2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)
Sheraton Arabella Park Hotel, Munich, Germany, July 3-7, 2017

Learning to Walk with Prior Knowledge

Martin Gottwald and Dominik Meyer and Hao Shen and Klaus Diepold

Abstract— **In this work a novel approach to Transfer Learning for the use in Deep Reinforcement Learning is introduced. The agent is realized as an actor-critic framework, namely the *Deep Deterministic Policy Gradient* algorithm. The Q-function and the policy are represented as deep feed-forward networks, that are trained by minimizing the mean squared Bellman error and by maximizing the expected reward, respectively. For Transfer Learning, the actor is modified with a new regularization term, called the *knowledge regularizer*. It allows to include prior knowledge in from of an existing policy in the learning process. The *knowledge regularizer* shifts the current weight vector during the gradient descent step towards a region of the weight space, that is centered around the existing policy. Because neural networks are universal and smooth function approximators, the weights of the existing policy and the new ones have to lie close to each other in the weight space. Solving a task therefore benefits from the prior knowledge, when it is used to manipulate the gradient given by the critic.**
We could experimentally verify, that the *knowledge regularizer* results in a higher performance achieved by the agent and in a reduction of the learning time. Furthermore, the *knowledge regularizer* can be used as a replacement for labeled training data, which renders it especially useful for physical applications.

I. INTRODUCTION

The creation of intelligent agents, that are able to make decisions on their own in arbitrary environments, has been of great interest for many decades. Classical machine learning approaches limit the agent to predefined situations. Thus the agent is unable to make proper decisions in unexpected situations, that have not been handled in advance. Reinforcement Learning enables the agent to learn automatically on a trial-and-error basis. Recent breakthroughs in the application of deep non-linear function approximation architectures in the Reinforcement Learning domain now allow the agent to process raw data directly. This renders the algorithms usable for real world problems.
A new problem arises with the need of labeled training data. It is used to train the deep function approximation architectures in advance, until a basic performance level is obtained. The actual problem is, that for real world applications the construction of appropriate training data is a very costly process. But solving a complex task can become challenging, if not impossible, for an agent if no prior data is available.
Transfer Learning techniques can be used to avoid the need of training data. Instead of solving a complicated task directly, the agent first learns with easier versions of this task, that do not require the existence of training data. The goal is then to master the original task by reusing the obtained skills. A significant speed-up can be achieved, because the agent does not have to learn the same things twice.

Department of Electrical and Computer Engineering, Technische Universität München, Arcisstr. 21, 80333 Munich, Germany

II. RELATED WORK

In [1] a survey about state-of-the-art methods for Transfer Learning in the Reinforcement Learning domain is presented. In the following, only those techniques are considered, where the source and the target task share the same state-action space.
The first mentioned approach is centered around altering the transition model of a MDP. First, the agent trains with an easier version of the task to solve. After that, it is able to learn the original version in less time than it would be required to learn it directly. The problem of this approach is, that one cannot say in advance, how long the agent has to train on the easy task and when to change to the original one.
In a second approach a task is decomposed in easy to solve subtasks. Thereby, skills that occur in several subtasks, only have to be learned once. While this method ensures a reduced training time, it is only applicable in problems, where a clear decomposition of the task is defined.
For discrete action spaces it has been proposed to reduce the amount of available actions per state. The agent has to select the action for the exploration step from less candidates and therefore can learn faster. However, it remains unclear, how this process could be automated without relying on the knowledge of the transition model of the world. Furthermore, this technique is not suited for continuous action spaces.
The last presented technique in the survey is called *Region Transfer*. The agents collects experience in several source tasks in advance. Then it starts to interact with the environment defined in the target task. Trajectories through the state-action space are matched to the stored experience. The most similar source task is used to transfer a skill.
In [2] an approach to Transfer Learning based on intrinsic motivation is described. The agent creates a representation of the transition model in form of a graph. The nodes correspond to clusters in the state space and the edges between the nodes belong to the basic skills, that moves the agent from cluster to cluster. A special internal reward signal encourages the agent to discover new clusters in the state space by using the created skills or to improve the quality of an existing skill. Similar problems as before arise. One cannot say in advance, how much time the agent needs to build a good representation of the world.
In [3], [4] or [5] training the agent from demonstration is realized. The required time for learning a policy can be significantly reduced, because the agent does not have to rely

