DDPG versus CMA-ES: a comparison

Olivier Sigaud
http://people.isir.upmc.fr/sigaud

Joint work with Arnaud de Froissard de Broissia

September 12, 2016
Motivation: why compare DDPG to CMA-ES?

- Towards “blind” policy search (CMA-ES) + DMPs (small domain)
- Requires DMP engineering
- In principle, actor-critic should be more data efficient
- But sensitive to value function approximation error
- DDPG brings accurate value function approximation and no feature engineering

Families of methods

- **Critic**: (action) value function → evaluation of the policy
- **Actor**: the policy itself
- **Critic-only methods**: iterates on the value function up to convergence without storing policy, then computes optimal policy. Typical examples: value iteration, Q-learning, Sarsa
- **Actor-only methods**: explore the space of policy parameters. Typical example: CMA-ES
- **Actor-critic methods**: update in parallel one structure for the actor and one for the critic. Typical examples: policy iteration, many AC algorithms
- **Q-learning and Sarsa** look for a global optimum, AC looks for a local one
Quick history

- Those methods proved inefficient for robot RL

Main messages

- All the processes rely on efficient backpropagation in deep networks
- DDPG is gradient-based, this improves efficiency
- Gradient calculation involves some averaging that is somewhat related to reward-weighted averaging in BB methods
DDPG: The paper

- Continuous control with deep reinforcement learning
- Timothy P. Lillicrap Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra
- Google Deepmind
- On arXiv since September 7, 2015
- Already cited >45 times

DDPG: ancestors

- DQN: Atari domain, Nature paper, small discrete actions set
- Most of the actor-critic theory for continuous problem is for stochastic policies (policy gradient theorem, compatible features, etc.)


General architecture

- Any neural network structure
- Actor parametrized by $w$, critic by $\theta$
- All updates based on backprop, available in TensorFlow, theano… (RProp, RMSProp, Adagrad, Adam?)
The Q-network in DQN

- Requires one output neuron per action
- Select action by picking the max
DDPG versus CMA-ES: a comparison

- Introduction: motivation
- Explaining DDPG

The critic in DDPG

- Used to update an actor
- Background: DDPG more sample efficient than CMA-ES
- But does not solve the exploration issue
Training the critic

In DPG (and RL in general), the critic should minimize the RPE:

\[ \delta_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1})|\theta) - Q(s_t, a_t|\theta) \]

- We want to minimize the critic error using backprop on critic weights \( \theta \)
- Error = difference between “some target value” and network output \( Q(s_t, a_t|\theta) \)
- Thus, given \( N \) samples \( \{s_i, a_i, r_i, s_{i+1}\} \), compute \( y_i = r_i + \gamma Q(s_{i+1}, \pi(s_{i+1})|\theta') \)
- The target value for sample \( i \) is \( y_i \), minimizing the error minimizes \( \delta_i \)
- So update \( \theta \) by minimizing the loss function (i.e. squared error) over the batch

\[ L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta))^2 \]  (1)
Training the actor

Deterministic policy gradient theorem: the true policy gradient is
\[ \nabla_w \pi(s, a) = \mathbb{E}_{\rho(s)}[\nabla_a Q(s, a|\theta) \nabla_w \pi(s|w)] \tag{2} \]

\[ \nabla_a Q(s, a|\theta) \] is obtained by computing the gradient over actions of \( Q(s, a|\theta) \) in the critic.

The gradient over actions is similar to the gradient over weights (symmetric roles of weights and inputs)

\[ \nabla_a Q(s, a|\theta) \] is used as an error signal to update the actor’s weights through backprop again.

Comes from NFQCA

DDPG versus CMA-ES: a comparison

Introduction: motivation

Explaining DDPG

General algorithm

1. Feed the actor with the state, outputs the action
2. Feed the critic with the state and action, determines $Q(s, a|\theta^Q)$
3. Update the critic, using (1) (alternative: do it after 4?)
4. Compute $\nabla_a Q(s, a|\theta)$
5. Update the actor, using (2)
Subtleties

- The actor update rule is

\[
\nabla_w \pi(s_i) \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta)|_{s=s_i, a=\pi(s_i)} \nabla_w \pi(s)|_{s=s_i}
\]

- Thus we do not use the action in the samples to update the actor.

- Could it be

\[
\nabla_w \pi(s_i) \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta)|_{s=s_i, a=a_i} \nabla_w \pi(s)|_{s=s_i}?
\]

- Work on \( \pi(s_i) \) instead of \( a_i \)

- Does this make the algorithm on-policy instead of off-policy?

