Integrating State Representation Learning into Deep Reinforcement Learning

Tim de Bruin[1], Jens Kober[1], Karl Tuyls[2] and Robert Babuška[1]

***Abstract*—Most deep reinforcement learning techniques are unsuitable for robotics, as they require too much interaction time to learn useful, general control policies. This problem can be largely attributed to the fact that a state representation needs to be learned as a part of learning control policies, which can only be done through fitting expected returns based on observed rewards. While the reward function provides information on the desirability of the state of the world, it does not necessarily provide information on how to distill a good, general representation of that state from the sensory observations. State representation learning objectives can be used to help learn such a representation. While many of these objectives have been proposed, they are typically not directly combined with reinforcement learning algorithms. We investigate several methods for integrating state representation learning into reinforcement learning. In these methods, the state representation learning objectives help regularize the state representation during the reinforcement learning, and the reinforcement learning itself is viewed as a crucial state representation learning objective and allowed to help shape the representation. Using autonomous racing tests in the Torcs simulator we show how the integrated methods quickly learn policies that generalize to new environments much better than deep reinforcement learning without state representation learning.**

***Index Terms*—Deep Learning in Robotics and Automation; Learning and Adaptive Systems; Sensor Fusion**

I. INTRODUCTION

DEEP Reinforcement Learning (DRL) is a promising framework for enabling robots to perform novel tasks. Instead of having to explicitly program the required behaviors, only a reward function that captures the success of the robot at performing the task needs to be specified. Unfortunately, while it has been shown that this reward function provides a signal that can be used to find a mapping directly from sensory signals to correct actuator commands (e.g., [1], [2]), the relatively uninformative nature of the reward signal results in a large amount of experiences required to learn successful policies.

Manuscript received: September, 9, 2017; Revised December, 9, 2017; Accepted January, 12, 2018.

This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments.

This work is part of the research programme Deep Learning for Robust Robot Control (DL-Force) with project number 656.000.003, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). This work was additionally supported by a Qualcomm Innovation Fellowship.

[1]Tim de Bruin, Jens Kober and Robert Babuška are with the Cognitive Robotics Department, Delft University of Technology, The Netherlands {t.d.debruin, j.kober, r.babuska}@tudelft.nl

[2]Karl Tuyls is with Google Deepmind and the University of Liverpool karltuyls@google.com

Digital Object Identifier (DOI): see top of this page.

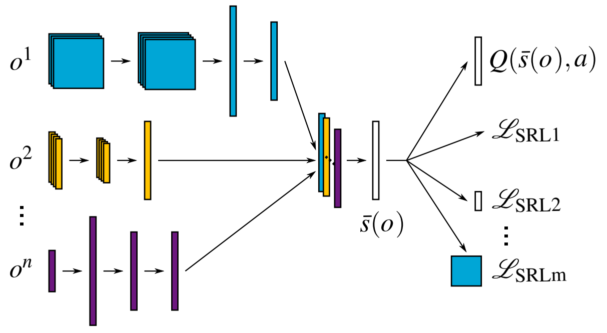
o^1
o^2
$\vdots$
o^n
$Q(\bar{s}(o),a)$
$\mathscr{L}_{\mathrm{SRL1}}$
$\mathscr{L}_{\mathrm{SRL2}}$
$\vdots$
$\mathscr{L}_{\mathrm{SRLm}}$
$\bar{s}(o)$

Fig. 1: General neural network architecture considered in this work. A shared embedding $\bar{s}(o)$ of all the different sensory modalities o^m is learned. Both the reinforcement learning cost function (2), as well as a number of state-representation learning cost terms $\mathscr{L}_{\mathrm{SRL1,\ldots,m}}$ are employed to shape the embedding during training.

This is especially troublesome when applying DRL to robotics. In this field, much of the complexity of learning a new task is in learning to perceive the world. Robot designers often equip their robots with several different types of sensors that estimate the state of the world through measuring different physical phenomena. Deep Learning has shown to be very capable of extracting descriptive features from high-dimensional, multi-modal inputs (e.g., [3]). However, while reward functions describe the desirability of the state of the world, they often provide only vague and indirect information on how to distill that state from the raw sensory observations. This further increases the number of required samples, which when combined with the high operating cost of robots, makes using reinforcement learning in this domain infeasible in general.

In this work, an autonomous racing car is used as a simulation benchmark. Images, range-finder readings and velocity data are observed and scalar rewards are given based on the velocity along the track-axis and the distance from the center of the track. With a wealth of high-dimensional sensor data and a scalar reward it is easy to learn the wrong causal relations. Was the negative reward due to the tree in the background, the combination of velocity and the distance to the track-edge or the color of the road markings? While some of these observations might be correlated to a good driving policy on one track, the learned policy will not necessarily work on a different circuit. Without a good compact and concise representation of the state of the problem, large amounts of diverse data will be required before the true causal connections outweigh the accidental ones and a general policy is found.

In order to make learning suitable state representations from the raw sensor data easier and faster, additional training criteria can be used to supplement the reinforcement learning objective [4]. These State Representation Learning (SRL) criteria can simplify the representation learning problem by encoding prior knowledge and can help to regularize the learned representation by making it adhere to some fundamental properties from physics. Examples of state representation learning criteria include the classical auto-encoding objective (e.g., [2], [5]–[7]), predicting instantaneous rewards (e.g., [8]–[10]), learning the (inverse) dynamics in the state embedding space (e.g., [11]–[13]) or encoding the belief that state representations should change only slowly over time (e.g. [9], [14]), while being diverse in general [15].

