Università della Svizzera italiana | Faculty of Informatics

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
FACULTY OF ENGINEERING

Advanced Topics In Machine Learning, Fall 2021

Reproducibility challenge ATML SA 2021
Generalization in Reinforcement Learning

Erick Franciskus[‡], Glejdis Shkëmbi[‡], Constantin Jehn[‡], Jitin Jami[‡]

[†] Faculty of Informatics, Università della Svizzera italiana, Switzerland
[‡] Faculty of Engineering, Friedrich-Alexander Universität Erlangen-Nürnberg, Germany

Abstract

Reinforcement learning (RL) is an important branch of artificial intelligence and is used in a variety of applications such as robotics, self-driving cars or solving complex board games like Go. In contrast to supervised learning techniques, where evaluation is exclusively done on unknown data, classic RL algorithms are evaluated on the same environment it was trained on and generalization in RL remains an active branch of research. We present a reproducibility study of a recent approach that was published by the Google Research Brain team under the title "Constrastive Behavioral Similarity Embeddings for Generalization in RL". Our study includes a placement of the proposed approach in the field of generalization in RL, as well as a research on incorporated concepts. Furthermore, we present the main benchmark environment and reproduce the main result from the paper. We also study the model with respect to its sensitivity to hyperparameters and discuss other benchmark environments.

Report Info

Published
December 2021

Course
Advanced Topics In Machine Learning
Fall 2021

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 considers 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

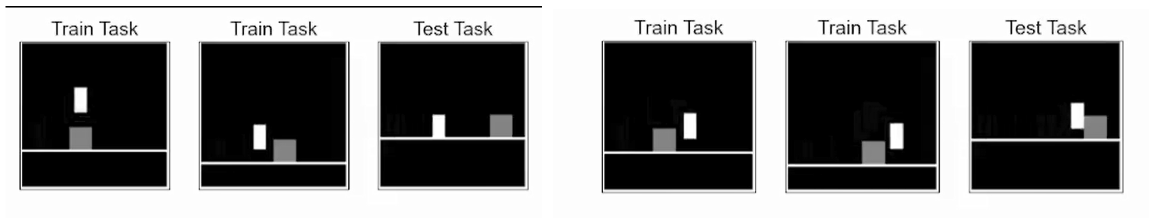


Figure 1: Train and Test task where agent fails

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 Generalization in RL

In supervised learning, training and test dataset are always held disjoint of each other. And the performance of a predictor is generally evaluated on unseen test data. In many RL benchmarks, like Atari, however the policy is evaluated on the same environment it was trained on and until now RL agents struggle to perform well on unseen environments [8]. Overcoming this limitation and making RL applicable to more realistic task is an active field of research. Research on improving generalization in RL tries to produce "[...] algorithms whose policies have the desired robustness, transfer and adaptation properties, challenging the basic assumption that train and test will be identical" [7].
The type of generalization can be classified with respect to the distribution that training and test environments are drawn from: [7]

- In singleton environments train and test environment are identical; no generalization is involved.
- In i.i.d. generalization environments train and test environments are drawn from the same distribution: $P_{\text{train}}(C) = P_{\text{test}}(C)$.
- The most challenging form of generalization are o.o.d generalization environments. Training and test environments are drawn from distinct distributions: $P_{\text{train}}(C) \neq P_{\text{test}}(C)$.

Furthermore, [7] categorizes current research on generalization in RL. The overview is shown in Figure 2. The branch "increasing similarity" aims to modify or augment the training environment to be similar to the training set. "Handling differences" tries to address the difficulty that features will be different in the test environment. The paper we study tries to learn invariances: features that are not semantically important to the task should be discarded. The first two groups could also be applied to supervised learning. The last branch aims to increase generalization by improving the optimisation on the training environment, which is specific to RL.

