DiGrad: Multi-Task Reinforcement Learning with Shared Actions

Parijat Dewangan^{1*}**, Phaniteja S**^{1*}**, K Madhava Krishna**^{1}**, Abhishek Sarkar**^{1}**,**
Balaraman Ravindran^{2}
^{1} Robotics Research Center, International Institute of Information Technology, Hyderabad
^{2} Indian Institute of Technology, Madras
{ parijat10, phaniteja.sp}@gmail.com, { mkrishna, abhishek.sarkar}@iiit.ac.in,
ravi@cse.iitm.ac.in

Abstract

Most reinforcement learning algorithms are inefficient for learning multiple tasks in complex robotic systems, where different tasks share a set of actions. In such environments a compound policy may be learnt with shared neural network parameters, which performs multiple tasks concurrently. However such compound policy may get biased towards a task or the gradients from different tasks negate each other, making the learning unstable and sometimes less data efficient. In this paper, we propose a new approach for simultaneous training of multiple tasks sharing a set of common actions in continuous action spaces, which we call as DiGrad (**Di**fferential Policy **Grad**ient). The proposed framework is based on differential policy gradients and can accommodate multi-task learning in a single actor-critic network. We also propose a simple heuristic in the differential policy gradient update to further improve the learning. The proposed architecture was tested on 8 link planar manipulator and 27 degrees of freedom(DoF) Humanoid for learning multi-goal reachability tasks for 3 and 2 end effectors respectively. We show that our approach supports efficient multi-task learning in complex robotic systems, outperforming related methods in continuous action spaces.

1 Introduction

There has been an increasing demand for reinforcement learning (RL)[Sutton and Barto, 1998] in the fields of robotics and intelligent systems. Reinforcement learning deals with learning actions in a given environment to achieve a goal. Classic reinforcement learning techniques make use of linear approximation or tabular methods to learn such correlation[Konidaris *et al.*, 2011].

With the advancements of deep neural networks in the recent times, learning non-linear approximations and feature extraction has becomes much simpler. It was believed that non-linear approximators like neural network are hard to train in reinforcement learning scenario. However recent advancements in RL has successfully combined the deep neural networks with RL and stabilized the learning process. Deep Q Networks(DQN) [Mnih *et al.*, 2015] used Convolutional neural networks(CNN) and fully connected layers to make the RL agents learn to play the ATARI games. Following the success of DQN, several improvements on this architecture like Double DQN[Van Hasselt *et al.*, 2016], Prioritized Replay[Schaul *et al.*, 2015], Duelling Network[Wang *et al.*, 2015] are proposed which propelled the use of Deep RL in multi-agents. [Lillicrap *et al.*, 2015] proposed Deep Deterministic Policy Gradient(DDPG) for continuous control tasks which further extended the scope of Deep RL applications to robotics.

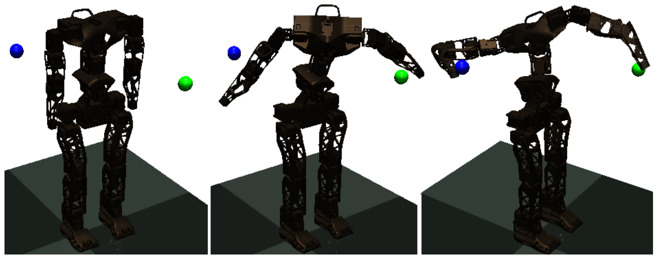


Figure 1: Humanoid robot multi-tasking to reach two goals simultaneously. The two goals are shown using blue and green coloured balls.

Robotic systems like open and closed kinematic chains offer fresh perspectives to employ deep RL algorithms. [Gu *et al.*, 2017] applied DDPG framework to learn manipulation tasks in complex 3D environments. Following this, [Phaniteja, 2017] applied DDPG to learn reachability tasks in Humanoid robot. Even though they are not multi-agent systems, they can be posed as multi-tasking systems where there are shared actions (common kinematic chains). As shown in Fig. 1, the spine/torso is the common chain which contributes to the reachability tasks of both the hands in humanoid robot. In this paper, we propose a novel framework called DiGrad based on differential policy gradients to learn multi-tasking in such robotic systems where different tasks may share a set of common actions.

