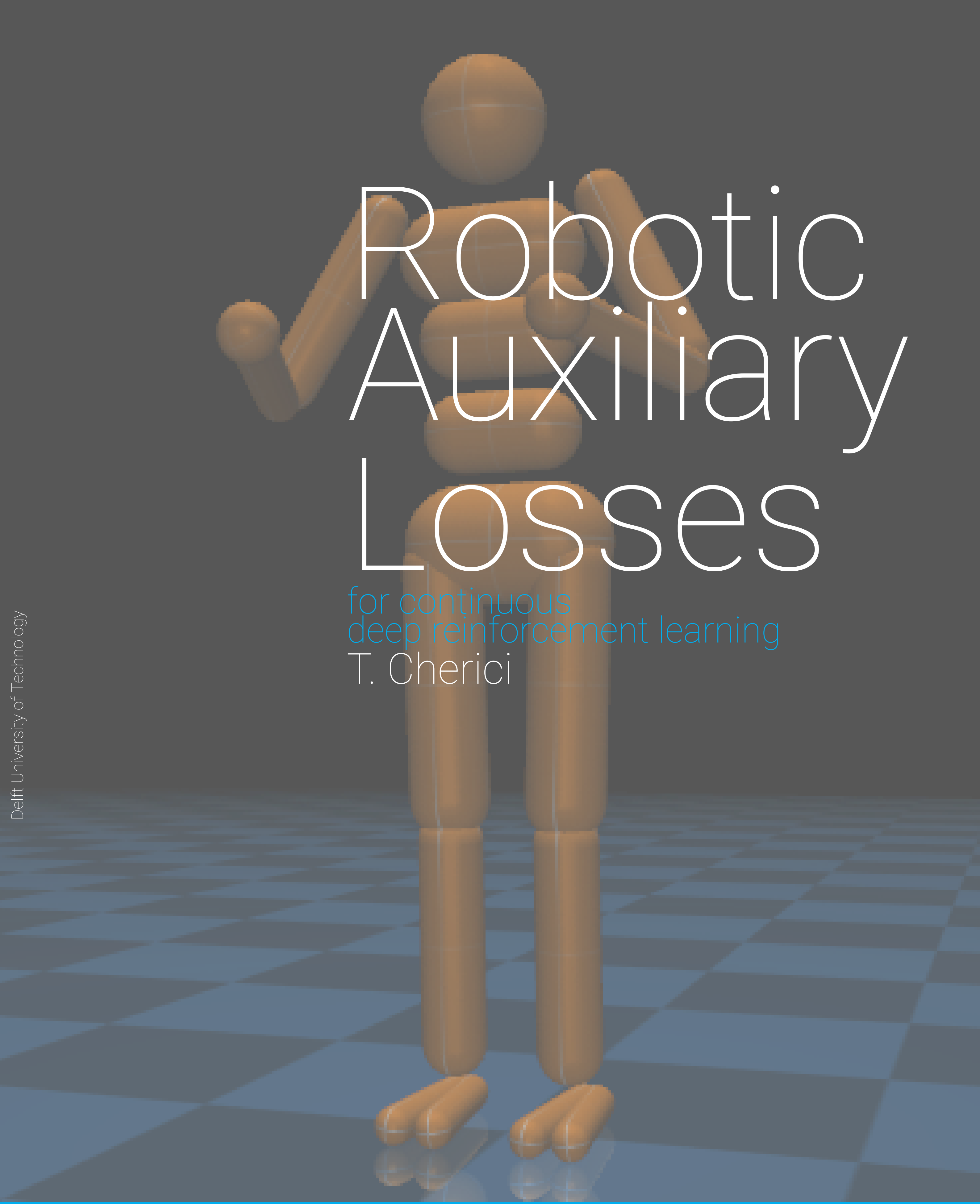

Robotic Auxiliary Losses

for continuous deep reinforcement learning

T. Cherici

Robotic Auxiliary Losses

for continuous
deep reinforcement learning

by

T. Cherici

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on the 27th of August 2018.

Student number:	4089200	
Project duration:	September 1, 2017 – August 27, 2018	
Thesis committee:	Prof. dr. ir. P. P. Jonker,	TU Delft, supervisor
	Drs. T. M. Moerland,	TU Delft, supervisor
	Dr. D. J. Broekens,	TU Delft
	Dr. ir. J. Kober,	TU Delft

Preface

The following work, *Robotic Auxiliary Losses for continuous deep reinforcement learning*, is meant as final graduation work for my Master of Science in BioRobotics at the Delft University of Technology, which I will be defending on the 27th of August 2018 at 3ME, Instructiezaal H.

This thesis is the result of 12 months of (human and machine) learning, of struggles and failures and progress and development. It may mark the conclusion of my student career, but also represents the definitive sparking of my passion for machine learning and artificial intelligence. I have chosen to present my work in the form of a conference paper, much like the many from this field that I have been reading in the past months.

First and foremost, I would like to thank my direct supervisor Thomas Moerland. I was captivated by his lectures on reinforcement learning, and enthralled by his offer to do my thesis in this exciting and quickly developing field. This thesis would not have been possible without his time, expertise and friendly support. I could not have asked for a better mentor. I would also like to thank my professor Pieter Jonker, and Joost Broekens, for their comments and suggestions. My final thanks go to my girlfriend Julia for her patience and encouragements in these sometimes hectic months, and my family and friends. Thank you all for your unwavering support.

I hope you enjoy reading.

T. Cherici
Delft, August 2018

Robotic Auxiliary losses for Continuous Deep Reinforcement Learning

Teo Cherici [*] Thomas Moerland [*] Pieter Jonker [*]

Abstract

Recent advancements in computation power and artificial intelligence have allowed the creation of advanced reinforcement learning models which could revolutionize, between others, the field of robotics. As model and environment complexity increase, however, training solely through the feedback of environment reward becomes more difficult. From the work on robotic priors by R.Jonschkowski et al.[1] we present robotic auxiliary losses for continuous reinforcement learning models. These function as additional feedback based on physics principles such as Newton's laws of motion, to be utilized by the reinforcement learning model during training in robotic environments. We furthermore explore the issues of concurrent optimization on several losses and present a continuous loss normalization method for the balancing of training effort between main and auxiliary losses. In all continuous robotic environments tested, individual robotic auxiliary losses show consistent improvement over the base reinforcement learning model. The joint application of all losses during training however did not always guarantee performance improvements, as the concurrent optimization of several losses of different nature proved to be difficult.

1. Introduction

In reinforcement learning[2](RL), a type of machine learning [1], the learner is not told what to do or what is right, but is instead placed in a closed loop system where it must maximize a reward function by choosing what actions to take. This is different than both supervised and unsupervised learning: the learner (agent) must gather its own data through the choices made during the learning phase. Furthermore, the feedback to the agent, in the form of positive or negative reward, is often delayed (i.e. not just the last action, but a whole sequence of actions has led the agent to the final reward).

The reinforcement learning field has made significant progress in recent years, with algorithms now capable of developing successful control policies in complex simulated environments[4, 5, 6]. Thanks to the exponential improvements in computational power, reinforcement learning models can process millions of iterations to develop a task-solving action policy without requiring any real-time human assistance or guidance.

The application of such techniques in real-world robotic tasks would allow the creation of flexible self-learning robots that require minimal human hand-engineering and that could be applied in a wide range of industrial, transportation, security and service fields[7]. Robotics application, however, are often characterized by a continuous action space: the motion of a robot is usually brought by applying varying voltage to its motors, and to do so the algorithm must be able to output continuous values to each of its motors instead of just choosing an action out of a discrete set. Specific reinforcement learning models must thus be utilized for tasks with continuous action space [8, 9, 10, 11].

Reinforcement learning models must also deal with the well-known issue of exploration vs. exploitation trade-off: when taking an action, the agent is expected to maximize the cumulative reward, by choosing an action that gave the best results in the past. To discover the best action possible, however, the agent should also try new unexplored actions, to possibly find a better solution. Because of this, training in complex environments can often be slow and unpredictable[12].

Recently, several papers introduced the concept of 'auxiliary tasks'. These tasks describe additional losses used as feedback in parallel with the standard cumulative reward objective, which may speed-up training by providing richer training gradients. The underlying idea is that the reward signal is usually sparse, which makes it an infrequent training signal. However, there is more significant information present in a typical agent-environment interaction, which may provide in itself gradients for representation learning. The inclusion of such extra loss signals can speed up the representation learning part of the task, and thereby speed-

[1]Machine learning techniques are a field of computer science that aims to have computers utilize data to learn (i.e. improve performance) and perform tasks, usually by means of statistical techniques.[3]

up the overall learning process.

Auxiliary tasks improved learning efficiency in discrete action space domains[5, 13, 14], but have not yet been utilized in continuous action space domains. Furthermore, nearly all auxiliary losses utilized in reinforcement learning strive to make no assumptions about the type of environments at hand, in order to be adaptable to all possible tasks. Not making assumptions about the environment at hand theoretically allows for reinforcement learning to be applied in every possible context, but if we do have prior knowledge about the environment available we can be more data efficient by providing directed high-level guidelines.

When applying reinforcement learning to an agent (robot) interacting with a real-life environment, basic principles of physics, such as Newtown's laws of motion, will be present independently of the type of robot utilized and task to be fulfilled, and "understanding" such principles could help the agent to generate a more consistent internal model of its environment and its dynamics.

