CSC413/2516 Tutorial 11

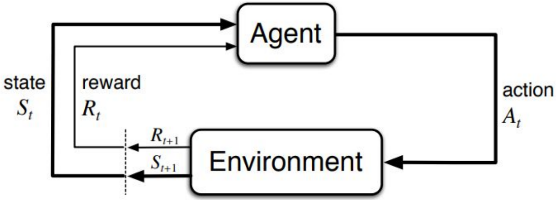
Reinforcement Learning, Policy Gradient

Based on Slides by Irene Zhang, Sheng Jia, Stephen Zhao

Winter, 2023

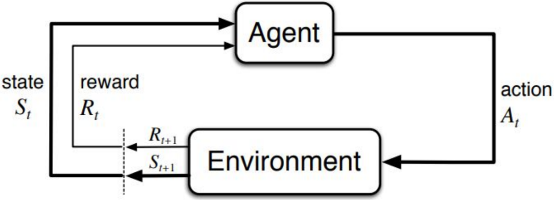
Problem Setup
State

- s_t: state at time step t. Is a complete description of the task/environment (assume full observability for simplicity in this tutorial), and is input to the agent
- a_t: action taken by the agent at time step t (output from the agent)



Problem Setup
State

- s_t: state at time step t. Is a complete description of the task/environment (assume full observability for simplicity in this tutorial), and is input to the agent
- a_t: action taken by the agent at time step t (output from the agent)



- Examples:
  - s_t = agent location on grid, a_t = movement direction
  - s_t = financial data, a_t = buy or sell
  - s_t = sequence of frames from a video, a_t = game action / robot movement

Problem Setup

Agent's Policy

- "Agent" is an abstract concept, but we can formulate how the agent behaves by a policy. This can be a conditional distribution that is parameterized by $\boldsymbol{\theta}$:

$$p_{\boldsymbol{\theta}}(a_t|s_t) = \pi_{\boldsymbol{\theta}}\left(a_t|s_t\right) = \pi\left(a_t|s_t;\boldsymbol{\theta}\right)$$

Problem Setup

Different implementations of a stochastic policy

$$p_{\boldsymbol{\theta}}(a_t|s_t) = \pi_{\boldsymbol{\theta}}\left(a_t|s_t\right) = \pi\left(a_t|s_t;\boldsymbol{\theta}\right)$$

Based on the problem, implement different types of stochastic policy

- $\mathcal{S}$ = state space (set of possible states)
- $\mathcal{A}$ = action space (set of possible actions)
- If both $\mathcal{S}$ and $\mathcal{A}$ are discrete and small, can simply use a table of mappings from states to probability distributions over actions

Problem Setup

Different implementations of a stochastic policy

$$p_{\boldsymbol{\theta}}(a_t|s_t) = \pi_{\boldsymbol{\theta}}\left(a_t|s_t\right) = \pi\left(a_t|s_t;\boldsymbol{\theta}\right)$$

Based on the problem, implement different types of stochastic policy

- $\mathcal{S}$ = state space (set of possible states)
- $\mathcal{A}$ = action space (set of possible actions)
- If both $\mathcal{S}$ and $\mathcal{A}$ are discrete and small, can simply use a table of mappings from states to probability distributions over actions
- If $\mathcal{A}$ is discrete, but $\mathcal{S}$ is continuous or too large (e.g. Atari), use a function approximator such as NN to map the state vector s to the distribution over actions using softmax for the output layer

Problem Setup

Different implementations of a stochastic policy

$$p_{\boldsymbol{\theta}}(a_t|s_t) = \pi_{\boldsymbol{\theta}}\left(a_t|s_t\right) = \pi\left(a_t|s_t;\boldsymbol{\theta}\right)$$

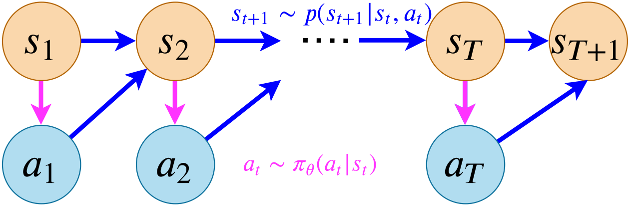
Based on the problem, implement different types of stochastic policy

- $\mathcal{S}$ = state space (set of possible states)
- $\mathcal{A}$ = action space (set of possible actions)
- If both $\mathcal{S}$ and $\mathcal{A}$ are discrete and small, can simply use a table of mappings from states to probability distributions over actions
- If $\mathcal{A}$ is discrete, but $\mathcal{S}$ is continuous or too large (e.g. Atari), use a function approximator such as NN to map the state vector s to the distribution over actions using softmax for the output layer
- If both $\mathcal{S}$ and $\mathcal{A}$ are continuous or too large (e.g. Robot control), map s to parameters associated with distributions such as μ and σ^2 for Gaussian distribution. Then sample actions from the distribution (A simpler solution is to discretize continuous action space. e.g. OpenAI Dota2 bot [1])

