Continuous Actions, DDPG, TD3

Milan Straka

November 14, 2022
Parallel Advantage Actor Critic

An alternative to independent workers is to train in a synchronous and centralized way by having the workers to only generate episodes. Such approach was described in May 2017 by Clemente et al., who named their agent parallel advantage actor-critic (PAAC).

Figure 1 of “Efficient Parallel Methods for Deep Reinforcement Learning” by Alfredo V. Clemente et al.
Algorithm 1 Parallel advantage actor-critic

1: Initialize timestep counter $N = 0$ and network weights $\theta, \theta_v$
2: Instantiate set $e$ of $n_e$ environments
3: repeat
4: \hspace{1em} for $t = 1$ to $t_{\text{max}}$ do
5: \hspace{2em} Sample $a_t$ from $\pi(a_t|s_t; \theta)$
6: \hspace{2em} Calculate $v_t$ from $V(s_t; \theta_v)$
7: \hspace{2em} parallel for $i = 1$ to $n_e$ do
8: \hspace{3em} Perform action $a_{t,i}$ in environment $e_i$
9: \hspace{3em} Observe new state $s_{t+1,i}$ and reward $r_{t+1,i}$
10: \hspace{2em} end parallel for
11: \hspace{1em} end for
12: $R_{t_{\text{max}}+1} = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_{t_{\text{max}}+1}; \theta) & \text{for non-terminal } s_t \end{cases}$
13: for $t = t_{\text{max}}$ down to 1 do
14: \hspace{1em} $R_t = r_t + \gamma R_{t+1}$
15: \hspace{1em} end for
16: $d\theta = \frac{1}{n_e \cdot t_{\text{max}}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{\text{max}}} (R_{t,i} - v_{t,i}) \nabla_\theta \log \pi(a_{t,i}|s_{t,i}; \theta) + \beta \nabla_\theta H(\pi(s_{e,t}; \theta))$
17: $d\theta_v = \frac{1}{n_e \cdot t_{\text{max}}} \sum_{i=1}^{n_e} \sum_{t=1}^{t_{\text{max}}} \nabla_{\theta_v} (R_{t,i} - V(s_{t,i}; \theta_v))^2$
18: Update $\theta$ using $d\theta$ and $\theta_v$ using $d\theta_v$.
19: $N \leftarrow N + n_e \cdot t_{\text{max}}$
20: until $N \geq N_{\text{max}}$

Algorithm 1 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.
The authors use 8 workers, \( n_e = 32 \) parallel environments, 5-step returns, \( \gamma = 0.99 \), \( \epsilon = 0.1 \), \( \beta = 0.01 \), and a learning rate of \( \alpha = 0.0007 \cdot n_e = 0.0224 \).

The \( \text{arch}_{\text{nips}} \) is from A3C: 16 filters \( 8 \times 8 \) stride 4, 32 filters \( 4 \times 4 \) stride 2, a dense layer with 256 units. The \( \text{arch}_{\text{nature}} \) is from DQN: 32 filters \( 8 \times 8 \) stride 4, 64 filters \( 4 \times 4 \) stride 2, 64 filters \( 3 \times 3 \) stride 1 and 512-unit fully connected layer. All nonlinearities are ReLU.

