Università della Svizzera italiana

Faculty of Informatics

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

FACULTY OF ENGINEERING

Technical Report - Efficient Computational Algorithms, Fall 2020

Reproducibility challenge ATML SA 2021

Generalization in Reinforcement Learning

Erick Franciskus[†], Glejdis[‡], Constantin[‡], Jitin[‡]

[†] Faculty of Informatics, Università della Svizzera italiana, Switzerland

[‡] Faculty of Engineering, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Abstract

Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.

At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga. Et harum quidem rerum facilis est et expedita distinctio. Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus.

Report Info

Published
November 2021

Course
Efficient Computational Algorithms
Fall 2020

Institutions
Faculty of Informatics
Università della Svizzera italiana
Lugano, Switzerland
&
Faculty of Engineering
Friedrich-Alexander Universität
Erlangen-Nürnberg, Germany

1 Introduction

Reinforcement Learning is used in sequential decision making for training agents in complex tasks. The agent interacts with the environment which, based on the agent's action, provides the agent with some reward (positive or negative). The agent then uses this feedback to update its behaviour with the goal of being optimal. Although current RL agents have demonstrated great potential in a variety of activities, they face difficulties in transferring these agent's capabilities to new unseen tasks. This happens even when the tasks are semantically equivalent. In other words, existing reinforcement learning agents frequently learn policies that do not generalize well to environments different than those environments these agents are trained on.

For example, the paper considered a jumping task, where we have an agent that needs to jump over an obstacle. The agent learns from image observations. In order for the agent to not collide with the obstacle, the agent needs to precisely time the jump at a particular distance from the obstacle. The various tasks consist of shifting the floor height or changing the obstacle position or both. If we train deep RL agents on some of these tasks where we vary the obstacle positions, they perform poorly at previously unseen locations (i.e. struggle to jump over the obstacle without hitting it). The challenge tackled in the paper is to generalize to unseen positions of the obstacle and floor heights in the test tasks, while using only a finite number of training tasks or environments sampled from a distribution of tasks.

A few proposed solutions to poor generalization, adapted from supervised learning, include regularization (such as l2-regularization, dropout or noise injection), Domain Randomization and Data Augmentation (such as RandConv, RAD, DrQ), which is being used more recently. The majority of these ideas centre

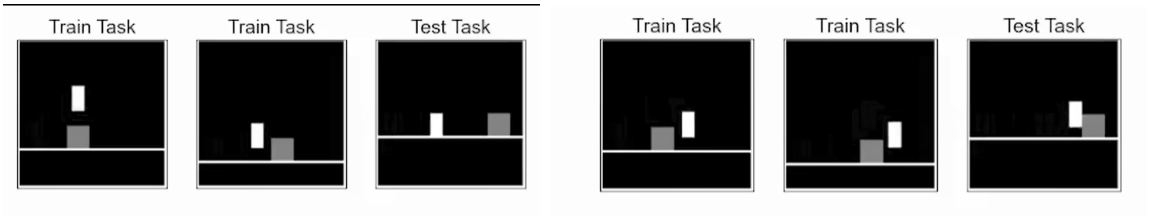


Figure 1: Train and Test task where agent fails

around improving the learning process in order to fully harness decision-making process, and they rarely use properties of the sequential aspect such as similarity in actions across temporal observations. Instead, this paper takes advantage of the fact that when an agent operates in similar tasks, the agent shows at least short sequences of behaviours that are similar across these tasks.

In their approach, the authors train the agents to learn a representation where the states are close when the optimal behaviour of the agent in the current states and future states are similar. They use the notion of behavioural similarity as it has the property of good generalization to observations across different tasks. This behavioural similarity between states across various tasks is quantified using the Policy Similarity Metric (PSM), which is a state-similarity metric based on bisimulation. Moreover, to enhance generalization, the agent is trained to learn state embeddings. This is done using Contrastive Similarity Metric (CSM).

