Weight Sharing in Deep Reinforcement Learning for Continuous Control

G109 | s1784227, s1786813, s1792338

Abstract

Robot locomotion is a hard task due to non-linear dynamics and uncertainty in sensory measurements. Deep reinforcement learning is well suited for this problem, because reinforcement learning is designed to deal with uncertainties and neural networks are able to model complex non-linear functions. Recently proposed algorithms for continuous control with deep reinforcement learning use fully connected neural networks that do not utilize the structurally equivalent parts in robots, such as arms or legs. We propose a neural network architecture that employs locally connected sub-networks for structural components and shared weights for equal parts. In this report, we furthermore introduce the methodology and reinforcement learning environment used, as well as initial baseline experiments of established deep reinforcement learning algorithms.

1. Introduction

A major challenge in artificial intelligence is to create autonomous robotic systems in which robots make their own decisions without relying on pre-programmed sequences or having a human in the loop. This is a hard problem, because most systems have non-linear dynamics that are hard to model explicitly and furthermore the system must deal with uncertainty in the sensory information. Thus, we need algorithms that can deal with uncertainty and do not need a exact model of the system.

The largest part of robotics deals with controlling the motion of robots. Traditionally, this is done with control algorithms that make sure the robot follows a pre-defined trajectory (Siciliano et al., 2009). These algorithms do not enable robots to act autonomously and it is a challenging task to create trajectories for more complex tasks such as humanoid robot walking.

Recent advances in deep reinforcement learning (LeCun et al., 2015; Mnih et al., 2015b) have made it possible to learn motion control given only a reward signal. In reinforcement learning, agents learn to take actions to maximize a cumulative reward in interaction with an environment. Applied to robotics, the reinforcement learning algorithm can control the torques applied to joints and observe the joint angles and velocities.

More recent algorithms for continuous control are based on the policy gradient method, where deep neural networks are used as policies (Lillicrap et al., 2015). One problem with policy gradient algorithms is that they are not sample efficient, in that they require a lot of trials until they learn the desired motion. Since it is quite expensive to operate real robots, most algorithms are first evaluated in physically simulated environments.

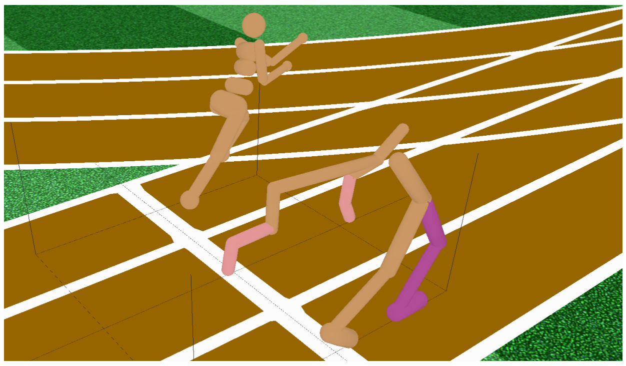
Figure 1. Three example robot simulations provided by the Roboschool package. From left to right: the humanoid, the half cheetah and the 2-d walker.

In this report, we present an approach to designing policy network architectures that takes advantage of robots having structurally equivalent parts. All recent approaches learn different weights for all actions that control the joints. In contrast, we propose to use locally connected sub-networks with shared weights for low-level control of structurally equivalent parts. Considering a humanoid robot (Figure 1 on the left), for example, we would learn one sub-network for the legs and one for the arms. For execution we would use two copies of these networks for both legs and arms respectively. Since the network gets different inputs from higher layers, they are able to apply different torques to the joints in each leg or arm. We want to investigate if this network structure makes training faster and more stable because less weights are trained and a coherent low-level control is learned for parts that are structurally equivalent.

In the next section, we present related work in the field of deep reinforcement learning for continuous control. After that we introduce the reinforcement setting and the deep reinforcement learning algorithm that we base our work on. Then we present the Roboschool environment and its robots, which we use as a test bed. In Section 5 we propose the network architecture that we plan to evaluate against baseline experiments. We then explain the main quality measures in deep reinforcement learning research and postulate our hypothesis. We start our series of experiments in this report with a hyperparameter search on a simple toy robot task and then apply the best hyperparameters to obtain baseline results on a more complex task.