<table>
<thead>
<tr>
<th>Game</th>
<th>Gorila</th>
<th>A3C FF</th>
<th>GA3C</th>
<th>PAAC arch(_{\text{nips}})</th>
<th>PAAC arch(_{\text{nature}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amidar</td>
<td>1189.70</td>
<td>263.9</td>
<td>218</td>
<td>701.8</td>
<td>1348.3</td>
</tr>
<tr>
<td>Centipede</td>
<td>8432.30</td>
<td>3755.8</td>
<td>7386</td>
<td>5747.32</td>
<td>7368.1</td>
</tr>
<tr>
<td>Beam Rider</td>
<td>3302.9</td>
<td>22707.9</td>
<td>N/A</td>
<td>4062.0</td>
<td>6844.0</td>
</tr>
<tr>
<td>Boxing</td>
<td>94.9</td>
<td>59.8</td>
<td>92</td>
<td>99.6</td>
<td>99.8</td>
</tr>
<tr>
<td>Breakout</td>
<td>402.2</td>
<td>681.9</td>
<td>N/A</td>
<td>470.1</td>
<td>565.3</td>
</tr>
<tr>
<td>Ms. Pacman</td>
<td>3233.50</td>
<td>653.7</td>
<td>1978</td>
<td>2194.7</td>
<td>1976.0</td>
</tr>
<tr>
<td>Name This Game</td>
<td>6182.16</td>
<td>10476.1</td>
<td>5643</td>
<td>9743.7</td>
<td>14068.0</td>
</tr>
<tr>
<td>Pong</td>
<td>18.3</td>
<td>5.6</td>
<td>18</td>
<td>20.6</td>
<td>20.9</td>
</tr>
<tr>
<td>Qbert</td>
<td>10815.6</td>
<td>15148.8</td>
<td>14966.0</td>
<td>16561.7</td>
<td>17249.2</td>
</tr>
<tr>
<td>Seaquest</td>
<td>13169.06</td>
<td>2355.4</td>
<td>1706</td>
<td>1754.0</td>
<td>1755.3</td>
</tr>
<tr>
<td>Space Invaders</td>
<td>1883.4</td>
<td>15730.5</td>
<td>N/A</td>
<td>1077.3</td>
<td>1427.8</td>
</tr>
<tr>
<td>Up n Down</td>
<td>12561.58</td>
<td>74705.7</td>
<td>8623</td>
<td>88105.3</td>
<td>100523.3</td>
</tr>
<tr>
<td>Training</td>
<td>4d CPU cluster</td>
<td>4d CPU</td>
<td>1d GPU</td>
<td>12h GPU</td>
<td>15h GPU</td>
</tr>
</tbody>
</table>

Table 1 of “Efficient Parallel Methods for Deep Reinforcement Learning” by Alfredo V. Clemente et al.
Parallel Advantage Actor Critic

Figure 3 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.
Parallel Advantage Actor Critic

Figure 4 of "Efficient Parallel Methods for Deep Reinforcement Learning" by Alfredo V. Clemente et al.
Parallel Advantage Actor Critic

Efficiency

- arch_{nips} GPU
- arch_{nature} GPU
- arch_{nips} CPU
- arch_{nature} CPU

Environment Interaction

- arch_{nips} GPU
- arch_{nature} GPU
- arch_{nips} CPU
- arch_{nature} CPU

Model Interaction

- arch_{nips} GPU
- arch_{nature} GPU
- arch_{nips} CPU
- arch_{nature} CPU

Figure 2 of “Efficient Parallel Methods for Deep Reinforcement Learning” by Alfredo V. Clemente et al.
Continuous Action Space

Until now, the actions were discrete. However, many environments naturally accept actions from continuous space. We now consider actions which come from range $[a, b]$ for $a, b \in \mathbb{R}$, or more generally from a Cartesian product of several such ranges:

$$\prod_i [a_i, b_i].$$

A simple way how to parametrize the action distribution is to choose them from the normal distribution. Given mean $\mu$ and variance $\sigma^2$, probability density function of $\mathcal{N}(\mu, \sigma^2)$ is

$$p(x) \overset{\text{def}}{=} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$
Utilizing continuous action spaces in gradient-based methods is straightforward. Instead of the softmax distribution, we suitably parametrize the action value, usually using the normal distribution.

Considering only one real-valued action, we therefore have

\[ \pi(a|s; \theta) \overset{\text{def}}{=} P \left( a \sim \mathcal{N} \left( \mu(s; \theta), \sigma(s; \theta)^2 \right) \right), \]

where \( \mu(s; \theta) \) and \( \sigma(s; \theta) \) are function approximation of mean and standard deviation of the action distribution.

The mean and standard deviation are usually computed from the shared representation, with

- the mean being computed as a common regression (i.e., one output neuron without activation);
- the standard deviation (which must be positive) being computed again as a single neuron, but with either \( \exp \) or \( \text{softplus} \), where \( \text{softplus}(x) \overset{\text{def}}{=} \log(1 + e^x) \).
Continuous Action Space in Gradient Methods