These additional optimization criteria have the potential to aid reinforcement learning, and even to substitute for reinforcement learning when a shaped reward function is not available [13], as is often the case is real world settings. However, realizing this potential can be non-trivial. When the auxiliary optimization terms are added to the reinforcement learning objective naively, performance can easily be reduced rather than improved. Many of the works that have introduced new state-representation learning criteria have done so for purposes other than reinforcement learning (e.g., [12], [13]), or with the state-representation learning separated from the reinforcement learning (e.g., [7], [10], [15], [16]). In this work we take a host of state-representation learning criteria from the literature. We then propose and compare different ways of integrating state-representation learning with popular deep reinforcement learning methods. Specifically, we make the following contributions:

- We propose and investigate a method for reducing the potential detrimental effects of changing the state representation on-line.
- We combine several state-representation learning objectives from the literature and investigate their contributions.
- We compare the effects of pre-training the state-representation and policy on data from a related domain to learning both from scratch during the reinforcement learning trials.
- We learn a shared state representation from multi-modal sensor observations.
- We perform state representation learning and reinforcement learning simultaneously, allowing the reinforcement learning process to help shape the state representation while the state representation learning helps regularize that representation.

The rest of this paper is organized as follows: in Section II we discuss the reinforcement learning and state representation learning objectives that we will employ in this work. Section III discusses the different methods we consider for combining these objectives in a way that maximally benefits the reinforcement learning process. To test these methods we perform experiments that are described in Section IV, the results of which we examine in Section V. Further discussion and conclusions are given in Section VI.

II. LEARNING OBJECTIVES

We consider an agent that interacts with an environment at discrete time-steps t. At each time-step the sensors of the agent perceive the effects of the hidden environment state s_t as sensory observations o_t^m. Here, $m \in \mathscr{M}$ indicates the sensor modality with $\mathscr{M}$ the set of modalities of the sensors that the agent is equipped with. We denote the set of all sensory observations at time-step t by o_t.

Based on these observations the agent sends an action a_t to the environment. As a consequence of the action, the environment transitions into a new state s_{t+1}. The agent receives a reward r_{t+1} which describes the desirability of the transition: $r_{t+1} = \rho(s_t, a_t, s_{t+1})$[1]. Since the true environment state is not directly accessible, the new state is perceived as o_{t+1}. Experience tuples $\{o_t, a_t, o_{t+1}, r_{t+1}\}$ are saved in an experience replay buffer $\mathscr{H}$ [1], [17] from which mini-batches are sampled uniformly at random during training.

The agent's goal is to choose those actions that maximize the cumulative discounted reward, or *return*, for any initial environment state s_0:

$$R = \sum_{t=0}^{T} \gamma^t r_{t+1} \qquad \gamma \in [0,1), \tag{1}$$

where T is the time-step at which the episode is terminated (∞ for continuous tasks).

To optimize for (1), a policy has to be found that maps the observations to actions: $a = \pi(o)$. These sensory observations tend to be high dimensional, noisy and redundant, which makes learning a policy from them directly based only on rewards both costly and prone to over-fitting.

Our aim therefore is to learn a mapping from the sensory observations to a low dimensional, concise representation of the task relevant aspects of the state: $\bar{s}(o)$. In this work we use a neural network to perform this mapping. This representation should allow learning a policy that generalizes better, while using less data. However, in contrast to many other works on representation learning, we consider reinforcement learning itself as a crucial state representation learning objective, and allow it to help shape the representation.

In the following subsections, we describe the different optimization criteria that we use to map the observations to state representations and the state representations to actions.

A. Reinforcement Learning

The first objective for which we optimize is the reinforcement learning objective (1). We use (Deep Double) Q-learning in this work [1], [18], [19] for its simplicity[2] and popularity. The algorithm learns to approximate the value of the return (1) when taking action a in a given state and following the optimal policy from the subsequent state onwards. It learns

[1]The reward might in practice be calculated based on information available in the observation, or based on additional information about the state that is only available during training, such as a motion capture setup.

[2]Continuous actions are future work, although we expect the results in this work to apply equally to that setting.

this function by minimizing the squared temporal difference error loss function:

$$\mathscr{L}_{RL} = \left(Q\big(\bar{s}(o_t), a_t\big) - \left[r_{t+1} + \mathbb{T}\gamma Q^{-}\left(\bar{s}(o_t), \pi(o_{t+1})\right)\right]\right)^2, \quad (2)$$

where $\mathbb{T} = 0$ for the terminal step of an episode and 1 otherwise. The network parameters of Q^{-} are older copies of those of Q, which are set equal to Q periodically for stability [1], [18]. The policy, excluding exploration, is simply $\pi(o) = \underset{a}{\arg\max} Q(\bar{s}(o_t), a)$.

B. Auto-encoding

Besides using the reinforcement learning objective to shape the state representation, we want to add additional objectives that encode some form of prior knowledge which can help simplify and regularize the state representation learning process by adding optimization targets and limiting the model search space. The most general prior knowledge that we encode is the knowledge that high dimensional sensory observations are often the result of a smaller number of relevant latent state variables [2], [5], [6], [20]. Additionally, we use the knowledge that the different sensors on a robot all measure different physical effects of the same environment state. We encode these beliefs in two ways.