There are several mathematical approaches which try to address the problem of multi-tasking in branched manipulators.

*These authors contributed equally.

However, classical methods based on Jacobian[Buss, 2004; Møller, 1993] has very limited capability in branched manipulators. Methods like Augmented Jacobian[Siciliano, 1990; Chiaverini, 1997] have constrained solution spaces, while methods based on optimization[Klein *et al.*, 1995] often doesn't provide real time control. Hence, RL based solvers are of great use in this domain which can sample the entire solution space and learn a real time controller in such complex robotic systems.

One direction for learning multiple tasks in such scenarios is to use DDPG to learn a compound policy taking care of all the tasks. However we found DDPG to be unstable for such multi-task scenarios. DiGrad addresses this problems by using differential policy gradient updates. We test our framework on branched manipulators shown in Fig. 3 for learning reachability tasks of multiple end effectors simultaneously. The proposed framework shows substantial improvement over DDPG and is considerably robust on all the experiments we conducted.

The rest of the paper is organised as follows. Section 2 and 3 discusses the related works and background. Section 4 explains the mathematical background behind the proposed framework and provides the detailed algorithm. Finally, Section 5 and 6 contain the experimental results and discussion respectively.

2 Related Works

Most of the multi-task reinforcement learning algorithms rely on transfer learning approaches. [Lazaric, 2012] shows a good collection of these methods. Some of the recent works based on this approach are [Rusu *et al.*, 2015; Yin and Pan, 2017]. Some works by [Borsa *et al.*, 2016] and [Zhang *et al.*, 2016] explored learning universal abstractions of state-action pairs or feature successors.

Apart from transfer learning, some works like [Lazaric and Ghavamzadeh, 2010], [Dimitrakakis and Rothkopf, 2011] investigated joint training of multiple value functions or policies. In a deep neural network setting, [Teh *et al.*, 2017] provided a framework for simultaneous training of multiple stochastic policies and a distilled master policy. Unlike our work, [Teh *et al.*, 2017] uses multiple networks for each policy and one more network for the distilled policy. In our work, we show how we can use a single network to learn multiple deterministic policies simultaneously.

All the above mentioned methods assume multi-agent scenario whereas in our paper, we concentrate on learning multiple tasks in a robotic system. Some very recent works in this scenario are [Yang *et al.*, 2017] and [Kulkarni *et al.*, 2016]. These works do not talk about the actions which are shared among different tasks, thus limiting their applicability. Unlike these frameworks, we explore the case of multi-task learning in branched manipulator which have shared action-spaces.

3 Background

We consider a standard reinforcement learning setup consisting of an agent interacting with an environment E in discrete time steps. At each time step t, the agent takes a state $s_t \in S$ as the input, performs an action $a_t \in A$ according to a policy, $\mu : S \rightarrow A$ and receives a reward $r_t \in R$. We assume a fully observable environment and model it as a Markov decision process with state space S, action space $A = \mathbb{R}^N$, an initial state distribution $p(s_1)$, state transition dynamics $p(s_{t+1}|s_t, a_t)$ and a reward function $r(s_t, a_t)$.

The goal in reinforcement learning is to learn a policy μ which maximizes the expected return $R_t = \sum_{t>0} \gamma^t r(s_t, a_t)$, where $\gamma \in [0, 1]$ is the discount factor. Since the return depends on the action taken, the policy may be stochastic but in our work we consider only deterministic policies. Hence the discounted state visitation distribution for a policy μ is denoted as ρ^μ.

The action-value function used in many reinforcement learning algorithms is described as the expected return after taking an action a_t in state s_t and thereafter following the given policy:

$$Q^\mu(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$$

DDPG is an off policy learning algorithm that uses an actor-critic based framework for continuous control tasks. In DDPG, both actor and critic are approximated using neural networks (θ^μ, θ^Q). The problem of instability in training is addressed by using target networks ($\theta^{\mu'}$, $\theta^{Q'}$) and experience replay using a replay buffer. In this setting, the critic network weights are optimized by minimizing the following loss:

$$L(\theta^Q) = (Q(s_t, a_t|\theta^Q) - y_t)^2 \quad (1)$$

where,

$$y_t = r(s_t, a_t) + \gamma Q'(s_{t+1}, {\mu'}_{t+1}(s_{t+1}|\theta^{\mu'})|\theta^{Q'}) \quad (2)$$

The gradient ascent update on policy network (actor) is given using Deterministic Policy Gradient(DPG) theorem([Silver *et al.*, 2014]). Suppose the learning rate is η, then:

$$\theta^\mu = \theta^\mu + \eta \nabla_a Q(s_t, a_t|\theta^Q)|_{a=\mu(s_t|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu) \quad (3)$$

Finally the updates on target networks are:

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

where, $\tau << 1$.