During training, we compute $\mu(s; \theta)$ and $\sigma(s; \theta)$ and then sample the action value (clipping it to $[a, b]$ if required). To compute the loss, we utilize the probability density function of the normal distribution (and usually also add the entropy penalty).

```python
mus = tf.keras.layers.Dense(actions)(hidden_layer)
sds = tf.keras.layers.Dense(actions)(hidden_layer)
sds = tf.math.exp(sds)  # or sds = tf.math.softplus(sds)

action_dist = tfp.distributions.Normal(mus, sds)

# Loss computed as $- \log \pi(a|s) \times \text{returns} - \text{entropy} \_\text{regularization}$
loss = - action_dist.log_prob(actions) * returns - \
    - args.entropy_regularization * action_dist.entropy()
```
Continuous Action Space

When the action consists of several real values, i.e., action is a suitable subregion of $\mathbb{R}^n$ for $n > 1$, we can:

- either use multivariate Gaussian distribution;
- or factorize the probability into a product of univariate normal distributions.

Modeling the action distribution using a single normal distribution might be insufficient, in which case a mixture of normal distributions is usually used.

Sometimes, the continuous action space is used even for discrete output – when modeling pixels intensities (256 values) or sound amplitude ($2^{16}$ values), instead of a softmax we use discretized mixture of distributions, usually logistic (a distribution with a sigmoid CDF). Then,

$$
\pi(a) = \sum_i p_i \left( \sigma \left( \frac{a + 0.5 - \mu_i}{\sigma_i} \right) - \sigma \left( \frac{a - 0.5 - \mu_i}{\sigma_i} \right) \right).
$$

However, such mixtures are usually used in generative modeling, not in reinforcement learning.
Combining continuous actions and Deep Q Networks is not straightforward. In order to do so, we need a different variant of the policy gradient theorem.

Recall that in policy gradient theorem,

$$\nabla \theta J(\theta) \propto \mathbb{E}_{s \sim \mu} \left[ \sum_{a \in A} q_{\pi}(s, a) \nabla \pi(a | s; \theta) \right].$$

### Deterministic Policy Gradient Theorem

Assume that the policy $\pi(s; \theta)$ is deterministic and computes an action $a \in \mathbb{R}$. Further, assume the reward $r(s, a)$ is actually a deterministic function of the given state-action pair. Then, under several assumptions about continuousness, the following holds:

$$\nabla \theta J(\theta) \propto \mathbb{E}_{s \sim \mu} \left[ \nabla \theta \pi(s; \theta) \nabla a q_{\pi}(s, a) \bigg|_{a = \pi(s; \theta)} \right].$$

The theorem was first proven in the paper Deterministic Policy Gradient Algorithms by David Silver et al in 2014.
The proof is very similar to the original (stochastic) policy gradient theorem. However, we will be exchanging derivatives and integrals, for which we need several assumptions:

- we assume that $h(s), p(s'|s, a), \nabla_a p(s'|s, a), r(s, a), \nabla_a r(s, a), \pi(s; \theta), \nabla_\theta \pi(s; \theta)$ are continuous in all parameters and variables;
- we further assume that $h(s), p(s'|s, a), \nabla_a p(s'|s, a), r(s, a), \nabla_a r(s, a)$ are bounded.

Details (which assumptions are required and when) can be found in Appendix B of the paper *Deterministic Policy Gradient Algorithms: Supplementary Material* by David Silver et al.
Deterministic Policy Gradient Theorem – Proof

\[ \nabla_\theta v_\pi(s) = \nabla_\theta q_\pi(s, \pi(s; \theta)) \]

\[ = \nabla_\theta \left( r(s, \pi(s; \theta)) + \int_{s'} \gamma p(s' | s, \pi(s; \theta)) \left( v_\pi(s') \right) ds' \right) \]

\[ = \nabla_\theta \pi(s; \theta) \nabla_a r(s, a) \bigg|_{a=\pi(s; \theta)} + \nabla_\theta \int_{s'} \gamma p(s' | s, \pi(s; \theta)) v_\pi(s') ds' \]

\[ = \nabla_\theta \pi(s; \theta) \nabla_a \left( r(s, a) + \int_{s'} \gamma p(s' | s, a) v_\pi(s') ds' \right) \bigg|_{a=\pi(s; \theta)} \]

\[ + \int_{s'} \gamma p(s' | s, \pi(s; \theta)) \nabla_\theta v_\pi(s') ds' \]

\[ = \nabla_\theta \pi(s; \theta) \nabla_a q_\pi(s, a) \bigg|_{a=\pi(s; \theta)} + \int_{s'} \gamma p(s' | s, \pi(s; \theta)) \nabla_\theta v_\pi(s') ds' \]

