Introduction to Gradient-Based Direct Policy Search

Adrien Bolland (adrien.bolland@uliege.be)

Outline

Notations

In this course, we use the classic reinforcement learning notations:

- $s \in \mathcal{S}$ for the states,
- $a \in \mathcal{A}$ for the actions,
- $\pi(a|s)$ for the stationary stochastic policy,
- $J(\pi)$ for the expected return of the policy π,
- $V^\pi(s)$ for the state value function of policy π,
- $Q^\pi(s, a)$ for the state-action value function of policy π.

Direct Policy Search

Markov decision process

An MDP is represented by its model $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, p_0, \gamma)$:

- States $s_t \in \mathcal{S}$,
- Actions $a_t \in \mathcal{A}$,
- Transition distribution $T(s_{t+1}|s_t, a_t)$,
- Reward function $r_t = R(s_t, a_t)$,
- Initial distribution $p_0(s_0)$,
- Discount factor $\gamma \in [0, 1)$.

In MDPs, states satisfy the Markov property

$$\begin{aligned} p(s_{t+1}|s_0, a_0, \ldots, s_t, a_t) &= p(s_{t+1}|s_t, a_t) \\ &= T(s_{t+1}|s_t, a_t). \end{aligned}$$

Stochastic policies in MDPs

Definition (Stationary stochastic policy)

A stationary stochastic policy $\pi \in \Pi = \mathcal{S} \rightarrow \Delta(\mathcal{A})$ is a mapping from a state to a distribution over the actions, whose density writes $\pi(a_t|s_t)$.

There exists an optimal stationary stochastic policy.

Definition (State value function)

The state value function $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$ of the policy equals

$$V^\pi(s) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right]$$

Direct Policy Search

Do not solve a more general problem as an intermediate step.

— Vladimir Vapnik, 1998

As we care about optimal behaviour, why not directly learning a policy?

Direct Policy Search – Objective Function

Definition (Problem Statement)

In direct policy search we look for a policy $\pi^* \in \Pi$ maximizing the expect discounted sum of rewards (i.e., the expected return of the policy)

$$J(\pi) = \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0(\cdot) \\ a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right].$$

Bellman optimal policies respect this optimality criterion.

Direct Policy Search – Advantages

Policy-based RL has several advantages compared to value-based RL:

1. We optimize the true control objective.
2. It extends to continuous state-action spaces.
3. Sometimes simple behaviours are optimal while value functions are complex.

We will focus on stochastic policies as their expected return is usually smoother than deterministic policies.

Policy-Gradient Methods

Policy Gradient Methods – Recipe

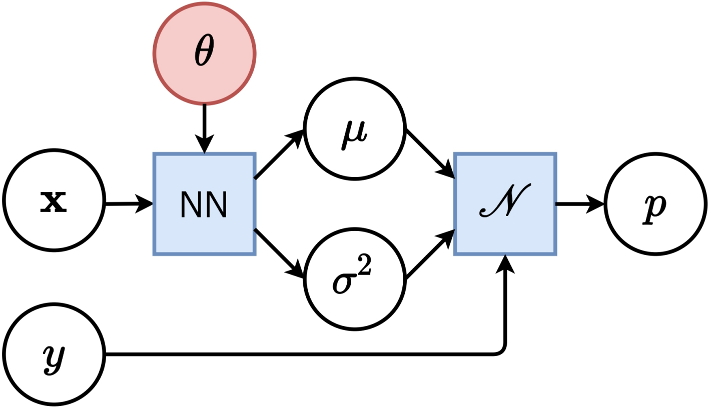
Policy-gradient algorithms are direct policy search algorithms where:

1. We represent the policy with a differentiable parametric function π_θ.
2. We perform stochastic gradient ascent on the expected return.

Gaussian policies are simple parameterizations in which actions are draw as

$$a_t \sim \mathcal{N}(\cdot | \mu_\theta(s_t), \Sigma_\theta(s_t)).$$

How to represent such a distribution with a neural network?



https://github.com/glouppe/info8010-deep-learning

Policy Gradient Theorem

$$\nabla_\theta J(\pi_\theta) = \nabla_\theta \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0(\cdot) \\ a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right]$$

1. The gradient of the Monte-Carlo estimate of the expectation equals zero.
2. How to compute the gradient?

Policy Gradient Theorem

Theorem (Policy Gradient Theorem)

For any differentiable policy π_θ, the gradient of $J(\pi_\theta)$ is [Sutton et al., 1999]

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\substack{s_0 \sim p_0(\cdot) \\ a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \right],$$

where

$$Q^{\pi_\theta}(s, a) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right].$$

- How to approximate the state-action value function Q^{π_θ}?
- How to approximate the expectation?

