Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Intrinsically Motivated Reinforcement Learning

Dissertation thesis

2023

Mgr. Matej Pecháč

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Intrinsically Motivated Reinforcement Learning

Dissertation thesis

Study program:	2508 Informatics
Department:	Department of Applied Informatics
Supervisor:	prof. Ing. Igor Farkaš, Dr.

Bratislava, 2023
Mgr. Matej Pecháč

I hereby declare that I wrote this work by myself, only with the help of the referenced literature.

Bratislava, 30. 4. 2023

....

Mgr. Matej Pecháč

Acknowledgement

I would like to express my gratitude to my thesis supervisor, prof. Igor Farkaš, for his patience and guidance, to the friend and former colleague Michal Chovanec for long and fruitful discussions as well as invaluable advice on this topic and to my wife Barbora for forbearance and understanding.

Abstract

Reinforcement learning can solve decision-making problems and train an agent to behave in an environment according to a predesigned reward function. However, such an approach becomes very problematic if the reward is too sparse and the agent does not come across the reward during the environmental exploration. The solution to such a problem may be in equipping the agent with an intrinsic motivation, which will provide informed exploration, during which the agent is likely to also encounter external reward. Novelty detection and state prediction are the promising branches of intrinsic motivation research. We present **Self-supervised Network Distillation** (SND), a class of intrinsic motivation algorithms based on the distillation error as a novelty indicator, where the target model is trained using self-supervised learning. We adapted three existing self-supervised methods for this purpose and experimentally tested them on a set of ten environments that are considered difficult to explore. We also applied self-supervised learning in the training of the forward model, denoted **Self-supervised Predictor** (SP) and showed that even for this approach to intrinsic motivation, it leads to an improvement in the agent's performance. The results show that our approach achieves faster growth and higher external reward for the same training time compared to the baseline models, which implies improved exploration in a very sparse reward environment.

Keywords: reinforcement learning, self-supervised learning, intrinsic motivation, novelty detection, state prediction, knowledge distillation, sparse reward

Abstrakt

Učenie posilňovaním dokáže vyriešiť rozhodovacie problémy a natrénovať agenta tak, aby sa správal v prostredí podľa vopred navrhnutej funkcie odmeny. Takýto prístup sa však stáva veľmi problematickým, ak je odmena príliš riedka a agent na odmenu počas prieskumu prostredia nenarazí. Riešením takéhoto problému môže byť vybavenie agenta vnútornou motiváciou, ktorá poskytne informované skúmanie, počas ktorého sa agent pravdepodobne stretne aj s externou odmenou. Detekcia nových stavov a predikcia stavu sú sľubnými vetvami výskumu vnútornej motivácie. V práci predstavujeme triedu algoritmov vnútornej motivácie založených na destilačnej chybe ako indikátoru novosti – **Self-supervised Network Destillation** (SND), kde sa cieľový model trénuje pomocou samokontrolovaného učenia. Na tento účel sme prispôsobili tri existujúce metódy samokontrolovaného učenia a experimentálne sme ich otestovali na súbore desiatich prostredí, ktoré sa považujú za ťažko preskúmateľné. Samokontrolované učenie sme aplikovali aj pri tréningu dopredného modelu – **Self-supervised Predictor** (SP), a ukázali sme, že aj pre tento prístup k vnútornej motivácii vedie k zlepšeniu výkonu agenta. Výsledky ukázali, že náš prístup dosahuje rýchlejší rast a vyššiu externú odmenu za rovnaký čas tréningu v porovnaní so inými modelmi, čo znamená zlepšenú exploráciu v prostrediach s veľmi riedkou odmenou.

Kľúčové slová: učenie posilňovaním, samokontrolované učenie, interná motivácia, detekcia nových stavov, predikcia stavov, prenos znalostí, riedka odmena

List of Figures

List of Tables

Acronyms

CNN Convolutional Neural Network. 51

ICM Intrinsic Curiosity Module. 27, 46, 56, 60, 66, 67, 69–71

IM Intrinsic Motivation. 1, 2, 67, 70

MPD Markov Decision Process. 3, 5, 6, 30

PCA Principal Component Analysis. 62

RL Reinforcement Learning. 1, 5–7, 20, 25, 68, 72

RND Random Network Distillation. 33, 45, 51, 53, 56, 60, 62–64, 69–71

SND Self-supervised Network Distillation. vii, 45, 46, 51, 55, 56, 60, 63, 67, 69–71

SP Self-supervised Predictor. 46, 49, 51, 55, 56, 60, 66, 67, 69–71

SSL Self-Supervised Learning. vii, 13, 22, 45, 46, 48, 50, 55, 60, 69–72

Contents

Chapter 1

Introduction

The development of reinforcement learning (RL) methods has achieved much success over the last decade, since together with advances in computer vision (He et al., 2016; Krizhevsky et al., 2012), it became possible to teach agents to solve various tasks, or play computer games (Mnih et al., 2013) (see overview in (Souchleris et al., 2023)), even surpassing human players (Mnih et al., 2015). Nevertheless, these single tasks require very long training times and a lot of computational resources. Coping with complex (continuous) environments such as the real world is still a challenge. There are several research opportunities, one of them being the search for more efficient learning methods, that can, among other things, provide better feature representations. Another is hardware development, which attempts to adapt to the requirements of neural networks that are currently being used in the RL field.

The complex environments with sparse rewards pose a special challenge for RL approaches. The most popular computational approach to make RL more efficient is based on a concept of *intrinsic motivation* (IM) (Baldassarre et al., 2014). IM introduces a new reward function – an intrinsic reward that the agent generates for itself. This reward function should reflect the agent's effort to gain as much knowledge as possible about the environment and acquire skills that will be useful for the given environment and help it maximize the environmental reward function – external reward. In this way, it is possible to ensure an informed exploration of the environment, life-long learning and open-ended development at the agent.

The aim of this thesis is to investigate and improve methods of IM. We focused on exploration strategies in environments with very sparse reward that require informed exploration. Knowledge-based intrinsic motivation deals with this area. Within it, we introduced a new class of algorithms for detecting the novelty of a state, which can serve as a suitable IM signal. This new class of algorithms combines intrinsic motivation with self-supervised learning methods, which are also gaining popularity and, in our opinion, will play an important role in the future development of agents. We think that the combination of IM and self-supervised learning methods is quite natural, since both principles rely only on the agent and its internal systems, without the need for designer intervention to define either the reward function or the labeled data. The source code for this work is available here: `https://github.com/Iskandor/SND`

The thesis is divided into six chapters. Chapter 2 is a review of basic principles of reinforcement learning, including specific on-policy and off-policy algorithms. In Chapter 3, we present the basic concepts of self-supervised learning, model architecture, metrics and loss functions. In Chapter 4, we define motivation, intrinsic motivation, which are followed by an overview of different approaches and their methods to generate an intrinsic reward for the agent. In Chapter 5 we present our methodology, experiments along with results and analysis, aiming to explain the success of our algorithms and to reveal the failure of the compared algorithms. Chapter 6 includes a summary of our research along with conclusions drawn from the results and analyses. At the end, we also attached several directions in which the work can be further developed and we present the sequence of our ideas that led to the creation of these models.

Chapter 2

Reinforcement learning

Reinforcement learning (RL) (Sutton and Barto, 1998, 2018) is a domain of machine learning focused on solving decision problems by learning from interaction of the agent and environment solely from its own experience. This interaction is continual, the agent selects actions from its repertoire and the environment responds to those actions and presents a new situation, denoted state, to the agent. With a new state the environment also provides the reward signal and the agent maximizes this reward. The scheme of this interaction is presented in Figure 2.1. Reinforcement learning is intensively studied in many areas such as control theory, robotics, game theory, multi-agent systems and many more. It is considered as one of three main learning paradigms together with supervised and unsupervised learning. The decision problem is often formalized as a Markov decision process.

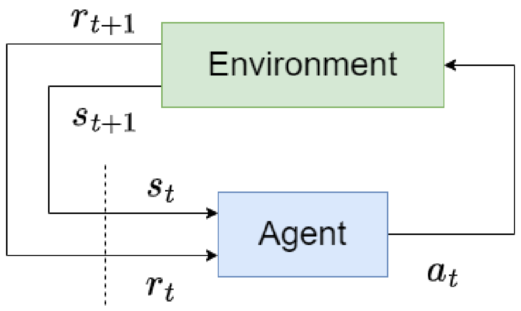


Figure 2.1: The scheme of agent-environment interaction.

2.1 Markov decision process

Markov decision process (MPD) (Bellman, 1957) is a mathematical framework from the field of optimal control theory which can be used to formalize decision and control problems in agent-environment interaction. It consists of quintuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the state space
- $\mathcal{A}$ is the action space
- $\mathcal{T}$ is the transition function $\mathcal{T}(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a)$
- $\mathcal{R}$ is the reward function
- γ is the future discount factor

Subsequently we can define *a stochastic policy* π what is a state dependent probability function $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, such that

$$\pi_t(s, a) = \pi_t(a|s) = p(a_t = a|s_t = s) \quad \text{and} \quad \sum_{a \in \mathcal{A}} \pi(s, a) = 1 \tag{2.1}$$

or *a deterministic policy* π which tells us exactly which action to take in state s

$$\pi_t(s_t) = a_t \tag{2.2}$$

The main goal of the agent is to maximize an expected return in each state

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.3}$$

2.1.1 Value functions

Value functions are functions of state (or state–action pair) which yield the expected return. They are always defined with respect to a particular policy. In other words they express how much reward will the agent accumulate when starting in state s it will follow a particular policy π. For MDPs we can define state–value function $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ formally as

$$V^\pi(s) = \mathbb{E}_\pi \left\{ R_t \mid s_t = s \right\} = \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\} \tag{2.4}$$

and action–value function $Q^\pi(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which defines the value of taking an action a in state s under a policy π as

$$Q^\pi(s, a) = \mathbb{E}_\pi \left\{ R_t \mid s_t = s, a_t = a \right\} = \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\} \tag{2.5}$$

Value functions V^π and Q^π can be approximated using agent's experience. We can accumulate a large number of samples of interactions between the agent and the environment and later use them for estimating the value functions. A fundamental property of the value functions is that they satisfy the following recursive relationship between value of the current state s and the value of the successor state s':

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \tag{2.6}$$

$$= \mathbb{E}_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_t = s\right\} \tag{2.7}$$

$$= \sum_{a}\pi(s,a)\sum_{s'}\mathcal{T}(s,a,s') \left[\mathcal{R}(s,a,s') + \mathbb{E}_{\pi}\left\{\gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_{t+1} = s'\right\}\right] \tag{2.8}$$

$$= \sum_{a}\pi(s,a)\sum_{s'}\mathcal{T}(s,a,s')\left[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')\right] \tag{2.9}$$

Deriving the last recursive form we obtained Bellman equation for V^{π}. Analogically we can obtain Bellman equation for Q^{π} whose final form is

$$Q^{\pi}(s,a) = \sum_{s'}\mathcal{T}(s,a,s')\left[\mathcal{R}(s,a,s') + \gamma\sum_{a'}\pi(s',a')Q^{\pi}(s',a')\right] \tag{2.10}$$

$$= \sum_{s'}\mathcal{T}(s,a,s')\left[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')\right] \tag{2.11}$$

2.1.2 Bellman optimality equations

Now we are going to introduce Bellman optimality equations, where the policy π is no more arbitrary, but it maximizes the value of V^{π} or Q^{π}. Such a policy is the best possible (optimal) policy π^* for given MPD. The MPD could have more optimal policies.
Bellman optimality equations are defined more formally

$$V^*(s) = \max_a \mathbb{E}\left\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right\} \tag{2.12}$$

$$= \max_a \sum_{s'}\mathcal{T}(s,a,s')\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right] \tag{2.13}$$

$$Q^*(s,a) = \mathbb{E}\left\{r_{t+1} + \gamma\max_{a'}Q^*(s_{t+1},a') \mid s_t = s, a_t = a\right\} \tag{2.14}$$

$$= \sum_{s'}\mathcal{T}(s,a,s')\left[\mathcal{R}(s,a,s') + \gamma\max_{a'}Q^*(s',a')\right] \tag{2.15}$$

2.2 Reinforcement learning methods

There are many approaches to solving RL problems. In general, they can be divided from the point of view of whether they use knowledge of the environment (transition function $\mathcal{T}$ and reward function $\mathcal{R}$) into two large groups:

- **Model-based methods** — rely on a model of the environment, either obtained directly from the environment (through the intervention of the designer) or learned, when searching for an optimal policy.
- **Model-free methods** — do not need to know the environment model to find the optimal policy.

According to whether the current policy generating actions is also a source of samples for learning, we divide RL methods into two groups:

- **On-policy methods** evaluate and improve the policy that also makes decisions and drives the agent's behaviour.
- **Off-policy methods** use the behaviour policy to generate actions that can be unrelated to the policy that is evaluated and improved which is called estimation policy.

2.2.1 Dynamic programming

Dynamic programming (DP) (Bellman, 1954) is a mathematical optimization method and a programming method which can be used to solve control problems i.e. to find an optimal policy for environment formalized as MPD. The main assumption for using DP is the precise knowledge of the model of environment. In other words the agent needs to know the transition function $\mathcal{T}$ and the reward function $\mathcal{R}$ and then it can apply some of the DP methods to compute optimal policy. It is a model–based method without bootstrapping (using one or more estimated values in the update step for the same kind of estimated value). Thus the application of DP is computationally expensive but it provides good theoretical foundation for other methods which are usually employed.

Policy iteration is a method of DP to find an optimal policy that constantly switches between two processes. The first is policy evaluation which refers to computation of the value function V^{π} for arbitrary policy π. The second process is policy improvement which searches for a better policy π' evaluated by value function V^{π} and then again evaluates a new policy π'. Such iterative algorithm yields a finite sequence of improving policies which converge to an optimal policy π^* because finite MPD has a finite number of policies. This algorithm manipulates the policy directly.

Value iteration is another way for finding an optimal policy by calculating an optimal value function and omitting full policy evaluation. This part is replaced by truncated policy evaluation that performs only one backup of each state. By turning the Bellman optimality equation to update rule we obtain

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{T}(s,a,s')\left[\mathcal{R}(s,a,s') + \gamma V_k(s')\right] \tag{2.16}$$

The important result is that the greedy policy is guaranteed to be optimal, as the value function converges to an optimal value function. From an empirical experience, the greedy policy is often optimal long before the value function has converged. This method finds an optimal policy indirectly.

2.2.2 Monte Carlo

We briefly mention Monte Carlo (MC) methods which are the class of algorithms solving RL problems by averaging sample returns. Therefore, the agent does not need to know the model of the environment and it is sufficient for it to gather experience in the form of state–to–state transitions. MC methods are thus model–free and do not use bootstrapping. The observation of more and more returns leads to convergence to the expected value. Monte Carlo policy evaluation learns the state–value function V^π for a given policy π. MC estimation of action values leads to estimation of state–action value function Q^π and is used in MC control which is MC version of policy iteration algorithm from dynamic programming mentioned above.

2.2.3 Temporal difference

Temporal difference (TD) methods refer to model-free reinforcement learning algorithms. Their advantage is that the agent does not need to have any knowledge about a model of the environment (unlike DP methods) and does not have to wait until the end of an episode where the return is known (like in Monte Carlo methods). They use a form of bootstrapping – making estimates based on more accurate estimates from the following step so they can be used on-line and in a fully incremental fashion. Selected representatives of TD algorithms are described in more detail in the following two sections.

2.3 Off-policy reinforcement learning methods

A common feature of off-policy algorithms is that they use the behaviour policy to generate actions, and this policy can be unrelated to estimation policy that is evaluated and improved. Thus, it is possible to use even old samples and recycle them in the learning process, which leads to stable learning and reduction of the necessary samples to achieve an optimal policy. From a computational point of view, these are more demanding algorithms, since each step uses several old samples to update the parameters.

2.3.1 Q-Learning

Q-learning algorithm (Watkins and Dayan, 1992) directly approximates the optimal Q^* independently of the followed policy. The next state–action pair is still determined by policy π but is not present in the update rule

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left[r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right] \tag{2.17}$$

It has been proven that for correct convergence it is only required that all pairs continue to be updated. Then the algorithm converges to optimal action–value function Q^* with probability one. We present equations for function approximation version

$$\begin{aligned} \delta_t &= r_t + \gamma \max_a Q_\theta(s_{t+1}, a) - Q_\theta(s_t, a_t) && (2.18) \\ \theta_{t+1} &= \theta_t + \alpha \delta_t \nabla_\theta Q_\theta(s_t, a_t) && (2.19) \end{aligned}$$

2.3.2 Deep Q-Network (DQN)

Deep Q-Network algorithm (Mnih et al., 2015) solved the main disadvantage of Q-Learning, which may suffer from instability and divergence when combined with an nonlinear Q-value function approximation and bootstrapping. The algorithm uses a model Q_θ that approximates the state–value function (critic) and the so-called target model Q_{θ^-}, which serves as a source of target values when updating the critic. The authors introduced two innovations:

- *Experience replay buffer* $\mathcal{D}_t = \{e_0, ..., e_t\}$ (functions as a queue), which stores a certain number of samples $e_t = (s_t, a_t, r_t, s_{t+1})$ and then random minibatches are selected from it to be used when updating the critic estimating Q^π function. In this way, one sample can be used more than once, which increases the efficiency of sampling and decorrelates the samples due to their random selection.

- *Periodically updated target model* Q_{θ^-} against which the critic is optimized. This model is frozen C steps and then its weights are cloned (hard update) from the critic's current weights. This increased the stability of the algorithm, which eliminates short-term oscillations.

The update of the critic is defined as

$$\theta_{t+1} = \theta_t - \alpha_Q \nabla_\theta \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \Big[Q_\theta(s,a) - \Big(r + \gamma \max_{a'} Q_{\theta^-}(s,a') \Big) \Big]^2 \tag{2.20}$$

where B is a minibatch sampled from the experience replay buffer $\mathcal{D}$.

2.3.3 Deep Deterministic Policy Gradient (DDPG)

The DDPG algorithm (Lillicrap et al., 2015) combines the Deterministic policy gradient (DPG) (Silver et al., 2014) algorithm and DQN, introduces an experience replay buffer for DPG, and extends both algorithms to the space of continuous actions. The DDPG model consists of an actor μ_ψ and a critic Q_θ and a target networks for the actor μ_{ψ^-} and critic Q_{θ^-}, which are updated by the so-called soft update or Polyak averaging $\theta^-_{t+1} = \tau\theta^-_t + (1-\tau)\theta$ where $\tau \ll 1$ (analogously for an actor). The critic is updated according to the rule

$$\theta_{t+1} = \theta_t - \alpha_Q \nabla_\theta \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \left[Q_\theta(s,a) - (r + \gamma Q_{\theta^-}(s', \mu_{\psi^-}(s'))) \right]^2 \tag{2.21}$$

and the following equation holds for the actor

$$\psi_{t+1} = \psi_t + \alpha_\mu \nabla_\psi \frac{1}{|B|} \sum_{s \in B} Q_\theta(s, \mu_\psi(s)) \tag{2.22}$$

where B is a minibatch sampled from the experience replay buffer $\mathcal{D}$.

Distributed Distributional DDPG (D4PG) (Barth-Maron et al., 2018) applies a set of improvements on DDPG to make it run in the distributional fashion. Multi-agent DDPG (MADDPG) (Lowe et al., 2017) extends DDPG to an environment where multiple agents are coordinating to complete tasks with only local information.

2.4 On-policy reinforcement learning methods

These methods evaluate and improve the policy that also makes decisions and drives the agent's behaviour. They are less efficient in the use of samples (they cannot recycle old samples), but thanks to the introduction of the technique of parallel collection of samples by several instances of the actor, they are able to eliminate this disadvantage from a technical point of view. Also, thanks to more advanced policy update constraints that were introduced later (Schulman et al., 2015a, 2017), their stability has increased and their performance is comparable to off-policy methods.

2.4.1 SARSA

SARSA (Rummery and Niranjan, 1994) is an on-policy control algorithm estimating the action–value function $Q^\pi(s,a)$ for current policy π and for all states s and actions a. Instead of using transitions from state to state, in this algorithm we use transitions from state–action pair to successive state–action pair which also form Markov chains with a reward. The next state–action pair is chosen according to current policy π and therefore, we are talking about the on-policy method. The update rule is defined as

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha\left[r_t + \gamma Q_t(s_{t+1},a_{t+1}) - Q_t(s_t,a_t)\right] \tag{2.23}$$

If s_{t+1} is a terminal state, then $Q(s_{t+1},a_{t+1})$ is defined as zero. Its name is derived from the quintuple $(s_t,a_t,r_{t+1},s_{t+1},a_{t+1})$ involved in the updating process describing the transition from one state–action pair to another.