2 Theory and Concepts

2.1 The π-bisimulation and Policy Similarity Metric

In the paper, the authors learn representations that encode behavioural similarity across states. To have a better understanding of the behavioural similarity, think of an agent who needs to reach spinach at the supermarket while maintaining social distancing of 6 feet. In the first environment to the left, the agent takes two actions to the right followed by two actions down. Now we look at the second environment on the right with a slightly different layout but where the agent takes the same actions (two steps right and two steps down) (as seen in figure 2). This is behavioural similarity where actions in the current state as well as in the future states are similar.

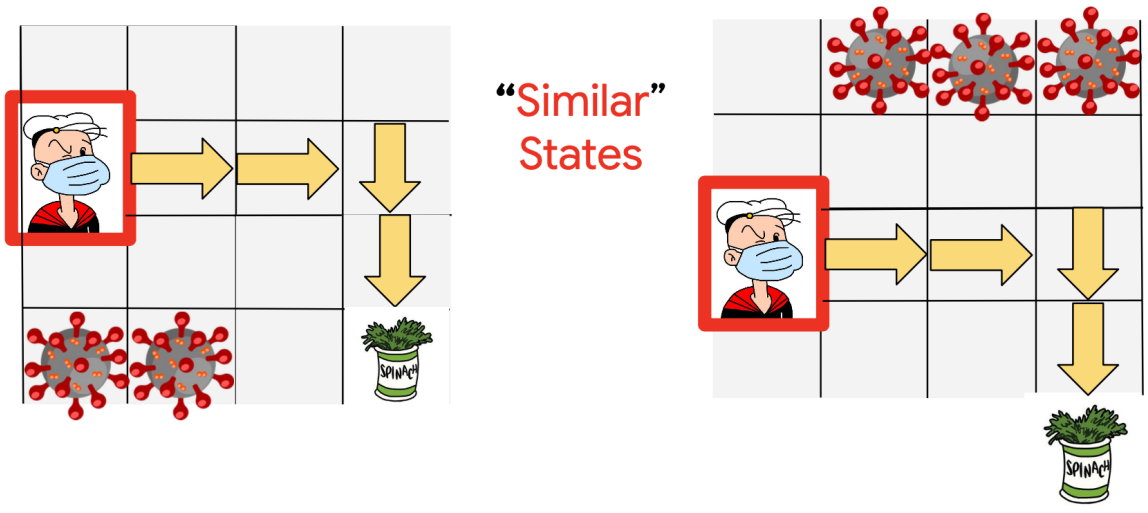


Figure 2: Behavioral similarity

To mathematically define the behavioural similarity, we use the concepts of bisimulation metric. Bisimulation metrics quantify the behavioural distance between two states in a Markov decision process. To mathematically define the bisimulation metric, we need some notations.

The Markov Decision Process (MDP) is defined as a 5-tuple M = (S, A, R, P, γ), where:

- S denoted the set of all possible state.
- A denotes set of all possible actions.
- R denotes a reward function.
  - $R(x, a, x')$ is the immediate reward received after a transition from state x to state x' because of action a
- P denotes the state transition probabilities.
  - $P(x, a, x')$ is the probability that action a in state x at time t will lead to a state x' at $t+1$.
- $\gamma \in [0, 1)$ is the discount factor which is used to generate a discounted reward.

Each state x encodes sufficient information about the environment such that an agent can learn how to behave in a consistent manner. Policy π is a controller that helps us select the actions to maximize expected return $\mathbb{E}_{a_t \sim \pi(\cdot|x)}[\sum_t \gamma^t R(x_t, a_t)]$. A policy $\pi(\hat{A}\cdot|x)$ maps states $x \in X$ to distributions over actions.

In RL, the goal is to find an optimal policy π^* that maximizes the cumulative expected return starting from an initial state x_0. In this paper, the goal is to learn a policy that generalizes well across related environments.

