Efficient Reinforcement Learning via Self-supervised learning and Model-based methods

Thomas Schmied
TU Wien
thomasschmied@hotmail.com

Maximilian Thiessen
TU Wien
maximilian.thiessen@tuwien.ac.at

Abstract

In recent years, Reinforcement Learning systems have led to remarkable accomplishments. Applications of Reinforcement Learning are vast and range from optimizing the energy consumption of data-centers and more efficient chip design to intelligent autonomous robots. However, Reinforcement Learning algorithms still have their limitations, most notably their lack of data efficiency. Different solutions were proposed that aim to mitigate this problem and in principle, there are two main approaches: Self-supervised learning and Model-based methods. This paper discusses the applications and theoretical foundations of Self-supervised learning and Model-based methods in the Reinforcement Learning context and investigates their individual strengths and weaknesses as well as their commonalities and intersections. The combination of both methods might lead to even better results and therefore, this paper will conclude by proposing potential ways to unify them.

1 Introduction

Current Reinforcement Learning (RL) systems are capable of achieving remarkable things. They were used to beat the world champions at humankind‘s most difficult board games, chess, shogi and most importantly at the game of Go [1, 2, 3]. Furthermore, reinforcement learning based systems defeated the world‘s champions at StarCraft and Dota, strategic real-time combat arena video games [4, 5]. But the applications of RL do not stop at games. Further potential applications are vast and range from optimizing the energy consumption of data centers and more efficient chip design to intelligent autonomous robots [6, 7, 8, 9].

Those are extraordinary achievements, but unfortunately, current reinforcement learning methods are incredibly sample-inefficient: AlphaStar, the StarCraft playing AI developed by DeepMind and the Dota agent by OpenAI had to acquire 200 and 10,000 years worth of real-time play experience, respectively [4, 10, 5].

However, the promise of reinforcement learning methods is profound. The broad RL framework offers a potential path to true generality of artificial agents. Nevertheless, their current limitations make them substantially difficult to apply to most real-world scenarios [11]. These problems are prevalent in all of deep learning to some degree, but in deep reinforcement learning the lack of efficiency is especially striking.

As a result, it remains to be figured out how to learn from fewer samples and interactions more efficiently [12]. Different solutions were proposed that aim to mitigate the problems. In principle, there are two main approaches: **Self-supervised learning** [13, 14, 15] and **Model-based methods** [16, 17, 18, 19].

Model-based methods aim to learn a predictive model of the environment and use it for planning. In contrast, the self-supervised approach aims to learn some useful representation of the environment which is then used to guide model-free methods. In this context “useful” means that the representation removes redundancy and as a result enables better generalization and more sample-efficient learning.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Lately, self-supervised learning has sparked enormous progress in the field of NLP [20, 21] and more recently also in computer vision [22, 23, 24, 25]. The next frontier is reinforcement learning.
This paper discusses the applications and theoretical foundations of self-supervised learning and model-based methods in the reinforcement learning context and investigates their individual strengths and weaknesses as well as their commonalities and intersections. The combination of both methods might lead to even better results and therefore, this paper will conclude by discussing potential ways to unify the two approaches.

2 Background

Reinforcement Learning. The reinforcement learning setting is the following: An agent (the decision-maker or learner) interacts with an environment (the world the agent lives in). At every time step t it finds itself in state s_t within the environment, performs action a_t, and receives reward r_t. Reinforcement learning is formalized by a Markov Decision Process $(\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma)$. $\mathcal{S}$ represents the state space, the set of all possible states $s_t \in \mathcal{S}$. In practice, it often is the case that the state of the environment is not completely observable, i.e. only partially observable. The environment then emits observations $o_t \in \mathcal{O}$ instead of states s_t. Similarly, the set of all possible actions, $\mathcal{A}$, is referred to as the action space with $a_t \in \mathcal{A}$. The transition probability function p from the current state to the next state represents the dynamics of the environment $p(s', r \mid s, a) = P\{s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a\}$. The discount factor $\gamma \in [0, 1]$ ensures that immediate rewards are perceived better than rewards received in the future. The overall goal in reinforcement learning is to maximize the cumulative expected reward, $G_t = \sum_{k=0}^{\infty} \gamma^t r_t$, where G_t is called the return [26].

Bellman equations. While the reward indicates what is beneficial in an immediate sense, value functions determine what is good in the long run. The state-value function determines the value of a state (i.e. determines how good it is to be in a state) and is defined as $V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s]$, where π represents the policy that maps states to actions and hence is responsible for action selection. The action-value function determines how good it is to perform a specific action in a specific state and is given by $Q_\pi(s, a) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s_{t+1}) \mid s_t = s, a_t = a]$. Those two equations are referred to as the Bellman equation for state-values and Bellman equation for action-values, respectively, and are fundamental throughout reinforcement learning [26].

