Policy gradient/ search is influential in NLP/ Proximal Policy Optimization (training GPT). The core idea:

1 Why Policy Gradient, not Value Based?

Intuition of Gradient Search

Approximate $V^\pi(s) \approx V_W(s)$ and $Q_w(s,a) \approx Q^\pi(s,a)$ by adjusting weight w.
Policy gradient: rather than generating policy from value (ϵ-greedy), directly parametrize policy with θ, i.e.

$$\pi_\theta(s,a) = \mathbb{P}[a|s;\theta]\text{: optimize } V(\theta) \text{ to find policy } \pi$$

The brief classification of policy gradient is as follows:

	Value-based	Policy-based	Actor-critic
Value function	learned	not present	learned
Policy	implicit (ϵ-greedy)	learned	learned

Instead of deterministic/ ϵ-greedy policies, need to focus heavily on **stochastic** for direct policy search!

- Repeated Trials, e.g. In rock paper scissors (of many rounds), deterministic policy is easily exploited by adversary.
- Boundary Condition, e.g. In gridworld, bound to only move one direction (else get stuck/ traverse for long time for slow convergence).

1.1 Gradient Free Policy Optimization

We begin with simple (but great) gradient-free baselines.

- Examples: Hill Climbing, Genetic Algo (evolution strategies, cross-entropy method, covariance matrix adaption)
- Known for decades but embarrassingly well: rivals standard RL techniques!
- Advantages: Flexible for any policy parameterization, easily to parallelize
  Disadvantage: Less sample efficient (ignores temporal structure)

2 Main Objective and Log-likelihood Trick for Policy Gradient

2.1 Policy Gradient

This section focuses on gradient descent; other popular algos include conjugate gradient and quasi-newton methods.
We assume **Episodic MDPs** for easy extension of objectives. We first outline the problem as follows:

Policy Objective Summary

- Goal: Given policy $\pi_\theta(s,a)$, find best parameter θ. Inherently, an optimization of $V(s_0,\theta)$ (i.e. the value fucntion depending on policy parameters).
- Purpose: Measure quality for policy π_θ with policy value at start state s_0.
- Works for: both episodic/ continuing and infinite horizons.

The method:

Vanilla Policy Gradient: Problem Formation

- Search the local maximum of policy value $V(s_0, \theta)$ with gradient increments:

$$\Delta\theta = \alpha \nabla_\theta V(s_0, \theta) = \alpha \begin{pmatrix} \frac{\partial V(s_0,\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial V(s_0,\theta)}{\partial \theta_n} \end{pmatrix}$$

- Assumption: π_θ differentiable (and known gradient $\nabla_\theta \pi_\theta(s, a)$)
- We can rewrite $V(s_0, \theta)$ in the following ways:
  1. **Visited States and Actions**: $\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t); \pi_\theta, s_0\right]$
  2. **Weighted Average of Q-values by Actions**: $\sum_a \pi_\theta(a|s_0) Q(s_0, a, \theta)$
  3. **Trajectories Sampled using π_θ**: $\sum_\tau P(\tau|\theta) R(\tau)$

2.2 Log-Likelihood Trick and Score

In particular, it is of interest to consider writing $V(s_0, \theta)$ in trajectory form:

To find the best policy parameter θ, we consider

$$\arg\max_\theta V(\theta) = \arg\max_\theta \sum_\tau P(\tau; \theta) R(\tau)$$

Taking gradient,

$$\begin{aligned} \nabla_\theta V(\theta) =& \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \\ & \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) (R \text{ being indep of } \theta) \\ & \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \\ & \sum_\tau R(\tau) P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) \quad \text{(log-likelihood)} \end{aligned}$$

Approximate in practice using m sample trajectories under π_θ:

$$\nabla_\theta V(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)}, \theta)$$

But trajectories can be decomposed into states and actions:

$$\begin{aligned} \nabla_\theta \log P(\tau^{(i)}; \theta) =& \nabla_\theta \log \left[\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}|a_{t+1}, s_{0:t}, a_{0:t}) \right] \\ =& \sum_\tau \nabla_\theta \log \pi_\theta(a_t|s_t) \end{aligned}$$

Here

- We call $\sum_\tau \nabla_\theta \log \pi_\theta(a_t|s_t)$ the **score function**.
- the initial state $\mu(s_0)$ is constant; dynamics model $P(s_{t+1}|a_{t+1}, s_{0:t}, a_{0:t})$ is invariant to θ.
- In other words, no dynamics model is required to approximate the policy parameter θ.

