# Enhancing the Robustness of Learning Directly from the Pixel

Pascal Schmidt
*Karlsruher Insititut für Technologie (KIT)*
Karlsruhe, Germany
pascal.schmidt3@student.kit.edu

Mohammd Karam Daaboul
*Institute of Applied Informatics and Formal Description Methods (AIFB)*
*Karlsruher Institute of Technology (KIT)*
Karlsruhe, Germany
mohammd.daaboul@kit.edu

***Abstract*—Reinforcement Learning (RL) is an important subfield of machine learning that is especially widely used in the field of robotics. Because pixel-based RL can be extremely sample-inefficient, meaning it takes a huge amount of training data and time, methods have to be found to circumvent that problem. Another big problem of RL is the ability to generalize to new environments, somthing that is imperative to real-world robotics applications. In the recent past, many papers and approaches have been published, all trying to solve these problems. This paper provides an overview over a couple of relevant publications, describing the algorithms and classifying them into different categories based on their approach.**

## I. Introduction

Reinforcement learning (RL) closely resembles how animals or humans learn and is - as all other machine learning algorithms - increasingly used in many applications. Logically, one of those applications is the transfer of learning like in the nature to the domain of robotics. RL is based on the concept of letting an agent learn via interaction with its environment and observing the consequences and the reward of that action. Ideally, a policy is learned that can find the optimal action (that maximizes the cumulative expected reward) for each state an agent is in. However, that approach is limited when the action space is continuous, as is often the case in robotics due to many different possibilities for joint combinations. To circumvent that problem, deep reinforcement learning (DRL) methods like Soft-Actor-Critic [1] are used to merely approximate the optimal policy [2] of an agent instead of fully computing it.

One big problem these algorithms face is **sample-inefficiency**. Neural networks - which are the basis of DRL approximation - traditionally need a lot of training data. While this problem is already present when using exact joint angles and positions as inputs (called state-based reinforcement learning), real-world tasks usually do not provide accurate robot states. Instead, the input observations consist of high-dimensional images and sensor data obtained from the robot [3]. The curse of dimensionality further amplifies the problem of sample-inefficiency to the point where training a network takes huge amounts of time and state-of-the-art hardware, limiting the possible users to big corporations.

In order to close the gap between state-based and pixel-based RL regarding their sample-efficiency, recent research has focused on simplifying the task. The idea behind representation learning is to project the high-dimensional original input into a lower-dimensional latent space without loss of relevant information. Subsequently, this projection can then be fed as input observations into the RL algorithm. In an ideal scenario, the found latent space is semantically equal to what the state-based RL would use as observations, thus eliminating the difference to the pixel-based approach.

To encode the observations into low-dimensional representations, an encoder network needs to be learned. Previous research has used Variational Autoencoders (VAE) to encode the image into a compressed latent state from which they can be rebuilt to resemble the original input. This approach is suitable at first glance, however it is bound to fail in a real-world task because irrelevant information about a (potentially changing) background environment is encoded as well. Regarding the case of robotics, it is especially important that an encoder learns a representation solely based on the actual visible agent and its relevant goal. The look of the background, a different viewpoint on the robot or occluded observation parts should be neglected, meaning an observation of the same state but with different background or viewpoint should result in similar latent representations produced by the encoder. (**generalization ability**)

Research on that topic has shown that data augmentation has a positive influence on reinforcement learning, because it makes the representations more robust to background noise or changes. This however, makes learning a good encoder for representation learning or a reinforcement learning policy an even harder task that requires more specialized learning and an increasing amount of training data.

There exist different ideas how both of these problems can be tackled, for example using a self-supervised task that can result into a good encoder network and representations. In this paper an overview will be presented of some data augmentation-based and self-supervised approaches to enhance RL, grouping them into three categories based on their methodology: *data augmentation only approaches*, *time-utilizing approaches* and *view-utilizing approaches*. Table I provides an overview over the methods that will be covered in this paper, already grouping them and showing whether they use representation learning or not (Repr).