We finish the proof as in the gradient theorem by continually expanding \( \nabla_\theta v_\pi(s') \), getting

\[ \nabla_\theta v_\pi(s) = \int_{s'} \sum_{k=0}^{\infty} \gamma^k P(s \rightarrow s' \text{ in } k \text{ steps} | \pi) \left[ \nabla_\theta \pi(s'; \theta) \nabla_a q_\pi(s', a) \bigg|_{a=\pi(s'; \theta)} \right] ds' \]

and then

\[ \nabla_\theta J(\theta) = \mathbb{E}_{s \sim h} \nabla_\theta v_\pi(s) \propto \mathbb{E}_{s \sim \mu} \left[ \nabla_\theta \pi(s; \theta) \nabla_a q_\pi(s, a) \bigg|_{a=\pi(s; \theta)} \right]. \]
Deep Deterministic Policy Gradients

Note that the formulation of deterministic policy gradient theorem allows an off-policy algorithm, because the loss functions no longer depends on actions (similarly to how expected Sarsa is also an off-policy algorithm).

We therefore train function approximation for both $\pi(s; \theta)$ and $q(s, a; \theta)$, training $q(s, a; \theta)$ using a deterministic variant of the Bellman equation:

$$ q(S_t, A_t; \theta) = \mathbb{E}_{S_{t+1}} \left[ r(S_t, A_t) + \gamma q(S_{t+1}, \pi(S_{t+1}; \theta)) \right] $$

and $\pi(s; \theta)$ according to the deterministic policy gradient theorem.

The algorithm was first described in the paper Continuous Control with Deep Reinforcement Learning by Timothy P. Lillicrap et al. (2015).

The authors utilize a replay buffer, a target network (updated by exponential moving average with $\tau = 0.001$), batch normalization for CNNs, and perform exploration by adding a Ornstein-Uhlenbeck noise to the predicted actions. Training is performed by Adam with learning rates of $1e^{-4}$ and $1e^{-3}$ for the policy and critic network, respectively.
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 \( \text{episode} = 1, M \) 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

Algorithm 1 of "Continuous Control with Deep Reinforcement Learning" by Timothy P. Lillicrap et al.
Figure 2: 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.
Deep Deterministic Policy Gradients

Results using low-dimensional (lowd) version of the environment, pixel representation (pix) and DPG reference (cntrl).

Table 1 of “Continuous Control with Deep Reinforcement Learning” by Timothy P. Lillicrap et al.

