Continuous control with deep reinforcement learning

Google Deepmind

Motivation

- DQN can only handle
  - discrete (not continuous)
  - low-dimensional action spaces
- Simple approach to adapt DQN to continuous domain is discretizing
  - 7 degree of freedom system with discretization $a_t \in \{-k, 0, k\}$
  - Now space dimensionality becomes $3^7 = 2187$
    - explosion of the number of discrete actions

Contribution

- Present a model-free, off-policy actor-critic algorithm
  - learn policies in high-dimensional, continuous action spaces
- Work based on DPG (Deterministic policy gradient)

Demo

- https://goo.gl/J4PIAz

Background

- Actions $a_t \in \mathbb{R}^N$, action space $\mathcal{A} = \mathbb{R}^N$
- Environment E will change to new state s_{t+1} and reward $r(s_t, a_t)$

- Agent's behavior: a policy $\pi: S \rightarrow \mathcal{P}(\mathcal{A})$
- Discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$
- Goal of RL is to learn a policy π which maximizes the expected return
  - from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$

Background (Cont'd)

- Action-value function
  - $\mathrm{Q}^{\pi}(\mathrm{s_t}, \mathrm{a_t}) = \mathbb{E}_{r_{i\geq t}, s_{i>t}\sim E, a_{i>t}\sim\pi}[R_t|s_t, a_t]$
  - Expected return after taking an action a^t in state s^t and following policy π.
- Bellman equation:
  - $\mathrm{Q}^{\pi}(\mathrm{s_t}, \mathrm{a_t}) = \mathbb{E}_{r_t, s_{t+1}\sim E}\big[r(s_t, a_t) + \gamma\mathbb{E}_{a_{t+1}\sim\pi}[\mathrm{Q}^{\pi}(\mathrm{s_{t+1}}, \mathrm{a_{t+1}})]\big]$
- With **deterministic** policy μ: $\mathcal{S} \rightarrow \mathcal{A}$
  - $\mathrm{Q}^{\mu}(\mathrm{s_t}, \mathrm{a_t}) = \mathbb{E}_{r_t, s_{t+1}\sim E}[r(s_t, a_t) + \gamma\mathrm{Q}^{\mu}(\mathrm{s_{t+1}}, \mu(s_{t+1}))]$

Background (Cont'd)

- Expectation only depends on the environment
  - possible to learn Q function

- Using function approximator parameterized by θ^Q
- Minimize the loss
  - $L(\theta^Q) = \mathbb{E}_{s_t, a_t, r_t}(Q(s_t, a_t|\theta^Q) - y_t)^2$
  - $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1}|\theta^Q)$

Deterministic Policy Gradient (DPG)

- Q-learning with greedy policy $\mu(s) = \arg\max_a Q(s, a)$
- In continuous space, finding the greedy policy requires an optimization of a_t at every time slot
  - too slow for large, unconstrained function approximators and nontrivial action spaces
- Instead, used an actor-critic approach based on the DPG algorithm
  - actor: $\mu(s|\theta^\mu)$: $\mathcal{S} \rightarrow \mathcal{A}$
  - critic: $Q(s, a|\theta^Q)$

DPG (Cont't)

- Actor is updated by applying the chain rule to the expected return from the distribution $\mathcal{J}$ w.r.t θ^{μ}

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

- Silver et al. (2014) proved that this is the *policy gradient*
  - the gradient of the policy's performance.
- Build Deep Neuron Networks for Actor and Critic

Some tricks used in algorithm

- Use replay buffer
- Use target network for stable learning but use “soft” target updates
- Use normalization to normalize each dimension such that different dimensions have unit mean
- Used an Ornstein-Uhlenbeck process to generate temporally correlated exploration
  - $\mu'(s_t) = \mu\left(s_t \middle| \theta_t^{\mu}\right) + \mathcal{N}$

Replay buffer

- NN for RL usually assume that the samples are i.i.d. but when the samples are generated from exploring sequentially in an environment, this assumption no longer holds.

