Policy-based methods/Policy gradient methods

Value-based methods

- assign a value to a state (value function) OR action given a state (action value function)
- policy is *derived* from these values
    - chose the action leading to the greatest return
    - $\argmax\act$ to implement the policy

Policy-based methods

- by contrast, construct the policy π directly
- as a parameterized (by parameters Θ) function

$$\pi_\theta(\actseq|\stateseq) = \prc\actseq_{=}\act\stateseq_{=}\stateseq$$

i.e., the policy is a probability distribution of actions $\act$, conditional on the state $\state$)

Here is a brief comparison of Value-based and Policy-based methods.

Aspect	**Value-Based Methods**	**Policy-Based Methods**
Output	State/action value functions	Directly parameterized policy
Policy Representation	Implicit (via greedy/exploratory actions)	Explicit (probability/distribution mapping)
Learning Objective	Value prediction loss minimization	Expected return maximization (gradient ascent)
Typical Example Algorithms	DQN, Q-learning, SARSA	REINFORCE, PPO, vanilla policy gradient
Action Space	Discrete (practical)	Handles continuous and discrete
Stochastic Policies	Limited	Natural/efficient
Exploration Strategies	Decoupled from policy (e.g. epsilon-greedy)	Inherent (stochastic policy outputs)

Policy-based methods are *necessary* it those cases in which Value-based methods are not possible:

- Continuous (versus discrete) action
    - the $\backslash\mathrm{argmax}\backslash\mathrm{act}$ that implements policy in Value based methods is not possible
- Stochastic policy necessary
    - games against an adversary: when an adversary can take advantage of Agent predictability

Policy-based methods are *desirable/preferable* when Value-based methods are impractical

- Large number of possible actions
- High dimensional state spaces
    - state is characterized by a (long) vector of characteristics

In both these cases:

- tables are impractical representations of Value function or Action value function

Scenario	Policy-Based Required	Policy-Based Desirable
Continuous action spaces	Yes	Yes
Stochastic strategies needed	Yes	Yes
Aliased or partially observable states	Yes	Yes
High-dimensional spaces	Sometimes	Yes
Discrete/simple environments	No	Sometimes

Policy Gradient methods

The predominant class of Policy based methods are those based on the Policy Gradient method.

Policy Gradient methods create a sequence of improving policies

$$\pi_0, \ldots, \pi_p, \ldots$$

by creating a sequence of improved parameter estimates

$$\theta_0, \ldots, \theta_p, \ldots$$

using Gradient Ascent on some objective function $J(\theta)$ to improve θ_p

$$\theta_{p+1} = \theta_p + \alpha * \nabla_\theta J(\theta_p)$$

- gradient of a Performance Measure $J(\theta)$
- with respect to parameters Θ

There a a few Policy base methods that *don't* use Policy Gradient

- in the module on Value based methods, we learned about Policy Iteration
- Policy iteration alternates
    - Policy Evaluation: improving the estimate of a Value function
    - Policy Improvement: improving the policy
        - use $\backslash\mathrm{argmax}\backslash\mathrm{act}$ to implement the current policy

Thus, Policy Iteration is both Value based and Policy based

- but does not evolve policy via gradients

Since we are trying to maximize objective function $J(\theta)$ rather than minimize a loss objective

- we use Gradient Ascent rather than Gradient Descent
- hence we add the gradient rather than subtract it, in the update

RL Book Chapt 12 (http://incompleteideas.net/book/RLbook2020.pdf#page=343)

Stochastic policy and environment

With Value based methods

- the Environment can be stochastic
- but the Policy must be deterministic
    - $\argmax\act$ to implement the policy

With Policy Gradient methods

- the policy can be stochastic (action is a probability distribution)
  $\pi(\act|\state;\theta) = \pr\actseq_{=}\act \,|\, \stateseq_{=}\state\,,\, \theta_{=}\theta$

The environment can *also* be stochastic

$$\transp(\state', \rew | \state, \act)$$

$$= \transp(\stateseq_{=} \state', \rewseq_{=} \rew \mid \stateseq_{-1} = \state, \actseq_{-1} =$$

- the response $(\state', \rew)$ by the environment is not deterministic

This poses a challenge to Value-based methods

- a single observation of $(\state', \rew)$ is a *high variance* estimate of $\transp(\state', \rew | \state, \act)$