- $\boldsymbol{\tau}$ = trajectory, a record of states and actions over T time steps
- Trajectory is a set of random variables, and its distribution is a joint distribution over $2T+1$ r.v.:

$$\boldsymbol{\tau} = (s_1, a_1, s_2, ..., s_T, a_T, s_{T+1})$$

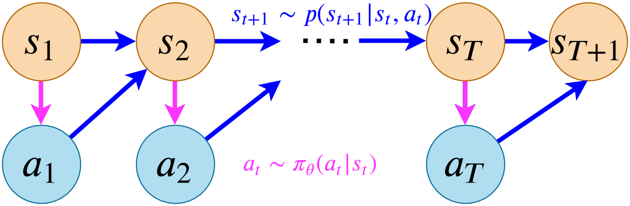
$$p(\boldsymbol{\tau}; \boldsymbol{\theta}) = p\left(s_1, a_1, s_2, ..., s_T, a_T, s_{T+1}; \boldsymbol{\theta}\right) = (\star)$$



- We can simplify using conditional independences from DAG (Markov assumption, state is a complete description):

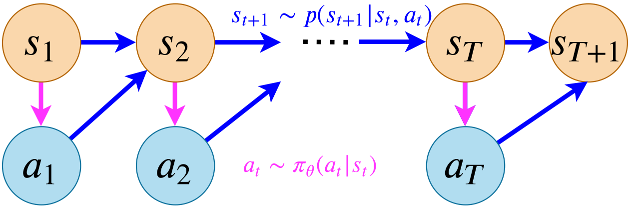
$$(\star) = \rho_0(s_1)\Pi_{t=1}^{T}\pi_{\boldsymbol{\theta}}(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

- Remark: we will use $p(\boldsymbol{\tau}; \boldsymbol{\theta})$ to denote that changing our policy parameters $\boldsymbol{\theta}$ induce a different trajectory distribution



- "Running/Executing the agent in a environment" means ancestral sampling from this DAG. (Sample the parent node and successively sample the child nodes)

$$s_1 \sim \rho_0(s) \quad a_t \sim \pi_\theta(a_t|s_t) \quad s_{t+1} \sim p(s_{t+1}|s_t, a_t)$$



Objective in Reinforcement Learning

Reward, Return

- Reward $r_t = R(s_t, a_t)$ measures how well action a_t is in state s_t for the agent. This is computed by a blackbox function $R(s_t, a_t)$ from the environment

Objective in Reinforcement Learning

Reward, Return

- Reward $r_t = R(s_t, a_t)$ measures how well action a_t is in state s_t for the agent. This is computed by a blackbox function $R(s_t, a_t)$ from the environment
- Return is the cumulative reward for the trajectory τ. (Consider finite-horzion undiscounted version in this tutorial)

$$R(\tau) = \sum_{t=1}^{T} R(s_t, a_t)$$

- Reward $r_t = R(s_t, a_t)$ measures how well action a_t is in state s_t for the agent. This is computed by a blackbox function $R(s_t, a_t)$ from the environment
- Return is the cumulative reward for the trajectory τ. (Consider finite-horzion undiscounted version in this tutorial)

$$R(\tau) = \sum_{t=1}^{T} R(s_t, a_t)$$

Return is also a random variable because it is a function of $2T$ random variables in the trajectory

Objective in Reinforcement Learning

Expected Return

- As $R(\tau)$ is random, the objective is to maximize the expected return $\mathbb{E}\left[R(\tau)\right]$ w.r.t $\boldsymbol{\theta}$. By the law of the unconscious statistician, we can write it as the expectation under τ distribution $p(\tau; \boldsymbol{\theta})$:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}\left[R(\boldsymbol{\tau})\right] = \mathbb{E}_{\boldsymbol{\tau} \sim p(\tau; \boldsymbol{\theta})}\left[R(\boldsymbol{\tau})\right] = (\star)$$

Objective in Reinforcement Learning

Expected Return

- As $R(\tau)$ is random, the objective is to maximize the expected return $\mathbb{E}[R(\tau)]$ w.r.t $\boldsymbol{\theta}$. By the law of the unconscious statistician, we can write it as the expectation under τ distribution $p(\tau; \boldsymbol{\theta})$:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}[R(\boldsymbol{\tau})] = \mathbb{E}_{\boldsymbol{\tau} \sim p(\tau;\boldsymbol{\theta})}[R(\boldsymbol{\tau})] = (\star)$$

And by ancestral sampling, we can further simplify:

$$(\star) = \mathop{\mathbb{E}}_{\substack{s_1 \sim \rho_0(s) \\ a_t \sim \pi_\theta(a_t|s_t) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t)}} \left[\sum_{t=1}^{T} R(s_t, a_t) \right]$$