TABLE I
OVERVIEW OVER STATE-OF-THE-ART APPROACHES.

| Name | Author | Year | Loss | Repr. | Group |
|---|---|---|---|---|---|
| CURL: Constrastive Unsupervised Representation Learning | Srinivas et al. | 2020 | InfoNCE | Y | Data Aug |
| DRQ: Data-regularized Q | Yarats et al. | 2021 | None | N | Data Aug |
| RAD: Reinforcement Learning with Data Augmentation | Laskin et al. | 2020 | None | N | Data Aug |
| SVEA | Hansen et al. | 2021 | SVEA-loss | Y | Data Aug |
| Self-Predictive Representations | Schwarzer et al. | 2021 | Cosine-Similarity | Y | Time |
| Proto-RL | Yarats et al. | 2021 | Cross-Entropy | Y | Time |
| Augmented Temporal Contrast | Stooke et al. | 2021 | InfoNCE | Y | Time |
| FERM: CURL+RAD | Zhan et al. | 2021 | InfoNCE | Y | View |
| Time Contrastive Networks | Sermanet et al. | 2018 | Triplet | Y | Time View |

## II. THEORETICAL BACKGROUND

### A. Reinforcement Learning

Reinforcement Learning refers to a subfield of machine learning that is based on the concept of learning via trial and error. This interaction can be formulated as a Markov Decision Processs (MDP) $M = (S, A, P, r, y)$, with $S = \mathbb{R}^n$ being the state space and $A$ being the action space. Since single images do not offer full observability, a state $s_t \in S$ is defined as a series of consecutive frames $(o_t, o_{t-1}, o_{t-2}, ...)$ sampled from the image space $O$. The state transition distribution $P = P(s_{t+1}|s_t, a_t)$ defines a probability distribution over all possible next states based on an input state $s_t$ and action $a_t$. The reward function $r_t = r(s_t, a_t)$ maps an observation and action at a specific time $t$ to a reward $r_t$, while $y \in [0, 1)$ is a discount factor. An agents goal is to learn a policy $\pi(a_t|s_t)$ that samples the best action based on an input state $s_t$. Best action refers to the one that maximizes the discounted expected cumulated reward $R = \sum_{t=0}^{\infty} y^t r_t$.

Typically the learning is based on a tuple consisting of state, action, reward, next state $(s_t, a_t, r, s_{t+1})$. This tuple is saved in a replay buffer when the agent moves within its environment. By observing the relationship between these variables, it is possible to learn a RL policy that always selects the best action $a_t$ based on a state $s_t$.

### B. Representation Learning & Contrastive Learning

Representation learning is an essential step in increasing the sample-efficiency of pixel-based RL. It aims to automatically discover and learn patterns and structure in raw data in order to construct good representations that capture the most relevant information. These representations can then be used as input for tasks like reinforcement learning where their lower-dimensional nature helps to improve the learning speed of algorithms and generalization abilities.

Most state-of-the-art algorithms employ the concept of contrastive learning via siamese networks that can tackle the topic of view-invariance while at the same time learning lower dimensional representations that reduce sample-inefficiency of the RL task. Contrastive learning usually requires three inputs called *anchor*, *positive* and *negative*. All three are transformed into latent space using an encoder network. Using a score function, which is usually a simple similarity measure like the bilinear product $f_{score}(q, k) = q^T W k$, the agreement between a query $q$ and key $k$ can be calculated. It is desirable that the agreement between *anchor* and *positive* is maximized, while at the same time minimizing the agreement between *anchor* and *negative*. Approaches then can choose a loss, e.g. InfoNCE loss [4], to optimize the encoders that produce latent states so that these two objectives are reached.

