École Polytechnique Fédérale de Lausanne
Laboratory of Computational Neuroscience

Learning Abstract Representations in Reinforcement Learning via Model Sparsity

by Sergei Volodin

Master Semester Project Report

Prof. Wulfram Gerstner
Advisor

Johanni Brea
Supervisor

CH-1015 Lausanne

July 19, 2021

Abstract

The problem of learning good abstract representations from high-dimensional states is arguably one of the foundational tasks in of Artificial Intelligence. One of the theoretical and philosophical approaches to learn abstract representations is the Consciousness Prior proposed by Yoshua Bengio. One of the key components in that proposal is the sparsity of the transition model, which hypothetically leads to good learned abstractions. In our project, we propose a practical framework for learning abstractions in Reinforcement Learning via sparsity of the transition model. To test it, we design a simple environment where abstractions can be learned. The results show that we are able to recover the correct representation. We provide theoretical formulation of the problem and the explanation of the results. We provide exciting future research directions and concrete questions in the domain of learning good abstractions.

Contents

Abstract (English/Français) **2**

0.1 Introduction . . . 4

0.2 Problem setup . . . 5

0.3 Solution and theory . . . 6

0.3.1 Environment design . . . 7

0.3.2 Losses design and analysis . . . 7

0.3.3 Training dynamics . . . 9

0.3.4 Analysis of the linear case . . . 10

0.4 Experiments . . . 12

0.5 Conclusion . . . 14

0.5.1 Future directions . . . 14

0.5.2 Related work . . . 17

0.5.3 Acknowledgements . . . 17

Bibliography **18**

0.1 Introduction

In this project, we focus on the problem of learning abstract representations from high-dimensional data. This can be seen as the ability to find long-term patterns in data that compress it the most. For example, after seeing a ball many times we give it a name "ball" and then try to re-use the same word for other similarly looking objects [1]. This way, we compress the high-dimensional visual data and replace it with only the relevant information. We take a practical approach to learning abstractions by constraining the model to be sparse. Specifically, when learning transition dynamics of a reinforcement learning environment, the transition function is required to have low description length. This forces the learned representation to be "compressable" in a sense that it is computationally easy to transition from a state to the successive state.

This work is based on a theoretical proposal by Yoshuo Bengio (the Consciousness Prior) [1]. Specifically, the idea to obtain good learned representations by regularizing the model with sparsity is taken from that paper. We make that theoretical proposal concrete by considering Reinforcement Learning dynamics where previous states and actions determine future states [3]. We collect data from this dynamical system of an agent and an environment, and fit a causal model to the empirical transition dynamics. We additionally add a decoder component which computes internally-used features from raw observations. The model is fitted on the features. The model is regularized for sparsity, and that should lead to good representation.

We propose a simple toy environment. It has linear transition function and the transition matrix is sparse. Then, states are artificially made less sparse by applying a general linear transformation. Because of it, the observations that the agent receives are not sparse, but can be made so by finding the right linear transformation. We show that even in that simple case, the problem of learning these "concepts" is NP-hard (by reduction from Sparse Dictionary Learning).

To solve this problem of learning abstraction in linear case we propose an heuristic algorithm. It first estimates the transition dynamics for the observations. Then, the resulting function is simplified by decomposing into a basis transformation ("decoder") and a sparse transition model.

The agent receives feedback from the model as well via the Curiosity reward [2]: if the decoder matrix becomes degenerate (some information is lost), the model becomes overfit and has high test loss (where the agent generates "test" distribution by optimizing for off-distribution data).

The obtained environment model's sparsity fits well within the causality framework (Judea Pearl). For example, if the model is linear, then the number of non-zero components in the model corresponds directly to the number of edges in the causal graph. Training the agent with a curiosity reward can be seen as performing causal interventions in the environment, in a sense that the policy tries to change the values of causal nodes to explore.