Value based methods. Value based methods refer to methods that aim to learn the state-value function, V_π , and/or action-value function , Q_π, based on which the actions are selected. Examples are Dynamic Programming, Temporal Difference Learning, Q-Learning and most famously DQN (the deep variant of Q-learning, which will appear later)[27, 28, 29, 30].

Policy gradient methods. On the other hand, policy gradient methods aim to directly learn the parametrized policy, $\pi_\theta(a \mid s)$ for action selection, instead of the value function. Policy gradient methods perform gradient descent on a performance measure $J(\theta)$, where the policy gradient is defined as $\nabla_\theta J(\theta) \propto \mathbb{E}_\pi[\sum_a Q_\pi(s_t, a) \nabla \pi_\theta(a \mid s_t)] = \mathbb{E}_\pi[G_t \nabla \log \pi_\theta(a_t \mid s_t)]$ [26]. For example, A3C and Soft Actor-Critic (SAC) (which will appear later) are policy gradient methods, to be more specific actor-critic methods, which additionally learn a value function acting as the critic [31, 32].

Model-based methods. Like value functions or the policy, the dynamics of the environment can be learned as well. The learned representation of the dynamics is referred to as the model. Whether an agent does or does not learn and use a model of the environment is one of the most important distinctions between different kinds of reinforcement learning algorithms. Methods that learn and use a model are called model-based methods whereas methods that do not learn and use a model are called model-free (e.g. value based or policy gradient methods) [26]. The most famous examples of model-based methods are Dyna and Monte Carlo Tree Search, the algorithm used in AlphaGo (+ its variants) [33, 34, 1]. Model-based methods will be discussed in detail in Section 4.

Deep Reinforcement Learning. In classic reinforcement learning the value functions, the policy and the model are represented by tables, hence this setting is referred to as the tabular case. However, once the state and/or action space get large it becomes infeasible to represent them by tables and therefore, they have to be approximated. Analogously, this is referred to as the approximate case. Now, the term deep reinforcement learning refers to using deep neural networks for the approximation. E.g. when approximating the value functions $V(s; \theta) \approx V_\pi(s)$ and $Q(s, a; \theta) \approx Q_\pi(s, a)$, where θ refers to the parameters in the network [26].

3 Self-supervised learning

In general, the Supervised learning paradigm refers to learning by example, i.e. based on labelled data. The term "supervised" originates from the fact that the labels were assigned by a supervisor, a human being. In contrast, in the Unsupervised learning paradigm there is no supervisor involved. Typically, the goal of Unsupervised learning is to find hidden structure in unlabelled data points [35]. Self-supervised learning unifies both paradigms. Unlike in Supervised learning, but like in Unsupervised learning, in Self-supervised learning no labels are given and unlike in Unsupervised learning, but like in Supervised learning, in Self-supervised learning the goal is not to find hidden structure in the data points but to predict them correctly. In Self-supervised learning the labels are constructed automatically based on implicit information in the data which can be very desirable as it allows to leverage vast amounts of unlabelled data. There are different approaches to achieve this: In Natural Language Processing one typically aims to predict the next word based on previous words (or the masked words based on context words) [36, 20]. In computer vision, common tasks include predicting the angle of rotation of an image, reconstructing cut out regions or image re-colourization [37, 38, 39]. Hence, Self-supervised learning is like Unsupervised learning as no labels are given and it is like Supervised learning as it tries to predict those labels. As a result, useful representations are learned.
In reinforcement learning, self-supervision is a very natural choice as it can be used to acquire knowledge about the world the agent lives in, a useful representation of it, and as a result facilitate the interaction with it. Usually, the learned representation guides the learning process of model-free methods. However, again there are various approaches to how those representations can be learned and applied. Most commonly they are framed as auxiliary tasks. In the following the main methods that incrementally raised the bar of the state-of-the-art in data efficiency will be discussed.

3.1 Auxiliary losses

Auxiliary losses were introduced in [13] and the general idea is to maximize pseudo-reward functions in addition to the regular RL-objective. The pseudo-reward functions or auxiliary tasks are learned jointly during the training process or beforehand during pre-training. Learning to solve auxiliary tasks aims to empower the agent to control its future experiences in the environment and as a consequence allows for more efficient interactions. In this section different auxiliary tasks will be discussed.