Introducing several auxiliary losses to be utilized in parallel to the main reward feedback can however also create unbalance in the training phase, as main and auxiliary losses can differ in range and nature.

The goal of this paper is to develop auxiliary losses specifically designed for the application of reinforcement learning to robotic tasks with continuous action space. The work from R.Jonschkowski et al.[1] is adapted to transform its robotic priors (assumptions about the physical world the robot interacts in) from state representation techniques into robotic auxiliary losses for continuous action space. Furthermore, different normalization techniques are introduced, as an effort to mitigate unbalances in the model's training effort between main and auxiliary losses.

We show that each loss can have a positive effect on the training depending on the type of task at hand, improving the total reward obtained with respect to trainings without auxiliary losses. For each environment tested, at least one of the auxiliary losses proved to be of significant improvement over the base model. Efforts to find an effective normalization technique to balance the training effort between the reinforcement learning losses and auxiliary losses, as presented in chapter 6, did improve the effectiveness of auxiliary losses. However, the application of several auxiliary losses concurrently often result in over-focusing on specific losses which usually hurt the model's performance.

We firstly introduce some preliminaries in chapter 2 and related work in chapter 3. In chapter 4 four robotic auxiliary losses are presented, as well as a state prediction task to be used as comparison for the losses' evaluation. In chapter 5 the results of the application of auxiliary losses are presented. Afterwards, in chapter 6 we introduce and compare several normalization techniques for the balancing of main and auxiliary losses, and in chapter 7 the results thereof are presented. Lastly we discuss the results, present some conclusions and suggest possible directions of research in future work in chapter 9.

2. Preliminaries

In this section we provide some background regarding reinforcement learning, neural networks and deep reinforcement learning. We also present deep deterministic policy gradient (DDPG), the base deep reinforcement learning model chosen for these experiments.

2.1. Reinforcement Learning

Reinforcement learning represent in general the principle of an agent learning to fulfill a task by means of its own interaction with its environment. At the core of the human and animal intelligence resides the capacity of learning from interaction and experience, and reinforcement learning is a goal oriented computational approach to learning[2, 15].

In reinforcement learning an agent makes transitions between states $s \in I\!R^m$ by means of actions $a \in I\!R^n$ which have an impact on its environment. Such actions are taken following a policy π, which is updated by means of a reward r that can be obtained when reaching specific states (e.g. negative reward for a failure, positive reward for reaching its goal).

This approach differs from standard supervised learning techniques as the agent must learn, by itself and through trial and error, how to correctly connect input and output (state and action). Reinforcement learning's goal is to reduce human effort to the minimum of defining the rewards and the agent's architecture, and through independent training an agent can find unusual solutions and outperform human-aided control policies, especially in complex tasks.

Value-based reinforcement learning algorithms also involve the estimation of a value function, that infers how good it is for the agent to be in a certain state, or to perform an action from a given state. If the agent is following a policy π, is in state s and takes action a, the *action-value function* Q for policy π could be expected discounted sum of all future rewards that will be obtained following the policy:

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{\mathrm{k}=0}^{\infty}\gamma^{\mathrm{k}}\mathrm{r}_{\mathrm{t+k+1}}\,|\,\mathrm{s_t}=\mathrm{s},\mathrm{a_t}=\mathrm{a}\right] \qquad (1)$$

where $I\!E_\pi$ is the expectation given the policy, $\gamma \in (0,1]$ is the discount factor and r is the reward. The simplest reinforcement learning environments have small discrete state and action spaces that can be represented as an array or table. Here, the optimal solution for the task can be found through dynamic programming [16], sweeping trough each possible state and action multiple times. The algorithm can then improve the action-value function every step with the Bellman equation:

$$Q_\pi(s,a) = \sum_{\mathrm{s}'} \mathrm{P}^{\mathrm{a}}_{\mathrm{ss}'} \left[\mathrm{r}^{\mathrm{a}}_{\mathrm{ss}'} + \gamma \sum_{\mathrm{a}'} \mathrm{Q}_\pi(\mathrm{s}', \mathrm{a}') \right] \tag{2}$$

The action-value function for state s given action a is the sum of, for each possible state we get to taking action a ($\sum_{s'} P^a_{ss'}$), the reward obtained plus the discounted sum of each action-value for all possible actions a' from state s'. After training the agent can follow the optimal policy by taking the action that will lead to the state with the highest value every time.

This full sweep over each state and action is however only feasible for trivial tasks, and in environments with large and continuous state and action space the agent must instead be able to generalize from a subset of the possible states (its training iterations) to states it has never visited before.

This kind of generalization can be achieved with *function approximation*. The most studied and applied reinforcement learning function approximators are artificial neural networks[2].

2.2. Neural networks

Neural networks are a set of machine learning techniques inspired by the anatomical analysis of the neural structures of the brains. Created firstly as a mathematical model by McCulloch et al.[17], neural networks have spiked in popularity in recent years thanks to several improvements and diversification of the algorithms, as well as significant improvements in computational power and an exponential increase in available data[18].

Training of neural networks happens by means of back-propagation: in supervised learning an input x is fed through the network and the network output $\hat{y}$ is obtained:

$$\hat{y} = f_\theta(x) \tag{3}$$

Where θ represents the network weights, parameters that are changed during training to improve the network performance. The network output $\hat{y}$ is then compared to the correct output y, and its difference is used as loss function for the correction of the weights of the last layer. Such correction is then back-propagated to the previous layer, which is then also corrected, and so on until all network layers have updated their neuron's weights. When presenting the same input x to the network, the output will now better match the correct output y, and performing this improvement several times with different inputs (training) will iteratively adjust the weights of network to be able to generally make correct predictions on input data similar to that used during training.

In reinforcement learning, however, there is no known correct output y. Instead, the model makes use of the reward as feedback for back-propagation and iterative improvement.

2.3. Deep reinforcement learning

Deep reinforcement learning models are characterized by a several stacked layers of neurons, that in recent years have shown generalization capabilities which allow the agent to learn policies from high dimensionality states such as camera input or a high DoF robotic arm's joint states.

The first deep reinforcement learning model able to learn an end-to-end control policy directly from visual input data was the DQN model from V.Mnih et al.[4]. Inspired by the success of deep neural networks in previous years, the authors attempted to connect these deep models to reinforcement learning. Their algorithm, *deep Q-learning* (DQN), made use of convolutional layers to directly map a down-sampled visual input (84x84 pixels) to a discrete set of possible actions to be taken. One crucial improvement for the algorithm was the use of an experience replay mechanism, which randomly samples its training data from all previous transitions thus smoothing the training distribution. Without such a mechanism the model would train from subsequent, highly correlated steps, which creates training instability.

2.4. Deep deterministic policy gradient

Many control tasks, especially in the robotics field, have continuous and highly-dimensional action spaces: the agent's policy is comprised of a real value for each of its actuators, instead of being a discrete choice between possible actions to take. Discrete reinforcement learning algorithms cannot be easily implemented for continuous control tasks, and the discretization of a continuous action space is usually not feasible for domains with more than a couple of degrees of freedom.

Silver et al.[10] approach this problem with a *deterministic policy gradient algorithm*, an off-policy actor-critic algorithm. Lillicrap et al.[11], with their deep deterministic policy gradient method (DDPG), construct on their work by also including the improvements from V.Minh et al.[4] in the algorithm: the network is trained off-policy from a replay buffer to decorrelate training samples, and making

[2]Sutton and Barto[2] illustrate a general application of function approximators in reinforcement learning in Chapter 9.

use of a slowly updating target Q-network.

Actor-critic models are comprised of an actor that determines the action to be taken, and a critic that evaluates the resulting state-action value Q. Given an observation, the model determines both the action to be taken (actor) and the value of taking that action in that state (critic). As a reward is then obtained, a better estimate of the state-action value Q can be made, and its difference with the previous guess is used as critic loss.

A key factor for the stability of training is the usage of *target* actor and critic networks. These are copies of the actor and critic networks that are utilized to make the target Q by which to improve the network:

$$Q_{target,t} = r_t + \gamma \cdot Q_{targetcritic}(s_{t+1}, a_{targetactor,t+1}) \tag{4}$$

The target actor and critic parameters are not improved directly through back-propagation as the original actor and target networks, but are instead slowly updated to copy their values after each training iteration:

$$\theta_{target} = (1 - \tau) \cdot \theta_{target} + \tau \cdot \theta_{original} \tag{5}$$

where $tau \in (0, 1]$ is a hyperparameter that determines how quickly the target networks copy their parameters from the original actor and critic networks. Doing so brings significant stability to the training, as it decorrelates target and prediction Q values.

DDPG utilizes off-policy training, meaning that the training phase is decoupled from the interactions with the environment. A training run is performed in cycles of two steps: firstly the model is utilized to perform actions given observations, which are stored in the replay buffer. Secondly, a batch of random uncorrelated transitions from the replay buffer is used to calculate the actor and critic loss (and eventually also the auxiliary losses for both actor and critic). The gradient of the loss is then calculated, and back-propagated to improve the model's parameters.

To improve exploration during training the action is taken from a copy of the actor network with added parameter noise: each neuron in the model receives a small stochastic noise, thus changing the final action obtained. This has been proven to be more effective than adding stochastic noise directly to the policy determined by the model[19].