2. Related Work

The policy network employed with the first baselines for handling continuous control mechanisms in Lillicrap et al. (2015) is a fully connected network. This implicitly induces a certain independence assumption between different state features and between the robot's joint's behaviors. This assumed degree of feature independence can be increased or reduced by engineering different policy network architectures.

One example is Heess et al. (2017), who use two networks to decouple the state of the robot's joints from the state of the environment's obstacles. Before the last layer both networks get conflated to yield the output actions as a single policy network. In such way Heess et al. (2017) increase the learning speed for robots moving in obstacle-dense environments. Sharma & Kitani (2018) observe that robot locomotion is often of cyclic nature. They introduce phase parametric policy and value networks in order to force the learning of cyclic movements. Our proposed network structure may learn similarly by letting the weight sharing part of the network learn low-level control and the fully connected part govern the cyclic coordination.

Heess et al. (2016) introduce a hierarchical network structure based on the observation that to achieve an abstract goal the actions must follow an implicit hierarchy, e.g. rotating a foot is part of making a step which again is part of a humanoid robot walking forward. Their policy network consists of a high level controller constantly generating commands for a low level controller which outputs the actions. They are able to accelerate learning by transferring learned low level controllers among different tasks for the same robot. Our approach compares to Heess et al. (2016) in the way that we also imply a hierarchy, as we expect the fully connected part of the network to learn high level actions and the part of the network with shared weights to learn low level actions. The weight sharing can also be seen as a form of instantaneous transfer learning, where a behavior is learned for multiple joints even when the learning evidence roots in a single joint — Heess et al. (2016), in contrast, emphasize on transferring between tasks.

3. Background

A standard reinforcement learning setup is utilized in this work, which consists of an agent interacting with an environment E in discrete timesteps. For each timestep the agent receives an observation x_t, takes an action a_t and receives a scalar reward r_t. This environment may be partially observed, meaning the full history of the observation and action pairs may be needed to describe the state $s_t = (x_1, a_1, \ldots, a_{t-1}, x_t)$. In the cases which this research shall be exploring, the environment is fully observed, meaning the state $s_t = x_t$.

The behaviour of an agent is defined by a policy π which maps states to a probability distribution over possible actions $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$, meaning that a policy π prescribes a distribution of actions to take upon observation of a state s_t. A stochastic environment E can be modelled as a Markov decision process with state space $\mathcal{S}$, real valued action space $\mathcal{A} = \mathbb{R}^N$ and an initial state distribution $p(s_1)$. The transition dynamics are given by $p(s_{t+1}|s_t, a_t)$, which means given an observed state s_t, an action a_t has this probability of entering state s_{t+1}. The reward is also a function of the action and state $r(s_t, a_t)$.

The return from a state is given by $R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$, which is a sum of future rewards discounted by a discounting factor $\gamma \in [0, 1]$. This is a means of prioritizing the importance of immediate rewards as opposed to longer term ones. The overall goal in reinforcement learning is to find a policy in which the expected return originating from the starting distribution is maximized. This often requires to estimate the action-value function which describes the expected return after taking an action a_t in state s_t after which the policy π is followed:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i \geq t} \sim E, a_{i > t} \sim \pi}[R_t | s_t, a_t] \tag{1}$$

This is can be expressed as the Bellman equation (Sutton & Barto, 1998), which is a method to recursively compute this expectation as follows,

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]\right] \tag{2}$$

which can then be simplified given a deterministic target policy. So instead of the stochastic policy π, we can use a deterministic policy $\mu : \mathcal{S} \rightarrow \mathcal{A}$ to simplify the inner expectation:

$$Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))] \tag{3}$$

This function Q^μ is dependent only on the environment, meaning off-policy observations can be used to learn Q^μ. That means we can use a policy different to μ, for example a stochastic behavior policy β, to retrieve the observations. Q^μ can be approximated using a neural network parameterized by θ^Q which can then be optimized by minimizing the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}\left[(Q(s_t, a_t | \theta^Q) - y_t)^2\right] \tag{4}$$

with ρ^π being the state visitation distribution for a policy π, and y_t given by

