LOCALLY CONSTRAINED REPRESENTATIONS IN REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The success of Reinforcement Learning (RL) heavily relies on the ability to learn robust representations from the observations of the environment. In most cases, the representations learned purely by the reinforcement learning loss can differ vastly across states depending on how the value functions change. However, the representations learned need not be very specific to the task at hand. Relying only on the RL objective may yield representations that vary greatly across successive time steps. In addition, since the RL loss has a changing target, the representations learned would depend on how good the current values/policies are. Thus, disentangling the representations from the main task would allow them to focus more on capturing transition dynamics which can improve generalization. To this end, we propose locally constrained representations, where an auxiliary loss forces the state representations to be predictable by the representations of the neighbouring states. This encourages the representations to be driven not only by the value/policy learning but also self-supervised learning, which constrains the representations from changing too rapidly. We evaluate the proposed method on several known benchmarks and observe strong performance. Especially in continuous control tasks, our experiments show a significant advantage over a strong baseline.

1 INTRODUCTION

Representation Learning is a crucial problem in machine learning, particularly in understanding how complex inputs can be represented in a compact, well-defined manner, while retaining useful information of the input. This has been studied extensively for vision and natural language processing tasks but not as much for Reinforcement Learning (RL). RL has shown great success in games like Atari Mnih et al. (2015), Go Silver et al. (2016; 2018) and Chess Silver et al. (2018). However, its application in the real-world setting is limited. One main reason for this problem is the lack of an appropriate input feature space. In games, the input space is rather well-defined, e.g. it can be the state of the game for board games or it can be the image of the game screen in computer games Mnih et al. (2015).

However, it has been shown that learning policies directly from high-dimensional inputs can be sample inefficient Lake et al. (2017). Additionally, in real-world scenarios, we often have inputs from different sensors that the agent needs to fuse into a compact representation, which can be challenging. Therefore, representation learning is a key to enabling reinforcement learning agents to solve real-world problems.

Most of the work on representation learning can be divided into the following categories depending on the learning methods: supervised Watter et al. (2015) and self-supervised Srinivas et al. (2020) learning, both of which can benefit from data augmentation Laskin et al. (2020); representation learning with additional auxiliary tasks Jaderberg et al. (2016); and using state metrics Ferns et al. (2011). Our approach falls into self-supervised learning category. The objective encourages state representations to be predictable as a linear combination of the representations of *nearby* states.

In this paper, we first provide a brief background of the supervised and self-supervised methods that exists in the literature2. We then explain the proposed method and how it fits into the general landscape of representation learning methods for RL.3 Following that, we detail the implementation

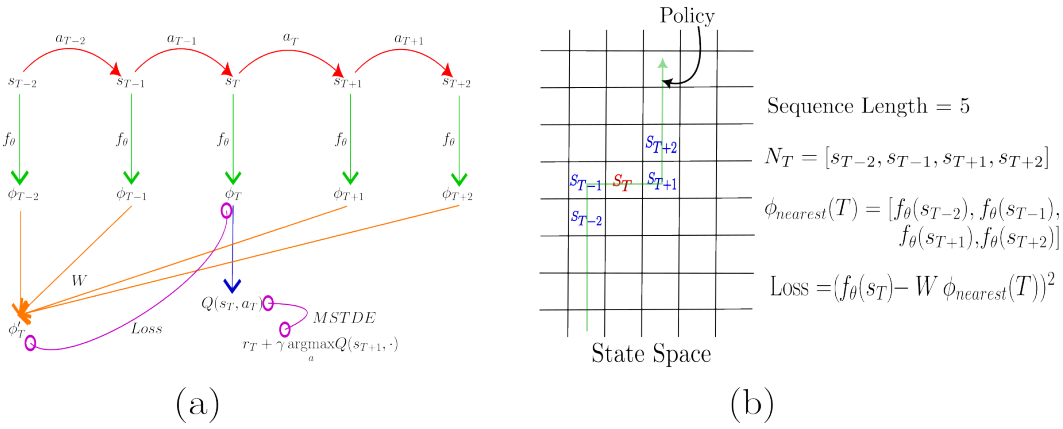


Figure 1: (a) Proposed Locally Constrained Representations Algorithm. (b) An example shown for sequence length of 5, where the current processed state is shown in red and the neighbouring states considered are in blue. The loss calculated is fed directly to the *Loss* in (a)

of the algorithm and evaluate its performance on several benchmarks4. Finally, we describe the key benefits of this algorithm and the scope of future improvements5.