An **equivalence relation**, $E \subset S \times S$, is a bisimulation relation if wherever 2 states, $(s, t) \in E$, are considered bisimilar (denoted $s \sim t$) if the following properties hold:

1. They have equal immediate rewards:
   - $\forall a \in A, R(s, a) = R(t, a)$
2. Transition with equal probability to bisimulation equivalence classes:
   - $\forall a \in A, \forall c \in S_E, P(s, a)(c) = \sum_{s' \in c} P(s, a)(s') = P(t, a)(c)$

In other words, two states are bisimilar if they have similar expected rewards and dynamics. In the illustration (figure 3), we look at the system where a bisimualtion equivalence relation would collapse the 8 states on the left into an equivalent 4 state MDP to the right.

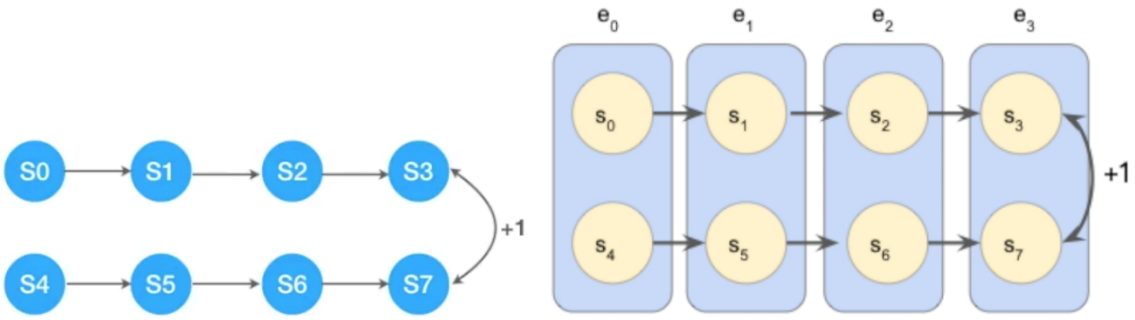


Figure 3: Bisimulation equivalence relation

Getting the exact equivalence would be quite tricky in every case, so we try to find a metric yielding a smoother notion of similarity than equivalence relation. The bisimulation metrics are a generalization of bisimualtion relations. Let M be all pseudo-metrics of S, ie, all metrics $d \in M$ where $\forall (s, t) \in S, d(s, t) = 0$. A fixed point $d_\sim$, and this fixed point is a bisimulation metric:

$$F(d)(s, a) = max_{a \in A}[|R(s, a) - R(t, a)| + \gamma \mathcal{W}_1(d)(P(s, a), P(t, a))] \tag{1}$$

where

- $s, t \in S$ are two states in the MDP.
- A is the action space.
- $R : S \times A \rightarrow R$ is the reward function.
- $P : S \times A \rightarrow \Delta(S)$ is the transition function.
- $\mathcal{W}_1(d)$ is the Wasserstein distance between two probability distributions under a state metric d.

A very nice theoretical property of such metric is that the bisimulation distance between two states is an upper-bound on their optimal value difference: $|V^*(s) - V^*(t)| \leq d \sim (s, t)$.

$$d_\pi(x, y) = |R^\pi(x) - R^\pi(x)| + \gamma \mathcal{W}_1(d_\pi)(P^\pi(\cdot|x), (P^\pi(\cdot|y)) \tag{2}$$

As it can be seen from equation 2, the bisimulation metric is a recursive equation with two terms. The first term calculates the reward difference, while the second term captures the long-term discounted future reward difference. Moreover, it defines the distance between two states x and y as the distance between expected rewards obtained when following policy π. The second term uses the Wasserstein-1 distance of the metric d_π. It quantifies the distance for the next state distributions following the policy π at states x and y. A very easy illustration (4) is to think of probability distributions as piles of sand. You can look at the sand pile as a probability mass. Then, the Wasserstein-1 distance calculates the dissimilarities between these piles of sand in terms of how much and how far we need to move the sand in order to transfer one pile into the other.