Contribution. We propose a toy environment with linear dynamics to learn abstract representations on. We show that even for this simple environment, the problem is NP-hard. We design an algorithm to solve this problem approximately, and test it in experiments. The results show that dimensionality of the problem helps learning better abstractions. Code can be found at `https://github.com/sergeivolodin/`

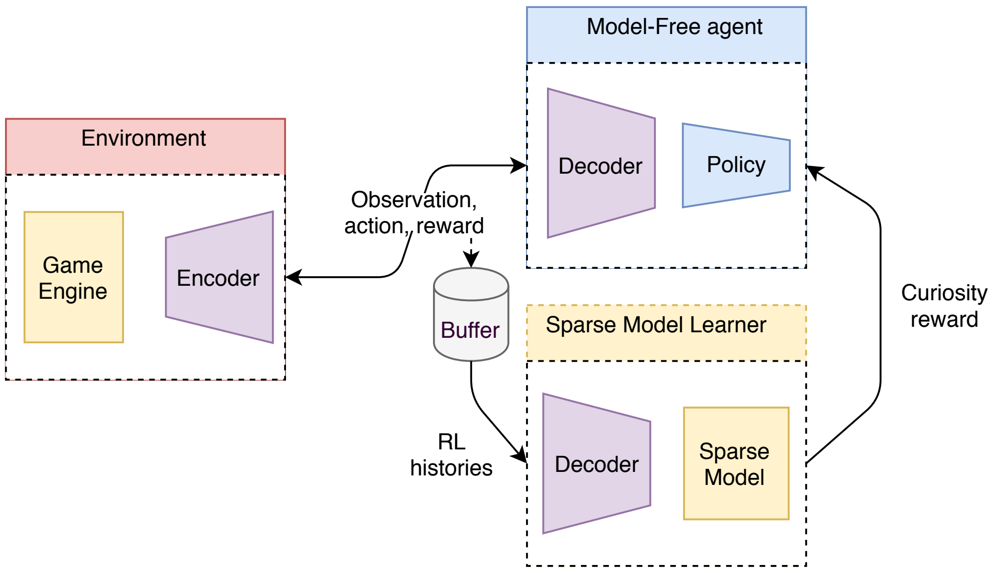


Figure 1: Proposed agent architecture. The environment consists of a Game Engine (for example, the old code running Atari games) which implements the game logic, and the Encoder – a function mapping latent states of the game engine to observations. The model-free agent samples down the observations (for example, via a CNN), and then trains a policy. The Sparse Model Learner decodes observations as well, and then fits a model to environment transition dynamics, with an additional regularization for model sparsity. The loss of the learned model on novel data is given to the agent as a reward to encourage exploration.

`causality-disentanglement-rl`. The slides can be found at `https://docs.google.com/presentation/d/1iYfbMrMZUn88cet36qe0h-2oHG9HpbFfEZHTVp4xlik/edit?usp=sharing`

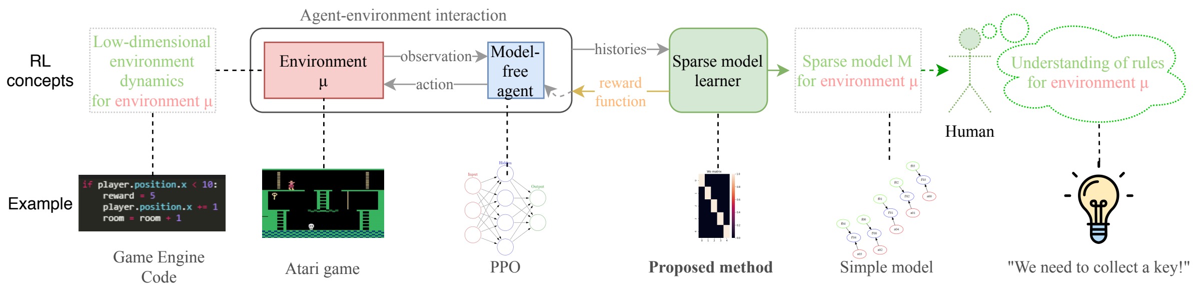
0.2 Problem setup

We use the standard RL notation with an environment μ and agent with a policy π. They interact and provide histories of observations, actions and rewards $(o_1, a_1, r_1), ..., (o_T, a_T, r_T)$. The environment internally has some *state* s_t dynamics, which is then converted to observations given as an output of the environment. The agent receives pre-processed states from the environment, and the reward is pre-processed as well. Specifically, we add the Curiosity reward (the agent is rewarded for "disproving" its model of the environment).