In the proposed framework, we use some of the basic concepts of DDPG and use DPG theorem to derive a policy gradient update which can support robust multi-task learning. Finally we explain how the learning can be improved by using a simple heuristic in case of shared actions.

4 DiGrad: Differential Policy Gradients

We propose a framework for simultaneous reinforcement learning of multiple tasks shared actions in continuous domain. The method is based on differential action-value updates in actor-critic based framework using DPG theorem. The framework learns a compound policy which optimally performs all the shared tasks simultaneously. Fig. 2 shows the higher level description of the method. In this section, we describe the mathematical framework behind this work.

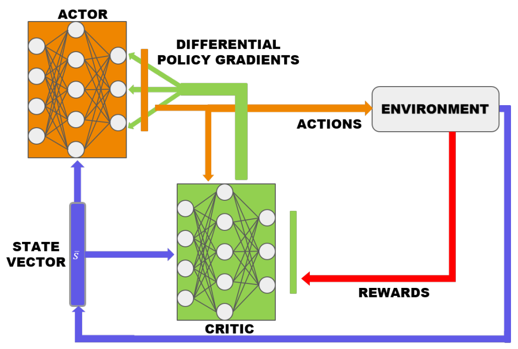


Figure 2: Overview of the algorithm showing differential policy gradient update

4.1 Environment setting

Consider n tasks in a standard RL setting and their corresponding n action spaces A_1, A_2, ..., A_n. We will assume that the state space S is the same across all the tasks and an infinite horizon with discount factor γ. Let A be a compound action space which performs the given set of n tasks simultaneously. Let $a \in A$ denote compound actions, $a_i \in A_i$ denote actions and $s \in S$ denote states. Therefore the relation between the compound actions and all the n actions is given by,

$$a = \bigcup_{i=1}^{n} a_i.$$

Suppose a_s corresponds to the set of actions which affects k of the n tasks, then a_s is given by

$$a_s = \bigcap_{j=1}^{k} a_j \qquad \text{where,} \quad a_j \subset a \tag{4}$$

The reward functions for the i^{th} task depend only on the corresponding actions a_i. Therefore, we denote the reward function as $r_i(s, a_i)$ for each task i; let $Q_i(s, a_i)$ be the corresponding action value function. Let μ be the compound deterministic policy and μ_i be the task-specific deterministic policies; therefore $\mu_i \subset \mu$, where

$$\mu_i(s) = a_i \quad \text{and} \quad \mu(s) = a$$

4.2 Proposed framework

An actor-critic based framework is proposed for multi-task learning in the given environment setting. We use a compound policy $\mu(s)$ parametrized by θ^{μ}, instead of multiple policy networks for each policy μ_i.

$$\mu : S \rightarrow A$$

A simple parametrization for action-value functions would be to have a separate network θ^{Q_i} for each $Q_i(s, a_i)$, which outputs an action-value corresponding to the i^{th} task. Another approach for modelling action-value function is to have a single network parametrized by θ^{Q} which outputs action-values for all the tasks. Here, Q_i is the action value corresponding to the i^{th} task. [Yang *et al.*, 2017] showed that a single critic network is sufficient in multi-policy learning. In this setting, the penultimate layer weights for each Q_i are updated based on only the reward $r_i(s, a_i)$ obtained by performing the corresponding action a_i. The remaining shared network captures the correlation between the different actions a_i. Hence this kind of parametrization is more useful in the case of shared actions. Apart from this, the number of parameters are significantly reduced by the use of a single network.

