Reinforcement Learning

Lecture 8: Policy gradient methods

Chris G. Willcocks

Durham University

Lecture overview

Lecture covers chapter 13 in Sutton & Barto [1] and examples from David Silver [2]

1. **Policy-based methods**
   - definition
   - characteristics
   - deterministic vs stochastic policies
2. **Policy gradients**
   - gradient-based estimator
   - Monte Carlo REINFORCE
3. **Actor-critic methods**
   - definition
   - algorithm
   - function approximation error in AC methods
   - extensions

Policy-based methods introduction

Definition: policy-based methods

Last week, we used a function approximator to estimate the value function:

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s),$$

and for control we estimated Q:

$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a).$$

This week we will estimate policies:

$$\pi_\theta(a|s) = P(a|s, \theta)$$

Given a state, what's the distribution over actions?

Example: what's the optimal policy?

Policy-based methods characteristics

Policy-based RL characteristics

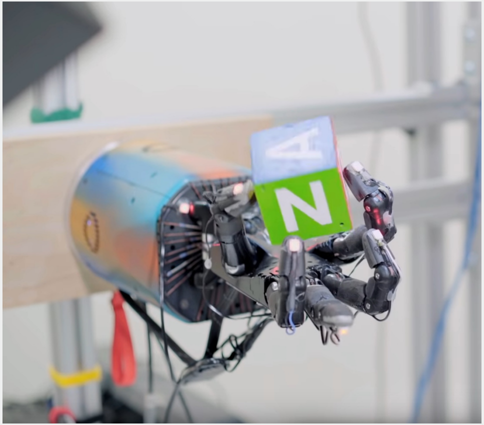
This approach has the following **advantages**:

- Can be more efficient that calculating the value function
- Better convergence guarantees
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

And the following **disadvantages**:

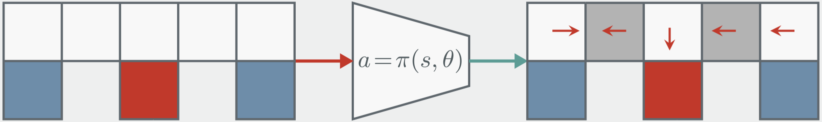
- Converges on local rather than global optimum
- Inefficient policy evaluation with high variance

Example: continuous action spaces

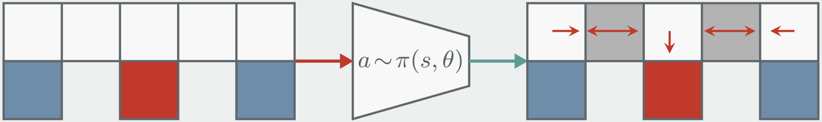


Example: deterministic vs stochastic policies

Deterministic policy for feature vectors describing the walls around a state:



Stochastic policy:



Example from [2].

Definition: gradient estimators

While we could optimise θ for non-differentiable functions using approaches such as genetic algorithms or hill climbing, ideally we want to use a gradient based estimator:

$$\mathcal{L}^{\text{PG}}(\theta) = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t \right]$$

where $\hat{A}_t$ is an estimate of the 'advantage' (difference between the return and the state values, you could also replace $\hat{A}_t$ with $q(s,a)$ instead for higher variance). The expectation $\hat{\mathbb{E}}_t$ is an empircal average over a finite batch of samples [3]. Typically π follows a categorical distribution (softmax) or a Gaussian for continuous action spaces.

Therefore we empirically follow the gradient that maximizes the likelihood of the actions that give the most advantage.

Definition: Monte Carlo REINFORCE

REINFORCE estimates the return in the previous equation by using a Monte Carlo estimate [4].

- Initialise some arbitrary parameters θ
- Iteratively sample episodes
- Calculate the complete return from each step
- For each step again, update in the gradient times the sample return

Algorithm: Monte Carlo REINFORCE

PyTorch example:

```
# initialise θ with random values
π = PolicyNetwork(θ)

while(True):
  # sample episode following π
  S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T ~ π

  for t in range(T - 1):
    G_t ← Σ_{k=t+1}^{T} γ^{k-t-1} R_k
    θ ← θ + α γ^t G_t ∇ ln π(A_t|S_t, θ)
```

$S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \sim \pi$

$G_t \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$

$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla \ln \pi(A_t|S_t, \theta)$

Actor-critic methods introduction

Definition: actor-critic methods

We combine policy gradients with action-value function approximation, using two models that may (optionally) share parameters.