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) | \theta^Q) \tag{5}$$

If were to use $\mu(s) = \operatorname{argmax}_a Q(s, a)$ as the deterministic policy, we would get the Q-learning algorithm from Watkins & Dayan (1992). But this is impractical with the continuous action space available to an agent in Roboschool

Algorithm 1 Deep Deterministic Policy Gradient (DDPG)

```
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 $\mathcal{R}$
**for** episodes=$1, 2, \ldots$ **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $s_1$
    **for** $t = 1, T$ **do**
        Selection action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to 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 $\mathcal{R}$
        Sample a random minibatch of $N$ transitions from $\mathcal{R}$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing 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 network:
            $$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
            $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
    **end for**
**end for**
```

(torques at each joint).

This is a problem which Lillicrap et al. (2015) have tackled in their design of deep deterministic policy gradient (DDPG), which is given in Algorithm 1. DDPG is an actor-critic approach based on the DPG algorithm by Silver et al. (2014). The critic function $Q(s, a)$ is learned as in Q-learning via the Bellman equation. The parameterized actor function $\mu(s|\theta^\mu)$ deterministically maps states to a specific action, thus specifying a policy. This actor is updated by applying the chain rule to the expected return from the start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$ with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_{\theta^\mu} Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_a Q(s,a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}]\end{aligned} \tag{6}$$

An issue when utilizing neural networks as function approximators in reinforcement learning is that making gradient steps based on samples, like in stochastic gradient descent, assumes that these samples are independently and identically distributed (as a Monte-Carlo estimate of the true gradient step). This is of course not the case when exploring environments sequentially. There is also the desire to learn in mini-batches to improve efficiency of learning. This is tackled by the use of a replay buffer, as introduced by Mnih et al. (2015a). The replay buffer is a finite sized cache $\mathcal{R}$ which stores tuples of previously explored transitions (s_t, a_t, r_t, s_{t+1}). When this cache is filled, old samples are discarded for new ones. At each timestep both the actor and critic networks are updated by sampling a mini-batch uniformly from the cache. Sampling from a large cache benefits the algorithm by allowing it to learn across a set of uncorrelated transitions.

Implementing Q-learning directly from equation 4 with neural networks can be particularly unstable as the network $Q(s, a|\theta^Q)$ that is being updated is also used in calculating the target value y_t given in equation 5. This makes the updates susceptible to divergence. The solution proposed by DDPG is to perform 'soft' target updates, as opposed to directly copying the weights. This is done by making a copy of both the actor and critic networks, $\mu'(s|\theta^{\mu'})$ and $Q'(s, a|\theta^{Q'})$ respectively to use specifically for calculating the target values. These are 'softly' updated by having them slowly track the learned networks in the following fashion

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{7}$$

with $\tau \ll 1$. This constrains the target values target to change slowly, inducing greater stability of learning. This slow updating of the network could be argued to slow learning, but in DDPG's proposal (Lillicrap et al., 2015) it was found the stability of learning greatly outweighed any other potential negatives.

There is also the matter of exploration. By this it is meant that the algorithm should have some mechanism to randomly explore action and policy space that has not been visited before. This is crucial to successful reinforcement learning in a fairly intuitive fashion. Firstly, the algorithm must know how to initially experiment to discover what actions are 'good' and return greater rewards. Furthermore, once a policy is discovered that does indeed increase reward, it may not necessarily be the most optimal policy to 'aim' towards, and there may well be benefit in exploring different areas in action space. Thus a deterministic policy requires some randomness to explore. This is achieved by adding some noise sampled from a noise process $\mathcal{N}$ to the actor policy

$$\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N} \tag{8}$$

This noise $\mathcal{N}$ should be temporally correlated for an efficient exploration in systems with inertia, we therefore use an Ornstein-Uhlenbeck process (Uhlenbeck & Ornstein, 1930) as suggested by Lillicrap et al. (2015).