Questions

1. Why trajectory form is practical ("better in training")?

   Two major reasons:
   (a) **Flexible**
   "Black box access" that only requires generated trajectory rollouts without differentiable envt. model.
   Allows turning $\nabla_\theta P(\tau)$ into $P(\tau)\nabla_\theta \log P(\tau)$, i.e. unbiased gradient estimate with simplicity.
   (b) **Easy Implementation**
   Transition probabilities $P(s_{t+1}|s_t, a_t)$ don't appear in gradient after log derivative.
   Only actions, episodes, states and rewards (not P) to compute gradient!

2. Why is log-likelihood ratio important here? What does it enable?

   Log trick enables PG without environment back-propagation.
   Without model **P** required, log likelihood enables additive (instead of multiplicative) decomposition.

2.3 Selecting a Right Policy

2.3.1 Softmax Policy

- In softmax, **exponentially weight** quantities of linear combination of features as probabilities (that add to 1):

$$\pi_\theta(s, a) = \frac{e^{\phi(s,a)^T\theta}}{\sum_a e^{\phi(s,a)^T\theta}}$$

- Then the score function can be written as $\nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - \mathbb{E}_{\pi_\theta[\phi(s,\cdot)]}$

2.3.2 Gaussian Policy

- A normal distribution is natural for continuous action spaces; at times used by deep NN.
- Action $a \sim N(\mu(s), \sigma^2)$. Mean $\mu(s) = \phi(s)^T\theta$ is a linear combination of state features.
- Then the score function can be written as $\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$

What are the purposes of selecting Softmax and Gaussian policies?

Policy classes should allow (1) Easy action sampling and (2) Straightforward gradient $\nabla_\theta \log \pi_\theta(s, a)$ computation.
The two policies are both differentiable, allowing (2). For (1):

- **Softmax**: **Discrete** action probabilities that returns a nice (simple) gradient form.
- **Gaussian**: **Continuous** actions featuring straightforward gradient, common in robotics/ continuous control.

2.4 Summary: PG

A summary of policy gradient:

Intermediate Summary of PG

- Core idea: $\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$

Optimize $\arg\max_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t r_t\right]$ with SGD on θ:

$$g = \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{\infty}\gamma^t \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t,a_t)\right]$$

- State-action pairs with higher $\hat{Q}$ increases probabilities in average
- Direction of θ dependent on gradient of $\ln \pi(S_t, A_t, \theta)$ AND Q-values/ returns
- NOT guaranteed to converge to global optima (just local!)

When to use?

1. Differentiable reward functions; No dynamics required
2. Useful for both infinite horizon and episodic settings
3. Intuition: $R(\tau^{(i)})$ is replacable by other functions that *measures the wellness of sample* x
   Essentially, moving in the direction of $\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$ pushes up the log probability proportionally.

The generalization of PG is as follows:

Policy Gradient Theorem

Assumption: Differentiable Policy $\pi_\theta(s,a)$, objective $J = \begin{cases} J_1 & \text{(Episodic)} \\ J_{avR} & \text{(Avg. Reward over time)} \\ \frac{1}{1-\gamma} J_{avV} & \text{(Avg. Value over time)} \end{cases}$ In any case, the policy gradient is $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$

Summary and Improvements of Policy-based RL

Criteria	Advantages	Disadvantages
Convergence	Better Properties	Typical Local Optimum
Flexibility	Effective in high-dim./ continuous action spaces Can learn stochastic policies	Inefficient, high-variance policy evaluation

3 Variance Issues and Remedies

3.1 Problem: High Variance in PG

Currently, use

$$\nabla_\theta V(\theta) = \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^{(i)}, s_t^{(i)})$$

to estimate

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

It is unbiased but noisy (high variance)!