We assume that the environment has some latent low-dimensional structure which corresponds to states s_t. Observations can be computed from the states: $o_t = E(s_t)$ where E is an *encoder function*. The agent pre-processes observations with a *decoder function* D and obtains features $f_t = D(o_t)$. In this setup, both encoder's and the decoder's output depends only on the *current* state/observation, but this restriction can be lifted. In this paper, we only focus on the simple case.

The agent fits a transition model $\hat{f}_{t+1} = M(f_t, a_t)$[1]. Note that we only consider 1-step de-

[1]We do not predict the reward explicitly for clarity of notation, as it can be added as as part of observation.


Agent-environment interaction
RL concepts
Low-dimensional environment dynamics for environment μ
Environment μ
observation
action
Model-free agent
histories
reward function
Sparse model learner
Sparse model M for environment μ
Human
Understanding of rules for environment μ
Example
Game Engine Code
Atari game
PPO
Proposed method
Simple model
"We need to collect a key!"

```
if player.position.x < 10:
    reward = 5
    player.position.x += 1
    room = room + 1
```

Figure 2: Explanation of the framework. In a typical RL environment, the (hidden from the agent) low-dimensional game engine code generates high-dimensional observations. Then, the policy learns to perform well in that environment. Our method tries to find a representation giving a sparse transition model, hopefully recovering the original simple dynamics again. This data can be used to further analyze the environment and interpret the behavior of the agent.

pendencies in our model (Markov assumption), but our framework is easily extendable to more time-steps. The model is regularized for sparsity with some function $S(M)$ (for example, l_1-norm). To fit the model, we design a fit loss L_f. The decoder D can become degenerate (for example, always output 0), and to avoid that, we introduce the non-degeneracy loss L_d. The curiosity reward is then equal to L_f – a metric that shows how well novel data fits the current model.

0.3 Solution and theory

To make a concrete advance on the problem of learning abstractions, we consider a linear case. Specifically, the environment's latent state evolves as $s_{t+1} = A_s s_t + A_a a_t$, where $s \in \mathbb{R}^{s 2}$, A_s is a matrix of size $s \times s$, and $A_a \colon s \times a$ (s rows and a columns). t is the time index $t \in \{0, 1, 2, ...\}$. Actions a_t are encoded as one-hot vectors.

In the same vein, the encoder function E is also linear, and $o_t = E s_t$ where the right-hand side is a matrix E of size $o \times s$, $o_t \in \mathbb{R}^o$ is a vector, and $E s_t$ is a matrix-vector product.

The agent consists of a policy π mapping observations and history oh_t into probability distributions over actions $a_t \sim \pi(oh_t)$; a decoder D, a matrix of size $f \times o$ where f is the *feature* dimensionality; and a model $M = [M_f, M_a]$, two matrices of size $M_f \colon f \times f$ and $M_a \times f \times a$. If a decoder is applied to an observation, the result is in the feature space: $f_t = D o_t$ where $f_t \in \mathbb{R}^f$. The policy consists of two parts: $\pi(o_t) = \pi'(f_t) = \pi'(D a_t)$. Thus, it only relies on the features (the learned representation) and not on the observations. The model is learned from the obtained history $H = (o_1, a_1, r_1), ..., (o_T, a_T, r_T)$ stored in a replay buffer: $M_{t+1} = \mathcal{M}(H_t)$. The policy π is trained using any reinforcement learning algorithm from histories: $\pi_{t+1} = \Pi(H_t)$.

When the environment gives a reward r_t, the agent's perceived reward is $r_t + L_f$ where the model fit loss L_f is computed on the current time-step o_{t-1}, a_{t-1}, o_t. The summary of the training procedure is shown below.

[2]We abuse the notation and write s a) as index for matrices b) as the state vector c) as the state dimensionality. Same for actions, observations and features.

