March 20, 2024

Lecture 03

Revision: Policy gradients

→ Motivation: Handling continuous action spaces

→ Instead of gradient descent, we will write the return G_t as our objective (function of policy) and do gradient ascent

→ Consider policy $\pi(a|s;\theta)$

→ $J(\theta) = \sum \pi(a|s_0;\theta) \cdot Q_\theta(s_0,a) = V_\theta(s_0)$ (s_0 = initial state)

$$\nabla V_\theta(s) = \nabla \left[\sum_a \pi(a|s;\theta) Q_\theta(s,a) \right]$$

s → general state

$$\Rightarrow \nabla J(\theta) \propto \sum_s \mu(s) \sum_a \nabla \pi(a|s;\theta) Q_\theta(s,a)$$

$Q_\theta(s,a)$: available for free in PyTorch; replace by Monte-Carlo returns

$\mu(s)$: Frequency of visits to s → measure in samples

$\sum_a \nabla \pi(a|s;\theta)$: this summation is tricky to get rid of

$$\nabla J(\theta) \propto E_\pi \left[\sum_a \nabla \pi(a|s;\theta) \cdot G_t(s,a) \right]$$

$$= E_\pi \left[\left(\sum_a \pi \right) \cdot \frac{\nabla \pi}{\pi} \cdot G_t \right]$$

absorb

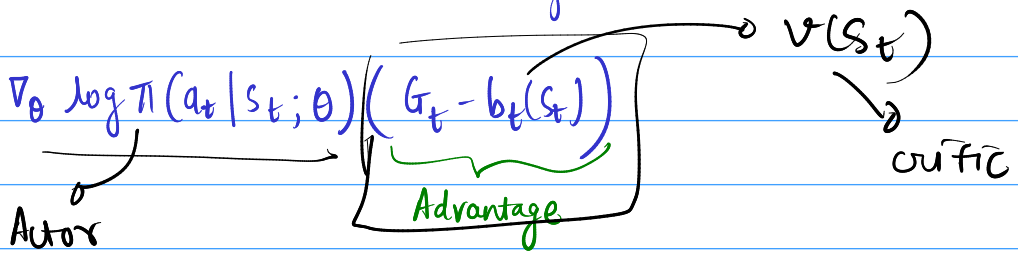
$$\nabla J(\theta) \propto E_\pi \left[\frac{\nabla \pi}{\pi} G_t \right] = E_\pi [\nabla \log \pi \cdot G_t]$$

$$\theta_{t+1} = \theta_t + \underbrace{\eta}_{\text{learning rate}} \cdot E_\pi [\nabla \log \pi \cdot G_t]$$

Actor critic methods in Deep RL

Basic policy gradient: $\nabla_\theta \log \pi(a_t|s_t;\theta)\, G_t$ — called R_t in the A3C paper

Difficult to distinguish between G_t when results are similar, more stable gradients:

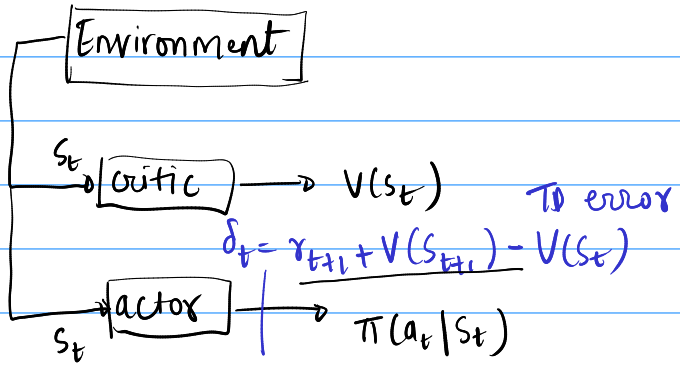


For us, $\theta \rightarrow$ parameters (weights, biases) of a neural network

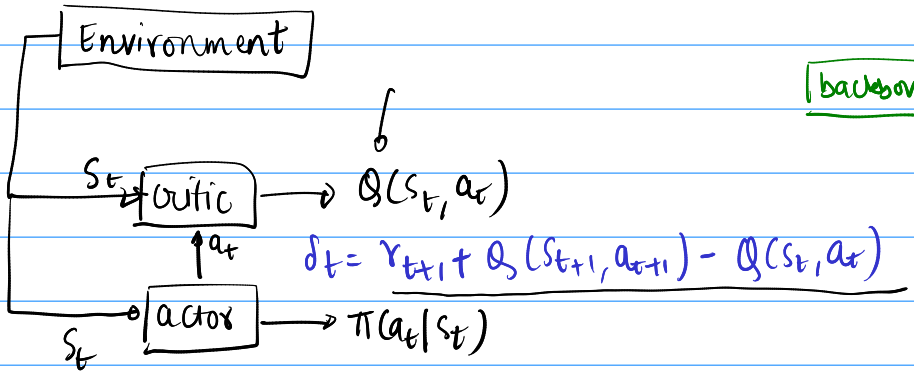
$\nabla_\theta \rightarrow$ gradient ascent

Two implementation options

①



②



* In both cases, one can use a common backbone for actor & critic

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

First look at policy based methods

In contrast to value-based methods, policy-based model-free methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameters θ by performing, typically approximate, gradient ascent on $\mathbb{E}[R_t]$. One example of such a method is the REINFORCE family of algorithms due to Williams (1992). Standard REINFORCE updates the policy parameters θ in the direction $\nabla_\theta \log \pi(a_t|s_t;\theta) R_t$, which is an unbiased estimate of $\nabla_\theta \mathbb{E}[R_t]$. It is possible to reduce the variance of this estimate while keeping it unbiased by subtracting a learned function of the state $b_t(s_t)$, known as a baseline (Williams, 1992), from the return. The resulting gradient is $\nabla_\theta \log \pi(a_t|s_t;\theta)\,(R_t - b_t(s_t))$.

we use notation G_t

gradient

A learned estimate of the value function is commonly used as the baseline $b_t(s_t) \approx V^\pi(s_t)$ leading to a much lower variance estimate of the policy gradient. When an approx-

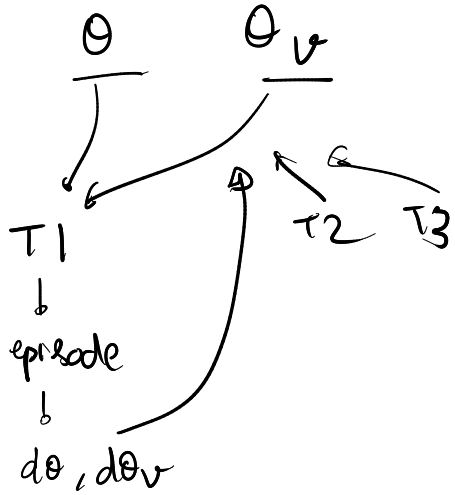
A learned estimate of the value function is commonly used as the baseline $b_t(s_t) \approx V^\pi(s_t)$ leading to a much lower variance estimate of the policy gradient. When an approximate value function is used as the baseline, the quantity $R_t - b_t$ used to scale the policy gradient can be seen as an estimate of the *advantage* of action a_t in state s_t, or $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$, because R_t is an estimate of $Q^\pi(a_t, s_t)$ and b_t is an estimate of $V^\pi(s_t)$. This approach can be viewed as an actor-critic architecture where the policy π is the actor and the baseline b_t is the critic (Sutton & Barto, 1998; Degris et al., 2012).

Skip to A3C now

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i;\theta'_v))^2 / \partial \theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

A3C



CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

DDPG → Deep deterministic policy gradients

Timothy P. Lillicrap,* Jonathan J. Hunt,* Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind
London, UK
{countzero, jjhunt, apritzel, heess, etom, tassa, davidsilver, wierstra} @ google.com

ABSTRACT

We adapt the ideas underlying the success of Deep Q-Learning to the continuous action domain. We present an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Using the same learning algorithm, network architecture and hyper-parameters, our algorithm robustly solves more than 20 simulated physics tasks, including classic problems such as cartpole swing-up, dexterous manipulation, legged locomotion and car driving. Our algorithm is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives. We further demonstrate that for many of the tasks the algorithm can learn policies "end-to-end": directly from raw pixel inputs

Many approaches in reinforcement learning make use of the recursive relationship known as the Bellman equation:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]\right] \tag{2}$$

If the target policy is deterministic we can describe it as a function $\mu : \mathcal{S} \leftarrow \mathcal{A}$ and avoid the inner expectation:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right] \tag{3}$$

The expectation depends only on the environment. This means that it is possible to learn Q^{μ} off-policy, using transitions which are generated from a different stochastic behavior policy β.