On the high variance, several remedies serve as improvements:

3.2 Fix 1: Temporal Structure

- Focus on single reward item at once:

$$\nabla_\theta \mathbb{E}[r_{t'}] = \mathbb{E}[r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)]$$

- Sum up over t to obtain:

$$\begin{aligned} V(\theta) = \nabla_\theta \mathbb{E}[R] &= \mathbb{E}[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)] \\ &= \mathbb{E}[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) \sum_{t'=t}^{T-1} r_{t'}] \quad \text{(because later decisions don't influence past rewards} \end{aligned}$$

- This can be further simplified: trajectory $\tau^{(i)}$ has return $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}^{(i)}$. Hence

$$\nabla_\theta \mathbb{E}[R] \approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t, s_t) G_t^{(i)}$$

Monte-Carlo Policy Gradient

Making use of likelihood ratio / score function and temporal structure, update param θ with

$$\Delta\theta_t = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$$

after initializing θ arbitrarily.

Q: How does Temporal structure reduce variance?

- Each $r_{t'}$ is paired with actions ONLY from time 0 to t'.
- Actions after t' does NOT affect $r_{t'}$:
  Assigning rewards to ONLY actions that influence it achieves noise reduction.

3.3 Fix 2: Baseline Function

As iteration costs time and computational resources, we desire quick convergence to local optima.

Baselines: Unbiasedness and Other Considerations

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau \left[\left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

Why it works?

- Unbiased for any b if b is a function of s but not θ, because
  $\mathbb{E}_\tau \left[\nabla_\theta \log \pi(a_t|s_t;\theta) b(s_t) \right] = 0$
- A near-optimal baseline choice is the expected return $b(s_t) \approx \mathbb{E}\left[\sum_{t'=t}^{T-1} r_{t'} \right]$
- Other choices: State-value function $V^\pi(s) = \mathbb{E}_{a\sim\pi}\left[Q^\pi(s,a) \right]$

Interpretation: Increase logprob of action at proportionally to how much returns $\sum_{t'=t}^{T-1} r_{t'}$ are better than expected.

Mathematically, baseline functions achieve variance reduction as follows:

Core idea: Break down t in the summation:

- As we sample from trajectories τ (MC),

$$\begin{aligned}\text{Var}[\nabla_\theta[R]] &= \text{Var}[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t;\theta)(R_t(s_t) - b(s_t))] \\ &\approx \sum_{t=0}^{T-1} \mathbb{E}_\tau \left[\text{Var}[\nabla_\theta \log \pi(a_t|s_t;\theta)(R_t(s_t) - b(s_t))]\right]\end{aligned}$$

- For each t, write variance $\text{Var}[X]$ as $\mathbb{E}[X^2] - (\mathbb{E}[X])^2$:

$$\mathbb{E}\left[((\nabla_\theta \log \pi(a_t|s_t;\theta))(R_t(s_t) - b(s_t)))^2\right] - \mathbb{E}\left[((\nabla_\theta \log \pi(a_t|s_t;\theta))(R_t(s_t) - b(s_t)))\right]^2$$

- Second term is not affected by choice of $b(s)$ (unbiased = same expectation).
  The variance equals $\arg\min_b \quad \mathbb{E}\left[((\nabla_\theta \log \pi(a_t|s_t;\theta)))^2 ((G_t(s_t) - b(s_t)))^2\right]$

$$= \arg\min_b \quad \mathbb{E}_{s\sim d^\pi}\left[\mathbb{E}_{a\sim\pi(\cdot|s),G|s,a}\left[((\nabla_\theta \log \pi(a_t|s;\theta)))^2 ((G_t(s_t) - b(s_t)))^2\right]\right]$$

- A **weighted least-squares** problem that minimizes

$$\sum_i \sum_t |b(s_t^i) - G_t^i|^2 \qquad (G \text{ (or } A = G - b) \text{ being target})$$

  with solution (after taking zero gradient)

$$b(s) \approx \mathbb{E}_{a\sim\pi(\cdot|s),G|s,a}[G_t(s)]$$

Q: Intuitively, what are the uses of baseline functions? How does variance get reduced?