Objective function

Recall that the return G_\tt of a single episode is the expected value of rewards accumulated starting in the state of step of the episode

```
\begin{array}\\
G_\tt & = & \sum_{k=0}^\tt {  \gamma^k * \rewseq_{\tt+k+1} } \\
    & = & \rew_{\tt+1}  + \gamma * G_{\tt+1} \\
\end{array}
```

The performance measure $J(\theta)$ that we define will be the *expected value* (across all possible episodes) of the return G_0 from initial state $\stateseq_0$ of the episode

$$J(\theta) = \Exp\tau(G_{0,\tau})$$

- using the notation

  $$G_{,\tau}$$

  to denote the return within episode τ of step of the episode.

Note that $G_{,\tau}$ is equivalent to $\statevalfun_\pi(\stateseq_)$ (relative to episode τ)

- the value function evaluated on the initial state

Taking the gradient of the Objective

This objective function presents some challenges in computing $\nabla_\theta J(\theta_p)$

The first is: how to take gradient of an expectation ?

We can do away with the expectation by replacing it with the sum

$$\backslash \mathrm{Exp}\tau(G_{,\tau}) = \sum_{\tau} \backslash \mathrm{pr}\tau; \theta * G_{,\tau}$$

where

$$\backslash \mathrm{pr}\tau; \theta$$

is the probability of episode τ.

In practical terms

- we don't sum over every possible episode
- we can approximate the Expectation through *trajectory sampling*
    - accumulate a batch of episodes
    - approximate the expectation as the average across the episodes in the batch

Note that the gradient of a sum is equal to the sum of the gradients

- so being able to compute the gradient of $J(\theta)$ depends on being able to compute the terms in the sum.

But this too presents a challenge

The probability of episode τ occurring is thus the product of each step occurring $$ \begin{array}{l} \pr{\tau; \theta} & = & \prod_{t=0}{ \transp({ \state', \rew | \state = \stateseq}\tt, \act = \actseq_\tt })

 * \pi(\actseq_\tt | \stateseq_\tt)
  }

\end{array} $$

The problem is the Transition Probability term in the product

$$\transp({ \state', \rew | \state = \stateseq_\tt, \act = \actseq_\tt })$$

- the reaction of the Environment to the agent choosing action $\actseq_\tt$ in state $\stateseq_\tt$
- is controlled by the environment
- generally: unknown
- *Model free* method

Policy Gradient Theorem

Our objective is to maximize the *expected return*

- where the expectation is over *all* possible trajectories τ generated by θ:

$$\tau \sim \prθ$$
$$J(\theta) = \Exp\tau \sim \prθ\ G_{\tau,0} = \Exp\tau \sim \prθ\ \rewseq(\tau)$$

where

- $\rewseq(\tau) = G_{\tau,0}$

is the return from the initial state 0 of a trajectory τ

Notes

- We assume the discount factor $\gamma = 1$ only to simplify notation
- To clarify that states, actions, rewards, returns, etc. depend on the specific trajectory τ
    - we add an extra subscript when necessary for clarification

$$\rewseq_{\tau,}$$

Using Gradient Ascent to solve the maximization, we need to compute

$$\nabla_\theta J(\theta)$$

The *Policy Gradient Theorem* shows us how to compute this gradient.

We can state it in two forms.

In the first formulation

- the reward is stated as a reward $\backslash\mathrm{rewseq}(\tau)$ for the entire *trajectory* τ

$$\nabla_\theta J(\theta) = \backslash\mathrm{Exp}\tau \sim \backslash\mathrm{pr}\theta \sum_{=0}^{|\tau|} \nabla_\theta \log \pi(\backslash\mathrm{actseq}_{\tau,} | \backslash\mathrm{stateseq}_{\tau,}))\ \backslash\mathrm{rewseq}(\tau)$$

Each step receives the *trajectory reward* $\backslash\mathrm{rewseq}(\tau)$.