The first is through the network structure. While the different sensory modalities have their own encoders, these encodings are then fused and embedded into a shared state embedding space, as shown in Figure 1. This embedding space is much lower dimensional than (some of) the sensory observations.

The second way in which we enable the state representation to encode significant aspects of the state is by reconstructing the observations of one or more of the sensory modalities from the shared embedding space, by minimizing the following loss:

$$\mathscr{L}_{AE^m} = \left\| \hat{o_t^m}\big(\bar{s}(o_t)\big) - o_t^m \right\|^2, \quad (3)$$

where $\hat{o_t^m}(\bar{s}(o_t))$ is the reconstruction of o_t^m made by a decoding layer in the network based on the state representation. In our autonomous car benchmark, we might for instance expect the representation that is learned to include the curvature of the road, which would help explain much of the variation in both the images and the range-finder measurements.

C. Reward prediction

While auto-encoding is very general, and encourages encoding all factors that can help explain the variation in the observed sensor data, we might want the state-representation to specifically focus on those aspects of the environment state that are relevant to the task that needs to be performed. The second SRL objective we consider is therefore predicting the instantaneous rewards received by the agent [8]–[10]. Doing this in addition to learning a value function helps especially when the rewards are sparse. However, even when this is not the case, this loss term is easier to optimize for than the temporal difference error (2) as changes to the policy will only change the data distribution and not the training targets.

We use the mean squared error as the reward prediction loss term:

$$\mathscr{L}_{rew} = \left(\hat{r}_{t+1}\left(\bar{s}(o_t), a_t, \bar{s}(o_{t+1})\right) - r_{t+1}\right)^2, \quad (4)$$

with $\hat{r}_{t+1}\left(\bar{s}(o_t), a_t\right)$ the prediction of the network, based on the state representation and action, of the reward r_{t+1}. For the driving task, predicting the reward would encourage encoding the velocity of the car as well as the position and orientation of the car relative to the track axis, regardless of the current policy. Note however that this is not sufficient for a good racing policy, as we would also need properties like the distance to the next corner, which does not influence the instantaneous reward. This is why we still consider the temporal difference loss (2) to be an important state-representation learning criterion.

D. Slowness and diversity

It is also possible to encode knowledge about physics, which should be applicable to state representation learning for robotics, regardless of the task [9]. This physical prior knowledge can be encoded as loss functions that act directly on the learned state-space embedding. One popular physical prior is that states should not change quickly over short periods of time [14]. A potential downside of encoding this belief is that its optimum is a state representation that does not change at all, and therefore contains no information. We can counter this by adding an additional loss term that encourages diversity between non consecutive states [15]. The slowness $\mathscr{L}_{slow}$ and diversity $\mathscr{L}_{div}$ loss terms we use are respectively:

$$\mathscr{L}_{slow} = \left\|\bar{s}(o_{t+1}) - \bar{s}(o_t)\right\|^2 \quad (5)$$

$$\mathscr{L}_{div} = e^{-\|\bar{s}(o_x) - \bar{s}(o_y)\|^2}. \quad (6)$$

In (6) o_x and o_y are non-consecutive observations. In practice we calculate the average of $\mathscr{L}_{div}$ over the experiences in a training mini-batch which is sampled uniformly at random from an experience replay buffer.

E. (Inverse) state dynamics

Since our eventual goal is to select optimal actions based on the state representation, it can also be beneficial to make sure the embedding specifically encodes those aspects of the world that can be changed by the agent's actions. This can be done by learning the inverse state-representation dynamics; predicting which action was responsible for the transition between two states [12], [13]. We also learn the forward dynamics by giving a prediction $\hat{\bar{s}}_{t+1}$ of the next state embedding $\bar{s}(o_{t+1})$ based on the current state embedding and action. We assume the environment state to be Markovian and in learning the forward dynamics we attempt to ensure that our state representation also has this property. Since we consider discrete actions in this work we use a classification loss term for the inverse dynamics $\mathscr{L}_{inv}$. The forward dynamics $\mathscr{L}_{fwd}$ are posed as a regression problem:

$$\mathscr{L}_{inv} = -\log\left(\hat{P}\left(a_t | \bar{s}(o_t), \bar{s}(o_{t+1})\right)\right), \quad (7)$$

$$\mathscr{L}_{fwd} = \left\|\hat{\bar{s}}_{t+1}\left(\bar{s}(o_t), a_t\right) - \bar{s}(o_{t+1})\right\|^2. \quad (8)$$

III. Integration Methods

The loss functions from the literature that were reviewed in the previous section have been used in a number of different ways. In some works, state-representation learning was not explicitly combined with reinforcement learning (e.g., [11], [13]). In others, the state-representation learning objectives were used during an initial pre-training phase while the state encoding was held (partially) fixed during the subsequent reinforcement learning phase (e.g., [7], [10], [15], [16]). Yet others use the auxiliary optimization objectives during the reinforcement learning phase (e.g., [9], [12], [21]) or even learn separate RL controllers that optimize for auxiliary tasks [8]. In this work, we use a large number of SRL objectives from the literature, and we propose and investigate different ways of integrating them with popular deep RL algorithms. The main aim for the combined methods is finding control policies that generalize well, while minimizing the number of required environment interactions.