- Second term: If $b(s)$ is "close" to true $V^\pi(s)$ (high correlation between the two), $G_t(s_t) - b(s_t)$ achieves lower variance.
- First term: Log-probability only updates strongly (decisively, to increase variance) if significant return differences to baseline is observed.

Q: What does it mean by "Near optimal"? When is it not?

- According to above, when variance reduction of the second term > variance increase of first term.
  Usually the case as $b(s) \approx V^\pi(s)$ typically, achieving maximal variance reduction in theory.
- If learned baseline is inaccurate (i.e. poor function approximator/ predictor), less reduction.
  Extreme cases: Taking $b(s) =$ const or 0 could be better.

3.4 Fix 3: Actor-Critic

The original G_t^i estimates expected discounted sum of returns (from single roll): unbiased but high variance. To solve this:

- Leverage bootstrapping and approximation (similar to TD vs MC and VFA) to introduce bias.
- Use "Critic" to estimate the ratio $\frac{V}{Q}$. The popular class of "Actor-critic" methods explicitly represents (and updates) policy and values.
- Essentially, replace $\sum_{t'=t}^{T-1} r_{t'} - b(s_t)$ [Vanilla MC] with Q-values $(Q(s_t, a_t; \mathbf{w}) - b(s_t))$ [This is essentially TD]
  or advantage function $\hat{A}^\pi(s_t, a_t)$ where $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

Alternative Targets to MC Estimators

With vanilla MC, the gradient is estimated by

$$\nabla_\theta V(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T-1} R_t^i \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$$

Proposed replacements:

1. N-step estimators:
   - $\hat{R}_t^{(1)} = r_t + \gamma V(s_{t+1})$, $\hat{R}_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})$
   - $\hat{R}_t^{(\infty)} = r_t + \gamma r_{t+1} + \gamma r_{t+2} + \dots$
2. Advantage estimators, by *subtracting baselines of $V(s_t)$ from above*
   - $\hat{A}_t^{(1)} = \hat{R}_t^{(1)} - V(s_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$ *(low variance, high bias)*
   - $\hat{R}_t^{(\infty)} - V(s_t) = r_t + \gamma r_{t+1} + \gamma r_{t+2} + \dots - V(s_t)$ *(high variance, low bias)*

What do the words "actor" and "critic" mean?

- "actor": Policy π_θ selecting actions
- "critic": VFA (say, trained via temporal-difference) to "criticize" actions chosen by actor
- Proposed $Q^\pi(s,a)$ or $A^\pi(s,a)$ serve as baselines/ targets for actor updates to reduce variance.

4 PPO and Its Two Variants

4.1 Problem: Poor Sample Efficiency of PG

A PG algo should minimize # iterations to reach a good (probably suboptimal) policy within time. The limitations of vanilla PG:

4.1.1 Poor sample efficiency

Variance reduces slowly (even after improvements), because PG is an **on-policy** expectation: Data immediately discarded after just one gradient step

- Collect sample estimates from trajectories of same policy (more stable), or other policies (off-policy, less stable).
- Opportunity: Can we take multiple gradient steps from old data before new policy?

Problems of Determining Gradient Step

Problem: Difficult to handle step size (dist. in parameter space $\neq$ dist. in policy space)

- e.g. Matrices in tabular case $\Pi = \{\pi : \pi \in \mathbb{R}^{|S|\times|A|}, \sum_a \pi_{s_a} = 1, \pi_{s_a} \geq 0\}$
  VS steps of policy gradient in parameter space $\implies$ unable to map/ gauge size!
- SGD of $\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k$ is subject to performance collapse with large steps!
  e.g. logistic function: small $\Delta\theta$ leads to big policy changes

4.2 Policy Performance Bounds

The solution to restrict policy change from more than intended is through policy performance bounds:

Distance in Value to Policy

To respect distance mapping in policy space, exploit relationships between policy performance:

$$J(\pi') - J(\pi) = \mathbb{E}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(s_t, a_t)\right] = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}, a\sim\pi'}\left[A^{\pi}(s,a)\right]$$