978-1-5090-6000-9/17/$31.00 ©2017 IEEE

on a random exploration step. The training data consists of transitions in the state-action space in the form of (s, a, r, s') tuples. These tuples belong normally to trajectories, that generate high rewards, and are directly shown to the agent. The training data can come from various sources. For example, a human supervisor can control the agent by hand or a feed-back controller could execute a sub-optimal policy to generate those tuples. In any case, creating the training data is an expensive process that easily becomes infeasible.

III. KNOWLEDGE REGULARIZATION

A. Deep Reinforcement Learning

We consider a standard Reinforcement Learning problem, which is modeled as a Markov Decision Process (MDP). A comprehensive introduction is available in [6]. The state and action space are vector spaces over $\mathbb{R}$, i.e. $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ and $\mathcal{A} \subseteq \mathbb{R}^{n_a}$ for $n_s, n_a \in \mathbb{N}$. The reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ maps transitions between states to scalar rewards. The successor state can be sampled from the transition model $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ of the environment. A discount factor $\gamma \in [0, 1]$ balances the immediate reward with long term rewards. We use $\gamma = 0.99$ for all experiments. The Q-function maps state-action tuples (s, a) to their expected reward and is defined as

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\substack{a_t \sim \pi(s_t) \\ s_t \in \mathcal{S}}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \middle| s_0 = s, a_0 = a \right]. \tag{1}$$

The Q-function obeys the Bellman equation

$$\begin{aligned} Q^\pi(s,a) &= \mathop{\mathbb{E}}_{s' \in \mathcal{S}} \left[r(s,a,s') + \gamma \cdot \mathop{\mathbb{E}}_{a' \sim \pi(s')} \left[Q^\pi(s', a') \right] \right] \\ &=: \mathcal{T}_\pi Q^\pi(s,a) \end{aligned} \tag{2}$$

and can be learned by successive application of the Bellman operator $\mathcal{T}_\pi$ onto an initial Q^π. The definition of $\mathcal{T}_\pi$ in (2) for Q-functions is analogous to the case with value-functions.

The basis of the Deep Reinforcement Learning agent is the *Deep Deterministic Policy Gradient* algorithm [7]. The Q-function $Q^\pi_\omega : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and the policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$ are represented as two deep feed-forward neural networks. The training of the networks is done with the actor-critic framework. The critic learns the Q-function under the current policy by minimizing the mean squared Bellman error. The performance objective, which is used in a gradient descent algorithm, is given by

$$\begin{aligned} J(\omega) &= \mathop{\mathbb{E}}_{s,a,s' \sim \mathcal{E}} \left[\left(\mathcal{T}_\pi Q^\pi_\omega(s,a) - Q^\pi_\omega(s,a) \right)^2 \right] \\ &= \mathop{\mathbb{E}}_{s,a,s' \sim \mathcal{E}} \left[\left(r + \gamma \cdot Q^\pi_\omega(s', \pi_\theta(s')) - Q^\pi_\omega(s,a) \right)^2 \right]. \end{aligned} \tag{3}$$

Due to the use of deterministic policies, the inner expectation over the actions can be removed.

The actor improves the policy over time by maximizing the expected reward. The performance objective for the actor is therefore defined via the critic as

$$J(\theta) = \mathop{\mathbb{E}}_{s \sim \mathcal{E}} \left[- Q^{\pi_\theta}_\omega(s,a)\big|_{a \sim \pi_\theta(s)} \right]. \tag{4}$$

The minus sign is necessary to exchange rewards with costs, which brings the performance objective in line with gradient descent algorithms, that always try to minimize errors.