Auxiliary control and reward tasks. The UNREAL (Unsupervised Reinforcement Learning with Auxiliary Learning) architecture incorporates two kinds of auxiliary tasks into the reinforcement learning framework: control tasks and reward tasks [13]. Both are learned in an off-policy fashion with data points sampled from an experience replay buffer [40].

Formally, the set of auxiliary control tasks is represented by $\mathcal{C}$, with the individual tasks $c \in \mathcal{C}$. The policy for auxiliary task c is denoted $\pi^{(c)}$. Then the total objective is defined as:

$$\arg\max_{\theta} \mathbb{E}_{\pi}[G_t] + \lambda_c \sum_{c \in \mathcal{C}} \mathbb{E}_{\pi_c}[G_t^{(c)}] \tag{1}$$

The first part of Equation 1 reflects the general RL objective and the second part the objective for the control tasks, which is to maximize the total reward obtained on the auxiliary control tasks. The auxiliary return is given by $G_t^{(c)} = \sum_{k=0}^{\infty} \gamma^t r_t^{(c)}$, with auxiliary rewards $r_t^{(c)}$. Again, θ are the parameters in the networks which represent π and all $\pi^{(c)}$. The relative importance of the control task objective is controlled by λ_c. Based on the objective the loss function can be formalized. Specifically, for each task the n-step version of the DQN loss is used, like in [31]:

$$\mathcal{L}_Q^{(c)} = \mathbb{E}[(G_{t:t+n}^{(c)} + \gamma^n \max_{a'} Q^{(c)}(s', a', \theta^-) - Q^{(c)}(s, a, \theta))^2] \tag{2}$$

The two auxiliary control tasks proposed are pixel control and feature control, but pixel control proved to be more effective. To perform pixel control, the input image is first separated into a grid of non-overlapping 4×4 cells. Then the auxiliary policy is trained to maximize the change in pixel intensity of the different regions which can be accomplished by selecting the appropriate actions. This task requires the agent to learn what actions are associated with particular changes in the visual scene.

Auxiliary reward tasks on the other hand aim to learn an understanding of the dynamics of the environment, specifically its reward structure. This has similarities with model-based methods but subtle dif-

ferences, as we will see later. Based on a sequence of previous frames $S_\tau = (s_{\tau-k}, s_{\tau-k+1}, ..., s_{\tau-1})$ the agent has to predict whether the next reward r_τ will be zero, positive or negative ($r_\tau \in \{+, -, 0\}$). Thus, the name of the task: reward prediction. Then the reward prediction $\mathcal{L}_{RP}$ is a multi-class cross-entropy loss across the three classes.
Incorporating auxiliary control and reward tasks results in the overall UNREAL loss function:

$$\mathcal{L}_{UNREAL}(\theta) = \mathcal{L}_{A3C} + \lambda_{VR}\mathcal{L}_{VR} + \lambda_{PC}\sum_c \mathcal{L}_Q^{(c)} + \lambda_{RP}\mathcal{L}_{RP} \tag{3}$$

In Equation 3, $\mathcal{L}_{A3C}$ and $\mathcal{L}_{VR}$ (value replay) refer to the losses of the A3C agent, a standard actor-critic architecture in reinforcement learning and we refer to the paper for its details [31]. The point is that the auxiliary tasks simply augment the A3C architecture and are learned jointly during training. Most importantly, those augmentations resulted in 10-18 times more data efficient learning.

Dynamics verification and Input reconstruction. Another paper that incorporated auxiliary tasks into the RL framework is called "Loss is its own reward" [14]. The authors took a similar approach to the one taken above by augmenting the A3C architecture. They proposed multiple further tasks and like in [13], the authors incorporate reward prediction.
Another task they propose is called dynamics verification, where the agent has to predict whether state-successor frames (s, s') or state-action-successor state pairs (s, a, s') are drawn from the environment.
A different option is reconstructing the input observation (the frame) using (variational) autoencoders or Generative Adversarial Networks [41, 42, 43]. However, reconstruction tasks have shown to be less effective.
The authors find that policies augmented with self-supervision converge to the same or better return and require fewer updates. Moreover, they evaluated different training approaches most importantly, policy pre-training and joint policy and auxiliary training. The results have shown an important finding: self-supervised pre-training purely on the auxiliary tasks followed by fine-tuning while the reinforcement learning training process already improves data efficiency significantly when compared to not using the auxiliary objectives. Nevertheless, optimizing the regular RL objective and the auxiliary objectives jointly (as was done in the previous section) gives better results.