The control is quite straightforward. As the algorithm iteratively estimates action-value function Q^π, at the same time the policy π greedily selects actions using current estimates. It has been proven that SARSA converges to an optimal action–value function (and therefore, to an optimal policy) when each state–action pair is visited an infinite number of times. There also exists possibility to use function approximation as done in TD(0). The tabular update is then changed as follows

$$\delta_t = r_t + \gamma Q_\theta(s_{t+1},a_{t+1}) - Q_\theta(s_t,a_t) \tag{2.24}$$

$$\theta_{t+1} = \theta_t + \alpha\delta_t\nabla_\theta Q_\theta(s_t,a_t) \tag{2.25}$$

2.4.2 REINFORCE

The policy π_θ induces a distribution of trajectories that generates a reward distribution. REINFORCE (Monte-Carlo policy gradient) algorithm tries to update the policy parameters θ using the gradient of an expected reward in an episode $\nabla_\theta\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} r_t\right]$. After applying the policy gradient theorem (Sutton and Barto, 2018), we get an update rule for the policy parameters

$$\theta_{t+1} = \theta_t + \alpha R_t \nabla_\theta \ln \pi_\theta(s_t,a_t) \tag{2.26}$$

The algorithm thus tries to strengthen actions that lead to a higher reward (return R_t here serves as a weighting of the update of the given action probability). A widely used variation of REINFORCE is to subtract a baseline value from the return to reduce the variance of gradient estimation while keeping the bias unchanged. Good candidates for bias are value functions Q^π, V^π and an advantage function A^π defined as

$$A^\pi(s_t,a_t) = Q^\pi(s_t,a_t) - V^\pi(s_t) \tag{2.27}$$

2.4.3 Actor–Critic (AC)

The main components of AC methods are the policy model (actor) and the value function (critic). This is a straightforward extension of the REINFORCE algorithm, where bootstrapping is introduced by adding a critic which already learns to estimate return R_t (by value function) instead of waiting for the completion of the trajectory and its evaluation. The critic updates the value function parameters θ and depending on the algorithm it could be an action–value function Q^π or a state–value function V^π. The update rule for actor model is similar to the REINFORCE algorithm, namely

$$\psi_{t+1} = \psi_t + \alpha\, Q_\theta(s_t, a_t) \nabla_\psi \ln \pi_\psi(s_t, a_t) \tag{2.28}$$

The signal driving the learning of critic is the TD error (in this case exactly the same as for SARSA algorithm)

$$\begin{aligned} \delta_t &= r_{t+1} + \gamma\, Q_\theta(s_{t+1}, a_{t+1}) - Q_\theta(s_t, a_t) & (2.29) \\ \theta_{t+1} &= \theta_t + \alpha\, \delta_t\, \nabla_\theta Q_\theta(s_t, a_t) & (2.30) \end{aligned}$$

Actor–critic methods can be very efficient because the policy is directly represented by a separate structure. There is no need to find an action with a maximum value as it is in employing Q-learning or SARSA independently in control tasks.

2.4.4 Advantage Actor–Critic (A2C)

Advantage Actor–Critic (A2C) algorithm (Mnih et al., 2016) is a slightly modified version of the traditional actor–critic algorithm. Several instances of the actor run each in its own thread and collect samples that are stored in the trajectory buffer. Several instances of the actor run each in its own thread and collect samples that are stored in the trajectory buffer $\mathcal{T}$. Subsequently, these samples are used for an update of the critic (which estimates the V^π function) and for an update of the actor whose actions are weighted by the advantage function A^π, which is calculated using the Generalized advantage estimation algorithm (GAE) (Schulman et al., 2015b). The parallel running of the actor on separate instances of the environment enables fast collection of samples and also contributes to the stability of learning, which partially levels the advantage of off-policy algorithms. Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) is a version of A2C with asynchronous parallel updates.

2.4.5 Proximal Policy Optimization (PPO)

The main idea of the PPO algorithm (Schulman et al., 2017) is that the policy parameters should not be updated in one step so that a large policy change occurs. A similar idea was already used in the TRPO algorithm (Schulman et al., 2015a) but it was implemented using the Kullback–Leibler divergence constraint. PPO simplified this limitation by introducing a clipped surrogate objective while maintaining similar performance. The critic is updated by the same rule as in the case of A2C. The algorithm also uses GAE to estimate the advantage function A^{π}. It is used in the clipped surrogate objective for the actor, which has the following form

$$r(\psi) = \frac{\pi_{\psi}(a_t|s_t)}{\pi_{\psi_{old}}(a_t|s_t)} \tag{2.31}$$

$$g(r,\epsilon) = \text{clip}\left(r, 1-\epsilon, 1+\epsilon\right) \tag{2.32}$$

$$\psi_{t+1} = \arg\max_{\psi} \frac{1}{|\mathcal{T}|T} \sum_{\tau\in\mathcal{T}} \sum_{t=0}^{T} \min\Big(r(\psi)A^{\psi'}(s_t,a_t), g(r(\psi),\epsilon)A^{\psi'}(s_t,a_t)\Big) \tag{2.33}$$

where T is the length of a trajectory τ from the trajectory buffer $\mathcal{T}$. Function *clip* stands for clipping function. Hsu et al. (2020) identified several cases where PPO fails and offered alternative surrogate objectives that should improve the performance and stability of the PPO algorithm.

For our research, we chose the PPO algorithm because of its computational efficiency and universality. It can solve tasks in environments with both discrete and continuous action space by simply changing the last layer of an actor (head). At the same time, it is not necessary to explicitly solve any exploration of the environment, as it inherently generates a stochastic policy. In addition, it is stable and converges well, its only drawback is the larger number of samples, which could be solved on a technical level by parallelizing the environments.

Chapter 3

Self-supervised learning

Self-supervised learning (SSL) is a paradigm of machine learning, where the model does not have labels provided by a human, but creates them on its own based on simple assumptions made by the designer. A big advantage of SSL is that it uses much more information provided by the data. While in the classification task (supervised learning) the model receives e.g. 10 bits (if it classifies into 10 classes) and even in a reinforcement learning task it is fractions of bits per step (0.001 if it encounters a reward (that is binary) after 1000 steps), so SSL algorithms can extract millions of bits of information from one sample.[1] SSL is currently used in most cases for pre-training the models, which are subsequently trained for some specific downstream tasks including language modeling, image classification, or speech recognition. For example it can be a pre-training of the ResNet network (He et al., 2016) and later it can be used for a classification task or as a source of representations for a detector, etc.

The most often used is the discriminative approach that aims at grouping similar samples closer and diverse samples far from each other. Therefore, it is necessary to introduce a similarity metric of two feature vectors in order to measure how close they are to each other.

3.1 Contrastive methods

Contrastive learning (Chopra et al., 2005) is a method of self-supervised learning used to learn the general features by teaching the model which data points are similar or different using positive and negative samples. The basic intuition behind contrastive learning paradigm is: **push original and augmented images closer to each other and push original and negative images away**. Depending on the way input data is selected in the context of self-supervised learning, we can identify several architectures

[1]https://www.slideshare.net/rouyunpan/deep-learning-hardware-past-present-future

listed below.

End-to-End Learning architecture works with fully differentiable models that can be trained using gradient methods. This architecture prefers large batches, where apart from the input sample and its augmented version (positive sample), other samples are considered negative. The architecture consists of two encoders: *Query* encoder Q and *Key* encoder K. The task of both encoders is to create a representation (feature vector) for the original sample x (encoder Q) and a positive or negative sample x' (encoder K). Positive and original samples move closer to each other in the feature space, while negative samples move away from the original ones. The representation of the samples z and z' are compared according to some similarity metric $sim(z, z')$ (e.g. cosine similarity), which will be mentioned later in the text, and is used in loss function $\mathcal{L}$. Simple scheme is presented in Fig. 3.1. An example of such an architecture is the SimCLR algorithm (Chen et al., 2020a), which has been successfully used for visual representations, or a model based on Contrastive Predictive Coding (CPC) (van den Oord et al., 2018) and their various modifications as Bachman et al. (2019); Henaff (2020); Hjelm et al. (2018); Ye et al. (2019). The number of negative samples available in this approach is coupled with the batch size as it accumulates negative samples from the current batch and it represents the main disadvantage of this architecture.

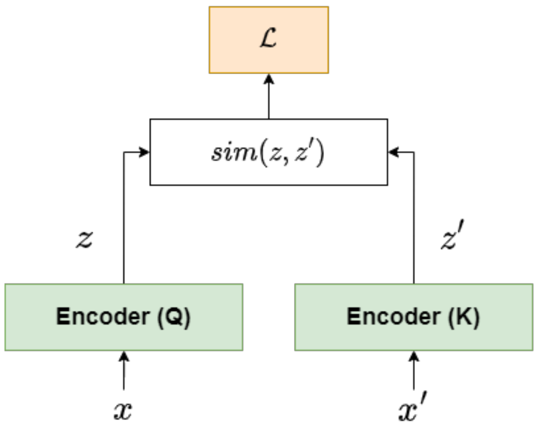


Figure 3.1: End-to-End architecture.

Memory Bank architecture maintains a buffer (memory bank) to accumulate a large number of feature representations of samples that are used as negative samples during training. With this approach, they try to eliminate the shortcoming of the previous architecture, which is limited by its batch size. For this purpose, a memory bank is created that stores feature vectors and is regularly updated. Memory bank stores for each sample s its representation z (e.g. encoder output). If the sample is seen more than once, the representation z is gradually updated by an exponential moving average. Thus, the average of the representation for the sample s from different time points is created. The simple scheme is presented in Fig. 3.2. An example of such an architecture is the PIRL model (Misra and Maaten, 2020), which uses a memory bank for training on visual data, or a model of Wu et al. (2018) that also uses this architecture in the context of contrastive learning in connection with a nonparametric variant of softmax classifier that is more scalable for big data applications. The disadvantage of this architecture is the potential computational complexity for maintaining memory banks. It can be challenging to quickly update stored representations.

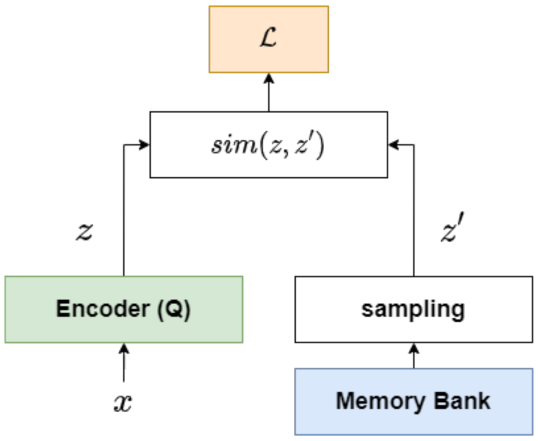


Figure 3.2: Memory Bank architecture.

Momentum Encoder is a module that replaces the memory bank to solve the mentioned shortcoming that it can be computationally demanding to update the memory bank. The momentum encoder uses a queue into which the latest mini-batch is queued and the oldest mini-batch is dequeued. This enables building a large and consistent dictionary on-the-fly that facilitates contrastive unsupervised learning. Momentum encoder module shares parameters with the Q encoder, but it is not trained after each mini-batch, but its parameters are updated using a soft-update rule, similar to the DDPG algorithm (see Sec. 2.3.3). The momentum update makes the momentum encoder evolve more smoothly than the Q encoder. The simple scheme is presented in Fig. 3.3. A representative of this architecture is the Momentum Contrast (MoCo) method (He et al., 2020). The advantage of this architecture is that there is no need to train two encoders or maintain a memory bank, which makes it computationally and memory efficient. MoCo V2 combined MLP projection head and stronger data augmentation from SimCLR, achieving even better transfer performance with no dependency on a very large batch size.

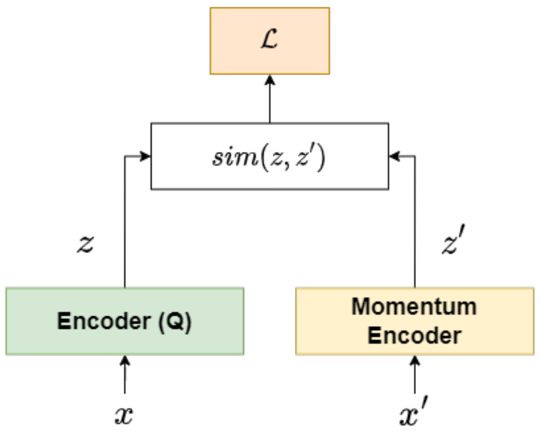


Figure 3.3: Momentum Encoder architecture.

Clustering Feature Representations architecture follows an end-to-end approach with two encoders that share parameters, but instead of using an instance-based contrastive approach, they utilize a clustering algorithm to group similar features together. The goal is not only to bring the current input and the positive sample closer to each other, but to ensure that similar features form clusters together. For example in an image task, features representing cats and dogs should be closer to each other than features representing houses or cars. This architecture tries to solve the shortcoming of all instance-based contrastive methods, where each instance is taken as a separate class. If a cat enters the input and another cat is selected as a negative sample, the representation model of these two samples tries to delay, although it should work in the opposite way. This problem is implicitly addressed by a clustering-based approach. A simple scheme is presented in Fig. 3.4. Representative of this architecture is the DeepCluster (Caron et al., 2018) which iteratively clusters features via k-means and uses cluster assignments as pseudo labels to provide supervised signals. Swapping Assignments between Views (SWaV) algorithm (Caron et al., 2020) computes a code from an augmented version of the image and tries to predict this code using another augmented version of the same image.

In the next subsections, we present several the most frequently used contrastive loss functions.

3.1.1 Cosine similarity

Cosine similarity is the most common similarity metric in contrastive setup that acts as a basis for different contrastive loss functions. The cosine similarity of two feature

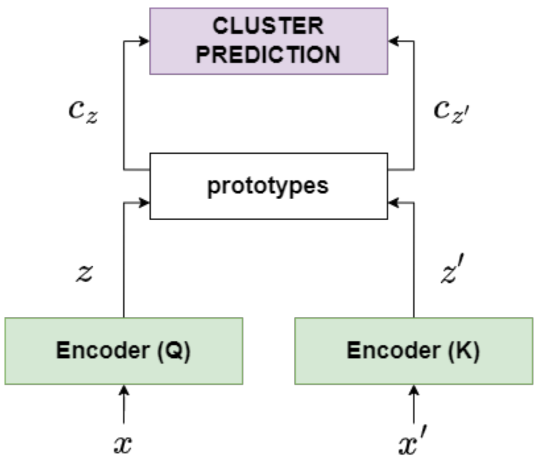


Figure 3.4: Clustering Feature Representations architecture.

vectors z and z' is defined as the inner product of both normalized vectors

$$\cos(z, z') = \frac{z^{\intercal} \cdot z'}{\|z\|\|z'\|} \tag{3.1}$$

3.1.2 Noise Contrastive Estimation (NCE)

Noise Contrastive Estimation (Gutmann and Hyvärinen, 2010) is a way of learning a data distribution by comparing it against a noise distribution, which we define. This allows us to cast an unsupervised problem as a supervised logistic regression problem. It is commonly used in cases where the probability distribution of the training data is difficult to model directly. NCE allows us to estimate the model parameters without directly computing the partition function of the model, which can be intractable. The partition function is a normalizing constant that is required to ensure that the probability distribution sums to one over all possible outcomes. NCE reduces the complexity of optimization by replacing the multi-class classification problem to a binary classification problem by treating the correct class as the positive example and randomly sampling noise classes as negative examples. Let x be the positive (target) sample from $P(x|C = 1; \theta) = p_\theta(x)$ and x' be the negative (noise) sample from $P(x|C = 0) = q(x')$. The model is then trained to distinguish between the positive and negative samples using a logistic regression or a neural network which models the logit l of a sample from the target data distribution instead of the noise distribution

$$l_\theta(u) = \log p_\theta(u) - \log q(u) \tag{3.2}$$

The NCE loss is then defined as cross-entropy loss function

$$\mathcal{L}_{\text{NCE}} = \frac{1}{N} \sum_{i=1}^{N} [\log \sigma(l_\theta(x_i)) + \log(1 - \sigma(l_\theta({x'}_i)))] \tag{3.3}$$

where $\sigma(\cdot)$ is the sigmoid function that converts logits to probabilities. The primary difference in implementation between NCE and Negative Sampling is that in NCE, the probability that a sample came from the noise distribution is explicitly accounted for, and the problem is cast as a formal estimate of the log-odds ratio that a particular sample came from the real data distribution instead of the noise distribution. NCE is closely related to Generative Adversarial Networks (Mirza et al., 2014), since it can be interpreted as the discriminator part of a GAN where the generative network is fixed.

3.1.3 InfoNCE

The InfoNCE (van den Oord et al., 2018) loss is inspired by NCE and it uses categorical cross-entropy loss to identify the positive sample amongst a set of unrelated noise samples. It is based on the concept of mutual information between two random variables. In contrastive setup, the mutual information between two representations of the original and positive sample is maximized, while the mutual information between representations of original and noise (negative) samples is minimized. Given a context vector c, the positive sample should be drawn from the conditional distribution $p(x|c)$, while negative samples are drawn from the proposal distribution $p(x)$, independent from the context c. The loss function is then defined as

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}\left[\log \frac{f(x,c)}{\sum_{x' \in X} f(x',c)}\right] \tag{3.4}$$

where x' is negative sample and $f(\cdot)$ is the scoring (similarity) function where $f(x,c) \propto \frac{p(x|c)}{p(x)}$. This connects it to mutual information defined as

$$I(x;c) = \sum_{x,c} p(x,c) \log \frac{p(x|c)}{p(x)p(c)} = \sum_{x,c} p(x,c) \log \frac{p(x|c)}{p(x)} \tag{3.5}$$

InfoNCE was succsefully used in algorithms focused on visual tasks like SimCLR (Chen et al., 2020a), SimCLRv2 (Chen et al., 2020b) or SupCon (Khosla et al., 2020). Jozefowicz et al. (2016) proposed the ranking-based version of NCE, which is equivalent to InfoNCE.

3.1.4 Contrastive loss

Contrastive loss (Chopra et al., 2005) takes the pairs of examples as input and trains a model to predict whether two inputs are from the same class or not. Let $f(\cdot)$ be some encoder or kernel returning representation for an input sample. Specifically, the loss can be written as

$$\mathcal{L}_{\text{cont}}(x_i, x_j) = \mathbf{1}\{c_i = c_j\} \|f_i - f_j\|_2^2 + \mathbf{1}\{c_i \neq c_j\} \max(0, \epsilon - \|f_i - f_j\|_2)^2 \tag{3.6}$$

where c_i and c_j are the corresponding classes for inputs x_i and x_j, f_i and f_j are abbreviation for $f(x_i)$ and $f(x_j)$ respectively, and ϵ is a margin parameter imposing the distance between examples from different classes to be larger than ϵ.

3.1.5 N-pair loss

In the N-pair loss function, each training example is paired with several other examples from the same class, N is the number of training examples. Contrastive loss is a special case of N-pair loss where $N = 1$, as well as Triplet loss function (Schroff et al., 2015) is special case of N-pair loss where $N = 2$. Triplet loss was used in face recognition and clustering at different angles and positions. It directly learned the mapping from face images to a compact Euclidean space where distances directly corresponded to a measure of face similarity. The triplet loss function teaches the model to maximize the distance between the original (called anchor in this case) and the negative sample, and vice versa to minimize the distance between the original and the positive sample. The loss function is defined as

$$\mathcal{L}_{\text{triplet}}(x, x^+, x^-) = \sum_{x \in X} \max\left(0, \|f(x) - f(x^+)\|_2^2 - \|f(x) - f(x^-)\|_2^2 + \epsilon\right) \tag{3.7}$$

where $f(\cdot)$ is an encoder or a kernel returning representation for an input sample. In order to steadily improve the model, it is critical to choose a challenging negative sample.

The multi-class version of the N-pair loss function (Sohn, 2016a) extends the idea of N-pair loss to the multi-class classification setting by considering all positive pairs and negative pairs across all classes. We can consider it as generalization of Triplet loss that includes comparison with multiple negative samples, thus getting rid of the mentioned disadvantage. We set $(N+1)$-tuplet of training samples including one original sample, one positive sample and $(N-1)$ negative samples $(x, x^+, x_1^-, ..., x_{N-1}^-)$. The loss is then defined as

$$\mathcal{L}_{\text{N}-\text{pair}}(x, x^+, \{x_i^-\}_{i=1}^{N-1}) = \log\left(1 + \sum_{i=1}^{N-1} \exp\left(f(x) \cdot f(x_i^-) - f(x) \cdot f(x^+)\right)\right) \tag{3.8}$$

where $\cdot$ is vector dot product. If we only sample one negative sample per class, it is equivalent to the softmax loss for multi-class classification.

3.2 Non-contrastive methods

Non-contrastive methods do not need negative samples for learning, which is one of the disadvantages of contrastive methods. They only work with original and positive samples.

A typical architecture representing non-contrastive learning is an autoencoder (AE) (Kramer, 1992), which tries to reconstruct the input at the output, while in its "bottleneck" a compressed representation (latent variable or a feature vector) is created that captures the underlying structure of the data. That, for example, can be used in further training or for generating synthetic data. Today, AEs are successfully used in representation learning and have their place in RL as well. Since the feature space AE is formed using only the reconstruction loss function, the metric of this space does not preserve the similarity of the inputs very much. Therefore, an improved version of the variational autoencoder (VAE) (Kingma and Welling, 2013) was introduced. The VAE is trained using a variational lower bound on the log-likelihood of the data. This lower bound is derived using the variational inference framework, and involves maximizing a lower bound on the expected log-likelihood of the data with respect to a probabilistic model of the latent variables. Last but not least, it is also possible to use a pair of encoders (either with shared or separated weights), similar to the case of the End-to-End architecture described in Sec. 3.1.

3.2.1 BYOL

The main idea of Bootstrap Your Own Latent (BYOL) (Grill et al., 2020) is to train two identical networks with the same architecture. One is called an online network parameterized by θ, the other a target network parameterized by ξ. The online network is trained on input samples and the target network using soft-update weights (Polyak averaging). Both networks consist of three modules: an encoder f, a projector g and a predictor q.

At the beginning of the training, two augmented versions v, v' are generated from the input x using a set of different transformations that are chosen randomly. Subsequently, their representations are generated using the encoders of both networks: $y_\theta = f_\theta(v)$ and $y' = f_\xi(v')$. These are then mapped in the latent space using projectors: $z_\theta = g_\theta(y_\theta)$ and $z' = g_\xi(y')$. Finally, the online network returns a prediction $q_\theta(z_\theta)$. The loss function $\mathcal{L}_v$ is defined as

$$\mathcal{L}_v = \text{MSE}\left(\frac{q_\theta(z_\theta)}{|q_\theta(z_\theta)|}, \frac{z'}{|z'|}\right) \tag{3.9}$$

The loss function $\mathcal{L}_{v'}$ is defined analogously, but the inputs are reversed, v' is at the input of the online network and v is at the input of the target network. The final loss function then takes the form

$$\mathcal{L}_{\text{BYOL}} = \mathcal{L}_v + \mathcal{L}_{v'} \tag{3.10}$$

This encourages the online network to learn a shared representation of data that is robust to different augmentations, while the target network serves as a slowly moving target for the online network. Unlike most popular contrastive learning based approaches, BYOL does not use negative pairs. Most bootstrapping approaches rely on pseudo-labels or cluster indices, but BYOL directly boostraps the latent representation.