3. Related work

Here we present the main works that inspired this experiment. We present state representation works and robotic priors, which aid the reinforcement learning through a pre-training phase, and auxiliary tasks, additional losses that aim to improve the learning process during the reinforcement learning training.

3.1. State representation and Robotic priors

Most research in the field of reinforcement learning for robotics is still bound to virtual agents in simulated physics environments. This is because of the prohibitive amount of interactions of the agent in the environment that are required for the development of complex control policies. Were, for example, a real-life robot to perform actions in an environment for a million iterations at a frequency of 5Hz (a considerable speed), it would require a continuous training time of $\approx 56h$. It is thus necessary to speed up and improve the reinforcement learning process, an effort undertaken in several different ways, of which we discuss some below.

Reinforcement learning from an high-dimensional state to action requires an often prohibitive amount of iterations to converge because of the high number of internal parameters to train: the agent must be able to both gather task-relevant features from the sensory observation s, and to infer from these task-relevant features to the action a to take.

To reduce this need for long training times, it would be convenient to first learn to convert this sensory observation s to a pertinent internal state space z of lower dimensionality[3] prior to utilizing such internal state z as input for the reinforcement learning process.

While such mapping from visual input to internal state can be developed as a separate hand-engineered step, doing so would make the performance of the model highly dependent on the intuition and capability of the engineer that designs the feature extraction from observation to internal state, and strongly hinders the flexibility of application of these algorithms for different tasks and environments. State representation learning methods are instead machine learning methods that try to map high-dimensional input states into a compact and pertinent internal state space, with minimal or no human feature engineering involved.

Some representation learning methods only make use of the input observation itself, or observations at previous steps, to maintain their application flexibility. Deep AutoEncoders[20, 21, 22] are an unsupervised deep neural network architecture used for the encoding of high-dimensional data. An AutoEncoder is usually composed of several layers of decreasing size (encoder) followed by as many layers of increasing size (decoder) to finally reach an output layer of the same size as the input, creating an "hourglass" shape (see Figure 1). The network is usually trained to reconstruct in the last layer an output to be equal to that of input layer, while having to pass through the bot-

[3]E.G. in navigation tasks this internal state z ideally represents the location of the agent in a 2D mapping of the environment, as such is the most compact and effective input state for navigational purposes.

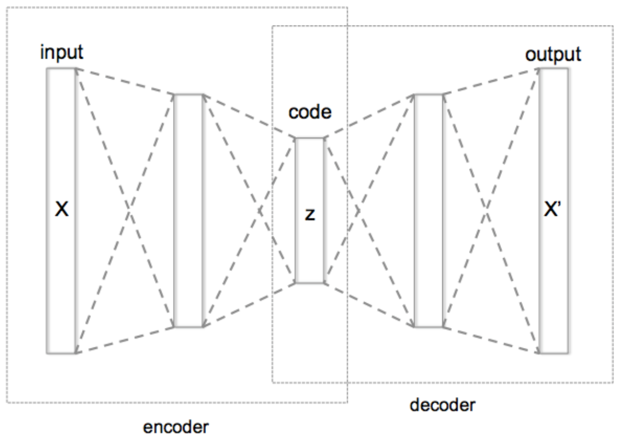

Figure 1. Schematic of an AutoEncoder network[20]. The input x on the right in RL is usually the state s_t, which gets transformed by the encoder into a smaller representation z_t, and then decoded into x', which can be either equal to the input or the next state s_{t+1}.

tleneck of the center layer of reduced dimensionality. This bottleneck will afterwards be used as low-dimensional internal state z, the effective input state for the reinforcement learning model.

A similar approach, more effective for reinforcement learning models, is to predict the future observation s_{t+1} from the input observation s and the action a taken at that state[23, 13, 24]. Doing so forces the state representation network to "understand" the dynamics of the environment, in order to be able to predict the future state, and so doing helps it encode action and task-relevant features in its parameters.

R.Jonschkowski et al. expand the principle of state representation in robotic reinforcement learning with the introduction of robotic priors[1]. By making some assumptions (priors) about the environment that the agent (robot) acts in, a state representation can be learned that is consistent with physics. The obtained state representation is especially effective for robotic tasks, and its application facilitates generalization in reinforcement learning.

The priors presented are inferred from Newton's three laws of motion[25], and in the paper the authors show that it is sufficient for these priors to be generally true and the resulting model will be robust to states that are inconsistent with the some of the priors.

In their work, R.Jonschkowski et al. apply their robotic priors to several navigation and control tasks with discrete action space and a camera input as state, showing their effectiveness as state representation methods.

3.2. End to end learning

Standard deep reinforcement learning models make use of the reward as only descriptor of the performance of the model to map the high dimensional input space to the action to be taken for such state. Furthermore the exploration of the environment happens often slowly, generating highly correlated (and poorly defining) training data. The utilization of state representation learning, on the other hand, makes the effective reinforcement learning training phase shorter, but it requires states' data itself beforehand, oftentimes defeating its purpose in real-life robotic applications, and introduces a two-phase training system that makes training more difficult and more time consuming for the engineer.

We define end-to-end systems as reinforcement learning models where other added losses improve the model's performance in concurrency with the reinforcement learning training, without requiring a two-phase training method.

In their recent work M.Jaderberg et al.[5] approach the issue of inefficient data utilization by means of Auxiliary Tasks: instead of focusing on attempting to map observations to task-relevant features in a separated state representation learning phase, they present several different Auxiliary Tasks whose losses continuously improve the model even in reward-less state transitions. The model has a core structure of a convolutional layer followed by an Long Short-Term Memory (LSTM) layer[4]. This core structure outputs directly the value and policy terms used by their Actor-Critic algorithm[27], but is also used as core part of the auxiliary tasks, and can thus be seen as the state representation structure that would be trained in a state representation phase.

In a similar work, E.Shelhamer et al.[13] explore the utilization of other self-supervised losses to improve training time and accuracy for Deep Reinforcement Learning. Starting from an asynchronous advantage actor-critic architecture[6], E.Shelhamer et al. explore the addition of several auxiliary losses that make use of observation, action and successors. In the paper the auxiliary losses are tested both in concurrency with the reinforcement learning and in state representation-like pre-training fashion, and the authors show that concurrent utilization is easier to implement and gives more effective results, usually reaching the performance of the base algorithm in just one-third of the iterations.

The application of auxiliary tasks in reinforcement learning models has been explored by all aforementioned authors with the objective of finding universal auxiliary tasks the can be applied in any reinforcement learning setting,

[4]LSTM layers are a variant of the standard Recurrent layer that has more long-term memory retention capabilities.[26]

and their results shown that the addition of extra feedback for the network can improve training duration and network performance.

When applying reinforcement learning in robotic scenarios in real life, however, the laws of physics dictate in great part the nature of interactions of the agent with the environment, and capturing these characteristics into auxiliary feedback to guide the network's training could provide richer and more effective feedback and thus be more data-efficient.

4. Robotic Auxiliary Losses

In this chapter the main innovation of this paper is presented: robotic auxiliary losses.

4.1. Loss functions

In this section the state representation learning work of R.Jonschkowski et al.[1] is re-adapted: The presented robotic priors are transformed to be utilized as auxiliary losses in an end-to-end deep reinforcement learning training without requiring a state representation pre-training. Furthermore, their application is specifically designed for reinforcement learning models with *continuous* action space, which is usually necessary for implementation in robotic tasks.

The aim of these losses is to provide the reinforcement learning agent with extra feedback from which it can more easily and effectively build an internal model of the physical world in which it acts.

The simplicity prior from R.Jonschkowski's work was not applied, as it would require creating a bottleneck in the network: while this is necessary in state representation to reduce the state dimensionality and accelerate the subsequent reinforcement learning training phase, in end-to-end training the application of auxiliary losses is parallel to the reinforcement learning training and no significant training speed would be obtained. A prediction task is also implemented, which does not involve robotic priors and can be used comparatively to assess the robotic auxiliary losses' effectiveness.

Temporal coherence loss *Task-relevant properties of the world change gradually over time.*
This loss enforces a continuity in the agent's internal state space. Given two successive internal states:

$$\begin{aligned} z_{\theta,t} &= g_\theta(s_t) \\ z_{\theta,t+1} &= g_\theta(s_{t+1}) \end{aligned} \tag{6}$$

where g_θ are the state representation layers (see Figure 2), we can infer that they should differ in magnitude similar to that of the action taken $||a_t||_2^2$. The auxiliary temporal coherence loss l_{tc} is composed of two terms, one that punishes the smallness in state changes $||\Delta z||_2^2$ for an action of high magnitude, and the second that punishes high magnitude state changes given an action of little magnitude:

$$\begin{aligned} \mathcal{L}_{tc,1} &= e^{-\lambda||\Delta z_\theta||_2^2}\,||a||_2^2 \\ \mathcal{L}_{tc,2} &= e^{-\lambda||a||_2^2}\,||\Delta z_\theta||_2^2 \\ \mathcal{L}_{tc} &= \mathcal{L}_{tc,1} + \mathcal{L}_{tc,2} \end{aligned} \tag{7}$$

The exponential of the negative magnitude $e^{-\lambda||x||_2^2}$ represents a similarity function: for $\lim_{x\to 0} e^{-\lambda||x||_2^2} = 1$ and for values of $x > 0$ the similarity value quickly decreases to 0. The λ coefficient determines the rate of descent of the similarity value, and is set to $\lambda = 10$ for this research.
This loss should enforce a normalization of the internal state placement, ensuring that changes in internal state brought by actions have magnitude relative to the magnitude of the actions taken.