A. Simultaneous optimization

The first and most straightforward way of integrating state-representation learning with reinforcement learning that we consider is to simply add the SRL objectives to the RL loss $\mathscr{L}_{RL}$ and optimize for this new loss function $\mathscr{L}_{sim}$ instead of the standard loss function used in the RL algorithm:

$$\mathscr{L}_{sim} = \mathscr{L}_{RL} + c_{SRL}\left(c_{SRL_1}\mathscr{L}_{SRL_1} + \cdots + c_{SRL_n}\mathscr{L}_{SRL_n}\right), \tag{9}$$

where $c_{SRL_{1,\dots,n}}$ are scaling constants for the individual SRL loss terms and c_{SRL} is an overall scaling term that trades off the SRL objectives with the RL objective. The individual loss scaling terms $c_{SRL_{1,\dots,n}}$ are the same in all our experiments. They were chosen once, such that the 2-norms of the gradients of the loss terms with respect to the embedding vector are of the same order of magnitude during the early stages of learning. We do vary the overall scaling c_{SRL} to investigate the effects of the trade-off between strictly enforcing our state-representation knowledge and allowing the reinforcement learning to mostly dictate the representation. In this work we refer to this method of optimizing simultaneously for all loss terms as `sim`.

During the learning process, we perform batch updates after each episode, with the number of updates dependent on the number of experiences obtained during the episode.

B. Alternating optimization with fixed Q values

One of the challenges of reinforcement learning, compared to supervised learning, is the fact that the training data distribution can change significantly as the result of a change in the policy. When using the `sim` method, this might cause some of the auxiliary loss terms to suddenly significantly change the state representation. This in turn can change the Q-values, which are dependent on the representation. As action gaps are generally small compared to the Q-values [22], even small changes in these values could inadvertently change their ordering and, as a consequence, the policy. This could further destabilize the learning process.

Fig. 2: Training and validation experiments are performed on four tracks. Pre-train data from a separate fifth track is used in some experiments.

To mitigate these effects we propose a second method, `alt`, in which the network parameters are updated in two alternating phases. First, we predetermine the experiences that will be sampled from the experience buffer. For these experiences, we determine the predicted Q-values with the current network parameters: $Q^i(\bar{s}(o), a)$ where i indicates the parameter update step at the start of the current SRL phase. We then first perform a number of update steps where we optimize for the state representation learning objectives, while attempting to minimize the changes to the predicted Q-values:

$$\mathscr{L}_{alt} = c_{\mathrm{Qfix}}\mathscr{L}_{\mathrm{Qfix}} + c_{SRL}\left(c_{SRL_1}\mathscr{L}_{SRL_1} + \cdots + c_{SRL_n}\mathscr{L}_{SRL_n}\right), \tag{10}$$

with:

$$\mathscr{L}_{\mathrm{Qfix}} = \left(Q\left(\bar{s}(o), a\right) - Q^i\left(\bar{s}(o), a\right)\right)^2. \tag{11}$$

After these updates we perform the same number of updates with the regular reinforcement learning objective $\mathscr{L}_{RL}$ (2).

IV. Experiments

We performed experiments with the Torcs [23], [24] racing simulator. The aim is to complete a lap of a track as quickly as possible. We use three different sensory modalities:

1) $o^{\mathrm{RGB}} \in \mathbb{R}^{12288}$: RGB images of 64 by 64 pixels, looking forward from the car.
2) $o^{\mathrm{T}} \in \mathbb{R}^{19}$: Measurements of the distance to the track edge at 10 degree intervals covering the front of the car. When the car is off the track the measurements are -1.
3) $o^{\mathrm{C}} \in \mathbb{R}^{5}$: The translational velocity of the car and the rotational velocities of each of the wheels.

For all experiments, the three different sensory modalities $o^{\mathrm{RGB}}, o^{\mathrm{T}}, o^{\mathrm{C}}$ are embedded into a 30-dimensional shared state-representation space $\bar{s}(o) \in \mathbb{R}^{30}$. We use the state-representation learning cost functions described in Section II, with both o^{RGB} and o^{T} as auto-encoding targets. For o^{RGB} we reconstruct a down-sampled image. Further details of the neural network architecture and reinforcement learning parameters are given in the Appendix.

As in other works, (e.g., [25]), we use a reward function that penalizes the distance from the center of the track as well as the velocity perpendicular to the track axis, while rewarding the velocity along the track axis:

$$r_{t+1} = v_{t+1}\left(\cos(\alpha_{t+1}) - |\sin(\alpha_{t+1})| - |d_{c_{t+1}}|\right), \tag{12}$$

with v the velocity of the car, α the angle between the car and the track-axis and d_c the distance between the car and the middle of the track. The distance d_c is normalized such that $|d_c| = 1$ at the edge of the track. Episodes are ended ($\mathbb{T} = 0$) when the car starts pointing in the wrong direction ($\cos(\alpha) < 0$), when the car stops after an initial grace period or when a lap is completed.

We use a pool of four tracks for training and testing. When we train an agent on one track, we test the generality of the learned controller on the remaining three. The hypothesis is that the SRL objectives will encourage a representation that allows the learned policies to generalize to new tracks as well. When we perform pre-training, the experiences are from a separate fifth track. All reported experiments are based on two trials per training track. Reported training performance is therefore averaged over 8 runs, while test performance is averaged over 24 runs.