The techniques described in [7] are used to achieve a stable training of the networks. Additionally, the magnitude of the gradient of (3) with respect to the critic's weights ω is upper bounded. The quadratic penalization in (3) is for that purpose replaced by a more complex function $f : \mathbb{R} \to \mathbb{R}$

$$f(\delta) \coloneqq \begin{cases} -\sigma & \delta < -\sigma \\ \frac{1}{2}\delta^2 & \delta \in [-\sigma, \sigma] \\ \sigma & \delta > \sigma \end{cases}.$$

This function $f(\cdot)$ quadratically penalizes the input only in the interval $[-\sigma, \sigma]$. Outside of this range the function is extended linearly. This modification of the performance objective renders the learning progress more steady over time, because larger changes in the weights due to large gradients are no longer possible.

B. Transfer Learning

The definition of a Transfer Learning scenario is very intuitive. There exist two tasks for the agent to solve. Each task corresponds to an instance of a MDP. The agent trains in one task first, which is called the source task, and learns a certain skill. Then, this skill can be used during the solving of the target task. The goal is to reduce the required learning time by exploiting the existing skill. It is of course possible, that the target task cannot be learned directly.

The most promising approaches to Transfer Learning are centered around manipulating the exploration process of the environment by the agent. Instead of using random policies, exemplary trajectories through the state-action space are demonstrated to the agent. By this method, the agent perceives, which transitions in the state-action space lead to high rewards, and can adapt the policy accordingly. As soon as a basic performance level of the policy is obtained, the agent can learn on its own. However, the application of this approach is very limited, because it relies on the existence of exemplary trajectories, i.e. labeled training data. The creation of this data is normally a very expensive process or even impossible.

The new approach presented in this paper is called *knowledge regularization* and is based on the manipulation of the underlying optimization problem.

Neural networks are universal function approximators and are smooth in their output and in their network weights. For that reason we assume, that two weight vectors, that are lying close to each other in the weight space, correspond to similar policies. In a Transfer Learning scenario, where the policy for the source and target task should share some common properties, it is an intuitive idea to use the policy from the source task to guide the search for the optimal

weights in the target task. We realize this idea as a novel regularization term for the performance objective of the actor. It is calculated from the current network weights and the prior knowledge θ^K

$$J^K(\theta) = J(\theta) + \lambda \cdot \sum_i \left(\theta_i - \theta_i^K\right)^2. \quad (5)$$

The prefactor λ can be used to control the impact of the regularization. It introduces an additional hyper-parameter that needs some careful tuning. θ^K are the network weights of the existing policy, that has been learned in the source task. The *knowledge regularizer* resembles a radial symmetrical attractor, that pulls the current weight vector θ towards the region of the weight space, where the source task has been solved before with the policy defined by θ^K. A small limitation is, that the *knowledge regularization* is restricted to a single specific network architecture, that has to be used in both the source and target task. Otherwise the distance term would not be defined properly anymore. The new performance objective is of course still non-convex, hence the gradient descent algorithm is likely to find only a local minimum. For that reason, all experiments are repeated several times with randomized initial weight vectors to produce meaningful statistical results.

We restrict the considered Transfer Learning scenarios to those with the same state-action space in all tasks. Learning the mapping between two different state-action spaces in order to transfer a certain skill is beyond the scope of this paper. Additionally, the reward function remains unchanged across all tasks, because it has a too strong influence on the Bellman error. If the actor benefits from a transfered skill, the critic would disturb the actor because of the completely wrong direction of the estimated gradient. Thus only the transition model of the MDP is allowed to change.

IV. EXPERIMENTS

A. Experimental Setup

An environment for evaluating the new approach to Transfer Learning is contained in the *OpenAI Gym* package [8]. Specifically, the two dimensional bipedal walker simulation *BipedalWalker*, which is based on the physics engine *Box2D*[1], is used for the experiments. In this world, the agent has to control the torques of four motors in order to reach the goal on the right side. The current state contains the physical properties of the walker (e.g. the joint angles) and the measurements of a laser range finder. In Fig. 1 an image of the environment is shown. Please refer to the documentation of the package for a full description.