Proportionality loss *The amount of change in the state caused by an action should be proportional to the magnitude of the action taken.*
When comparing two unrelated states, $z_{\theta,j}$ and $z_{\theta,k}$, if the actions taken in both states have similar magnitude then the state change should also be similar. Note that this loss is not about similarity between internal states, but rather about similarity in amount of change caused by actions of set magnitude. The auxiliary proportionality loss is constructed as follows:

$$\begin{aligned} f_{\Delta z_{magdiff}} &= (||\Delta z_{\theta,j}||_2^2 - ||\Delta z_{\theta,k}||_2^2)^2 \\ f_{a_{sim}} &= e^{-\lambda||a_j - a_k||_2^2} \\ \mathcal{L}_{prop} &= f_{\Delta z_{magdiff}}\, f_{a_{sim}} \end{aligned} \tag{8}$$

The proportionality loss ensures that the magnitude of change brought by an action is consistent across all states. R.Jonschkowski et al. relate this prior to Newton's second law of motion: for every force (action) there is a resulting proportional acceleration (change in state).

Causality loss *The state and action together should determine the reward obtained.*
This loss enforces the state mapping to be task-relevant: if, suppose, two observations are mapped to similar states and the same action is taken from said states, then the reward obtained from both states should be similar. If this is not the case the mapping from observation to state failed to encode some task-relevant feature.
Thus given two unrelated states z_j and z_k, their state similarity is punished relative to the similarity of the actions

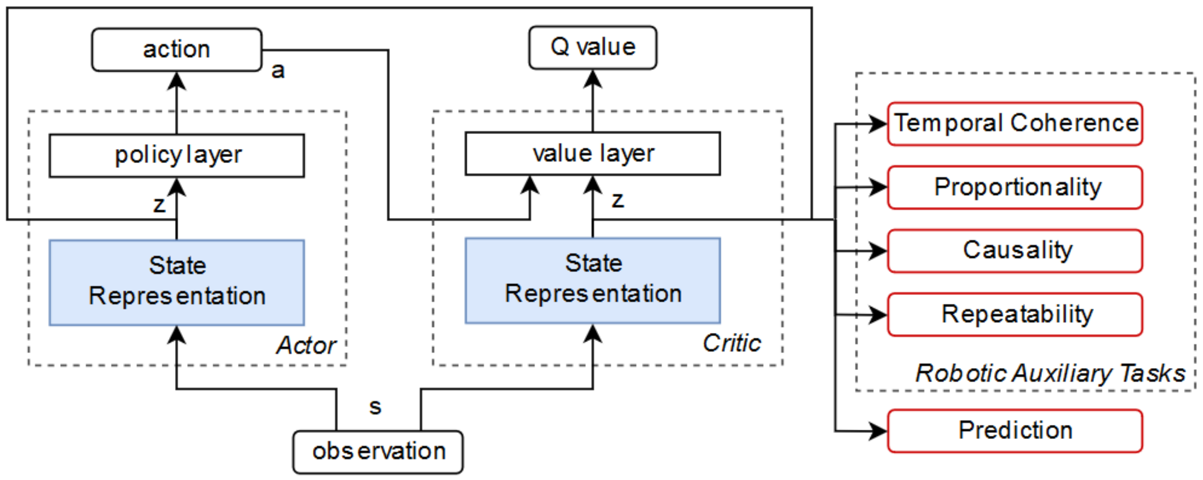

Figure 2. Overview of the DDPG model with Robotic auxiliary losses (and prediction task). The internal state z obtained by the state representation, both for the actor and critic, is also utilized as input for the auxiliary losses. The auxiliary losses' dependencies on action and reward are not depicted for clarity.

taken and magnitude of the difference in obtained reward:

$$\begin{aligned} f_{z_{sim}} &= e^{-\lambda||z_{\theta,j}-z_{\theta,k}||_2^2} \\ f_{a_{sim}} &= e^{-\lambda||a_j-a_k||_2^2} \\ f_{r_{diff}} &= ||r_j - r_k||_2^2 \\ \mathcal{L}_{caus} &= f_{z_{sim}}\, f_{a_{sim}}\, f_{r_{diff}} \end{aligned} \tag{9}$$

The causality loss is especially effective in environments with descriptive rewards, but is mostly irrelevant in case of sparse rewards, as most of the states compared will have the same null reward, bringing the causality loss to 0.

Repeatability loss *The state and action together determine the resulting change in these properties.*
Similarly to the Causality loss, this loss enforces the change in state caused by action to be consistent for similar states: if two unrelated states z_j and z_k are similar and similar action is taken for both of them then the resulting change in state should also be similar, both in magnitude and in direction:

$$\begin{aligned} f_{z_{sim}} &= e^{-\lambda||z_{\theta,j}-z_{\theta,k}||_2^2} \\ f_{a_{sim}} &= e^{-\lambda||a_j-a_k||_2^2} \\ f_{\Delta z_{sim}} &= e^{-\lambda||\Delta z_{\theta,j}-\Delta z_{\theta,k}||_2^2} \\ \mathcal{L}_{repeat} &= f_{z_{sim}}\, f_{a_{sim}}\, f_{\Delta z_{sim}} \end{aligned} \tag{10}$$

This loss, together with the causality loss, enforces causal determinism: if similar actions are taken in what the agent considers similar states, they should lead to similar changes. R.Jonschkowski et al. write that *repeatability prior and the causality prior together constitute the Markov property of states.*

State prediction task The state prediction task also has as focus the encoding of environment dynamics in the state representation layers of the network: given the internal state $z_{\theta,t}$ from an observation s_t and the action taken in that time step, the network should be able to predict the observation it will receive in the following step:

$$\begin{aligned} \hat{s}_{\theta,\phi,t+1} &= h_\phi(z_{\theta,t}, a_t) \\ \mathcal{L}_{pred} &= ||s_{t+1} - \hat{s}_{\theta,\phi,t+1}||_2^2 \end{aligned} \tag{11}$$

To do so, the action is concatenated to the internal state, and two dense layers are attached that "mirror" the state representation sequence, with independent weights ϕ. The last layer's output matches the observation size, and the task loss $\mathcal{L}_{pred}$ becomes the cross-entropy of the difference between predicted and actual observation at the next time step. This auxiliary task differs from the robotic losses in that there are added model parameters to be trained ϕ, but its validity is maintained as a good state representation z_θ is necessary to be able to correctly predict the future observation.

4.2. Experiments

The reinforcement learning platform used for this experiment is the OpenAI Gym[28][5]. The physics engine used

[5]https://gym.openai.com/

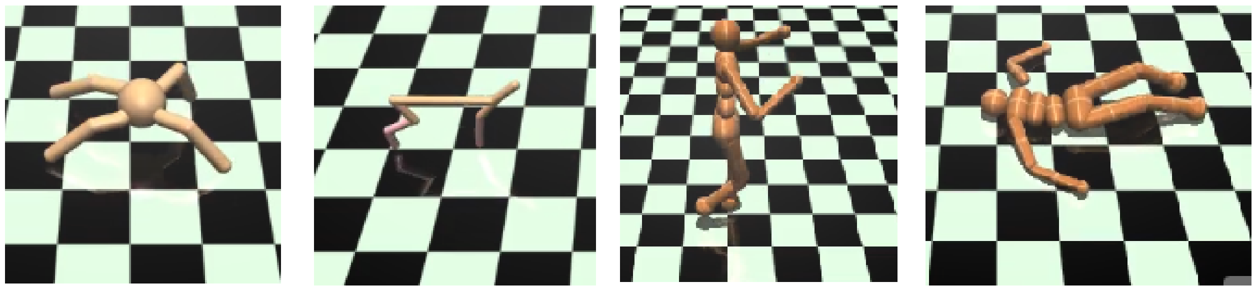

Figure 3. Overview of the four openAI enviroments in which the robotic auxiliary losses will be evaluated. From left to right: *Ant-V2*, *Cheetah-v2*, *Humanoid-v2* and *HumanoidStandUp-v2*. For all environments, the input state is a vector of each joint's angular position and velocity and contact forces, and the action space is a vector of real values of activation of each joint.

is the MuJoCo engine, which is specifically designed for model-based control[29]. Four robotic control environments with continuous action space from the Openai Gym library, which make use of the MuJoCo engine, have been utilized for these experiments: *Ant-v2*, *HalfCheetah-v2*, *Humanoid-v2* and *HumanoidStandingUp-v2*. A rendering of the environments is visualized in Figure 3. For all these tasks the observation consists of a vector with angular position and velocity of each joint of the agent, and the action is a vector comprising the activation force for each joint's motor.

A conceptual visualization of the neural network architecture with auxiliary losses can be seen in Figure 2. We define the first two fully connected layers as "state representation" layers, as those layers' parameters will be shared with the auxiliary losses training in our implementation: during training, their parameters will be updated firstly through the back-propagation of their respective (actor or critic) loss, and secondly through the back-propagation of the auxiliary losses applied.

As the observations received by the environment are vectors containing (normalized) real-valued numbers, both the state representation layers and policy and value layers are fully connected layers[30]. In case of visual observation input convolutional layers should be used instead for the state representation layers, whose output is then flattened to form the internal state vector z.

