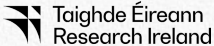
Policy Gradient Methods in Reinforcement Learning

Anton Baykalov

Ollscoil na Gaillimhe
University of Galway

MathLearn — June 2026



Plan

In this lecture:

- Intro to policy gradient methods
- REINFORCE: basic pgm-algorithm
- from REINFORCE to TRPO and PPO (most modern standard policy gradient method)

Literature:

- Reinforcement Learning, An Introduction. Richard S. Sutton and Andrew G. Barto
- Trust Region Policy Optimization, John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel
- Proximal Policy Optimization Algorithms. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov

Pawlicy gradient method

$\text{Cat} \approx \text{Car}$ – definitely a machine

Policy-base learning

- In the last lecture we approximated the value or action value function using parameters $\theta \in \mathbb{R}'$:

$$V_\theta \approx V^\pi(s) \text{ and } Q_\theta \approx Q^\pi(s, a)$$

- A policy was generated directly from the value function (ϵ-greedy or even deterministic).
- Why not parametrise the policy π **directely**?

$$\pi_\theta(a \mid s, \theta) = \mathbb{P}[A_t = a \mid S_t = s, \theta_t = \theta]$$

  - Note that such policy is **stochastic**!

Policy Gradient Methods

- **Policy Gradient Methods.** Methods for learning the policy parameter based on the gradient of some scalar performance measure $J(\theta)$ (**objective function**) with respect to the policy parameter.
- **Goal:** maximize $J(\theta)$, so updates of PGM approximate gradient ascent in J:

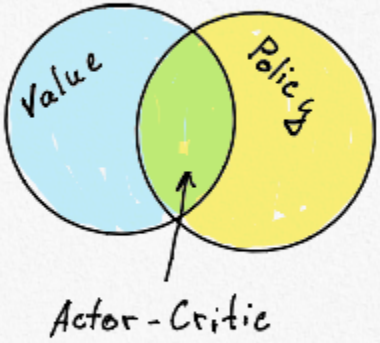
$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

where $\widehat{\nabla J(\theta_t)} \in \mathbb{R}^{d'}$ – approximates $\nabla J(\theta_t)$.

Example

$J(\theta) = \mathbb{E}_{\pi_\theta}(G_0)$, where $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

- Value based:
  - Learns Value function
  - Policy is implicit (e.g. ϵ-greedy)
- Policy based:
  - No Value function
  - Learns policy
- Actor-critic:
  - Learns Value function
  - Learns Policy



Parametrisation θ

If the action space is discrete and not too large, then a natural and common kind of parametrization is to form parametrized numerical **preferences**:

- $h(s, a, \theta) \in \mathbb{R}$ and then **soft-max** them:

$$\pi_\theta(a, s) := \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}$$

- $h(s, a, \theta)$ could be computed by an NN (MLP, CNN, GNN, any NN really) or even just linearly:

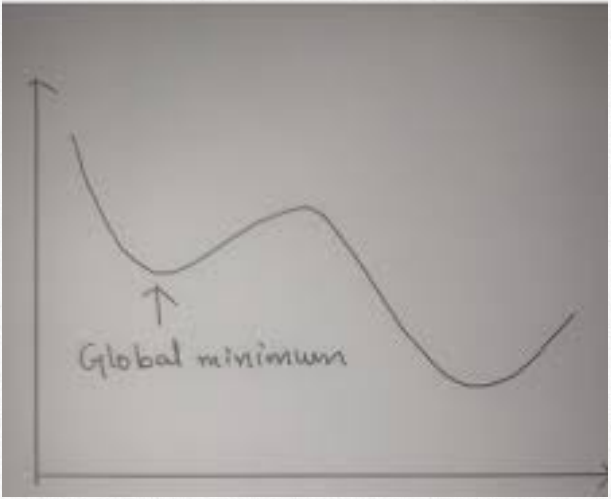
$$h(s, a, \theta) = \theta^\top \mathbf{x}(s, a)$$

where $\mathbf{x}(s, a)$ are **features** visible in state s while taking action a (basically a vector representing the state s or state and action, some kind of **observation**)

Why tho?