2.2 State representation in RL

Besides generalization, sample-efficiency is a major challenge in RL. Learning successful policies from high-dimensional inputs like pixels of an image takes a huge amount of data. Therefore, learning lower dimensional latent representations of environments is an active area of research in RL. The agent then operates and learns policies in a latent space, which must be fully differentiable. [5]. A useful representation, in

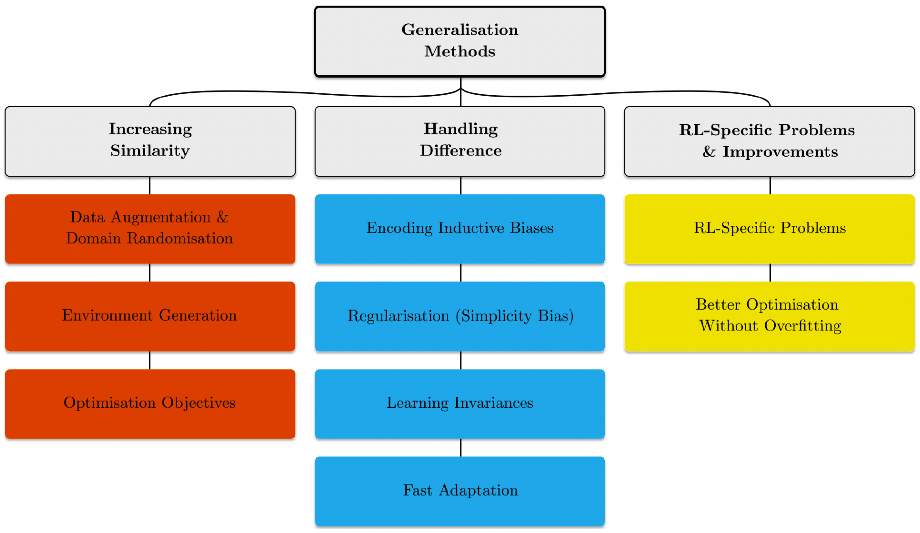

Generalisation Methods
Increasing Similarity
Handling Difference
RL-Specific Problems & Improvements
Data Augmentation & Domain Randomisation
Encoding Inductive Biases
RL-Specific Problems
Environment Generation
Regularisation (Simplicity Bias)
Better Optimisation Without Overfitting
Optimisation Objectives
Learning Invariances
Fast Adaptation

Figure 2: Categorization of current research on generalization in RL according to [7]. The studied paper [1] is an instance of learning invariances.

the context of RL, would encode only task-relevant information of the original space. There are well established dimensionality reduction methods like the Variational Auto-Encoder (VAE). However, such methods assume all information of the input space to be important, which is not true for RL environments. For instance, the task of jumping over block, depicted in Figure 9, is certainly invariant to the colour of the block. Yet, this information would be encoded using a VAE, hindering generalization to the semantically identical task with a block of different colour. Actively ignoring information that is irrelevant to the task is hence very desirable and can enhance generalization. [5] [7]

2.3 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 3). This is behavioural similarity where actions in the current state as well as in the future states are similar.

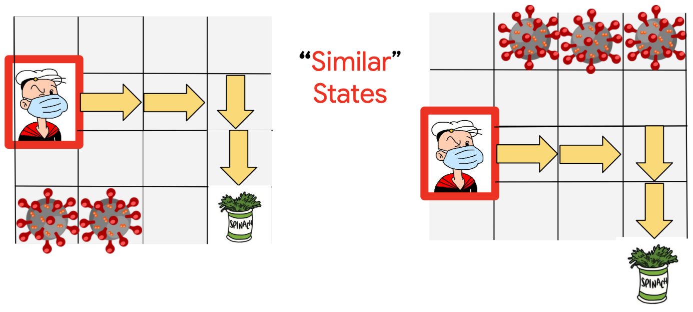


Figure 3: Understanding behavioral similarity. The agent needs to obtain the reward (spinach) while maintaining distance from the virus. Even though the initial states are visually different, they are similar in terms of their optimal behavior at current states as well as future states following the current state. Policy similarity metric (PSM) assigns high similarity to such behaviorally similar states and low similarity to dissimilar states.[10]

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. [9]

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 π(Â·|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. [1]

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:
   - $\forall a \in A, \forall c \in S_E, P(s, a)(c) = \sum_{s' \in c} P(s, a)(s') = P(t, a)(c)$ [2]

In other words, two states are bisimilar if they have similar expected rewards and dynamics. In the illustration (figure 4), 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.