This works out well

- in the case where there is a single reward *received at the end* of the trajectory
    - all steps assigns equal "credit"

The second formulation credits a reward for each step of trajectory τ

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{=0}^{T-1} \nabla_\theta \log \pi_\theta(\backslash\mathrm{actseq}_{\tau,} | \backslash\mathrm{stateseq}_{\tau,t}) G_{\tau,} \right]$$

where $G_{\tau,}$ is the return-to-go of the trajectory τ from step (state $\backslash\mathrm{stateseq}_{\tau,}$).

In the second formulation (which is algebraically equivalent to the first) step receives the *return to go* $G_{\tau,}$.

This works out well

- when there are intermediate rewards
    - each step is rewarded in proportion to the future return attributed to the step's action
    - reduces variance of the gradient estimate by assigning precise credit

Note

You may sometimes see

$$G_{\tau,}$$

written as a Q-function

$$Q(\backslash \mathrm{actseq}_{1\tau,} | \backslash \mathrm{stateseq}_{\tau,t})$$

Whichever way we write it

- the Policy Gradient Theorem is the foundation for all policy-based methods
- it tells us how to change the parameters θ of the Policy NN in a direction leading to optimality

We will study a few of these methods in a later section.

Computing $\nabla_\theta J(\theta)$

Our first step will be to turn the expectation

$$\Exp\tau \sim \pr\theta \ \rewseq(\tau)$$

into a sum

$$\sum_{\tau\sim\pr\theta} \pr\tau * \rewseq(\tau)$$

where

$$\pr\tau$$

is the probability of trajectory τ

With stochastic policy and Environment and trajectory τ

$$\tau = \stateseq_{\tau,0}, \actseq_{\tau,0}, \rewseq_{\tau,1}, \stateseq_{\tau,1} \ldots \stateseq_{\tau,}, \actseq_{\tau,}, \rewseq_{\tau,+1}, \stateseq_{\tau,+1}, \ldots$$

we can compute $\pr\tau$

$$\pr\tau = \pr\stateseq_{\tau,0} * \prod_{=0}^{|\tau|} \pi(\actseq_{\tau,}|\stateseq_{\tau,}) * \transp(\stateseq_{\tau,+1}, \rewseq_{\tau,+1}|\stateseq_{\tau,}, \actseq_{\tau,})$$

- as the chained (multiplicative) probability of each step
- where the probability of each step is a product of
  - the probability
    $$\pi(\actseq_{\tau,}|\stateseq_{\tau,})$$
    that the agent choses $\actseq_{\tau,}$ as the action
  - the probability
    $$\transp(\stateseq_{\tau,+1}, \rewseq_{\tau,+1}|\stateseq_{\tau,}, \actseq_{\tau,})$$
    responds by changing the state to $\stateseq_{\tau,+1}$ and giving reward $\rewseq_{\tau,+1}$

Taking the gradient

We have $\pr\tau$ computed as chained probability above.

Taking the log of a product gives the sum of logs

$$\log \pr = \log \left(\pr\stateseq_{\tau,0} * \right.$$

$$\prod_{=0}^{|\tau|} \pi(\actseq_{\tau,}|\stateseq_{\tau,}) * \transp(\stateseq_{\tau,+1}, \rewseq_{\tau,+1}|$$
$$\actseq_{\tau,})$$

$$= \log \pr\stateseq_{\tau,0} +$$
$$\sum_{=0}^{|\tau|} \left(\log \pi(\actseq_{\tau,}|\stateseq_{\tau,}) + \log \right.$$
$$\transp(\stateseq_{\tau,+1}, \rewseq_{\tau,+1}|\stateseq_{\tau,}, \actse$$

So

Thus we can rewrite the final expectation

$$\begin{aligned}\Exp\tau \sim \pr\theta \nabla_\theta \log \pi(\actseq_{\tau,}|\stateseq_{\tau,}))\ \rewseq(\tau) &= \Exp\tau \sim \pr\theta \\ &= \Exp\tau \sim \pr\theta\end{aligned}$$

Notes on Proof of Policy Gradient theorem

Likelihood ratio trick

The likelihood ratio trick states that

- for a parameterized probability distribution $p_\theta(x)$
- and a function $f(x)$:

the gradient of an expectation can be converted into an expectation over the gradient.