- Use a buffer
  - Buffer all (s_t, a_t, r_t, s_{t+1}) transitions
  - Sample a batch of transitions to train the network

- Speed up training

Soft target update

- Training is not stable
- Duel networks for Actor and Critic

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 $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

- Use target networks for

Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

- When update the target network, use “soft” update

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

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

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:

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

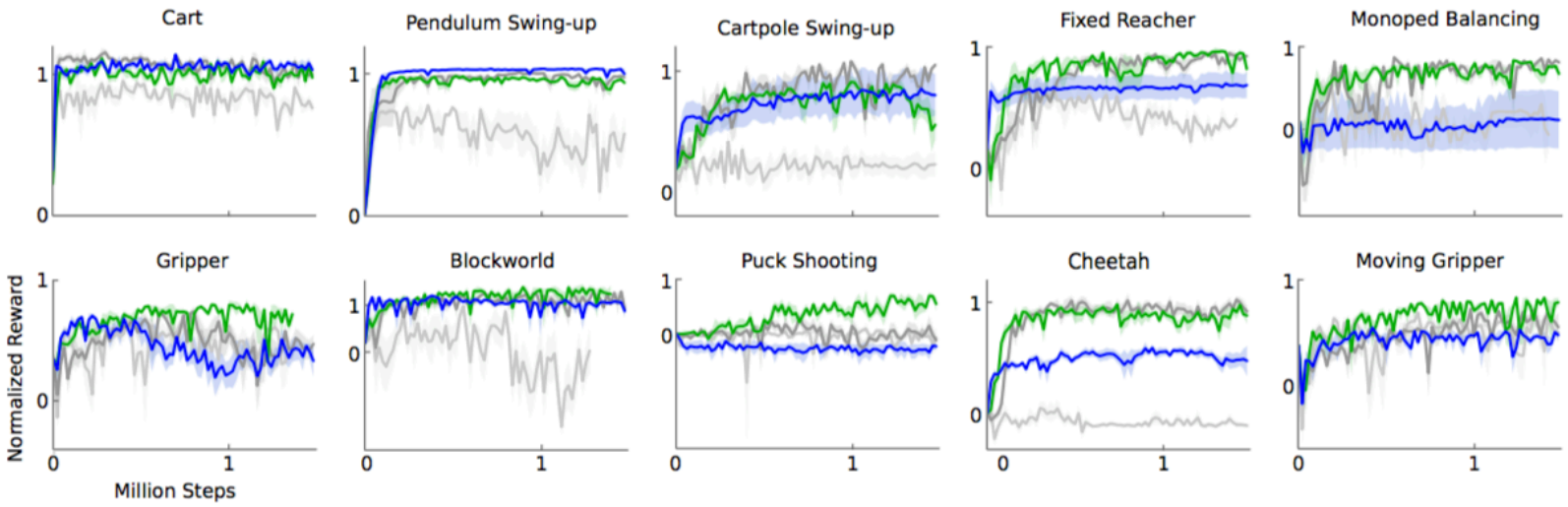
  end for
end for

Experiment details

- Using Adam Optimizer. With learning rates:
  - $lr^{\mu} = 10^{-4}, lr^{Q} = 10^{-3}$
- Discount factor: γ = 0.99
- τ = 0.001

- NN: 2 hidden layers with 400 and 300 units
- ReLU for hidden layers, tanh for output layer of the actor to bound the actions
- Replay buffer $|R| = 10^{6}$
- Ornstein-Uhlenbeck process: θ = 0.15, σ = 0.2
- Stop after about 2.5 million steps

Results



Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

Take away

- Tricks of how to train NN for Reinforcement Learning.
  - Replay buffer
  - Duel networks
  - Normalization
  - Noise process as exploration

- Continuous Action space, it is still unknown whether DDPG is suitable for Action space like $\mathbb{Z}^N$