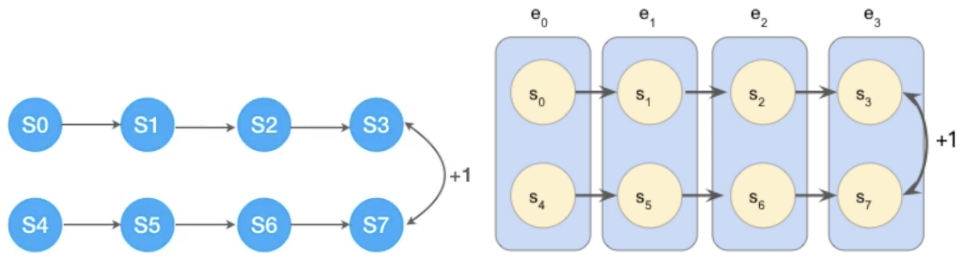


Figure 4: Bisimulation equivalence relation. The 8 states on the image on the left would collapse to an equivalent 4-states MDP [3]

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 $\forall (s, t) \in S, d(s, t) = 0$. Define $F : M \to M$ to have a unique 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.

[2]

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: $|V^*(s) - V^*(t)| \leq d\sim(s,t)$. [2]

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. [11]

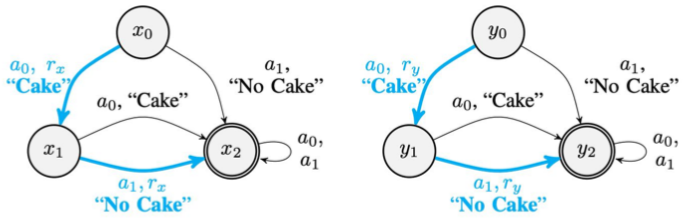


Figure 5: Same policies different rewards: Cyan edges represent optimal actions with a positive reward. The rest have zero reward. x_0, y_0 are the start states and x_2, y_2 denote the terminal states [1]

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. [11]

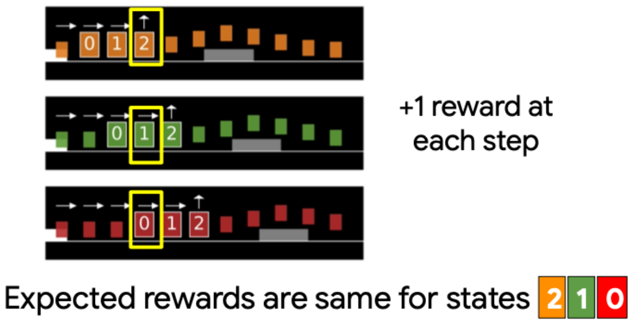


Figure 6: Same reward different policies: he three states with a yellow boundary have the same expected rewards, which means that the bisimulation value is 0. However, the exhibit different behaviour. [10]

3. Pessimistic: the maximum is considering the worst possible case. [2]
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. [2]

5. Full state enumerability: Existing methods for computing/approximating bisimulation metrics require full state enumerability (countability). [2]

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. they are called reward-agnostic. The pi-bisimulation metric is given by: [2]

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

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 (7) 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.

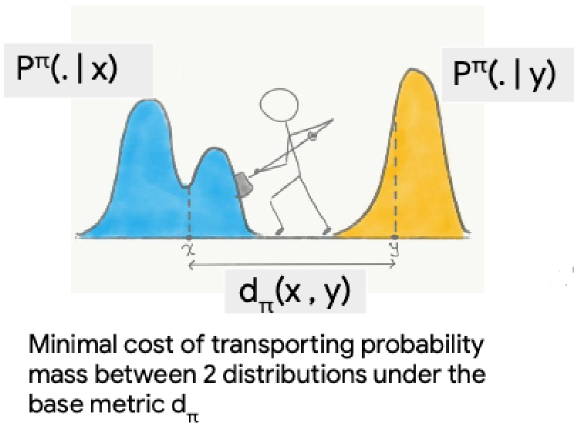

$P^\pi(.\,|\,x)$
$P^\pi(.\,|\,y)$
x
y
$d_\pi(x\,,y)$
Minimal cost of transporting probability mass between 2 distributions under the base metric d_π

Figure 7: Sand Pile analogy of Wasserstein metric [10]

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).

$$d_\pi(x,y) = DIST(\pi^*(x),\pi^*(y)) + \gamma \mathcal{W}_1(d^*)(P^{\pi^*}(\cdot|x),(P^{\pi^*}(\cdot|y)) \tag{3}$$

[2]

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. [2]

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. [2] 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.