2 Background & Related Works

In Reinforcement Learning (RL), the agent-environment interface is represented by Markov Decision Processes (MDPs), defined by $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} >$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{P}$ represent the state space, the action space, the reward function and the transition function of the environment, respectively. RL is a sequential decision making problem where the goal is to find an optimal policy (a mapping from states to actions). An optimal policy is a policy that maximizes the total *discounted* return with discount factor, $\gamma \in [0, 1]$, obtained when starting from a particular state and following that policy. Thus, mathematically, an optimal policy, π^* is:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | S_t = s, A_t = a]$$

where $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$ and $R_t \in \mathcal{R}$ are the states, actions and rewards at time t. The key problem we intend to address here is how to find a good representation of the *agent state* in conjunction with the main RL problem. In this section, we will discuss the current approaches for representation learning using supervised and self-supervised methods and describe where our algorithm fits in the literature space.

Reconstruction Loss and Learning Forward Models: The representation is learned by two losses, the original RL loss, that is, the Mean Squared TD Error (MSTDE) and the Auxiliary Loss. The auxiliary target $\hat{Y}_t$ depends on the representation learning algorithm used. For example, it can be trying to predict the next reward or the next state Chung et al. (2018) or even reconstruct the current state. Additionally, it can also be General Value Functions Sutton et al. (2011) that learn subtasks. The most popular approach is to design a low-dimensional representation that is capable of reconstructing the current state. The commonly used method is Auto Encoders Hinton & Salakhutdinov (2006) that can learn well from high-dimensional observations, such as images Mattner et al. (2012); van Hoof et al. (2016); Watter et al. (2015). Recently, Gelada et al. (2019) and Schwarzer et al. (2021) developed forward models that can predict the next state in the latent space without the need for reconstruction. Additionally, these methods can predict multiple steps in the future instead of a one-step prediction.

A large portion of the work on forward models was mainly developed for building world models Ha & Schmidhuber (2018). These were used for learning the complete dynamics of the world, so that the agent can perform model-based reinforcement learning without the need for interaction with the world. Recently, Hafner et al. (2018), Hafner et al. (2019) and Hafner et al. (2020) have developed model-based RL approaches using forward models in the latent space.

Data Augmentations & Contrastive Learning: Data augmentation methods are commonly used for image input data. These methods randomly apply different transformations, such as cropping, rotation, or jitter, to generate additional training data, which helps the agent to be robust to small variations in the input space. Laskin et al. (2020) and Yarats et al. (2021b) showed that augmenting the states with such modulations can improve the performance of RL without dedicated representation learning objectives.

Contrastive learning is a self-supervised learning algorithm that takes advantage of similarity constraints between data for representation learning van den Oord et al. (2018); Chen et al. (2020). The most popular integration of contrastive learning comes with data augmentation, where positive pairs are modulations of the same image, whereas positive and negative pairs are modulations from different images. The main goal of the algorithm Srinivas et al. (2020) is to increase the similarity between positive pairs while keeping negative pairs far apart. Other recent approaches include prototypical representations Yarats et al. (2021a), predictive information Lee et al. (2020) and augmented temporal contrast Stooke et al. (2020) to associate pairs within a short time.

Constraining State Representations by Prior Knowledge: Often, the agents can leverage prior knowledge of the environment to constrain the representations. A very common method would be to assume that the "interesting" features of the state vary slowly, and thus setting a constraint on the state representation can be beneficial Kompella et al. (2011); Jonschkowski et al. (2017). Jonschkowski et al. (2017) also used another prior that the positions of important objects (example, the agent) would vary and this forces the representations to change. Using both slow and variable features creates a trade-off and prevents the representations from becoming constant.

Jonschkowski & Brock (2015) introduced two priors for robotics which are proportionality and repeatability. Proportionality introduces a constraint where the representations of two states would vary by the same amount for the same actions taken in the two states. Repeatability adds a constraint that forces representations of similar states to vary similarly when the same action is applied to them.

