Continuous control with deep reinforcement learning (DeepRL Reading, RDL, UQ)

Vektor Dewanto

vektor.dewanto@gmail.com

June 18, 2018

Outline

1. Introduction
2. Background
3. Methods
4. Setup
5. Results
6. Conclusions
7. References

1. Introduction
2. Background
3. Methods
4. Setup
5. Results
6. Conclusions
7. References

Introduction

On:
Continuous control with deep reinforcement learning (Lillicrap et al., 2015)

Problems:

- continuous action space
- raw pixels for observations

Deep DPG (=DPG+DQN) components:

- Deterministic Policy-Gradient (DPG) (Silver et al., 2014)
- Deep Q-Network (DQN) (Mnih et al., 2013)
- Actor-Critic Methods (Sutton & Barto, 1998)

Note: Deep DPG is gradient-based for continouos control (action), c.f. gradient-free, e.g. GPS-ABT (Seiler, Kurniawati, & Singh, 2015)

See: app video!

Introduction: vs Predictron (Silver et al., 2016)

Table: Predictron vs Deep DPG

Predictron	**Deep DPG**
yet another fn approx, nn arch	yet another actor-critic approach
deep model	deep policy and deep value
for model-based, model is abstract	for model-free
integrate planning and learning	integrate policy- and value-based
tested on MRPs	tested on MDPs

1. Introduction
2. **Background**
3. Methods
4. Setup
5. Results
6. Conclusions
7. References

Background: MDP

The environment, E:
a Markov decision process with a state space S, action space $A = \mathbb{R}^N$, an initial state distribution $p(s_1)$, transition dynamics $p(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$.

At each discrete timestep t,
the agent receives an observation $x_t = s_t$ (fully observable), takes an action a_t and receives a scalar reward r_t.

The return from a state:
$R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$ with a discounting factor $\gamma \in [0, 1]$.

Goal:
to learn a policy, $\pi : S \mapsto P(A)$, which maximizes $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$.

Action-value function:
$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi}[R_t | s_t, a_t]$.

Background: MDP with deterministic policy

Bellman equation (the recursive relationship):

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

If the target policy is deterministic, $\mu : S \mapsto A$, then:

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

Thus:

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

Background: Fn approximator

Consider fn approximators parameterized by θ^Q,
which we optimize by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a \sim \beta, r_t \sim E}\left[\left(Q(s_t, a_t|\theta^Q) - y_t\right)^2\right] \tag{3}$$

where:
$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$, and
ρ^β: the discounted state visitation distribution for a policy β.

Typically, ignore the fact that y_t is also dependent on θ^Q.

Background: Actor-critic approach

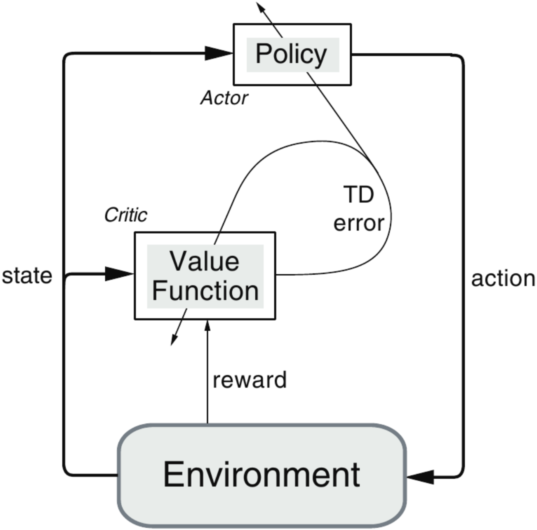
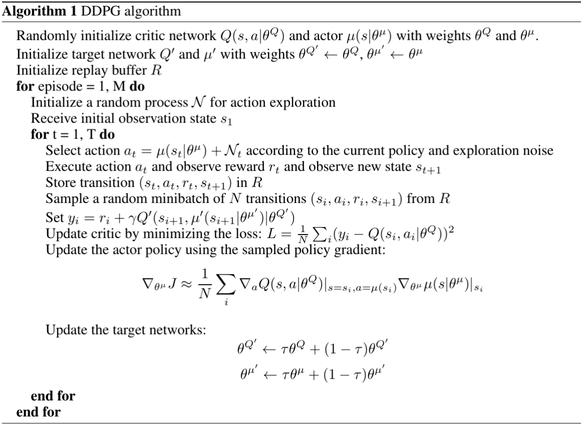


Figure 6.15 The actor–critic architecture.