To compare the algorithms across different tracks, we here define the *performance* as the mean reward observed during an episode, normalized between 0 and 1. For each track we define 0 as the mean reward during the first episode of the trials, when the controllers are untrained, and 1 as the best observed mean reward for any single episode over all experiments performed on the same track.

V. RESULTS

We start by comparing the performance of plain reinforcement learning to that of the two algorithms that include state representation learning considered in this work. For both `sim` and `alt` we choose $c_{SRL} = 0.5$ such that the 2-norms of the gradients of the state representation learning terms in the cost function with respect to the state embedding vector are about half that of the reinforcement learning term. For `alt`, we consider a version with c_{Qfix} chosen such that the 2-norm of the gradient of this term is similar to that of the reinforcement learning cost term and a version with $c_{\text{Qfix}} = 0$.

From the results in Figure 3 it can be seen that all algorithms manage to find control policies that yield good performance on the tracks that they are trained on. When the learned controllers are tested on different tracks however, it becomes clear that the controllers trained by reinforcement learning alone do not generalize well. The performance when including the state-representation learning cost terms is significantly better. As can be seen from Table I, the `alt` algorithm with the $\mathscr{L}_{\text{Qfix}}$ loss term consistently produced the best performing controllers on the training tracks, while the `alt` algorithm without this loss term gave the best generalization performance on three of the four tracks.

Sensitivity to c_{SRL}

Since we are interested in the integration of RL and SRL, we have investigated the effect of changing the weight c_{SRL} of all SRL terms compared to the RL cost term. We observed that scaling c_{SRL} down from 0.5 to 0.25 and 0.05 for the `sim` method resulted in slightly better training performance, with no clear difference in test performance. Overall, the algorithms sensitivity to this hyper-parameter seems quite limited, as a change of an order of magnitude did not produce clear performance differences. Still, adapting the scale of the individual loss terms dynamically [26] could be useful future work, as it could eliminate the tuning step altogether.

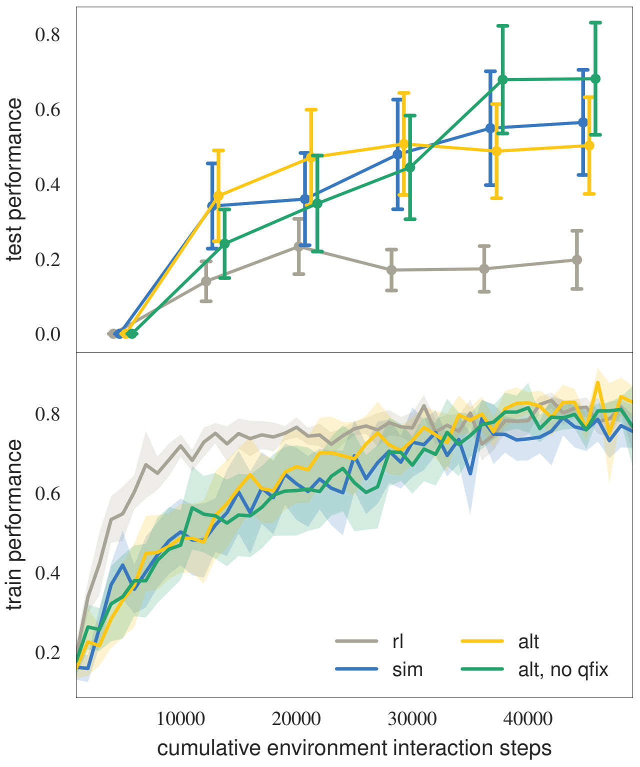

Fig. 3: Normalized performance for the `rl`, `sim` and `alt` methods, as well as the `alt` method without the $\mathscr{L}_{\text{Qfix}}$ (11) loss term. For the test performance, every 10^4 steps the network parameters that gave the best training performance up to that point are evaluated on the test tracks, without any additional training on those tracks. The mean $\pm$ half a standard deviation of the performance criterion from Section IV are shown.

Learning speed

Besides the potential for regularizing the state representation in a way that benefits generalization across domains, the other main appeal of adding the extra state-representation learning objectives is making the optimization problem for the state-representation easier by providing more stable and simpler objectives. Other authors, such as those of [8], have found

TABLE I: Best performing algorithm (of `rl`, `sim` and `alt` with and without the $\mathscr{L}_{\text{Qfix}}$ (11) loss term) on each of the training track / evaluation track combinations.

evaluation:	track 1	track 2	track 3	track 4
training track 1	`alt`	`alt`(no qfix)	`alt`(no qfix)	`alt`
training track 2	`sim`	`alt`	`alt`(no qfix)	`alt`(no qfix)
training track 3	`sim`	`alt`(no qfix)	`alt`	`alt`(no qfix)
training track 4	`sim`	`alt`(no qfix)	`alt`(no qfix)	`alt`