In order to evaluate the Transfer Learning capabilities of the *knowledge regularization*, two independent tasks have to be created. They serve later on as a source and target task. For the bipedal walker, this is realized by altering the overall mass of the walker. This introduces different moments of inertia and therefor corresponds to a changed transition model of the MDP. One task is given by the walker with the original mass, another one is available through a walker with only half the mass. It has to be tested, whether this change in the transition model defines two clearly separated tasks. For that purpose, the agent is trained on the heavy walker first and is tested afterwards on the walker with the reduced mass. The learned policy results in oversteering of the joints due to the wrong mass and the agent struggles after a couple of steps. In the other direction, where the agent trains a policy for the light walker, the produced torques are too small for the heavier joints, the walker remains idle.

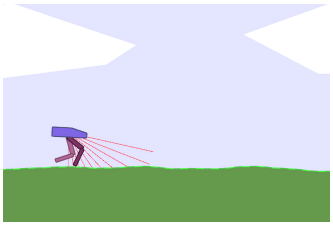

Fig. 1. The *BipedalWalker* environment of the *OpenAI Gym* package. The red lines visualize the laser range finder.

To compare the learning progress of various experiments, a score is defined as a scalar value in the range $[0, 1]$. The score is the percentage of the highest achievable performance of a policy and allows us to introduce a rating for them, i.e. $\pi_1 < \pi_2 \Leftrightarrow score(\pi_1) < score(\pi_2)$. With the score, the learning progress of the agent can be visualized easily and is independent of the expected reward. The Q-function can only be used to measure the performance of a policy, if the network weights correspond to a global minimum in the Bellman error. However, in the actor-critic framework, the policy and the Q-function are learned simultaneously. Hence the output of the critic might not be reliable at the beginning. In the bipedal walker simulation the score is defined as the average value of two sub-scores. One sub-score is defined via the distance of the walker's center of mass to the goal. Only the x-component is used. This sub-score is zero, if the x-position is negative and one, if the agent is beyond the goal. The second sub-score is calculated from the accumulated one-step rewards of an episode without using a discount factor. The world is designed by the authors in such a way, that an agent receives about 300 reward per episode on average only with an energy efficient policy. The sub-score is the accumulated reward divided by the average one, clipped at zero and one. The overall score is a suitable performance measure for the experiments. In order to get the full score, the agent has to learn a repetitive walking pattern, which is robust enough to handle various landscapes, and it has to be energy efficient at the same time.

The achieved score depends strongly on the initialization of the network weights and on the initial state of the environment. In order to produce statistical meaningful statements, every score is calculated ten times in a row for each epoch. The ten values are combined to a mean, minimum and maximum score. These three quantities describe the performance of the policy independent of the start state of the world. The impact of the initial network weights can be removed by repeating the whole experiment several times.

[1]Box2D: `https://github.com/erincatto/Box2D`

The mean, minimum and maximum scores of every repetition are then averaged and produce the final graphs shown in the following sections.
The average minimum score shows directly the quality of the learning algorithm. The higher this value, the better the performance of the worst performing policy. In the same way the average maximum score reveals, whether the agent can learn a good policy at all. For the best case, the average score is high and the area spanned by the upper and lower bound is very small.