The auxiliary losses' gradients back-propagate through the state representation part of the network only (i.e. the action taken, although being obtained from the actor network, is used as constant and has no gradient pass through it). DDPG being an off-policy reinforcement learning algorithm is especially apt to auxiliary losses: a training batch is comprised of an uncorrelated set of state transitions from the *experience replay,Atari*, which can also be directly used as input to calculate the losses of the auxiliary losses. The experience selection algorithm is modified to ensure that when comparing transitions for the robotic auxiliary losses such transitions are always uncorrelated[6].

For all tests, 3 trainings are run with different seeds for model parameters initialization and batch selection from the experience replay. The continuous loss normalization technique is implemented in all tests (see chapter 6 for further details and implementation hereof).

The aim of the of the model evaluation if to answer the following questions:

1. Should auxiliary losses be applied to the actor, critic or both networks concurrently?

2. What robotic auxiliary losses improve training speed or performance? Is the performance increase environment-dependent?

3. How is training affected by the concurrent addition of all robotic auxiliary losses?

5. Results of auxiliary losses

Here we present the results of the application of auxiliary losses on the selected environments.

5.1. Actor, critic, or joint application

In a model with individual actor and critic networks we can apply auxiliary losses for the improvement of the internal state representation to either actor or critic, or jointly to both. A visualization of the individual and joint application of auxiliary losses can be seen in Figure 4. As seen in the Figure, the application of the auxiliary loss to either actor or critic improves performance over the base model.

[6]This was done in practice by selecting for each transition a second transition randomly picked between 100 and 200 steps further in the experience pool.

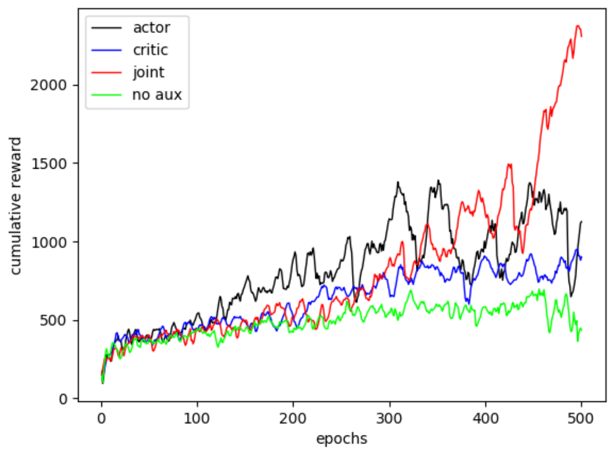

Figure 4. Results of run on Humanoid-v2 environment with causality loss applied to either actor, critic, or simultaneously to both. A run with the same seed but no auxiliary loss has been included for comparison.

However, the performance improves most when the auxiliary losses are applied to actor and critic jointly. We infer from the results that the networks perform best if their parameters are guided by the auxiliary losses in the same ”direction”, while instead resulting in unbalance between actor and critic if the auxiliary losses are not applied to either network.

The joint application of auxiliary losses to actor and critic also mimic the implementation from M.Jaderberg et al.[5] and E.Shelhamer et al.[13], whose asyncronous advantage actor-critic model has a single network which outputs both an action vector (actor) and a state value (critic).

Based on these results, all auxiliary losses are applied to both actor and critic networks for the rest of the experiments.

5.2. Individual losses

We investigate the performance impact of each individual robotic auxiliary loss and compare it to the base model without auxiliary losses and to the model with the added auxiliary task of state prediction. The results for these experiments is shown in Figure 5. What is firstly evident from the figure is that auxiliary losses have different performance impacts for different environments: The temporal coherence task, while not effective for *Ant* and *Humanoid*, improve performance in *HalfCheetah* and *HumanoidStandUp*; proportionality task addition is significantly improving the performance of the model in *Ant*, while not being effective in *HumanoidStandUp* and hurting performance in *HalfCheetah* and *Humanoid*; The causality task addition hurts performance in the *Ant* environment, but is very effective in *HalfCheetah*, *Humanoid* and *HumanoidStandUp*; and finally repeatability is not very effective in *Ant* and *HumanoidStandUp*, significantly hurts performance in *HalfCheetah*, but improves performance in *Humanoid*.

Each auxiliary loss can improve the model’s performance in at least one of the environments tested, but worsens performance in other environments, thus there is no auxiliary loss which guarantees an improvement across all environments.

5.3. Joint auxiliary losses implementation

Next we observe the results of applying all robotic auxiliary losses concurrently. In Figure 6 we can see that the application of all auxiliary losses concurrently impedes some of the losses from being optimized, with their loss value remaining constant or even increasing during the training, while other losses’ values quickly decrease as the networks modify their parameters.

Which losses are or are not being optimized changes in our tests depending on the initialization of the parameters: utilizing a different seed usually results in different auxiliary tasks being optimized or ignored, as shown in Figure 7. In the Figure, a training is run with the same parameters and all robotic auxiliary losses concurrently, only changing the seed of the network parameter initialization and experience replay batch selection. As shown, in the training with seed 1 the causality loss is being optimized and proportionality loss is ignored, and the opposite happens in seed 2. Because of this, there is a significant difference in model performance, as the optimization of causality loss performs much better than proportionality loss on the selected environment (Humanoid-v2).

A specific contrast between auxiliary losses is not present either, as depending on seed all combinations of two singular losses can be optimized during training. Note that this inability to optimize all auxiliary losses together is not related to the continuous normalization method that is applied in the results shown, as these results are seen also when applying the other kinds of loss normalization discussed below.

6. Loss normalization

When introducing several auxiliary losses to a reinforcement learning algorithm, the goal is to improve training by giving extra guidelines to the network. However, when naively summing the auxiliary losses to the main reinforcement learning loss the relative value of such losses cause the training to often either ignore or over-focus on specific losses. The first case will result in some auxiliary losses

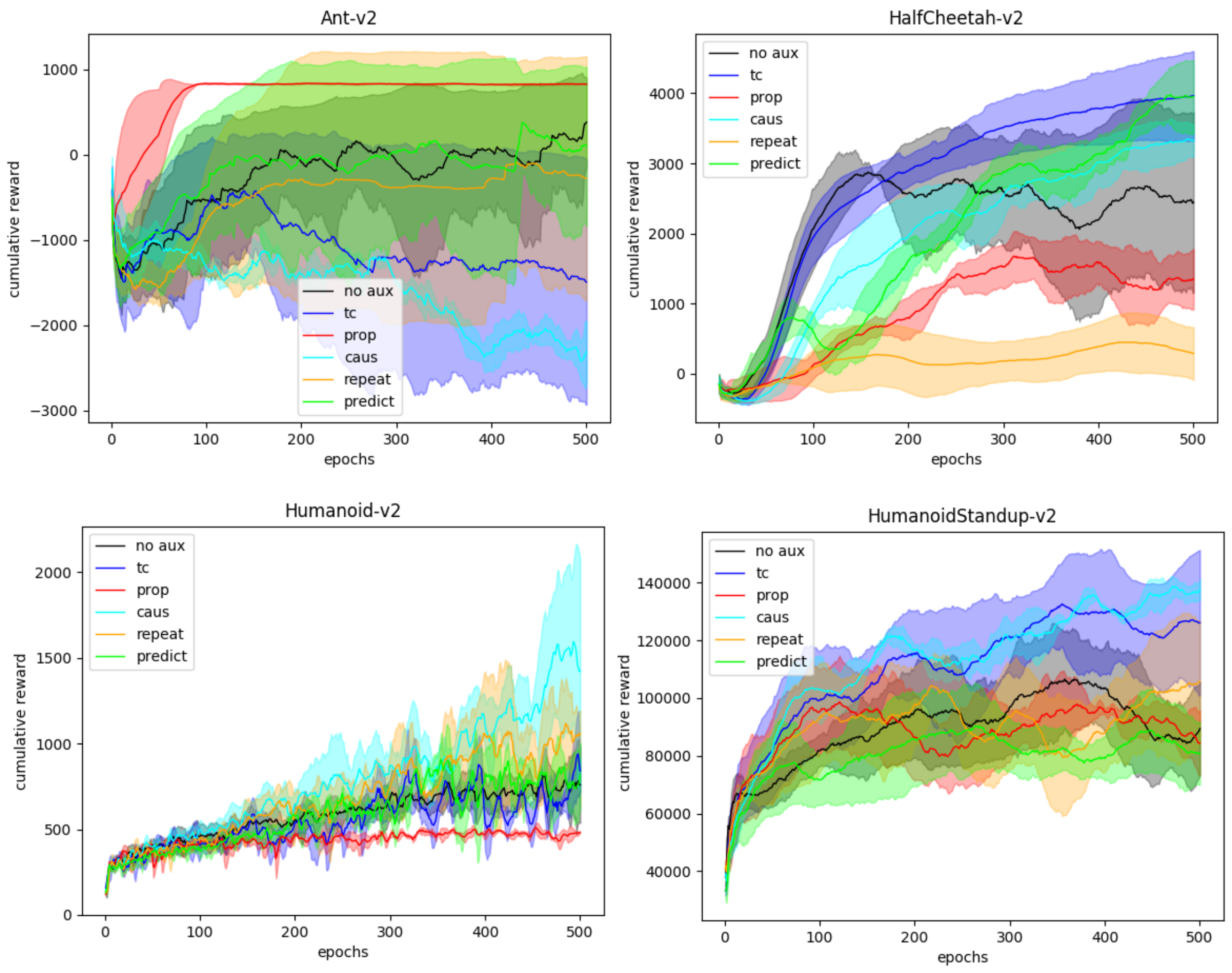

Figure 5. Individual auxiliary losses performance per environment. *no aux*: Base DDPG model. *tc*: with temporal coherence. *prop*: with proportionality. *caus*: with causality. *repeat*: with repeatability. *predict*: with state prediction. (best viewed in color). The plot shows the cumulative reward of the agent per epoch (higher is better). Each evaluation is shown with mean (solid line) and standard deviation (light area of same color).

bringing no benefit to the training, the second case will often cause training failure as the network loses focus on its main reinforcement learning objective.

