Multi-Task Deep Reinforcement Learning for Continuous Action Control*

Zhaoyang Yang[1,2], Kathryn Merrick[1], Hussein Abbass[1], Lianwen Jin[2]
[1]School of Engineering and Information Technology, University of New South Wales, Australia
[2]School of Electronic and Information Engineering, South China University of Technology, China
[1,2]yangzhaoyang6@126.com,[1]K.Merrick@adfa.edu.au,[1]h.abbass@adfa.edu.au,[2]lianwen.jin@gmail.com

Abstract

In this paper, we propose a deep reinforcement learning algorithm to learn multiple tasks concurrently. A new network architecture is proposed in the algorithm which reduces the number of parameters needed by more than 75% per task compared to typical single-task deep reinforcement learning algorithms. The proposed algorithm and network fuse images with sensor data and were tested with up to 12 movement-based control tasks on a simulated Pioneer 3AT robot equipped with a camera and range sensors. Results show that the proposed algorithm and network can learn skills that are as good as the skills learned by a comparable single-task learning algorithm. Results also show that learning performance is consistent even when the number of tasks and the number of constraints on the tasks increased.

1 Introduction

Reinforcement learning [Sutton, 1988; Watkins and Dayan, 1992] has long been an important topic in the area of robotics and intelligent agents. It addresses the problem of how agents should learn to take actions to achieve goals in a given environment. Classical work used linear function approximations to enhance algorithms' generalization in complex and infinite environments [Grounds and Kudenko, 2008; Konidaris, *et al.*, 2011].

In recent years, deep learning methods have achieved significant progress in several research topics, including many vision and linguistic applications. With hundreds of thousands of auto-learned parameters and a number of different kinds of network architectures such as convolutional networks (CNN) [Krizhevsky, *et al.*, 2012] and long-short term memory (LSTM) networks [Graves, *et al.*, 2013], deep neural networks have shown unprecedented feature extraction ability and robust generalization capability.

Though it was generally believed that non-linear approximators such as deep neural networks are hard to train in reinforcement learning scenarios, recent advances in reinforcement learning have successfully combined deep learning to make significant improvements. Well-known work includes using deep reinforcement learning agents to play Atari games [Mnih, *et al.*, 2015] and Go games [Silver, *et al.*, 2016].

Different from game decision making, robot control always involves continuous action spaces [Lillicrap, et al., 2016; Mnih, et al., 2016] and many physical factors. The problem becomes more challenging when controlling robots to achieve high level goals, where robots have to reuse basic skills. It is therefore necessary for robots to learn multiple skills that can be assembled when faced with, for example, hierarchically or intrinsically motivated learning scenarios.

In this paper, we are going to explore deep reinforcement learning in a multi-task continuous control domain. Specifically, based on the Deep Deterministic Gradient Descent (DDPG) algorithm [Lillicrap, *et al.*, 2016], we propose a new algorithm to enable the robot to learn multiple skills concurrently. We call this algorithm multi-DDPG.

In the proposed algorithm, we make use of both sensor data and images collected from a camera set on a simulated robot. Images are in the first-person view and record what the robot can see in front of it. The simulated robot learns multiple skills within the same training process.

In order to reduce the huge amount of parameters we need for combining images and sensor data, we propose a new network architecture which makes use of multi-layer perceptron convolutional (mlpconv) layers [Lin, *et al.*, 2013]. With the new network architecture, we reduce the number of parameters originally needed for single task learning by 75% and avoid parameter overloading when expanding to multiple tasks.

Our simulations were conducted in Gazebo 2 built in a ROS Indigo environment. We used a Pioneer 3AT robot model and set sensors and a camera on it. The robot is moving in an obstacle-free, walled space. We tested both the new network architecture and the multi-DDPG algorithm for 12 movement-based control tasks. The network architecture along with the proposed algorithm achieve high performance and considerable robustness on all the tasks we tested.

We organize the rest of the paper as follow. We first introduce related work and background in Sections 2 and 3. In Section 4, we detail the proposed algorithm and network

* This work was supported by the Australian Research Council under Grant DP160102037.

architecture. Section 5 shows the experiments and results whilst Section 6 concludes the paper.

2 Related Work

Prior to deep reinforcement learning, most multi-task oriented algorithms sought help from transfer learning to realize proper control over different tasks. [Lazaric, 2012] is a good collection of these methods. Besides, some work investigated joint training of multiple value functions [Lazaric and Ghavamzadeh, 2010] or policy functions [Dimitrakakis and Rothkopf, 2011] over a set of tasks. However, the functionalities of these algorithms were limited by hand crafted features.