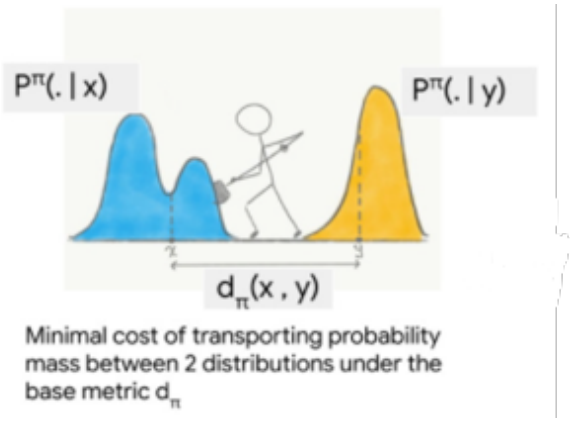


Figure 4: Sand Pile analogy of Wasserstein metric

There are a few drawbacks of using reward difference in bisimulation for measuring state similarities:

1. Two MDPs can have the same policies/behaviours but different expected rewards. For example, consider the two MDPs shown in figure 5. Once the agent has taken the action to eat cake, the agent should stop, because eating cake is good, but eating too much cake is bad. The rewards are different in these two MDPs. For certain values of these rewards, the bisimulation distance captures correct state similarity.

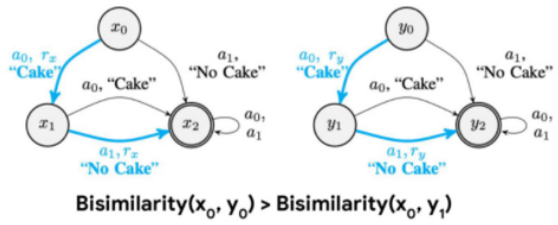


Figure 5: Same policies different rewards

2. Two states can have the same long term expected reward but different optimal policies. For instance, figure 6 illustrates three jumping tasks with different positions of the obstacles. The three states with a yellow boundary have the same expected rewards, which means that the bisimulation value is 0. However, as illustrated with the little arrow, the optimal action in the first task in the highlighted is to jump, while in the other tasks the optimal action is to go right.

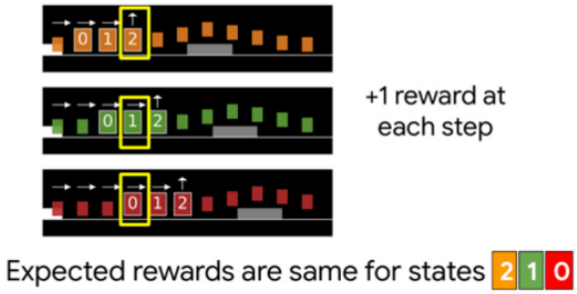


Figure 6: Same reward different policies

3. Pessimistic: the maximum is considering the worst possible case

4. Computational expense: Bisimulation metrics have been solved using dynamic programming, but as this requires updating the metric estimate for all state-action pairs at every iteration, it is computationally expensive. This means computing the Wasserstein distance $|S| \times |A|$ times at each iteration.

5. Full state enumerability: Existing methods for computing/approximating bisimulation metrics require full state enumerability (countability).

Therefore, to address these issues, in this paper, the π-bisimulation metric, which is based on the bisimulation metric, is being used. In contrast to bisimulation metrics, which is built on reward differences, π-bisimulation metrics are built on differences in optimal policies. The absolute reward differences are replaced by policy differences. Furthermore, as the goal is to perform well in previously unseen environments, we are interested in optimal behaviour, and so, we use π^* as the grounding policy. This yields Policy Similarity Metric (PSM).

$$d_\pi(x,y) = DIST(\pi^*(x),\pi^*(y)) + \gamma W_1(d^*)(P^{\pi^*}(\cdot|x),(P^{\pi^*}(\cdot|y)) \tag{3}$$

In PSM, states are considered similar when the optimal policies in these states as well as future states are similar. The DIST term captures the local optimal behaviour differences, while the Wasserstein term captures the long-term optimal behaviour differences. Moreover, the discount factor γ assigns the weights.

Now, take into consideration transferring an optimal policy from one to another environment using PSM. We do this by finding the nearest neighbour x for each state y. This nearest neighbour has the minimum PSM distance to y. To define the transfer policy at y, the policy at x can be used. If we use this nearest neighbour transfer scheme, we have the advantage of an upper bound on the suboptimality of policies or optimal policies transferred from one environment to another by PSM, in contrast to bisimulation metric where we do not have such feature. This illustrates how PSM provides a way of lifting generalization across inputs as a supervised learning problem to generalization across environments. As we aim to achieve a good generalization, we learn policy similarity embeddings that encode this PSM. This is done by adapting SimCLR, which is a constructive method for learning embeddings of image inputs.