2.4 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. [12] 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. [6] The policy similarity metric d (introduced in Section 2.3) 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, $\tilde{x}_y = \operatorname{argmax}_{x\in\chi'} \Gamma(x, y)$, $y \in Y$. 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 $z_\phi(x)$ 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. [4]. The contrastive loss is given by:

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

where s_θ is the embedding similarity. The loss functions aims for a representation where positive pairs $((\tilde{x}_y, y))$ have a high similarity in behaviour Γ and in the embedding s_θ at the same time, corresponding to high value in the term (1) $\Gamma(\tilde{x}_y, y)\exp(\lambda s_\theta(\tilde{x}_y, y))$. On the contrary negative pairs $(x' \in \chi' \setminus \{\tilde{x}_y\})$ should be forced to be far from each other in the embedding space. So, small values in term (2) $\sum_{x'\in\chi'\setminus\{\tilde{x}_y\}}(1-\Gamma(x', y))\exp(\lambda s_\theta(x', y))$ are favourable. Note that the whole loss is wrapped in a negative logarithm so small values of the argument lead to a high loss. The argument can take values in the range $(0, 1)$. High values in term (1) push the argument near 1 and leads to a "compensation" of values in term (2). Low values in term (2) bring the argument closer to 1 and therefore a minimal loss.
From here we can introduce the total contrastive loss for the MDPs $\mathcal{M}_\chi$ and $\mathcal{M}_y$. It uses 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 : $x_{t+1} \sim P_\chi^{\tau^*}(\cdot \mid x_t)$ and similar $y_{t+1} \sim P_y^{\tau^*}(\cdot \mid y_t)$. 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: $\tilde{x} = \operatorname{argmax}_{x\in\tau^*_\chi} \Gamma(x, y)$. 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.

Figure 8 gives an overview over the learning procedure as well as how inference is done. An input pair (x,y) is fed in to the architecture. First and optionally a data augmentation is applied: $\Psi_x := \Psi(x)$ and respectively $\Psi_y := \Psi(y)$. Secondly the encoder f_θ is applied, which is the mapping we are in particular interested to train, $f_x = f_\theta(\Psi_x)$ and $f_y = f_\theta(\Psi_y)$. The loss however is applied on a non-linear projection of the representation, $z_x = h_\theta(f_x)$ and $z_y = h_\theta(f_y)$, following the SimCLR approach [4]. The inference of a suitable policy is done in the representation space f_y as an affine function $\pi_\theta(\cdot \mid y) = W^T f_y + b$ with learnable parameters W and b. The encoding learnt simultaneously during training of the agent, with auxiliary loss of Equation 6.

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.

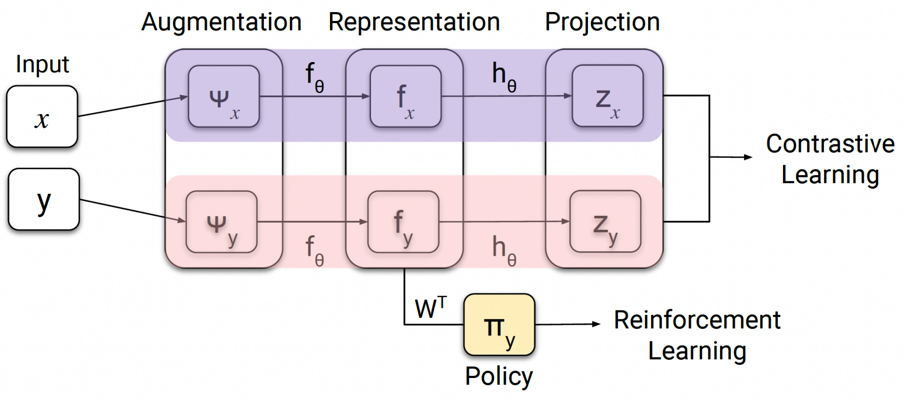


Figure 8: An input pair (x,y) is first augmented and then mapped into the representation space f. The loss function defined in Equation 6 is applied to a non-linear projection z of this representation space. [1]

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 9. 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.