4.3 Critic update

Consider a single actor-critic based framework where critic $Q(s, a)$ is given by function approximators parametrized by θ^{Q} and the actor $\mu(s)$ is parametrized by θ^{μ}. Let the corresponding target networks be parametrized by $\theta^{\mu'}$, $\theta^{Q'}$. Since we have multiple critic outputs Q_i, we optimize θ^{Q} by minimizing the loss as given by [Yang *et al.*, 2017]:

$$L(\theta^{Q}) = \sum_{i=1}^{n} (Q_i(s_t, a_{it}|\theta^{Q}) - y_{it})^2 \tag{5}$$

where the target y_{it} is

$$y_{it} = r_i(s_t, a_{it}) + \gamma Q'(s_{t+1}, {\mu'}_{t+1}(s_{t+1}|\theta^{\mu'})|\theta^{Q'}) \tag{6}$$

If there are multiple critic networks, network parameters are optimized by minimizing the corresponding loss:

$$L(\theta^{Q_i}) = (Q_i(s_t, a_{it}|\theta^{Q}) - y_{it})^2 \tag{7}$$

In both the settings of critic, there is only a single actor which learns the compound policy μ. The differential policy gradient update on the compound policy is explained in the next subsection.

4.4 Differential Policy Gradient

Each task has a corresponding reward, $r_i(s, \mu_i(s))$ and hence to learn all the tasks we need to maximize the expected reward for each of the task with respect to the corresponding action, a_i. Therefore the performance objective to be maximized is:

$$J(\mu, \{\mu_i\}_{i=1}^{n}) = \sum_{i=1}^{n} \mathbb{E}_{s\sim\rho^{\beta}}[r_i(s, \mu_i(s))] \tag{8}$$

The update on the parametrized compound policy $\mu(s|\theta^{\mu})$ is given by applying deterministic policy gradient theorem on Eq. (8). The resulting update is:

$$\nabla_{\theta^{\mu}} J \approx \sum_{i=1}^{n} \mathbb{E}_{s\sim\rho^{\beta}}[\nabla_{\theta^{\mu}} Q_i(s, a_i|\theta^{Q})|_{a_i=\mu_i(s|\theta^{\mu})}]$$

$$= \sum_{i=1}^{n} \mathbb{E}_{s\sim\rho^{\beta}}[\nabla_{a_i} Q_i(s, a_i|\theta^{Q})|_{a_i=\mu_i(s|\theta^{\mu})} \nabla_{\theta^{\mu}} \mu_i(s|\theta^{\mu})] \tag{9}$$

DiGrad with shared tasks

In the above environment setting, let all the n tasks share a common set of actions a_s, i.e., $k = n$. Let a_1, a_2, ..., a_k be the actions corresponding to these k-tasks. Therefore from Eq. (4),

$$a_s = \bigcap_{j=1}^{k} a_j$$

Now the compound action a can be written as:

$$a = \{a_1 \cup a_2 ... \cup a_k\}$$

$$= \{\{a_1 \setminus a_s\} \cup \{a_2 \setminus a_s\} ... \cup \{a_k \setminus a_s\} \cup a_s\}$$

$$= \{a_1^d \cup a_2^d ... \cup a_k^d \cup a_s\}$$

where, $a_j^d = \{a_j \setminus a_s\}$

Here we can see that a_1^d, a_2^d, a_k^d, a_s are disjoint sets. Therefore we can write a_i = $[a_i^d, a_s]$. Similarly, $\mu_i = [\mu_i^d, \mu_s]$. From here onwards, to make it succinct we drop $s \sim \rho^\beta$ from the subscript of $\mathbb{E}_{s\sim\rho^\beta}$ and simply represent it as $\mathbb{E}$.

Substituting these in Eq. (9):

$$\nabla_{\theta^\mu} J \approx \sum_{i=1}^{k} \mathbb{E}[\nabla_{[a_i^d, a_s]} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu), a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu}[\mu_i^d(s|\theta^\mu), \mu_s(s|\theta^\mu)]]$$

On expanding with respect to gradient operator we get,

$$= \sum_{i=1}^{k} \mathbb{E} \begin{bmatrix} \nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \\ \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \end{bmatrix}^T \begin{bmatrix} \nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu) \\ \nabla_{\theta^\mu}\mu_s(s|\theta^\mu) \end{bmatrix}$$

$$= \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu) + \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_s(s|\theta^\mu)]$$

$$= \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu)] + \mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_s(s|\theta^\mu)] \tag{10}$$

We can see that the second term on R.H.S of Eq.(10) will be zero if all the action spaces are disjoint, that is, $a_s = \emptyset$. Hence this framework can be used even when there are no shared actions. Since the update on the actor are the sum of gradients of different action values, we call this a differential gradient update. It is different from the standard gradient update where an actor is updated based on a single action value [Lillicrap *et al.*, 2015].