Depth prediction and Loop closure classification. Another milestone paper is called "Learning to navigate in complex environments" [44]. Again, the authors of this paper proposed new auxiliary tasks, more specifically, auxiliary depth prediction and a loop closure classification task. The resulting architecture learned to solve large and visually rich 3D mazes solely from raw sensory inputs.
At every time step the agent obtains the current state of the environment in the form of an RGB frame and from this image depth information can be extracted. The depth prediction aims to predict this depth channel based on the colour channel of the input image. This has the effect that the agent acquires valuable information about the 3D structure of the environment. The task can be framed in two ways, either as a regression task based on the mean squared error or as a classification task by discretising depth at each position into different bands. Experiments showed that the classification formulation leads to faster convergence.
Loop closure classification is more specific for the navigation tasks. In this task the agent has to predict whether the current position has previously been visited given a local trajectory. For a visual representation of loop closure we refer to Figure 4 in the paper.
In contrast to [13], the proposed architecture learns the auxiliary tasks in an online fashion and does not require experience replay.

3.2 Contrastive Learning

Another exciting stream of research is called contrastive predictive coding (CPC) or contrastive learning [22]. The key idea of contrastive predictive coding is to learn useful representations from high dimensional input data by predicting future latent space. It achieves this by learning similar and dissimilar representations from similar and dissimilar data points. Contrastive learning is not specific to reinforcement learning but rather a universally applicable framework and has seen widespread adoptions in speech recognition [45, 46] and computer vision [47, 25, 24]. The idea of predictive coding is to predict future missing or contextual information [48]. Predictive coding principles are extensively studied in the neurosciences, as they appear to be supported by the human brain, especially at the level of the visual cortex [49, 50].
The original formulation of contrastive predictive coding was given in [22]. The architecture consists of a non-linear encoder to transform input observations into their latent counterparts and an autoregressive model that constructs a context vector based on the latent states prior to a specific point in

time. Furthermore, the authors proposed the loss function InfoNCE used for contrasting positive and negative samples. Since the original formulation several enhancement have been suggested to augment the architecture, including CPCv2, momentum contrast (MoCO), SimCLR and MoCov2 [47, 23, 24, 25].

3.2.1 CURL: Contrastive Unsupervised Representation Learning for Reinforcement Learning

Recently, in [15] the contrastive framework was adjusted and applied specifically to reinforcement learning. The resulting architecture, CURL, achieved new state-of-the-art results in data efficiency on the Deep Mind Control Suite benchmark (DMControl) [51] and Atari games benchmark [52] (100k interaction steps as proposed by [53]), the most common benchmarks for data efficiency in reinforcement learning.
Similar to [23], the contrastive learning component is framed as a dictionary look-up task but is conceptually the same as in [22]. Therefore, CURL aims to construct queries and keys from input observations (i.e. image frames). The query (or anchor) is represented by q and is considered the ground truth. The keys (or targets) are represented by $\mathbb{K} = \{k_0, k_1, ..., k_n\}$ of which one is a positive sample, denoted k_+. The others are negative samples and denoted $\mathbb{K} \setminus \{k_+\}$. Anchor and positive samples are obtained from the same input observation o, but are different crops thereof, i.e. o_q and o_{k_+}. In contrast, negative keys are cropped regions from different images, o_{k_-}. For each query and key, an 84×84 region is cropped from a 100×100 image.
However, the observations are not fed directly, but are encoded into a latent representation first: $q = f_{\theta_q}(o_q)$ and $k = f_{\theta_k}(o_k)$. With the parameters of the query encoder network θ_q and the parameters in the key encoder network θ_k.
Moreover, it is common to share the encoder parameters. Thus, θ_k is an exponential moving average of the query encoder parameters: $\theta_k = m\theta_k + (1-m)\theta_q$. As a result, during training no gradients need to be calculated for updating the key encoder. This approach is known as momentum contrast (MoCo)[23].
To compute similarities between anchor and targets CURL opts for the bilinear product $sim(q, k) = q^T W k$, with W a learned weight matrix. The loss function is the same as in the original formulation of CPC, the InfoNCE loss:

$$\mathcal{L}_q = \log \frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{i=0}^{n} \exp(q^T W k_i)} \tag{4}$$

Hence, the model has to learn to distinguish between positive and negative samples given the context and by doing that it develops a rich internal understanding of the environment. As a consequence, the encoded query q is a more compact representation of the current state that removes redundancy and therefore enables much more efficient learning. Consequently, the RL-algorithm is only fed the representation of the state q. For a visual representation of the architecture we refer to Figure 1 in [15].
At its core CURL is a general framework that extends reinforcement learning with contrastive learning. Essentially, it is applicable to any kind of RL algorithm. The authors augmented the Soft Actor-Critic (SAC) architecture and Rainbow DQN and achieved new state-of-the-art results on DMControl and Atari100k, respectively [32, 54]. Overall, CURL achieves a $2.8\times$ mean higher performance scores on the DMControl suite benchmark and a $1.6\times$ performance gain on Atari when compared to prior methods.