2.2 Policy Similarity Embeddings - PSEs

Aiming for generalization of learned policies to related environments, the authors represent states in an embedding space, where closeness of states correlates with similar optimal policies. The embedding is built by self-supervised contrastive learning. Self-supervised learning refers to learning techniques that work without explicit annotations of the data. [5] Instead, feedback signals are generated from the data itself. In this case this feedback comes from the policy similarity metric. The contrastive instance of self-supervised learning works with positive pairs that are similar to each other, and negative or dissimilars pairs. [3] The

policy similarity metric d (introduced in Section 2.1) is transformed to the similarity measure Γ using the Gaussian kernel:

$$\Gamma(x, y) = \exp(-d(x, y)/\beta) \tag{4}$$

For two state spaces $\chi' \subseteq \chi$ and Y resulting from MDPs the positive pair $(\tilde{x}_y, y)$ is chosen as the one with highest similarity. Hence, $\tilde{x}_y = \text{argmax}_{x \in \chi'} \Gamma(x, y)$, $y \in Y$.

All other states are grouped to negative pairs. Eventually, we aim to map unknown states into this embedding to infer successful policies. Therefore, a function is needed that maps states into the embedding. The contrastive loss to learn this mapping is inspired from SimCLR, which also uses self supervised learning and heavy data augmentation to create a representation space of images that can be used for classification. [2]. The contrastive loss is given by:

$$l_\theta(\tilde{x}_y, y; \chi') = -\log \frac{\Gamma(\tilde{x_y}, y)\exp(\lambda s_\theta(\tilde{x}_y, y))}{\Gamma(\tilde{x}_y, y)\exp(\lambda s_\theta(\tilde{x}_y, y)) + \sum_{x' \in \chi' \setminus \tilde{x}_y}(1 - \Gamma(x', y))\exp(\lambda s_\theta(x', y))} \tag{5}$$

This mapping function z_ϕ is composed of an encoding f_ϕ leading to the latent representation of the state and a non-linear projection h_ϕ, following SimCLR. The contrastive loss (Equation 5) is applied to the projection to train the network. The inference of a suitable policy is done in the representation space as an affine function $\pi_\theta(\cdot \mid y) = W^T f_y + b$ with learnable parameters W and b. The training procedure is depicted in the Figure 7.

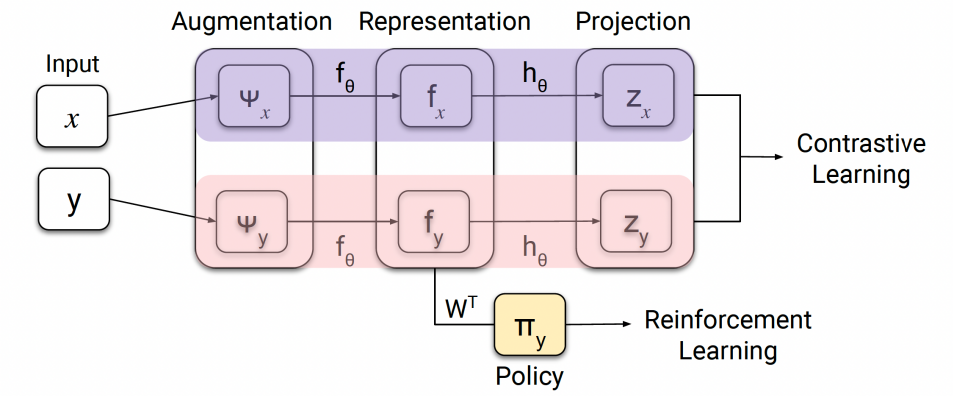


Figure 7: An input pair (x,y) is first augmented and then mapped into the representation space f. The loss function defined in Equation 5 is applied to a non-linear projection z of this representation space. [1]