Heuristic of direction

From Eq.(10), we can see that the policy gradient update for the policy(μ_s) of shared action a_s is taken as sum of the gradients of action values corresponding to the tasks it affects, whereas for policy(μ_j^d) of other actions a_j^d, the gradient update is taken as the gradient of only the corresponding Q_j. Thus, this uneven scaling of gradients may lead to delayed convergence and sometimes biasing. In order to equally scale all the gradient updates, we take the average value of the gradients obtained from the different Q-values for the shared action a_s. This modification will not affect the direction of gradient, only the magnitude will be scaled. By applying this heuristic, the differential gradient update on shared actions becomes:

$$\nabla_{\theta^\mu} J \approx \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu)] + \frac{1}{k}\mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_s(s|\theta^\mu)] \tag{11}$$

Generalisation

Suppose there are k tasks which share a set of action a_s as above and $n - k$ tasks which are independent, with corresponding actions $a_{k+1}, ..., a_n$, then Eq. (11) can be written as:

$$\nabla_{\theta^\mu} J \approx \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu)] + \frac{1}{k}\mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_s(s|\theta^\mu)] + \sum_{i=k+1}^{n} \mathbb{E}[\nabla_{a_i} Q_i(s, a_i|\theta^Q)|_{a_i=\mu_i(s|\theta^\mu)} \nabla_{\theta^\mu}\mu_i(s|\theta^\mu)] \tag{12}$$

From Eq. (12), we can observe that the policy gradient update for the policy (μ_s) of the shared action set a_s is the average of the gradients of the action-value functions of all the tasks it affects.

This can be easily extended to cases where there are more than one set of shared tasks. Our framework can accommodate heterogeneous dependent action spaces as compared to related multi-task RL algorithms which assume that action spaces are homogeneous or independent or both. This demonstrates the wider applicability of our framework.

4.5 Algorithm

In this section we explain the algorithm to learn multiple tasks using DiGrad. The flow of the algorithm is very similar to standard DDPG but there are significant differences in terms of the critic and actor updates as shown in the previous subsections. In DiGrad, compound action a is executed in the environment which returns a vector of rewards $\vec{r}_t$ corresponding to each task instead of a single reward. The replay buffer B stores the current state s_t, compound action a_t, observed state after executing action s_{t+1} and the vector of rewards $\vec{r}_t$. The entire flow of the algorithm is shown in Algorithm 1.

5 Experiments and Results

The proposed framework was tested in different settings in order to analyse the advantages of each setting. We considered four different network settings for DiGrad as follows:
(1) Single critic network with heuristics
(2) Single critic network without heuristics
(3) Multi critic network with heuristics
(4) Multi critic network without heuristics.

Algorithm 1 Multi-task learning using DiGrad

```
1: Randomly initialise actor ($\mu(s|\theta^{\mu})$) and critic network
   ($Q(s, a|\theta^{Q})$) with weights $\theta^{\mu}$ and $\theta^{Q}$.
2: Initialize the target network with weights
   $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$.
3: for $i$ = 1 to E$_{max}$
4:     Initialise random noise $N$ for exploration.
5:     Reset the environment to get initial state $s_1$.
6:     for $t$ = 1 to Step$_{max}$
7:         Get action $a_t$ = $\mu(s_t|\theta^{\mu}) + N$.
8:         Execute action $a_t$ and get corresponding reward
           vector $\vec{r}_t$ and updated state $s_{t+1}$.
9:         Store transition $(s_t, a_t, \vec{r}_t, s_{t+1})$ in replay buffer $B$.
10:        Randomly sample a mini-batch $M$ from replay
           buffer $B$.
11:        Update critic $\theta^{Q}$ according to Eqs.(5), (6), (7).
           Update actor policy $\theta^{\mu}$ according to Eq.(12)
12:        Update the target networks $\theta^{\mu'}$ and $\theta^{Q'}$
13:    end for
14: end for
```

We compare all of them with a standard DDPG setting. We use the same set of hyper parameters in all the five settings. The critic network architecture is the same for both single and multiple critic case in all aspects except in the number of outputs. The actor network parameters are also same for all the cases. We show the comparison of average reward as well the mean scores of each task in all the plots. Note that the average reward curves for DDPG are not shown as the reward function settings for DDPG is different than that of DiGrad.