3.3 Exploration

Reinforcement learning agents have to behave non-optimally in order to learn the optimal behaviour. To learn the optimal behaviour, they need to explore the state and/or action-space of the environment. This dilemma is known as "exploration vs. exploitation": Exploring the environment to discover more beneficial areas of state and/or action space versus exploiting what is already known about the environment [26]. Finding the optimal exploration strategy in reinforcement learning is an active area of research itself and still an open research question.
There are various approaches to exploration but currently the most common strategy is to act in an ϵ-greedy fashion: Acting greedily most of the time, but occasionally performing a random action (with probability ϵ) [26]. Self-supervised learning is an attractive alternative to enable more focused exploration of the environment, which results in less required environment interaction and hence

more sample-efficient learning. Recently, various different ideas to self-supervised exploration have been proposed [55, 56, 57, 58, 59, 60].

One interesting and promising approach to self-supervised exploration is *curiosity*. While standard reinforcement learning systems learn purely from extrinsic reward signals (obtained from the environment), curiosity is a framework for intrinsic reward signals (come from the agent itself). This approach was proposed in [55]. The reward at time step t is then $r_t = r_t^i + r_t^e$ with r_t^i and r_t^e the intrinsic and extrinsic reward, respectively.
Specifically, the authors formulate curiosity as the error in agent's ability to predict the consequences of its action. To achieve this the authors propose an Intrinsic Curiosity Module (ICM), which consists of an inverse dynamics model g and a forward dynamics model f (similar to models in Section 4, but without planning). The inverse dynamics model predicts the action that was taken for a given state-next_state pair (s_t, s_{t+1}): $\hat{a} = g(s_t, s_{t+1}; \theta_I)$. It also encodes the states s_t and s_{t+1} into their latent representations $\phi(s_t)$ and $\phi(s_{t+1})$. The latent vectors are then used in the forward dynamics model f which predicts the next latent state given the current latent state and action: $\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t; \theta_F)$. The parameters in the networks of the inverse and forward dynamics model are represented by θ_I and θ_F, respectively.
The intrinsic reward is then the error between the predicted latent state and the actual latent state: $r_t^i = \frac{\eta}{2} \mid\mid \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \mid\mid^2$ with a scaling factor η. Intuitively, the intrinsic reward indicates how surprising a transition is to the agent. Hence, it is rewarded for seeking novel experience.
Overall, incorporating curiosity into the architecture led to fewer required interactions with the environment and enabled agents to solve tasks that they could not solve with random exploration.
Other ideas on curiosity-driven exploration were introduced in [56, 59] and led to further improvements.

4 Model-based methods

As was discussed in Section 2 the dynamics of the environment can be learned. This is the idea of model-based methods. Once the model is learned, it can be used to simulate transitions. Based on the simulated experience the value functions and/or policy can be optimized; this is known as *planning* [26]. So how do we learn the transition dynamics? One approach is to learn a parametrized function $p_\theta(s, a)$, with θ the parameters in the network, and minimize the mean squared error over some samples: $L = \sum_i \mid\mid p_\theta(s_i, a_i) - s_i' \mid\mid^2$. The transition model is denoted as $p_\theta(s_{t+1} \mid s_t, a_t)$. Similarly, a reward model can be formalized: $p_\theta(r_t \mid s_t)$. As the input data can be very high-dimensional, learning those models in observation space can be challenging. Therefore, modern model-based methods usually learn an additional representation model that maps from the potentially high-dimensional observation o (e.g. image frame) to a latent representation of the state: $p_\theta(s_{t+1} \mid s_t, a_t, o_{t+1})$. In this case, the transition and reward model operate in compact latent space without observing or constructing the actual image. Note that the parameters θ can be shared or learned separately.
To learn those models the samples come from a replay buffer $\mathcal{D}$ that stores the relevant tuples. In case all three models are learned, it contains state-action-reward-next state tuples: $\mathcal{D} = \{(s, a, r, s')_i\}$.
For humans understanding the world we live in seems to be very intuitive. Very early on, we have or acquire a foundational understanding, models, of the natural phenomenons like numbers, space, physics and other humans that govern our environment. Those models enable us to imagine future events, guide our behaviour and allow us to learn tasks efficiently [12]. Therefore, model-based learning is considered one of the core ingredients towards more intelligent artificial systems [12, 61]. In the following we will discuss some of the most important ideas and selected state-of-the-art methods in the model-based domain that significantly improved data efficiency of reinforcement learning.