Our proposed Locally Constrained Representations (LCR) algorithm falls in this category. LCR imposes the constraint that state representations should be linearly predictable by representations of surrounding states. In the following section, we describe the intuition behind this approach, formulate an algorithm and provide details on its implementation.

3 Locally Constrained Representations

3.1 Intuition

Our main motivation for locally constraining state representations comes from the assumption that in most Reinforcement Learning environments the main objects of interest rarely change rapidly with each action. Under this assumption, the dynamics should be relatively simple in local policy space, where the states are a few actions apart.

In order to exploit this for representation learning, we introduce the soft constraint that the representation of a state should be closely approximated by a linear combination of neighboring state representations. This constraint encourages the state representation to remain close to a manifold on which local dynamics can be expressed in a simple function of linear form. Without this constraint, the representations can change arbitrarily from one state to the next as the representations become intertwined with the local reward signal from the environment. This might result in over-fitting as the representation would fail to consider the dynamics of the environment.

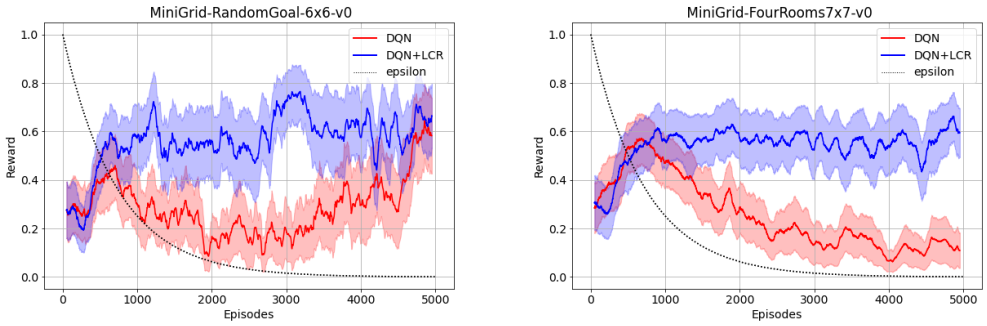
Even predicting a single future step Chung et al. (2018) in a self-supervised learning loss can take advantage of the smoothness of the state space. Predicting multiple steps Gelada et al. (2019); Schwarzer et al. (2021) in the future can improve the learned representation by helping it capture features that do not change much (example: slow-changing features, such as objects). Similarly in LCR, since we use multiple neighboring states for prediction, it can help to capture such features.

3.2 ALGORITHM

The core idea of this method is to impose a constraint on the representation of a state and to encourage it to be linearly predictable by the representations of nearby states. This learning formulation is inspired by locally linear embeddings Roweis & Saul (2000), where the embeddings are constrained to be linearly representable by the nearest data points. Here we define the neighbourhood to be a window of K states in the sequence of transitions that are chosen by the current behaviour policy. In this way, the representation of a state takes into consideration neighboring states instead of only focusing on the RL objective, which can improve generalization. However, we do not want features that are completely unrelated to the main task, so we only impose a *soft constraint* on the representation.

Algorithm 1: Locally Constrained Representations

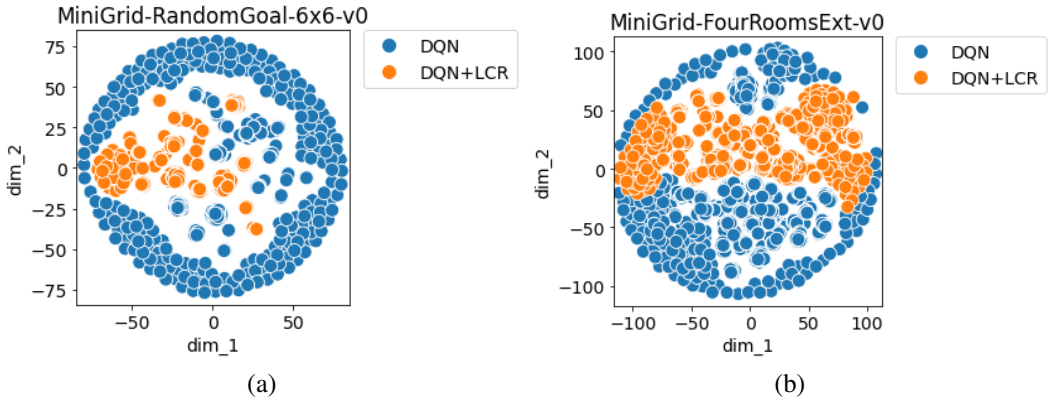
```
Function LCR(sequence length, steps, batch):
    t ← 0
    for update in 1,2,3 ... updates do
        Observe state s
        Take Action a
        Train policy/value network by base RL algorithm
        if t % batch == 0 then // Run LCR on the last batch
            All States = GetLastBatch(t, batch)
            Initialize W ~ Uniform(0,1)
            N ← Set of all Neighbours in Sequence
            for state in All States do
                N[T] = GetSequence(sequence length, state)
            end for
            φ_nearest(T) = f(N)
            Loss = (f(All States) − W φ_nearest(T))^2
            for step in 1, 2, 3 ... steps do
                Run Gradient Descent on LCR Loss
            end for
        end if
    end for
```