Here $d^{\pi}(s) = (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P(s_t = s|\pi)$ is the **weighted** distribution of states. Making use,

$$\max_{\pi'} J(\pi') = \max_{\pi'} J(\pi') - J(\pi) = \max_{\pi'} \mathbb{E}_{\tau\sim\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi}(s_t, a_t)\right]$$

Now, rewrite the objective:

$$= \max_{\pi'} \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}, a\sim\pi'}\left[A^{\pi}(s,a)\right]$$

$$= \max_{\pi'} \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi'}, a\sim\pi}\left[\frac{\pi'(a|s)}{\pi(a|s)}A^{\pi}(s,a)\right]$$

Why rewrite?

1. Now, performance of π' is defined in advantages from π.
2. Requires trajectories sampled from π' (desired: from π because our tweak features $s \sim d^{\pi}$).

We have a useful approximation:

Relative Policy Performance Bounds

$$J(\pi') - J(\pi) \approx \mathbb{L}_{\pi}(\pi') \qquad \text{for close } \pi' \text{ and } \pi \ (d^{\pi'} = d^{\pi})$$

Approximation quality is ensured by relative policy performance bounds:

$$|J(\pi') - (J(\pi) + \mathbb{L}_{\pi}(\pi'))| \leq C\sqrt{\mathbb{E}_{s\sim d^{\pi}}\left[D_{KL}(\pi'||\pi)[s]\right]}$$

But what is $\mathbb{E}_{s\sim d^{\pi}}\left[D_{KL}(\pi'||\pi)[s]\right]$?

KL-Divergence

Such divergence measures distance between probability distributions:

$$D_{KL}(P||Q) = \sum_x P(x)\log\frac{P(x)}{Q(x)}$$

KL satisfies $D_{KL}(P||P) = 0, D_{KL}(P||Q) \geq 0, D_{KL}(P||Q) \neq D_{KL}(Q||P)$.
Between policies, $D_{KL}(\pi'||\pi)[s] = \sum_{a\in A}\pi'(a|s)\log\frac{\pi'(a|s)}{\pi(a|s)}$

After the approximation, we can optimize using trajectories sampled from the old policy π!

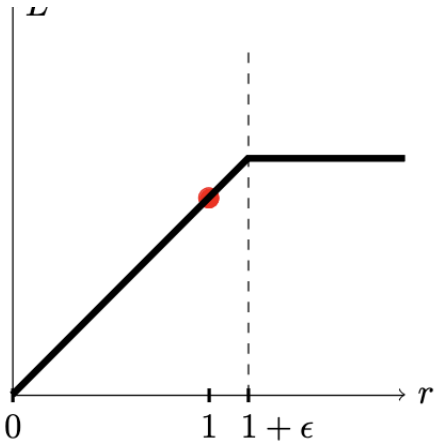
Policy Optimization under Bounded KL Approximation

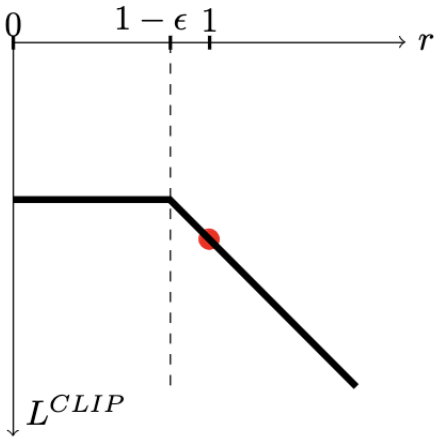
Policy improvement can be estimated by sampling from old policy π!

$$J(\pi') - J(\pi) \approx L_{\pi}(\pi') = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t\frac{\pi'(a_t, s_t)}{\pi(a_t, s_t)}A^{\pi}(s_t, a_t)\right]$$

4.3 Variants of PPO

Taking advantage of policy performance bounds, PPO penalizes large policy change iterations. There are two methods:





1. regularization term on KL-divergence (Adaptive Penalty)
2. pessimistic objective on far-away policies (Clipped Objective)

Adaptive Penalty

$$\theta_{k+1} = \arg\max_{\theta} L_{\theta_k}(\theta) - \beta \bar{D}_{KL}(\theta||\theta_k)$$