<table>
<thead>
<tr>
<th>environment</th>
<th>$R_{av,lowd}$</th>
<th>$R_{best,lowd}$</th>
<th>$R_{av,pix}$</th>
<th>$R_{best,pix}$</th>
<th>$R_{av,cntrl}$</th>
<th>$R_{best,cntrl}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>blockworld1</td>
<td>1.156</td>
<td>1.511</td>
<td>0.466</td>
<td>1.299</td>
<td>-0.080</td>
<td>1.260</td>
</tr>
<tr>
<td>blockworld3da</td>
<td>0.340</td>
<td>0.705</td>
<td>0.889</td>
<td>2.225</td>
<td>-0.139</td>
<td>0.658</td>
</tr>
<tr>
<td>canada</td>
<td>0.303</td>
<td>1.735</td>
<td>0.176</td>
<td>0.688</td>
<td>0.125</td>
<td>1.157</td>
</tr>
<tr>
<td>canada2d</td>
<td>0.400</td>
<td>0.978</td>
<td>-0.285</td>
<td>0.119</td>
<td>-0.045</td>
<td>0.701</td>
</tr>
<tr>
<td>cart</td>
<td>0.938</td>
<td>1.336</td>
<td>1.096</td>
<td>1.258</td>
<td>0.343</td>
<td>1.216</td>
</tr>
<tr>
<td>cartpole</td>
<td>0.844</td>
<td>1.115</td>
<td>0.482</td>
<td>1.138</td>
<td>0.244</td>
<td>0.755</td>
</tr>
<tr>
<td>cartpoleBalance</td>
<td>0.951</td>
<td>1.000</td>
<td>0.335</td>
<td>0.996</td>
<td>-0.468</td>
<td>0.528</td>
</tr>
<tr>
<td>cartpoleParallelDouble</td>
<td>0.549</td>
<td>0.900</td>
<td>0.188</td>
<td>0.323</td>
<td>0.197</td>
<td>0.572</td>
</tr>
<tr>
<td>cartpoleSerialDouble</td>
<td>0.272</td>
<td>0.719</td>
<td>0.195</td>
<td>0.642</td>
<td>0.143</td>
<td>0.701</td>
</tr>
<tr>
<td>cartpoleSerialTriple</td>
<td>0.736</td>
<td>0.946</td>
<td>0.412</td>
<td>0.427</td>
<td>0.583</td>
<td>0.942</td>
</tr>
<tr>
<td>cheetah</td>
<td>0.903</td>
<td>1.206</td>
<td>0.457</td>
<td>0.792</td>
<td>-0.008</td>
<td>0.425</td>
</tr>
<tr>
<td>fixedReacher</td>
<td>0.849</td>
<td>1.021</td>
<td>0.693</td>
<td>0.981</td>
<td>0.259</td>
<td>0.927</td>
</tr>
<tr>
<td>fixedReacherDouble</td>
<td>0.924</td>
<td>0.996</td>
<td>0.872</td>
<td>0.943</td>
<td>0.290</td>
<td>0.995</td>
</tr>
<tr>
<td>fixedReacherSingle</td>
<td>0.954</td>
<td>1.000</td>
<td>0.827</td>
<td>0.995</td>
<td>0.620</td>
<td>0.999</td>
</tr>
<tr>
<td>gripper</td>
<td>0.655</td>
<td>0.972</td>
<td>0.406</td>
<td>0.790</td>
<td>0.461</td>
<td>0.816</td>
</tr>
<tr>
<td>gripperRandom</td>
<td>0.618</td>
<td>0.937</td>
<td>0.082</td>
<td>0.791</td>
<td>0.557</td>
<td>0.808</td>
</tr>
<tr>
<td>hardCheetah</td>
<td>1.311</td>
<td>1.990</td>
<td>1.204</td>
<td>1.431</td>
<td>-0.031</td>
<td>1.411</td>
</tr>
<tr>
<td>hopper</td>
<td>0.676</td>
<td>0.936</td>
<td>0.112</td>
<td>0.924</td>
<td>0.078</td>
<td>0.917</td>
</tr>
<tr>
<td>hyq</td>
<td>0.416</td>
<td>0.722</td>
<td>0.234</td>
<td>0.672</td>
<td>0.198</td>
<td>0.618</td>
</tr>
<tr>
<td>movingGripper</td>
<td>0.474</td>
<td>0.936</td>
<td>0.480</td>
<td>0.644</td>
<td>0.416</td>
<td>0.805</td>
</tr>
<tr>
<td>pendulum</td>
<td>0.946</td>
<td>1.021</td>
<td>0.663</td>
<td>1.055</td>
<td>0.099</td>
<td>0.951</td>
</tr>
<tr>
<td>reacher</td>
<td>0.720</td>
<td>0.987</td>
<td>0.194</td>
<td>0.878</td>
<td>0.231</td>
<td>0.953</td>
</tr>
<tr>
<td>reacher3dFixedTarget</td>
<td>0.585</td>
<td>0.943</td>
<td>0.453</td>
<td>0.922</td>
<td>0.204</td>
<td>0.631</td>
</tr>
<tr>
<td>reacher3dRandomTarget</td>
<td>0.467</td>
<td>0.739</td>
<td>0.374</td>
<td>0.735</td>
<td>-0.046</td>
<td>0.158</td>
</tr>
<tr>
<td>reacherSingle</td>
<td>0.981</td>
<td>1.102</td>
<td>1.000</td>
<td>1.083</td>
<td>1.010</td>
<td>1.083</td>
</tr>
<tr>
<td>walker2d</td>
<td>0.705</td>
<td>1.573</td>
<td>0.944</td>
<td>1.476</td>
<td>0.393</td>
<td>1.397</td>
</tr>
<tr>
<td>torcs</td>
<td>-393.385</td>
<td>1840.036</td>
<td>-401.911</td>
<td>1876.284</td>
<td>-911.034</td>
<td>1961.600</td>
</tr>
</tbody>
</table>
While the exploration policy could just use Gaussian noise, the authors claim that temporarily-correlated noise is more effective for physical control problems with inertia. They therefore generate noise using Ornstein-Uhlenbeck process, by computing