Using Monte-Carlo over n i.i.d. trajectories, we get

$$\hat{\nabla}_\theta J(\pi_\theta) = \left\langle \sum_{t=0}^{T-1} \gamma^t \hat{Q}_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right\rangle_n$$
$$\hat{Q}_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_t \ ,$$

where $\langle \cdot \rangle_n$ is the mean of the n estimates.

- This estimate is unbiased for $T \to \infty$.
- It is common practice to neglect the γ^t in the gradient expression.
- It can be shown that $\|Q^{\pi_\theta} - \mathbb{E}[\hat{Q}_t]\|_\infty \leq \frac{\gamma^{T-t}}{1-\gamma} \max_{s,a} R(s,a)$.

REINFORCE algorithm

In summary, the REINFORCE algorithm writes as follows.

Algorithm 1: REINFORCE algorithm

Initialise θ randomly.
for $k \leftarrow 1, \ldots, K$ **do**
    Sample n trajectories with the current policy in the MDP
    Update $\theta_k = \theta_{k-1} + \alpha_k \hat{\nabla}_\theta J(\pi_\theta)$

The policy gradient is usually computed by automatic differentiation on the loss

$$\mathcal{L}(\theta) = - \left\langle \sum_{t=0}^{T-1} \gamma^t \hat{Q}_t \log \pi_\theta(a_t | s_t) \right\rangle_n .$$

Variance Reduction and Actor-Critic Methods

Baseline

- The gradient estimate can be subject to a large variance!
- Subtracting a baseline from the cumulative reward can decrease the variance.

$$\hat{\nabla}_\theta J(\pi_\theta) = \left\langle \sum_{t=0}^{T-1} \gamma^t \left(\hat{Q}_t - b_t\right) \nabla_\theta \log \pi_\theta(a_t|s_t) \right\rangle_n$$

- In practice, it is common to choose the mean cumulative reward.

$$b_t = \left\langle \sum_{t'=t}^{T-1} \gamma^{t'-t} r_t \right\rangle_n .$$

Baseline in General

Baselines keep the gradient estimate unbiased!

Theorem (Policy Gradient Theorem with Baseline)

For any differentiable policy π_θ, for any function of the state f, the gradient of $J(\pi_\theta)$ is

$$\nabla_\theta J(\pi_\theta) = \underset{\substack{s_0 \sim p_0(\cdot) \\ a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}}{\mathbb{E}} \left[\sum_{t=0}^{\infty} \gamma^t (Q^{\pi_\theta}(s_t, a_t) - f(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right].$$

When we use the mean cumulative rewards as baseline, we use an approximation of the state value function!

Advantage Actor-Critic

- Actor-Critic Algorithms use a function approximator (the critic) when estimating $Q^{\pi_\theta}(s_t, a_t) - f(s_t)$!
- Advantage Actor-Critic (A2C) learns the value function V_ϕ of the current policy [Mnih et al., 2016]

$$\hat{\nabla}_\theta J(\pi_\theta) = \left\langle \sum_{t=0}^{T-1} \gamma^t \left((\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} + \gamma^T V_\phi(s_T)) - V_\phi(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t) \right\rangle_n .$$

How to learn the parameters of V_ϕ?

Policy Evaluation

The objective is to find the parametric functions such that $\forall s$

$$V_\phi(s) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right].$$

An approximation is found solving the regression problem of minimizing $\forall s$

$$D(\phi) = \frac{1}{2} \left(V_\phi(s) - \mathop{\mathbb{E}}_{\substack{a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right] \right)^2 .$$

It can be solved by gradient descent following the negation of

$$\nabla_\phi D(\phi) = \left(V_\phi(s) - \mathop{\mathbb{E}}_{\substack{a_t \sim \pi_\theta(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right] \right) \nabla_\phi V_\phi(s).$$

Value Function Evaluation with Monte-Carlo

The first approach is Monte-Carlo Learning!

The gradient is estimated using n i.i.d. trajectories

$$\hat{\nabla}_\phi D(\phi) = \left\langle \sum_{t=0}^{T-1} \left(V_\phi(s_t) - \sum_{t'=t}^{T-1} \gamma^{t} r_{t'} \right) \nabla_\phi V_\phi(s_t) \right\rangle_n .$$

The gradient can be computed with automatic differentiation on the loss

$$\mathcal{L}(\phi) = \frac{1}{2} \left\langle \sum_{t=0}^{T-1} \left(V_\phi(s_t) - \sum_{t'=t}^{T-1} \gamma^{t-t'} r_{t'} \right)^2 \right\rangle_n .$$

Gradient estimates are unbiased but subject to high variance!