4.1 World Models

One milestone paper is called "World Models" [17, 62]. The proposed architecture consists of 3 components: Vision (V), Memory (M) and Controller (C). An input observation (image frame), o_t, is fed into the Vision component, a Variational Autoencoder (VAE) [42, 63], which learns a compressed representation of it. Hence, V acts as the representation model and maps o_t to latent representation s_t. Next, the latent representation is fed into the Memory component. M is a mixture density network [64] combined with a RNN (MDN-RNN) [65] and acts as the transition model. The choice for the

MDN-RNN was made, as it allows to capture time dependencies via the hidden state. Therefore, the transition model is defined as $p_\theta(s_{t+1} \mid s_t, a_t, h_t)$ with h_t the hidden state of the MDN-RNN. Finally, the latent state s_t and hidden state h_t are concatenated and fed jointly into the Controller component. C is a simple linear layer and maps concatenated latent and hidden state to action: $a_t = W_c[s_t h_t] + b_c$. The weight matrix W_c and bias vector b_c are learned during training.
Once, the state dynamics component M is trained it can be used to generate simulated transitions in latent space, based on which C can be improved upon.
The proposed architecture successfully learned to solve a race-car and a take-cover task of the OpenAI Gym framework [66]. It is interesting to note that the Controller component, which selects the actions, requires only a tiny fraction of the total number of parameters in this model.

4.2 Model based reinforcement learning for Atari

Next, we will discuss Simulated Policy Learning (SimPLe), the first model-based architecture that performed well on the common Atari benchmark [53, 52]. SimPLe learns a world model that is similar to the architecture proposed in [67], a deep action conditional convolutional feedforward network. This architecture was specifically developed in the context of video prediction in Atari and we refer to the paper for more details. Unlike, the previous world model, this model does not operate in latent space. Hence, the actual next frame is generated. Once the model is learned, it is used to train a Proximal Policy Optimization (PPO) agent, a common architecture in RL [68]. The PPO agent then collects experience in the real environment that, in turn, is used to further optimize the model.
While humans are able to play Atari games successfully within minutes [69], the Atari benchmark represents a long-standing challenge for reinforcement learning algorithms and highlights their data-inefficiency. Prior to SimPLe, the best performing algorithms on this benchmark were Rainbow-DQN and PPO, model-free methods [54, 68]. SimPLe learns faster than the model-free methods on nearly all games. Depending on the game SimPLe is 2-10$\times$ more data-efficient than Rainbow-DQN. Furthermore, on some games, SimPLe achieves the same scores after 100k steps as pure PPO on 10M steps. However, the overall scores SimPLe achieves after 10M steps are lower than the ones of model-free methods. Nevertheless, the results are promising and suggest model-based reinforcement learning as a highly efficient alternative to model-free methods.

4.3 Dream to control: Learning behaviors by latent imagination

Other important ideas that build on [18] were introduced in "Dream to control: Learning behaviors by latent imagination" [19]. The resulting architecture is called Dreamer and learns to solve long-horizon tasks in the DMControl suite purely by latent imagination. Dreamer learns a representation model, transition and reward model with shared parameters θ. In contrast to the previous architecture, Dreamer additionally learns an action-value function, $q_\phi(s_t, a_t)$, and state-value function, $v_\psi(s_t)$ with the parameters ϕ and ψ in the respective dense neural networks. Both action and state-value function are learned entirely in latent imagination.
The training process of Dreamer is characterized by three phases: (1) Dynamics learning, (2) Behavior learning and (3) Environment interaction. This distinction originates from [33]. In (1) the representation, transition and reward model are learned based on a batch sampled from the replay buffer $\mathcal{D}$. In (2) the state and action-value function are learned based on transitions dreamed up by the transition model and reward model. And in (3) the action-value function is used to gather new experience in the actual environment, but without any parameter updates happening.
Dreamer outperformed the previous state-of-the-art method D4PG by a significant margin, while only requiring 5% of the number of training steps on the DeepMind control suite. This translates to a training time of only 3 hours compared to 24 hours for D4PG. Prior to CURL (discussed in Section 3.2.1) Dreamer achieved the top scores on the DMControl benchmark.