Q-learning (Watkins & Dayan, 1992), a commonly used off-policy algorithm, uses the greedy policy $\mu(s) = \arg\max_a Q(s, a)$. We consider function approximators parameterized by θ^Q, which we optimize by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}\left[\left(Q(s_t, a_t|\theta^Q) - y_t\right)^2\right] \tag{4}$$

where

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q). \tag{5}$$

While y_t is also dependent on θ^Q, this is typically ignored.

Using chain rule:

$$\begin{aligned}\nabla_{\theta^{\mu}} J &\approx \mathbb{E}_{s_t \sim \rho^{\beta}}\left[\nabla_{\theta^{\mu}} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^{\mu})}\right] \\ &= \mathbb{E}_{s_t \sim \rho^{\beta}}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_{\mu}} \mu(s|\theta^{\mu})|_{s=s_t}\right]\end{aligned}$$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^{\mu})$ with weights θ^Q and θ^{μ}.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^{\mu}) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

  end for
end for

Soft Actor-Critic:
Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor

Tuomas Haarnoja [1] Aurick Zhou [1] Pieter Abbeel [1] Sergey Levine [1]

2009). This challenge is further exacerbated in continuous state and action spaces, where a separate actor network is often used to perform the maximization in Q-learning. A commonly used algorithm in such settings, deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), provides for sample-efficient learning but is notoriously challenging to use due to its extreme brittleness and hyperparameter sensitivity (Duan et al., 2016; Henderson et al., 2017).

We explore how to design an efficient and stable model-

effectively they think of the actor as the action optimiser

with the expected entropy of the policy over $\rho_\pi(\mathbf{s}_t)$:

$H = \mathbb{E}[\log p] = \mathbb{E}[-\log p]$

note this is inside expectation.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi(\cdot|\mathbf{s}_t)) \right]. \quad (1)$$

If used as a regulariser, it will appear as an add-on term outside the rest of the objective

The temperature parameter α determines the relative importance of the entropy term against the reward, and thus controls the stochasticity of the optimal policy. The maximum entropy objective differs from the standard maximum expected reward objective used in conventional reinforcement learning, though the conventional objective can be recovered in the limit as $\alpha \to 0$. For the rest of this paper, we will omit writing the temperature explicitly, as it can always be subsumed into the reward by scaling it by α^{-1}.

This objective has a number of conceptual and practical

a modified Bellman backup operator $\mathcal{T}^\pi$ given by

$$\mathcal{T}^\pi Q(\mathbf{s}_t, \mathbf{a}_t) \triangleq r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V(\mathbf{s}_{t+1}) \right], \quad (2)$$

where

positive terms

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi} \left[Q(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t) \right] \quad (3)$$

$\log\left(\frac{e^Q}{\pi}\right)$

~~Atari~~ MuJoCo

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}} \left(\pi'(\cdot|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q^{\pi_{\text{old}}}(\mathbf{s}_t, \cdot)\right)}{Z^{\pi_{\text{old}}}(\mathbf{s}_t)} \right)$$

$\frac{e^{x_i}}{\sum e^{x_i}}$

$D_{KL}(p \| q) = \mathbb{E}_p \log p/q$

and policy according to Equation 3. This quantity can be

acts like step size

$\pi_{new} = \arg\min \mathbb{E}\left[\log \pi' - \underbrace{Q^{\pi_{old}}}_{\text{acts as gradient}} + \log Z \right]$ fixed normalisⁿ constant, ignore

ψ = value func. parameters (state-value)
θ = soft Q
ϕ = policy

r-Critic

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
  end for
end for