Here, KL-divergence is an expectation:

$$\bar{D}_{KL}(\theta||\theta_k) = \mathbb{E}_{s\ simd^{\pi_k}} D_{KL}\left(\theta_k(\cdot|s), \pi_\theta(\cdot|s)\right)$$

Penalty coefficient β_k changes between iterations:

- Initiate policy param θ_0, initial KL penalty β_0, target KL-divergence δ
- Compute policy update (iterate θ) by K steps of minibatch SGD (via Adam)
- Control KL-divergence to be around δ by adjusting penalty:
  If $\bar{D}_{KL}(\theta||\theta_k) \geq 1.5\delta$ then $\beta_{k+1} = 2\beta_k$;
  elif $\bar{D}_{KL}(\theta||\theta_k) \leq \frac{\delta}{1.5}$ then $\beta_{k+1} = \frac{1}{2}\beta_k$

This KL penalty is called "adaptive" because of how β_k changes (adapts quickly) according to KL-divergence.

An alternative approach is clipping: restrict via pessimistically treating objective value far away from θ_k.

Clipped Objective on Policy Changes

- Define relative probability change: $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$
- The new objective is

$$L^{\text{Clip}}_{\theta_k}(\theta) = \mathbb{E}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min\left(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_k}_t\right)\right]\right]$$

  In other words, $r_t(\theta)$ is clipped between $(1-\epsilon, 1+\epsilon)$; hyperparameter ϵ usually set at 0.2.
- Policy update: $\theta_{k+1} = \arg\max_{\theta} L^{\text{Clip}}_{\theta_k}(\theta)$

Here, Clipping discentivizes going far from θ_{k+1}:

- Left graph below: when the advantage function $A > 0$; right graph when $A < 0$.
- $L^{\text{Clip}}_{\theta_k}(\theta)$'s increase is suppressed at extreme values towards the sign of A.
- Clipping is simple to implement but works well compared to KL penalty.

PPO's performance consistently tops other algos, hence wildly popular.

- Today, it is a key component of ChatGPT (readings: OpenAI blog (2017), Publication by Schulman et. al. (2017))
- Different outcomes with reward scaling/ learning rate annealing.

Why KL-divergence applied is an expectation?

- Recall that the two distributions (of the same variable) occurs in probabilities.
- To compare, take average (expectation) over states (discounted distribution).

How does the two methods compare?

KL Penalty

- Idea: Tune penalty β adaptively to control KL around the target threshold
- Pros: Conceptually direct (literally penalize KL)
- Cons: Tricky to schedule/ tune/ stabilize β

Clipped Objective

- Idea: Force objective $r_t\theta$ to plateau when significantly deviate from 1.
- Pros: Simple to implement and good practical performance
- Cons: Less direct than penalty/ constraint

In practice, clipping is much more common/ popular given implementation and performance.

4.4 Generalized Advantage Estimator (GAE)

- Idea: Exponentially weighted average of k-step estimators
- $\hat{A}_t^{GAE(\gamma,\lambda)} = (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \dots) = (1-\lambda)\left[\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V)\right] = \sum_{i=0}^{\infty}(\gamma\lambda)^i\delta_{t+i}^V$
- PPO uses truncated GAE:
  $\hat{A}_t^{PPO} = \sum_{i=0}^{T-t-1}(\gamma\lambda)^i\delta_{t+i}^V$
  Benefit: Only run policy for T timesteps before updating - improve gradient estimation!

Properties of GAE

- GAE$(\gamma, \lambda = 0)$ is the advantage using TD(0): $\hat{A}_t^{(1)}$
- Given the series representation, we use $\lambda \in (0, 1)$: doesn't make sense with $\lambda = 0$
- Increasing λ introduces more variance, less bias.
  Larger bias on GAE$(\gamma, \lambda = 0)$ than GAE$(\gamma, \lambda = 0.99)$

Clarity: What is λ here? Both a technical and intuitive explanation?