1. Introduction
2. Background
3. **Methods**
4. Setup
5. Results
6. Conclusions
7. References

Methods: Deep DPG



Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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
```

Deep DPG (=DPG+DQN) components:

- Deterministic Policy-Gradient (DPG) (Silver et al., 2014)
- Deep Q-Network (DQN= QLearning + deepLearning) (Mnih et al., 2013)
- Actor-Critic Methods (Sutton & Barto, 1998)

Methods: DPG (Silver et al., 2014)

DPG: Deterministic Policy Gradient, an actor-critic approach:

- actor: $\mu(s|\theta^{\mu})$, specifies the current policy by deterministically mapping states to a specific action
- critic: $Q(s, a)$, learned using the Bellman equation as in Q-learning.

Update θ^{μ} by applying the chain rule to the expected return from the start distribution, J, with respect to the actor parameters, θ^{μ}:

$$\begin{aligned} \nabla_{\theta^{\mu}} J &\approx \mathbb{E}_{s_t \sim \rho^{\beta}} \left[\nabla_{\theta^{\mu}} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t|\theta^{\mu})} \right] \\ &= \mathbb{E}_{s_t \sim \rho^{\beta}} \left[\nabla_{\theta^{\mu}} Q(s, a|\theta^{Q})|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_t} \right] \end{aligned} \tag{4}$$

Equ. 4 is the policy gradient (the gradient of the policy's performance), in practice the discount in ρ^{β} is ignored.

Methods: Replay-buffer, as in DQN (Mnih et al., 2013)

Why:

- NeuralNets assume iid samples;
  the samples from exploring sequentially in an environment are **not** iid
- to make efficient use of hardware optimizations:
  to learn in minibatches rather than (fully) online

How:

- replay buffer, R, contains (s_t, a_t, r_t, s_{t+1}) sampled from E using an exploration policy
- At each timestep the actor and critic are updated by sampling a minibatch uniformly from the buffer

Methods: Fix target network, as in DQN (Mnih et al., 2013)

Why:

- Q-learning (eq.3) with NeuralNets is unstable, since the network $Q(s, a|\theta^{Q})$ being updated is also used in calculating the target value, y

How:

- create a copy of the actor and critic networks, i.e. $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$
- use $Q'(s, a|\theta^{Q'})$ and $\mu'(s|\theta^{\mu'})$ for calculating the target values
- target networks update:
  $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$ with $\tau \ll 1$
  (warn: may slow learning, since the target network delays the propagation of value estimations, but in practice, this was greatly outweighed by the stability of learning)

Methods: Feature scaling: batch normalization

Why:

- different components of the observation may have different physical units (for example, positions versus velocities)
- their ranges may vary across environments

How:

- normalizes each dimension across the samples in a minibatch to have unit mean and variance.
- maintains a running average of the mean and variance to use for normalization during testing (during xploration or evaluation)

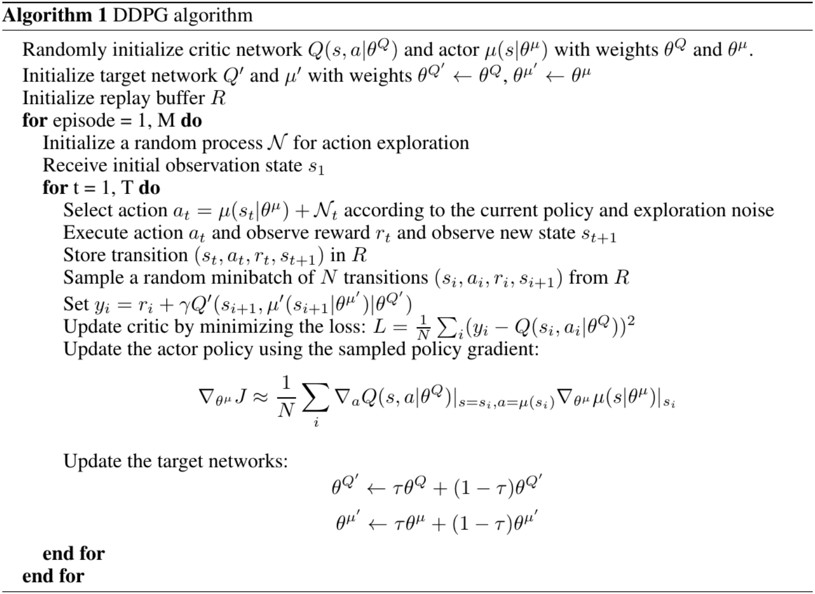
Methods: Exploration

Why:
Exploration-exploitation dillema

How:

- Construct an exploration policy μ^{xplor} by adding noise sampled from a noise process $\mathcal{N}$ to our actor policy: $\mu^{xplor} = \mu(s_t|\theta^{\mu}_t) + \mathcal{N}$.
- Here, $\mathcal{N}$ is from Ornstein-Uhlenbeck process,
  to generate temporally correlated exploration for exploration efficiency in physical control problems with inertia.

Methods: Deep DPG Algo



Algorithm 1 DDPG algorithm