One step of the training algorithm is described below.

1. Model M, decoder D and the policy π are initialized randomly
2. We repeat the following steps until convergence
3. Agent is run for an episode and the data H_t is collected.
4. Agent is trained using the obtained data (in a standard way), assuming the decoder D is differentiable
5. Loss $L_f + S(M)$ is minimized to update our model, which updates M and D

0.3.1 Environment design

To reduce the amount of work for a proof-of-concept experiment, we make it as simple as possible, while preserving the qualitative properties of the problem (high-dimensional non-sparse representation with some underlying low-dimensional sparse dynamics that we want to uncover). We design a simple environment with these properties.

We define the `Vector Increment` environment with a state $s \in \mathbb{R}^s$. The actions are $a \in \{1, ..., s\}$ which we sometimes write in terms of a one-hot representation of dimension s. Given an action vector a_t and being in state s_t, the environment computes $s_{t+1} = s_t + a_t$. So, for example, action $a = 3$ would increment the third component of s. The matrices A_s and A_a are identity matrices of size $s \times s$ (four our environment, $a \equiv s$). Identity matrices are the most sparse (they have only s non-zero components out of s^2) among non-degenerate matrices of size $s \times s$.

The reward of the environment encourages the agent to stay on the "diagonal": $r_t = \frac{\max_i\{s_t^i\} - s_t^{a_t}}{\max_i\{s_t^i\} - \min_j\{s_t^j\}}$. The optimal strategy is thus to always choose the smallest component: $a_t^* = \arg\min_i s_t^i$. This results in the desired behavior – the vector stays close to the "diagonal" $s_t^1 = s_t^2 = ... = s_t^s$. This reward is chosen so that the agent explores all possible actions.

Observation is computed by applying the encoder E to the states: $o_t = Es_t$. The encoder E is initialized randomly and is fixed throughout the usage of the environment.

For this environment we have a clear expectation of what we want from the agent. We expect it to learn $D = E^{-1}$ and then it uncovers matrices M_f, M_a with sparsity of $S(A_s)$ and $S(A_a)$. Since M has to be non-degenerate as well, and A had best sparsity, M has best sparsity as well. Note that the problem is invariant under feature order permutations, thus, the solution is not unique, and it is not neccery that $M = A$.

Note that for this problem we don't expect that current methods would learn the disentangled latent dynamics because it depends crucially on the sparsity of the model rather than on its prediction quality.

0.3.2 Losses design and analysis

Now we define concretely what L_f, L_d and S are. We develop several method to fit the data.

Losses defined from data:

1. $L_f = \|Do_{t+1} - MDo_t\|$, $L_d = \|RDo_t - o_t\|$ for a reconstructor matrix $R\colon o \times f$. Both losses are convex if we consider the variables D, M, R separately. It is not convex if we consider all of them together because of the product MD.

2. $L_f = \|o_{t+1} - RMDo_t\|$, $L_d = \|RDo_t - o_t\|$

3. $L_f = \|Do_{t+1} - MDo_t\|$, $L_d = \|D^\dagger\|$

For each of these cases, we can either use two optimizers for L_f and L_d, or run an optimizer on a linear combination of these losses: $L = \lambda_f L_f + \lambda_d L_f \to \min$. The optimization procedure will often find saddle points and local minima. A good metric to diagnose the training procedure is to look at the cosine between gradients of two losses. If $\cos \measuredangle(\nabla_D L_f, \nabla_D L_d) = -1$, then the training procedure will likely to stagnate: losses "pull" the decoder D in different directions.

Losses defined from the observation model W. We can split the problem of learning M into two stages. First, we fit a model of observation dynamics W, for our case it would be $\hat{o}_{t+1} = W_o o_t + W_a a_t$ for matrices $W_o \times o \times o$ and $W_a\colon o \times a$. We do so by minimizing the loss $L_o|t = \|\hat{o}_{t+1} - o_{t+1}\|$. Next, given matrices W_o and W_a, we estimate the best decoder D and the model M_f, M_a such that the model is most sparse.