(a) (b)

Figure 2: Performance of DQN (red) and DQN with LCR (blue) on the MiniGrid Environments. Both algorithms were trained for 10 runs with LCR using a sequence length of 11, 100 gradient steps and batch size 5000. The detailed hyperparameters are mentioned in the appendix.

We need to keep a batch of state transitions of size B, on the basis of which we constrain the representations. Since RL is online, we need additional memory to keep track of the samples. If the main RL algorithm already uses an experience replay buffer Lin (2004); Mnih et al. (2015), we can directly take the last B samples from the buffer. More training samples mean better state-



(a) (b)

Figure 3: tSNE plots of the state representations obtained by 20 random trajectories of the respective environments.

space coverage and better generalization. So higher batch sizes are often better and this comes as a trade-off with respect to compute and memory requirements.

Once we have a batch, B, of states, we obtain K neighbours of each sample from the trajectory, assuming total sequence length is $K+1$, including the sample itself. We then add a constraint to encourage the representation of the sample to be predictable by the representations of the neighbors.

Let s_T be a sample drawn from the buffer of length B. Also, let $s_{T-K/2}, \ldots, s_{T-1}, s_{T+1}, \ldots, s_{T+K/2}$ be the nearest neighbors. Let

$$\phi_T = f(s_T)$$

be the D-dimensional representation of state s_T, where f is a parameterized function. For all of our experiments, we use a neural network to implement f. Also, we define

$$\phi_{nearest}(T) = [\phi_{T-K/2}, \ldots, \phi_{T-1}, \phi_{T+1}, \ldots, \phi_{T+K/2}]^{\mathrm{T}},$$

where $\phi_{nearest}(T) \in \mathbb{R}^{K \times D}$ is the matrix of neighbouring state representations stacked row-wise. We define the loss

$$\sum_{T=1}^{B} ||W\phi_{nearest}(T) - \phi_T||_2^2,$$

where $W\phi_{nearest}(T)$ is the predicted representation of the state by linear combination of the neighbours with coefficients $W \in \mathbb{R}^{1 \times K}$. We learn W and f by minimizing this loss using gradient descent steps. After every new batch of samples of size B, this optimization is done in an interleaved manner with the standard RL training. Since this is added as an auxiliary loss for representation learning, it can be applied on top of any RL algorithm as an auxiliary loss.

We want a soft constraint on the representation and thus we do not solve this optimization completely but take only a few gradient descent steps while keeping W positive (by clipping all negative weights to 0) to limit the solution space. This is because we are only solving this on a limited batch size with a short state-space coverage, so taking too many steps can overfit to the samples in the batch. Figure 1 (a) shows a diagram of the method.

4 Results

In this section, we test our algorithm in 14 different environments (2 from MiniGrid environments Chevalier-Boisvert et al. (2018) and 8 environments from Mujoco Todorov et al. (2012), and 4 environments from Arcade Learning Environment (Atari) Bellemare et al. (2013)). The reason for choosing most of these environments is that there are well-defined learnable dynamics and the states change *slowly* in a given local policy space. For diversity, we also analyze the performance of LCR in simpler environments such as CartPole and Acrobot.