```
Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target network $Q'$ and $\mu'$ 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
```

1. Introduction
2. Background
3. Methods
4. **Setup**
5. Results
6. Conclusions
7. References

Experiment Setup: Tasks

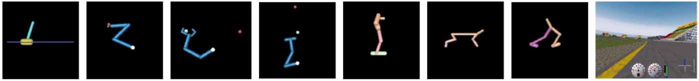


Figure 1: Example screenshots of a sample of environments we attempt to solve with DDPG. In order from the left: the cartpole swing-up task, a reaching task, a gasp and move task, a puck-hitting task, a monoped balancing task, two locomotion tasks and Torcs (driving simulator). We tackle all tasks using both low-dimensional feature vector and high-dimensional pixel inputs. Detailed descriptions of the environments are provided in the supplementary. Movies of some of the learned policies are available at `https://goo.gl/J4PIAz`

- 20 simulated physics task, including:
    - gripperRandom: Agent must use an arm with gripper appendage to grasp an object and manuver the object, which are initialized in random locations
    - reacherObstacle: Agent is required to move a 5-DOF arm around an obstacle to a randomized target position.
- features, both:
    - a low-dimensional state description (such as joint angles and positions)
    - high-dimensional renderings of the environment

Experiment Setup: Tasks

task name	$\dim(\mathbf{s})$	$\dim(\mathbf{a})$	$\dim(\mathbf{o})$
blockworld1	18	5	43
blockworld3da	31	9	102
canada	22	7	62
canada2d	14	3	29
cart	2	1	3
cartpole	4	1	14
cartpoleBalance	4	1	14
cartpoleParallelDouble	6	1	16
cartpoleParallelTriple	8	1	23
cartpoleSerialDouble	6	1	14
cartpoleSerialTriple	8	1	23
cheetah	18	6	17
fixedReacher	10	3	23
fixedReacherDouble	8	2	18
fixedReacherSingle	6	1	13
gripper	18	5	43
gripperRandom	18	5	43
hardCheetah	18	6	17
hardCheetahNice	18	6	17
hopper	14	4	14
hyq	37	12	37
hyqKick	37	12	37
movingGripper	22	7	49
movingGripperRandom	22	7	49
pendulum	2	1	3
reacher	10	3	23
reacher3daFixedTarget	20	7	61
reacher3daRandomTarget	20	7	61
reacherDouble	6	1	13
reacherObstacle	18	5	38
reacherSingle	6	1	13
walker2d	18	6	41

Table 2: Dimensionality of the MuJoCo tasks: the dimensionality of the underlying physics model $\dim(s)$, number of action dimensions $\dim(a)$ and observation dimensions $\dim(o)$.

Experiment Setup: Baseline, Assumptions

Assumption:

- env is MDP, the environment is fully-observed

Baseline:

- 2 types:
  - **naive policy:** the mean return from a naive policy which samples actions from a uniform distribution over the valid action space;
  - **iLQG (Todorov & Li, 2005)**: a planning based solver with full access to the underlying physical model and its derivatives.
- normalize baseline scores so that
  - **naive policy**: has a mean score of 0
  - **iLQG**: has a mean score of 1

1. Introduction
2. Background
3. Methods
4. Setup
5. **Results**
6. Conclusions
7. References

Results: Performance, 5 runs, score: naive= 0, iLQG= 1

environment	$R_{av,lowd}$	$R_{best,lowd}$	$R_{av,pix}$	$R_{best,pix}$	$R_{av,cntrl}$	$R_{best,cntrl}$
blockworld1	1.156	1.511	0.466	1.299	-0.080	1.260
blockworld3da	0.340	0.705	0.889	2.225	-0.139	0.658
canada	0.303	1.735	0.176	0.688	0.125	1.157
canada2d	0.400	0.978	-0.285	0.119	-0.045	0.701
cart	0.938	1.336	1.096	1.258	0.343	1.216
cartpole	0.844	1.115	0.482	1.138	0.244	0.755
cartpoleBalance	0.951	1.000	0.335	0.996	-0.468	0.528
cartpoleParallelDouble	0.549	0.900	0.188	0.323	0.197	0.572
cartpoleSerialDouble	0.272	0.719	0.195	0.642	0.143	0.701
cartpoleSerialTriple	0.736	0.946	0.412	0.427	0.583	0.942