4.3.1 Planning to Explore via Self-Supervised World Models

In this section we will briefly discuss an architecture that establishes the link between model-based methods and self-supervised exploration (discussed in 3.3). The architecture Plan2Explore was proposed in [70]. Plan2Explore learns to explore environments by leveraging planning. While the methods discussed in Section 3.3 learn a model-free exploration policy, Plan2Explore learns the exploration policy purely from imagined trajectories, i.e. in a model-based fashion. To achieve this, the architecture is based on the Dreamer, the agent discussed in the previous Section 4.3. First,

Plan2Explore leverages planning to explore the environment in a self-supervised fashion, i.e. without any external rewards but solely based on intrinsic rewards. By doing that it learns a global model of the world. After the exploration phase the agent has to learn to solve different downstream tasks in a zero-shot or few-shot manner.

5 Discussion

Overall, we found that both self-supervised learning and model-based methods can be powerful tools in the RL-toolbox and result in more sample-efficient learning. In this section we will first discuss the commonalities and intersections between the two approaches and then propose some methods to unify them.

The world models learned in model-based methods are actually learned in a self supervised fashion. However, the representations differ for both approaches. The learned representation of the environment of model-based methods has to be as accurate and close to the real environment as possible, as the model is used for planning. This is necessary since otherwise the generated experience could be quite distant from real experience. As a result the behaviour learned in simulation would not transfer well to the real environment. Therefore, a wrong model could even have detrimental effects on the learned value functions and/or policy [26]. On the other hand, for self-supervised methods the representation does not necessarily have to be as accurate as possible. It only has to be useful in some way. But of course to be useful, it has to reflect the environment sufficiently, but in some beneficial way (i.e. by removing redundancy).

Despite their superficial differences, we believe that the two approaches are largely compatible in many ways as they address orthogonal shortcomings. In Section 3 three possibilities to extend the reinforcement learning framework with self-supervision were examined: auxiliary losses, contrastive learning and exploration. Section 4.3.1 discussed an architecture that incorporates self-supervised exploration into the model-based framework. In the following, we will briefly discuss potential methods to combine auxiliary losses and contrastive learning with model-based methods and thus exploiting the strengths of both approaches.

Take contrastive learning: As addressed in Section 4, modern model-based methods usually learn a representation model $p_\theta(s_{t+1} \mid s_t, a_t, o_{t+1})$ that maps a high dimensional state to its latent counterpart. The representation model could be learned in a contrastive fashion with queries $q = f_{\theta_q}(o_q)$ and keys $k = f_{\theta_k}(o_k)$ as used in Section 3.2.1. The dynamics and reward models can then be learned as previously. However, one could go one step further and learn the dynamics and reward model in a contrastive fashion as well, by sampling positive and negative states/rewards from the replay buffer. Incorporating those changes would allow to leverage both the power of contrastive learning and of model-based methods. Furthermore, this would require a replay buffer of observations to learn the representation model. Nevertheless, it is not clear whether the contrastive representations would be advantageous. This needs to be examined experimentally and remains future work.

How can auxiliary losses and model-based methods be unified? A promising candidate is the auxiliary dynamics verification task. Based on state-action-next state tuples (s, a, s'), the goal is to predict whether the transition comes from the environment or not. This auxiliary loss could potentially be incorporated into the loss function of the transition model and learned jointly.

6 Conclusion

Despite many remarkable advances in reinforcement learning in the last few years one of the most notable limitations of reinforcement learning algorithms remains: data-inefficiency. This paper discusses two promising approaches for more efficient reinforcement learning, self-supervised learning and model-based methods. In Section 3 different self-supervised methods are covered. Specifically, the section discusses auxiliary tasks, contrastive learning and self-supervised exploration strategies. Section 4 examines different model-based approaches and current state-of-the-art architectures, including SimPle, Dreamer and Plan2Explore. Overall, we conclude that both self-supervised learning and model-based methods can be powerful extensions to the reinforcement learning framework and are fruitful approaches towards more efficient reinforcement learning architectures and we look forward to exploring the limits of those methods.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[4] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[5] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[6] N. Lazic, C. Boutilier, T. Lu, E. Wong, B. Roy, M. Ryu, and G. Imwalle, "Data center cooling using model-predictive control," in *Advances in Neural Information Processing Systems*, pp. 3814–3823, 2018.

[7] A. Mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. M. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, A. Babu, Q. V. Le, J. Laudon, R. C. Ho, R. Carpenter, and J. Dean, "Chip placement with deep reinforcement learning," *arXiv preprint arXiv:2004.10746*, 2020.

[8] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, *et al.*, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.