B. Transfer Learning Capabilities of the Knowledge Regularizer

In the first experiment, the *knowledge regularizer* is used to transfer a skill from the walker with the original mass to the one with the reduced mass. A policy, that serves as prior knowledge, is created by learning from demonstration. The bipedal walker simulation contains a hard-coded policy, that is able to produce a sub-optimal walking pattern. This hard-coded policy is executed on the walker to generate (s, a, r, s') tuples, which are use to fill the agent's memory. With this technique, several policies are created and investigated by hand. Due to the demonstration, it is very likely that the agent learns several good performing policies. The weights of a policy, that achieves a high performance over many repetitions, are chosen to be the prior knowledge θ^K.
Now the impact of the *knowledge regularizer* is evaluated for different values of λ. The tested values are: $\lambda \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. The remaining hyper-parameters are kept constant. Fig. 2a contains the score of a reference run, where the agent learns without any Transfer Learning techniques. The scores, that are obtained with the *knowledge regularizer*, are shown in the Fig. 2b to 2f. In order to highlight the results, the plots are extended with three guide lines (inspired by [1]). The *time-to-threshold* denotes the number of epochs required to reach a certain threshold in the score. In the case, where the threshold is exceeded, this is shown as the black dash-dotted line. The *asymptotic performance* stands for the average score in the last quarter of the training process and is marked with the red solid line. Additionally, the *minimum asymptotic performance* is calculated similarly using the lower bound of the scores and is shown as a green dashed line.
The tests reveal, that the *knowledge regularizer* has the desired effect. The optimal value for the strength of the regularizer is $\lambda = 10^{-1}$, which corresponds to Fig. 2c. The usage of the prior knowledge on the lighter walker boosts the learning progress so much, that the score exceeds the threshold of 0.9 after 148 epochs. This is not possible in the reference run, as one can see in Fig. 2a. The *asymptotic performance* and the *minimum asymptotic performance* are increased by 0.13 and 0.29 respectively. A noteworthy result is, that around epoch 370 in Fig. 2c the average minimum score takes a value of ≈ 0.95. This shows, that all repetitions of the experiment result in very robust policies, that achieve good scores independent of the initialization of the environment and the network weights.
If $\lambda \geq 10^0$, the influence of the *knowledge regularizer* is too strong and hinders the learning. In Fig. 2b the according score is shown. The *asymptotic performance* is with 0.66 smaller than the reference value. Also the upper bound on the score is significantly lower, meaning that all produced policies perform worse than when learning from scratch.
The score of the agent drops also for too small values of λ. While $\lambda = 10^{-2}$ still results in a better score over time than without using prior knowledge, the experiments with $\lambda = 10^{-3}$ and $\lambda = 10^{-4}$ are almost indistinguishable from the reference experiment. In both cases the *asymptotic performance* is increased by only 0.01. However, the threshold of 0.9 is still exceeded within the considered timespan during single epochs.
An interesting artifact is visible in Fig. 2d. After epoch 250, the average maximum score first decreases to 0.8 and raises again to the original value after 100 epochs. This occurs, because two of the ten repetitions of the experiment have a sharp decline in their scores during those epochs, while the others remain at their level.
In the second experiment, the reverse direction is tested. The source task is now the walker with the half mass and is used to generate the prior knowledge θ^K. The target task is given by the original walker. The prior knowledge θ^K is chosen from the experiment shown in Fig. 2a. A well performing and robust policy is selected. The strength of the *knowledge regularizer* is set to $\lambda = 10^{-1}$. Fig. 3a shows again a reference run. In Fig. 3b the score is shown, that is achieved by the agent, when it learns from demonstration. In Fig. 3c the score obtained with the *knowledge regularizer* is presented.
The development of the score in Fig. 3c shows, that the *knowledge regularizer* still works with the reverse direction, but the agent is not as good as when learning directly from demonstration (Fig. 3b). Compared to the reference score in Fig. 3a, the agent achieves a higher *asymptotic performance*. In particular right at the beginning, the regularizer outperforms the learning from scratch. For later epochs, the development of the score is more similar to Fig. 3b, where the replay memory has been filled in order to learn walking. The reason, that the effect of the *knowledge regularizer* is not as pronounced as before, is, that learning on the heavier walker is the more difficult task. This can be seen when comparing the scores of Fig. 3a and Fig. 2a. Learning from scratch on the walker with the original mass progresses much slower than with half the mass and already saturates within the considered time span at a lower *asymptotic performance*.
The second experiment highlights the importance of the *knowledge regularization*. If no training data would be available in the world, the agent could not learn directly to walk with the original walker. However, altering the transition model of the environment to create an easier task is at any time possible without much effort. The agent is able to find a walking policy for the easier task on its own from scratch. With the *knowledge regularizer* it is now possible to include the existing policy in the original task and the agent can learn to walk with the heavier walker without the need of