cheetah	0.903	1.206	0.457	0.792	-0.008	0.425
fixedReacher	0.849	1.021	0.693	0.981	0.259	0.927
fixedReacherDouble	0.924	0.996	0.872	0.943	0.290	0.995
fixedReacherSingle	0.954	1.000	0.827	0.995	0.620	0.999
gripper	0.655	0.972	0.406	0.790	0.461	0.816
gripperRandom	0.618	0.937	0.082	0.791	0.557	0.808
hardCheetah	1.311	1.990	1.204	1.431	-0.031	1.411
hopper	0.676	0.936	0.112	0.924	0.078	0.917
hyq	0.416	0.722	0.234	0.672	0.198	0.618
movingGripper	0.474	0.936	0.480	0.644	0.416	0.805
pendulum	0.946	1.021	0.663	1.055	0.099	0.951
reacher	0.720	0.987	0.194	0.878	0.231	0.953
reacher3daFixedTarget	0.585	0.943	0.453	0.922	0.204	0.631
reacher3daRandomTarget	0.467	0.739	0.374	0.735	-0.046	0.158
reacherSingle	0.981	1.102	1.000	1.083	1.010	1.083
walker2d	0.705	1.573	0.944	1.476	0.393	1.397
torcs	-393.385	1840.036	-401.911	1876.284	-911.034	1961.600

Results: Estimated Q values

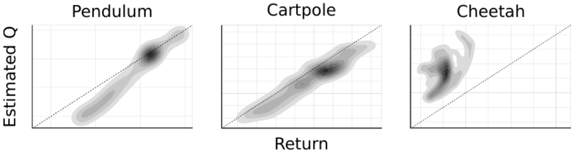


Figure 3: Density plot showing estimated Q values versus observed returns sampled from test episodes on 5 replicas. In simple domains such as pendulum and cartpole the Q values are quite accurate. In more complex tasks, the Q estimates are less accurate, but can still be used to learn competent policies. Dotted line indicates unity, units are arbitrary.

Results: In words...

- 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.
- for many of the tasks the algorithm, can learn policies end-to-end: directly from raw pixel inputs.
- in some simpler tasks, learning policies from pixels is just as fast as learning using the low-dimensional state descriptor.
  (may also be that the convolutional layers provide an easily separable representation of state space, which is straightforward for the higher layers to learn on quickly)

See: supplemental video!

1. Introduction
2. Background
3. Methods
4. Setup
5. Results
6. Conclusions
7. References

Conclusions

Deep DPG: DPG that uses neural network function approximators to learn in large state and action spaces online.

Limitations:

- requires a large number of training episodes to find solutions
- assume fully observability
- lack of obstacles
- no planning, since model-free

Follow up:

- Data-efficient Deep Reinforcement Learning for Dexterous Manipulation
- Benchmarking Deep Reinforcement Learning for Continuous Control
- DeepMind Control Suite

References I

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., . . . Wierstra, D. (2015). Continuous control with deep reinforcement learning. Retrieved from `http://arxiv.org/abs/1509.02971` doi: 10.1561/2200000006

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. , 1–9. Retrieved from `http://arxiv.org/abs/1312.5602` doi: 10.1038/nature14236

Seiler, K. M., Kurniawati, H., & Singh, S. P. N. (2015, May). An online and approximate solver for pomdps with continuous action space. In *2015 ieee international conference on robotics and automation (icra)* (p. 2290-2297). doi: 10.1109/ICRA.2015.7139503

References II

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 387–395.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., . . . Degris, T. (2016). The predictron: End-to-end learning and planning. *CoRR*, *abs/1612.08810*. Retrieved from `http://arxiv.org/abs/1612.08810`

Sutton, R. S., & Barto, A. G. (1998). *Introduction to reinforcement learning* (1st ed.). Cambridge, MA, USA: MIT Press.

Todorov, E., & Li, W. (2005, June). A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, american control conference, 2005.* (p. 300-306 vol. 1). doi: 10.1109/ACC.2005.1469949

Discussion time and thank you.