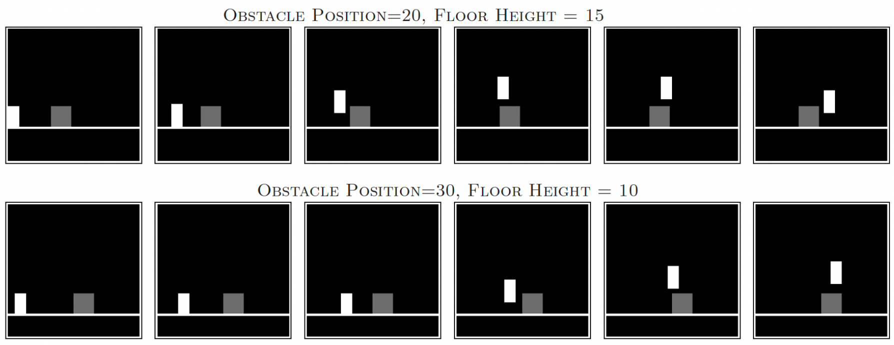

Obstacle Position=20, Floor Height = 15
Obstacle Position=30, Floor Height = 10

Figure 9: 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 11. 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:

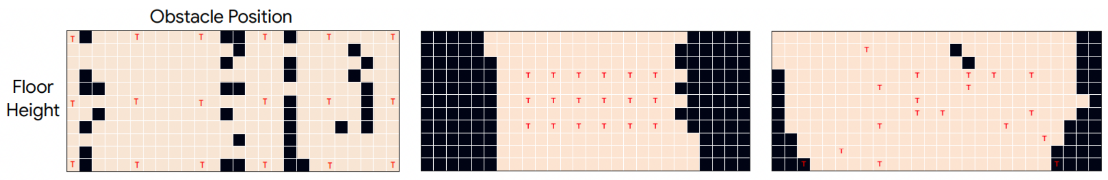
- The wide grid tests generalization via interpolation
- The narrow grid tests out-of-distribution generalization via extrapolation
- The random grid evaluates generalization like supervised learning where the training and test samples are drawn i.i.d. (independently and identically distributed)

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. It aims to confront the agent with visually divers observations and hence mitigating overfitting to specific fix features in the input space. [8]

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 10: 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 11. 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]

The average performance across 100 runs is reported in Figure 10. 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 11.



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

Figure 11: 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 across 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]

4 Results of reproducibility study

In this section, we elaborate on how we attempted to reproduce the results that were presented in the paper. The authors of this paper published their codes for both the jumping task from pixels task and the distracting control suite tasks with 6 different environments. We discuss in detail the reproducibility of these tasks and the corresponding results from these experiments.

4.1 Jumping Task from Pixels : Main result

In the jumping task code, the authors generate imitation learning dataset from a certain hardcoded ideal policy of the pixel jumping task game itself, whereby the action jump would be selected only when the agent state is at a certain horizontal distance away from the obstacle. Otherwise the ideal policy would always maintain a "go right" action for the agent. Referring to the scheme described by the authors in figure 8, the states of the agent from this ideal policy are the x inputs and the actions of agent from this ideal policy are the y inputs. These inputs are then augmented and projected before being fed into an adapted Nature DQN. During training with the training environment grid, the model would then learn ideal policies, which are iteratively evaluated against evaluation jumping task environments. These are done until a certain number of training epochs is achieved. The author set the default training epoch as 2000 epochs. The results for the different augmentation, embedding and loss function ablation configurations are then averaged over 100 runs as summarized in Figure 10.

The code itself is well structured and relatively reproducible. We encountered minor difficulty setting up the framework due to the discrepancies between the required packages versions as specified in their requirements.txt file and the machines where we run our experiments on. However, this did not significantly hinder us from running the code. The custom written gym based jumping task environment was also relatively simple to register and use. Changing the hyperparameters, training grid width and the different augmentation, embedding and loss function options is also very intuitive as one only needs to modify the command line option flags when running the main training script. For each run, the logs of various metrics such as number of tasks solved, loss and solved plots are saved within tensorboard logs. This also allows us to obtain the experiment results and visualize them easily.

When the script is run from our local PC CPUs, each training run of 2000 epochs takes in average about 2 hours to complete. The code provides GPU acceleration possibilities. When the script is run from a HPC cluster with an Intel Xeon CPU E5-26802 with four corse and two NVIDIA P100 GPUs, each training run of 2000 epochs takes in average about 50 minutes to complete. Therefore, this jumping task experiment itself may not be very easy to reproduce for those who do not have access to ample computing power.

Due to time and computing power allocation in the HPC cluster constraints, we reduce the number of training runs from 100 runs to 5 runs. The hyperparameters, training grid width settings and ablation parameters are set according to the instruction given by the author in the readme of the Git repository. The results of the jumping task study with different training grid width and ablation parameters are summarized in the following table.

Additionally, for visualization purposes, we have also obtained the training log plots and the training grid visualizations that we can compare with the figures found within the paper.

Talk about our result vs publication.