Although a lot of work has been done to improve deep reinforcement learning algorithms over single tasks, there is much less work done for multi-task scenarios. Recent papers on this topic are [Bangaru, *et al.*, 2016], [Borsa, *et al.*, 2016] and [Zhang, et al., 2016]. But different from our work, [Bangaru, *et al.*, 2016] mostly focused on exploration and generative models. The other two explored learning universal abstractions of state-action pairs or feature successors, which are similar in nature to transfer learning.

Some work deals with hierarchical tasks [Krishnamurthy, *et al.*, 2016; Kulkarni, *et al.*, 2016] or intrinsically motivated agents [Mohamed and Rezende, 2015] using deep reinforcement learning. While all agents trained in these papers are able to achieve multiple different sub-goals to fulfill a final task, they are actually guided by the same reward signals. In our method, we use different reward functions for each of the tasks to assume minimum cross-correlation among the tasks.

Another topic that may share some common points with our work is multi-agent learning [Foerster, *et al.*, 2016; Tampuu, *et al.*, 2015]. However, all of these works involve at least two different agents, while ours aims to train one agent for multiple tasks.

3 Background

We consider the standard reinforcement learning setup, where an agent is interacting with the environment $\boldsymbol{E}$ in discrete timesteps. At each timestep t, the agent receives a state $s_t \epsilon \boldsymbol{S}$, takes an action $a_t \epsilon \boldsymbol{A}$ according to policy $\pi: \boldsymbol{S} \rightarrow \boldsymbol{A}$, gets to the next state s_{t+1} according to a probability distribution $P: \boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{S}$ and then receives a reward $r_t \in \boldsymbol{R}$.

The goal is to learn a policy π that can maximize the expected future return $R = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $\gamma \epsilon (0,1)$ is the discount factor. Note that the policy π may be stochastic, but in our case, we are considering only deterministic policies.

The state-action value function is commonly used in many reinforcement learning algorithms. It is an estimator of the expected future return:

$$Q^{\pi}(s_t, a_t) = E[R_t | s_t, a_t] \tag{1}$$

Unlike other value-based and policy-based reinforcement learning algorithms, algorithms based on the actor-critic architecture [Peters, *et al.*, 2005] combine a state-action value function (the critic) and policy function (the actor) within one framework. This enables the algorithm to learn more complex policies for controls in continuous space.

DDPG [Lillicrap, *et al.*, 2016] is a deep reinforcement learning algorithm that deals with continuous control tasks. In DDPG, both the critic and the actor are approximated by deep neural networks (θ^Q, θ^{π}). Replay memory and target networks $(\theta^{Q'}, \theta^{\pi'})$ have addressed the problem of training instability and brought DDPG great success.

With a single loss function:

$$L(\theta^Q) = (Q(s_t, a_t | \theta^Q) - y_t)^2 \tag{2}$$

where

$$y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1} | \theta^{\pi'}) | \theta^{Q'}) \tag{3}$$

DDPG can train a policy by updating the critic with:

$$\theta^Q \leftarrow \theta^Q - \mu_Q \cdot \nabla_{\theta^Q} L(\theta^Q) \tag{4}$$

and the actor with:

$$\theta^{\pi} \leftarrow \theta^{\pi} - \mu_{\pi} \cdot \nabla_a Q(s_t, , \pi(s_t | \theta^{\pi}) | \theta^Q) \cdot \nabla_{\theta^{\pi}} \pi(s_t | \theta^{\pi}) \tag{5}$$

where symbol ∇ donates gradients and μ_Q, μ_{π} represent the learning rate of critic and actor respectively.

The mlpconv layer [Lin, *et al.*, 2013] is a relatively new network architecture. Traditional convolutional layers that simply activate rendered feature maps with:

$$f_{i,j,k} = F(\omega_k^T x_{i,j} + b_k) \tag{6}$$

Where F is the activation function. But for mlpconv layers, before activating, feature maps are linearly recombined across different map channels, as a result:

$$f_{i,j,k}^1 = F\left(\omega_k^{nT} x_{i,j} + b_{k_n}\right) \tag{7}$$

Where n indicates the number of perceptrons in the layer. This reorganization of information across channels is proposed to add to the representative of the feature maps.

In the proposed multi-DDPG algorithm, we keep some of the basic concepts of DDPG, then make use of the mlpconv layers to achieve a network architecture with fewer parameters and finally extend it to multi-task scenarios.