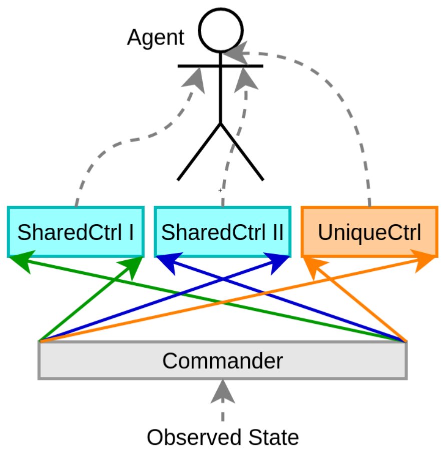

Figure 2. Two controllers with shared weights control the left and right arm of a humanoid robot. An independent third controller is in charge of its neck.

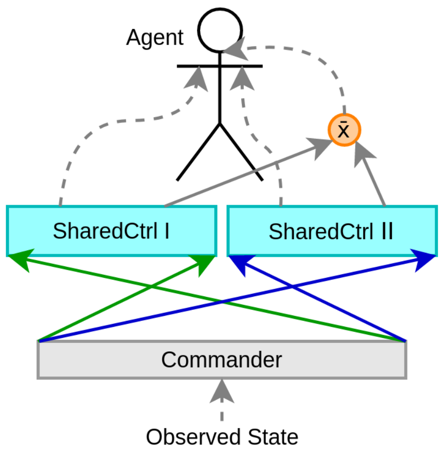

Figure 3. Redundant output averaging allows for incorporating unique joints into shared controllers that are in charge of structural equivalent groups of joints.

4. Roboschool Environment

In this work, the tasks that the agents are trained and tested on are the Roboschool robot simulations from OpenAI [1], built into environments in the OpenAI Gym toolkit (Brockman et al., 2016). The agents are trained with the intention to learn to control robots to move. The robot types are fairly varied in type and complexity, featuring a 4-legged ant, a 2-dimensional walker and a humanoid, among others. Some examples are pictured in Figure 1. The OpenAI Gym framework allows for easier experimentation with RL agents, which is enabled by easy access to the agent-environment loop. In this loop, at each timestep the agent (our algorithm) issues an action to the environment managed by the Gym framework, after which an observation and reward is returned back to the agent. Helpful information as to the completion or failure of tasks is also returned, such as a robot completely toppling over.

The data circulated around this loop is entirely environment dependent. In the case of robots, the observation about the environment would be the joint angles and joint velocities of the robot. Available actions are to apply torque to the joints, and the rewards are specific to the environment. For example, the most basic environment supplies a reward based on the distance moved in a single direction, but there are other environments that give rewards for moving to target locations that are randomly relocated. Negative rewards or penalties are also given in certain circumstances. In the basic humanoid robot walking environment (which simply rewards based on distance moved), negative rewards are given when contact with the floor with body parts other than the feet occur. This reward function is also sometimes more complex to encourage more realistic behaviour, such as that of the humanoid robot which subtracts the energy cost of actions from the reward.

5. Proposed Policy Network Architectures

In recent robot locomotion baselines, policy networks are fully connected. Their output layers produce all joint's actions together. As a first step, we introduce local connectivity to the output facing side of the network. The policy network still starts with fully connected layers but at some point splits up into different locally connected subnetworks. These subnetworks ultimately output the actions for a specific group of joints. We call the fully connected input subnetwork *commander* and the locally connected ouput subnetworks *controller*.

We can decouple joints by assigning them to different controllers. Thereby a group of joints that belong to a structural part of a robot, e.g. the arm of a humanoid, can be assigned to the same controller. Next, we try to exploit the structure in a robot by sharing weights between controllers that control structurally equivalent parts. As sketched in Figure 2, this can be combined with non-shared controllers that control structural parts without equivalent counterparts.

This means that unique joints without equivalent counterparts cannot be coupled with joints that have structural equivalencies in their shared controllers. This independence may not be desired, e.g. it could be useful to include the humanoid's neck joint in the shared controllers of both arms. We suggest a solution to this that we call *redundant output averaging*. The output for the desired unique joint is included in each shared controller redundantly and gets averaged over all controllers. This averaging shown in Figure 3 is just a computational step through which the error can easily be backpropagated in the training phase.