### C. Data Augmentation in RL

To improve the generalization abilities of a reinforcement learning task, data augmentation can be used. It refers to transforming an input observation $o$ using an augmentation function $aug$ to an augmented observation $\tilde{o} = aug(o)$. In the case of pixel-based reinforcement learning, where $o$ are images, some examples for $aug$ are rotation, cropping, cutout, color-jitter or flipping. This list is not definitive as the possibilites for augmentation functions are immense, partly due to the fact that they can be combined and used sequentially to produce multi-augmented observations $\tilde{o} = aug_1(aug_2(o))$. Because augmentations can theoretically be used with any of the tasks and are usually interchangeable, the applied augmentation function will not be explicitely noted further in this paper. However, there will be a note that data augmentation has been used.

## III. IMPROVING REINFORCEMENT LEARNING USING DATA AUGMENTATION

Algorithms in this section are classified to majorly rely on data augmentation to improve either the reinforcement learning task or the representation learning task (if present).

### A. CURL

Contrastive Unsupervised Representation Learning [5] (CURL) proposed by Srinivas et al. is one of the most popular representation learning algorithms for pixel-based RL. Its main idea is very similar to the general concept of representation learning via siamese networks. The goal is to learn representations that disregard the performed data augmentation on an observation and only focus on relevant parts of images.

The concept can be seen in Fig. 1 and is as follows: An observation $o$ is sampled from the replay buffer and subsequently augmented two times using random crop to produce an anchor $o_q$ and a positive $o_k$. Both of them are then encoded into latent space, the anchor $o_q$ using *encoder* $f_{\theta_q}$ and the positive $o_k$ using a *momentum encoder* $f_{\theta_k}$, defined as momentum

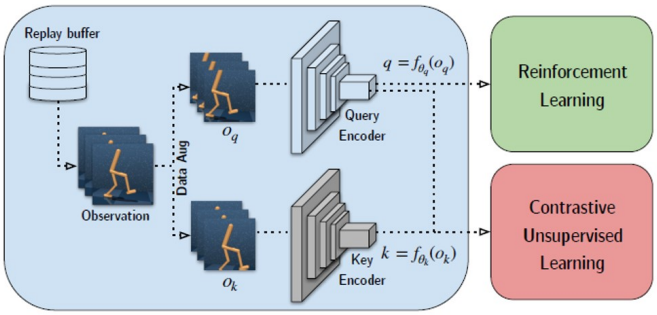


Fig. 1. Architecture of CURL [5]. An observation is augmented two times, then both are encoded indivudally and further used for either the RL task or the contrastive learning task.

averaged version of $f_{\theta_q}$. Only the anchor encoding $q = f_{\theta_q}(o_q)$ further serves as input for the reinforcement learning task. For the contrastive learning both, anchor encoding $q$ and positive encoding $k = f_{\theta_k}(o_k)$ are used to compute a loss that can be utilized to improve the encoder $f_{\theta_q}$. The loss applied in CURL is the InfoNCE loss [4] described in Eq. 1.

$$\mathcal{L}_q = \log \frac{\exp(q^T W k)}{\exp(q^T W k) + \sum_{i=0}^{K-1} \exp(q^T W k_i)} \tag{1}$$

In that equation, $q^T W k$ refers to the bilinear inner product, a simple way to measure similarity between two encodings of a query and a key. As negative keys, the positive keys of all other anchor-positive pairings are used (denoted as $k_i$ in the denominator)

Overall CURL is an important representation learning technique, because it was one of the first algorithms to demonstrate the ability to increase the sample-efficiency of a RL task by using self-supervised representation learning. As a consequence, later algorithms share similar concepts to CURL, some showing very similar structures while others are only more distantly related.

### B. RAD

Reinforcement Learning with Augmented Data [6] (RAD) proposed by Laskin et al. aims to improve sample-efficiency and generalization abilities of reinforcement learning. Crucially, the approach does not employ any self-supervised auxilliary tasks, nor does it use the RL loss itself. Instead it tries to enhance a reinforcement learning task simply by applying different combinations of data augmentation to the observations before using them as input for the RL.