- Does this make a difference?
Trick 1: Sample buffer (from DQN)

- In most optimization algorithms, samples are assumed independently and identically distributed (iid)
- Obviously, this is not the case of behavioral samples \((s_i, a_i, r_i, s_{i+1})\)
- Idea: put the samples into a buffer, and extract them randomly
- Use training minibatches, to make profit of GPU
- The replay buffer management policy is an issue

---

Trick 2: Stable Target Q-function (from DQN)

- Compute the critic loss function from a separate target network $Q'(\ldots | \theta')$
- So compute $y_i = r_i + \gamma Q'(s_{i+1}, \pi(s_{i+1}) | \theta')$
- In DQN, the $\theta$ is updated after each batch
- In DDPG, they rather allow for slow evolution of $Q'$ and $\pi'$

$$\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$$

- The same applies to $\mu$, $\mu'$
- From the empirical study, this is the critical trick
Trick 3: Batch Normalization (new)

- Covariate shift: as layer $N$ is trained, the input distribution of layer $N + 1$ is shifted, which makes learning harder
- To fight covariate shift, ensure that each dimension across the samples in a minibatch have unit mean and variance at each layer
- Add a buffer between each layer, and normalize all samples in these buffers
- Makes learning easier and faster
- Makes the algorithm more domain-insensitive
- But poor theoretical grounding, and makes network computation slower

Algorithm

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 $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}$ 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 gradient:
        \[
        \nabla_{\theta^\mu} \mu|_{s_i} \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{a=s, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}
        \]
        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

- Notice the slow $Q'$ and $\pi'$ updates (instead of copying as in DQN)
Applications: impressive results

- End-to-end policies (from pixels to control)
- Works impressively well on “More than 20” (27-32) such domains
- Coded with MuJoCo (Todorov) / TORCS
Comparison

- Based on the mountain car benchmark
- Cost = squared acceleration per time step
- Reward if goal reached
- Very simple actor: two input, one output, 2 hidden layers with 20 and 10 neurons respectively
- No batch normalization nor weight normalization
DDPG versus CMA-ES: a comparison

Empirical comparison to CMA-ES

Performances

Collected reward

Length of an episode

- \( X = \) number of calls to simulator, \( Y = \) number of steps to reach goal, average over 10 trials
- The time to compute both results is similar
- This illustrates that DDPG is much more sample efficient
Final policies

- Similar trajectories
- DDPG has a more complex policy
Scalability

From 51 parameters to 281
- No visible effects on DDPG
- Slower convergence for CMA-ES
Influence of minibatches

More training at each steps

- Faster convergence
Second Experiment : Application partially observable task

- The task : Collectball
- Information through sensors : incomplete information
- DDPG : no memory of previous observations
- Complexe environement : The goal requires several sub-goals to be completed

Methods

We used four different methods:

- Direct application of DDPG (reward when a ball is collected, 0 otherwise)
- DDPG with bootstrap
- DDPG with an improved reward (positive reward for picking up a ball, negative reward when dropping a ball, and when not moving)
- DDPG with an horizon of 3 observations and an improved reward
First results and analysis

Direct application

Improved reward

Bootstrap

3 observations

- No ball collected
- Partial observability does not prevent interesting behaviors
- Exploration limits
DDPG is more sample efficient than CMA-ES (and probably other black-box optimization algorithms)

- Analytic gradient descent versus stochastic gradient-free search
- Better reuse of samples

But DDPG is limited by its exploration power

- Noise on actions
- Gradient of increasing rewards
Need for better exploration

- DDPG still looks for a local minimum, like any actor-critic method
- DDPG does not help to find scarce rewards (the needle in the stack): no specific exploration
- Source of randomness in CMA-ES: drawing the samples
- Source of randomness in DDPG: exploration noise in the policy
- Despite the name, stochastic gradient descent (SGD) is not a source of exploration
- Exploration noise in the policy: in DDPG, action perturbation rather than policy parameter perturbation
- In previous work, we have shown that the latter performs better in gradient-free methods
- Get inspired by diversity search in evolutionary techniques


Approximate the advantage function

- Other option: encode the advantage function
  \[ A_\theta(s_i, a_i) = Q(s_i, a_i | \theta) - \max_a Q(s_i, a | \theta) \]
- Very good recent paper
- Or see GProp...


Back to natural gradient

- Batch normalization
- Weight normalization
- Natural Neural networks


DDPG versus CMA-ES: a comparison
- Empirical comparison to CMA-ES
- Improving DDPG: alternatives for the critic

Any question?
Compatible value gradients for reinforcement learning of continuous deep policies.

The importance of experience replay database composition in deep reinforcement learning.
_In Deep RL workshop at NIPS 2015._

Natural neural networks.
_In Advances in Neural Information Processing Systems_ (pp. 2062–2070).

Continuous deep q-learning with model-based acceleration.

Reinforcement learning in feedback control.
_Machine learning, 84_(1-2), 137–169._

Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Continuous control with deep reinforcement learning.

Human-level control through deep reinforcement learning.
_Nature, 518_(7540), 529–533._