6. Hypotheses and Research Questions

In reinforcement learning, an episode corresponds to a coherent simulation in which reward is accumulated. For deep reinforcement learning approaches and algorithms, there are different quality measures that apply. One is an agent's learning speed of obtaining accumulated reward in subse-

[1] https://github.com/openai/roboschool

quent episodes. Since in many environments successful episodes are longer than failed ones, we are interested in the number of single simulation steps. If a task is unsolved or unbounded, the highest achieved reward can be an important measure independent of the learning time spent. Lastly, deep reinforcement learning algorithms tend to yield highly varying results, both with respect to different initializations, as well as within a single training run. This makes the robustness of an approach to another often investigated measure of quality.

Our main hypothesis is that weight sharing with respect to structurally equivalent groups of joints can improve learning for certain robotic agents as a form of instantaneous transfer learning. The term 'improving' relates to the stated quality measures in comparison to a similar agent trained with an equivalent fully connected policy network.

To be able to verify our main hypothesis for a certain robotic agent, we have to define the notion of equivalency between a fully connected and a weight sharing policy network. For a given fully connected network, we define the naive equivalent weight sharing network as the one with the same layer depth and (summed up) hidden dimensionality. This network has fewer free parameters than the original fully connected one, so we see it as the lower bound on the equivalent weight sharing network. Scaling the weight sharing network's hidden dimensionality up to the point where its parameter count equals the parameter count of the original fully connected network gives us an upper bound on the equivalent weight sharing network. Finding a weight sharing policy network that outperforms a baseline network while being in its equivalency bounds would validate our hypothesis for a particular agent and task.

Optionally, if previous experiments are successful, we can evaluate our second hypothesis: The learned hierarchy induces that the learned controllers are independent of high level tasks. This would mean that solving different abstract tasks like capture the flag and navigating in a maze would work with similar controllers but a different commander. Transferring the controllers from one task to another would then speed up the learning process.

7. Experiments

We start with hyperparameter exploration on the inverted pendulum task. This toy task is very efficient to simulate and comparatively fast to solve. It is therefore suitable to be used for a broad exploration of the hyperparameter space in order to obtain general insights on the influence of DDPG's hyperparameters. After that, we run the more complex ant task with the best hyperparameters found before, to obtain baseline results, to which we can compare later work.

We evaluated different hyperparameter choices for the DDPG algorithm on a task where a inverted pendulum is attached to a cart that can be pushed left and right. The goal is to balance the pole in an upright position. The hyperparameters we considered were: the learning rate that

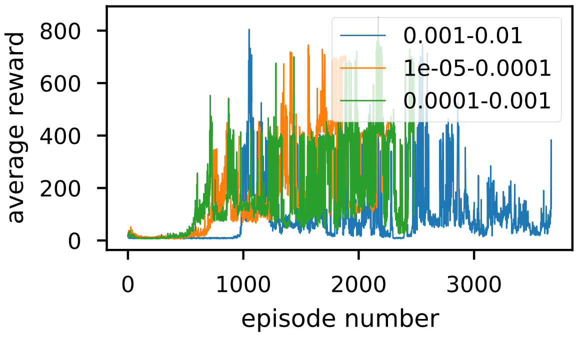

Figure 4. Average reward for varying learning rates, in the legend the first number denotes the actor's learning rate and the second critic's.

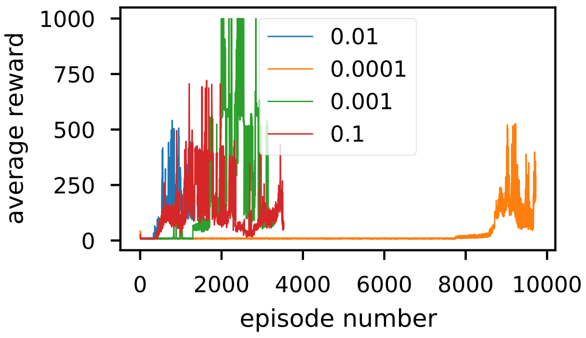

Figure 5. Average reward for varying τ.

we used for the Adam optimizer of the actor and critic networks, the parameter τ for the soft updates of the target networks and the discount factor γ.