After testing different state-of-the-art algorithms and comparing Pixel SAC and State SAC to RAD+SAC, the authors concluded that data augmentations alone can significantly improve sample-efficiency and generalization ability of RL. In some cases, RAD+SAC applied on image observations even matched or outperformed State SAC, thus successfully closing the gap between state-based RL and pixel-based RL. It was the first paper to perform extensive research about the effect of data augmentations alone, without combining it with other changes to the RL task.

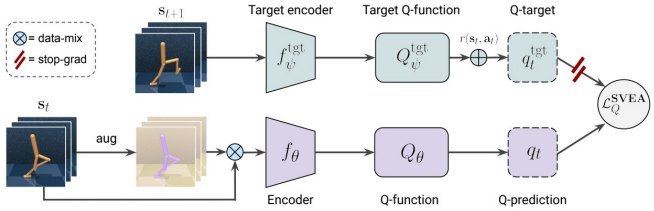


Fig. 2. Architecture of SVEA [8]. A combination of original and augmented observations is used to update encoder and Q-function, while target encoder and target Q-function operate on original observations only.

### C. DRQ

In the paper "Image Augmentation is all you need: Regularizing Deep Reinforcement Learning from Pixels" [7] by Yarats et al. another approach to improve reinforcement learning is described. As with RAD, the improvement is also solely based on data augmentation, without any auxilliary losses or self-supervised representation learning tasks. The approach is very simple: in classic reinforcement learning the Q-target value of a state is calculated based on only one input state. DRQ proposes to use multiple augmentations of the same state, computing the Q-target value for each of the newly obtained (augmented) states and averaging the Q-target value of that state over all augmentations. In the paper, augmentations consist of padding an image and then random cropping within the padded image to obtained a shifted version of it. By averaging over multiple augmentations, the variance of the estimated Q-target value is reduced.

Evaluation has shown that regularizing the Q-function by using averages can bring pixel-based RL closer to its state-based alternative, almost closing the gap between both.

### D. SVEA

Contrary to the last two methods, SVEA [8] proposed by Hansen et al. in their paper "Stabilizing Deep Q-Learning with ConvNets and Vision Transformers under Data Augmentation" is again a representation learning approach. However, similar to DRQ, it is also based on the concept of variance reduction. For SVEA, this is done by mixing together augmented and non-augmented observations. The full architecture can be seen in Fig. 2.

It is notable that augmentations are only used to learn the encoder $f_\theta$ and the Q-function $Q_\theta$, while the target encoder $f_\psi^{tgt}$ and the target Q-function $Q_\psi^{tgt}$ are working on non-augmented data. The learning of them is then done using the loss described in Eq. 2

$$\mathcal{L}_Q^{SVEA}(\theta, \psi) = \mathbb{E}_{s_t, a_t, s_{t+1}\ \mathcal{B}} \left[ \alpha || Q_\theta(f_\theta(s_t), a_t) - q_t^{tgt} ||_2^2 + \beta || Q_\theta(f_\theta(s_t^{aug}), a_t) - q_t^{tgt} ||_2^2 \right] \tag{2}$$

with the parameters $\alpha$ and $\beta$ balancing the weight of non-augmented and augmented data respectively.

Overall, the method was found to greatly improve Q-value estimation when using data augmentations, while still being sample-efficient and computationally inexpensive.

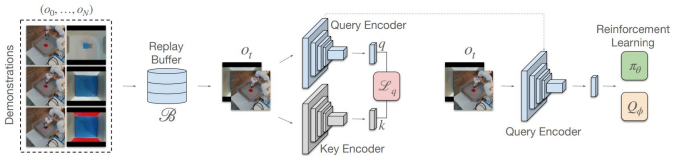


Fig. 3. Architecture of FERM [9]. The 3 major steps of creating human demonstrations, pre-training an encoder and learning a policy are shown.

## IV. Improving Representation Learning using View

### A. FERM: CURL+RAD

"A Framework for Efficient Robotic Manipulation" [9] (FERM) developed by Zhan et al. is the first and only algorithm solely grouped into view-utilizing approaches. It aims to achieve sample-efficient training of robot-arm policies from human videos by combining the concepts of contrastive pre-training from CURL and data augmentation from RAD. It is divided into three major parts as shown in Fig. 3.