[9] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[10] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II." `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[11] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[12] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and brain sciences*, vol. 40, 2017.

[13] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in *ICLR*, 2016.

[14] E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell, "Loss is its own reward: Self-supervision for reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*, OpenReview.net, 2017.

[15] M. Laskin, P. Abbeel, and A. Srinivas, "Curl: Contrastive unsupervised representation learning for reinforcement learning," in *ICML 2020: 37th International Conference on Machine Learning*, vol. 1, 2020.

[16] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine learning proceedings 1990*, pp. 216–224, Elsevier, 1990.

[17] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[18] D. Hafner, T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," in *ICML*, 2019.

[19] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," in *International Conference on Learning Representations*, 2019.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

[21] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[22] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *CoRR*, vol. abs/1807.03748, 2018.

[23] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9729–9738, 2020.

[24] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *ICML 2020: 37th International Conference on Machine Learning*, vol. 1, 2020.

[25] X. Chen, H. Fan, R. B. Girshick, and K. He, "Improved baselines with momentum contrastive learning," *arXiv preprint arXiv:2003.04297*, 2020.

[26] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[27] R. Bellman, "The theory of dynamic programming," tech. rep., Rand corp santa monica ca, 1954.

[28] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[29] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[30] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[31] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, pp. 1928–1937, 2016.

[32] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, pp. 1861–1870, 2018.

[33] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[34] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*, pp. 72–83, Springer, 2006.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[36] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI Blog*, vol. 1, no. 8, p. 9, 2019.

[37] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," in *ICLR 2018*, 2018.

[38] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2536–2544, 2016.

[39] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European conference on computer vision*, pp. 649–666, Springer, 2016.

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

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

[42] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2014.

[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[44] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, OpenReview.net, 2017.

[45] K. Kawakami, L. Wang, C. Dyer, P. Blunsom, and A. v. d. Oord, "Learning robust and multilingual speech representations," *arXiv preprint arXiv:2001.11128*, 2020.

[46] M. Rivière, A. Joulin, P.-E. Mazaré, and E. Dupoux, "Unsupervised pretraining transfers well across languages," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7414–7418, IEEE, 2020.

[47] O. Henaff, "Data-efficient image recognition with contrastive predictive coding," in *ICML 2020: 37th International Conference on Machine Learning*, vol. 1, 2020.

[48] P. Elias, "Predictive coding–i," *IRE Transactions on Information Theory*, vol. 1, no. 1, pp. 16–24, 1955.

[49] R. P. Rao and D. H. Ballard, "Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects," *Nature neuroscience*, vol. 2, no. 1, pp. 79–87, 1999.

[50] G. B. Keller and T. D. Mrsic-Flogel, "Predictive processing: a canonical cortical computation," *Neuron*, vol. 100, no. 2, pp. 424–435, 2018.

[51] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[52] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[53] Ł. Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, *et al.*, "Model based reinforcement learning for atari," in *International Conference on Learning Representations*, 2019.

[54] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *AAAI*, 2018.

[55] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jul 2017.

[56] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," in *International Conference on Learning Representations*, 2018.

[57] Y. Aytar, T. Pfaff, D. Budden, T. Paine, Z. Wang, and N. de Freitas, "Playing hard exploration games by watching youtube," in *Advances in Neural Information Processing Systems*, pp. 2930–2941, 2018.

[58] A. Anand, E. Racah, S. Ozair, Y. Bengio, M.-A. Côté, and R. D. Hjelm, "Unsupervised state representation learning in atari," in *Advances in Neural Information Processing Systems*, pp. 8766–8779, 2019.

[59] D. Pathak, D. Gandhi, and A. Gupta, "Self-supervised exploration via disagreement," in *International Conference on Machine Learning*, pp. 5062–5071, 2019.

[60] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, *et al.*, "Never give up: Learning directed exploration strategies," in *International Conference on Learning Representations*, 2019.

[61] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017.

[62] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems*, pp. 2450–2462, 2018.

[63] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International Conference on Machine Learning*, pp. 1278–1286, 2014.

[64] C. M. Bishop, "Mixture density networks," 1994.

[65] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[66] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym.(2016)," *arXiv preprint arXiv:1606.01540*, 2016.

[67] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh, "Action-conditional video prediction using deep networks in atari games," in *Advances in neural information processing systems*, pp. 2863–2871, 2015.

[68] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[69] P. A. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman, "Human learning in atari," in *2017 AAAI Spring Symposium Series*, 2017.

[70] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, and D. Pathak, "Planning to explore via self-supervised world models," in *ICML*, 2020.