State Value Function

- State Value Function $V^{\pi}(\mathbf{s})$ of a state $\mathbf{s}$ under policy π : the expected discounted return if we start in s and follow π

$$V^{\pi}(\mathbf{s}) = \mathbb{E}\left[G_t \mid \mathbf{s}_t = \mathbf{s}\right] = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i} \mid \mathbf{s}_t = \mathbf{s}\right]$$

- Computing the value function is generally impractical because we do not have the model of the environment

$$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma V\left(s'\right)\right]$$

- The benefit is credit assignment: see directly how an action affects future returns rather than wait for rollouts

Action-State Value Function

- Action-State Value Function $Q^\pi(\mathbf{s}, \mathbf{a})$ of a state $\mathbf{s}$ and action $\mathbf{a}$ under policy π is the expected discounted return if we start in s, take action a and then follow π

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}\left[G_t \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]$$

- Relationship:

$$V^\pi(\mathbf{s}) = \sum_{\mathbf{a}} \pi(\mathbf{a} \mid \mathbf{s}) Q^\pi(\mathbf{s}, \mathbf{a})$$

- Optimal action:

$$\arg\max Q^\pi(\mathbf{s}, \mathbf{a})$$

Optimal Bellman Equation

- The optimal policy π^* is the one that maximizes the expected discounted return, and the optimal action-value function Q^* is the action-value function for π^*.
- The Optimal Bellman Equation gives a recursive formula for Q^* :

$$Q^*(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})} \left[\max_{\mathbf{a}'} Q^* \left(\mathbf{s}_{t+1}, \mathbf{a}' \right) \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a} \right]$$

- This system of equations characterizes the optimal action-value function. So maybe we can approximate Q^* by trying to solve the optimal Bellman equation!

Q-learning: Off-policy TD learning

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 $Q(\mathit{terminal}, \cdot) = 0$

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)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

Policy Optimization by Policy Gradient Ascent

A method to "skill up" the agent

- Our goal: find the optimal policy $\theta^* = argmax_\theta \mathcal{J}(\theta)$

Policy Optimization by Policy Gradient Ascent

We can make a one-step optimization for the current policy $\pi_{\boldsymbol{\theta}_k}(a_t|s_t)$ to $\pi_{\boldsymbol{\theta}_{k+1}}(a_t|s_t)$ for maximizing $\mathcal{J}(\boldsymbol{\theta})$ by gradient ascent:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k}$$

Policy Optimization by Policy Gradient Ascent

A method to "skill up" the agent

Policy Optimization by Policy Gradient Ascent

We can make a one-step optimization for the current policy $\pi_{\boldsymbol{\theta}_k}(a_t|s_t)$ to $\pi_{\boldsymbol{\theta}_{k+1}}(a_t|s_t)$ for maximizing $\mathcal{J}(\boldsymbol{\theta})$ by gradient ascent:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k}$$

Gradient of the objective w.r.t policy (Policy Gradient)

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k} = \mathop{\mathbb{E}}_{\substack{s_1 \sim \rho_0(s) \\ a_t \sim \pi_{\theta_k}(a_t|s_t) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t)}} \left[\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_k}(a_t|s_t) \left[\sum_{t'=1}^{T} R(s_{t'}, a_{t'}) \right] \right]$$

REINFORCE Algorithm

Putting the above together, we get the most simple policy gradient method, the REINFORCE algorithm:

1. Sample $\left\{\tau^{i}\right\}$ from $\pi_{\theta}\left(\mathbf{a}_{t} \mid \mathbf{s}_{t}\right)$ (run in the environment)
2. Compute the gradient estimate:
$\nabla_{\theta} \mathcal{J}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_k} \approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=1}^{T} \log \pi_{\boldsymbol{\theta}_k}(a_t^{(i)}|s_t^{(i)}) \left[\sum_{t'=1}^{T} R(s_{t'}^{(i)}, a_{t'}^{(i)})\right]\right]$
3. Update the policy via gradient ascent
4. Repeat the above

Deep Q-Network

1. Represent the state-action value function, Q with a deep neural network with parameters θ: $Q(s, a; \theta)$
2. Remember Optimal Bellman Equation:

$$Q^*(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{p(\mathbf{s}'|\mathbf{s},\mathbf{a})}\left[\max_{\mathbf{a}'} Q^*\left(\mathbf{s}_{t+1}, \mathbf{a}'\right) \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]$$

3. Forward Pass
Loss function: $L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s, a; \theta_i\right)\right)^2\right]$ where
$y_i = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) \mid s, a\right]$
4. Backward Pass
Gradient update (with respect to Q-function parameters θ):

$$\nabla_{\theta_i} L_i\left(\theta_i\right) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) - Q\left(s, a; \theta_i\right)\right)\nabla_{\theta_i} Q\left(s, a; \theta_i\right)\right]$$

Reference

[1] Christopher Berner et al. "Dota 2 with Large Scale Deep Reinforcement Learning". In: *arXiv preprint arXiv:1912.06680* (2019).