First, a number of human demonstrations $o_0, ..., o_N$ are collected and stored in a replay buffer. After that, contrastive pre-training takes place, training a query and key encoder by example of the previously described method CURL. In the last step, the reinforcement learning of the robot arm itself is the center of attention. Building on the idea of RAD, observations $o_t$ coming from the cameras during the the reinforcement learning step are primarily augmented. After that they are encoded via the learned query encoder from the pre-training step and lastly used to train the optimal policy $\pi_\theta$ for the robot arm via reinforcement learning.

Interestingly, the authors chose for the demonstrations to contain multiple camera perspectives concatenated into one single observation. This is why the method is grouped in view-utilizing approaches, while CURL and RAD individually are grouped to data augmentation only.

Overall, FERM contributes to research by presenting a conceptually simple, yet data-efficient efficient way of using reinforcement learning to train on real robots.

## V. Improving Representation Learning using Time

In this section a group of algorithms is presented that all utilize the component of *time* to achieve better performance and output representations of their self-supervised representation learning task.

### A. Time Contrastive Networks

The main contribution of Time Contrastive Networks [10] (TCN) proposed by Sermanet et al. is that they utilize both, time-variance and viewpoint-variance in their approach. This means they are a hybrid and could be classified into the last category of view-utilizing methods as well. While the temporal component of the approach results in agent invariance (an agents pose changes over time), the viewpoint component results in viewpoint invariance. TCNs employ a simple triplet loss where the anchor $s_t^a$ and the positive $s_t^p$ are coming from different views at the same time, while the negative $s_{t+1}^n$ comes from the same view as the anchor but at at different point in time. The architecture of such a network is shown in Fig. 4.

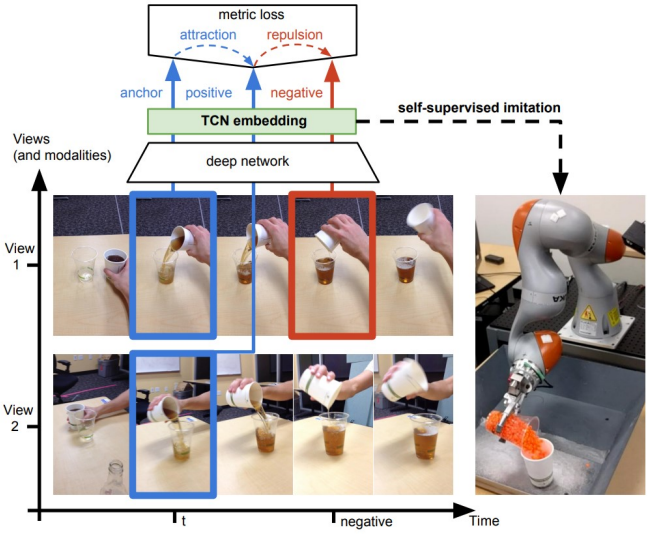


Fig. 4. Architecture of a TCN [10] with visualization of the sources for anchor, positive and negative.

$$\mathcal{L}(s_t^a, s_t^p, s_t^n) = max(||s_t^a - s_t^p||_2^2 - ||s_t^a - s_t^n||_2^2 + \alpha, 0) \quad (3)$$

The triplet loss described in Eq. 3 ensures that representations of frames occuring at the same time but over different views are grouped close together as they represent the same state (anchor $s_t^a$ and positive $s_t^p$), while representations of frames that are temporally very different but might look similar due to the same view are pulled apart (anchor $s_t^a$ and negative $s_t^n$). The hyperparamter $\alpha$ ensures a minimal distance between positive and negative pairs to avoid a very dense latent space result. The encoder network is then trained to minimize that loss. As the authors put it, the "model learns to explain what is common between images that look different" along the view axis and "to explain what is different between similar-looking images" on the temporal axis.