We compared the hyperparameters used by Lillicrap et al. (2015). The comparison was based on the development of the cumulative reward in an episode over time in training. The best reward that could be achieved was 1000. After each episode we ran the policy for another episode without added noise and without training to determine its performance. Due to random initialization of the neural networks and the randomness in exploration in reinforcement learning, there is a high variance in the performance across runs. Therefore, we averaged the results over 3 runs to get a better estimate of the expected performance. We used neural networks with two hidden layers consisting of 40 hidden units each for the actor and the critic. We ran each simulation for a fixed number of steps, since episodes can have different length, the plots of the rewards per episode end at different positions.

The results for the learning rates are given in Figure 4. The learning rates used by Lillicrap et al. (2015) were 1×10^{-4} for the actor network and 1×10^{-3} for the critic network. We stuck to this difference between the learning rates and evaluated two other pairs (1×10^{-5}, 1×10^{-4}) and (1×10^{-3}, 1×10^{-2}). We found that the proposed learning rates from Lillicrap et al. (2015) indeed yielded the best performance.

Figure 5 shows the results for varying τ which influences how fast the target networks are updated. It shows that for small τ the system needs a lot episodes until it learned

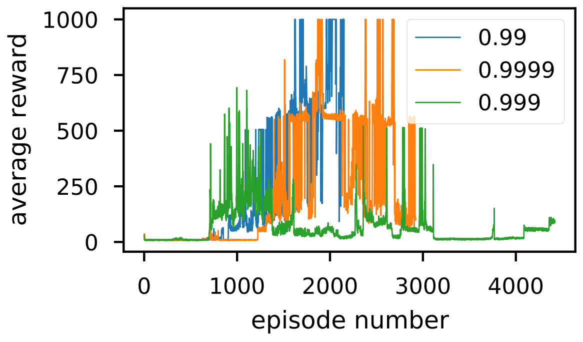

Figure 6. Average reward for varying discount rates γ.

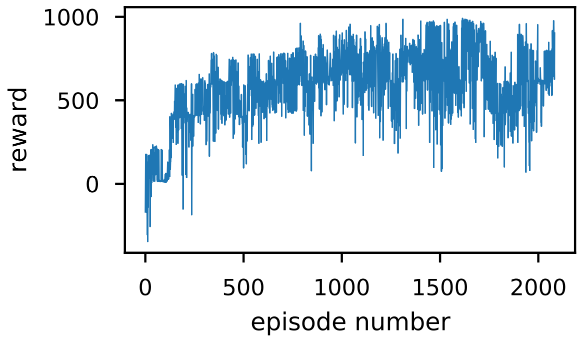

Figure 7. Reward per episode for the fully connected DDPG ant baseline.

something and also does not reach as high rewards compared to smaller τ. Setting $\tau = 0.001$ seems to yield the best performance, as this is the only setting in which the average reward reaches the maximum of 1000.

In Figure 6 we compared different discount factors γ which influence the time horizon of the reward expectation. Setting $\gamma = 0.99$ or $\gamma = 0.9999$ seemed to give the best results, where $\gamma = 0.99$ required less episodes until it reaches an average reward of 1000. The performance for $\gamma = 0.999$ is quite unexpected since this value lies between the other two. We believe that this is a result of randomness in performance. Since $\gamma = 0.99$ yielded reasonable results and is the proposed value by Lillicrap et al. (2015), we used this value in following experiments.

To create a baseline for further work, we used the best hyperparameters found in our experiments and otherwise the parameters used by Lillicrap et al. (2015). Furthermore, we used fully connected neural networks with 400 hidden units and two hidden layers for the Roboschool ant task. The results are shown in Figure 7 and they show that this baseline is able to learn and is even more stable than the results of the hyperparameter experiments. We believe that the reason is that a single wrong action can lead to a failure in the inverted pendulum task, whereas the ant walking task is more forgiving.