3.2.2 Barlow Twins

Barlow Twins algorithm (Zbontar et al., 2021) learns a shared representation of data by minimizing a new loss function that encourages the cross-correlation between the feature vectors of two randomly augmented views of the same input image to be close to a target cross-correlation matrix. Specifically, the loss function is designed to decorrelate the feature vectors of the two views while preserving the information content in the data. It naturally avoids trivial solutions (i.e. constant representations), and is robust to different training batch sizes. The algorithm creates two augmented versions y and y' of the input x and maps them to their feature vectors z and z' using an encoder. The loss function is defined as

$$\mathcal{L}_{\mathrm{BT}} = \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{i \neq j} C_{ij}^2 \tag{3.11}$$

where λ is the scaling parameter, C is the cross-correlation matrix computed between outputs from two identical networks along the batch dimension. C_{ij} is the cosine similarity between the network output vectors z_i and z'_j, where i, j are the dimension indices. Feature vectors z, z' are normalized along the batch dimension. The first term of the loss function ensures invariance, while the second term reduces redundancy in feature vectors.

3.2.3 VICReg

VICReg (Bardes et al., 2022) means variance, invariance, covariance regularization. It is an improved successor of the Barlow Twins algorithm, which uses the three mentioned regularization losses to create a suitable feature space. The model has one encoder and works the same as Barlow Twins, it receives two augmented versions of the input sample and maps them to feature vectors. The variance regularization term $\mathcal{L}_{v(Z)}$ is defined as a hinge function on the standard deviation of the features along the batch dimension

$$\mathcal{L}_{v(Z)} = \frac{1}{d} \sum_{j=1}^{d} \max(0; \tau - \sigma(Z_j)) \tag{3.12}$$

where d is the dimensionality of the feature space, Z_j is j-th feature vector from the batch Z, σ is the actual standard deviation and $\tau = 1$ is a constant target value for the standard deviation. The covariance regularization term $\mathcal{L}_{c(Z)}$ is defined as the sum of the squared off-diagonal coefficients of the covariance matrix $C(Z)$

$$\mathcal{L}_{c(Z)} = \frac{1}{d} \sum_{i \neq j} [C(Z)]_{i,j}^2 \tag{3.13}$$

The invariance criterion $\mathcal{L}_{s(Z,Z')}$ between two batches Z and Z' is defined as the mean-squared Euclidean distance between each pair of feature vectors

$$\mathcal{L}_{s(Z,Z')} = \frac{1}{d} \sum_{i=1}^{d} \|Z_i - Z'_i\|_2^2 \tag{3.14}$$

The overall loss $\mathcal{L}_{\text{VIC}}$ then takes the form

$$\mathcal{L}_{\text{VIC}} = \lambda \mathcal{L}_{s(Z,Z')} + \mu \left[\mathcal{L}_{v(Z)} + \mathcal{L}_{v(Z')}\right] + \nu \left[\mathcal{L}_{c(Z)} + \mathcal{L}_{c(Z')}\right] \tag{3.15}$$

where λ, μ and ν are the scaling parameters.

In our research, we decided to test one method from both SSL categories. We chose an algorithm using N-pair loss function as a representative of contrastive learning and VICReg as a representative of a non-contrastive approach. In terms of computational and memory requirements, non-contrastive methods appear to be more advantageous, which was also confirmed in the results of our experiments.

Chapter 4

Intrinsic motivation

Motivation represents a complex of psychological phenomena such as novelty, surprise, incongruity or challenge. Therefore, various theories have been developed, out of which we will briefly describe some:

- Theory of drives (Hull, 1943) is based on a statement that humans are looking for options to secure their basic needs, to explore the environment (Montgomery, 1954), or look for ways to control it (Harlow, 1950).
- Theory of effectance (White, 1959) is described as the desire for effective interaction with the environment.
- Theory of cognitive dissonance (Festinger, 1962) explains motivation as a reduction in the differences between the experience gained and the expectations that were created in internal cognitive structures.
- Optimal incongruity theory (Hunt, 1965) proposes that a person is motivated by stimuli that differ from the standard stimuli the person has already experienced.
- Theory of "Flow" (Csikszentmihalyi, 1991) assigns the greatest motivation to solving problems with optimal difficulty. In simple tasks, a person begins to get bored quickly, and on the other hand, without the ability to find a solution a person becomes frustrated.
- Synchronicity detection is a crucial mechanism in object interaction skills (Watson, 1972) or self-modelling (Rochat and Striano, 2000).

In formalizing the concept of motivation, we can divide it into *external* and *internal*, depending on the mechanism that generates motivation for the agent. If the source of motivation comes from outside, we are talking about *external* motivation, and it is always associated with a particular goal in the environment. If the motivation is generated within the structures that make up the agent, it is an *internal* motivation.

Another dimension for the differentiation, extrinsic or intrinsic, is less obvious (see also Morris et al. (2022)). *Extrinsic* motivations pertain to behaviors whenever an activity is done in order to attain some separable outcome (Oudeyer and Kaplan, 2009). Some variability exists in this context, since these behaviors can vary in the extent to which they represent self-determination (see the details in Ryan and Deci (2000)). On the other hand, *intrinsic* motivation is defined as doing an activity for its inherent satisfaction rather than for some separable consequence (or instrumental value). It has been operationally defined in various ways, backed up by different psychological theories, which point to some uncertainty in what intrinsic motivation exactly means. Nevertheless, Baldassarre (2019) offers a solution of an operational definition of intrinsic motivation as processes that can drive the acquisition of knowledge and skills in the absence of extrinsic motivations. Furthermore, the author proposes (and explains why) a new term of *epistemic motivations* as a suitable substitution for intrinsic motivations. Despite some uncertainty, intrinsic motivation has remained a well coined term in the literature.

Intrinsic motivation has a strong biological basis (Morris et al., 2022; Ryan and Deci, 2000) since it is observed among higher animals, especially in humans, keeping them engaged in various activities. Intrinsic motivation appears early in life and guides the biological agents during their entire life. Intrinsic motivation is considered one of the prerequisites for open-ended (or, life-long) learning. If we want to achieve this capacity with artificial agents (Parisi et al., 2019), we have to master this first step and equip the agents with an ability to generate their own goals and acquire new skills. Therefore, computational approaches concerned with intrinsic motivations and open-ended development provide the potential in this direction leading to more intelligent systems, in particular those capable of improving their own skills and knowledge, autonomously and indefinitely (Baldassarre, 2019; Baldassarre et al., 2014).

Intrinsic motivation is a crucial factor that helps the agent not only to remain in open-ended learning hence solving different tasks (Parisi et al., 2019), but it also helps to solve single difficult tasks with extremely sparse rewards.

There exists a variety of approaches aiming to use intrinsic motivation signal for agent learning. Information-theoretic view on intrinsic motivation is well represented in the literature, involving the concepts of novelty, surprise and skill-learning. The recent review (Aubret et al., 2023) suggests that novelty and surprise can assist the building of a hierarchy of transferable skills which abstracts dynamics and makes the exploration process more robust. In this context, abstraction is a key feature of the agent's architecture where it makes sense to introduce learning mechanisms to enforce formation of proper internal representations that lead to improved agent's performance.

In the following subsections we present different approaches to *intrinsic* motivation. The main categorization is influenced by earlier work of Oudeyer and Kaplan (2009). The computational approaches to *intrinsic* motivation can be divided into three main categories:

1. *Knowledge-based* approach is focused on exploration of the environment and contains prediction-based, novelty-based, information-based and learning progress methods. This approach is based on the theory of drives, theory of cognitive dissonance and optimal incongruity theory.

2. *Competence-based* approach motivates the agent to achieve higher level of performance in the environment. Which means to acquire desired actions to achieve self-generated goals. Its psychological basis includes theory of effectance and the theory of flow.

3. *Morphological* approach is based on the synchronicity detection theory and motivates the agent to stay in more or less (depending on the motivation signal) stable state according only to its sensory inputs.

In the context of RL, intrinsic motivation can be realized in various ways, but most often it is a new reward signal r_t^{intr} scaled by parameter $\eta > 0$, which is generated by the motivational part of the model (we refer to it as the motivational module) and is added to the external reward r_t^{ext}

$$r_t = r_t^{\text{ext}} + \eta \, r_t^{\text{intr}} \tag{4.1}$$

The general scheme of the agent with motivation can be found in Fig. 4.1.

4.1 Knowledge-based category

During the learning process, the agent must explore the environment to encounter an external reward and to learn to maximize it. This can be ensured by adding noise to the actions, if the policy is deterministic, or it is already its property, if the policy is stochastic. In both cases, we say that these are uninformed environmental exploration strategies. The problem arises if the external reward is very sparse and the agent cannot use these strategies to find the sources of reward. In such a case, it is advantageous to use informed strategies, which include the introduction of intrinsic motivation. The goal of introducing such a new reward signal is to provide the agent with a source of

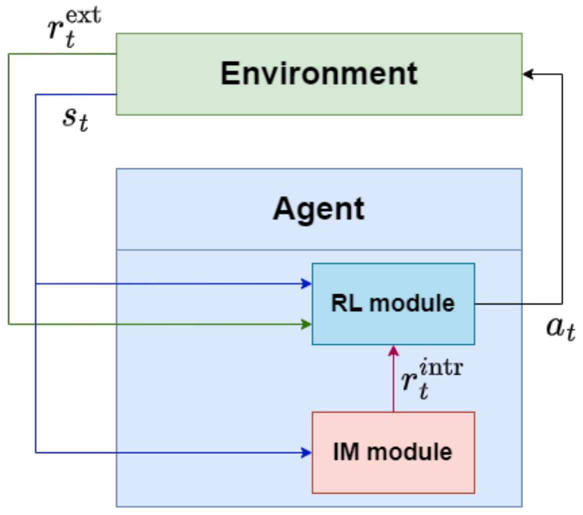


Figure 4.1: Simplified architecture of an agent with intrinsic motivation.

information that is absent from the environment when the reward is sparse, and thus to facilitate the exploration of the environment and the search for an external reward.

This approach is focused on the agent's knowledge of the environment, and rewards are given e.g. for observing situations which were not anticipated by some internal module modelling the environment dynamics or were anticipated and did not occur. It models phenomena such as uncertainty, novelty, surprise and familiarity.

The knowledge-based category focusing on exploration can be divided into *prediction-based*, *novelty-based* and *information-based* approaches (Aubret et al., 2019). But we want to emphasize that the inclusion in these categories is ambiguous, because these approaches are often combined and therefore it is not possible to establish a clear taxonomy of these methods. We analyze this category in more detail than the others, since our method also belongs to it.

4.1.1 Prediction-based approach

In general, these methods operate by learning a function that predicts some target quantity based on the input quantity and use the error of this function e_t (predictive error) as a signal for internal motivation. Lee et al. (2019) provides a theoretical connection of exploration based on the predictive error with a problem of state marginal matching, where the aim is to learn a policy for which the state marginal distribution matches a given target state distribution. General architecture is presented in Figure 4.2.

Schmidhuber (1991) was among the first to use this approach, he introduced a fully recurrent neural model consisting of two components. The first one was the controller

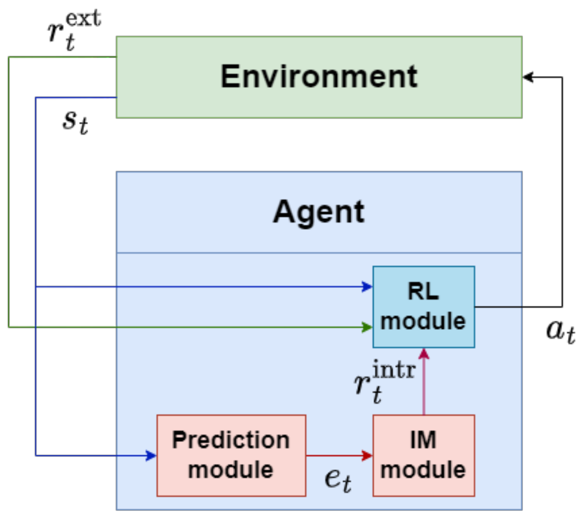


Figure 4.2: General architecture of agent with prediction-based approach to intrinsic motivation.

whose aim was to learn to maximize reward and to minimize pain (there were two input units coding this information). The second component was trying to predict the next input based on the current input and the action of the controller, and the main goal was to build a model of the world. The controller got rewarded for action sequences leading to still unpredictable inputs.

Predictive error motivation methods have at their core a forward model that learns to predict the dynamic consequences of the agent's actions. The difference between the predicted next state and the observed next state then serves as a source of intrinsic motivation. The assumption is that the forward model will more accurately predict frequently experienced transitions between states, so its high prediction error can be a suitable signal informing the agent that it is experiencing something new and unexpected.

Intrinsic Curiosity Module (ICM) (Pathak et al., 2017) extended agent implementing A3C algorithm (controller) with ICM in the role of a predictor, generating intrinsic reward, which is presented in Figure 4.3. It was composed of a feature extraction convolution network, a forward model and an inverse model.

The aim of the feature extraction network was to transform a high-dimensional input s_t into the feature vector $\Phi(s_t)$, motivated by the hypothesis that it could enhance generalization what was supported by results in their work. The inverse model predicted an action $\hat{a}_t$ from the feature vectors $\Phi(s_t)$ and $\Phi(s_{t+1})$ of two successive states. The optimization of the inverse model played supportive role for the learning of feature vectors. The forward model obtained feature vector of the current state

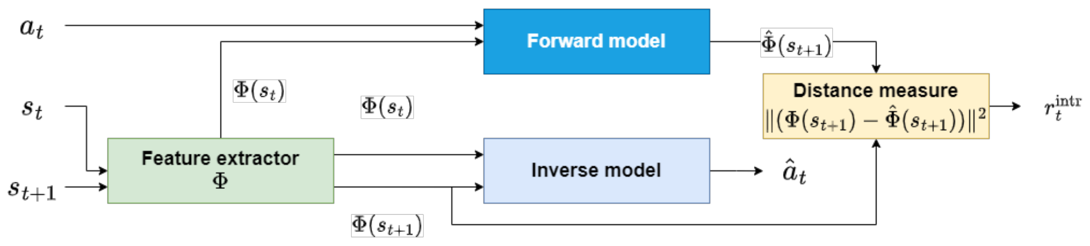


Figure 4.3: Intrinsic Curiosity Module scheme.

$\Phi(s_t)$ and action a_t on the input and learned to predict feature vector of the next state $\hat{\Phi}(s_{t+1})$. Both forward and inverse models were implemented by feedforward neural networks. The intrinsic reward signal was computed as a Euclidean distance between the predicted and the actual next state

$$r_t^{\text{intr}} = \frac{\eta}{2}\|\hat{\Phi}(s_{t+1}) - \Phi(s_{t+1})\|_2^2 \tag{4.2}$$

where $\frac{\eta}{2}$ is the scaling factor.

Intrinsically-motivated self-aware agent (Haber et al., 2018) architecture employees the world-model which solves a dynamics prediction problem. Simultaneously, a self model seeks to predict the world model's loss. Actions are chosen to antagonize the world model, leading to novel and surprising events in the environment.

Model-Based Active eXploration (MAX) (Shyam et al., 2019) uses an ensemble of forward models to plan observing novel events. The measure of novelty is derived from the Bayesian perspective of exploration, which is estimated using the disagreement between the features predicted by the ensemble members.

Plan2Explore (Sekar et al., 2020) is an agent, which during exploration leverages planning to seek out expected future novelty. It's model is using self-supervised learning to create world model and ensemble of predictors (forward models), predicting the next state in the latent space of the world model. Model uncertainty is quantified as the variance over the predicted means of the different ensemble members, and this disagreement is used as the intrinsic reward

$$r_t^{\text{intr}} = \operatorname{Var}\left(\{\mu(w_k, s_t, a_t) \,|\, k \in [1; K]\}\right) \tag{4.3}$$

where w_k are parameters of k-th predictive model and K is a number of predictive models in an ensemble.

Exploration with Mutual Information (EMI) (Kim et al., 2018) extracts predictive signals that can be used to guide exploration based on the forward prediction in the representation space. The model seeks to learn the representation of states $\phi(s_t)$ and the actions $\psi(a_t)$ such that the representation of the corresponding next state $\phi(s_{t+1})$

follow the linear dynamics i.e. $\phi(s_{t+1}) = \phi(s_t) + \psi(a_t)$. Further there is the error model S which takes the state and action as input and estimates the irreducible error under the linear model. The intrinsic reward is defined as

$$r_t^{\text{intr}} = \|\phi(s_t) + \psi(a_t) + S(s_t, a_t) - \phi(s_t + 1)\|^2 \tag{4.4}$$

Predictive surprise motivation rewards the states that occur and were not expected or do not occur and were expected (Oudeyer and Kaplan, 2009). This is a simple definition of surprise which is modelled by this approach. To formalize expectations the second predictor MetaΠ is introduced, which aims to predict the error e_t of the first predictor Π at time t

$$\text{Meta}\Pi(s_t) = \tilde{e_t} \tag{4.5}$$

where $\tilde{e_t}$ is the predicted absolute error of predictor Π. Based on the ratio of predicted error $\tilde{e_t}$ and real error e_t, the intrinsic reward is defined as

$$r_t^{\text{intr}} = C \cdot \frac{e_t}{\tilde{e_t}} \tag{4.6}$$

This reward would provide high values for highly surprising situations. Pecháč and Farkaš (2021) tested this concept on several continuous environments, where was used gating of the predictive error. The source of predictive error was the forward model Π and surprise was modeled as in eq. 4.6, where the meta-critic module MetaΠ estimated the error of the forward model. Such an approach showed a slight improvement in some tasks. The intrinsic reward was defined as

$$r_t^{\text{intr}} = \max(\epsilon_{\text{fm}} \tanh(r_t^{\text{ifm}}), \epsilon_{\text{mc}} \tanh(r_t^{\text{imc}})) \tag{4.7}$$

where ϵ_{fm} and ϵ_{mc} are scaling factors, r_t^{ifm} is equal to error of forward model and r_t^{imc} is surprise generated by meta-critic as

$$r_t^{\text{imc}} = \begin{cases} e_t/\hat{e}_t + \hat{e}_t/e_t - 2, & \text{if } |e_t - \hat{e}_t| > \sigma \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

where $\hat{e}_t$ is the estimation of the forward model error e_t.

Predictive familiarity motivation is an opposite to the previous approach (Oudeyer and Kaplan, 2009). This one motivates the agent to search known predictable states and the intrinsic reward has a form

$$r_t^{\text{intr}} = C \cdot \frac{1}{e_t} \tag{4.9}$$

The predictive error e_t can be substituted by the mean of the error $\left\langle e_t^{\mathcal{R}_n} \right\rangle$ over the states belonging to the same region $\mathcal{R}_n$ and then the intrinsic reward is transformed to

$$r_t^{\text{intr}} = C \cdot \frac{1}{\left\langle e_t^{\mathcal{R}_n} \right\rangle} \tag{4.10}$$

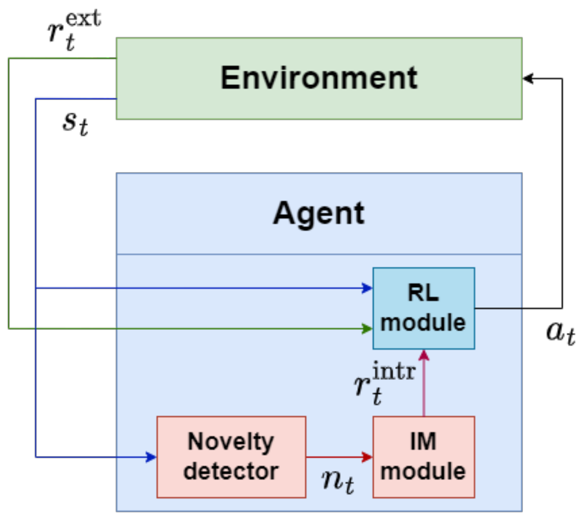


Figure 4.4: General architecture of an agent with novelty-based approach to intrinsic motivation.

4.1.2 Novelty-based approach

The novelty-based approaches monitor the state novelty n_t and the intrinsic signal is based on its value. There is a relatively large group of methods that express novelty in some form of count, where generalized count is a novelty measure that quantifies how dissimilar a state is from those already visited. General architecture is presented in Figure 4.4.

Count-based methods try to count the number of visits to each state and reward the agent for visiting a state that has a low or zero counter value. Model-based Interval Estimation with Exploration Bonus algorithm (Strehl and Littman, 2008) (MBIE-EB) was based on count-based approach and the idea of providing an exploration bonus reward depending on the state-action visit-count was originally under a tabular setting. Estimates of the Q function were realized by solving the MPD model and estimating its $\mathcal{R}$ and $\mathcal{T}$ functions. Subsequently, the internal motivation in the form of the inverted square root of the count of transitions $n(s, a)$ was added to the estimate of the Q function as follows

$$\tilde{Q}(s,a) = \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s'|s,a) \max_{a'} \tilde{Q}(s',a') + \frac{\beta}{\sqrt{n(s,a)}} \tag{4.11}$$

Implementation of this approach requires introduction of a look-up table which will store each count for each state–action pair (or state). Obviously it is limited to environments with a small state space due to the exponential increase in memory consumption as the size of the space state increases. Also in large state spaces there is only a small chance that one state is visited more times and thus the quantity counting visits cannot help to evaluate novelty of the state.