The contrastive metric embeddings learnt with policy metrics are referred to as policy similarity metrics. They are learnt simultaneously during training, where an auxiliary objective derived from the contrastive loss is introduced. The total contrastive loss for the MDPs $\mathcal{M}_\chi$ and $\mathcal{M}_y$ are derived from the optimal trajectories $\tau^*_\chi = \{x_t\}_{t=1}^N$ and $\tau^*_y = \{y_t\}_{t=1}^N$. x_t and y_t are the single states in the optimal trajectories. x_{t+1} is the result of applying the optimal policy π^* to transition dynamics P of the system : $x_{t+1}\ P_\chi^{\tau^*}(\cdot \mid x_t)$ and similar $y_{t+1}\ P_y^{\tau^*}(\cdot \mid y_t)$. The authors set $\chi' = \tau^*_\chi$ (compare to Equation 5) and then define the total contrastive loss as:

$$L_\theta(\mathcal{M}_\chi, \mathcal{M}_y) = \mathbb{E}_{y \sim \tau^*_y}[l_\theta(\tilde{x}_y, y; \tau^*_\chi)]. \tag{6}$$

$\tilde{x}$ is chosen as the most similar state to the given state y: $\tilde{x} = \text{argmax}_{x \in \tau^*_\chi} \Gamma(x, y)$. Then the loss between the Markov processes $\mathcal{M}_\chi$ and $\mathcal{M}_y$ are hence the expected value for y from the optimal trajectory of the contrastive loss (Equation 5) of $\tilde{x}$ and y w.r.t. to the optimal trajectory of x. The loss becomes low, if we can conclude states $\tilde{x}$ that are close w.r.t. its optimal trajectory from the similarity, in the representation space, to y.

3 Experiments from the paper

In this section we want to introduce the paper's main experiment: the jumping task from pixels. The authors also studied generalization on the dm control suite where agents have to ignore visual distraction. However,

we want to focus our reproducibility study on the jumping task.

3.1 Jumping Task from Pixels: A Case Study

In the jumping task, we have an agent that, learning from the image observations on pixel level, needs to jump over an obstacle. The agent can either go right or jump. To avoid a collision the agent needs to time the jump precisely. The environment can be adapted by changing the floor height and the position of the obstacle. Two possible environments and their optimal trajectory are shown in Figure 8. Generalization is reached, when the agent becomes invariant to such changes. There are 26 different positions of the obstacle and 11 different values of the floor height. To stress generalization, the problem is split into 18 training tasks and the remaining 268 tasks are evaluation tasks.

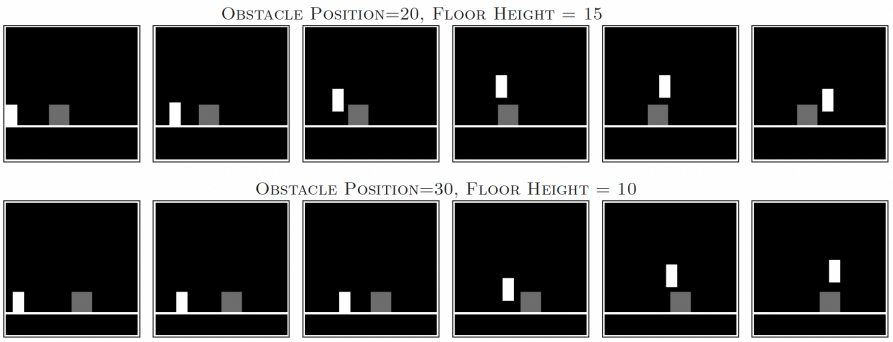


Figure 8: Optimal trajectories are shown for two different environments. The trajectory is a sequence of right actions interrupted by a single jump action. [1]

3.2 Classification of the problem

The states of the environment are discrete. This becomes apparent as we are able to classify all possible problems in a grid as combinations of obstacle position and floor height in Figure 10. The actions of the agent are discrete as well: go right or jump. The problem is deterministic because the optimal trajectory can be concluded with certainty. In particular, by processing the fully observable input: all pixel values are available to the agent. Furthermore, the problem is stationary because the optimal trajectory for a given task is time-invariant. As the agent is the only active participant in the environment the problem can be considered a single agent task.
So the task actually falls into the "easiest" category of tasks for reinforcement learning. However, the aim of the authors is to study and improve generalization in RL. And the notion of generalization is very clear in this task and also measurable. Therefore, we think the task is very well chosen to study and show the ability of PSEs to improve generalization in RL.

