network
- Function Approximation: $P_\varphi : S \times A \rightarrow R \times S'$
- Model rollout: $S_t \rightarrow A_t | \pi_\theta(S_t) \rightarrow R_t | P_\varphi(S_t, A_t) \rightarrow S_{t+1} | P_\varphi(S_t, A_t) \rightarrow \cdots$
- (Probabilistic) Ensemble networks

MBRL VS MODEL FREE: INTUITION

- Psychology: Unconscious vs Conscious
- Biology: Habitual vs Goal-directed

MBRL VS MODEL FREE

- More sample efficient(Short term)
- Compounding error(Stochasticity & Environment variety)
- Challenge: State representation & Use of prior domain knowledge

- Legged robot application: MBRL to stand and Model free RL to walk(Surrounding obstacles)