4 Multi-DDPG

In this paper, we propose a new algorithm, which we call multi-DDPG to handle multiple continuous control tasks. We propose a new network architecture which makes use of mlpconv layers to significantly reduce the number of parameters needed and combine images and sensor data as input. Compared to DDPG, the proposed algorithm contains only one critic but multiple actors. While each actor learns a different control task, all actors are trained concurrently within the same training process. Figure 1 gives an overview of the algorithm.

4.1 Combining Sensors and Camera Data

We combine two kinds of data collected from the environment. The first kind of data is images collected from the camera. The camera is mounted at the front of the robot and functions like its eyes. The images collected by the camera are in the first-person view that represents what the robot can see in front of it.

While images can enable the agent to learn high level representations by providing the agent with vivid and rich information about the environment, we find that information

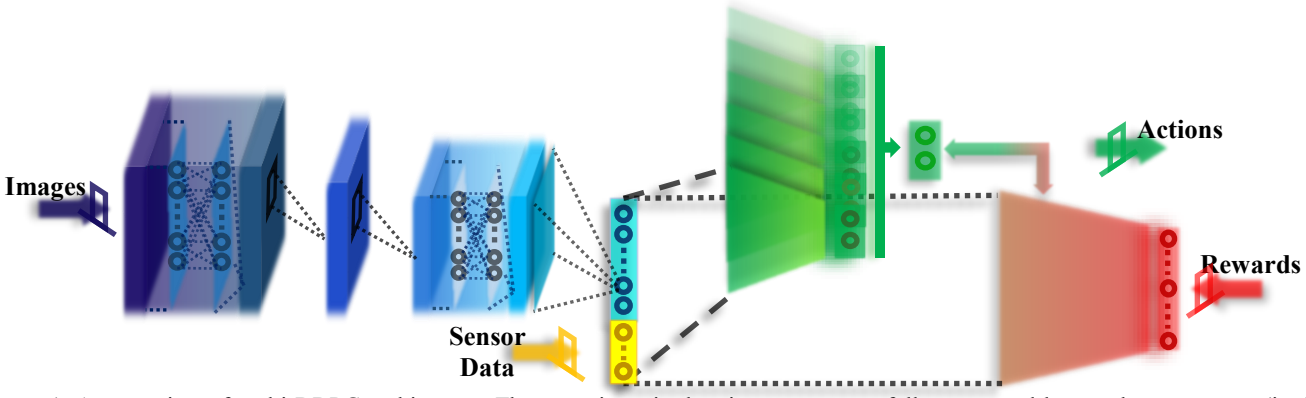

Figure 1: An overview of multi-DDPG architecture. The trapeziums in the picture represent fully connected layers, the green ones (in the middle) for actors and red one (on the right) for critic. The square-dotted lines indicates that there is back-propagation between two layers, while dashed lines do not involve back-propagation.

from raw sensors is more straightforward and sometimes can help the agent to learn more efficiently. For example, while raw sensors can directly send the agent distance information, this information is very implicit and hard to infer from images.

This inspired us to introduce sensor data as the second kind of data in our algorithm. Specifically, we feed the image data at the beginning of the networks and extract it with the convolutional layers. Sensor data are included right after the last convolutional layer and concatenated with the image features to go through the remaining layers of the network (See the two concatenated vectors in the middle of Figure 1).

4.2 Using mlpconv Layers

Combining images and sensor data will introduce additional parameters in the network. For our network, which has relatively small fully connected layers, most of the new parameters will be introduced when all feature maps of the last convolutional layer are flattened to form a long vector to connect to the next fully connected layer.

These new parameters become even more significant when we have multiple actors. A single duplication of the actor network for a new task will add at least 400,000 new parameters to the whole agent system. This huge number of parameters is not only tricky to train, but also redundant to some extent. So it is important to find ways to reduce the parameters in the network.

[Lin, *et al.*, 2013] proposed a new network layer for image classification problems. The proposed network layer in that paper is called the mlpconv layer. In addition to the traditional convolutional layer, two perceptron layers are added to reconstruct the feature maps rendered by that layer. This reconstruction of the feature maps can merge information across different channels and enable the network to learn more abstract features. With the help of this reconstruction, the network achieves excellent classification performance by simply implementing an average pooling layer on the last mlpconv layer.