3.3 Results from the paper

In the paper, three different grid configurations, that represent different styles of generalization are used:

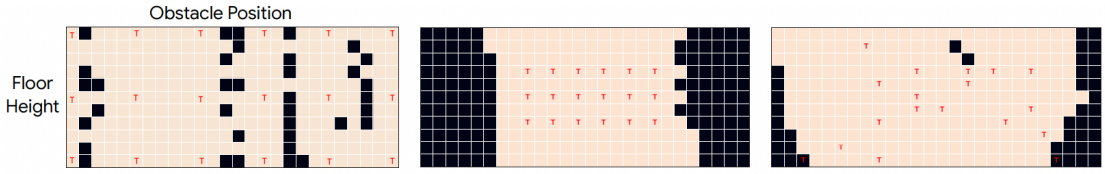
- The wide grid tests generalization via interpolation
- The narrow grid tests out-of-distribution generalization via extrapolation
- The random grid evaluates generalization like supervised learning where the training and test samples are drawn i.i.d. (independently and identically distributed)

All configurations are evaluated using the same hyperparameters that were tuned on the "wide" grid to show the robustness of PSEs to hyperparameter tuning. The authors compare the PSE to other regularization methods. Also the effect of data augmentation using RandConv is studied. RandConv is a recently proposed data augmentation method to improve generalization in RL. It uses a randomized convolutional neural network to perturb input observations randomly. [4]. The average performance across 100 runs is reported in Figure 9.

Data Augmentation	Method	Grid Configuration (%)		
		"Wide"	"Narrow"	Random
✗	Dropout and ℓ_2 reg.	17.8 (2.2)	10.2 (4.6)	9.3 (5.4)
	Bisimulation Transfer[4]	17.9 (0.0)	**17.9** (0.0)	30.9 (4.2)
	PSEs	**33.6** (10.0)	9.3 (5.3)	**37.7** (10.4)
✓	RandConv	50.7 (24.2)	33.7 (11.8)	71.3 (15.6)
	RandConv + π^*-bisimulation	41.4 (17.6)	17.4 (6.7)	33.4 (15.6)
	RandConv + PSEs	**87.0** (10.1)	**52.4** (5.8)	**83.4** (10.1)

Figure 9: The table summarizes the main results of the authors, giving percentages of how many test tasks could be solved using different methods with or without data augmentation. The grid configurations are shown in Figure 10. It can be observed that PSEs outperform the compared methods in most configurations except for the narrow grid without data augmentation. Also, PSEs profit substantially from data augmentation. [1]

PSEs outperform all other methods except for the case without data augmentation on the narrow grid. It is also apparent that PSEs work particularly well with data augmentation. The different grid types as well as the performance of PSEs + data augmentation are shown in Figure 10.



(a) "Wide" grid (b) "Narrow" grid (c) Random grid

Figure 10: The red letter T shows which are the training tasks for the three grid configurations and the background colour of each tile shows the performance of PSEs accross the task configurations, which is reported as the median of 100 runs per configurations using data augmentation. Beige tiles correspond to tasks that could be solved and black tiles could not be solved. [1]

References

[1] Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning, 2021.

[2] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

[3] Raia Hadsell, Sumit Chopra, and Yann Lecun. Dimensionality reduction by learning an invariant mapping. In *In Proc. Computer Vision and Pattern Recognition Conference (CVPR'06)*. IEEE Press, 2006.

[4] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning, 2020.

[5] Yuandong Tian, Xinlei Chen, and Surya Ganguli. Understanding self-supervised learning dynamics without contrastive pairs, 2021.