For applications where a second view is not available, the authors also proposed a variant called single-view TCN. In that case, the positive observation is chosen within a range around the anchor observation, meaning they likely represent similar states. Using that positive range, a margin is set around the anchor observation. The negative example is subsequently sampled from outside of that margin range, meaning it likely represents an extremely different state. Although this approach has been observed to be inferior to a multi-view TCN, it is still a viable option as representation learning task when the application is limited to only one viewpoint.

While the main goal of the paper was to use the TCN for imitating human poses from a video with a robot, the contrastive learning approach can certainly be used for other applications where noise-, agent- and viewpoint-invariant representations are needed and a second viewpoint is obtainable.

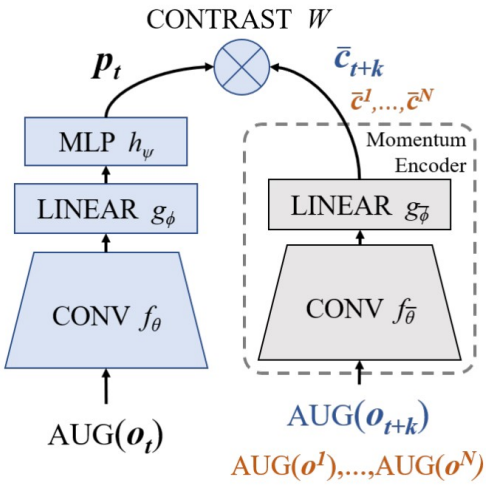


Fig. 5. Architecture of ATC [11]. Observations are encoded and associated with future observations encoded through a momentum encoder.

### B. Augmented Temporal Contrast

Similar to the concept of a TCN, Augmented Temporal Contrast [11] (ATC) proposed by Stooke et al. also relies on the dimension of time. Its general idea is to associate current observations with future observations of the same action sequence. As the name already says, ATC uses image augmentations - namely random shift - before encoding observations into latent space. For the purpose of simplicity, the data augmentation will subsequently not be explicitely noted. The main difference to TCN is, that the anchor $s_t^a = aug(o_t)$ and positive $s_{t+k}^p = aug(o_{t+k})$ both come from the same view but at different points in time.

The architecture of a ATC network can be seend in Fig. 5 and contains four components that are learned during training: the latent space *encoder* $f_\theta$, the *compressor* $g_\phi$, a *predictor* $h_\psi$ and a *contrastive transformation matrix* $W$. A momentum encoder, built as a slow moving average of the encoder and compressor, is used to encode the positives $\bar{c}_{t+k} = g_{\bar{\phi}}(f_{\bar{\theta}}(s_{t+1}^p))$, while the rest of the components are used to encode the anchor representation, compress it and forward it $p_t = h_\psi(g_\phi(f_\theta(s_t^a)))$. Rewriting observations as $s_t$ sampled from a dataset $S$ and their respective anchors as $s_{t+}$, the similarity can be defined as $l_{t,k+} = p_t W \bar{c}_{k+}$. The InfoNCE loss [4] is then applied, with the positives of other anchors serving as negatives (a concept similar to CURL).

$$\mathcal{L}^{ATC} = -\mathbb{E}_{\mathcal{O}}\left[log\frac{\exp(l_{t,t+})}{\sum_{s_k \in S}\exp(l_{t,k+})}\right] \tag{4}$$

Results of the original paper have shown that the ATC task alone is enough to train an encoder. This means that the downstream RL task and the training of the convolutional encoder can be decoupled, allowing the encoder to be trained individually without loss of performance due to the lack of RL loss. Previous work has not achieved to build a self-supervised representation learning task that does not suffer a performance loss when its training is decoupled from the training of the RL.