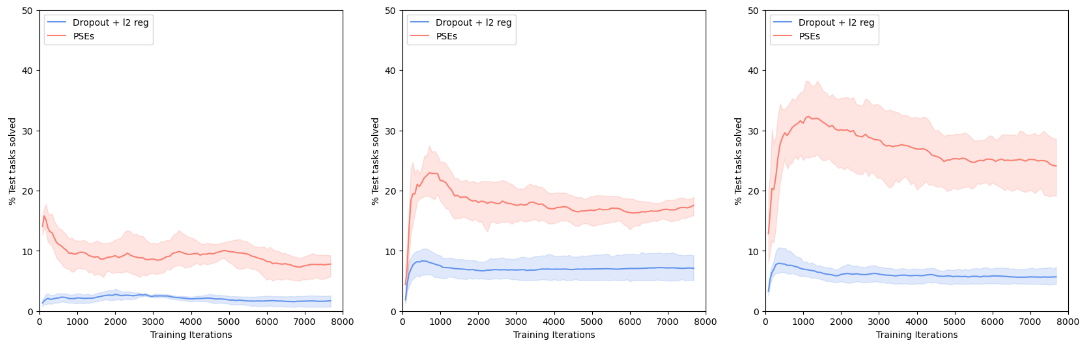


Figure 12: Test performance curves in the setting **without data augmentation** on the wide, narrow, and random grids. We plot the median performance across 5 runs. Shaded regions show 25 and 75 percentiles

4.2 Jumping Task from Pixels : Hyperparameter Sensitivity tests

The default hyperparameters used by the author for the jumping taks pixels are summarized in the following table.

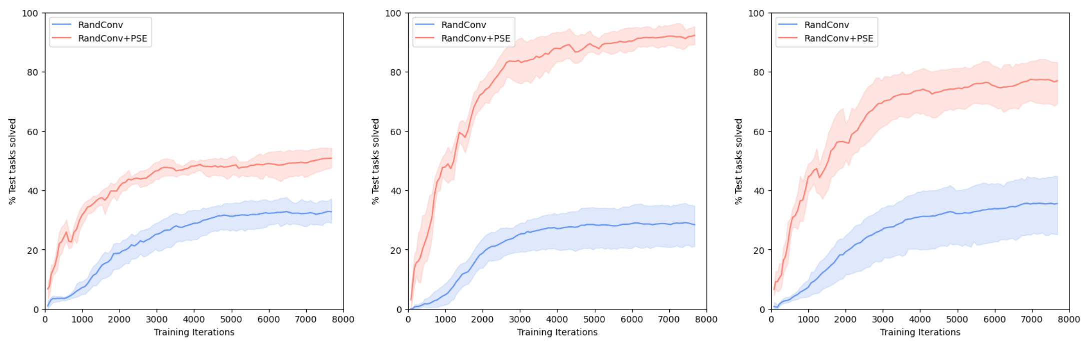


Figure 13: Test performance curves in the setting **with data augmentation** on the wide, narrow, and random grids. We plot the median performance across 5 runs. Shaded regions show 25 and 75 percentiles

Table 1: Percentage (%) of test tasks solved by different methods without and with data augmentation. The wide, narrow, and random grids. We report average performance across 5 runs with different random initializations, with standard deviation between parentheses

Data Augmentation	Method	Grid Configuration (%)		
		Wide	Narrow	Random
✗	Dropout and l_2 reg	6.86(2.27)	1.86(1.05)	5.82(1.64)
	Bisimulation Transfer	17.91(0.23)	**14.55**(1.33)	14.77(3.96)
	PSEs	**18.58**(2.35)	7.91(1.25)	**23.65**(5.95)
✓	RandConv	27.38(7.95)	32.23(5.04)	36.26(10.43)
	RandConv + π^*-bisimulation	18.35(0.72)	28.58(14.13)	29.62(13.58)
	RandConv + PSEs	**93.58**(4.01)	**50.97**(3.05)	**78.20**(8.08)

These hyperparameters were obtained from validation experiments done by the author with wide grid training grids, random convolutions and PSE loss, whereby the chosen values result to the best possible performance of the agent in that specific setting. We are interested in investigating the sensitivity of four of the hyperparameters used, namely learning rate, temperature, soft-coupling temperature, and α. We set a 5% decrease and increase to each one of these four hyperparameter values while keeping the remaining hyperparameters as the default chosen values. We then compare the performance, which in this case is the number of tasks solved, with the default wide grid result obtained from the previous subsection. Due to time and computing power allocation in the HPC cluster constraints, each hyperparameter sensitivity test is done in 3 runs. The overall sumamry of the effect of each hyperparameter to the performance of the agent is summarized in the following table.

