Continuous Control With Deep Reinforcement Learning

Reminders

- DQN takes processed observations as input and gives a score for each action as output
- Stochastic policy - e.g. bimodal distribution, or uncertainty in actuators
- Stochastic environment - e.g. uncertain state transitions
- [1] *Covariate shift* refers to the change in the input distribution
  – This affects the network parameters
  – These changes amplify as the network becomes deeper
  – *Solution*: Batch normalization. $\hat{y} = \gamma\hat{x} + \beta$, with learnable γ, β so it can even be converted back to the initial distribution
- Whitened i.i.d. inputs make the networks converge faster

Overview

- The same algorithm robustly solves 20+ *simulated* physics tasks, including classic control problems
- DQN is a good way to start, but cannot deal with continuous actions. Discretization would not work:
  – Curse of dimensionality
  – Naive way discards essential information
- Can sometimes exceed performance of the LQR planner, even only from pixel inputs
- Nonlinear function approximators $\implies$ no stability guarantees
- Performance compared to iLQG expert and naive (random) expert

Key ingridients

- Replay buffer and target network for stability taken from DQN
- Batch normalization
- Using actor-critic framework
  – Critic learns the Q function following the (fixed) policy μ, as in DQN (Bellman eq.), by minimising:

$$L(\theta^Q) = \mathrm{E}\Big[(Q(s_t, a_t | \theta^Q) - y_t)^2\Big],$$

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

  – Actor learns by optimizing over policy parameters:

$$\nabla_{\theta^\mu} J \approx \mathrm{E}\Big[\nabla_a Q(s_{t+1}, \mu(s_{t+1}))\nabla_{\theta^\mu}\mu(s_t | \theta^\mu)\Big]$$

- Soft update (for increased stability) of both target networks: $\theta \leftarrow \tau\theta_+ + (1-\tau)\theta,$ with $\tau \ll 1$
- Batch normalization is applied to almost all layers
- Exploration is treated independently from learning (possible since it is off-policy), by adding process noise $\mathcal{N}$ to the policy

Comments

- DDPG learns both the Q function and the policy separately
- Stability is a major concern here, since two networks have to work together, and both don’t have proven stability guarantees
- Less training steps needed (2.5 million) compared to DQN (around 50 million)

[1] *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, S. Ioffe, et. al.