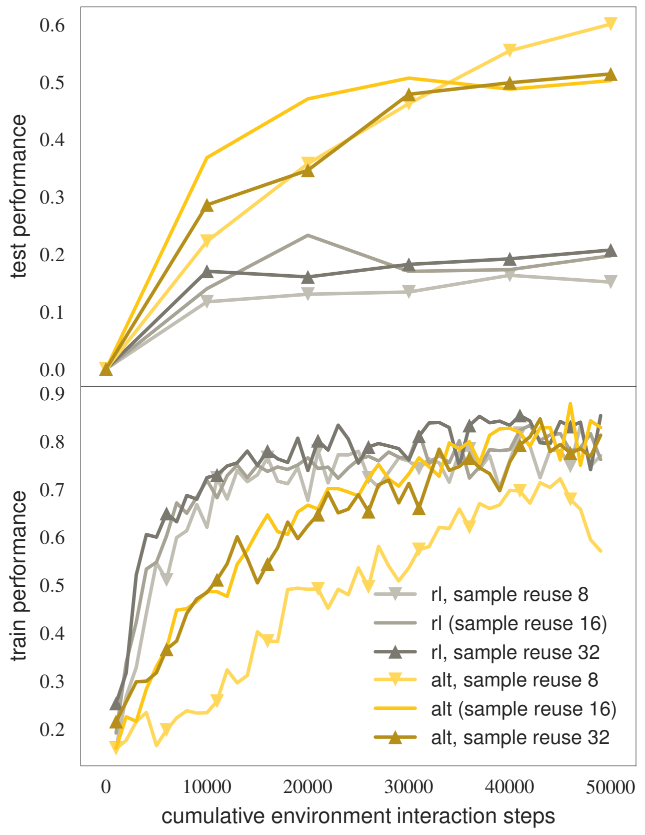

Fig. 4: The influence of the sample-reuse hyper-parameter on the train and test performance of the `rl` and `alt` methods. The means of the performance criterion from Section IV are shown.

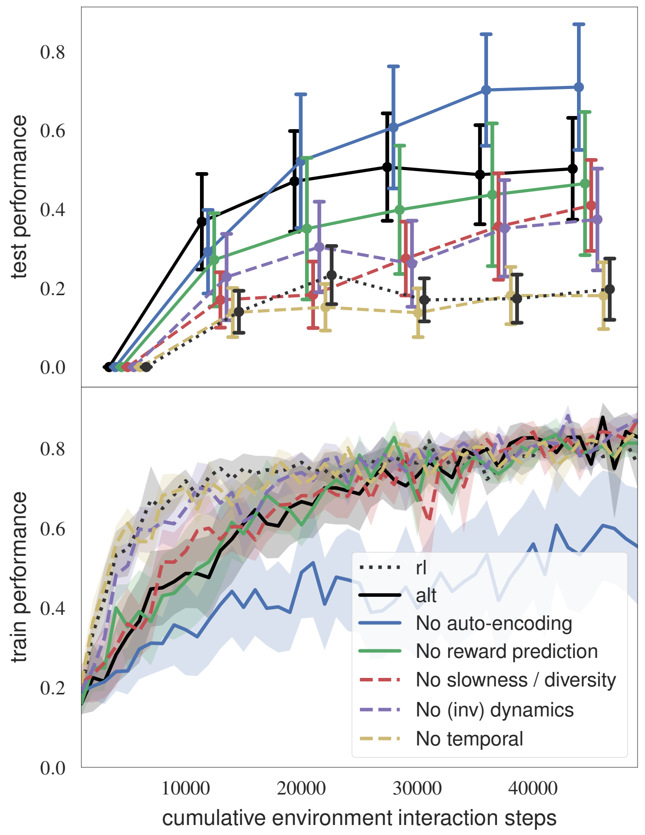

Fig. 5: The effect on the performance of the `alt` algorithm of turning off individual SRL losses. The no temporal experiment excludes both the (inv) dynamics and the slowness / diversity losses. The performance with all SRL losses (`alt`) and with no SRL losses (`rl`) are shown for comparison. The mean $\pm$ half a standard deviation of the performance criterion from Section IV are shown.

that the use of additional training objectives can speed up the learning process, also on the training domain. While Figure 3 showed that the `sim` and `alt` methods were able to find generalizing controllers more quickly, learning on the training domain was slower than that of plain `rl`.

A likely reason for these observations is that while some of the individual optimization problems posed by the SRL loss terms might be simple, we are optimizing for many of them at once, which makes the optimization more difficult. Additionally, by enforcing the physical priors we limit the space of suitable state representations, as the state representation should not only allow fitting the Q-values, but additionally adhere to the SRL constraints. While this results in the improved generalization performance, it makes finding a representation that allows decent training performance more challenging, as most SRL objectives do not limit the parameter search space. To further investigate the cause of the slower learning we vary the experience reuse and perform an ablation study of the individual SRL loss components.

1) Experience reuse: After each training episode, the newly observed experiences are added to the experience replay buffer and a number of optimization steps is performed. The number of updates is determined by the sample reuse hyper-parameter; the expectation of the number of times each experience is sampled as part of a mini-batch for an update. As the SRL objectives add richer training targets and serve as regularizers, we might expect that increasing the number of updates performed per new experience would be beneficial, especially compared to doing the same without the SRL objectives. To test this, we varied the sample reuse for both the `alt` and the `rl` algorithms from 16, which we use for all other experiments, to 8 and 32. The results are shown in Figure 4 and show that for our tests there is little to be gained from increasing the sample reuse beyond 16.

2) Individual SRL losses: We are also interested in how the individual losses contribute to both the speed of learning and the performance of the learned controllers. Figure 5 shows an ablation study in which, for the `alt` algorithm, each of the losses is separately turned off.