\[ n_t \gets n_{t-1} + \theta \cdot (\mu - n_{t-1}) + \varepsilon \sim \mathcal{N}(0, \sigma^2), \]

utilizing hyperparameter values \( \tau = 0.15 \) and \( \sigma = 0.2 \).
Figure 4. Example MuJoCo environments (a) HalfCheetah-v1, (b) Hopper-v1, (c) Walker2d-v1, (d) Ant-v1.
The paper Addressing Function Approximation Error in Actor-Critic Methods by Scott Fujimoto et al. from February 2018 proposes improvements to DDPG which

- decrease maximization bias by training two critics and choosing the minimum of their predictions;
- introduce several variance-lowering optimizations:
  - delayed policy updates;
  - target policy smoothing.

The TD3 algorithm has been together with SAC one of the best algorithms for off-policy continuous-actions RL training (as of 2022).
TD3 – Maximization Bias

Similarly to Q-learning, the DDPG algorithm suffers from maximization bias. In Q-learning, the maximization bias was caused by the explicit max operator. For DDPG methods, it can be caused by the gradient descent itself. Let $\theta_{\text{approx}}$ be the parameters maximizing the $q_\theta$ and let $\theta_{\text{true}}$ be the hypothetical parameters which maximise true $q_\pi$, and let $\pi_{\text{approx}}$ and $\pi_{\text{true}}$ denote the corresponding policies.

Because the gradient direction is a local maximizer, for sufficiently small $\alpha < \varepsilon_1$ we have

$$\mathbb{E}[q_\theta(s, \pi_{\text{approx}})] \geq \mathbb{E}[q_\theta(s, \pi_{\text{true}})].$$

However, for real $q_\pi$ and for sufficiently small $\alpha < \varepsilon_2$, it holds that

$$\mathbb{E}[q_\pi(s, \pi_{\text{true}})] \geq \mathbb{E}[q_\pi(s, \pi_{\text{approx}})].$$

Therefore, if $\mathbb{E}[q_\theta(s, \pi_{\text{true}})] \geq \mathbb{E}[q_\pi(s, \pi_{\text{true}})]$, for $\alpha < \min(\varepsilon_1, \varepsilon_2)$

$$\mathbb{E}[q_\theta(s, \pi_{\text{approx}})] \geq \mathbb{E}[q_\pi(s, \pi_{\text{approx}})].$$
Analogously to Double DQN we could compute the learning targets using the current policy and the target critic, i.e., $r + \gamma q_{\theta'}(s', \pi_{\varphi}(s'))$ (instead of using the target policy and the target critic as in DDPG), obtaining DDQN-AC algorithm. However, the authors found out that the policy changes too slowly and the target and current networks are too similar.

Using the original Double Q-learning, two pairs of actors and critics could be used, with the learning targets computed by the opposite critic, i.e., $r + \gamma q_{\theta_2}(s', \pi_{\varphi_1}(s'))$ for updating $q_{\theta_1}$.

The resulting DQ-AC algorithm is slightly better, but still suffering from overestimation.
The authors instead suggest to employ two critics and one actor. The actor is trained using one of the critics, and both critics are trained using the same target computed using the *minimum* value of both critics as

$$ r + \gamma \min_{i=1,2} q_{\theta_i'}(s', \pi_{\phi'}(s')). $$

The resulting algorithm is called CDQ – Clipped Double Q-learning.

Furthermore, the authors suggest two additional improvements for variance reduction.

- For obtaining higher quality target values, the authors propose to train the critics more often. Therefore, critics are updated each step, but the actor and the target networks are updated only every $d$-th step ($d = 2$ is used in the paper).