- We use a **critic** to estimate the Q values:

$$q_{\mathbf{w}}(s,a) \approx q^{\pi_\theta}(s,a)$$

- We use an **actor** to update the policy parameters θ in the direction suggested by the critic.

Example: Actor critic

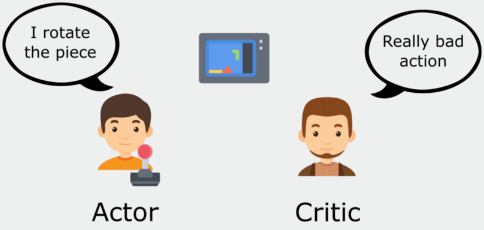


Image from freecodecamp.org

Actor-critic methods algorithm

Definition: actor-critic

Putting this together, actor-critic methods use an approximate policy gradient to adjust the actor policy in the direction that maximises the reward according to the critic:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) q_{\mathbf{w}}(s, a)$$

Algorithm: Actor-Critic (**PyTorch**)

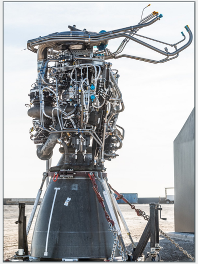
```
# initialise s, θ, w randomly
# sample a ~ π_θ(a|s)
for t in range(T):
  sample r_t and s' from environment(s, a)
  sample a' ~ π_θ(a'|s')
  θ ← θ + α q_w(s, a) ∇_θ ln π_θ(a|s) # update actor
  δ_t = r_t + γ q_w(s', a') − q_w(s, a) # TD error
  w ← w + α δ_t ∇_w q_w(s, a) # update critic
  a ← a', s ← s'
```

Actor-critic methods TD3

Function approximation error in actor-critic

In practice, tricks are needed to reduce function approximation in actor-critic methods. TD3 [5]:

- Dual critic networks + actor network
- Adds $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$ to actions
- Delayed policy updates, replay buffer



Algorithm: TD3 [5]

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Extensions and further reading

Extensions

This has introduced the foundations, hopefully now you have a good platform to read about the extensions to this.

1. **Recommended further study (papers & code):**
2. **Recommended further study (theory & STAR):**

Extensions include:

- Advantage actor critic (A3C & A2C) [6]
- Experience replay & prioritised replay [7]
- Proximal policy optimisation [3]
- Addressing Function Approximation Error in Actor-Critic Methods (TD3) [5] - recommended starter

Take Away Points

Summary

In summary:

- Policy gradients open up many new extensions
- Choose extensions to reduce variance to stabilise training
- Consider regularisation to encourage exploration
- Going off-policy gives better exploration
- Its possible for the actor and critic to share some lower layer parameters, but be careful about it
- Experience replay can increase sample efficiency (where simulation is expensive)

References I

[1] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction (second edition). Available online. MIT press, 2018.

[2] David Silver. Reinforcement Learning lectures. https://www.davidsilver.uk/teaching/. 2015.

[3] John Schulman et al. “Proximal policy optimization algorithms”. In: arXiv preprint arXiv:1707.06347 (2017).

[4] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: Machine learning 8.3-4 (1992), pp. 229–256.

[5] Scott Fujimoto, Herke Hoof, and David Meger. “Addressing function approximation error in actor-critic methods”. In: International Conference on Machine Learning. PMLR. 2018, pp. 1587–1596.

[6] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: International conference on machine learning. 2016, pp. 1928–1937.

References II

[7] Ziyu Wang et al. “Sample efficient actor-critic with experience replay”. In: arXiv preprint arXiv:1611.01224 (2016).

[8] Shixiang Gu et al. “Q-prop: Sample-efficient policy gradient with an off-policy critic”. In: arXiv preprint arXiv:1611.02247 (2016).