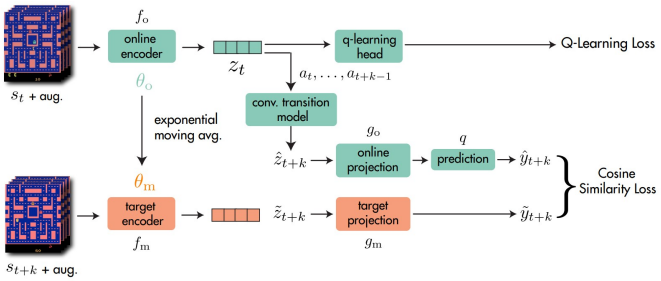


Fig. 6. Architecture of a Self-Predictive Representation Learning task [12].

### C. Self-Predictive Representations

A third method that relies on the dimension of time is described by Max Schwarzer et al. in their paper called "Data-Efficient Reinforcement Learning with Self-Predictive Representations" [12]. Its key idea is similar to that of ATC, because it also aims to find representations based on a temporal contrast of a sequence while using the same viewpoint. However, as the name self-predictive already hints, the approach tries to predict a whole sequence of future states based on only one input state and a sequence of actions. The full architecture of such a task can be seen in Fig. 6. An (augmented) input observation $s_t$ is first encoded by the *online encoder* $f_o$ to produce a representation $z_t$. This representation is further used for Q-learning, but more importantly: for the the self-supervised task. A *convolutional transition model* $h$ takes this representation as input, along with a sequence of actions $a_{t:t+K-1}$ and outputs a sequence of $K$ future observation predictions $\hat{z}_{t+1:t+K}$. Crucially, every one of these predictions is based on the last prediction made by the *transition model*, meaning they are generated iteratively following $\hat{z}_{t+k+1} \triangleq h(\hat{z}_{t+k}, a_{t+k})$. These states then go through an *online projector* $g_o$ and a *prediction head* $q$ to produce final future latent states $\tilde{y}_{t+1:t+K}$. At the same time, the actual future states $s_{t+1:t+K}$ are encoded via a *target encoder* $f_m$ and a *target projector* $g_m$, both defined as exponential moving average of the corresponding components in the online network to produce actual latent representations $\hat{y}_{t+1:t+K}$. Now the predicted future states and the actual future states can be compared and the predicition loss can be calculated by summing over the cosine similarities of each corresponding pair at all timesteps $t+k$ for $1 \leq k \leq K$:

$$\mathcal{L}_\theta^{SPR}(s_{t:t+K}, a_{t:t+K}) = -\sum_{k=1}^{K}\left(\frac{\tilde{y}_{t+k}}{||\tilde{y}_{t+k}||_2}\right)^T\left(\frac{\hat{y}_{t+k}}{||\hat{y}_{t+k}||_2}\right) \tag{5}$$

This loss can then be used to train $f_o$, $g_o$, $q$ and $h$, while the RL loss can additionally be used to train $f_o$ as well.

During evaluation the authors determined different key things, among them being the need for a seperate target encoder, the need for the projection networks and prediction

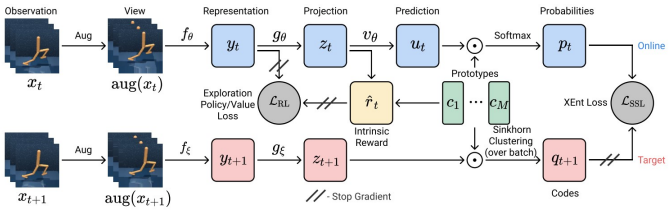


Fig. 7. Architecture of Proto-RL [13].

networks $g_o$ and $g_m$ and the importance of using the cosine similarity instead of a quadratics loss.

### D. Proto-RL

Even though the concept of Proto-RL [13], developed by Yarats et al., is fundamentally different from the previous three approaches, it also utilizes the component of time to its advantage. It is built with the intention to learn representations that are able to generalize to unseen RL tasks and that promote a faster exploration. To achieve that, the authors introduce the concept of prototypes, which serve as a basis for representations and exploration. The full architecture can be seen in Fig. 7.