Pseudo-count was introduced by Bellemare et al. (2016) as a solution for problems with implementations of count methods under a tabular setting, and proposed an algorithm for deriving a new quantity denoted as pseudo-count from an arbitrary density model. This enabled to generalize count-based methods to environments with large or continuous state spaces. Pseudo-count function $\hat{N}_n(s)$ and pseudo-count total $\hat{n}$ are derived from two linear equations expressing the property that after observation of instance of state s the density models increase in prediction of the same state s corresponds to unit increase in pseudo-count.

$$\rho_n(s) = \frac{\hat{N}_n(s)}{\hat{n}} \tag{4.12}$$

$$\rho'_n(s) = \frac{\hat{N}_n(s)+1}{\hat{n}+1} \tag{4.13}$$

The variable $\rho'_n(s)$ above is the recoding probability of a state s defined as

$$\rho'_n(s) = \rho(s; s_{1:n}s) \tag{4.14}$$

$$\rho'_n(s) = p_\rho(S_{n+2} = s | S_1 ... S_n = x_{1:n}, S_{n+1} = s) \tag{4.15}$$

where $s_{1:n}s$ means concatenating s to former sequence $s_{1:n}$. The pseudo-count itself is then defined as

$$\hat{N}_n(s) = \frac{\rho_n(s)(1-\rho'_n(s))}{\rho'_n(s)-\rho_n(s)} = \hat{n}\rho_n(s) \tag{4.16}$$

and the intrinsic motivation reads as

$$r_t^{intr} = \frac{\beta}{\sqrt{\hat{N}_n(s)}} \tag{4.17}$$

Pseudo-count has additional important properties making it a good choice for use instead of empirical count: (1) it is almost zero for unvisited states, (2) it grows linearly with empirical counts, (3) it respects the ordering of state frequency, (4) it exhibits credible magnitudes, and (5) is robust w.r.t. the presence of non-stationary data.

Bellemare et al. (2016) used a Context-Tree Switching (CTS) (Veness et al., 2012) density model, while Ostrovski et al. (2017) used PixelCNN and PixelRNN generative models (Van Den Oord et al., 2016) as the state density estimators. Their advantage was that they were able to calculate internal motivation in the online mode.

Count-Based Exploration based on hash functions (Tang et al., 2017) maps states s_t to hash codes $\Phi(s_t)$, which allows to count their occurrences within a hash table. These counts are then used to compute a reward bonus according to the classic count-based exploration theory. The authors compared an approach with a regular hash function (SimHash) with an approach with a learned hash function $\Phi(s_t)$ obtained using the latent space of the auto-encoder. The intrinsic motivation than has form

$$r_t^{\text{intr}} = \frac{\beta}{\sqrt{n(\Phi(s))}} \tag{4.18}$$

where $n(\cdot)$ is counter function (lookup table). ϕ-exploration bonus method (Martin et al., 2017) (ϕ-EB) constructs a density model over the feature space that assigns higher probability to states that share more features with more frequently observed states. This is a straightforward extension of the pseudo-count idea. Their Φ-pseudo-count is then defined as

$$\hat{N}_n^{\Phi}(s) = \frac{\rho_n(\Phi(s))(1 - \rho_n'(\Phi(s)))}{\rho_n'(\Phi(s)) - \rho_n(\Phi(s))} \tag{4.19}$$

where Φ is the feature extractor. The intrinsic motivation reads as

$$r_t^{\text{intr}} = \frac{\beta}{\sqrt{\hat{N}_n^{\Phi}(s)}} \tag{4.20}$$

Count-Based Exploration via Embedded State Space method (Liu et al., 2022) uses an embedding network Φ and an inverse model, which tries to predict an action based on two embedding vectors for the state and its successor. Its loss serves as a signal source that forms the embedding space. When calculating the internal reward, the embedding for the given state is used and encoded with a hash function using SimHash algorithm (Sadowski and Levin, 2007). Next, the counts in the hash function are updated for state embedding. The intrinsic motivation has form

$$r_t^{\text{intr}} = \frac{\beta}{\sqrt{n(\Phi(s))}} \tag{4.21}$$

Count-Based Exploration with the Successor Representation (Machado et al., 2020) shows that the norm of the successor representation, while it is being learned, can be used as a reward bonus to incentivize exploration. In order to better understand this transient behavior of the norm of the SR the authors introduce the substochastic successor representation and show that it implicitly counts the number of times each state (or feature) has been observed. The intrinsic motivation is defined as

$$r_t^{\text{intr}} = \frac{1}{\|\Psi(s_t, \theta^-)\|_1} \tag{4.22}$$

where $\Psi(s_t, \theta^-)$ denotes the successor features of the state s_t parametrized by θ^-. These are parameters of the target network, which is updated less often for stability purpose. Features for the successor representation $\Psi(s_t, \theta^-)$ are created based on the output of the feature extractor Φ. An auxiliary task is used for its training, in which the agent has to predict the next state s_{t+1}.

Distillation-based methods define novelty as a measure of inconsistency between two models, one of which maps the state in the feature space (works as a feature extractor) and the other tries to distill its knowledge. More formally, the target model Φ^{T} generates features, and the learned model Φ^{L} tries to replicate them. This process is called *knowledge distillation*. Intrinsic motivation, expressed as an intrinsic reward, is computed as the distillation error

$$r_t^{\mathrm{intr}} = \|(\Phi^{\mathrm{L}}(s_t) - \Phi^{\mathrm{T}}(s_t))\|^2. \tag{4.23}$$

It is assumed that the learned model will be able to more easily replicate feature vectors for states it has seen multiple times, while new states will induce a large distillation error. The Random Network Distillation (RND) (Burda et al., 2018) model, shown in Fig. 4.5, is a representative of this type of intrinsic motivation. Never-give-up (NGU) framework (Badia et al., 2020) introduces intrinsic reward that combines per-episode and life-long novelty to explicitly encourage the agent to repeatedly visit all controllable states in the environment over an episode. Life-long novelty multiplicatively modulates the episodic similarity signal and is driven by a RND error. Episodic novelty uses an episodic memory filled with all previously visited states, encoded using the self-supervised objective of Pathak et al. (2017) to avoid uncontrollable parts of the state space. Episodic novelty is then defined as a similarity of the current state to previously stored states.

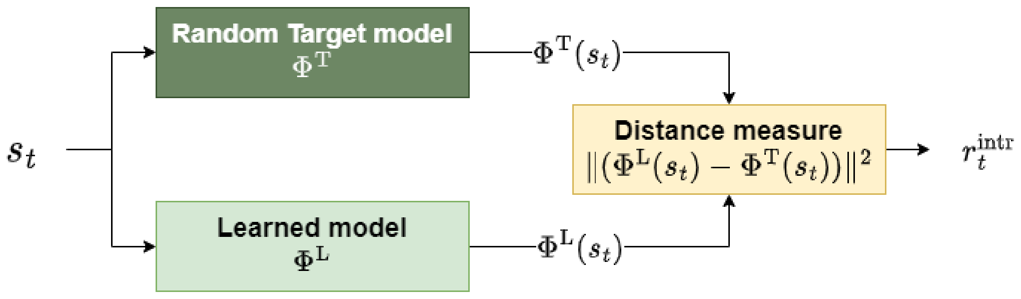


Figure 4.5: The basic principle of generating an exploration signal in random network distillation.

4.1.3 Information-based approach

Information-based approaches use concepts from information theory (Shannon, 1948), such as information gain, mutual information, or entropy, and try to maximize the information obtained by the agent from the environment. These methods assume that the agent has information quantity estimator (e.g. for estimating probability distribution of states across the state space $\mathcal{S}$) and yields some information metric

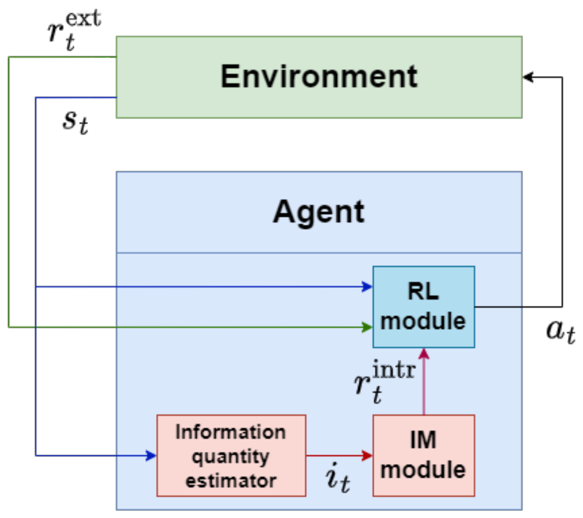


Figure 4.6: General architecture of agent with information-based approach to intrinsic motivation.

i_t. General architecture is presented in Figure 4.6. Information-based methods use information metrics either directly to calculate intrinsic motivation or as supporting loss functions.

Predictive information motivation is based on information quantity denoted predictive information (Bialek et al., 2001; Bialek and Tishby, 1999) and motivates the agent to search for novelty states which are unpredictable. It is defined for temporal sequence of random variable S as the mutual information between the past and the future

$$PI(S) = MI(S_p; S_f) \tag{4.24}$$

where $S_p = \{s_0, s_1, ..., s_t\}$ represents entire sequence of observed states s_t and $S_f = \{s_{t+1}, s_{t+2}, ...\}$ represents all future states that will be observed. It can be also defined as Kullback–Leibler divergence between the joint distribution $P_{S_p,S_f}(s, s')$ and the product of the independent distributions $P_{S_p}(s)$ and $P_{S_f}(s')$

$$PI(S) = D_{\text{KL}}(P_{S_p,S_f}(s, s') \| P_{S_p}(s) P_{S_f}(s')) \tag{4.25}$$

It is obvious that exact calculation is impossible because we have no knowledge about the entire future, so there is employed *one-step predictive information* PI^* defined as the difference between the entropy of s_{t+1} and the conditional entropy of s_{t+1} given s_t

$$\begin{aligned} PI^*(s) &= MI(s_{t+1}; s_t) && (4.26) \\ &= H(s_{t+1}) - H(s_{t+1}|s_t) && (4.27) \end{aligned}$$

The first term leads to maximization of exploration behaviour of the agent which is driven to observe every state with equal probability and the second term expresses uncertainty of the next state s_{t+1} when the agent possesses some knowledge about the current state s_t which pushes the agent to choose actions leading to predictable states. The intrinsic reward can be defined as

$$r_t^{\text{intr}} = C \cdot PI^*(s_t) \tag{4.28}$$

There were elaborated two forms of combining extrinsic and intrinsic reward based on predictive information, oth were tested on embodied agents in a continuous environment and yielded different results. In one form (Montúfar et al., 2016) the reward signal was

$$r_t^{\text{intr}} = \sum_{s'} p_t(s'|s_{t-1}) \log \frac{p_t(s'|s_{t-1})}{p_t(s')} \tag{4.29}$$

where p_t is the empirical distribution of states from the last T' time step and the whole reward for the agents is computed as a product of extrinsic and intrinsic signal

$$r_t = (r_t^{\text{extr}})^{\xi} \cdot (r_t^{\text{intr}})^{1-\xi} \tag{4.30}$$

This avoids optimization of only one of the rewards, but there is an assumption that the agent receives extrinsic reward after each action, otherwise the whole reward would be zero and intrinsic reward would not be taken into account.

There were proposed other modifications, such as

$$\begin{aligned} r_t &= f\left[(r_t^{\text{extr}})^{\xi} \cdot (r_t^{\text{intr}})^{1-\xi}\right] && (4.31) \\ &= \operatorname{sign}(r_t^{\text{extr}})\left[(r_t^{\text{extr}})^{\xi} \cdot (r_t^{\text{intr}})^{1-\xi}\right] && (4.32) \end{aligned}$$

where f is some monotonic (decreasing) function. In this experiment the agent was a simulated hexapod and its goal was to walk the maximal distance in a limited time. Employing such a signal showed that it improved learning speed and the agent reached more extrinsic reward with the intrinsic signal than without it. The best results were achieved for $\xi = 0.75$.

On the other hand, similar experiments were done by Zahedi et al. (2013) who considered episodic tasks in a continuous environment and defined intrinsic reward as a linear combination of extrinsic reward and one-step predictive information

$$r_t = r_t^{\text{extr}} + \beta\, r_t^{\text{intr}} = r_t^{\text{extr}} + \beta\, PI^*(s_t) \tag{4.33}$$

where β is the factor scaling with respect to the maximal external reward. They performed three different experiments. One was a cart-pole swing-up task, the second and the third involved hexapod locomotion and self-rescue task. After evaluation of the experiments they observed no significant effect of predictive information signal on learning compared to set-ups where the agent only received extrinsic reward.

Information gain is a measurement of the reduction in uncertainty about a random variable based on new information. It helps the agent make better decisions by choosing the action that leads to the greatest reduction in uncertainty about the environment. It is equal to Kullback–Leibler divergence (Kullback and Leibler, 1951) of probability distributions Q and P. It is defined as the difference between the cross-entropy and the entropy

$$D_{\mathrm{KL}}(P \| Q) = H(P, Q) - H(P) \tag{4.34}$$

and after substitution and rearrangements

$$D_{\mathrm{KL}}(P \| Q) = -\sum_i P(i) \log \left(\frac{Q(i)}{P(i)} \right) \tag{4.35}$$

It can be also defined as a difference between entropies of the same state in two time steps t_0 and t_1

$$IG(s) = H(s, t_0) - H(s, t_1) \tag{4.36}$$

Then the intrinsic reward is proportional to the information gain for state s

$$r_t^{\mathrm{intr}} = \eta \cdot IG(s_t) = \eta \cdot H(s_t, t-1) - H(s_t, t) \tag{4.37}$$

One of the drawbacks is that it can be computationally expensive to estimate information gain in continuous environments.

Pseudo-count (Bellemare et al., 2016), that was mentioned in Sec. 4.1.2, can be also connected to information gain which is commonly used in intrinsic motivation. To make such a connection one needs to introduce the mixture model ξ over a class of density models $\mathcal{M}$

$$\xi_n(s) = \xi(s; s_{1:n}) = \int_{\rho \in \mathcal{M}} w_n(\rho) \rho(s; s_{1:n}) \mathrm{d}\rho \tag{4.38}$$

where $w_n(\rho)$ is the posterior weight of ρ and is defined recursively, starting from a prior distribution w_0 over $\mathcal{M}$

$$w_{n+1}(\rho) = w_n(\rho, s_{n+1}) \tag{4.39}$$

$$w_n(\rho, s) = \frac{w_n(\rho)\rho(s; s_{1:n})}{\xi_n(s)} \tag{4.40}$$

In Bayesian inference Kullback–Leibler divergence $D_{\mathrm{KL}}(P \| Q)$ is defined as a measure of the information gained from correction of one's beliefs from the prior probability distribution Q (which can be interpreted as a theory or a model of P) to the posterior probability distribution P (interpreted as observation or true distribution of data) (Burnham and Anderson, 2003). Then information gain of state s can be defined as Kullback–Leibler divergence from prior to posterior

$$IG_n(s) = IG(s; s_{1:n}) = D_{\mathrm{KL}}(w_n(\cdot, s) \| w_n) \tag{4.41}$$

The authors introduced a quantity called *prediction gain* which provides good approximation to information gain of density models. It is defined as the difference between the logarithm of recoding probability and the logarithm of probability, both obtained from the density model ρ

$$\begin{aligned} PG_n(s) &\approx IG_n(s) && (4.42) \\ PG_n(s) &= \log \rho'_n(s) - \log \rho_n(s) && (4.43) \end{aligned}$$

This quantity can then be related to the pseudo-count

$$\hat{N}_n(s) \approx \frac{1}{\exp(PG_n(s)) - 1} \tag{4.44}$$

Variational Information Maximizing Exploration (VIME) (Houthooft et al., 2016) approximates the environment dynamics and uses the information gain of the learned dynamics model as intrinsic rewards. Bayesian neural network (BNN) (Graves, 2011) implements agent's dynamics model. Intrinsic motivation was then defined as

$$r_t^{\text{intr}} = D_{\text{KL}} \left[q(\theta; \phi'_{t+1}) \| q(\theta; \phi_{t+1}) \right] \tag{4.45}$$

where $q(\theta, \phi)$ is the dynamics model (BNN) weight distribution, ϕ is prior distribution of the dynamics parameters and ϕ' is posterior distribution of the dynamics parameters. According to this motivational signal, the agent should take actions that maximize the reduction in uncertainty about the environment dynamics.

State Entropy explicitly encourage the agent to uniformly visit all states by maximizing the entropy of the state distribution. The theoretical basis for methods of internal motivation based on the maximization of state entropy is provided by Hazan et al. (2019) and to compute state entropy they used state density estimation. Random Encoders for Efficient Exploration (RE3) (Seo et al., 2021) is an exploration method that utilizes state entropy as an intrinsic reward. The key idea of RE3 is k-nearest neighbor entropy estimation in the low-dimensional representation space of a randomly initialized encoder. Intrinsic motivation was defined as

$$r_t^{\text{intr}} = \log(\|y_i - y_i^{kNN}\|_2 + 1) \tag{4.46}$$

where $y_i = f_\theta(s_i)$ is a fixed representation from a random encoder f and y_i^{kNN} is the k-th nearest neighbor of y_i within a set of N representations $y_1, y_2, ..., y_N$. Seo et al. (2021) claim that measuring the distance between the states in the fixed representation space produces a more stable intrinsic reward, since the distance between a given pair of states does not change during training.

Active Pre-Training (APT) method proposed by Liu and Abbeel (2021) uses unsupervised pre-training method for generating suitable representations and non-parametric particle-based entropy estimator (Singh et al., 2003) for generating intrinsic reward. The key idea is to explore the environment by maximizing a non-parametric entropy computed in an abstract representation space (created by contrastive learning), which avoids challenging density modeling. The reward function for each transition is given by

$$r_t^{\text{intr}} = \log \left(\frac{1}{k} \sum_{z^{(j)} \in N_k(f_\theta(s))} \| f_\theta(s) - z^{(j)} + 1 \|_2 \right) \tag{4.47}$$

where f_θ is a trained encoder, $N_k(\cdot)$ denotes k nearest neighbors around a particle z.

Uncertainty motivation rewards visitation of states that have low probability of observation (Oudeyer and Kaplan, 2009). The intrinsic reward after observation of state s_t is defined as inversely proportional to the probability $p(s_t)$

$$r_t^{\text{intr}} = C \cdot (1 - p(s_t)) \tag{4.48}$$

where C is the scaling parameter. This signal would lead to behaviour denoting optimism in the face of uncertainty and we can model a phenomenon of novelty for the agent.

4.1.4 Learning progress approach

Learning progress methods are based on an idea to reward the decrease of prediction errors. The basic approach is to compare the mean of prediction errors between two times, t and $t - \theta$. This may lead to confusions in situations where the transition which leads from highly unpredictable states to highly predictable is rewarded but they are absolutely qualitatively different and thus incomparable. Therefore, the agent has to compare qualitatively equal situations which are parts of regions $R_n(s_t)$ of the state space split by some mechanism. As an example we provide a mechanism based on threshold T_f such that the region

$$R_n(s_t^i) = \left\{ s_t^j | \text{dist}(s_t^j, s_t^i) < T_f) \right\} \tag{4.49}$$

The intrinsic reward is then defined as

$$r_t^{\text{intr}} = \left\langle e_{t-\theta}^{\mathcal{R}_n} \right\rangle - \left\langle e_t^{\mathcal{R}_n} \right\rangle \tag{4.50}$$

where $\left\langle e_t^{\mathcal{R}_n} \right\rangle$ is the mean prediction error of the state that belongs to the region R_n.

4.2 Competence-based category

Competence-based approach computes intrinsic reward using a measure of competence of the agent to achieve a self-determined goal. In other words it should drive the agent to acquisition of multiple skills. This approach is appropriate for hierarchical reinforcement learning (particularly option framework (Sutton et al., 1999)) paradigm where one option can be considered as one skill and competence-based motivation can boost their learning. The competence of the agent can be defined as an ability to reach the desired goal in reasonable time.

The basic idea is that the agent itself generates a goal (it may be some desired state) denoted g_k. The agent's component, which chooses actions (the planning system) and is responsible for reaching of the goal g_k, generates a plan that is later performed. The execution of actions can be stopped when the goal g_k is reached or the number of steps exceeds the limit T_g. The process of learning is divided into episodes where each episode is indexed by time step t_g. An episode begins after generating a new goal g_k and ends when the agent reaches the goal $\tilde{g}_k(t_g)$. At the end of the planned sequence the reached goal $\tilde{g}_k(t_g)$ is compared to the desired goal $g_k(t_g)$ and the level of achievement (performance) is computed as

$$l_a(g_k, t_g) = \|\tilde{g}_k(t_g) - g_k(t_g)\| \tag{4.51}$$

The following methods of competence-based approach are based on the level of achievement $l_a(g_k, t_g)$. General architecture is shown in Fig. 4.7.

Maximization of incompetence motivation drives the agent to reaching goals where it has low level of achievement so it is highly motivated for the most challenging tasks (Oudeyer and Kaplan, 2009). The following equation defines intrinsic reward obtained at the end of episode

$$r_t^{\text{intr}} = C \cdot l_a(g_k, t_g) \tag{4.52}$$

In the case that there is high variance in the reaching of the desired goal, it is more suitable to use mean value of level of performance the same goal g_k. Then the equation would be

$$r_t^{\text{intr}} = C \cdot \langle l_a(g_k, t_g) \rangle \tag{4.53}$$

In the next step one could introduce segmentation of goals into regions $\mathcal{R}_n$ to enhance generalization such that

$$\mathcal{R}_n(g_k) = \{g_l | dist(g_k, g_l) < \sigma_g\} \tag{4.54}$$

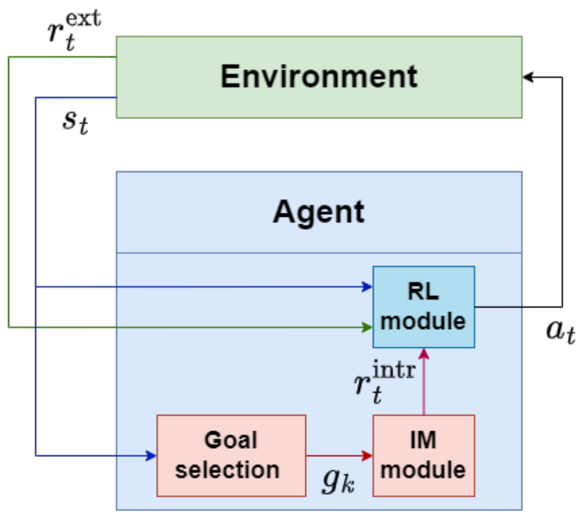


Figure 4.7: General architecture of competence-based approach to intrinsic motivation.

where σ_g is some threshold and *dist* is the distance function. There is defined goal $g_k^{\mathcal{R}_n} \in \mathcal{R}_n$ and intrinsic reward as the mean of performance in reaching the goal $g_k^{\mathcal{R}_n}$

$$r_t^{\text{intr}} = C \cdot \langle l_a(g_k^{\mathcal{R}_n}, t_g) \rangle \tag{4.55}$$
$$= C \cdot \langle l_a(g_k^{\sigma_g}, t_g) \rangle \tag{4.56}$$

The main drawback of this approach could be pushing the agent to reach goals which are unreachable (e.g. a child trying to reach the ceiling) or have a small impact on the whole task.