4.5 Monotonic Improvement Theory

Recall:

- Approximation reason: No data ($\tau \sim \pi'$; only have π)
- Why use $\pi \approx \pi'$: approximation quality is ensured by KL-divergence

$$|J(\pi') - (J(\pi) + \mathbb{L}_\pi(\pi'))| \leq C\sqrt{\mathbb{E}_{s\sim d^\pi}\left[D_{KL}(\pi'||\pi)[s]\right]}$$

- Rearranging, $J(\pi') - J(\pi) \geq \mathbb{L}_\pi(\pi') - C\sqrt{\mathbb{E}_{s\sim d^\pi}\left[D_{KL}(\pi'||\pi)[s]\right]}$
  Maximizing RHS w.r.t. π' guarantees improving over π (majorize-maximize)
  RHS: $\mathbb{L}_\pi(\pi')$ and KL-term both estimatable from π samples!

Proof Sketch of Monotonic Improvement

- Set $\pi_{k+1} = \arg\max_{\pi'} \mathbb{L}_{\pi_k}(\pi') - C\sqrt{\mathbb{E}_{s\sim d^{\pi_k}}\left[D_{KL}(\pi'||\pi_k)[s]\right]}$
- π_k feasible with $J(\pi_{k+1}) - J(\pi_k) = 0$ because:
  (i) $\mathbb{L}_{\pi_k}(\pi_k) \propto \mathbb{E}\left[A^{\pi_k}(s,a)\right] = 0$
  (ii) $D_{KL}(\pi_k||\pi_k)[s] = 0$
- This guarantees optimal value ≥ 0: will improve by perf bounds!

4.6 Further Improvements

Performance Bounds: Practical Improvements

- Problems: Theoretical C is too high when $\gamma \to 1$, leading to small steps
- Solution: Tune KL (PPO); Use KL constraint (Trust region)

Despite the practical performance improvement, no longer guarantee monotonic improvements!

Summary: PPO

- Method: Clipping/ KL constraint to increase monotonic improvement likelihood
- Rationale: Take several gradient steps before gathering more data (improves eff.)
- Result: Converges to local optima
- Popularity: easy to implement ($\Rightarrow$ ChatGPT tuning)

Summary: Policy Gradient

- Advantage: Usable in non-differentiable reward $\to$ Hence more popular and useful
- Used in conjunction with model-free value methods, e.g. actor critics

5 Imitation Learning

IL is motivated by automating existent, very good decision policies for simple, cheap supervision:

- Human provides reward signals, RL algos make decisions.
- Strength: When easier for expert demonstration of "desired behaviour" than specifying reward that generates such.

IL feeds from the following inputs:

IL Inputs

(1) Reward Shaping
- Requires time-dense rewards to guide agents. This occurs in two ways:
  Manual: brittle design; Implicit: through demonstrations
- Demonstrations are from experts providing sequences of $\{s, a\}$ (demo trajectories).

(2) Inputs
- State s, Actions a, Transition $P(s'|s,a)$, demo. set $(s_0, a_0, s_1, a_1, \dots)$ from expert policy π^*
- No reward function R

5.1 Behavourial Cloning

The aim here: reduce expert policy learning to supervised learning.

- Method: (1) Fix policy class (NN/ Trees) and (2) Estimate from training examples
- Practical use: Bidirectional Convolutional Recurrent NN (BCRNN)
- Problem: Compounding Errors - SL assumes i.i.d. (s, a) pairs and time errors: this ignores the temporal structure! In other words: Data distribution mismatch (e.g. racing track case - no data on how to recover)
- Approx Intuition: Instead of uncorrelated errors ($\mathbb{E}[\text{Errors}] \propto T$), should be $\propto T^2$!

A remedy to solve the above is **DAGGER (Dataset Aggregation)**:

- Idea: More labels of expert action along paths taken by policy computed
- Output: Stationary deterministic policy under induced state distribution

> Q: What does it mean to have "no R" in inputs of IL? Q: Key limitations of DAGGER?