4.3 DM Control Tasks

The author also published their code for DM Control Tasks, whereby the generalization ability of PSE are tested with 6 different environments with continuous states and actions, namely Ball in Cup Catch, Cartpole+Swingup, Cheetah-Run, Finger-Spin, Reacher-Easy, and Walker-Walk. The model are trained in environments with 2 different video distractions obtained from DAVIS dataset playing in the background and then evaluated against environments with 30 other video distractions from DAVIS dataset in the background. The author utilizes Soft Actor Critic Network with clipped double-Q network for the actor network. The loss term combines four loss terms, namely actor loss, critic loss, temperature loss, and PSE loss. Optimal policies are approximated by policies from each environment obtained after 500000 iteration steps.

Reproducing the DM Control Task was significantly more intricate compared to the jumping task. First of all, the video distraction used in this test requires both distraction_control and dm_control infrastructure packages, which were both specially developed by Deepmind. These 2 packages requires an additional UC Davis images and videos database folder which needs to be separately downloaded into the local machine. Additionally, these packages also require an installation of MuJoco, which is the Deepmind multiphysics simulation infrastructure. While the installation steps of these frameworks are specified within the repositories of each of these packages, there were still multiple separate components which needed to be installed and configured separately. Furthermore, some of the requirements.txt given in the repositories point to deprecated versions of Python packages, which also result to conflicts and errors while attempting to set up this environment. While these conflicts could still be solved, this still significantly hinders the setting up process of this task.

Secondly, the soft actor critic network requires an action replay buffer. In the code written by the authors, they utilize tf-agents with an experimental Tensorflow specific action replay buffer developed by Deepmind called reverb. This was not possible to install in our local machines as this reverb action replay buffer framework is currently only supported in Linux OS and not in Windows or Mac OS, as indicated in the repository of the reverb package. This is not specified at all in the dm control tasks code repository. Therefore, it was not possible for us to reproduce the dm control tasks at all.

4.4 Different Environment Tests

It would also be interesting to test the generalization ability of PSE with different environments. While this was already done within the DM control tasks with 6 different environments, it is currently not reproducible due to the aforementioned issues. However, one of the biggest issue of testing a different environment is obtaining the optimal policy for the imitation learning process. For the jumping task problem, the optimal policy was easily hardcoded due to the sheer simplicity of the jumping task itself. However, such an optimal policy cannot be easily obtained in many different common RL test environments. While this could also be approximated with policies obtained from a very high number of iterations as obtained in the DM Control Tasks, it may also be unrealistic to approximate policies as such with limited computing resources and time. As such, we come to the conclusion that testing PSE with different environments and setup may not be feasible at the moment.

4.5 Overall Remarks Regarding Reproducibility

4.6 Workload Partition among Group Members

Constantin and Glejdis worked mostly on the theoretical part of the report, by providing a thorough examination on the methods used and the mathematical explanations behind it, and provided the results from the original paper. While Glejdis focused more on behavioural similarity, the π-bisimualtion and Policy Similarity Metric, Constantin analysed Generalization and State Representation in RL, and Policy Similairty Embeddings.

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] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic markov decision processes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[3] Pablo Samuel Castro. Scalable methods for computing state similarity in deterministic mdps. https://psc-g.github.io/posts/research/rl/scalable/, 2021.

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

[5] Huy Ha, Sian Lee Kitt, and William Zheng. Deep bisimulation dreaming: Combating distractions with state abstractions.

[6] 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.

[7] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

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

[9] M Puterman. Markov decision processes: Discrete stochastic dynamic programming. *Wiley Series in Probability and Statistics*, 2005.

[10] Marlos C. Machado Rishabh Agarwal. Contrastive behavioural similarity embeddings for generalization in reinforcement learning. https://agarwl.github.io/pse/pdfs/slides.pdf, 2021.

[11] Marlos C. Machado Rishabh Agarwal. Contrastive behavioural similarity embeddings for generalization in reinforcement learning. https://slideslive.com/38942373/contrastive-behavioral-similarity-embeddings-for-generalization-in-reinforcement-learning, 2021.

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