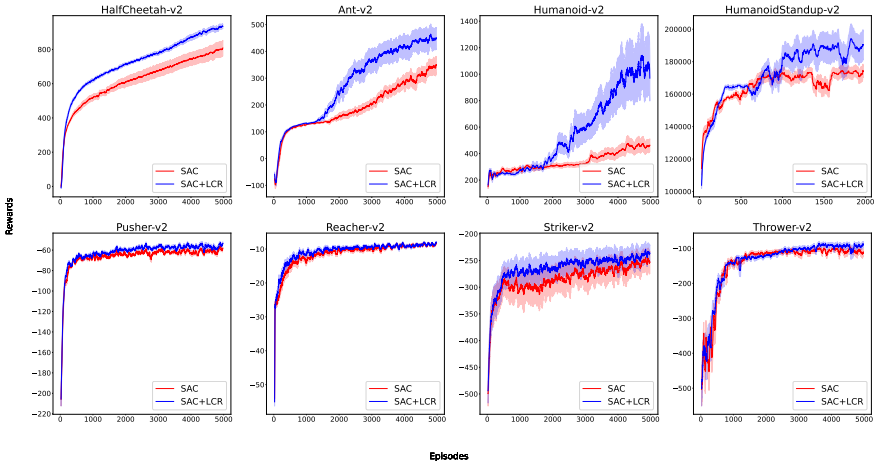


Figure 4: Training curves on 8 Mujoco environments using SAC with and without LCR across 5 runs. For LCR, we used sequence length of 11, 100 gradient steps and batch size of 5000.

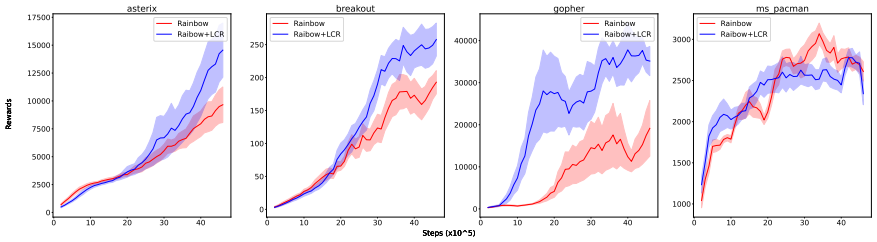


Figure 5: Training curves on 4 Atari environments using Rainbow with and without LCR for 5 Million Frames across 5 runs. For LCR, we used sequence length of 11, 20 gradient steps and batch size of 1000.

4.1 Implementation Details and Baselines

The description of the environments and the detailed hyper-parameters are mentioned in the supplementary material along with the code for all the environments for reproducibility of our results. We provide ablation studies of the respective hyper-parameters introduced by LCR, both in the main paper and in the supplementary material, and discuss a suitable thumb rule so that extensive sweeps are not required.

For our baselines, we intentionally avoided choosing other auxiliary representation learning methods. Our main aim in this paper is to highlight how this strategy of localizing representations can actually help in generalization in representation learning and all our experiments henceforth highlight that. Furthermore, the experiments that directly perform similar constraints on the environment Jonschkowski et al. (2017), were mostly designed for robotics, while Locally Constrained Representations (LCR) is a more general applicable method. Since this is an additional constraint that is applied in the form of an auxiliary loss, it can be easily added on top of any aforementioned representation learning approach.

4.2 MiniGrid Environments

LCR helps in learning representations which prevents over-fitting in environments by constraining the representations locally. So, given a stochastic environment, is the representation learned by LCR more generizable? We try to answer this question by choosing two environments from the MiniGrid

suite Chevalier-Boisvert et al. (2018), where the goals change randomly after every episode. In both environments, the agent has to navigate a grid and reach a goal. The agent receives a reward of $+1$ on reaching the goal and -0.01 for each step that it takes. We designed the environment *RandomGoal*, which is an extension of the EmptyRoom environment from MiniGrid Chevalier-Boisvert et al. (2018) with the goal position changing randomly at the end of every episode. *FourRooms* environment is the same as the generic one from MiniGrid Chevalier-Boisvert et al. (2018), but with reduced size to alleviate the problem of exploration which we are not studying in this paper. We run DQN Mnih et al. (2015) and DQN with LCR in both environments with fully observable input. Since the output is a 2D matrix, both architectures have 3 convolutional layers, with the first 2 followed by a max-pooling layer. The output of the last convolutional layer is "flattened", that is, reshaped into a vector. This entire network corresponds to f_θ in Fig. 1 (a). The output of this network is then passed to a value network which consists of two dense layers and which outputs the Q values. The entire of flatten layer is representation layer (ϕ in Fig. 1 (a)) on which LCR is applied. Exploration is handled by the decaying ϵ-greedy strategy, which is shown by the dotted black line.