- Advantages:
  - Such approx. policy can approach/converge to a deterministic policy in contrast to a soft-max distribution based on action values
  - Effective in high-dimensional or continuous spaces
  - Enables the selection of action with arbitrary probabilities: the optimal approximate policy may be stochastic.
  - Policy may be a simpler function to approximate (tetris, for example).
  - A way to inject prior knowledge about desired form of the policy.
- Disadvantages (naive, vanilla version):
  - Typically converges to a local optimum
  - Evaluating a policy could be inefficient and high variance

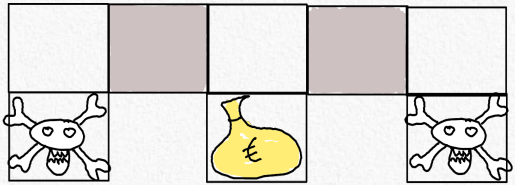
I'm going to tell my kids this is a global minimum



Examples!

- Rock-paper-scissors:
  - Deterministic policy is easily exploited
  - Uniform random policy is optimal (Nash equilibrium)
- Restricted knowledge of the state, i.e. $\mathbf{x}(s) = \mathbf{x}(s')$ for some states s and s'.

Deadly Grid World (1)

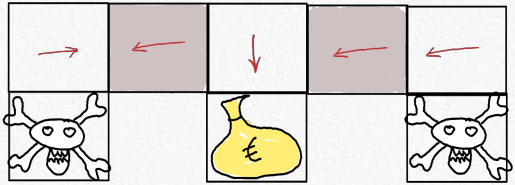


- Only know **features** of the state:

$$\mathbf{w}(s) = (x_1, x_2, x_3, x_4)$$

- $x_1 = 1$ if wall to N
- ...

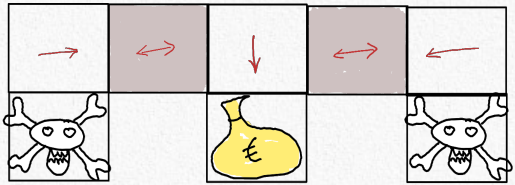
Deadly Grid World (2)



Optimal **deterministic** policy will either:

- move W in grey states
- move E in grey states

Deadly Grid World (3)



Optimal **Stochastic** policy:

- $\pi_\theta(\text{move E} \mid \text{wall to N and S}) = 1/2$
- $\pi_\theta(\text{move W} \mid \text{wall to N and S}) = 1/2$

How to get gradient of $J(\theta)$?

- $J(\theta) = \mathbb{E}_{\pi_\theta}(G_0) \approx \sum_{t=0}^{\infty} \gamma^t r_{t+1}$ depends on both **action selections** and **distribution of states** in which those selections are made, and both of these are affected by the policy parameters
- Given a state, the effect of parameters on the actions, and thus on reward, can be computed in a relatively straightforward way from knowledge of the parametrization.
- effect of the policy on the state distribution is a function of the environment and is typically unknown

Question

How can we estimate the performance gradient with respect to the policy parameter when the gradient depends on the unknown effect of policy changes on the state distribution?

Theorem

Theorem (Policy Gradient Theorem)

$$\nabla J(\theta) \propto \sum_{s} \mu(s) \sum_{a} q_\pi(s, a) \nabla \pi(a \mid s, \theta)$$

- $\mu(s) = \eta(s) / \sum_{s'} \eta(s')$ – on policy distribution of states under π, i.e. fraction of time spent in each state normalised to some to one
- $$\eta(s) = \Pr[\text{episode starts in } s] + \sum_{s'} \eta(s') \sum_{a} \pi(a \mid s') \Pr(s|s', a)$$

– number of time-steps spent in state s in a single episode **on average**

REINFORCE

- Notice that

$$\sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a \mid s, \theta) = \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a \mid S_t, \theta) \right]$$