- To explicitly model that similar actions should lead to similar results, a small random noise is added to the performed actions when computing the target value:

$$ r + \gamma \min_{i=1,2} q_{\theta_i'}(s', \pi_{\phi'}(s')) + \varepsilon \text{ for } \varepsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c). $$
Algorithm 1 TD3

Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network $\pi_\phi$
with random parameters $\theta_1, \theta_2, \phi$
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $B$

for $t = 1$ to $T$ do

Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon,$
$\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
Store transition tuple $(s, a, r, s')$ in $B$

Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $B$
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
y $\leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
if $t \mod d$ then

Update $\phi$ by the deterministic policy gradient:
$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a = \pi_\phi(s)} \nabla_\phi \pi_\phi(s)$

Update target networks:
$\theta'_i \leftarrow \tau \theta_i + (1 - \tau) \theta'_i$
$\phi' \leftarrow \tau \phi + (1 - \tau) \phi'$
end if

end for

Algorithm 1 of "Addressing Function Approximation Error in Actor-Critic Methods" by Scott Fujimoto et al.
## TD3 – Algorithm

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Ours</th>
<th>DDPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critic Learning Rate</td>
<td>$10^{-3}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Critic Regularization</td>
<td>None</td>
<td>$10^{-2} \cdot</td>
</tr>
<tr>
<td>Actor Learning Rate</td>
<td>$10^{-3}$</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>Actor Regularization</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
<td>Adam</td>
</tr>
<tr>
<td>Target Update Rate ($\tau$)</td>
<td>$5 \cdot 10^{-3}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Batch Size</td>
<td>100</td>
<td>64</td>
</tr>
<tr>
<td>Iterations per time step</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Discount Factor</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>Reward Scaling</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Normalized Observations</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>Gradient Clipping</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Exploration Policy</td>
<td>$\mathcal{N}(0, 0.1)$</td>
<td>OU, $\theta = 0.15$, $\mu = 0$, $\sigma = 0.2$</td>
</tr>
</tbody>
</table>

*Table 3 of “Addressing Function Approximation Error in Actor-Critic Methods” by Scott Fujimoto et al.*
Figure 5 of “Addressing Function Approximation Error in Actor-Critic Methods” by Scott Fujimoto et al.

Table 1 of “Addressing Function Approximation Error in Actor-Critic Methods” by Scott Fujimoto et al.
The AHE is the authors' reimplementation of DDPG using updated architecture, hyperparameters, and exploration. TPS is Target Policy Smoothing, DP is Delayed Policy update, and CDQ is Clipped Double Q-learning.
## TD3 – Ablations

<table>
<thead>
<tr>
<th>Method</th>
<th>H Cheetah</th>
<th>Hopper</th>
<th>Walker2d</th>
<th>Ant</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD3</td>
<td>9532.99</td>
<td>3304.75</td>
<td>4565.24</td>
<td>4185.06</td>
</tr>
<tr>
<td>DDPG</td>
<td>3162.50</td>
<td>1731.94</td>
<td>1520.90</td>
<td>816.35</td>
</tr>
<tr>
<td>AHE</td>
<td>8401.02</td>
<td>1061.77</td>
<td>2362.13</td>
<td>564.07</td>
</tr>
<tr>
<td>AHE + DP</td>
<td>7588.64</td>
<td>1465.11</td>
<td>2459.53</td>
<td>896.13</td>
</tr>
<tr>
<td>AHE + TPS</td>
<td>9023.40</td>
<td>907.56</td>
<td>2961.36</td>
<td>872.17</td>
</tr>
<tr>
<td>AHE + CDQ</td>
<td>6470.20</td>
<td>1134.14</td>
<td>3979.21</td>
<td>3818.71</td>
</tr>
<tr>
<td>TD3 - DP</td>
<td>9590.65</td>
<td>2407.42</td>
<td>4695.50</td>
<td>3754.26</td>
</tr>
<tr>
<td>TD3 - TPS</td>
<td>8987.69</td>
<td>2392.59</td>
<td>4033.67</td>
<td>4155.24</td>
</tr>
<tr>
<td>TD3 - CDQ</td>
<td>9792.80</td>
<td>1837.32</td>
<td>2579.39</td>
<td>849.75</td>
</tr>
<tr>
<td>DQ-AC</td>
<td>9433.87</td>
<td>1773.71</td>
<td>3100.45</td>
<td>2445.97</td>
</tr>
<tr>
<td>DDQN-AC</td>
<td><strong>10306.90</strong></td>
<td>2155.75</td>
<td>3116.81</td>
<td>1092.18</td>
</tr>
</tbody>
</table>

*Table 2 of “Addressing Function Approximation Error in Actor-Critic Methods” by Scott Fujimoto et al.*