Sparsity losses:

1. $S(M) = \|M_f\|_0 + \|M_a\|_0$. The issue with this loss is that it is non-differentiable

2. $S(M) = \lambda \cdot (\|M_f\|_1 + \|M_a\|_1)$ with $S(M)$ added to the loss. We need to choose the λ parameter

3. Same loss, but with projection onto an l_1-ball instead of λ. Here we need to set the ball radius explicitly

4. $S(M) = \|m_T\|_1$ where m is the matrices M_f, M_a flattened into a vector, and $m = [m_H, m_T]$ – head and the tail of that vector. Here we explicitly set the number of components we want to preserve

The l_1 penalty can be implemented either as a regularizer or as a projection step. The problem with the first option is that the parameter's effect depends on the magnitude of the loss. Since we have a decoder with non-fixed norm of columns, the magnitude of the loss is different for different random seeds. A problem with projection is that the ball size can be different as well. Another way is to use l_p norm for $p \in (0, 1)$. Yet another way is to use hard sparsity where weights are zeroed externally.

A more principled approach would be to automatically select the regularization parameter based on the loss. If the loss is low, we can increase regularization, since we already found a way to fit the data. However, if the loss is high for a long time, regularization is likely affecting the model too much and not allowing it to converge. This iterative self-regulating system would result in a process similar to simulated annealing. Specifically, the temperature parameter here is the regularization parameter. If it is low, then the temperature is high, and the model can consider many alternatives. If regularization becomes stronger, the system converges to a local minimum with more crisp abstractions. The main problem here is that the model might choose the wrong "valley" to go into. The idea of our approach is to detect that automatically via tracking the magnitude of the loss over time. If it does not improve, it makes sense to relax constraints, since it is likely that we are in the wrong "valley".

We use tf-agents to implement this and we use the following components in the following way:

1. The keras model M, the decoder D and the policy π are initialized randomly. The agent's reward is $r'_t = r_t + L_f|t$. The policy $\pi(o_t) = \pi'(Do_t)$ only depends on the features.
2. The standard tf-agents driver runs episode loops with agent's policy π, environment μ and reward r'_t for N_{agent} episodes, giving histories H_t. The agent's performance loss on its own replay data is minimized with D trainable: $\pi_{t+1} = \Pi(H_t)$
3. Another tf-agents driver runs the causality training loop for $N_{causality}$ batches from the replay buffer: the error $L_f + S(M)$ is minimized on replay data H_t, and the model is updated: $(D_{t+1}, M_{t+1}) = \mathcal{DM}(H_t)$. The agent is not invoked here, and actions are taken from the buffer. D is trainable here as well.
4. Agent's replay buffer is cleaned (since reward r'_t has changed due to update in M). At later stages when $L_f \approx 0$, we can skip this step, as the environment dynamics stabilizes.
5. Repeat the loop from item 3.

0.3.3 Training dynamics

In all our setups, we have a complicated dynamics that is not decomposable into just RL or just supervised learning due to the interactions between components: policy affects observations, which affect the model, which affects the reward, which affects the policy. This is how the agent and the model have a closed feedback loop. In addition, the decoder D is trainable both from the agent and from the model losses. The problem can be decomposed into two convex (in the linear case) supervised learning problems. For example, the option $L_f + S(M) = \|Do_{t+1} - MDo_t\| + S(M)$, $L_d = \|RDo_t - o_t\|$ consists of just two supervised learning problems.

Let's consider two losses L_f and L_d which both affect the decoder. Like in GANs, one of the losses is responsible for the quality of the model (like the generator loss), and the other one is responsible for keeping the decoder within the correct range of parameters (like the discriminator loss). While both losses separately $L_f + S(M)$ and L_d are convex (for our linear case) in the spaces of M and D separately, they are not convex together. Specifically, the matrix product MD makes the L_f loss non-convex.