- We could stop here and just do **all actions** algorithm

$$\theta_{t+1} := \theta_t + \alpha \sum_a \hat{q}_\pi(S_t, a) \nabla \pi(a \mid S_t, \theta)$$

where $\hat{q}$ is some learned approximation to q_π

REINFORCE

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left[\sum_a \pi(a \mid S_t, \theta) q_\pi(S_t, a) \frac{\nabla \pi(a \mid S_t, \theta)}{\pi(a \mid S_t, \theta)} \right]$$

$$\mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)} \right] = \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)} \right]$$

- REINFORCE update:

$$\theta_{t+1} := \theta_t + \alpha G_t \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}$$

Baseline

$$\nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s,a) - b(s)) \nabla \pi(a \mid s, \theta)$$

- $b(s)$ – could be any function that does not vary with a

$$\sum_a b(s) \nabla \pi(a \mid s, \theta) = b(s) \sum_a \nabla \pi(a \mid s, \theta) = b(s) \nabla 1 = 0$$

$$\theta_{t+1} := \theta_t + (\alpha G_t - b(S_t)) \frac{\nabla \pi(A_t \mid S_t, \theta)}{\pi(A_t \mid S_t, \theta)}$$

- natural choice for $b(s)$ is an estimate of the state value $\hat{v}(S_t)$.

REINFORCE problems

- High variance – returns are noisy, learning is unstable
- Sample inefficiency — on-policy, every episode is thrown away after one update
- No guarantee on step size – a bad update can catastrophically collapse the policy

Trust Regeon

- How much can we change θ in one update without destroying the policy?
- We want to stay close in **behaviour** space, not just parameter space.
- **Kullback–Leibler** (**KL**) **divergence**:

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}$$

For P and Q – probability distributions.

TRPO: Trust Region Policy Optimization (2015)

- maximise **surrogate objective** (wrt θ):

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)} \left(\hat{Q}_\pi(a_t, s_t) - \hat{V}_\pi(s_t) \right) \right]$$

  here $\hat{\mathbb{E}}_t$ indicates the empirical average over a finite batch of samples

- subject to

$$\hat{\mathbb{E}}_t \left[KL[\pi_\theta(\cdot \mid s_t), \pi_{\theta_{old}}(\cdot \mid s_t)] \right] \leqslant \delta$$

- painful in practice: requires computing the Fisher Information Matrix + conjugate gradient + line search — very expensive per update

PPO: Proximal Policy Optimization (2017)

- $A_t = Q_\pi(a, s) - V_\pi(s)$ – advantage
- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
- maximaze

$$L_t^{CPIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right]$$

default $\epsilon = 0.2$

- $$clip(r_t, 1-\varepsilon, 1+\varepsilon) = \begin{cases} 1-\varepsilon & \text{if } r_t < 1-\varepsilon \\ r_t & \text{if } 1-\varepsilon \leqslant r_t \leqslant 1+\varepsilon \\ 1+\varepsilon & \text{if } r_t > 1+\varepsilon \end{cases}$$

PPO on practice

- advantage-function estimators make use of a learned state-value function $V(s)$
- If using a NN architecture that shares parameters between the policy and value function, use a loss function that combines the policy surrogate and a value function error term.
- Objective can further be augmented by adding an entropy bonus to ensure sufficient exploration

$$L_t^{CPIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CPIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right]$$

TRPO vs. PPO

- PPO was designed as a direct answer to TRPO's implementation complexity
- Fisher Information Matrix and conjugate gradient are gone
- The trade-offs PPO does have are different ones:
  - Clipping is a heuristic, not a theorem — you can still occasionally get a bad update if ϵ is too large or number of epochs per batch is too high
  - Sensitive to the entropy bonus and clipping hyperparameter
  - It's still on-policy — you throw away the rollout after K epochs, so it's less sample efficient than some off-policy methods
- None of these come close to TRPO's computational burden.

PPO algorithm (actor critic style)

```
for iteration=1, 2, . . . do
    for actor=1, 2, . . . , N do
        Run policy $pi_old$ in environment for T timesteps
        Compute advantage estimates A_1, . . . ,A_T
    end for
    Optimize surrogate L wrt theta, with K epochs and
        minibatch size M <= NT
    theta_old := theta
end for
```