In our algorithm, we follow this structure to extract image features. The basic idea is that compared to other image processing problems, such as object detection and recognition, the information the robot needs to infer from images is less complex, especially when we have the help of sensor data. Thus we do not need pixel level information in the feature maps. Instead, by implementing mlpconv layers with global average pooling, we can obtain a shorter feature vector with more refined information about the environment. At the same time, we suggest that short feature vectors of images are beneficial for the network to infer information from sensor data, while long vectors of flattened feature maps can easily drown the short vector of sensor data.

In our algorithm, we replace the first and third (also the last) convolutional layers of the networks with a mlpconv layer (see left part of Figure 1). The first mlpconv layer contains 32 kernels. The second one contains 64 kernels and is followed by a global average pooling layer to transfer feature maps into a vector. This vector is then concatenated with sensor data to be fed into the remaining fully connected layers.

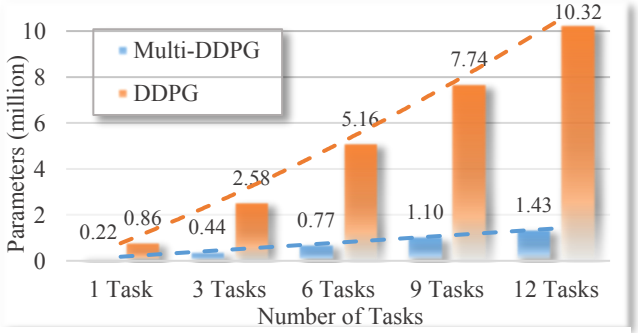

Figure 2: A comparison of the number of parameters in the agent. The proposed network significantly reduces the number.

By doing so, we significantly reduce the number of parameters we need for a single task from 430,000 to around 110,000. Figure 2 gives a comparison of the number of parameters introduced in the two networks. We can also see in the figure how the network is going to reduce the number as the number of tasks increases.

4.3 Sharing Parameters

As discussed above, the output feature vector of the last mlpconv layer will contain information at a comparable level of abstraction to raw sensor data, therefore we can treat it in

the same way we treat sensor data. This enables us to share the parameters of the convolutional layers amongst all networks naturally (see the inputs of fully connected layers shown in Figure 1, note that the multi-perceptron parts of the last mlpconv layer are not shared).

In addition to parameter sharing, we only update the convolutional layers when we are training the critic. Actor update only involve their unshared layers, although the image part of their inputs has to be processed by the convolutional layers before reaching the fully connected layers.

4.4 Training Multiple Actors

Based on the proposed light-weight network architecture and parameter sharing scheme discussed above, we extend the algorithm from single task to multi-task learning. To achieve this, instead of a single actor in the actor-critic architecture, we use multiple actors with each actor responsible for one task (see the multiple actors in the middle of Figure 1). These actors are learned from the same inputs and within the same training process.

While we extend the single actor to multiple actors, we do not add any new critics in our algorithm. This means that the single critic in the system must be able to guide all actors to update properly. Therefore, instead of a single output state-action value, our critic has to output multiple state-action values, one for each actor. Correspondingly, we will assess executed actions according to all rewarding criterions we have for different tasks to form a vector of rewards, regardless of which actor produced that action. Then instead of the original loss function in (2) and (3), we will have:

$$L(\theta^Q) = \sum_{g=1}^{G} (Q_g(s_t, a_t|\theta^Q) - y_{g,t})^2 \qquad (8)$$

where g is identity number of the task and G the total number of tasks we have. The supervising signal becomes:

$$y_{g,t} = r_{g,t} + \gamma Q_g(s_{t+1}, \pi(s_{t+1}|\theta^{\pi'})|\theta^{Q'}) \qquad (9)$$

Note that only one actor will be activated to choose actions in each timestep. During exploration, actors will be activated iteratively. We do not distinguish actions produced by different actors and all transitions will be stored in the same replay memory.

Also note that as we do not distinguish actions produced by different actors, we can simply iteratively choose a target actor to calculate $y_{g,t}$ for all input data in a training iteration. This is benefited by the fact that critic training and actors' training are not synchronous. It turns out that the critic will not be trained to be task specific and it will always be able to infer state-action values of all tasks we have for any input (s_t, a_t) pairs, whichever actors produced the a_t.

After the critic is updated, we update all actors one after another. For each actor update, the updating gradient is:

$$\nabla\theta^{\pi_g} = \mu_{\pi_g} \cdot \nabla_a Q_g\big(s_t, \pi_g(s_t|\theta^{\pi_g})|\theta^Q\big) \cdot \nabla_{\theta^{\pi_g}} \pi_g(s_t|\theta^{\pi_g}) \qquad (10)$$

See that action gradient $\nabla_a Q_g$ is task specific and for each actor, it is the gradient with respect to the corresponding state-action value output of the critic.