Manually multiplying each loss by a constant to bring their values to the same range is tedious and ineffective, as their relative range can vary wildly, and would need to be adjusted per individual environment and loss specification, adding several hyper-parameters to an already saturated tuning list, and making thus their application defeat reinforcement learning's principle of self-supervised training.

Furthermore, the main reinforcement learning losses for the actor and critic differ in nature from the auxiliary losses we present in this paper: the auxiliary losses presented are applied in a supervised training fashion, with gradually decreasing losses as the model gets better at respecting the auxiliary loss' constraint. On the other hand, the closed-loop characteristics of reinforcement learning cause the agent to reach new, better states as it improves, and the losses (especially those of the critic) are often increasing instead of decreasing as newly found states must be evaluated. Because of this, the auxiliary losses' gradients decrease during training while the main losses' gradients remain in the same range as at the start or even grow significantly, resulting in the auxiliary losses' gradients to be ignored.

These issues make the parallel implementation of different losses difficult and give unexpected results. We explore below different techniques for the calibration of losses with the focus of having each loss be of continuous relevance during training, without ”overpowering” the state repre-

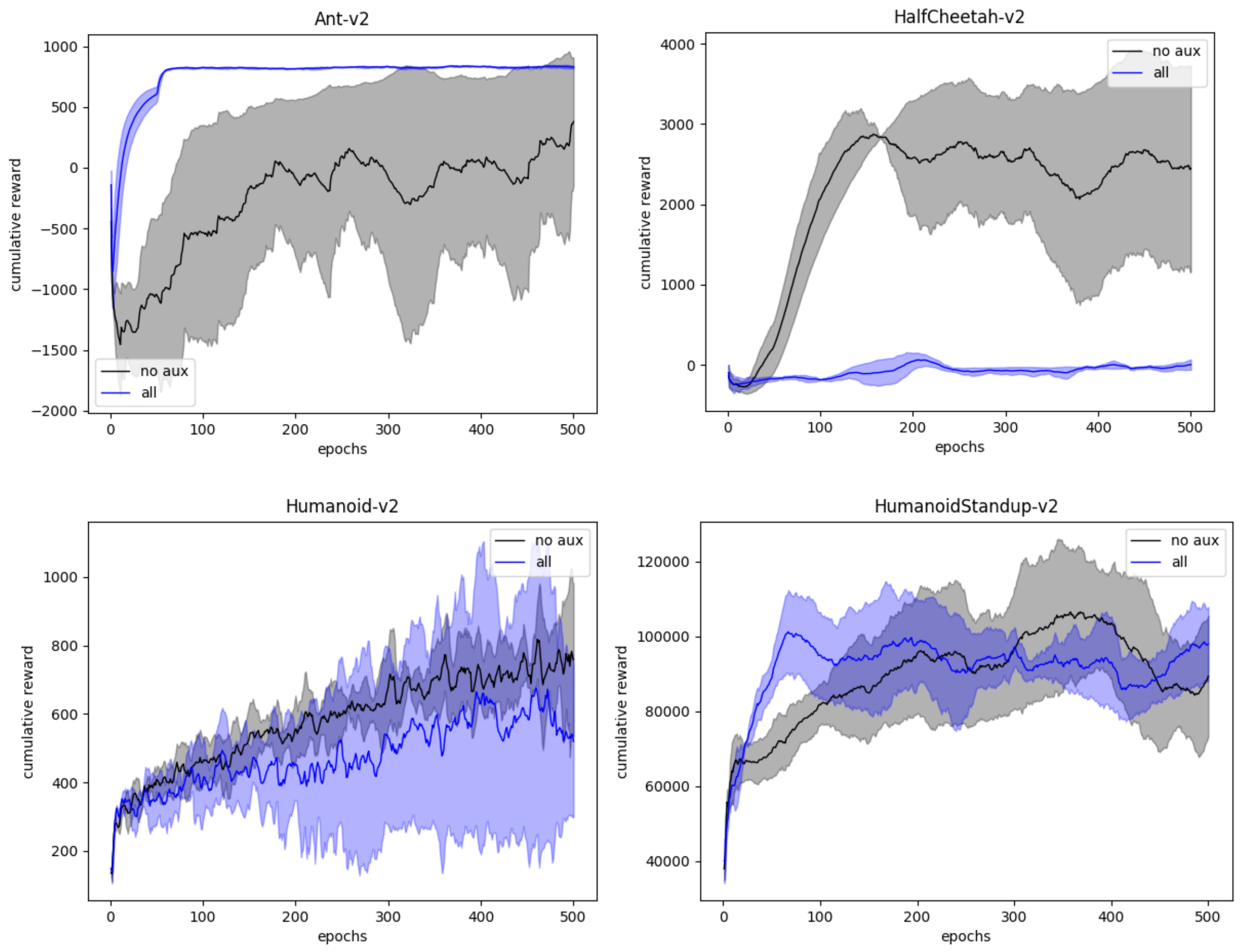

Figure 6. Joint robotic auxiliary losses performance per environment. *no aux*: Base DDPG model. *all*: with all 4 robotic auxiliary losses. (best viewed in color). The plot shows the cumulative reward of the agent per epoch (higher is better). Each evaluation is shown with mean (solid line) and standard deviation (light area of same color).

sentation layers and negatively affecting the reinforcement learning process.

6.1. Gradient clipping

A well known and often applied normalization technique is that of gradient clipping. This technique aims to hinder overshooting and exploding gradients by re-scaling them when their $l2$ norm exceeds a given threshold:

$$\delta\theta_{clip} = \frac{\delta\theta \cdot \lambda}{||\delta\theta||^2} \tag{12}$$

We can apply this technique with a low threshold value (e.g. unit value) to use it as a strong regularizer. As the gradient norm is usually higher than the chosen threshold, all gradients get re-scaled and should thus affect training in similar matter. Note however that as the auxiliary losses are added togheter before the gradient calculation this does not affect individual losses.

6.2. Initial loss normalization

A simple normalization method would be to estimate the values of each loss at the beginning of the training, and set the inverse of it as a constant by which each loss is multiplied for the rest of the training:

$$\begin{aligned} \Lambda_{init} &= 1/\,\mathcal{L}_0 \\ \mathcal{L}_{t,norm} &= \mathcal{L} \cdot \Lambda_{init} \end{aligned} \tag{13}$$

Doing so would mean that each loss begins at the same unit value, but is free to diverge during training.

