Deep Deterministic Policy Gradients

Edward Young - ey245@cam.ac.uk

Quick facts

1. DDPG is a (deterministic) *policy gradient method*.
2. DDPG is *off-policy*.
3. DDPG is *model-free*.
4. DDPG is applicable to any kind of state space but only *continuous* action spaces.
5. DDPG is an *actor-critic* method, this means that it learns both a policy (an *actor*) and a value function (a *critic*).

High-level strategy

DDPG uses both policy networks and value networks, making it an **actor-critic** method. As with most actor critic methods, the value network (the critic) is trained in a manner similar to the method used in DQN - by using bootstrapping to form regression targets and then minimising a mean squared error. The policy network (the actor) is trained to output actions that are better, according to the value network (the critic). The function of the value network is to critique the action selection of the policy network .

Networks

As with DQN, we will use two networks which approximate the action-value function:

1. The **value network**, $Q(S, A; \phi)$ which uses the parameters ϕ.
2. The **target value network**, $Q(S, A; \phi^-)$ which has the same architecture as the value network, but uses a lagged set of parameters ϕ^-.

Note that unlike DQN, both these networks take in *state-action pairs* and output a single value, rather than taking in states and outputting the value for each action in that state. In addition to two networks for action-value function approximation, we will also use two networks which take in states and output an action for each state:

1. The **policy network**, $\mu(S; \theta)$, which uses parameters θ.
2. The **target policy network**, $\mu(S; \theta^-)$, which has the same architecture as the policy network, but uses a lagged set of parameters θ^-

Training the base networks

We now address updating the parameters of the value network ϕ and of the policy network θ. To do this, we sample a **batch** of N transitions from a **memory replay buffer** (just like with DQN). A transition consists of a state, action, reward, and next state, (S, A, R, S').

The parameters of the value network will be learned by performing gradient descent on the **mean squared error loss**:

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{i=1}^{N} \left(Q(S_i, A_i; \phi) - y_i \right)^2 \tag{1}$$

where the **regression targets** y_i are formed via the following formula:

$$y_i = R_i + \gamma Q(S'_i, \mu(S'_i; \theta^-); \phi^-) \tag{2}$$

The regression targets are formed using the target networks. Note that this is almost the same formula that we used for the regression targets in DQN, except that the next-step action is chosen according to the target policy rather than by taking a maximum.

The parameters of the policy network are learned by performing gradient ascent on the **mean action value objective**:

$$J(\theta) = \frac{1}{N}\sum_{n=1}^{N} Q(S_i, \mu(S_i;\theta);\phi^-) \tag{3}$$

This encourages the policy network to output actions that have a higher value, according to the target value function. This choice of objective is justified by the **Deterministic Policy Gradient Theorem**, which shows that maximising this objective approximates maximising the total reward accumulated across the episode.

Updating the target networks

To update the target networks we will use **Polyak averaging**. This means we take a weighted average of the current target network parameters and the current base network parameters. The update rules are:

$$\phi^- \leftarrow \rho\phi^- + (1-\rho)\phi,\ \theta^- \leftarrow \rho\theta^- + (1-\rho)\theta \tag{4}$$

ρ is the **Polyak parameter**, with $1-\rho$ analogous to the **learning rate**. We perform Polyak averaging whenever we update the base network parameters. Note that Polyak averaging with $\rho = 1$ results in no learning, whereas when $\rho = 0$ we recover the **synchronisation** rule we used for DQN.

Trick: Noisy exploration policy

During training time, we want to use a policy which *explores* the environment, generating diverse experience for us to learn from. We do this by adding random noise to our policy network output. Action selection during training is therefore implemented by:

$$S \mapsto \mu(S;\theta) + \mathcal{N}(0,\sigma^2) \tag{5}$$

σ^2 controls the exploration level, with larger σ^2 indicating more exploration. It is analogous to ϵ for ϵ-greedy action selection. We will also clip the action to be within the action space.

Conclusion: Putting it all together

DDPG proceeds through the following loop:

1. Gather experience using an **noisy exploration policy** (5) and load that experience into the **experience replay buffer**.
2. Every 10 interaction steps, randomly sample a **batch** of N transitions from the replay buffer.
   (a) To train the value network form **regression targets** (2) and minimise the **mean squared error loss** (1).
   (b) To train the policy network, maximise the **mean action value objective** (3).
3. Every time you update the base parameters, perform **Polyak averaging** (4) to update the target network parameters.