Algorithm 1 Multi-DDPG

Input: maximum training episode E_{max}, maximum steps in each episode S_{max}, mini-batch size M, replay memory **P.**
Initialization: randomly initialize networks weights θ^Q, $\theta_1^\pi, \dots\dots, \theta_G^\pi$ and target networks weights $\theta^{Q'} \leftarrow \theta^Q, \theta_g^{\pi'} \leftarrow \theta_g^\pi$.
while episode < E_{max}
  Initialize random noise *N* for exploration
  Iteratively select activated actor
  Get initial state s_1
  while step <S_{max} **and** episode **not** terminated
    Select action a_t using selected actor and add *N*
    Execute a_t and get reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in **P**
    Randomly sample a batch of M transitions from **P**
    Update θ^Q according to (8) and (9)
    Update $\theta^{Q'}$
    for g < *G*
      Update θ_g^π according to (10)
      Update $\theta_g^{\pi'}$
    end for
  end
end

Finally, we update target networks according to the soft updating rule. Then by the end of training, we will get a single critic that outputs state-action values for all tasks we have and multiple actors each producing actions to achieve a different task. We summarize the algorithm in Algorithm 1.

5 Experiments

We conducted two sets of experiments to test and analyse the proposed algorithm. We first conducted one set of experiments to test the new network architecture we proposed in the algorithm. In this set of experiments, we tested our network against the original network in [Lillicrap, *et al.*, 2016] with 12 movement-based control tasks. These 12 tasks are highly constrained movement controls of the robot, including going forward and backward at high and low speed, moving forward-left and forward-right slowly and quickly, and reversing-left and reversing-right slowly and quickly. These movement patterns are shown in Figure 3. Note that all experiments in this set are single task based.

In the second set of experiments, we analysed both the robustness and performance of the proposed multi-DDPG algorithm with different numbers of tasks. As the number of tasks increases, the tasks become more constrained, which means the robot must fulfil more conditions to receive positive rewards. We tested it in 3-task and 6-task scenarios, and then finally in a 12-task scenario with the same 12 tasks we have in the first set of experiments to give a comparison.

The experiments were conducted in a simulation with Gazebo 2 under a ROS Indigo environment. We used the robot model of Pioneer3AT and added a laser and camera on it. While the laser can provide diverse information, we only collected distance information from four angles (front, back, left, right). The robot is spawned in an obstacle-free walled space, in which it should move.