In order to test the proposed multi-task learning framework, we considered two different environments. In both these environments, training involved learning reachability tasks for all the end effectors simultaneously, i.e., learning a policy on the joint space trajectories to reach any point in their workspace. For all the experiments, we define error and score for a particular task i as,

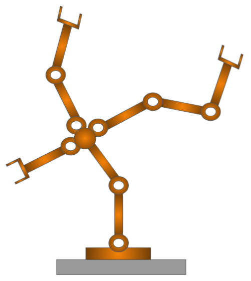
$$error_i = ||G_i - E_i||, \quad score_i = -log(error_i)$$

where G_i and E_i are the coordinates of goal and end-effector of the i_{th} chain.

In all the experiments, agents were implemented using TensorFlow Code base consisting of 2 fully connected layers. RMSProp optimizer is used to train both actor and critic networks with learning rates 0.001 and 0.0001 respectively. We used CReLu activation in all the hidden layers. While training, a normally distributed decaying noise function was used for exploration. Whereas, while testing this noise is omitted. We set the discount factor to be 0.99 in all the settings. In all the tasks, we use low-dimensional state description which include joint angles, joint positions and goal positions. The actor output is a set of angular velocities $\dot{q}$. Hence the policy learns a mapping from the configuration space to joint velocity space.

Reward function

The reward function for DiGrad settings is modelled keeping in mind the multi-task application. As defined before, r_i is the reward corresponding to the action a_i of the i_{th} task. We give a small positive reward to r_i if task i is finished. Also, if all the end effectors reach their respective goals, a positive reward is given to all the tasks. For all other cases, a negative reward proportional to the $error$ is given. In DDPG setting, there is a single reward unlike DiGrad. A positive reward is given when all the end effectors reach their goals simultaneously. Else, a negative reward is given proportional to the sum of $error$ of all the tasks, that is, sum of distances between the respective goal and its corresponding end effector.


(a) 8-link Planar Manipulator

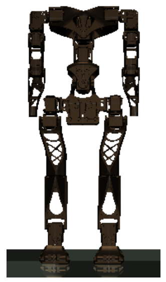

(b) Humanoid Robot

Figure 3: Environments

Environments

8-link manipulator: The first environment is an 8 DoF planar manipulator with 3 end effectors as shown in Fig. 3a. We can observe that the shared sub-chain of 2 joints is common to all the 3 tasks. Also, the action dimension of the non-shared chains are kept different in order to check the robustness of the framework. The dimensions of each action is given as: $a_1 = 3$, $a_2 = 4$, $a_3 = 5$ and $a_s = 2$.

Fig.4(A) shows the performance curves for the five different settings. From the mean score curves, we can see that all the DiGrad based settings converge faster and achieve better final performance than DDPG. Even though the action dimension of each task was different, all the network settings in DiGrad framework worked equally well for all the tasks. Whereas, DDPG showed comparable performance for only Task-2.

From 4(A1), we can see that the single critic framework consistently has better average reward per episode than multi critic frameworks. Thus, modelling all the action value functions in one critic network doesn't affect the performance. In fact, the shared parameters in single critic framework could help the network capture the correlation among the actions much better than multi-critic case. Note that, single critic framework is achieving these performances with lesser parameters than the multi-critic framework.