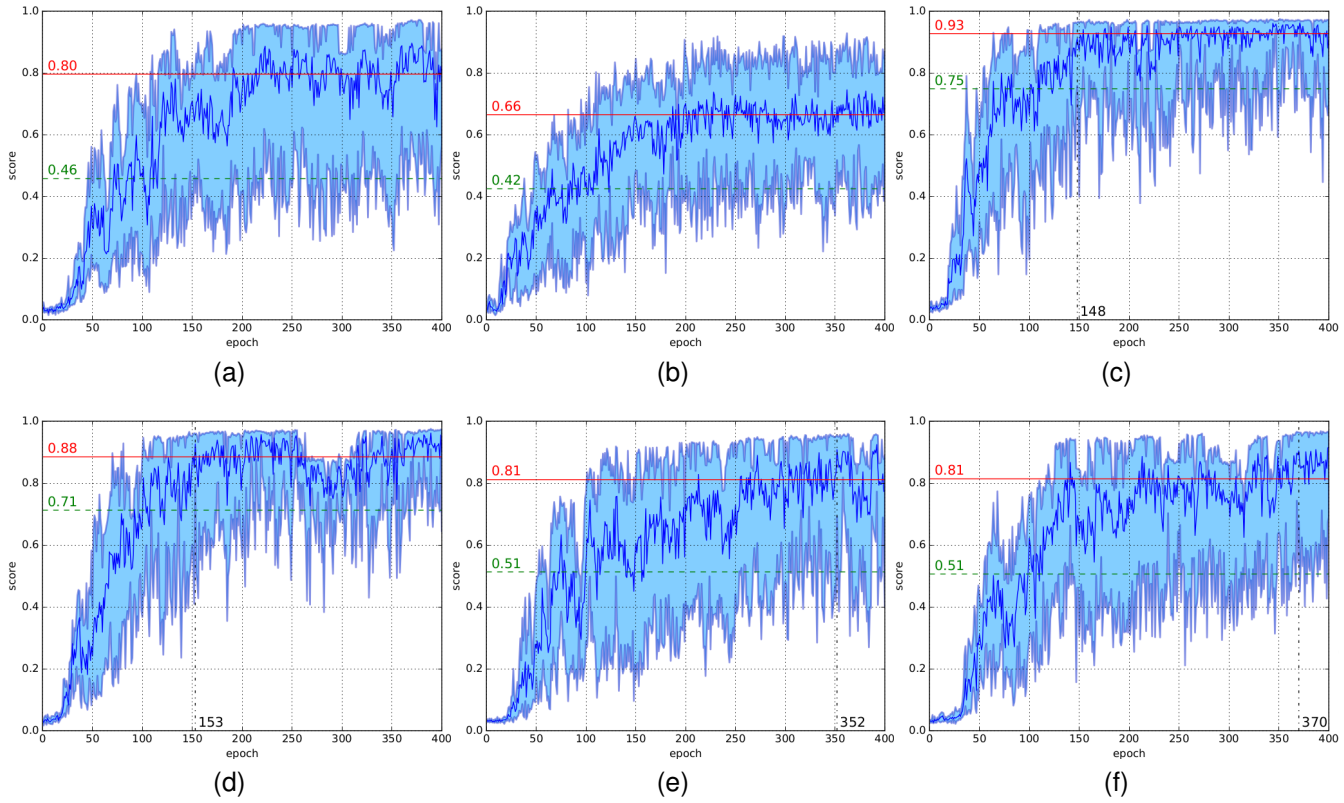

Fig. 2. The development of the scores depending on the completed epochs. An epoch denotes the combination of several training episodes and the calculation of the score. The score reflects the performance of the agent as a proportion of the maximum possible performance. It is calculated several times and is plotted as the average, minimum and maximum score to allow for statistical meaningful statements. **(a)**: A reference run with the half mass for the walker. The agents learns from scratch without help. The *time to threshold* is omitted, the score never exceeds 0.9. **(b)** to **(f)**: The scores created with the knowledge regularizer. Fig. (b), (c), (d), (e) and (f) correspond to $\lambda = \{10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. In (c), for $\lambda = 10^{-1}$, the agent performs significantly better than in the reference run.