For all the experiments, the agent was given reward of

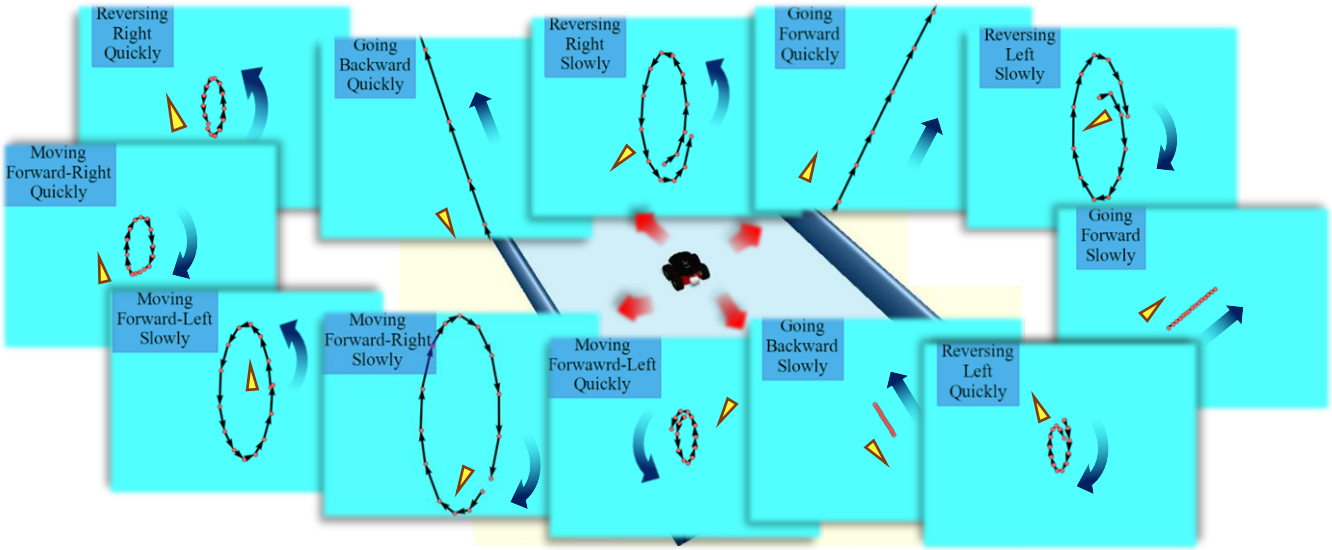

Figure 3: A collection of movements of the 12 highly constrained control tasks. At most 15 action steps are shown in each picture. The red nodes are the location of the robot, and black arrows in between are its movement trajectories. In each picture, yellow triangle indicates the initial orientation of the robot, while bold blue arrow shows the overall moving direction of the robot.

value 1 for achieving a task. On the contrary, the agent was penalized a value -0.5 if the robot made dangerous movements such as crashing into walls or turning over. Otherwise, the reward was 0.

During training, exploration was governed by an Ornstein-Uhlenbeck process [Uhlenbeck and Ornstein, 1930] to randomize and balance exploration. When testing the performance, this exploration was no longer permitted. For all experiments, we trained the model 3 times and tested it intermediately. Each model is trained for 5,000 episodes which contain approximately 60,000 training iterations. Adam [Kingma and Ba, 2015] is used to train the network, with initial learning rates 0.001 and 0.0001 for the critic and actor respectively. We set the discount factor to be 0.9 and train our networks in TensorFlow [Abadi, *et al.*, 2016].

5.1 Testing the Proposed Network Architecture

We show the results of the 12 tasks tested in the first set of experiments in Figure 4. We can see from these three graphs that our proposed network achieved comparable performance against the network proposed for DDPG. While both networks showed stability through each training process, they also showed robustness with low standard deviations. Moreover, both networks were able to collect high average rewards per action which indicates that they were both acting as expected during testing.

We can see that the average rewards per action may be a little lower when high speed is required for that task. This is mainly caused by the acceleration at the beginning of the testing. We show how the robot trained with our network was acting by drawing out its movement trajectories of the 12 tasks in Figure 3. We can see that the robot was moving as expected. Although the trajectories of those turning tasks are not in perfect circles, they are good enough to collect rewards.

Note that our network is achieving these performances with far fewer parameters compared to the network in DDPG. A comparison of the number of parameters introduced is in Figure 2.

5.2 Testing Multi-Task Performance

We first tried the proposed multi-DDPG algorithm to learn 3 tasks and 6 tasks concurrently. These tasks are less constrained controls of the robot compared to the 12 tasks above (such as simply moving straight without direction and speed restrictions). The results of these two early trials show that our algorithm can deliver robust multi-task training of these less constrained tasks, in which actors started to act according to their corresponding reward signals in early stage of the training.

Finally, we tested multi-DDPG with the 12 highly constrained control tasks scenario. The results are also shown in Figure 4. We can see that the performances of multiple actors trained by multi-DDPG are comparable to actors trained by single task scheme. The narrow shadow areas and mild vibrations in the graphs demonstrate the robustness of the algorithm and the stability of the performances of trained actors.

Note that the increased number of tasks and their constraints do not increase the number of episodes needed to stabilize training. This may be owing to the parameter and replay memory sharing amongst all tasks, which helps the agent to avoid dangerous actions and increase the chance of collecting rewards during exploration. The high average rewards collected in every action across all individual tasks further prove the effectiveness of the algorithm.

These results suggest that the proposed multi-DDPG algorithm can not only train high performance actors but also remain robust either when the number of tasks or the constraints of tasks increase. Its light-weight architecture as well as parameter sharing strategy also make it flexible