DiGrad frameworks with heuristics perform at par with the non-heuristic frameworks. On applying the aforementioned heuristic, no significant improvement in the average reward curve is observed. But in the mean score curves, specially Task-1 and Task-2 curve, we can see that the application of heuristics helps the network to be more stable as compared to their respective non-heuristic curves. Thus, we can say that normalising the gradient of action values of the shared action as in Eq. (11) could help the network deliver robust multi-

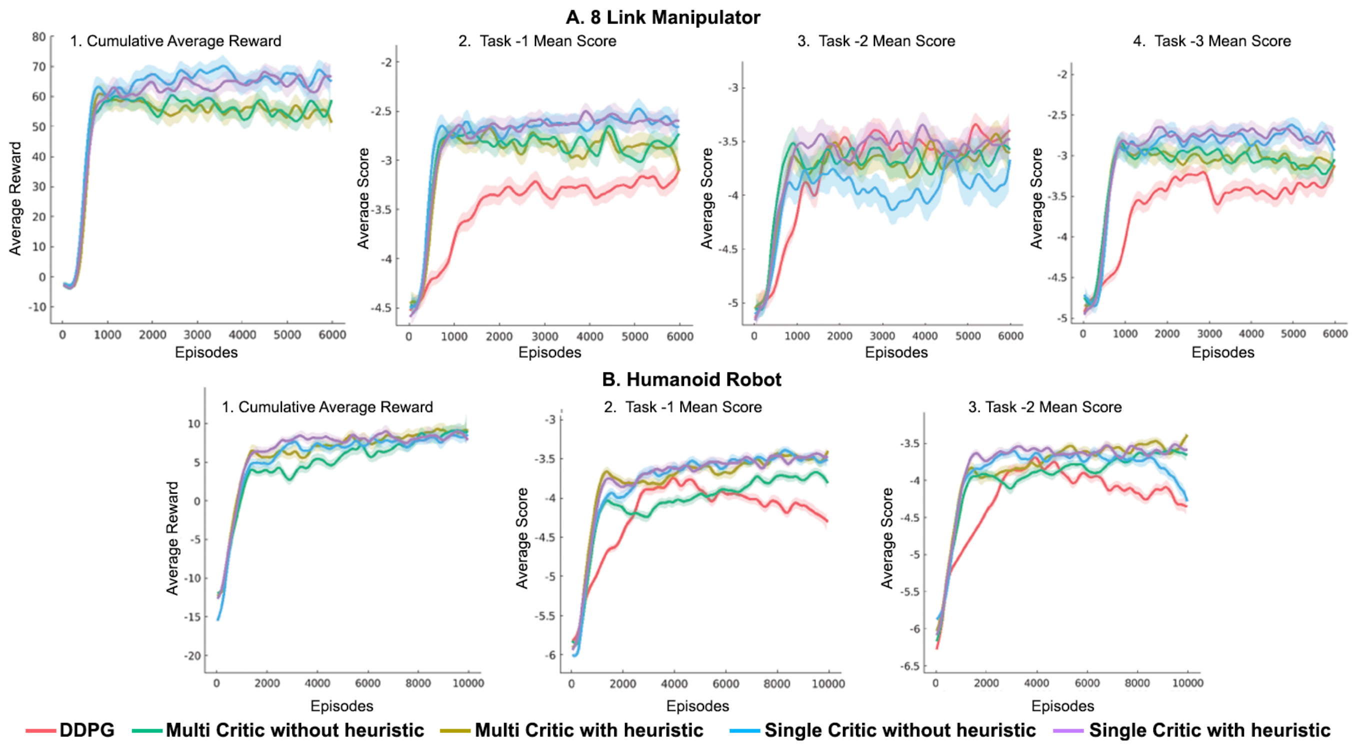


Figure 4: Performance curves of reachability task experiments on 8 link manipulator and humanoid. The bold line shows the average over 3 runs and the coloured areas show average standard deviation over the tasks. Note that, average reward curve is not plotted for DDPG as the reward function for it is different from DiGrad frameworks.

task training.

Humanoid robot: Secondly, we test our framework on a 27 DoF humanoid robot (Fig. 3b). This experiment involved reachability tasks of the 2 hands of the humanoid robot using the upper body (13 DoF) consisting of an articulated torso. The articulated torso is the shared chain which is affecting both the tasks. It is noteworthy that the articulated torso has 5 DoF whereas, the arms have 4 DoF each. Thus, the contribution of shared action (articulate torso) to the task is more than the non shared actions (arms). The environment for training is developed in MATLAB and the trained models were tested in a dynamic simulation environment MuJoCo.

Fig.4B summarizes the results for this case. We found that DPPG is generally unstable for solving multi-tasks problems. In some runs, it may learn initially but suffers degradation later. We observe that the DiGrad algorithm yields better final results while having greater stability.