Figure 2 demonstrates that LCR is able to perform better because it does not overfit to one goal location. This problem can happen with learning representations with the RL loss only. As a result, DQN (red) performs poorly in both settings, with complete failure in the FourRooms environment. Adding LCR on top of DQN however, improves the performance significantly, reaching near-optimal policies in both cases. Note, that the final rewards are noisy because the optimal reward changes with the location of the goal.

Constrained Representations: Figure 3 additionally shows the tSNE van der Maaten & Hinton (2008) plots in 2 dimensions for the representations learned by DQN and DQN with LCR after training for 5000 episodes. The plots represent the representations of the states ϕ, in 20 trajectories with the same random policy (so that they all visit the same states equally). It can be seen that the spread of different state representations is much smaller for LCR even though both of the algorithms visit the exact same states. This indeed confirms that LCR learns a much more constrained representation. This is especially true for the *RandomGoal* environment, where most of the states would be easily represented by neighboring states because the grid is empty. On the other-hand, the representations learned by DQN without LCR would be biased because value estimation is the only learning signal and as a result the representations are somewhat saturated which can lead to over-fitting.

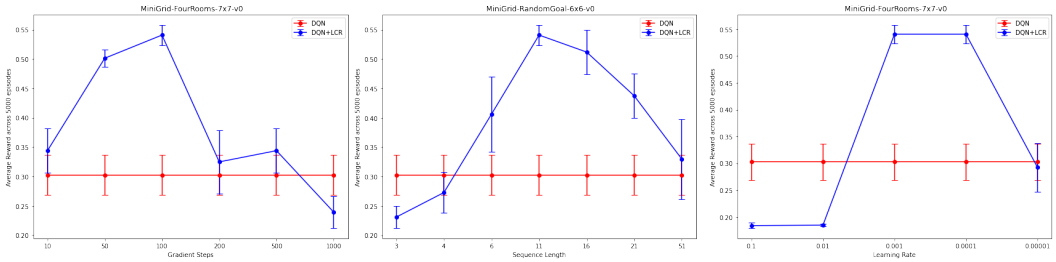
4.3 MUJOCO

For our second set of experiments, we take *Mujoco*, a very popular suite Todorov et al. (2012); Brockman et al. (2016) with well-defined physics, and observe the performance of LCR in easy and difficult tasks. For the experiments on Mujoco, we used Soft Actor Critic (SAC) Haarnoja et al. (2018) implementation of Ding et al. (2020), which is known to have good performance on these tasks. For these experiments, we used the default architecture from Ding et al. (2020) with 2 hidden layers followed by an output layer. The last hidden layer is used as the representation layer ϕ on which LCR was implemented. For LCR, we used the exact set of hyper-parameters that as the *MiniGrid* environment: a batch size B of 5000, $1/10^{th}$ of the SAC learning rate, sequence length of 11 and gradient steps for LCR optimization as 100.

Figure 4 shows the learning curves on *Mujoco*. For relatively difficult tasks, such as HalfCheetah-v2, Ant-v2, Humanoid-v2 and HumanoidStandup-v2, we can see that adding LCR on top of SAC dramatically improves performance, and it actually learns a lot faster. This is probably due to the fact that the dynamics of Mujoco is well defined, and, as such, LCR encourages the representations to consider them better as opposed to learning representations with just the policy and value losses in SAC. For the easier environments however, it is to be noted that adding LCR does not hurt performance at all, rather in some cases like Pusher-v2 and Striker-v2 it is marginally better than SAC.

4.4 ATARI

Although the *Arcade Learning Environment* does not have continuous dynamics like Mujoco, we tested Rainbow Hessel et al. (2017) with and without LCR in 4 Atari environments. For Rainbow, we used the original configuration of Hessel et al. (2017) with 3 convolutional layers followed by NoisyLinear layers Fortunato et al. (2017). The flattened output of the convolutional layer is chosen



(a) (b) (c)

Figure 6: Sensitivity to LCR hyper-parameters for the MiniGrid FourRooms environment over 10 runs. The constant hyper-parameters are sequence length of 11, 100 steps and learning rate of 0.0001.