Maximizing competence progress addresses the problem mentioned above and instead of measuring performance, it measures progress in performance and so the goals where there is no progress are no more interesting for the agent and thus it can focus on other goals, which are reachable for it (Oudeyer and Kaplan, 2009). This method is influenced by the theory of flow and seems to be a good way for its implementation in agents. The intrinsic reward is defined as a difference between two levels of performance at time t and time $t - \theta$ (the previous attempt) for the same goal g_k

$$r_t^{\text{intr}} = C \cdot [l_a(g_k, t_g - \theta) - l_a(g_k, t_g)] \tag{4.57}$$

We can apply an extension from previous method dealing with high variance in performance where instead of subtracting performance in one episode, we subtract the mean of performance

$$r_t^{\text{intr}} = C \cdot [\langle l_a(g_k, t_g - \theta) \rangle - \langle l_a(g_k, t_g) \rangle] \tag{4.58}$$

The goals can also be segmented into regions to improve generalization, then the intrinsic reward would be

$$r_t^{\text{intr}} = C \cdot \left[\langle l_a(g_k^{\sigma_g}, t_g - \theta)\rangle - \langle l_a(g_k^{\sigma_g}, t_g)\rangle\right] \tag{4.59}$$

These two methods were further elaborated and explored in Santucci et al. (2013) where a simple set of 8 tasks was proposed. Two of these tasks were easy to reach (the agent needed less than 2000 trials to do that), two were difficult (more than 20000 trials) to reach and four were unreachable. The agent (shown in Figure 4.8) was composed of a predictor, a selector and a controller. The controller contained $n = 8$ actor-critic experts, each dedicated to one task. The selector was implementing Q-value function where the action was selecting one expert with probability P_k computed by softmax rule

$$P_t^k = \frac{\exp(Q_t^k/\tau)}{\sum_{i=0}^n \exp(Q_t^i/\tau)} \tag{4.60}$$

The selector updates its Q-values using a simple rule

$$Q_{t+1}^k = Q_t^k + \alpha\left[r_t^{\text{intr}} - Q_t^k\right] \tag{4.61}$$

where r_t^{intr} is the intrinsic reward calculated from the predictor. The predictor outputs the values in the range $[0, 1]$, where 1 means absolute certainty that the chosen expert will reach the goal and 0 means the opposite.

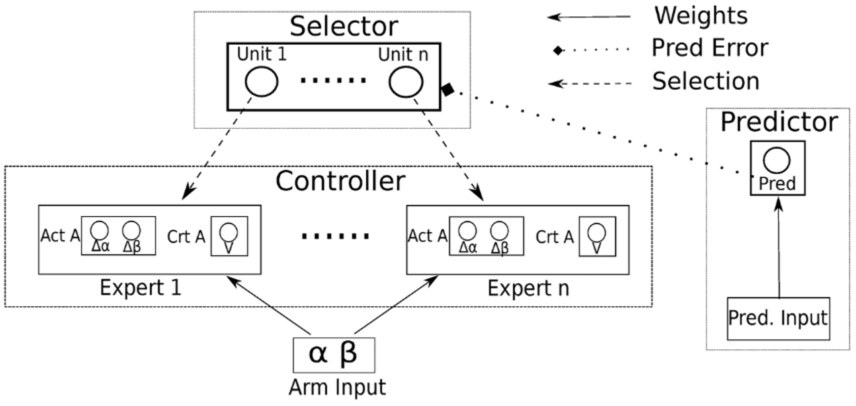


Figure 4.8: Architecture of the hierarchical system of actor–critic experts. Adapted from Santucci et al. (2013).

The authors identified several forms of predictors generating errors belonging to maximizing incompetence or maximizing competence progress methods:

- **State–action predictor** – in this case the predictor has state s_t and action a_t on the input and if predicts if the next state will reach the goal and complete the task. The predictor makes such predictions step-by-step and is trained through the delta rule.

- **State predictor** – has only the state s_t on the input and is more closely coupled to the competence of the system than the previous method. The predictor is also trained through a delta rule and runs in a similar fashion as the first method.

- **Temporal difference state–action predictor** – is a slightly modified version of the state-action predictor which does not work well in continuous environments as shown in experiments by Santucci et al. (2014). This predictor is trained using the TD rule.

- **Temporal difference state predictor** - is a modified version of the state predictor, trained using the TD rule.

- **Task predictor** – was proposed by Santucci et al. (2012) and shows the best coupling to the competence of the agent in the related skill. At the beginning of the trial, there is only a binary vector on the input, encoding which expert was selected (that means which task is going to be solved) and the predictor yields if the expert will be successful or not. Without any further information its anticipating of the achievement of the target state is solely dependent on the agent's competence in that skill. The predictor is trained through the delta rule.

- **Temporal difference predictor** – uses TD error from the critic of the selected expert which can be considered as the measure of achieving the competence.

All listed forms of predictors can generate an error that can be transformed to a variant where we calculate the difference between errors from two time steps (or a difference between the means of errors) exactly like was described in maximizing competence progress paragraph.

The results showed that in case the authors used maximizing incompetence method, the most successful predictor was *task predictor*. Somewhat less optimal performance to be was achieved by TD variants of *state predictor* and *state-action predictor* and their "delta" variants proved inappropriate. When they used maximizing competence progress method for computation of reward, the *task predictor* showed once again the most suitable, but the other forms of predictor reached qualitatively comparable results.

Maximizing competence can be defined as motivation pushing the agent to improve well-mastered skills (Oudeyer and Kaplan, 2009). There can be applied all variations of calculating the reward mentioned in previous methods. When we apply the approach with a goal space segmentation into regions the form of intrinsic reward would be

$$r_t^{\text{intr}} = \frac{C}{\langle l_a(g_k^{\sigma_g}, t_g) \rangle} \tag{4.62}$$

4.3 Morphological-based category

This approach is based on morphological properties of states being observed by the agent. There is no special module that would provide predictions of the computing distribution. General architecture is shown in Figure 4.9.

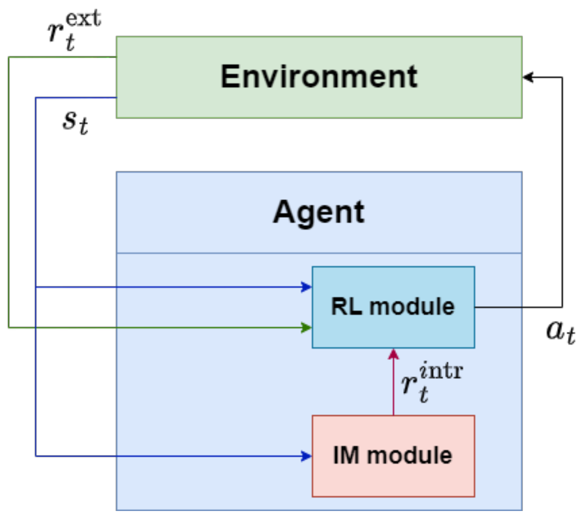


Figure 4.9: General architecture of a morphological computational approach to intrinsic motivation.

Synchronicity motivation is based on a short-term correlation between a number of sensory channels – components of the state vector (Oudeyer and Kaplan, 2009). States where is high short-term correlation between maximum number of sensory channels are considered interesting. There can be defined a set of sensory information sources S with elements S_i and its value at time t $S_i(t) = s_i$. There are more measures which can define synchronicity $sync(S_j, S_i)$. The first one is the normalized information distance (Crutchfield, 1990)

$$d(S_j|S_i) = \frac{H(S_i|S_j) + H(S_j|S_i)}{H(S_i|S_j)} \tag{4.63}$$

where $H(S_i|S_j)$ is the conditioned entropy

$$H(S_i|S_j) = -\sum_{s_i}\sum_{s_j} p(s_i, s_j)\log_2 p(s_i, s_j) \tag{4.64}$$

and synchronicity is defined as

$$sync(S_j, S_i) = \frac{C}{d(S_j|S_i)} \tag{4.65}$$

It can be also defined as mutual information

$$sync(S_j, S_i) = MI(S_i, S_j) \tag{4.66}$$

or the correlation between two time series

$$sync(S_i, S_j) = \frac{\sum_t (s_i(t) - \langle s_i \rangle) \cdot (s_j(t) - \langle s_j \rangle)}{\sqrt{\sum_t (s_i(t) - \langle s_i \rangle)^2} \cdot \sqrt{\sum_t (s_j(t) - \langle s_j \rangle)^2}} \tag{4.67}$$

Independently on the way how synchronicity was obtained, the intrinsic reward is defined as

$$r_t^{\text{intr}} = C \cdot \sum_i \sum_j sync(S_i, S_j) \tag{4.68}$$

Stability motivation rewards actions leading to only small changes in observed states and prevents the agent from observing large variance in states (Oudeyer and Kaplan, 2009). The interesting states are close to the mean value $\langle s \rangle_T$ from the past T steps

$$r_t^{\text{intr}} = \frac{C}{\|s_t - \langle s \rangle_T\|_2} \tag{4.69}$$

Variance motivation is an opposite method to stability (Oudeyer and Kaplan, 2009). The reward is given for actions leading to highly variable observations of the states

$$r_t^{\text{intr}} = C \cdot \|s_t - \langle s \rangle_T\|_2 \tag{4.70}$$

This section concludes the theoretical part of the dissertation thesis, in which we addressed three essential areas on which our research is based. In the next section, the research part of the thesis begins with a description of the methodology and results. Their analysis and interpretation follows. The research part is then concluded with discussion and further directions of possible development are outlined.

Chapter 5

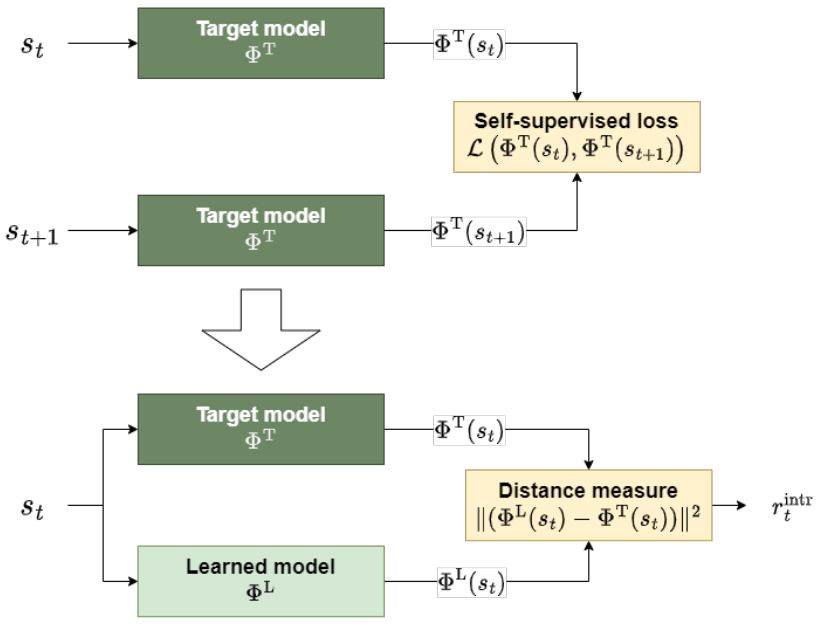
Self-supervised Predictors

We modified the concept of distillation of randomly initialized static network RND (Burda et al., 2018) and instead we distilled a network that learns continuously using self-supervised algorithms. We denote the methods **SND** (Pecháč et al., 2023). The architecture of such a model consists of a target model Φ^{T} and a learned model Φ^{L}, but with the essential difference that the network generating the target feature vectors (target model) is learned. The schematic representation of the proposed approach is shown in Fig. 5.1. In order to be able to use the trained target model as a suitable source of target feature vectors for the learning network, it is necessary that it fulfills the following conditions:

1. Two identical states must map to the same feature vector.
2. Two similar (e.g. successive states) are mapped on two similar feature vectors, such that their L_2 distance is small.
3. Two different states are mapped on two different feature vectors, such that their L_2 distance is large.

The feature space formed in this way can be distilled and then this process can be used as a source of internal motivation, because the new states will have different feature vectors than the states seen by the agent so far. We introduced three methods for forming a feature space that satisfies the above conditions. All three methods are based on SSL. In the context of SND methods, the RND is their special case, where the learning rate of the target model is 0.

We also did research in the area of methods based on the prediction error of the forward model Ψ whose task is to predict the next state s_{t+1} if it receives the current state s_t and the action a_t. The difference (e.g. mean-squared error) between the new observed state s_{t+1} and the predicted state $\hat{s}_{t+1}$ can be used as an intrinsic reward. The assumption is that the forward model will yield lower errors in states it has visited



s_t
Target model Φ^{T}
$\Phi^{\mathrm{T}}(s_t)$
Self-supervised loss $\mathcal{L}\left(\Phi^{\mathrm{T}}(s_t), \Phi^{\mathrm{T}}(s_{t+1})\right)$
s_{t+1}
Target model Φ^{T}
$\Phi^{\mathrm{T}}(s_{t+1})$
Target model Φ^{T}
$\Phi^{\mathrm{T}}(s_t)$
s_t
Distance measure $\|(\Phi^{\mathrm{L}}(s_t) - \Phi^{\mathrm{T}}(s_t))\|^2$
r_t^{intr}
Learned model Φ^{L}
$\Phi^{\mathrm{L}}(s_t)$

Figure 5.1: The basic principle of generating an exploration signal in the regularized target model and training of the SND target model using two consecutive states and the self-supervised learning algorithm.

many times and, conversely, higher errors for states that were visited rarely or never. The task can be simplified so that the forward model is trained to make predictions in the latent space (generated by feature extractor Φ), which eliminates the need to use a decoder that maps a point from the latent space to the state space. Instead, the difference between the feature vector z_{t+1} for s_{t+1} and the predicted feature vector $\hat{z}_{t+1}$ is used as a source of motivation.

The latent space representations can be created in different ways. For example, ICM model (Pathak et al., 2017) used the loss function based on the forward and inverse model. Due to the successful use of SSL methods in the case of the SND model, we decided to apply a similar principle here. We named the method Self-supervised Predictor (**SP**). SP uses a feature extractor trained by SSL loss and a forward model connected to it serves as a source of internal reward.

5.1 Methods

We introduced four new methods using SSL algorithms for feature space training. Three are based on target network distillation (SND) and one is based on forward model (SP).

5.1.1 SND-STD

SND-STD method (Pecháč et al., 2023; Pecháč and Farkaš, 2022) uses the Spatio-Temporal DeepInfoMax (ST-DIM) algorithm (Anand et al., 2019) (the simple diagram can be found in Fig. 5.1) leveraging multi-class N-pair losses (Sohn, 2016b) (see Sec. 3.1.5):

$$\mathcal{L}_{\mathrm{GL}} = -\sum_{i=1}^{I}\sum_{j=1}^{J} \log \frac{\exp(g_{i,j})}{\sum_{s_t^* \in S_{\mathrm{next}}} \exp(g_{i,j})} \tag{5.1}$$

$$\mathcal{L}_{\mathrm{LL}} = -\sum_{i=1}^{I}\sum_{j=1}^{J} \log \frac{\exp(f_{i,j})}{\sum_{s_t^* \in S_{\mathrm{next}}} \exp(f_{i,j})} \tag{5.2}$$

where $f(.) = f(s_t, s_{t+1})$ and $g(.) = g(s_t, s_{t+1})$ are score functions for local-local objective $\mathcal{L}_{\mathrm{LL}}$ and global-local objective $\mathcal{L}_{\mathrm{GL}}$, respectively. Function $g_{i,j}$ is defined as dot product between transformed global features $\Phi^{\mathrm{T}}(s_t)$ and the local features $\Phi^{\mathrm{T}}_{(l,i,j)}(s_{t+1})$ of the intermediate layer l in Φ^{T}, where (i, j) is the spatial location. Analogically $f_{i,j}$ is dot product between transformed local features $\Phi^{\mathrm{T}}_{(l,i,j)}(s_t)$ and $\Phi^{\mathrm{T}}_{(l,i,j)}(s_{t+1})$. The details of this algorithm are provided in Anand et al. (2019). S_{next} corresponds to the set of next states, (s_t, s_{t+1}) represents a pair of consecutive states, (s_t, s_t^*) represents a pair of non-consecutive states and I, J are the width and the height from output shape of intermediate convolutional layer of the target model. The resulting loss function is then defined as

$$\mathcal{L} = \frac{1}{IJ}(\mathcal{L}_{\mathrm{GL}} + \mathcal{L}_{\mathrm{LL}}) \tag{5.3}$$

Following this objective function, the target model becomes a good feature extractor adapting to new states discovered by the agent. Our experiments revealed that if the ST-DIM algorithm works on an incomplete dataset that takes on new samples (the authors probably did not test it in such conditions), there is an instability and an exponential increase of activity in the feature space at certain moments. This is related to the use of cross-entropy loss function in its core (which does not limit the values of inputs, logits), where derivatives can reach large values and subsequently inflate the entire feature space. During the development of the model, it turned out that it is best to minimize the L_2-norm of logits represented by functions f and g that enter the cross-entropy:

$$\mathcal{L}_n = p_{\mathrm{GL}} + p_{\mathrm{LL}} = \sum_{i=1}^{I}\sum_{j=1}^{J}(\|f_{i,j}\| + \|g_{i,j}\|) \tag{5.4}$$

In addition, we tried to maximize the entropy of the distributions generating the respective logits and minimize the L_2-norm of global features. However, both described approaches failed to sufficiently stabilize the algorithm.

Finally, we added one more regularization term $\mathcal{L}_v$ that maximizes the standard deviation σ of the feature vector components and thus ensures that all dimensions of the feature space are used:

$$\mathcal{L}_v = -\sigma(\Phi^{\mathrm{T}}(s_t)) \tag{5.5}$$

The final objective function, with the scaling parameters $\beta_1 = \beta_2 = 0.0001$ (found experimentally), was defined as

$$\mathcal{L} = \frac{1}{IJ}(\mathcal{L}_{\mathrm{GL}} + \mathcal{L}_{\mathrm{LL}} + \beta_1 \mathcal{L}_n) + \beta_2 \mathcal{L}_v \tag{5.6}$$

5.1.2 SND-VIC

SND-VIC method (Pecháč et al., 2023) is based on VICReg algorithm which is desribed in more details in Sec. 3.2.3. The regularization function consists of three terms: the invariant term, which brings the feature vectors closer to each other, the variance term, which ensures that the feature vectors within one batch have different values, and the covariance term, which ensures the decorrelation of the feature vectors and prevents information collapse. The original method does not need any negative samples, it only takes the input, creates two augmented versions, and their feature vectors are updated using the mentioned terms of the regularization function. Our version uses the state s_t and its successor s_{t+1} (the same as ST-DIM, the simple diagram can be found in Fig. 5.1) instead of two augmentations of the same state. The overall loss $\mathcal{L}$ takes the form

$$\mathcal{L} = \lambda \mathcal{L}_{s(Z,Z')} + \mu \left[\mathcal{L}_{v(Z)} + \mathcal{L}_{v(Z')}\right] + \nu \left[\mathcal{L}_{c(Z)} + \mathcal{L}_{c(Z')}\right] \tag{5.7}$$

where the scaling parameters were set to $\lambda = 1$, $\mu = 1$ and $\nu = 1/25$.

5.1.3 SND-VINV

SND-VINV method is an extension of the previous SND-VIC method by an inverse model Φ^{I} whose goal is to predict actions based on two feature vectors for a pair of input states s_t and s_{t+1}. The assumption is that the inverse model learns a feature space that encodes information relevant for predicting the agent's actions only (this assumption was also expressed in Pathak et al. (2017)), which could contribute to improving the properties of the overall feature space and filter out unnecessary information that does not apply directly to the agent. The goal of the model is to remove noise from the feature space, which arises as a result of various (often periodic or random) changes in the state space, which do not affect the agent in any way, but SSL algorithms try to separate them and thus generate the illusion of novelty for the agent, who thus wastes time investigating insignificant phenomena in the environment.

The inverse dynamics model optimizes the cross-entropy loss function if the action space $\mathcal{A}$ is discrete:

$$\mathcal{L}_{id(Z,Z')} = -\frac{1}{d}\sum_{i=1}^{d}\sum_{c=1}^{|\mathcal{A}|} a_{i,c}\log\left(\Phi^{\mathrm{I}}(Z,Z')_{i,c}\right) \tag{5.8}$$

where $\Phi^{\mathrm{I}}(Z,Z')_{i,c}$ is the prediction of the inverse model converted to probability that action a_c led to a transition from state s_t to state s_{t+1}. The overall loss $\mathcal{L}$ then takes the form

$$\mathcal{L} = \lambda\mathcal{L}_{s(Z,Z')} + \mu\left[\mathcal{L}_{v(Z)} + \mathcal{L}_{v(Z')}\right] + \nu\left[\mathcal{L}_{c(Z)} + \mathcal{L}_{c(Z')}\right] + \iota\mathcal{L}_{id(Z,Z')} \tag{5.9}$$

where the scaling parameters are set to $\lambda = 1$, $\mu = 1$, $\nu = 1/25$ and $\iota = 1/2$.