First, an observation $x_t$ is augmented via random shift and subsequently encoded by the *online encoder* $f_\theta$ to form a representation $y_t$. This representation is further projected into a smaller latent space $z_t$ by a *projector* $g_\theta$ and features $u_t$ are obtained by the *predictor* $v_\theta$. Now the features $u_t$ are projected into a basis that is defined by $M$ *prototypes* $\{c_i\}_{i=1}^{M}$, obtaining probabilities $p_t$ for each prototype. Then, a batch of next augmented observations $x_{t+1}$ is encoded using the *target encoder* $f_\xi$ and the *target projector* $g_\xi$, both defined as exponential moving average of the respective components in the online network. Using the Sinkorn-Knopp clustering procedure, which makes sure that each prototype is assigned to an equal number of observations, the target probabilities $q_{t+1}$ are obtained. To update the parameters of the online network, a cross-entropy loss between the corresponding $p_t$ and $q_{t+1}$ is used as in Eq. 6:

$$\mathcal{L}^{SSL}(p_t, q_{t+1}) = -q_{t+1}^T \log p_t \tag{6}$$

The algorithm also computes an intrinsic reward to help exploration using the prototypes. However, the loss is not backpropagated to any of the representation learning components and is thus not relevevant for this paper.

Overall, the approach tries to spread out prototypes evenly during a task-agnostic pre-training phase. These prototypes can then subsequently be used for mapping them to states during a downstream RL task and approximate the value of this state, making exploration faster and cheaper.

## VI. Conclusion

This paper focused on presenting some state-of-the-art approaches to tackle the main problems faced by pixel-based reinforcement learning, them being sample-efficiency and generalization ability. Some of the approaches are based on representation learning, each using their own ideas how an architecture could look like that results in good representations of an image observation. Others are focused on data-augmentation without any representation learning in front of the downstreal RL task.

All methods were grouped into one (or multiple) of the three categories *data augmentation only approaches*, *time-utilizing approaches* or *view-utilizing approaches*. This grouping is only one of the many possibilities algorithms could be grouped and is only based on the small sample of methods described in the work. Due to the creativity and importance of finding a solution many researchers have come up with their own ideas and thus many more different methods exist.

Some presented approaches in this paper already successfully achieved to close the gap between state-based and pixel-based reinforcement learning, if only one single tasks or benchmarks. It is very likely that in the near future, further advances will be made that eventually fully close the gap, making pixel-based RL a very powerful tool.

## References


[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," p. 10.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Dec. 2013, arXiv:1312.5602 [cs]. [Online]. Available: http://arxiv.org/abs/1312.5602

[3] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, Apr. 2021. [Online]. Available: http://journals.sagepub.com/doi/10.1177/0278364920987859

[4] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation Learning with Contrastive Predictive Coding," Jan. 2019, arXiv:1807.03748 [cs, stat]. [Online]. Available: http://arxiv.org/abs/1807.03748

[5] M. Laskin, A. Srinivas, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5639–5650.

[6] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in neural information processing systems*, vol. 33, pp. 19 884–19 895, 2020.

[7] D. Yarats, I. Kostrikov, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," p. 21, 2021.

[8] N. Hansen, H. Su, and X. Wang, "Stabilizing deep q-learning with convnets and vision transformers under data augmentation," *Advances in neural information processing systems*, vol. 34, pp. 3680–3693, 2021.

[9] A. Zhan, R. Zhao, L. Pinto, P. Abbeel, and M. Laskin, "A framework for efficient robotic manipulation," in *Deep RL Workshop NeurIPS 2021*, 2020.

[10] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain, "Time-contrastive networks: Self-supervised learning from video," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1134–1141.

[11] A. Stooke, K. Lee, P. Abbeel, and M. Laskin, "Decoupling representation learning from reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9870–9879.

[12] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman, "Data-efficient reinforcement learning with self-predictive representations," *arXiv preprint arXiv:2007.05929*, 2020.

[13] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto, "Reinforcement learning with prototypical representations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 920–11 931.