Value Function Evaluation with TD-Learning

The second approach is Temporal-Difference (TD) Learning!

The gradient is estimated with a bootstrap estimate of the value function

$$\hat{\nabla}_\phi D(\phi) = \left\langle \sum_{t=0}^{T-1} \left(V_\phi(s_t) - (r_t + \gamma V_\phi(s_{t+1})\right) \nabla_\phi V_\phi(s_t) \right\rangle_n .$$

The gradient can be computed with automatic differentiation on the loss

$$\mathcal{L}(\phi) = \frac{1}{2} \left\langle \sum_{t=0}^{T-1} \left(V_\phi(s_t) - (r_t + \gamma V_\phi(s_{t+1})\right)^2 \right\rangle_n ,$$

where the gradient of $V_\phi(s_{t+1})$ is neglected.

This approach is more stable but provides biased gradient estimates!

Value Function Evaluation with Multi-step TD learning

Multi-step TD-learning combines both world and use

$$\nabla_\phi D(\phi) = \left\langle \sum_{t=0}^{T-1} \left(V_\phi(s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - \gamma^{T-t} V_\phi(s_T) \right) \nabla_\phi V_\phi(s_t) \right\rangle_n .$$

The gradient can be computed with automatic differentiation on the loss

$$\mathcal{L}(\phi) = \frac{1}{2} \left\langle \sum_{t=0}^{T-1} \left(V_\phi(s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - \gamma^{T-t} V_\phi(s_T) \right)^2 \right\rangle_n ,$$

where the gradient of $V_\phi(s_T)$ is neglected.

A2C algorithm

In summary, the A2C algorithm writes as follows.

Algorithm 2: A2C algorithm

Initialize θ randomly.
for $k \leftarrow 1, \ldots, K$ **do**
    Sample n trajectories with the current policy in the MDP
    Update $\phi_k = \phi_{k-1} - \alpha_k \hat{\nabla}_\phi D(\phi)$
    Update $\theta_k = \theta_{k-1} + \beta_k \hat{\nabla}_\theta J(\pi_\theta)$

A2C algorithm

- It is said to be an on-policy algorithm.
- As such, the algorithm is prone to converge towards local extrema.

Entropy Regularization

Local Optimality

- Large variance decreases the expected return of the policy.
- In practice the gradient ascent thus tends to reduce the variance.
- The policy converges towards a deterministic policy.
- The policy has a larger but less concave return.

The gradient ascent converges to a locally optimal deterministic policy!

Avoiding Local Optimality – Variance Control

A simple approach is to add a constant disturbance to the actions, a Gaussian policy would provide actions distributed as

$$a_t \sim \mathcal{N}(\cdot|\mu_\theta(s_t), \Sigma_\theta(s_t) + \Lambda_k).$$

This approach is less trivial with other action distributions (e.g., mixture, beta-distribution, normalizing flow).

Avoiding Local Optimality – Entropy Regularization

The preferred approach is to provide an entropy bonus $\mathcal{H}(\pi_\theta)$ to the return.

$$\mathcal{H}(\pi_\theta) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(\cdot|s_t) \\ s_{t+1} \sim T(\cdot|s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t \log \pi_\theta(a_t|s_t) \right]$$

The gradient can be estimated.

1. By automatic differentiation if (for each state) the conditional entropy of the policy has a closed form.
2. By providing reward bonuses proportional to the log-likelihood of the sampled actions otherwise.

Avoiding Local Optimality – Algorithm

Algorithm 3: A2C algorithm with entropy regularization

Initialise θ randomly.
for $k \leftarrow 1, \ldots, K$ **do**
    Sample n trajectories with the current policy in the MDP
    Update $\phi_k = \phi_{k-1} - \alpha_k \hat{\nabla}_\theta \mathcal{L}(\phi)$
    Update $\theta_k = \theta_{k-1} + \beta_k \hat{\nabla}_\theta J(\pi_\theta) + \lambda_k \hat{\nabla}_\theta \mathcal{H}(\pi_\theta)$

Conclusions

Conclusion

In summary:

- We introduced direct policy search.
- We saw how to optimize a policy with the PG Theorem.
- Variance reduction lead us to the A2C algorithm.
- Entropy regularization enhances the performance of A2C.

Next week:

- We will dive into more complex on-policy algorithms.
- We will see a first off-policy method.

References

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.