Overall, we found that there is indeed a high variance in the performance of the algorithms for all parameter choices. In most cases, the reinforcement learning system unlearned already learned behavior which resulted in poor performances after the maximum reward has already been reached. The experiments from Lillicrap et al. (2015) do not show this behavior, thus this problem will be further investigated in further work.

8. Interim Conclusion

In this report, we introduced the reinforcement learning setting with the Roboschool simulation and the deep reinforcement learning methodology. We presented our idea of low-level controllers based on subnetworks that are locally connected and can share weights. Furthermore, we stated our hypothesis that weight sharing with respect to structural equivalent joints in a robot can improve learning.

For our experiments we implemented DDPG on our own in PyTorch[2]. This task was harder than expected, because the information given in (Lillicrap et al., 2015) is not sufficient for a straight forward implementation. The experiments that we have done on the inverted pendulum task have shown us that deep reinforcement learning is quite sensitive to hyperparameter settings, because bad hyperparameters can lead to no learning at all. In addition, the experiments showed that there is a high variance in the performance and already learned behavior can be unlearned.

Finally, our hyperparameter exploration experiments enabled us to find a first baseline for one of our target robots. Experimenting with these more sophisticated robots turned out to be feasible if the hyperparameter space can be narrowed down beforehand. Since we only change the network structure, this indicates that we should be able to train agents with our proposed policy network structure based on this hyperparameter setting. It might be necessary to reevaluate some hyperparameters for much more complex tasks such as the humainoid walker.

9. Further Work

In the next step we will construct and evaluate weight sharing policy networks for Roboschool's 2D walker's, ant's and humanoid's forward moving tasks. The ant has the interesting property of having four similar legs. Thus a policy network containing four weight sharing controllers can be constructed. A further hypothesis is that the ant therefore profits the most from weight sharing policy networks. The humanoid is interesting for its complexity. It is the only agent in our selection, that features unique joints without any counterparts. Because of this we can evaluate the performance of redundant output averaging on it.

The risk that the tasks can not be solved at all with weight sharing policy networks is relatively low. With a deep enough fully connected commander, the shared controller can always at least learn the identity function. The result would be the equivalent of a plain fully connected deep policy network. If we are not able to improve upon this, we will at least report our approaches.

[2]http://pytorch.org/

References

Brockman, Greg, Cheung, Vicki, Pettersson, Ludwig, Schneider, Jonas, Schulman, John, Tang, Jie, and Zaremba, Wojciech. Openai gym, 2016.

Heess, Nicolas, Wayne, Gregory, Tassa, Yuval, Lillicrap, Timothy P., Riedmiller, Martin A., and Silver, David. Learning and transfer of modulated locomotor controllers. *CoRR*, abs/1610.05182, 2016.

Heess, Nicolas, TB, Dhruva, Sriram, Srinivasan, Lemmon, Jay, Merel, Josh, Wayne, Greg, Tassa, Yuval, Erez, Tom, Wang, Ziyu, Eslami, S. M. Ali, Riedmiller, Martin A., and Silver, David. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017. URL http://arxiv.org/abs/1707.02286.

LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *nature*, 521(7553):436, 2015.

Lillicrap, Timothy P., Hunt, Jonathan J., Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. URL http://arxiv.org/abs/1509.02971.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, feb 2015a. ISSN 0028-0836. doi: 10.1038/nature14236. URL http://www.nature.com/doifinder/10.1038/nature14236.

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

Sharma, Arjun and Kitani, Kris M. Phase-parametric policies for reinforcement learning in cyclic environments. In *AAAI Conference on Artificial Intelligence*, February 2018.

Siciliano, B, Sciavicco, L, Villani, L, and Oriolo, G. Robotics–modelling, planning and control. advanced textbooks in control and signal processing series, 2009.

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic Policy Gradient Algorithms. *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 387–395, 2014. ISSN 1938-7228. URL http://proceedings.mlr.press/v32/silver14.pdf.

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

Uhlenbeck, George E and Ornstein, Leonard S. On the theory of the brownian motion. *Physical review*, 36(5): 823, 1930.

Watkins, Christopher JCH and Dayan, Peter. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.