5.1.4 SP

The SP model has a feature extractor Φ, which is trained using the ST-DIM algorithm analogously to the SND-STD method. The forward model Ψ, based on the input feature vector z_t of state s_t and action a_t, tries to predict the feature vector $\hat{z}_{t+1}$ of the next state s_{t+1}. The architecture diagram is presented in Fig. 5.2. Internal motivation is then calculated as

$$r_t^{\mathrm{intr}} = \|\hat{z}_{t+1} - z_{t+1}\|^2 \tag{5.10}$$

where $\hat{z}_{t+1} = \Psi(z_t, a_t)$, $z_t = \Phi(s_t)$, $z_{t+1} = \Phi(s_{t+1})$ and $\|.\|$ denotes the Euclidean norm. The loss function $\mathcal{L}_\Phi$ for feature extractor is defined according to eq. 5.3 and the loss function $\mathcal{L}_\Psi$ for the forward model is defined as

$$\mathcal{L}_\Psi = \frac{1}{N}\sum_{\hat{z}_{t+1}, z_{t+1}\in\mathcal{T}} \frac{1}{d}\|\hat{z}_{t+1} - z_{t+1})\|^2 \tag{5.11}$$

where $\mathcal{T}$ is the trajectory of length N and d is the dimensionality of feature vectors z. The overall loss $\mathcal{L}$ then takes the form

$$\mathcal{L} = \mathcal{L}_\Phi + \mathcal{L}_\Psi \tag{5.12}$$

while there are two options for training the feature extractor. Either asymmetrically using $\mathcal{L}_\Phi$ (representations of the forward model Ψ get closer to the extractor Φ) or symmetrically using both loss functions $\mathcal{L}_\Phi$ and $\mathcal{L}_\Psi$ (representations of both models try to get closer to each other).

5.2 Experiments

All together, we tested our methods on 10 environments (Atari and Procgen) that are considered difficult for exploration.

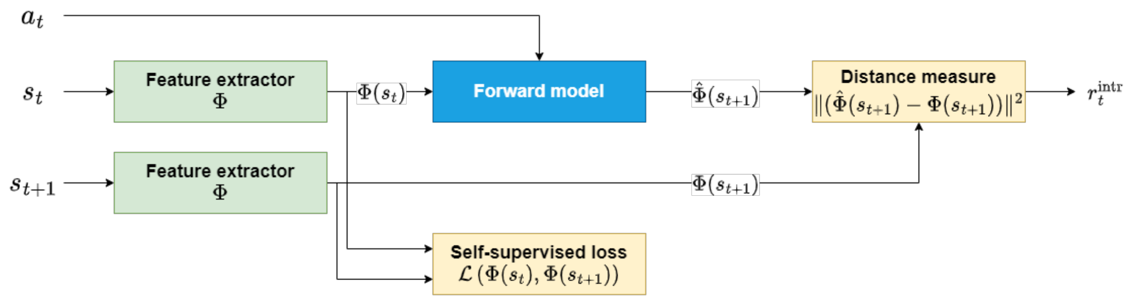


Figure 5.2: The scheme of the SP model with feature extractor trained by SSL algorithm and the forward model.

These include 4 Procgen environments: Climber, Caveflyer, Coinrun, and Jumper. Procgen is a set of procedurally generated environments, designed primarily for testing agent's generalisation (Cobbe et al., 2019). The paper shows several problems for generalisation in RL, requiring special training and a huge number of samples. For our purpose, interesting findings are provided in Appendix B.1 in Cobbe et al. (2019). For several seeds, the baseline agent was not able to reach a non-zero score. Those seeds lead to hard exploration environments, with only a single reward at the end. The state is represented by 2 consecutive RGB color images, with the overall size 6×64×64 pixels. The action space is discrete, consisting of 15 actions.

We also tested 6 Atari environments: Montezuma's Revenge, Gravitar, Venture, Private eye, Pitfall, and Solaris. The agent receives a reward of +1 for each increase in the score, regardless of its size. It does not receive any other reward or punishment. The state is represented by 4 consecutive frames of pixels on a grey scale, so the dimensionality of the state representation is 4×96×96. The action space is discrete, consisting of 18 actions, of which only some make sense (depending on the environment), the other actions have no impact on the environment.

5.2.1 Training setup

We ran 9 simulations for each environment, taking 128M steps for Atari and 64M steps for Procgen games. Before the main training, we tried 3 hand-selected settings of hyperparameters for individual motivational models (mainly the scaling of the motivational signal, or regularization terms) and we always chose the one that had the best results. These short probes lasted 32M (Atari) or 16M (Procgen) steps and consisted of 2 to 3 simulations.

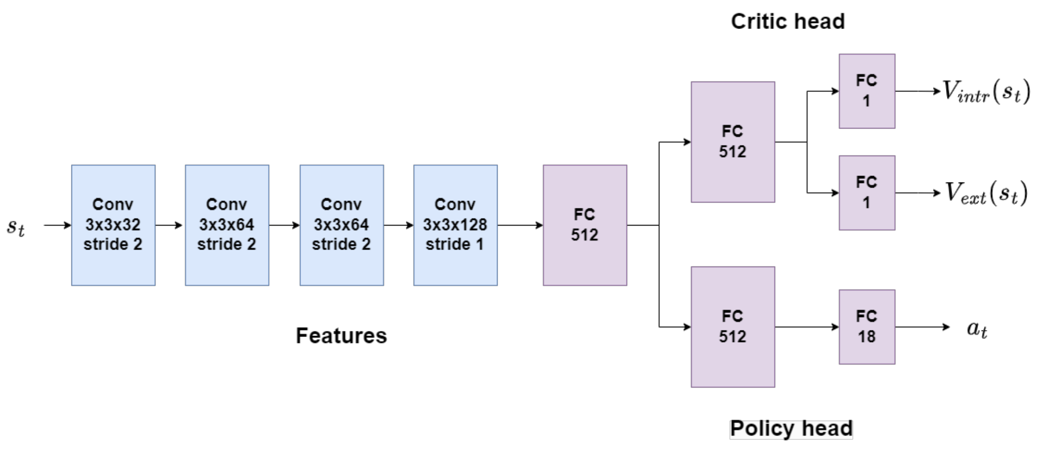
All agents were trained with the PPO algorithm (Schulman et al., 2017) using Adam optimizer (Kingma and Ba, 2015) to tune the parameters of all modules. The basic agent consists of an actor and a critic, which are two multi-layer perceptrons sharing a common convolutional neural network (CNN) that processes the video input. The critic has two outputs (heads), one for estimating the value function for the external reward and the other for the internal reward. We used the orthogonal weight initialisation with a magnitude $\sqrt{2}$. Model architectures are presented in Figures 5.3a to 5.3c. The motivational module of SND models consists of two CNNs (the target and the learned network), which receive input from a single frame. The learning network has two more linear layers to have an increased capacity over the target model.

The feature extractor of the SP model consisted of a CNN with four convolutional layers and one fully connected layer with an output dimension of 512, which was the feature space dimension. The forward model was formed by a classic feedforward network with three layers. The input dimension was 512 + the number of environment actions (18 for Atari / 15 for Procgen) and the output was again a 512-dimensional vector. The architecture of both models is shown in Fig. 5.3d and hyperparameters can be found in Tab. 5.4.

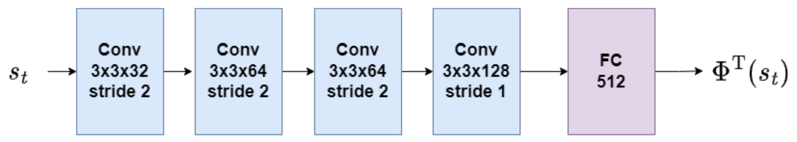
We followed Burda et al. (2018) for setting the hyperparameters, to have comparable results. We ran 128 parallel environments. For Atari we used 1M samples for each environment (total 128M frames), for Procgen we used 0.5M samples for each environment (total 64M frames). In Atari experiments we used gray scale downsampled 4 frames stacked observation. For Procgen we used 2 frames stacking, and a fully RGB colored observation. The intrinsic motivation modules used no frame stacking, a single gray scale image for Atari environments and a single RGB image for Procgen. The summary of all environment hyperparameters is in Table 5.1. The discount factors were set to $\gamma^{\text{ext}} = 0.998$ for external reward and $\gamma^{\text{intr}} = 0.99$ for intrinsic reward. We found the importance of intrinsic reward scaling, the best results were achieved for $\eta = 0.5$. The learning rate for all models was set to 0.0001 with Adam optimizer. Actor and Critic models used ReLU, and motivation models worked best with ELU activation function. We also find the deeper model with 3×3 convolutions works better then standard Atari model using 8×8 or 4×4 convolutions in Mnih et al. (2013). We retrained RND models to obtain comparable results and find faster convergence. The summary of PPO agent's hyperparameters is in Table 5.2. Hyperparameters of the intrinsic module are in Table 5.3. More hyperparameters and further details of the learning process and the architectures of modules can be found in our source codes.

5.2.2 State preprocessing

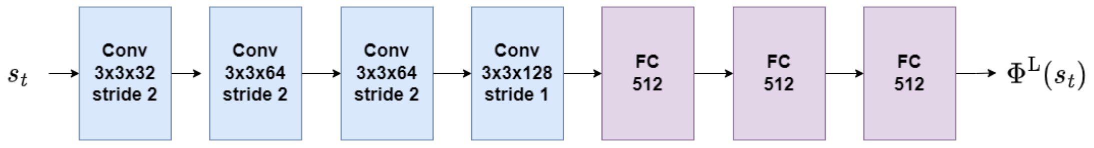
The state before entering the motivation module of the SND model can undergo preprocessing. We tested three preprocessing methods:



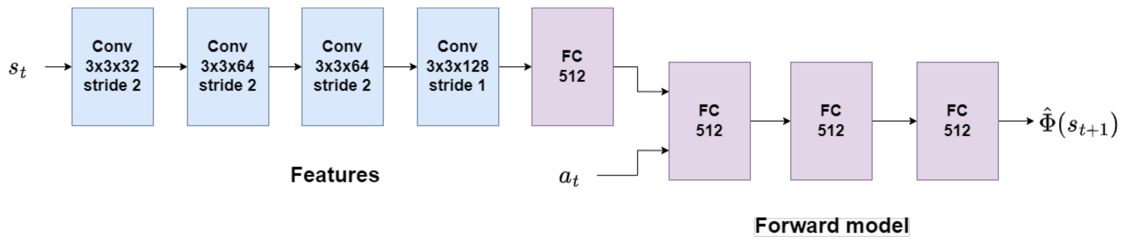
(a) The PPO agent model architecture.



(b) The SND target model architecture.



(c) The SND learned model architecture.



(d) The SP motivation module architecture.

Figure 5.3: Architectures of individual networks of agent with SND and SP motivation modules.

Table 5.1: Environment hyperparameters

Hyperparameter	Atari	Procgen
Observation downsampling	96×96	64×64
Frame stacking	4	2
State shape for PPO	4×96×96	6×64×64
State shape for IM modules	1×96×96	3×64×64
Parallel environments count	128	128
State normalisation	$s/255$	$s/255$
Samples per environment	1M	0.5M

1. State normalization using the running mean and the standard deviation
2. Subtraction of the running mean value from the state
3. No preprocessing.

We performed two training runs for each preprocessing method in 32M steps on Montezuma's Revenge environment. For testing we used the SND-STD model. Table 5.5 demonstrates that the state preprocessing did not have a significant effect on agent's performance (maximum reward achieved), only on the speed of learning. This also agrees with our assumption that operations such as subtraction of the mean or normalization should be able to find the network itself trained using the self-supervised loss function. Therefore it is not necessary for the designer to put them into the learning process explicitly. These conclusions will still need to be confirmed by statistical analysis. The RND used mean subtraction and SND-V, SND-STD together with SND-VIC did not use input state preprocessing.

5.3 Results

We processed several quick experiments on Montezuma's Revenge to explore an optimal setup. First we have to test the optimal architecture of Φ^{T} and Φ^{L} models. We experimented with 4 architectures:

1. identical models, one fully connected output layer, ELU activations
2. identical models, two fully connected layers, ELU activations
3. identical models, one fully connected layer, ReLU activations
4. asymmetric models, three fully connected layers for Φ^{L}, none for Φ^{T}, ELU activations fixed to fully connected convention

Table 5.2: PPO hyperparameters

Hyperparameter	Value
PPO model learning rate	0.0001
Discount factor γ^{ext}	0.998
Discount factor γ^{intr}	0.99
Advantages ext coefficient	2.0
Advantages intr coefficient	1.0
Intrinsic reward scaling	0.5
Rollout length	128
Number of optimization epochs	4
Entropy coefficient	0.001
Epsilon clipping	0.1
Gradient norm clipping	0.5
GAE λ coefficient	0.95
Optimizer	Adam
Weight initialisation	orthogonal

Table 5.3: SND hyperparameters

Hyperparameter	Value
Target model Φ^{T} learning rate	0.0001
Learned model Φ^{L} learning rate	0.0001
Weight initialisation	orthogonal

Table 5.4: SP hyperparameters

Hyperparameter	Value
Feature extractor Φ learning rate	0.0001
Forward model Ψ learning rate	0.0001
Weight initialisation	orthogonal

Results of different model architectures are in Figure 5.4. The best result was achieved for the asymmetric architecture.

Finally, we tested intrinsic reward scaling. Low values can lead to stacking the agent into a non-exploring policy. High values can prevent the agent from collecting extrinsic rewards, or make it too sensitive to small unimportant changes, both causing instability. We tested three values if intrinsic reward scaling: 0.25, 0.5 and 1.0. Figures 5.5 and 5.6 capture the cumulative external reward per episode and the standard deviation of

Table 5.5: Average cumulative reward (with standard deviation) per episode for all 3 preprocessing methods and maximal reward achieved by the agents.

Method	Average reward	Max. reward
normalization	3.60 ± 0.14	7
mean subtraction	4.13 ± 0.12	7
none	2.31 ± 0.20	7

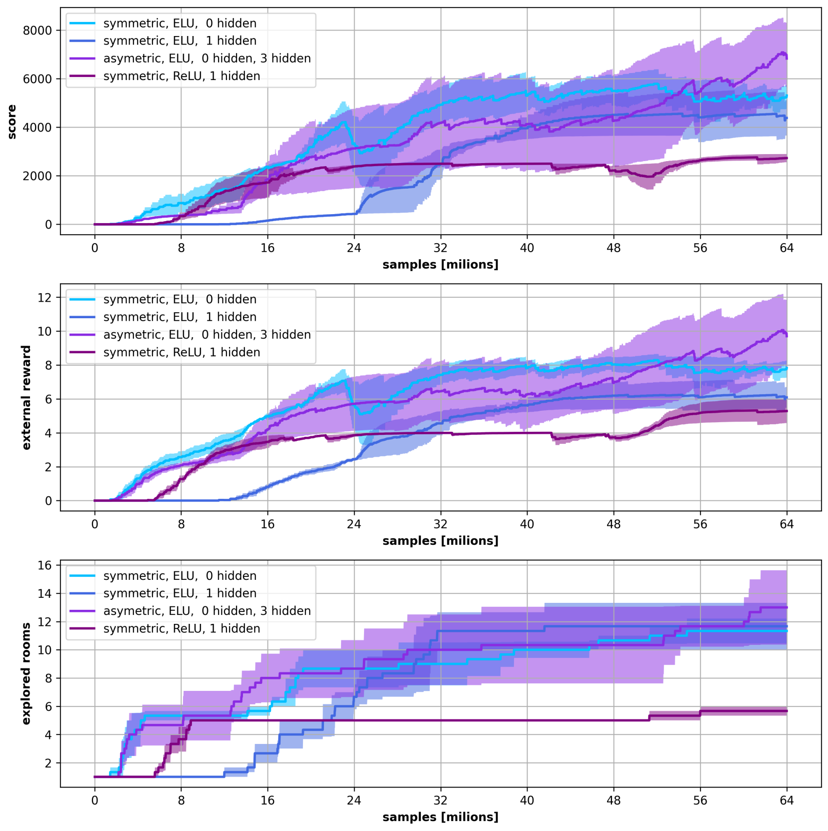


Figure 5.4: Agent's performance based on various learned model architectures, evaluated in terms of the overall score, external reward obtained and the number of rooms explored.

the tested models in 9 different environments for SND and SP methods, respectively. In Tables 5.6 and 5.8, these indicators are then averaged over the number of episodes. We compared methods based on distillation error separately from methods based on a prediction error, as our goal was to show that SSL algorithms can improve both types of intrinsic motivation. Table 5.7 shows the maximum achieved score for Atari environments, which is often used for model comparison (although we would like to emphasize that the agent never receives this score as a reward and therefore it is not its goal to maximize it). Of the tested environments, Pitfall game could not be mastered exceeded by any of the tested algorithms, since none of them achieved a single

Table 5.6: Average cumulative external reward per episode for tested SND models. The best model (not necessarily in statistical sense) for each environment is shown in bold face.

	Baseline	RND	SND-STD	SND-VIC	SND-VINV
Climber	0.00 ± 0.00	0.00 ± 0.00	1.48 ± 2.40	6.62 ± 3.24	**6.91 ± 2.75**
Caveflyer	0.00 ± 0.00	0.00 ± 0.00	10.86 ± 4.37	11.14 ± 2.35	**13.13 ± 1.27**
Coinrun	0.00 ± 0.00	0.25 ± 0.50	**9.40 ± 0.07**	5.70 ± 4.18	5.04 ± 4.51
Jumper	0.00 ± 0.00	0.03 ± 0.02	**9.76 ± 0.04**	9.76 ± 0.03	9.74 ± 0.04
Montezuma	0.00 ± 0.00	5.33 ± 0.23	7.76 ± 1.73	**8.45 ± 1.12**	N/A
Gravitar	1.19 ± 0.00	6.63 ± 1.55	5.89 ± 0.43	**10.05 ± 0.66**	N/A
Venture	0.00 ± 0.00	11.18 ± 0.42	9.54 ± 0.90	**11.36 ± 0.37**	N/A
Private Eye	0.81 ± 0.01	2.41 ± 0.95	3.79 ± 1.24	**5.93 ± 0.47**	N/A
Pitfall	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	N/A
Solaris	8.08 ± 0.15	3.84 ± 0.25	**10.83 ± 1.67**	10.85 ± 1.20	N/A

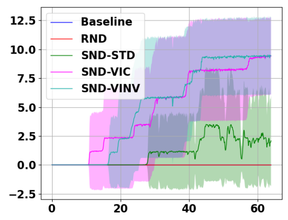
Table 5.7: Average maximal score reached by tested models on Atari environments. The best model for each environment is shown in bold face.

	Baseline	RND	SND-STD	SND-VIC	SP
Montezuma	400	6689	7212	**7838**	4267
Gravitar	2611	5600	4643	**6712**	5594
Venture	22	2167	2138	**2188**	1422
Private Eye	14870	14996	15089	**17313**	13779
Pitfall	0	0	0	0	0
Solaris	12344	10667	**12348**	11865	11282

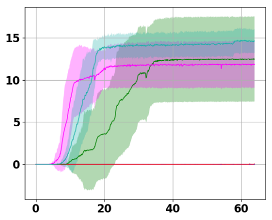
reward point; thus we omitted the chart for the Pitfall environment. In the remaining 9 environments, the best results were achieved with the models based on SND motivation, while in 8 cases it was with a significant lead over the existing algorithms (in Venture environment the results were almost the same as those of the RND model). When evaluating the score, the SND models achieved the highest score in 5 Atari environments (with an exception of the already mentioned Pitfall) and in 3 cases (Montezuma's Revenge, Gravitar, Private Eye) it was significantly higher than the compared models. The SP method outperformed ICM in 6 out of 8 environments, and it was always by a significant margin (ICM was better only in Caveflyer and Private Eye environments). In 4 environments, the ICM model failed to find any reward, while in the case of SP, it was only in 1 environment (the mentioned Pitfall, where all algorithms failed).

Table 5.8: Average cumulative external reward per episode for tested ICM and SP models. The best model for each environment is shown in bold face.

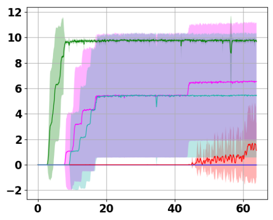
	Baseline	ICM	SP
Climber	0.00 ± 0.00	0.00 ± 0.00	**8.31 ± 1.85**
Caveflyer	0.00 ± 0.00	**6.20 ± 5.38**	0.75 ± 1.42
Coinrun	0.00 ± 0.00	0.00 ± 0.00	**8.68 ± 0.92**
Jumper	0.00 ± 0.00	7.84 ± 2.19	**9.75 ± 0.07**
Montezuma	0.00 ± 0.00	1.15 ± 0.48	**3.97 ± 0.96**
Gravitar	1.19 ± 0.00	5.69 ± 1.57	**11.60 ± 1.25**
Venture	0.00 ± 0.00	0.00 ± 0.00	**4.87 ± 4.76**
Private Eye	0.81 ± 0.01	**6.66 ± 0.20**	6.22 ± 0.57
Pitfall	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Solaris	8.08 ± 0.15	8.13 ± 0.60	**10.09 ± 0.48**



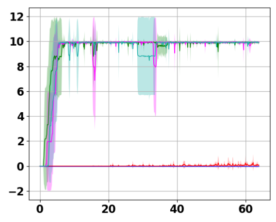
(a) Climber



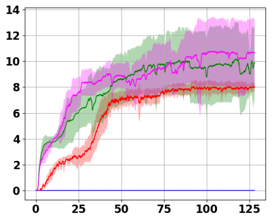
(b) Caveflyer



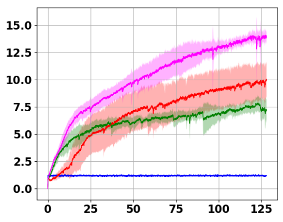
(c) Coinrun



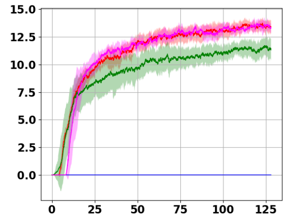
(d) Jumper



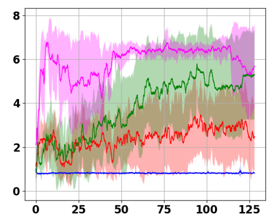
(e) Montezuma's Revenge



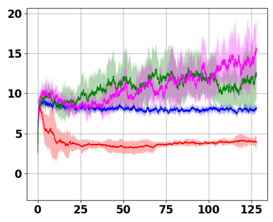
(f) Gravitar



(g) Venture

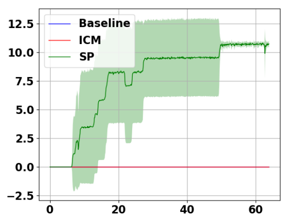


(h) Private Eye

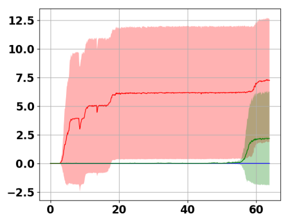


(i) Solaris

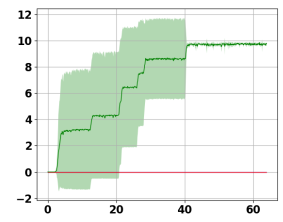
Figure 5.5: The cumulative external reward per episode (with the standard deviation) received by the agent from the tested environment. The horizontal axis shows the number of steps in millions, the vertical axis refers the external reward. We compared Baseline, RND and SND methods.