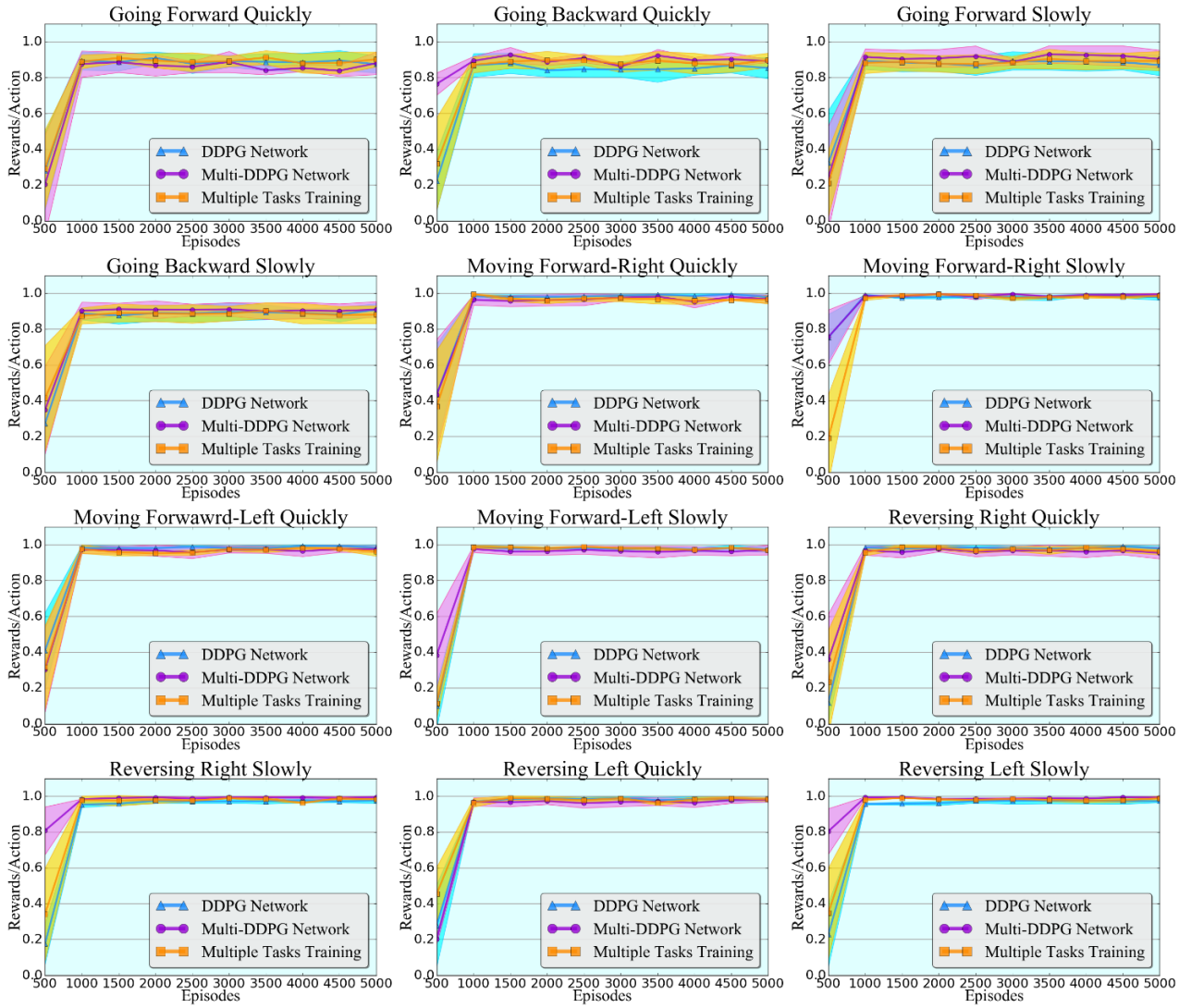

Figure 4: The performances of 12 highly constrained control tasks obtained from different experiments. The lines in the graphs show the average performances, while the corresponding shadow areas show standard deviations.

enough to be expanded to handle more actors and tasks.

6 Conclusion

In this paper, based on the DDPG algorithm proposed in [Lillicrap, *et al.*, 2016], we present a multi-task training deep reinforcement learning algorithm called multi-DDPG which combines images and sensor data as its input. In order to reduce the number of parameters needed to extract image features and learn multiple actors, we also introduce a new network architecture which takes advantage of mlpconv layers [Lin, *et al.*, 2013].

We conducted two sets of experiments using a Pioneer 3AT mobile robot in Gazebo 2 under a ROS Indigo Environment. Our first set of experiments show our new network achieves comparable performance to DDPG with only 25% of the parameters introduced. The second set of experiments demonstrates the robustness of our proposed algorithm against the increasing numbers of tasks and tasks' constraints. Also it shows that the algorithm achieves stable multi-task training without any decrease in the performance of each individual task.

However, we should note that while the algorithm achieves multi-task training, it cannot be used to choose which task it should do under given conditions. This means the algorithm does not consider how to use these learned skills to fulfil other goals. Nevertheless, the skills learned by our algorithm are accurate enough to be reused for high level goals. Our future work will focus on how to embed hierarchical and intrinsically motivated learning into the algorithm's framework to properly choose these learned tasks to achieve other high level goals.