$$
\begin{aligned}
\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] &= \nabla_\theta \int p_\theta(x) f(x) dx && \text{convert expectation to int} \\
&= \int \nabla_\theta p_\theta(x) f(x) dx && \text{move grad inside the integ} \\
&= \int f(x) \nabla_\theta p_\theta(x) dx && \text{rearrange term} \\
&= \int f(x) \left(p_\theta(x) \nabla_\theta \log p_\theta(x)\right) dx && \text{log trick:} \\
& && \nabla_\theta p_\theta(x) = p_\theta(x) \nabla_\theta \log p_\theta(\\
&= \mathbb{E}_{x\sim p_\theta}\left[f(x) \nabla_\theta \log p_\theta(x)\right] && \text{convert integral back to ex}
\end{aligned}
$$

The "log trick" follows from the rules of calculus

Why substituting $\rewseq_\tau$, for $G_{\tau,0}$ is algebraically the same

Consider the two expressions from the policy gradient theorem:

$$\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot R(\tau) \quad \text{and} \quad \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_t$$

where:

- $R(\tau) = G_0 = \sum_{k=0}^{T-1} \gamma^k r_k$ is the total discounted return of the entire trajectory,
- $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_k$ is the return-to-go starting at step t.

- Step 1: Start with the total return $R(\tau) = G_0$

$$L = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot G_0 = G_0 \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Step 2: Expand G_0 as the sum over rewards

$$G_0 = \sum_{k=0}^{T-1} \gamma^k r_k$$

Substitute into L:

$$L = \left(\sum_{k=0}^{T-1} \gamma^k r_k\right) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)$$

- Step 3: Express as a double sum

$$L = \sum_{t=0}^{T-1} \sum_{k=0}^{T-1} \gamma^k r_k \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Step 4: Separate sums over past and future rewards relative to t

$$L = \sum_{t=0}^{T-1} \left(\sum_{k=0}^{t-1} \gamma^k r_k + \sum_{k=t}^{T-1} \gamma^k r_k\right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Step 5: Rewrite future rewards shifted by t

Define $j = k - t$:

$$\sum_{k=t}^{T-1} \gamma^k r_k = \gamma^t \sum_{j=0}^{T-1-t} \gamma^j r_{t+j} = \gamma^t G_t$$

- Step 6: Substitute back into L

$$L = \sum_{t=0}^{T-1} \left(\sum_{k=0}^{t-1} \gamma^k r_k + \gamma^t G_t\right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- Step 7: Expectation zeroes out past rewards term

Because rewards before time t do not depend on action a_t, their expected contribution is zero:

$$\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \sum_{k=0}^{t-1} \gamma^k r_k\right] = 0$$

- Step 8: Final form of the policy gradient

Thus,

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \gamma^t G_t\right]$$

With the standard definition of G_t absorbing the discount

Alternate Proof of the Policy Gradient Theorem (Per-Step Reward Perspective)

Let $J(\theta)$ be the expected discounted sum of rewards under policy π_θ:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]$$

- Step 1: Expand the Expectation

Rewrite the expectation explicitly:

$$J(\theta) = \sum_\tau P_\theta(\tau) \left(\sum_{t=0}^{T-1} \gamma^t r_t \right)$$

- Step 2: Differentiation w.r.t. θ

$$\nabla_\theta J(\theta) = \sum_\tau \nabla_\theta P_\theta(\tau) \left(\sum_{t=0}^{T-1} \gamma^t r_t \right)$$

Apply the **likelihood ratio trick**: $\nabla_\theta P_\theta(\tau) = P_\theta(\tau) \nabla_\theta \log P_\theta(\tau)$ So,
$\nabla_\theta J(\theta) = \sum_\tau P_\theta(\tau) \nabla_\theta \log P_\theta(\tau) \left(\sum_{t=0}^{T-1} \gamma^t r_t \right)$ Or equivalently,
$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\theta \log P_\theta(\tau) \left(\sum_{t=0}^{T-1} \gamma^t r_t \right) \right]$

- Step 3: Break Down $\log P_\theta(\tau)$

Recall, $\log P_\theta(\tau) = \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \text{terms independent of } \theta$ So,
$\nabla_\theta \log P_\theta(\tau) = \sum_{t'=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'})$ Substitute:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \cdot \left(\sum_{t=0}^{T-1} \gamma^t r_t\right)\right]$$

- Step 4: Swap Order of Summation

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T-1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'})\gamma^t r_t\right]$$

Switch the order:

$$= \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \sum_{t'=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'})\gamma^t r_t\right]$$

- Step 5: Analyze the causal relationship

The gradient w.r.t. $a_{t'}$ can only affect rewards *from* t' onward (not earlier rewards due to the Markov property), so for $t < t'$ the expectation is zero.