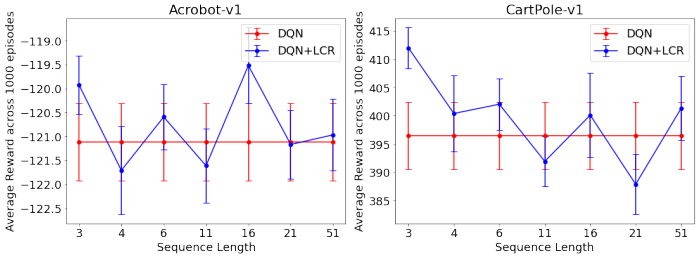


Figure 7: Results on CartPole and Acrobot, with the sensitivity to the sequence length to demonstrate how LCR affects low-dimensional states

as the representation layer ϕ for LCR. We use a batch size of 1000 for LCR, because the state space (images) is much more high dimensional, and to avoid overfitting, we reduced the gradient steps to 20 and the learning rate to $1/10^{th}$ of the original RL learning rate.

Figure 5 shows the learning curves of LCR compared with the baseline. Adding LCR significantly improves the performance and sample efficiency in 3 out of 4 environments. In *MsPacman* LCR does not show improvement over the baseline but also does not seem to hurt.

4.5 Ablation Studies

LCR introduces 4 new hyper-parameters: the size of the batches, B over which we apply LCR, the number of steps of gradient descent, the learning rate, and the neighbourhood size K.

Figure 6 (a) illustrates the parameter sensitivity on the *MiniGrid* domains w.r.t. the number of steps of gradient descent. In both environments, we find a similar plot. With very few steps, LCR does not work as well, and the same is true for higher number of steps, where the representations are constrained heavily to focus on the linear representations and thus collapses. Similarly, Figure 6 (b) demonstrates the sensitivity to the number of nearest neighbours. On the other hand, this depends on the environments. For example, in the *RandomGoal* environment, we found higher values of sequence length does equally well if not better, because the grid is empty and having representations to be linearly represented by the neighbours does not affect performance. However, with higher sequence length for *FourRooms*, LCR does not perform as well because the representations are forces to be representable from a large portion of the state space, which violates the neighbours being *local*. Thus, we see a drop in performance for higher K. Figure 6 (c) highlights the sensitivity to the learning rate. For most of our experiments, we found that too small of a learning rate does not help learning at all compared to DQN, because the updates to the representations are not big enough to have a sufficient effect on the actions taken. On the other hand, a high value of learning rate can be detrimental, as the representations become too constrained. For our experiments, setting the learning rate to be $1/10^{th}$ of the learning rate used for the main RL agent works well. The fourth hyper-parameter, batch size, has no such trade-offs. The higher the batch size, the less is the chance of over-fitting on the main task, and thus better the performance.

Now, it might seem like we need to run sweeps across all these hyper-parameters to find the best one, and that would be tedious. However, as mentioned in our experiments we found that normally setting sequence length to 11, number of steps to 100 and learning rate $1/10^{th}$ of the RL learning rate worked really well. Thus, as a thumb rule, we set these parameters for all our experiments.

Detailed ablation studies across all the hyper-parameters are provided in the supplementary material.

4.6 Adding LCR to low dimensional inputs

In very simple RL environments, separate representation learning is normally not required. Since we have the assumption of representations being linearly predictable by the neighbours, it might happen that the representations for simple tasks get constrained to the point that the agent does not learn anything. To test the performance of LCR with low-dimensional state space, we run LCR on two very simple environments, *CartPole* and *Acrobot* Brockman et al. (2016). For these environments we used simple DQN Mnih et al. (2015) with 2 hidden layers followed by the value layer. The output of the second hidden layer is the representation. From Figure 7, we can see that LCR does not impact the performance of the DQN over various sequence lengths. In fact, for some sequence lengths, DQN with LCR outperforms DQN.

5 Conclusion and Future Work

LCR adds an auxiliary loss to constrain the state representations in a local policy space. This can improve generalization and robustness of the representations learned. In any environment where the states do not change rapidly, with slow-moving features, LCR would show improvement. Furthermore, the addition of LCR does not hamper performance in simpler environments, where learning representations with the main RL loss is sufficient.