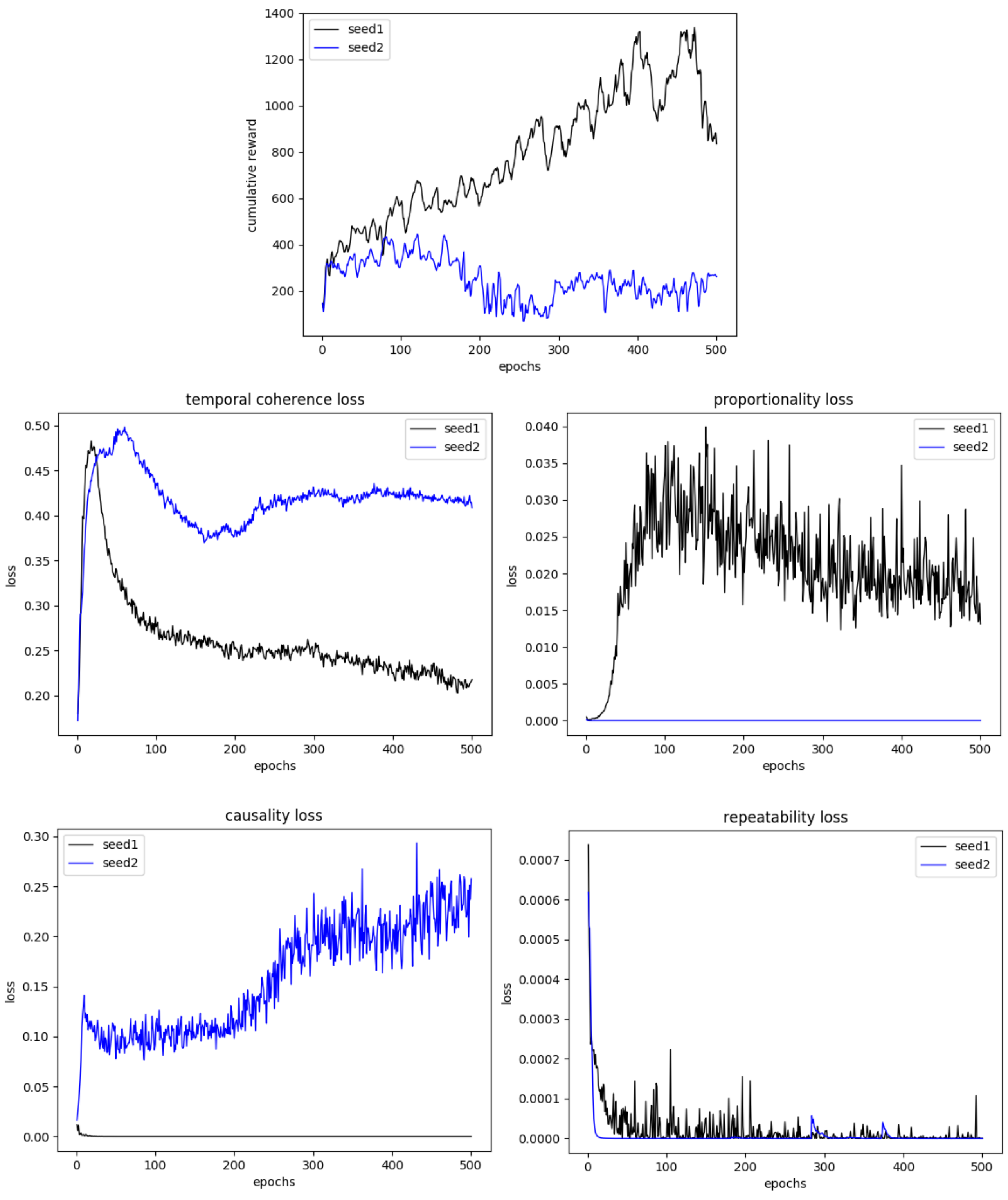

Figure 7. Joint robotic auxiliary losses comparison in Humanoid-v2 for two different seeds (best viewed in color). The top plot shows the cumulative reward of the agent per epoch (higher is better). The four plots below it show the loss value during training for the four different robotic auxiliary losses.

6.3. Continuous loss normalization

An alternative to the above solution is to instead force all losses to be of unit value during the whole training. In order to do this, we divide each loss by its absolute value:

$$\mathcal{L}_{norm} = \frac{\mathcal{L}}{[\![\, |\,\mathcal{L}\,|\,]\!] + \epsilon} \tag{14}$$

The absolute value of the loss is applied in the function as a constant value, explicitly blocking the back-propagation of the gradient. This is done with the stop-gradient identity function $[\![\, x \,]\!] = x$ (as used in the work by Z. Xu et al.[31]). Were this not done, the lower and upper terms of the fraction would match rendering the gradient null at all times. Note that a small value ϵ is added to the denominator in order to avoid division by zero for null losses.

By applying the continuous loss normalization to all auxiliary losses, as well as the actor and critic losses from the reinforcement learning algorithm, it is also possible to set a fixed ratio of loss between main and auxiliary losses. If taking unit normalized losses for actor and critic losses, for example, we can set the sum of auxiliary losses to be half of that:

$$\mathcal{L}_{aux} = \frac{\sum_{k=1}^{n} \mathcal{L}_{k,norm}}{2\,k} \tag{15}$$

The goal of this loss normalization technique is to force all auxiliary losses to remain relevant during training, by matching their values to the main actor and critic loss throughout the training.

7. Results of loss normalization

We evaluate the different loss normalization techniques by comparing the gradients of each technique for actor, critic and auxiliary losses , and by observing the change in performance for each technique when applying an auxiliary loss (see Figure 8).

From the figures we can see that of all techniques proposed, only the continuous loss normalization technique application results in a change of performance when applying auxiliary losses to the model.

The utilization of gradient clipping has as main consequence the re-scaling of the critic gradient to be in the same range as that of the actor and auxiliary losses. However, the total auxiliary loss' gradient quickly descends and because of this the final performance of the model remains unaffected by auxiliary losses (see $clip_norm$'s cumulative reward compared to that of no_norm in Figure 8).

The initial loss normalization technique, on the other hand, results in the auxiliary loss' gradients to be significantly higher at the start, in a range similar to that of the critic network. Nonetheless, the performance of the model is unaffected by the addition of whichever auxiliary loss.

The closed-loop nature of reinforcement learning makes the actor and critic losses to be significantly less prone to decreasing in value during training, as opposed to the auxiliary losses, which can be more quickly optimized and become ineffective during training. With continuous loss normalization, however, each auxiliary loss is re-scaled to remain in the same range, thus continuing to provide improvements across the training.

Based on these results the continuous loss normalization technique is applied throughout the experiment.

8. Discussion

The application of individual auxiliary losses and their improvement on the model's internal state representation affect performance mostly further in the run. This results contrast with those by E.Shelhamer et al.[13], where the auxiliary losses' application accelerates the training but does not significantly improve the maximum total reward at the end of the training phase. This suggests that the application of the right auxiliary losses can help the network find better policies by forming a richer and more accurate internal state representation.

Comparatively, it is clear that the prediction task is not able to significantly impact the training performance. This is most likely caused by the nature of the input state: input state prediction has been mostly applied (with success) to visual input, which is of larger size and whose information density is significantly lower. This is also true of the robotic priors application by R.Jonschkowski et al.[1], which was tested solely on visual input.

If the agent is trained on all robotic auxiliary losses concurrently, its performance change is dictated by the specific losses that are being optimized with the given seed. A possible cause of the network's inability to "focus" on all auxiliary tasks jointly is the layers' width of 64 neurons, which may create a bottleneck that hinders the state representation capabilities of the network (comparatively, work from M.Jaderberg et al.[5] and E.Shelhamer et al.[13] make use of layers with width of 256). Further work could explore the balance between layers' width and auxiliary losses optimization capabilities, and their effect on the reinforcement learning performance.

The joint application of several auxiliary losses has proven to be all but trivial: invariably, some auxiliary losses will dominate the training phase, eclipsing other losses and making their application unreliable. Efforts towards a normalization technique that ensures the effective and balanced application of auxiliary losses has shown promising

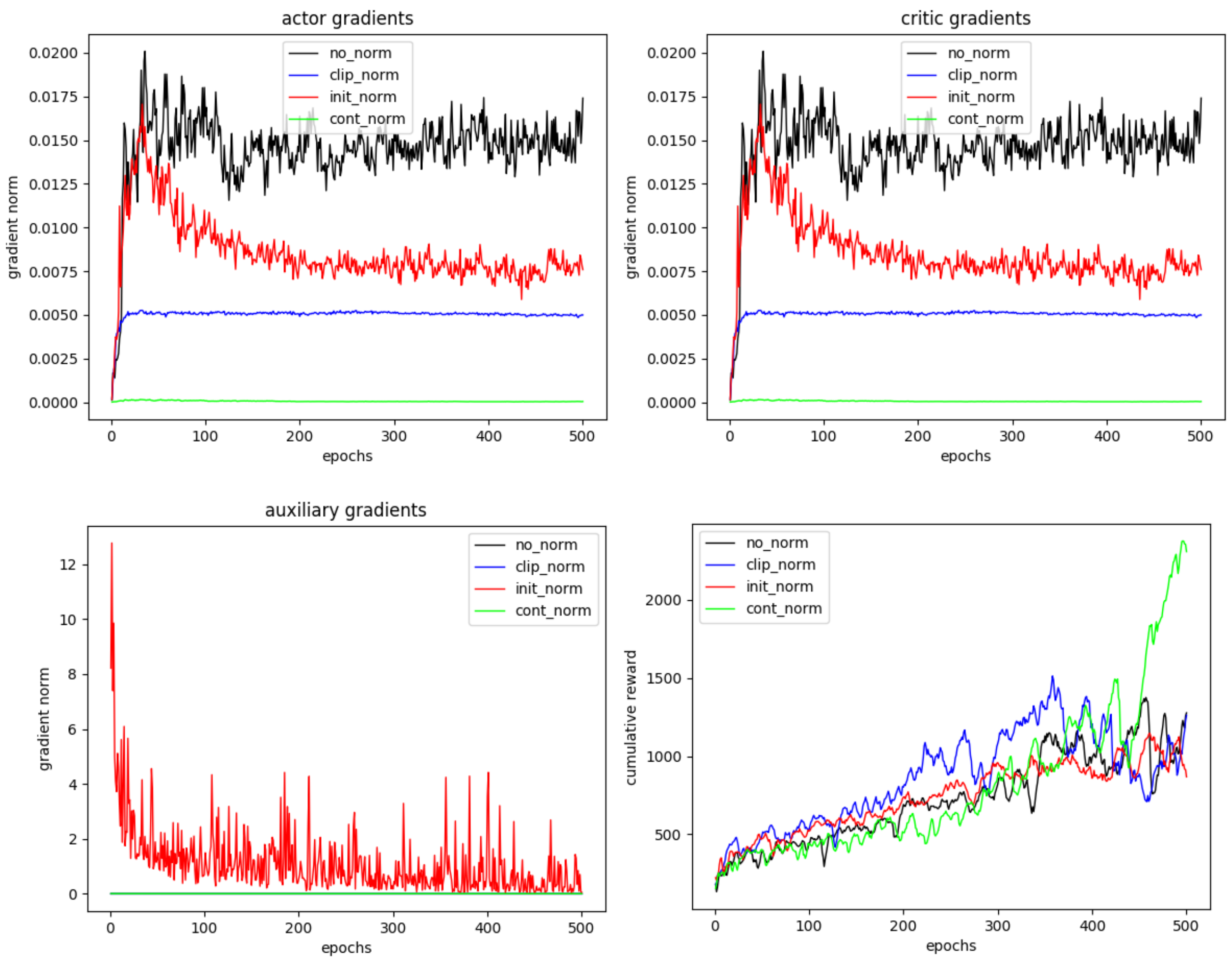

Figure 8. Gradient norm and cumulative reward for the different types of normalization. The causality auxiliary loss and Humanoid-v2 environment were utilized for these figures. The values of the auxiliary gradients for all norm types except for the initial loss norm are in the range of $1e-6$.

results with the application of continuous loss normalization, which improves the effectiveness and impact of auxiliary losses added to the model in all environments tested. Nonetheless, in the testing model, the application of multiple auxiliary losses was hindered by an inability of the model to balance its training between all given auxiliary losses. In future work, the gradients of each individual auxiliary loss could be compared to better investigate the relations and competitions between auxiliary losses. Furthermore, gradient clipping or other gradient normalization techniques could be explored for the each applied auxiliary loss individually, instead of as a whole vs. the main reinforcement learning gradients.