An analogy for this game would be a teacher explaining to a student (decoder) what to do. The student does most of the work independently (loss L_f), but she needs to be guided to avoid bad local minima and other degenerate solutions (loss L_d). Naturally, there should be a balance between L_f and L_d. For example, if L_d is too "strong" (either because it is optimized using a more powerful optimizer, or because it has a large coefficient in the linear combination for the total loss), then the model will quickly converge to a non-degenerate solution (L_d is low), but it will not fit the data (L_f is high) – the model is too constrained to stay at the point which is the furthest from being degenerate (an analogy would be a teacher who is overly dogmatic and the student who cannot do any independent exploration because of that). On the other hand, if L_f is too strong, the model will likely converge to low L_f but high L_d. This corresponds to the case where the teacher ignores mistakes in the student's work, and the student thinks everything is going fine.

Therefore, the success of finding a good minimum depends on the balance between L_d and L_f.

0.3.4 Analysis of the linear case

Our overall goal now is to find the most sparse matrix M_s, M_a and a decoder D such that the model fits the observations dynamics transformed by the decoder.

First, let's understand what the optimal observation model W should be. By rewriting $o_t = Es_t$, we get $L_o|t = \|W_o Es_t + W_a a_t - Es_{t+1}\|$. Now, we know from the environment dynamics that $s_{t+1} = A_s s_t + A_a a_t$. Therefore, $L_o|t = \|W_o Es_t + W_a a_t - EA_s s_t - EA_a a_t\| = \|(W_o E - EA_s)s_t + (W_a - EA_a)a_t\|$. Under reasonable non-degeneracy assumption (see section 0.3.4) we get that $W_o E = EA_s$ and $W_a = EA_a$. Under more non-degeneracy conditions, we get $W_o = EA_s E^\dagger$.

Next, let's consider the model M that the agent needs to learn. If the model has converged to zero loss (which is possible under non-degeneracy conditions), then $L_f = 0$, and, therefore, $Do_{t+1} = MDo_t$. Using the equality $o_{t+1} = W_o o_t + W_a a_t$, we get that $DW_o = M_s D$ and $DW_a = M_a$. Under more non-degeneracy assumptions, this means $M_s = DW_o D^\dagger$ and $M_a = DW_a$.

Now, if we combine the two equations together, we get: $DEA_s = M_s DE$ and $DEA_a = M_a$ which gives $M_s = DEA_s(DE)^{-1}$. If we re-parameterize with $Z = DE$, $M_a = ZA_a$ and $M_s = ZA_s Z^{-1}$. Now, the objective is to minimize $S(M) = S(ZA_a) + S(ZA_s Z^{-1})$. Note that here the only free variable is the decoder D. Given a non-degenerate decoder, the model is computed from it by the equations given here.

All of the formulations of the problem in the previous section are equivalent mathematically – solution for one of the formulations can be transformed into a solution for another one. However, computationally they are vastly different in our experiments.

Let's consider the computational class of the problem that we have defined. If $W_s = 0$ (the environment has no memory, actions just determine the next stage. This is the case of multi-armed bandits), the problem becomes $S(M) = S(DW_a) \to \min$ s.t. D is non-degenerate. The problem can be interpreted in the following way. Given a set of vectors $[W_a^1, ..., W_a^a] = W_a$ (each of the vectors is a column of W_a, we need to find a basis transformation D such that the resulting vectors are most sparse. This is exactly the formulation of the Sparse Dictionary Learning problem. This problem is known to be NP-hard. Now, our problem is even more complex in case if $W_o \neq 0$. In this case, apart from making vectors of W_a sparse under basis transformation D, we also want to make the matrix W_o sparse, which is transformed with $DW_o D^\dagger$. Thus, Sparse Dictionary Learning is a special case of our linear abstraction learning problem. Thus, our problem is NP-hard as well.

Note that in the other extreme case if $A_a = 0$ (actions do not influence the state in any way), the problem becomes very simple. We need to find a sparse basis for a matrix W_o using a decoder D: $M_s = DW_o D^\dagger$. Consider the rank of $rkW_o = r$. Now, M_s cannot have less than r non-zero components, or otherwise it would have rank less than r, and some information will be lost. And to achieve only r non-zero components in M_s, we simply can diagonalize the matrix W_o using any standard algorithm. Therefore, the case of $A_a = 0$ is solvable in polynomial time.