However, the neighboring representations may not be linearly dependent on each other, and we can introduce non-linear locally constrained representations as well. This will be a much softer constraint on the representations, as nonlinearity will make it much more flexible. Another interesting future direction could be its extension to vision based tasks with the help of a decoder, where the state representations can be reconstructed with the help of its neighbors.

References

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020. URL `https://arxiv.org/abs/2002.05709`.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Wesley Chung, Somjit Nath, Ajin Joseph, and Martha White. Two-timescale networks for nonlinear value function approximation. In *International Conference on Learning Representations*, 2018.

Zihan Ding, Tianyang Yu, Yanhua Huang, Hongming Zhang, Luo Mai, and Hao Dong. Rlzoo: A comprehensive and adaptive reinforcement learning library. *arXiv preprint arXiv:2009.08644*, 2020.

Norm Ferns, Prakash Panangaden, and Doina Precup. Bisimulation metrics for continuous markov decision processes. *SIAM Journal on Computing*, 40(6):1662–1714, 2011. doi: 10.1137/10080484X. URL `https://doi.org/10.1137/10080484X`.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. *CoRR*, abs/1706.10295, 2017. URL http://arxiv.org/abs/1706.10295.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *CoRR*, abs/1906.02736, 2019. URL http://arxiv.org/abs/1906.02736.

David Ha and Jürgen Schmidhuber. World models. *CoRR*, abs/1803.10122, 2018. URL http://arxiv.org/abs/1803.10122.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018. URL http://arxiv.org/abs/1801.01290.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *CoRR*, abs/1811.04551, 2018. URL http://arxiv.org/abs/1811.04551.

Danijar Hafner, Timothy P. Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *CoRR*, abs/1912.01603, 2019. URL http://arxiv.org/abs/1912.01603.

Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *CoRR*, abs/2010.02193, 2020. URL https://arxiv.org/abs/2010.02193.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647. URL https://www.science.org/doi/abs/10.1126/science.1127647.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016. URL http://arxiv.org/abs/1611.05397.

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39:407–428, 10 2015. doi: 10.1007/s10514-015-9459-7.

Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin A. Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *CoRR*, abs/1705.09805, 2017. URL http://arxiv.org/abs/1705.09805.

Varun Raj Kompella, Matthew D. Luciw, and Jürgen Schmidhuber. Incremental slow feature analysis: Adaptive and episodic learning from high-dimensional input streams. *CoRR*, abs/1112.2113, 2011. URL http://arxiv.org/abs/1112.2113.

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017. doi: 10.1017/S0140525X16001837.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *CoRR*, abs/2004.14990, 2020. URL https://arxiv.org/abs/2004.14990.

Kuang-Huei Lee, Ian Fischer, Anthony Z. Liu, Yijie Guo, Honglak Lee, John F. Canny, and Sergio Guadarrama. Predictive information accelerates learning in RL. *CoRR*, abs/2007.12401, 2020. URL https://arxiv.org/abs/2007.12401.

Longxin Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 2004.

Jan Mattner, Sascha Lange, and Martin Riedmiller. Learn to swing up and balance a real pole based on raw visual input data. In Tingwen Huang, Zhigang Zeng, Chuandong Li, and Chi Sing Leung (eds.), *Neural Information Processing*, pp. 126–133, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-34500-5.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000. doi: 10.1126/science.290.5500.2323. URL https://www.science.org/doi/abs/10.1126/science.290.5500.2323.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uCQfPZwRaUu.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. *CoRR*, abs/2004.04136, 2020. URL https://arxiv.org/abs/2004.04136.

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *CoRR*, abs/2009.08319, 2020. URL https://arxiv.org/abs/2009.08319.

Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '11, pp. 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0982657161.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748, 2018. URL http://arxiv.org/abs/1807.03748.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL http://www.jmlr.org/papers/v9/vandermaaten08a.html.

Herke van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3928–3934, 2016. doi: 10.1109/IROS.2016.7759578.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/a1afc58c6ca9540d057299ec3016d726-Paper.pdf.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. *CoRR*, abs/2102.11271, 2021a. URL https://arxiv.org/abs/2102.11271.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=GY6-6sTvGaf.