Thus, the only contributing terms are where $t' \leq t$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t'=0}^{T-1} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \left(\sum_{t=t'}^{T-1} \gamma^t r_t\right)\right]$$

- Step 6: Recognize the return-to-go term

$$\sum_{t=t'}^{T-1} \gamma^t r_t = \gamma^{t'} \sum_{j=0}^{T-1-t'} \gamma^j r_{t'+j} = \gamma^{t'} G_{t'}$$

So, we may write:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \gamma^t G_t\right]$$ Often, γ^t is absorbed into the definition of G_t.

- Step 7: Final form (policy gradient theorem)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right]$$

Conclusion:
By starting with the expectation of per-step rewards and applying the likelihood ratio trick, we arrive at the same policy gradient theorem:
each action's gradient is weighted by the return-to-go from that time (not just the immediate reward).

Comparison of Policy-Based Reinforcement Learning Methods

Method	Gradient-Based	Main Objective	Key Characteristics	Stability & Sample Efficiency	Typical Application Domains
PPO (Proximal Policy Optimization)	Yes	Maximize expected reward with clipped surrogate objective	Uses policy gradients with clipping to limit policy update magnitude, balancing exploration and exploitation	High stability; more sample efficient than vanilla policy gradients; widely used for continuous and discrete control tasks	Robotics, games, continuous control, benchmark RL tasks
DPO (Direct Preference Optimization)	Yes	Directly optimize policy based on preference data	Uses a preference-based loss to train policy without explicit reward modeling; bypasses traditional RL complexities	Improved stability by leveraging human preferences; avoids some issues of reward misspecification	Alignment of language models with human preferences, NLP-focused RL
GRPO (Group Relative Policy Optimization)	Yes	Optimize policy using group-relative advantage estimates	Does not require a separate value function; updates policy based on relative advantages within a group of candidate outputs	More memory efficient, stable; effective for large-scale policy optimization with reduced critic reliance	Training large language models, large-scale policy optimization
REINFORCE	Yes	Maximize expected cumulative reward by direct policy gradient	Uses Monte Carlo sampled returns, applies likelihood ratio trick; pure policy gradient without value function	High variance and sample inefficient; simpler but less stable than actor-critic or PPO	Fundamental policy gradient algorithm, baseline for many RL studies

Actor-Critic

Value-based methods learn a function approximation of the *value* of a state or a state/action pair.

- policy is chosen based on the value of successor states

Simple Policy-based methods learn a parameterized policy function.

- using a NN to learn the policy
- using an objective function $J(\theta)$ that depends on an approximation of either
  - the value $\statevalfun(\state)$ or G_\tt
  - or action/value function $\actvalfun(\state, \act)$

Actor-Critic-Policy-based methods used Neural Networks to learn

- *both* the value function and policy function approximations
- the agent is called the *Actor*
- the NN providing estimates of G_t or $\actvalfun(\state, \act)$ is called the *Critic*

Notice that, in the REINFORCE algorithm, $G_{,\tau}$ is computed for *each trajectory* τ independently

- there is no memory of the prior stochastic response

  $$\transp(\state', \rew | \state, \act)$$

  $$= \transp(\stateseq_{=} \state', \rewseq_{=} \rew \mid \stateseq_{-1} = \state, \acts$$

  $$= \act$$

  for the same state $\state$ and action $\act$ of a previous episode
  - this leads to high variance estimates of $G_{,\tau}$

By using a NN to estimate G_\tt

- our estimate includes multiple examples of the stochastic response to action $\act$ in state $\state$
- hopefully leading to a lower variance approximation

- RL tips and tricks (https://stable-baselines3.readthedocs.io/en/master/guide/rl_tips.html)
- RL book contents (http://incompleteideas.net/book/RLbook2020.pdf#page=7)
- RL book notation (http://incompleteideas.net/book/RLbook2020.pdf#page=20)

In [2]:
```
print("Done")
```

```
Done
```