Non-degeneracy conditions in the linear case

If we solve for $L_f = 0$, we get $(DEA_s - M_f DE)s_t = 0$ and $(DEA_a - M_a)a_t = 0$ for all visited states s_t and taken actions a_t. Consider the equation $(DEA_s - M_f DE)s_t = 0$. Denote $Q_s = DEA_s - M_f DE$, matrix of size $f \times s$. If there are states $s_1, ..., s_t$ such that $S = [s_1, ..., s_s]$ has rank s, we have $Q_s S = 0$ leading to $Q_s = 0$ (since we can multiply both sides by $S^\dagger$). We call this case "states are non-degenrate". The same logic applies for actions. If both actions and states are non-degenerate, then $Q_s = 0$ and $Q_a = 0$. If they are not, $Q_s = 0$ and $Q_a = 0$ is also a sufficient condition for $L_f = 0$, but not a necessary one. Let us consider the case of degenerate states. It would mean that some of the state components is always a linear combination of other components. In that case, we can safely remove that component. This would only increase the sparsity of the transition dynamics $S(A)$.

From this point on, we assume that states and actions are non-degenerate. Then, it leads to $Q_s = 0$ and $Q_a = 0$. This means that $DEA_s = M_f DE$ and $DEA_a = M_a$. Denote $Z = DE$, an $f \times s$ matrix.

Now, if $o < s$, it means that some information is lost when encoding the states, and the environment is partially-observable. In our current framework it is impossible to reconstruct the correct dynamics, as our model only considera 1-step dynamics. Therefore, we assume $o \geq s$.

If $o > s$, it means that some of the components are not required for the reconstruction, and some of them can be simply eliminated. Thus, we can safely assume $o = s$.

Now consider the feature dimensionality f. If $f < o$, we cannot make $L_f = L_d = 0$ because otherwise it would have been possible to find $o > f$ linearly independent vectors in an f-dimensional vector space, which is impossible. If $f > o$, the optimal solution would zero the extra components out, since it would lead to most sparsity. Overall, we can assume $f = o$ without loss of generality.

While theoretically we can assume $f = o = s$, in real world settings, we have $s < o > f$ with $s = f$. States and features are low-dimensional (for example, state of a card game), while the observations are high-dimensional (for example, images from a camera looking at the cards and other players). In this case, the relationship between the states and observations is highly non-linear: there is no simple function to translate the game state into the image that the camera would see. Even if that function was linear, we do not know apriori which features in the model can be discarded. Thus, the case $s < o > f$ and $s = f$ is practically interesting and we will consider this case.

We thus have two non-degeneracy conditions. First, the observations must be such that the states can be reconstructed from them. This means that $o \geq s$ and E is a full-rank matrix. The same goes for the features: we should be able to reconstruct the observations from features, which means that $f \geq s$ and D has a rank of s. In case of $f = s$, D must be full-rank.

Define $Z = DE\colon f \times s$ and consider matrix $Z^T Z$ of size $s \times s$. Since D and E were full-rank, the square matrix $Z^T Z$ is non-degenerate, and there exist a left pseudo-inverse $Z^\dagger = (Z^T Z)^{-1} Z^T$.

Therefore, we obtain $A_s = Z^\dagger M_f Z$.

To express M_f via A_s, we assume that $f = s$. If so, ZZ^T is a square non-degenerate matrix as well, and $M_f = ZA_sZ^T(ZZ^T)^{-1} = ZA_sZ^\dagger$, where $Z^\dagger = Z^T(ZZ^T)^{-1}$, the right pseudo-inverse. This is an explicit formula determining the model given the decoder. Now, for actions, we have $M_a = DEA_a$. So, if the decoder is fixed, the problem reduces to fitting a linear model to data in the realizable non-degenerate case, which leads to a unique solution for M_f and M_a.

Therefore, in the non-degenerate case (full-rank states, actions, A_s, A_a, E), the problem can be re-formulated as: $S(ZA_sZ^{-1}) + S(ZA_a) \to \min_D$ s.t. D has full rank. Note that in case if $A_s = 0$, this is precisely the formulation of Sparse Dictionary Learning. Thus, the problem is NP-hard, and we do not expect to find a good explicit solution (only heuristics).