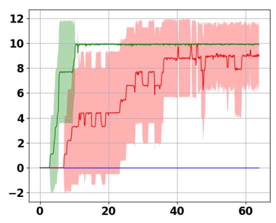
(a) Climber



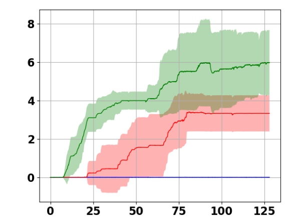
(b) Caveflyer



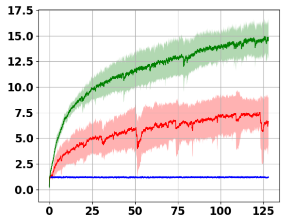
(c) Coinrun



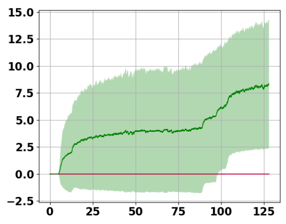
(d) Jumper



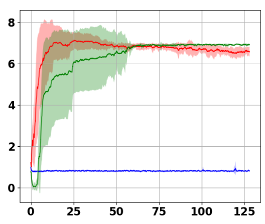
(e) Montezuma's Revenge



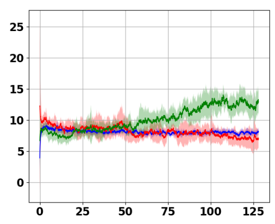
(f) Gravitar



(g) Venture



(h) Private Eye



(i) Solaris

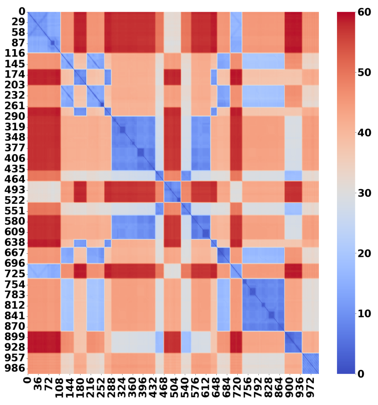
Figure 5.6: The cumulative external reward per episode (with the standard deviation) received by the agent from the tested environment. The horizontal axis shows the number of steps in millions, the vertical axis refers the external reward. We compared Baseline, ICM and SP method.

5.4 Analysis

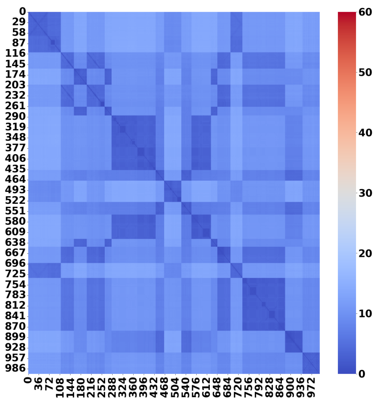
For a better idea of how the SSL methods work, we visualized the feature space by collecting 1000 states in the Montezuma's Revenge environment by a trained agent (used SND-STD method) and calculating the distance matrix for the collected states. Figure 5.7 illustrates that the SSL methods can preserve and transfer the structure of the state space to the feature space. The patterns on the diagonal correspond to individual rooms in the environment, where the states are similar to each other and therefore their mutual distance is small (blue color), while the distance to the states from other rooms is large (red color). We can notice that in the case of both RND and SND-STD, a similar structure of distances is preserved as in the case of states, but with smaller differences. When comparing RND and SND-STD with each other, it can be seen that the feature vectors created by the SND-STD method still have larger distances, which is indicated by the light blue color. In contrast, SND-VIC is extremely contrastive, and apart from the main diagonal, there is no distinct original structure. However, the light red color indicates that the distances for almost any states are larger than the two previous methods, which indicates better discriminative capabilities of SND-VIC. With the ICM and SP methods, we see a significantly different picture. ICM features are formed only by the loss functions of the forward and inverse models. We can see in the picture that this led to a considerable distance of the feature vectors at the beginning of the trajectory (significantly greater than with SND methods), but at the end of the trajectory the feature vectors are already much closer to each other. With the SP method, whose feature space is formed using the forward model and the ST-DIM algorithm, we see a similar picture as with the ICM, but with the difference that the representations of the end states are also more distant from each other.

We can further visualise learned feature vectors Z in 2D using t-SNE method (van der Maaten and Hinton, 2008). Figure 5.8 shows the resulted features of trained Φ^{T} on Montezuma's Revenge. The randomly initialised trained network (the same as in Burda et al. (2018) in Figure 5.8a) can well distinguish between different rooms, however within the room the variance is low, pointing to the lack of exploration abilities. On the other hand, self-supervised regularized target model in Figure 5.8 provides a much larger variance of features, which provides a more sensitive novelty detection signal.

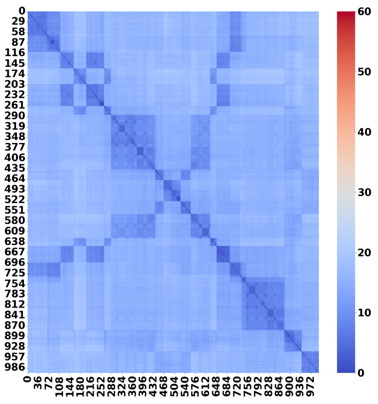
The main goal of the analysis was to find out the differences (not only visually) between the individual spaces of features, to describe them with some quantities and to find a possible connection with the performance of the algorithm and the mentioned quantities. From the set of examined models for one environment, we always selected the model with the highest obtained reward and generated 10,000 samples of input



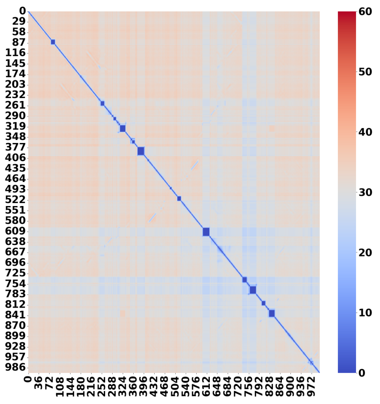
(a) States



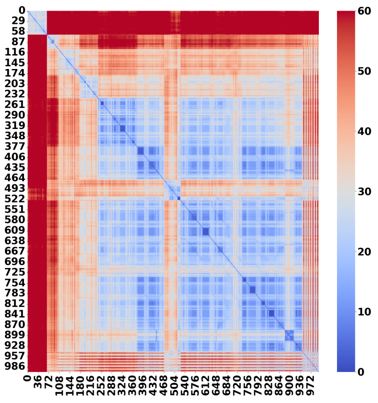
(b) RND feature vectors



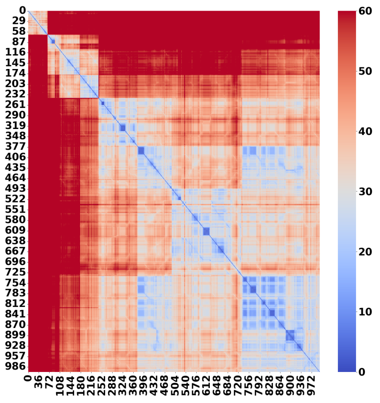
(c) SND-STD feature vectors



(d) SND-VIC feature vectors

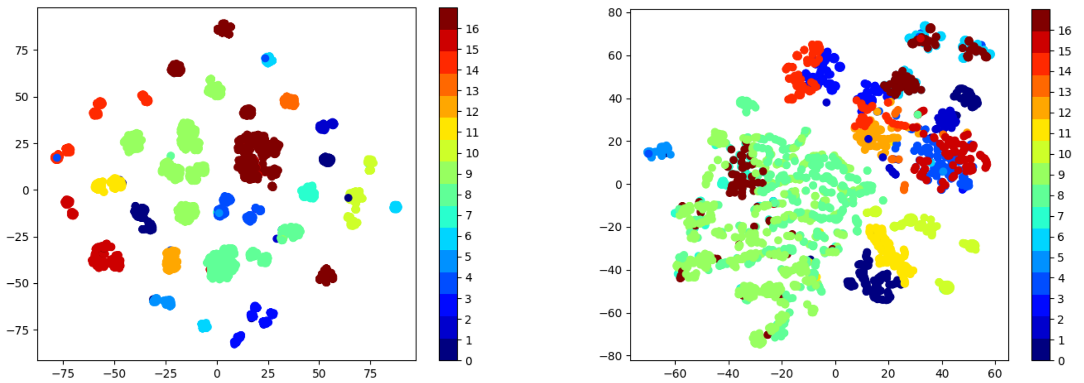


(e) ICM feature vectors



(f) SP feature vectors

Figure 5.7: The distance matrix of 1000 states and feature vectors collected from Montezuma's Revenge environment. Small distances are displayed by blue color, while large distances are displayed by red color.



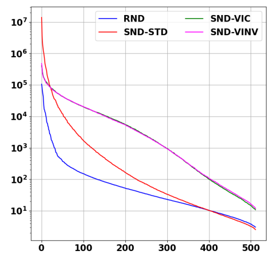
(a) Random target model (b) Trained target model

Figure 5.8: The t-SNE projected feature representations of the target models (random and trained) in Montezuma's Revenge task. The colors correspond to different rooms.

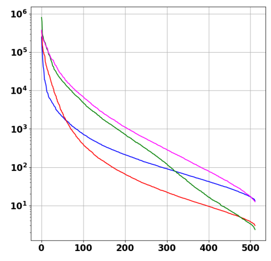
states by running it in the environment. Subsequently, each model generated feature vectors for a sample of previously collected input states. Thus, we obtained feature space samples Z of each model. Although feature spaces are the result of non-linear transformations, we decided to use the tools of linear algebra to get at least a rough idea of the properties of these spaces. First, we used the QR decomposition, which factorizes the matrix A into the unit matrix Q and the triangular matrix R. If A has n linearly independent columns, then the first n columns of Q form an orthonormal basis for the column space of A. From the rank of the matrix Q the actual dimensionality of the feature space can be determined. For all methods, the rank Q of the matrix was equal to the dimension of the vectors, which means that none of the dimensions of the vector space is just a linear combination of other dimensions (in other words, all dimensions are linearly independent). Then, using principal component analysis (PCA), we found the linear envelope of the high-dimensional manifold that forms the feature space. We examined the mean value and especially the variance of the feature vectors and also the eigenvalues obtained using PCA, which at least indicate the basic shape of the feature space (i.e. the sizes of the individual dimensions).

The results of this analysis are shown in Fig. 5.9, Fig. 5.10 and Tab. 5.9, Tab. 5.10. For the evaluation, we decided to use the following parameters: mean value and standard deviation of the L_2-norm of features, 25th, 50th, 75th and 95th percentiles of eigenvalues to obtain a rough representation of stretching of the feature space in individual dimensions. To this, we add the maximum achieved external reward ($\max r_{\text{ext}}$), so that it is possible to search for a connection between the parameters of the feature space and the performance of the method.

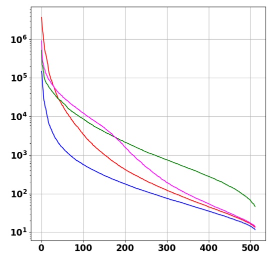
It can be seen that in almost all cases the RND target model has smaller eigenvalues



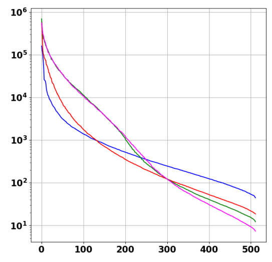
(a) Climber



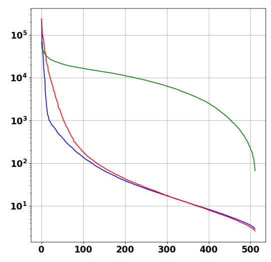
(b) Caveflyer



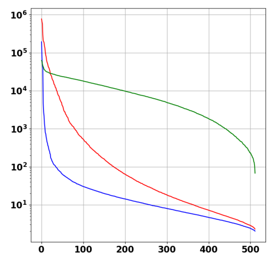
(c) Coinrun



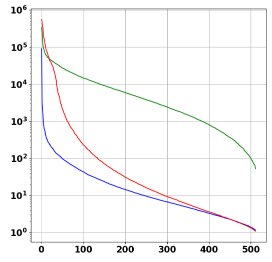
(d) Jumper



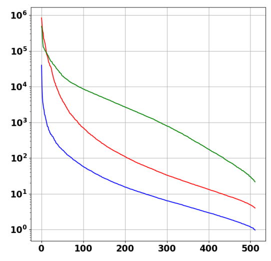
(e) Montezuma's Revenge



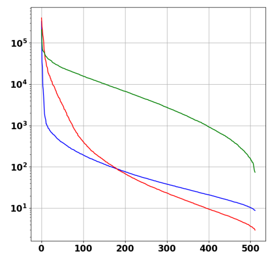
(f) Gravitar



(g) Venture



(h) Private Eye

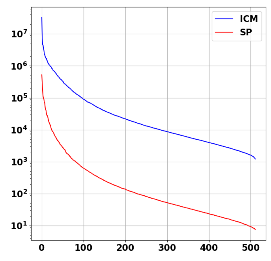


(i) Solaris

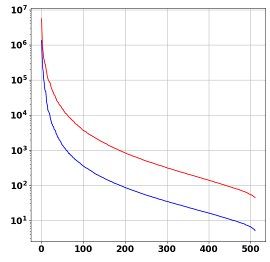
Figure 5.9: Descendingly ordered eigenvalues of the linear envelope obtained using the PCA method on RND and SND methods, which show the stretching of the feature space in individual dimensions. The horizontal axis shows the indices of eigenvalues, the vertical axis denotes the magnitude of eigenvalue on a logarithmic scale.

than the SND models. This can also be seen in the L_2-norm values that the entire RND feature space seems to have a smaller volume compared to the SND feature space. We can see that RND and SND-STD are similar in shape. Their curve has a convex shape, with SND-STD having more stretched dimensions. The shape of SND-VIC curve is ensured by the variance (eq. 3.12) and covariance (eq. 3.13) components of its loss function.

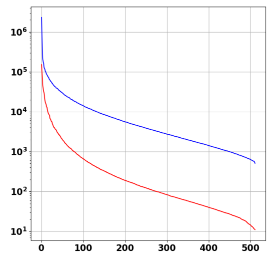
In contrast, the shape of the SND-STD curve is convex and the dimensions are used unevenly. After these analyses, we tried to improve the variance within the dimensions by adding a regularization term (eq. 5.5) which tried to maximize the variance within the feature vector. However, such a term had an expansive effect on the feature space



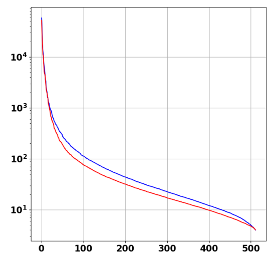
(a) Climber



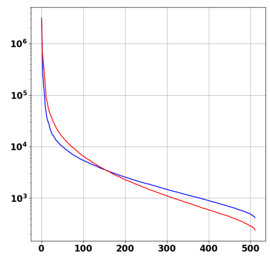
(b) Caveflyer



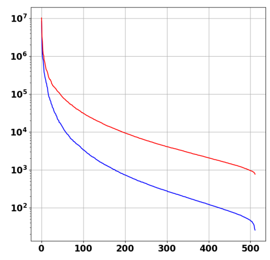
(c) Coinrun



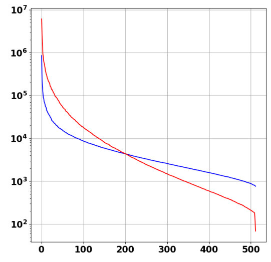
(d) Jumper



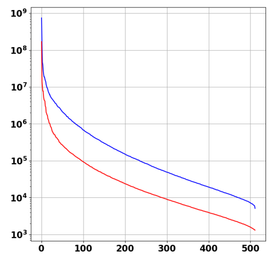
(e) Montezuma's Revenge



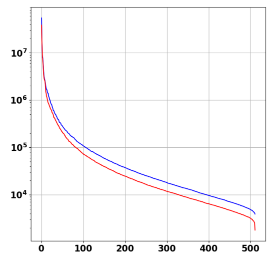
(f) Gravitar



(g) Venture



(h) Private Eye



(i) Solaris

Figure 5.10: Descendingly ordered eigenvalues of the linear envelope obtained using the PCA method on ICM and SP methods, which show the stretching of the feature space in individual dimensions. The horizontal axis shows the indices of eigenvalues, the vertical axis denotes the magnitude of eigenvalue on a logarithmic scale.

and it was not possible to give it much weight, because the loss function (eq. 5.3) of the ST-DIM algorithm itself has an expansive effect, and the addition of another expansive term led to problems with the uncontrolled expansion of the feature space. Despite the small influence of the variance component of the loss function, the performance of SND-STD improved and it helped prevent agents from getting stuck in certain cases. If we compare RND and SND-STD feature spaces (in terms of eigenvalues) they look similar, but the latter model was able to achieve better results in 7 out of 9 environments. Our findings show that when training the target model, it is important to enforce the decorrelation of features and the equal use of all dimensions of the feature space. Such a model seems to be relatively robust and sufficiently sensitive to novelty. Interestingly,

Table 5.9: Description of the target model feature space created by SND methods.

Environment	Method	$\max(r_{\text{ext}})$	L_2-norm	Q_{25}	Q_{50}	Q_{75}	Q_{95}
Climber	RND	0	7.22 ± 3.61	12	33	106	1549
	SND-STD	11	47.25 ± 22.64	12	64	823	54189
	SND-VIC	11	13.26 ± 3.99	145	2185	14114	75335
	SND-VINV	11	12.56 ± 3.73	152	2147	14069	70965
Caveflyer	RND	10	5.06 ± 2.98	48	128	474	4792
	SND-STD	16	9.69 ± 5.90	11	34	206	13452
	SND-VIC	16	9.89 ± 5.70	23	271	2638	49517
	SND-VINV	16	10.27 ± 6.25	96	507	3814	63714
Coinrun	RND	10	5.07 ± 3.00	40	109	389	4029
	SND-STD	10	29.23 ± 17.06	54	200	1651	76730
	SND-VIC	10	11.23 ± 5.77	332	1140	5414	44014
	SND-VINV	10	16.86 ± 8.98	66	445	7520	67400
Jumper	RND	10	8.63 ± 2.19	139	337	1008	6733
	SND-STD	10	15.51 ± 3.73	61	189	1044	22008
	SND-VIC	10	20.79 ± 6.18	47	257	6271	74860
	SND-VINV	10	15.55 ± 4.79	37	303	6037	77653
Montezuma	RND	9	1.93 ± 1.15	9	24	89	778
	SND-STD	16	4.85 ± 2.45	9	26	109	6835
	SND-VIC	17	6.22 ± 2.99	3066	8429	14858	25634
Gravitar	SND-STD	15	11.66 ± 2.82	8	30	250	20402
	RND	20	1.12 ± 0.81	5	10	24	139
	SND-VIC	21	7.26 ± 3.01	2263	6837	15173	27908
Private Eye	RND	9	1.40 ± 0.65	3	9	37	410
	SND-STD	9	8.03 ± 4.46	15	54	371	24025
	SND-VIC	10	4.29 ± 3.56	232	1401	6197	47342
Pitfall	RND	0	1.31 ± 0.39	3	10	47	410
	SND-STD	0	10.64 ± 2.81	8	32	319	39904
	SND-VIC	0	5.62 ± 1.79	153	1542	9827	44126
Venture	RND	18	0.68 ± 0.80	4	9	30	188
	SND-STD	18	4.50 ± 3.79	4	15	115	32272
	SND-VIC	18	4.91 ± 3.89	1006	3644	11256	41899
Solaris	RND	55	3.68 ± 3.42	29	62	172	776
	SND-STD	81	5.77 ± 4.28	12	37	211	11171
	SND-VIC	87	6.00 ± 4.14	1649	5079	13163	32377

Table 5.10: Description of the target model feature space created by SP methods.