While all losses contribute to learning general control policies during the early stages of learning, the auto-encoding loss seems to hurt generalization in the later stages. Initially, this loss might help quickly shape the convolutional feature maps through the dense training targets. The input reconstruction objective is however the most general objective and the least specialized towards reinforcement learning. While the objective seems to help with the learning stability[3], it might hurt the test performance in the later stages, where it forces the state representation to capture information that, while it might help explain the variation in the sensor data on the training track, might not be relevant to the task at hand. The other losses all benefit the generalization performance.

The (inverse) dynamics loss can be seen from Figure 5 to be the primary reason for the slower learning on the training domains, as excluding it yields a similar learning curve to the `rl` method. Leaving out either the (inverse) dynamics or the slowness and diversity loss terms results in a large drop in generalization performance.

MDP dynamics encoding

When leaving out both the slowness and diversity as well as the (inverse) dynamics losses (*no temporal* in Figure 5), the generalization performance degrades to the level of the plain `rl` method. This shows that, at least on the Torcs domain, explicitly learning to encode the temporal aspects of the environment into the representation of the state is the most beneficial for the generalization performance.

The incorporation of the MDP dynamics into the state representation to aid generalization is also the idea behind successor representations [27]–[29]. These methods use a prediction of the occupancy of future states as a representation of the current state, something that might be biologically plausible [30]. While for successor representations the representation is a function of the MDP dynamics and the current policy, our state representation learning losses are all off-policy and only a function of the environment dynamics. However, the slowness objective does encourage successive states in the experience buffer to have similar representations.

Pre-training

So far we have investigated integrating state-representation learning directly into the reinforcement learning process. An alternative to this approach is to first learn a state-representation and to then perform reinforcement learning, either while keeping the representation fixed, or while allowing it to be adjusted further. We investigate the potential of pre-training using a fixed dataset obtained by a reinforcement learning controller on a separate track.

In Figure 6 the performance with pre-training is compared to the performance without. We pre-train using the SRL objectives with or without the RL loss. Subsequently, we train while either keeping the representation fixed (and only changing the parameters of the RL decoder), or we perform training with the `rl` or `alt` algorithms as usual. The results show that while pre-training enables quick adaption to a new track and can help generalization, the learned representation is not good enough to allow competitive performance without adaption of the representation in the on-line learning phase.

[3]Note that since the test performance is evaluated every 10^4 steps for the best performing controllers up to that point, the test performance is less sensitive to the learning stability than the training performance.

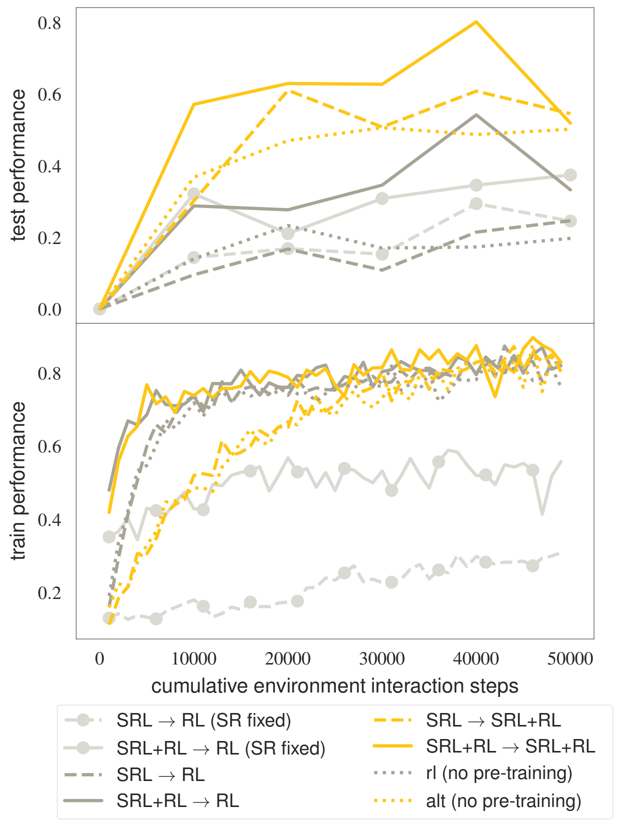

Fig. 6: Performance when starting with a network pre-trained with data from a separate track. The notation is *pre-train method* → *train method*. SR fixed indicates that the state representation is kept fixed during the on-line learning phase and only the RL decoder is adapted. The means of the performance criterion from Section IV are shown.

VI. CONCLUSIONS

We have investigated several ways of integrating State Representation Learning (SRL) objectives into standard deep Reinforcement Learning (RL). During all stages of learning, we allowed the reinforcement learning objective to help shape the state representation and we used the state representation learning objectives to regularize that representation.

The regularization resulted in a small improvement on the training domain and a significant improvement on the test domain. While we combined state representation learning criteria from a number of different works, little effort was put into scaling their relative importance, and it was found that the methods were not sensitive to the hyper-parameter that determined the ratio between the weights of the SRL and RL objectives. The *slowness and diversity* and the *(inverse) dynamics* SRL objectives were found to be most beneficial to the generalization performance, while the auto-encoding objective benefited the learning stability and speed at the cost of the eventual generalization performance.

Compared to combining SRL and RL directly in a single update, our proposed method of alternating between these objectives yielded better performance. While limiting the changes to the value function predictions during the SRL updates consistently gave the best performance on the training domain, not doing so tended to result in better performance on the test domain.