5.2 Inverse RL and Reward R

- Even under expert's optimal policy, there are infinitely many possible R values!
- With linear features, use linear VFA: $R(s) = \mathbf{w}^T x(s)$ where $\mathbf{w} \in \mathbb{R}^n$ is to be identified from demo.
- Express the value $V^\pi(s_0) = \mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)|s_0\right] = \mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t x(s_t)|s_0\right] = \mathbf{w}^T \mathbb{E}_{s\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t x(s_t)|s_0\right] = \mathbf{w}^T \mu(\pi, s_0)$
  Here $\mu(\pi, s_0)$ is the discounted weighted frequency of state features under policy π from state s_0.

In fact, if expert demos are from optimal policy π^*, sufficient to match discounted summed feature expectations to expert's policy (feature matching):

- Objective: $V^* \geq V^\pi$. Happens if $(w^*)^T \mu(\pi^*) \geq (w^*)^T \mu(\pi) \forall \pi \neq \pi^*$
- Feature matching: $||\mu(\pi) - \mu(\pi^*)||_1 \leq \epsilon \leftrightarrow \forall \mathbf{w} : ||\mathbf{w}||_\infty \leq 1, |\mathbf{w}^T \mu(\pi) - \mathbf{w}^T \mu(\pi^*)| \leq \epsilon$
- Issue: Which stochastic policy (infinite to match feature counts) or reward functions (infinite for same optimal policy) to choose?
- Suggestions: Max. Entropy inverse RL, generative adversarial IL

5.3 Principle of Max Entropy

Denote linear reward $R(s) = \mathbf{w}^T x(s)$, total feature count on single trajectory τ_j: $\mu_{\tau_j} = \sum_{s_i \in \tau_j} x(s_i)$; average feature count $\tilde{\mu} = \frac{1}{m} \sum_{j=1}^{m} \mu_{\tau_j}$.

5.3.1 Deterministic MDP

- Under linear reward model, a deterministic MDP's distribution of trajectory completely specifies policy!
  Principle: Choose $\mathbf{w}$ to select max entropy constrained to matching feature expectations.
  1. Constraint: $\sum_\tau P(\tau) = 1$, $P(\tau)\mu_\tau = \tilde{\tau}$
  2. Objective: $\max_P \left(\sum_\tau P(\tau) \log P(\tau)\right)$

The principle implies to maximize likelihood of observed data under max entropy (i.e. exponential) distribution:

$$P(\tau_j|\mathbf{w}^T) = \frac{1}{Z(\mathbf{w}^T)} \exp\left(\sum_{s_i \in \tau_j} \mathbf{w}^T x(s_i)\right), \quad Z(\mathbf{w}^T, s) = \sum_{\tau_s} \exp\left(\mathbf{w}^T \mu_{\tau_s}\right)$$

In other words, strongly prefer low-cost paths!

5.3.2 Stochastic MDP

For stochastic MDPs, the distribution over paths depends both on the reward weights and stochastic dynamics:

$$P(\tau_j|\mathbf{w}, P(s'|s,a)) \approx \frac{e^{\mathbf{w}^T \mu_{\tau_j}}}{Z(\mathbf{w}^T, P(s'|s,a))} \prod_{s_i, a_i \in \tau_j} P(s_{i+1}|s_i, a_i)$$

5.3.3 Then, how to learn the weight?

Learning the Weights via GD to Maximize Likelihood

Learn $\mathbf{w}$ to maximize data likelihood:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} L(\mathbf{w}) = \arg\max_{\mathbf{w}} \sum_{\text{examples}} \log P(\tau|\mathbf{w})$$

Apply GD again, where

$$\nabla L(\mathbf{w}) = \tilde{\mu} - \sum_{\tau} P(\tau|\mathbf{w})\mu_\tau = \tilde{\mu} - \sum_{s_i} D(s_i)x(s_i)$$

Here $\tilde{\mu}$ is the learner's expected empirical feature count, and the expected empirical feature counts can be expressed as expected state visitation frequencies.

Why does max entropy IRL work well here?

- Principled way for selecting among many possible reward function Rs.
- No longer need to know transition model P!
- Convenient to proceed (from this learned R) to use regular RL (ongoing research interest).

Q: Why approx, not equals, in the stochastic MDP? Are there any other reasons than the path dependence τ? Q: Does behavourial cloning require knowing P?