From the mean scores of the tasks, we can see that the single critic frameworks converge faster and are stable throughout the experiment as compared to the multi-critic frameworks. The best setting is again the single critic with heuristic, outperforming all the others in all the cases.

Note that, the reward function for DDPG is kept different from the DiGrad framework. We also tried a different reward setting taking the sum of individual rewards r_i as a reward signal to DDPG framework, where r_i is same as defined in the DiGrad reward setting. We observed that this reward setting led to biasing, where one of the tasks dominated others. This behaviour could have been due to the negative interference across tasks, which didn't happen in DiGrad.

6 Conclusion

In this paper, we propose a deep reinforcement learning algorithm called DiGrad for multi-task learning in a single agent. This framework is based on DDPG algorithm and is derived from DPG theorem. In this framework, we have a dedicated action value for each task, whose update depends only on the action and reward corresponding to the task. We introduced a differential policy gradient update for the compound policy.

We tested the proposed framework for learning reachability tasks on two environments, namely 8-link manipulator and humanoid. These experiments show that our framework gave better performance than DDPG, in terms of stability, robustness and accuracy. These performances are achieved keeping the number of parameters comparable to DDPG framework. Also, the algorithm learns multiple tasks without any decrease in the performance of each task.

Our work focuses on learning coordinated multi-actions in the field of robotics, where a single agent performs multiple tasks simultaneously. The framework was able to capture the relation among tasks where some tasks had overlapped action space. Our future work will focus on extending the framework to multi-agent domain.

References

[Borsa *et al.*, 2016] Diana Borsa, Thore Graepel, and John Shawe-Taylor. Learning shared representations in multi-task reinforcement learning. *arXiv preprint arXiv:1603.02041*, 2016.

[Buss, 2004] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

[Chiaverini, 1997] Stefano Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Transactions on Robotics and Automation*, 13(3):398–410, 1997.

[Dimitrakakis and Rothkopf, 2011] Christos Dimitrakakis and Constantin A Rothkopf. Bayesian multitask inverse reinforcement learning. In *European Workshop on Reinforcement Learning*, pages 273–284. Springer, 2011.

[Gu *et al.*, 2017] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3389–3396. IEEE, 2017.

[Klein *et al.*, 1995] Charles A Klein, Caroline Chu-Jenq, and Shamim Ahmed. A new formulation of the extended jacobian method and its use in mapping algorithmic singularities for kinematically redundant manipulators. *IEEE Transactions on Robotics and Automation*, 11(1):50–55, 1995.

[Konidaris *et al.*, 2011] George Konidaris, Sarah Osentoski, and Philip S Thomas. Value function approximation in reinforcement learning using the fourier basis. In *AAAI*, volume 6, page 7, 2011.

[Kulkarni *et al.*, 2016] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[Lazaric and Ghavamzadeh, 2010] Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian multi-task reinforcement learning. In *ICML-27th International Conference on Machine Learning*, pages 599–606. Omnipress, 2010.

[Lazaric, 2012] Alessandro Lazaric. Transfer in reinforcement learning: a framework and a survey. In *Reinforcement Learning*, pages 143–173. Springer, 2012.

[Lillicrap *et al.*, 2015] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Møller, 1993] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[Phaniteja, 2017] Guhan Sarkar K Madhava Phaniteja, Dewangan. A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots. *arXiv preprint arXiv:1801.10425*, 2017.

[Rusu *et al.*, 2015] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[Schaul *et al.*, 2015] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[Siciliano, 1990] Bruno Siciliano. Kinematic control of redundant robot manipulators: A tutorial. *Journal of intelligent and robotic systems*, 3(3):201–212, 1990.

[Silver *et al.*, 2014] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[Teh *et al.*, 2017] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4499–4509, 2017.

[Van Hasselt *et al.*, 2016] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 16, pages 2094–2100, 2016.

[Wang *et al.*, 2015] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[Yang *et al.*, 2017] Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, and Lianwen Jin. Multi-task deep reinforcement learning for continuous action control. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3301–3307, 2017.

[Yin and Pan, 2017] Haiyan Yin and Sinno Jialin Pan. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *AAAI*, pages 1640–1646, 2017.

[Zhang *et al.*, 2016] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. *arXiv preprint arXiv:1612.05533*, 2016.