The evaluation of the robotic auxiliary losses as implemented in this paper was done on environments with a compact and descriptive state vector and descriptive immediate rewards, because of hardware and time limitations. We expect the robotic auxiliary losses to be of more prominent and consistent impact in environments with visual input and sparse or delayed rewards, and future work could explore their effectiveness in this kind of environments.

9. Conclusion

We have presented robotic auxiliary losses, functions that provide extra feedback to the agent in continuous reinforcement learning models applied to robotic tasks. Robotic auxiliary losses focus on Newton's principles of motion, instead of trying to add meaningful feedback for any arbitrary environment, and are thus specifically designed for application in the physical world (or simulations thereof). As auxiliary losses, they are applied concurrently to the reinforcement learning losses during training, guiding the formation of an internal state representation in the agent that is consistent with the laws of physics.

The application of robotic auxiliary losses has shown promising results: in each environment evaluated specific individual losses improved performance when applied jointly to the actor and critic networks. Concurrent application of all robotic auxiliary losses proved however to be less successful, with some losses being optimized during training and other losses being ”ignored”, possibly because of the (insufficient) width of the network’s layers.

More research is needed to best manage the influx of multiple losses on the network’s parameters, and to improve and stabilize their application in robotic reinforcement learning tasks. In future work, robotic auxiliary tasks could also be applied to environments with sparser rewards, or environments with visual state input, which may better benefit from the additional losses’ feedback to create more consistent internal state representation.

References

[1] R. Jonschkowski and O. Brock, *Learning state representations with robotic priors,* Autonomous Robots **39**, 407 (2015).

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, Vol. 1 (Springer, 1998).

[3] A. L. Samuel, *Some studies in machine learning using the game of checkers,* IBM Journal of research and development **3**, 210 (1959).

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, *Playing atari with deep reinforcement learning,* CoRR **abs/1312.5602** (2013).

[5] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, *Reinforcement learning with unsupervised auxiliary tasks,* CoRR **abs/1611.05397** (2016).

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, *Asynchronous methods for deep reinforcement learning,* in *International Conference on Machine Learning* (2016) pp. 1928–1937.

[7] J. Kober, J. A. Bagnell, and J. Peters, *Reinforcement learning in robotics: A survey,* The International Journal of Robotics Research **32**, 1238 (2013).

[8] J. C. Santamaría, R. S. Sutton, and A. Ram, *Experiments with reinforcement learning in problems with continuous state and action spaces,* Adaptive behavior **6**, 163 (1997).

[9] W. D. Smart and L. P. Kaelbling, *Practical reinforcement learning in continuous spaces,* in *ICML* (2000) pp. 903–910.

[10] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *Deterministic policy gradient algorithms,* in *ICML* (2014).

[11] 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,* arXiv preprint arXiv:1509.02971 (2015).

[12] S. B. Thrun, *Efficient exploration in reinforcement learning,* (1992).

[13] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell, *Loss is its own reward: Self-supervision for reinforcement learning,* arXiv preprint arXiv:1612.07307 (2016).

[14] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, *et al.*, *Learning to navigate in complex environments,* arXiv preprint arXiv:1611.03673 (2016).

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, *Reinforcement learning: A survey,* Journal of artificial intelligence research **4**, 237 (1996).

[16] R. Bellman, *Dynamic programming* (Courier Corporation, 2013).

[17] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity,* The bulletin of mathematical biophysics **5**, 115 (1943).

[18] J. Gantz and D. Reinsel, *The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east,* IDC iView: IDC Analyze the future **2007**, 1 (2012).

[19] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz, *Parameter space noise for exploration,* arXiv preprint arXiv:1706.01905 (2017).

[20] G. E. Hinton and R. R. Salakhutdinov, *Reducing the dimensionality of data with neural networks,* science **313**, 504 (2006).

[21] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion,* J. Mach. Learn. Res. **11**, 3371 (2010).

[22] S. Lange, M. Riedmiller, and A. Voigtlander, *Autonomous reinforcement learning on raw visual input data in a real world application,* in *Neural Networks (IJCNN), The 2012 International Joint Conference on* (IEEE, 2012) pp. 1–8.

[23] B. Boots, S. M. Siddiqi, and G. J. Gordon, *Closing the learning-planning loop with predictive state representations,* The International Journal of Robotics Research **30**, 954 (2011).

[24] R. Jonschkowski and O. Brock, *Learning task-specific state representations by maximizing slowness and predictability,* in *6th international workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS)* (2013).

[25] I. Newton, C. Huygens, A. Motte, and S. P. Thompson, *Mathematical principles of natural philosophy*, Vol. 34 (Encyclopaedia Britannica, 1952).

[26] S. Hochreiter and J. Schmidhuber, *Long short-term memory,* Neural computation **9**, 1735 (1997).

[27] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, *A survey of actor-critic reinforcement learning: Standard and natural policy gradients,* IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **42**, 1291 (2012).

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, *Openai gym,* arXiv preprint arXiv:1606.01540 (2016).

[29] E. Todorov, T. Erez, and Y. Tassa, *Mujoco: A physics engine for model-based control,* in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (IEEE, 2012) pp. 5026–5033.

[30] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley, and B. W. Suter, *The multilayer perceptron as an approximation to a bayes optimal discriminant function,* IEEE Transactions on Neural Networks **1**, 296 (1990).

[31] Z. Xu, J. Modayil, H. P. van Hasselt, A. Barreto, D. Silver, and T. Schaul, *Natural value approximators: Learning when to trust past estimates,* in *Advances in Neural Information Processing Systems* (2017) pp. 2117–2125.

[32] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization,* arXiv preprint arXiv:1607.06450 (2016).

[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, *Tensorflow: A system for large-scale machine learning.* in *OSDI*, Vol. 16 (2016) pp. 265–283.

Table 1. Model hyperparameters:

Hyperparameter	***Value***	
Layer normalization	True	*Normalize layer parameters by re-scaling and re-centering its weights[32]*
Observations normalization	True	*Normalize input observations by re-scaling and re-centering its values*
L2-regularization value	1e-2	*L2 regularization is applied to the critic to avoid overfitting*
Actor learning rate	1e-4	*Learning rate for actor model*
Critic learning rate	1e-3	*Learning rate for critic model*
Gamma (γ)	0.99	*Discount rate for state-action value determination (see equation 1)*
Tau (τ)	0.001	*Update rate for target actor and critic network (see equation 5)*
Similarity coefficient (λ)	10	*Steepness of descent factor of the similarity value in robotic auxiliary losses*
Adaptive Noise	0.2	*Desired standard deviation value of adaptive parameter noise for actor exploration*
Epochs	500	*Number of epochs to be run in a training*
Cycles per Epoch	20	*Number of cycles performed for each epoch*
Rollout steps	100	*Number of steps taken by the agent per epoch cycle*
Train steps	50	*Number of training iterations by the agent per epoch cycle*
Batch size	64	*Amount of state-action transitions to use at the same time for each training episode*
Representation layers	2	*Number of fully connected layers in the State Representation block (see Figure 2)*
Layers' width	64	*Number of neurons in the neural network's fully connected hidden layers*

A. Experiment setup

The OpenAI Baselines implementation of the Deep Deterministic Policy Gradient model is used as basis to which add auxiliary losses[7]. This implementation is written in Python, and makes use of the Google Tensorflow[33] Machine Learning library for the model building and training. All of the model hyperparameters hava been left unchanged, to properly evaluate the implementation of the robotic auxiliary losses against the accepted standard base DDPG model. The choices for all hyperparameters can be found in Table 1. For the robotic auxiliary losses backpropagation the respective learning rate for actor and critic networks has been used.

All tests are run for 500 epochs. Each epoch is comprised of 20 cycles of: (1) 100 iterations of the agent acting in the environment, which are stored in the experience replay (2) 50 training iterations, with a batch size of 64. A total of 1 million iterations are performed by the agent in the environment in a single complete run.

The code necessary to run the experiments presented in this paper can be found at:

https://github.com/TCherici/RoboticAuxiliaryLosses

[7]https://github.com/openai/baselines/tree/master/baselines/ddpg