References

[Abadi, *et al.*, 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

arXiv preprint arXiv:1603.04467, 2016.

[Bangaru, *et al.*, 2016] Sai Praveen Bangaru, JS Suhas and Balaraman Ravindran. Exploration for Multi-task Reinforcement Learning with Deep Generative Models. In *Neural Information Processing Systems Deep Reinforcement Learning Workshop*, 2016.

[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.

[Dimitrakakis and Rothkopf, 2011] Christos Dimitrakakis and Constantin A. Rothkopf. Bayesian Multitask Inverse Reinforcement Learning. In *European Workshop on Reinforcement Learning*, pp.273-284, 2011.

[Foerster, *et al.*, 2016] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pp.2137-2145, 2016.

[Graves, *et al.*, 2013] Alex Graves, Abdel-rahman Mohamed and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp.6645-6649. IEEE, 2013.

[Grounds and Kudenko, 2008] Matthew Grounds and Daniel Kudenko. Parallel reinforcement learning with linear function approximation. In *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning,* pp.60-74. Springer, Berlin, Heidelberg, 2008.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

[Konidaris, *et al.*, 2011] George Konidaris, Sarah Osentoski and Philip Thomas. Value Function Approximation in Reinforcement Learning using the Fourier Basis. In *Proceedings of Proc. AAAI Conference on Artificial Intelligence*, pp.380-385. AAAI, 2011.

[Krishnamurthy, *et al.*, 2016] Ramnandan Krishnamurthy, Aravind Lakshminarayanan, Peeyush Kumar and Balaraman Ravindran. Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks. In *International Conference on Machine Learning Abstraction in Reinforcement Learning Workshop*, 2016.

[Krizhevsky, *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Network. In *Proceedings of Advances in Neural Information Processing Systems*, pp.1097-1105. MIT Press, 2012.

[Kulkarni, *et al.*, 2016] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi and Joshua B. Tenenbaum. Hierarchical Deep Reinforcement Learning Integrating Temporal Abstraction and Intrinsic Motivation. In *Proceedings of Advances in Neural Information Processing Systems*, pp.3675-3683. 2016.

[Lazaric, 2012] Alessandro Lazaric. Transfer in Reinforcement Learning: a Framework and a Survey. In *Reinforcement Learning,* Berlin, Heidelberg, 2012.

[Lazaric and Ghavamzadeh, 2010] Alessandro Lazaric and Mohammad Ghavamzadeh. Bayesian Multi-Task Reinforcement Learning. In *Proceedings of International Conference on Machine Learning*, pp.599-606. ACM, 2010.

[Lillicrap, *et al.*, 2016] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, *et al.* Continuous Control with Deep Reinforcement Learning. In *Proceedings of International Conference on Learning Representations*. 2016.

[Lin, *et al.*, 2013] Min Lin, Qiang Chen and Shuicheng Yan. Network in Network. *arXiv preprint arXiv:1312.4400*, 2013.

[Mnih, *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, *et al.* Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of International Conference on Machine Learning*. ACM, 2016.

[Mnih, *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, *et al.* Human-level Control through Deep reinforcement Learning. *Nature*, 518(7540):529-533. 2015.

[Mohamed and Rezende, 2015] Shakir Mohamed and Danilo J. Rezende. Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning. In *Proceedings of Advances in Neural Information Processing Systems*, pp.2125-2133. MIT Press, 2015.

[Peters, *et al.*, 2005] Jan Peters, Sethu Vijayakumar and Stefan Schaal. Natural Actor-Critic. In *Proceedings of European Conference on Machine Learning*, pp.280-291. 2005.

[Silver, *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, *et al.* Mastering the Game of Go with Deep Neural Networks and TreeSearch. *Nature*, 529(7587):484-489. 2016.

[Sutton, 1988] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3(1):9-44. 1988.

[Tampuu, *et al.*, 2015] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, *et al.* Multiagent Cooperation and Competition with Deep Reinforcement Learning. *arXiv preprint arXiv:1511.08779*, 2015.

[Uhlenbeck and Ornstein, 1930] George E Uhlenbeck and Leonard S Ornstein. On the Theory of the Brownian Motion. *Physical Review*, 36(5):823. 1930.

[Watkins and Dayan, 1992] Christopher J.C.H. Watkins and Peter Dayan. Q-Learning. *Machine Learning*, 8(3):279-292. 1992.

[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:1612.05533*, 2016.