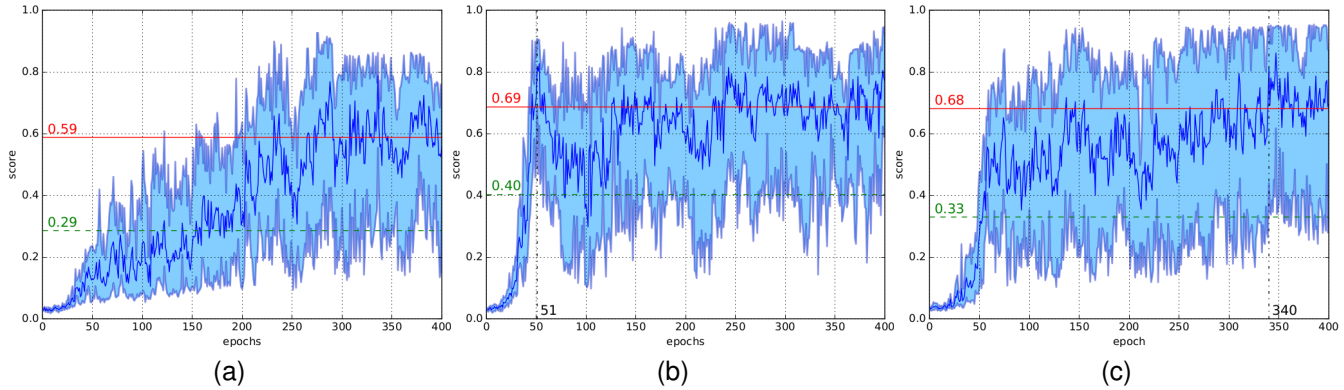

Fig. 3. The development of the scores depending on the completed epochs. An epoch denotes the combination of several training episodes and the calculation of the score. The score reflects the performance of the agent as a proportion of the maximum possible performance. It is calculated several times and is plotted as the average, minimum and maximum score to allow for statistical meaningful statements. **(a)**: A reference run with the orig. mass for the walker. The agents learns from scratch without help. The *time to threshold* is omitted, the score never exceeds 0.8. **(b)**: The agents learns from demonstration and achieves higher scores than in the reference run. In particular, after 51 epochs, the threshold of 0.8 is exceeded. **(c)**: The agent is trained with the *knowledge regularizer*. No training data is involved. The agent performs better than when learning from scratch and is able to reach the threshold of 0.8 after 340 epochs.

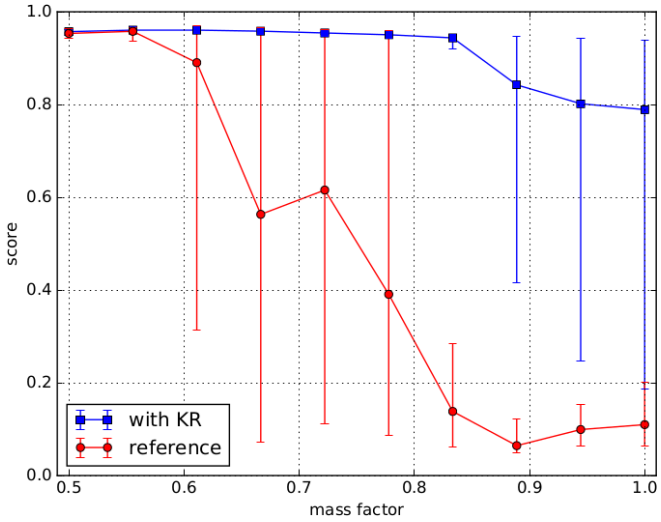

Fig. 4. The development of the scores depending on the mass of the walker. The score reflects the performance of the agent as a proportion of the maximum possible performance. It is calculated ten times for every policy and is plotted as the average score with error bars, that indicate the minimum and maximum score that have occurred. Two policies are trained, one with the knowledge regularizer with $\lambda = 10^{-1}$ (blue line) and one without prior knowledge (red line). The graphs show, that the knowledge regularizer can be used to preserve the existing skill. The average score remains high for almost all masses. The reference policy performs worse for only slight changes of the mass.

labeled training data. The threshold of 0.8 is exceeded after 340 epochs (Fig. 3c). This is not the case in the reference experiment, as one can see in Fig. 3a.