Environment	Method	max(r_{ext})	L_2-norm	Q_{25}	Q_{50}	Q_{75}	Q_{95}
Climber	ICM	0	77.12 ± 28.05	4577	12590	55344	798106
	SP	11	7.22 ± 2.32	27	77	368	8948
Caveflyer	SP	16	15.39 ± 9.55	161	471	2263	51232
	ICM	19	6.76 ± 3.77	18	50	215	5479
Coinrun	ICM	0	23.71 ± 14.25	1588	3725	10409	51649
	SP	10	6.07 ± 3.00	46	118	431	5198
Jumper	ICM	10	3.78 ± 1.03	13	30	83	703
	SP	10	3.71 ± 0.98	11	22	57	549
Montezuma	ICM	5	16.67 ± 8.57	965	1873	4276	17329
	SP	9	16.41 ± 6.06	649	1491	4622	33561
Gravitar	ICM	16	23.36 ± 9.09	139	406	2024	47337
	SP	25	24.32 ± 10.69	2315	5761	20956	179683
Private Eye	SP	9	75.54 ± 34.26	4508	13467	58819	636182
	ICM	10	187.59 ± 95.90	22188	79620	418410	4709582
Pitfall	ICM	0	118.42 ± 41.71	44101	90429	235133	1013765
	SP	0	12.60 ± 4.09	709	1444	3626	19099
Venture	ICM	0	13.97 ± 10.16	1730	3209	6920	25560
	SP	18	13.04 ± 10.91	687	2324	11384	132229
Solaris	ICM	71	62.48 ± 30.42	10621	24314	75349	726208
	SP	88	52.08 ± 29.98	7121	15816	50221	541324

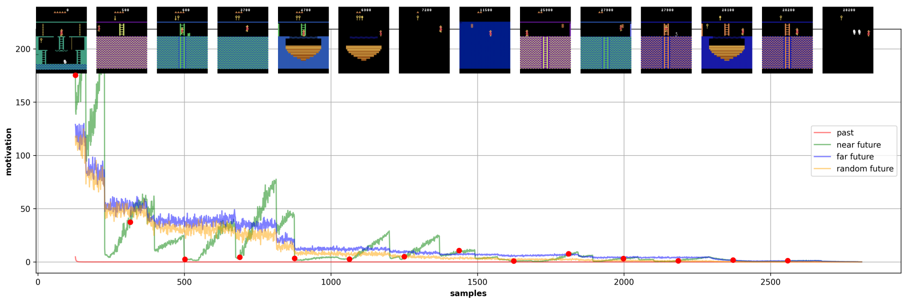
for Pitfall task (not shown in Figure 5.9), despite their failure, our methods still tried to take advantage of the feature space dimensions. From the analysis of the trained agents, we saw that they were able to explore several rooms, but in each there were enough moving objects that made the given state space rich and thus made it difficult to train the learned model, which led to a very slow decrease of the internal reward (we observed a similar behavior after short training sessions in other environments). We assume that with a larger number of training steps, the agent would eventually be able to reach the reward.

From the point of view of the eigenvalues, the SP and ICM methods form a very similar feature space, which mostly differed only in the size of the dimensions. Interestingly, for the Climber and Coinrun environments, the ICM feature space stretched by one to two orders of magnitude and failed to find any reward. In contrast, in the Caveflyer environment, the size of the spaces was swapped, and SP had a slightly more stretched space than ICM, and in this environment, ICM was able to find more external reward. From this, we conclude that for the Procgen environments we selected, models that created a smaller feature space always had an advantage, which probably led to a smaller prediction error and thus to a smaller reward, which then did not confuse the agent with a false signal. The Jumper environment turned out to be relatively simple, and both methods produced a feature space with similar properties. The situation seems to be slightly different for Atari environments. It seems that in the case of Montezuma's Revenge and Venture, the SP curves are similar and part of the dimension is more elongated compared to ICM and the other part is more contracted. In both cases, SP achieved a significantly higher external reward than ICM, suggesting that a larger feature space leading to the larger error and reward is more beneficial in these environments. This is even more pronounced in the case of Gravitar, where the SP, according to the curve, created a larger feature space by one to two magnitudes and this ensured a good exploration of the environment. With Private Eye, the order changed and in this case ICM achieved 1 point higher external reward than SP, which for now supports the mentioned trend: **the larger feature space = the better exploration**. The Solaris environment seems to be sensitive to high internal rewards, which can introduce more noise into the exploration (we have already observed this in the SND experiments), and therefore a smaller feature space, leading to a lower internal reward, proved to be more beneficial here.

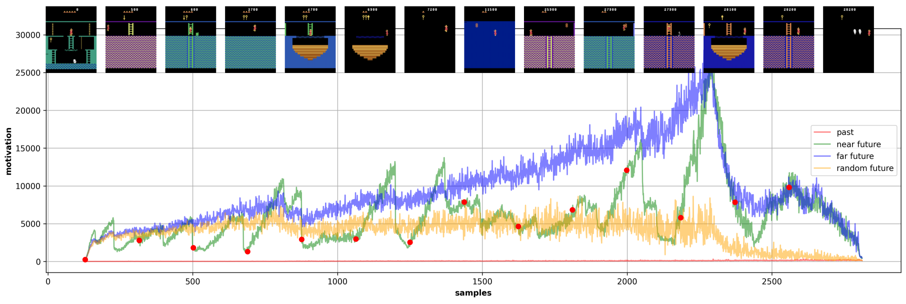
Another approach for different regularisation losses is understanding its time evaluation and the ability to provide large IM signal for previously unseen states. For the purpose of exploration, the most important is the ability to detect near future states, which are very close to already seen ones. We collected a set of 2700 states, from our best agent playing Montezuma's Revenge. During the experiment, we trained the intrinsic motivation modules only on past data, and tested on future data. The testing batch was selected from the following 4 time horizons, with respect to the agent being in step n, and testing batch indices m:

1. past: already seen states, $m < n$
2. near future: $n < m < n + 128$ steps in the future
3. far future: $m > n$
4. random: any batch from the set

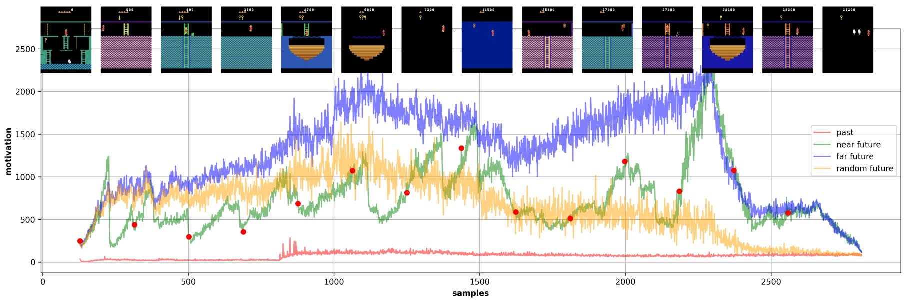
We hypothesised that the Random Network distillation provides a sufficient signal only at the beginning of learning. Converging to zero leads to limited exploration abilities. This degradation corresponds to our results in Figure 5.11a. On the other hand, continuously updated target models can provide useful signals for the entire run. The corresponding results are displayed in Figures 5.11b and 5.11c. On all three



(a) RND



(b) SND-STD



(c) SND-VIC

Figure 5.11: Novelty detection for different regularisation losses as response to different future windows. The states were collected on Montezuma's Revenge with the best agent, red dots correspond to state examples above.

losses, the intrinsic motivation is much higher for non-seen states, and not converging into zero. The strong peeks for near future correspond to new rooms finding. The self-supervised regularisation prevents collapsing the motivation signal to zero. This insight provides requirements for an exploration signal. We also have a simple methodology ready for testing the exploration abilities, without training the whole RL agent which can be time consuming.

Chapter 6

Discussion

We introduced a class of intrinsic motivation algorithms, based on distillation error as a novelty indicator (SND), where the target model is trained using self-supervised learning. We adapted three existing self-supervised methods for this purpose and experimentally tested them on a set of environments that are considered difficult to explore. The proposed variants have been shown to eliminate the identified shortcomings of the RND model – the need for good initialization, low variance of intrinsic reward on different states and the loss of the motivational signal caused by the adaptation of the learning network. We also created a simple forward model Self-supervised Predictor (SP) using SSL methods. This forward model makes predictions in the feature space, and this feature space is shaped by one of the SSL algorithms that we have previously tested for SND as well.

In the experiments, we tested the overall performance of the agents in 10 environments. With the exception of one environment, it was confirmed that the SND algorithms achieved better results than other methods with which we compared them. For the Atari environments, we also evaluated the achieved game scores so that they could be compared with other published models that we did not include in in this work. Also from the point of view of the SND score, the models dominated the compared models. The SP model was able to outperform the ICM model with which we compared it.

In the analytical part, we focused on deeper understanding of the SND and SP methods. We used a geometric approach trying to capture, at least in rough outlines, the properties of feature spaces. A comparison between a randomly initialized feature space and a feature space formed using one of the SND algorithms shows the correctness of our assumptions that self-supervised algorithms can distinguish even subtle differences within the state space. This turned out to be one of the weaknesses of the RND algorithm, which, while being good at distinguishing between sufficiently different states (e.g. different rooms in Montezuma's Revenge), it placed similar states close to each other in the feature space, making the work of the learned model easier. In the experiments, we thus observed a decrease in the standard deviation of the average intrinsic reward per episode, which meant that most of the visited states generated a similar reward. In the feature space, which was formed by the forward model and the inverse model (ICM) or with the SSL method (SP), we observed significant differences in distances. It appears that the states visited more times during training (located at the beginning of the trajectory) are much further apart in this space than the states visited less often (located towards the end of the trajectory). It is possible that this unevenness of the feature space also affects the lower success of both algorithms, especially in environments with very sparse rewards. It seems that the forward model provides only a kind of "local" pressure on the feature space, which does not have good generalization properties, but we have not analyzed this hypothesis in any way.

We experimented with different target model architectures and intrinsic reward scaling. We found that the target model using ELU activation and only one fully connected layer with the learned model with three hidden layers performs the best. The scaling of intrinsic reward shows big sensitivity to this parameter. The best working value was 0.5, however we think this value should be optimized separately for each specific environment.

We did not specifically investigate the robustness of SND methods with regard to the initialization of the target model (which was again a problem with RND). We assume that self-supervised learning algorithms can cope with a poorly initialized model to a certain extent, but from the training experience we found that it is better to initialize the target models of SND-STD, SND-VIC and SND-VINV to small values ($gain = 0.5$) and letting it expand itself while RND was initialized to higher values ($gain = \sqrt{2}$).

We also compared the effect of state preprocessing on the performance of the SND-STD model. It turned out that the state preprocessing is not necessary since it has no significant effect on the agent's performance.

We also performed an analysis of novelty detection abilities of selected methods. After comparison with RND as a baseline, we can conclude that this baseline suffers from an IM-based reward vanishing problem. After adding the regularisation to the target model, much better features were obtained, with significant change compared to the baseline. Intrinsic reward vanishing disappears for all the tested losses. This is cross-validated also on t-SNE features visualisation, where regularised features yield much higher variance, which means larger sensitivity to novelty.

A direct extension of SND methods will be the merging of the target model with the model to which the actor and critic are connected. We have already done some pilot research in this direction and it seems to be a feasible task. This would greatly optimize the entire model and speed up its training in terms of computing time. There was also no room left to test the SP method with other SSL algorithms that were used for SND. It would certainly be interesting to try a combination of VICReg together with an inverse model like SND-VINV. This would lead exactly to the ICM configuration.

In addition to the presented research, which is the main part of this thesis, many related ideas were created. We tested them to a certain extent, but their inclusion in the text and a more detailed description would not be entirely consistent with the main content and would rather introduce confusion. At the very beginning, a model was created, which we technically called QRND. It was a simple extension of the RND model, where we added an action to the input. QRND suffered from all the shortcomings of RND, but we just wanted to test on it the possibility of distilling representations of transitions between states, which should provide a more complex input space. The main goal was to find out whether increasing the complexity of the input delayed the collapse of the intrinsic reward, which was basically not confirmed and the model achieved a similar score to the classical RND.

However, the QRND tests led to another idea for a rather complex model inspired by the dopaminergic system technically called DOP. This model consisted of one critic and several actors (multi-head actor) that generated actions based on common input. QRND served as a motivational module and evaluated an intrinsic reward for each action. The DOP then selected the actions that led to the largest intrinsic reward. The actors tried to train themselves to generate actions leading to the maximum intrinsic reward, and the selected actor simultaneously maximized the external reward. However, this led to chaotic behavior as the actor's heads switched very quickly. Therefore, there was a need to manage this switching. This created a new control module, which, based on the accumulated input state, was supposed to decide when to give which actor the opportunity to control the agent. It turned out that such a control module can have the same structure as the controlled module, i.e. it can consist of a critic, a multi-headed actor, a motivational module. This is how it was possible to stack these modules and create a hierarchical structure that could learn to work in different time scales on individual layers. The problem was how to ensure the creation of suitable representations for individual layers. And here SSL algorithms came into play, from which the mentioned SND and SP methods were created, and DOP remained only as an idea, a proposal and a prototype. We think that simple modules (residual blocks, LSTMs, multi-head attention blocks) have always been behind the revolution in deep learning. It was possible to stack them on top of each other, increasing the capacity of the models to solve more complex tasks. And the construction of a similar module will bring a revolution to decision-making and RL. If we manage to successfully complete the DOP model, it may be one of the candidates for this position.

At the same time, we think that SSL approach to intrinsic motivation can be an inspiration for a new class of algorithms that will specialize in creating feature mapping capturing the relationship of the environment to the agent itself, since current self-supervised methods are agnostic to these relationships.

Based on the presented results in this thesis, we can conclude that self-supervised learning methods are definitely promising in the creation of novelty detectors, which can be successfully used from the point of view of intrinsic motivation and improve the agent's exploration.

Bibliography

Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M., and Hjelm, R. D. (2019). Unsupervised state representation learning in Atari. *CoRR*, abs/1906.08226.

Aubret, A., Matignon, L., and Hassas, S. (2019). A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*.

Aubret, A., Matignon, L., and Hassas, S. (2023). An information-theoretic perspective on intrinsic motivation in reinforcement learning: A survey. *Entropy*, 25.

Bachman, P., Hjelm, R. D., and Buchwalter, W. (2019). Learning representations by maximizing mutual information across views. *Advances in Neural Information Processing Systems*, 32.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. (2020). Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*.

Baldassarre, G. (2019). Intrinsic motivations and open-ended learning. arXiv:1912.13263v1 [cs.AI].

Baldassarre, G., Stafford, T., Mirolli, M., Redgrave, P., Ryan, R. M., and Barto, A. (2014). Intrinsic motivations and open-ended development in animals, humans, and robots: An overview. *Frontiers in Psychology*.

Bardes, A., Ponce, J., and LeCun, Y. (2022). VICReg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations*.

Barth-Maron, G., Hoffman, M. W., Budden, D., Dabney, W., Horgan, D., Tb, D., Muldal, A., Heess, N., and Lillicrap, T. (2018). Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515.

Bellman, R. (1957). A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684.

Bialek, W., Nemenman, I., and Tishby, N. (2001). Predictability, complexity, and learning. *Neural Computation*, 13:2409–2464.

Bialek, W. and Tishby, N. (1999). Predictive information. *arXiv preprint cond-mat/9902341*.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. arXiv:1810.12894.

Burnham, K. P. and Anderson, D. R. (2003). *Model Selection and Multimodel Inference: a practical information-theoretic approach.* Springer Science & Business Media.

Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2018). Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. *Advances in Neural Information Processing Systems*, 33:9912–9924.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607. PMLR.

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. E. (2020b). Big self-supervised models are strong semi-supervised learners. *Advances in Neural Information Processing Systems*, 33:22243–22255.

Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 539–546 vol. 1.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2019). Leveraging procedural generation to benchmark reinforcement learning. *CoRR*, abs/1912.01588.

Crutchfield, J. P. (1990). Information and its metric. In *Nonlinear Structures in Physical Systems*, pages 119–130. Springer.

Csikszentmihalyi, M. (1991). *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, NY.

Festinger, L. (1962). *A Theory of Cognitive Dissonance*. Stanford university press.

Graves, A. (2011). Practical variational inference for neural networks. *Advances in neural information processing systems*, 24.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. (2020). Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, pages 297–304. JMLR Workshop and Conference Proceedings.

Haber, N., Mrowca, D., Wang, S., Fei-Fei, L. F., and Yamins, D. L. (2018). Learning to play with intrinsically-motivated, self-aware agents. *Advances in neural information processing systems*, 31.

Harlow, H. F. (1950). Learning and satiation of response in intrinsically motivated complex puzzle performance by monkeys. *Journal of Comparative and Physiological Psychology*, 43(4):289–294.

Hazan, E., Kakade, S., Singh, K., and Van Soest, A. (2019). Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*.

Henaff, O. (2020). Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning*, pages 4182–4192. PMLR.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. (2018). Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117.

Hsu, C. C.-Y., Mendler-Dünner, C., and Hardt, M. (2020). Revisiting design choices in proximal policy optimization. *arXiv preprint arXiv:2009.10897*.

Hull, C. L. (1943). *Principles of Behavior: An Introduction to Behavior Theory*. New York, London: D. Appleton-Century Company, Inc.

Hunt, J. (1965). Intrinsic motivation and its role in psychological development. In *Nebraska symposium on motivation*, volume 13, pages 189–282. University of Nebraska Press.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., Maschinot, A., Liu, C., and Krishnan, D. (2020). Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673.

Kim, H., Kim, J., Jeong, Y., Levine, S., and Song, H. O. (2018). EMI: Exploration with mutual information. arXiv:1810.01176.

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. arXiv:1312.6114.

Kramer, M. A. (1992). Autoassociative neural networks. *Computers & Chemical Engineering*, 16(4):313–328.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. (2019). Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv:1509.02971.

Liu, H. and Abbeel, P. (2021). Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34:18459–18473.

Liu, X., Li, Q., and Li, Y. (2022). Count-based exploration via embedded state space for deep reinforcement learning. *Wireless Communications and Mobile Computing*, 2022.

Lowe, R., Wu, Y. I., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30.

Machado, M. C., Bellemare, M. G., and Bowling, M. (2020). Count-based exploration with the successor representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5125–5133.

Martin, J., Sasikumar, S. N., Everitt, T., and Hutter, M. (2017). Count-based exploration in feature space for reinforcement learning. *arXiv preprint arXiv:1706.08090*.

Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., Goodfellow, I. J., and Pouget-Abadie, J. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems*, 27:2672–2680.

Misra, I. and Maaten, L. v. d. (2020). Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6707–6717.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with deep reinforcement learning. In *Neural Information Processing Systems*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Montgomery, K. C. (1954). The role of the exploratory drive in learning. *Journal of Comparative and Physiological Psychology*, 47(1):60–64.

Montúfar, G., Ghazi-Zahedi, K., and Ay, N. (2016). Information theoretically aided reinforcement learning for embodied agents. abs/1605.09735.

Morris, L. S., Grehl, M. M., Rutter, S. B., Mehta, M., and Westwater, M. L. (2022). On what motivates us: a detailed review of intrinsic v. extrinsic motivation. *Psychological Medicine*, 52(10):1801–1816.

Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. (2017). Count-based exploration with neural density models. In *International Conference on Machine Learning*, pages 2721–2730.

Oudeyer, P.-Y. and Kaplan, F. (2009). What is intrinsic motivation? a typology of computational approaches. *Frontiers in Neurorobotics*, 1:6.

Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54–71.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. arXiv:1705.05363.

Pecháč, M., Chovanec, M., and Farkaš, I. (2023). Exploration by self-supervised exploitation. *arXiv preprint arXiv:2302.11563*.

Pecháč, M. and Farkaš, I. (2021). Intrinsic motivation model based on reward gating. In *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part IV 30*, pages 688–699. Springer.

Pecháč, M. and Farkaš, I. (2022). Intrinsic motivation based on feature extractor distillation. In *Kognice a umělý život XX*, pages 84–91. ČVUT v Praze.

Rochat, P. and Striano, T. (2000). Perceived self in infancy. *Infant Behavior and Development*, 23(3-4):513–530.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England.

Ryan, R. and Deci, E. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary Educational Psychology*, 25(1):54–67.

Sadowski, C. and Levin, G. (2007). Simhash: Hash-based similarity detection.

Santucci, V. G., Baldassarre, G., and Mirolli, M. (2012). Intrinsic motivation mechanisms for competence acquisition. In *International Conference on Development and Learning and on Epigenetic Robotics (icdl)*, pages 1–6. IEEE.

Santucci, V. G., Baldassarre, G., and Mirolli, M. (2013). Which is the best intrinsic motivation signal for learning multiple skills? *Frontiers in Neurorobotics*, 7:22.

Santucci, V. G., Baldassarre, G., and Mirolli, M. (2014). Cumulative learning through intrinsic reinforcements. In *Evolution, Complexity and Artificial Life*, pages 107–122. Springer.

Schmidhuber, J. (1991). A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 815–823.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv:1707.06347.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. (2020). Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR.

Seo, Y., Chen, L., Shin, J., Lee, H., Abbeel, P., and Lee, K. (2021). State entropy maximization with random encoders for efficient exploration. In *International Conference on Machine Learning*, pages 9443–9454. PMLR.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Shyam, P., Jaśkowski, W., and Gomez, F. (2019). Model-based active exploration. In *International Conference on Machine Learning*, pages 5779–5788.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

Singh, H., Misra, N., Hnizdo, V., Fedorowicz, A., and Demchuk, E. (2003). Nearest neighbor estimates of entropy. *American Journal of Mathematical and Management Sciences*, 23(3-4):301–321.

Sohn, K. (2016a). Improved deep metric learning with multi-class n-pair loss objective. *Advances in Neural Information Processing Systems*, 29.

Sohn, K. (2016b). Improved deep metric learning with multi-class n-pair loss objective. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Souchleris, K., Sidiropoulos, G. K., and Papakostas, G. A. (2023). Reinforcement learning in game industry — review, prospects and challenges. *Applied Sciences*.

Strehl, A. L. and Littman, M. L. (2008). An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*, volume 135. MIT Press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

Tang, H. et al. (2017). #Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762.

Van Den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. In *International conference on machine learning*, pages 1747–1756. PMLR.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Veness, J., Ng, K. S., Hutter, M., and Bowling, M. (2012). Context tree switching. In *2012 data compression conference*, pages 327–336. IEEE.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

Watson, J. S. (1972). Smiling, cooing, and "the game". *Merrill-Palmer Quarterly of Behavior and Development*, 18:323–339.

White, R. W. (1959). Motivation reconsidered: The concept of competence. *Psychological Review*, 66(5):297–333.

Wu, Z., Xiong, Y., Yu, S. X., and Lin, D. (2018). Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer vision and Pattern Recognition*, pages 3733–3742.

Ye, M., Zhang, X., Yuen, P. C., and Chang, S.-F. (2019). Unsupervised embedding learning via invariant and spreading instance feature. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6210–6219.

Zahedi, K., Martius, G., and Ay, N. (2013). Linear combination of one-step predictive information with an external reward in an episodic policy gradient setting: a critical analysis. *CoRR*, abs/1309.6989.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. *CoRR*, abs/2103.03230.