APPENDIX

Network architectures

In this work we used the following neural network architecture. Three separate encoders were employed:

- For the camera images o^{RGB}, the same convolutional architecture as in [1] was used, without the final fully connected layer but with normalization layers [31].
- For the track (range finder) measurements o^{T} a fully connected encoder was used with two ReLU layers of 50 and 25 units respectively.
- The car observations o^{C} encoder, uses a single layer of 25 units with ReLU nonlinearities.

All encoders are followed by a single linear layer of size 30. The shared representation is obtained by averaging over the different modalities to get a single shared representation $\bar{s}(o) \in \mathbb{R}^{30}$.

The decoders, for the Q-values, downsized (32x32x3) RGB targets, and the reconstruction of the track measurements all use an affine transformation of the state embedding.

Optimization was done with the ADAM algorithm, using a learning rate of $3 \cdot 10^{-4}$ and $\beta_1 = 0.9, \beta_2 = 0.999$ [32].

Reinforcement learning

Epsilon greedy exploration was used with a $\varepsilon = 0.15$ during training. To evaluate the performance of the learned controllers, tests were performed on the test track for 2000 environment steps with $\varepsilon = 0.05$ to ensure some variation in the roll-outs and to test robustness. We used a discount factor of $\gamma = 0.95$. The considered actions were $a \in \{[-0.5, 0.2], [0.5, 0.2], [-0.2, 0.4], [0.2, 0.4], [0, 0.6], [0, -0.8]\}$, with the first dimension the steering command and the second the acceleration / break command. Experiences were stored in a replay buffer with a capacity of $2 \cdot 10^{4}$. We used a batch size $k = 16$ and sample reuse $K = 16$.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *Int. Conf. Robotics and Automation (ICRA)*, 2016.

[3] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Int. Conf. Machine Learning (ICML)*, 2011.

[4] R. Caruana, "Multitask connectionist learning," in *Connectionist Models Summer School*, 1993.

[5] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[6] S. Lange, M. Riedmiller, and A. Voigtlander, "Autonomous reinforcement learning on raw visual input data in a real world application," in *Int. Joint Conf. Neural Networks (IJCNN)*, 2012.

[7] H. van Hoof, N. Chen, M. Karl, P. van der Smagt, and J. Peters, "Stable reinforcement learning with autoencoders for tactile and visual data," in *Int. Conf. Intelligent Robots and Systems (IROS)*, 2016.

[8] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *Int. Conf. Learning Representations (ICLR)*, 2017.

[9] R. Jonschkowski and O. Brock, "Learning state representations with robotic priors," *Autonomous Robots*, vol. 39, no. 3, pp. 407–428, 2015.

[10] J. Munk, J. Kober, and R. Babuška, "Learning state representation for deep actor-critic control," in *Conf. Decision and Control (CDC)*, 2016.

[11] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," in *Neural Information Processing Systems (NIPS)*, 2015.

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

[13] P. Agrawal, A. V. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: Experiential learning of intuitive physics," in *Neural Information Processing Systems (NIPS)*, 2016.

[14] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural Computation*, vol. 14, no. 4, pp. 715–770, 2002.

[15] R. Jonschkowski, R. Hafner, J. Scholz, and M. Riedmiller, "PVEs: Position-velocity encoders for unsupervised learning of structured state representations," in *New Frontiers for Deep Learning in Robotics Workshop at RSS*, 2017.

[16] A. X. Lee, S. Levine, and P. Abbeel, "Learning visual servoing with deep features and fitted Q-iteration," in *Int. Conf. Learning Representations (ICLR)*, 2017.

[17] L.-H. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3/4, pp. 69–97, 1992.

[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Conf. Artificial Intelligence (AAAI)*, 2016.

[19] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[20] A. Ghadirzadeh, A. Maki, D. Kragic, and M. Björkman, "Deep predictive policy training using reinforcement learning," *arXiv:1703.00727*, 2017.

[21] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu *et al.*, "Learning to navigate in complex environments," in *Int. Conf. Learning Representations (ICLR)*, 2017.

[22] M. G. Bellemare, G. Ostrovski, A. Guez, P. S. Thomas, and R. Munos, "Increasing the action gap: New operators for reinforcement learning." *Conf. Artificial Intelligence (AAAI)*, 2016.

[23] B. Wymann, E. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "Torcs, the open racing car simulator," *Software available at http://torcs.sourceforge.net*, 2000.

[24] N. Yoshida, "Gym-torcs," *Software available at https://github.com/ugo-nama-kun/gym_torcs*, 2016.

[25] G.-H. Liu, A. Siravuru, S. Prabhakar, M. Veloso, and G. Kantor, "Learning end-to-end multimodal sensor policies for autonomous navigation," *arXiv:1705.10422*, 2017.

[26] H. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Neural Information Processing Systems (NIPS)*, 2016.

[27] P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Computation*, vol. 5, no. 4, pp. 613–624, 1993.

[28] A. Barreto, R. Munos, T. Schaul, and D. Silver, "Successor features for transfer in reinforcement learning," *Neural Information Processing Systems (NIPS)*, 2017.

[29] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman, "Deep successor reinforcement learning," *arXiv:1606.02396*, 2016.

[30] K. L. Stachenfeld, M. M. Botvinick, and S. J. Gershman, "The hippocampus as a predictive map," *Nature Neuroscience*, vol. 20, no. 11, pp. 1643–1653, 2017.

[31] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Int. Conf. Machine Learning (ICML)*, 2015.

[32] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Int. Conf. Learning Representations (ICLR)*, 2015.