C. Preserving Knowledge

In a final experiment, an additional benefit of the *knowledge regularization* is investigated. If the prior knowledge θ^K is used to learn a new policy for a target task, this new policy should still be usable in the source task. Informally, the new policy can be defined as the prior knowledge θ^K plus something new. To test this property, two policies, that are trained for the walker with half the mass, are reused on walkers with stepwise increased masses. One of the two policies is learned from scratch without any Transfer Learning technique. The policies, that were used to generate the scores in Fig. 2a, are investigated by hand and one with a high quality is selected. The other policy is trained with the *knowledge regularizer* and is chosen from the experiment with the according score shown in Fig. 2c. Again a policy with a high performance is selected in order to provide a fair basis. In Fig. 4 the resulting scores for different masses are visualized. As before, the scores are calculated ten times to remove the impact of the initialization.

The plot shows, that the *knowledge regularizer* can be used to preserve the existing skill. The score of the reference policy has a high variance, if the mass of the walker is changed only slightly. The mean score drops below 0.2, if the mass scaling factor approaches the mass of the source task, i.e. $m = 1.0$. The score of the policy, that is generated with the *knowledge regularizer*, remains high and the variance is almost zero for a large range. For the original mass $m = 1.0$ the average score is reduced to 0.8. Seven of the ten repetitions still have gained the full score, only three times less performance is achieved.

V. CONCLUSION

In this work a novel approach to Transfer Learning is introduced. The approach is based on a new regularization term, which is used in the performance objective of the actor, and is called *knowledge regularization*. Various experiments have been conducted to show the impact of the regularizer.

In the first experiment, prior knowledge is transfered from the heavy walker to the light one. For a proper chosen value for the prefactor of the regularizer, the agent outperforms a reference experiment significantly. If the regularization is too strong, the agent performs worse than the reference run. If it is too weak, the effect of the *knowledge regularizer* can be neglected.

In the second experiment, the reverse direction is evaluated. Here the source task is the walker with the reduced mass and the target task is the original one. Due to the *knowledge regularization*, the agent shows a development of the performance over time similar to the learning from demonstration. But the important difference is, that in this experiment no training data is involved, that could be used as a demonstration. The agent achieves the higher performance on the original walker solely by training on the light version first and reusing its existing experience in the target task.

A final experiment has revealed, that the *knowledge regularizer* produces a new policy, that is able to solve both the target task and the source task at the same time. The existing skills remain present.

The application of the *knowledge regularization* is not restricted to the bipedal walker or even to the robotic domain. Every situation, in which an easy to solve source task and a more complicated target task are present, can benefit from the presented approach. It has to be investigated, whether the optimal value for λ depends on the environment and the tasks to solve, or if the value $\lambda = 0.1$, which is used in this paper, is suitable for a more broad spectrum of applications.

REFERENCES

[1] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research*, 2009.
[2] J. Metzen and F. Kirchner, "Incremental learning of skill collections based on intrinsic motivation," *Frontiers in Neurorobotics*, 2013.
[3] W. D. Smart and L. P. Kaelbling, "Effective reinforcement learning for mobile robots," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, 2002.
[4] D. L. Moreno, C. V. Regueiro, R. Iglesias, and S. Barro, "Using prior knowledge to improve reinforcement learning in mobile robotics," in *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*, 2004.
[5] K. R. Dixon, R. J. Malak, and P. K. Khosla, "Incorporating prior knowledge and previously learned information into reinforcement learning," Carnegie Mellon University, Tech. Rep., 2000.
[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.
[7] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *Computing Research Repository*, 2015.
[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *Computing Research Repository*, 2016.