Mathematical interpretation of our problem is to find a basis of size f, $Z\colon f \times s$ for a matrix $A_s\colon s \times s$ and a set of vectors $(A_a^1, ..., A_a^s) = A_a\colon s \times a$ such that they are most sparse together.

The total loss cannot be made convex under any re-parametrization. Indeed, if we permute rows of decoder in an optimial solution, the resulting solution has the same value, but the points in the middle are not (they are not sparse).

Now, in our case if $A_s = I$ is an indentity matrix, we always have $W_o = I$ and $M_f = I$, since in any basis an identity transformation looks the same. This is the only non-degenerate matrix that is invariant under basis transformations – all other cases (even just diagonal A_s) will look different after a suitable basis transformation.

The decoder matrix D has fo components and 1 constraint for non-degeneracy. This leaves $fo - 1$ free components. By fixing the magnitude of rows of D, we gain another f constraints, and by fixing their order, we get $f - 1$ constrains. Thus, in total we have $fo - 2f$ free components.

Consider the loss $L_f|t = \|Do_{t+1} - M_fDo_t - M_aa_t\|$ and consider a still point $\nabla L_f = 0$. It gives the following tensor equation:

$$(M_f)_{\alpha\alpha} - (D^{-1}M_fD)_{\beta\beta} = (DE)_{\alpha\gamma}\frac{\partial D^{-1}_{\gamma\delta}}{\partial D_{\alpha\beta}}(E^{-1}A_s)_{\delta\beta}$$

On the right-hand side we use the Einstein tensor notation in a sense that we sum over γ and δ.

In total, this equation determines fo equations, as $\alpha \in [f]$ and $\beta \in [o]$. The equation is over M_f and D, which meanss that in total we have $f^2 + fo$ unknowns. So, we have $f^2 + fo - fo = f^2$ free variables. As previously noted, we have $2f$ constraints on D coming from fixing the scale and order of rows. Thus, now we have $f^2 - 2f$ free parameters. In case if $f = 2$, the set of fixed points is discrete. However, if $f > 3$, it is possible to have connected manifolds of fixed points.

0.4 Experiments

We test experimentally various components of the setup, and then test the whole setup together. We consider several loss designs:

1. (as two problems) Estimate $\hat{o}_{t+1} = W_oo_t + W_aa_t$, then solve $\|DW_oD^\dagger\|_1 + \|DW_a\|_1 + \lambda\|D^\dagger\|_2 \to \min$

2. (as one problem). Here, we set $L_f = \|Do_{t+1} - M_f Do_t - M_a a_t\|$ or $L_f = \|o_{t+1} - RM_f Do_t - RM_a a_t\|$; and $L_d = \|RDo_t - o_t\|$ or $L_d = \|D^\dagger\|$. Finally, we use $S(M) = \|M\|_1$ or SparseSep, or the thresholding sparsity regularization from Keras.

Setups. First, we try to learn a sparse model on synthetic data (generated without the agent). Next, we plug the sparse model learner component together with RL and measure the results.

Synthetic data. The preliminary results on synthetic data show that only one problem formulation works – the one with two problems decoupled from each other. See `l1-sparsity-playground-encoder-decoder.ipynb`.

All components together. We use VectorIncrement environment with a time limit of 20 and we use REINFORCE with default parameters as an agent. We consider dimensionality $s = o = f$ and $s = 2, 3, 5, 10$. We set $p = 1$, curiosity reward coefficient $\alpha = 1$. The coefficient for regularization when combining L_f and L_d $\varepsilon = 1$. We limit the dataset size to 5000 and re-sample from the dataset to remove old points. The sparse model learner is run every 10 iterations of agent training. We collect 20 episodes at each agent training iteration.

The metrics are:

- `train_return`. Reward from the environment with an added curiosity reward, obtained by agent when training
- `eval_return`. Reward from the environment when evaluating the agent
- `train_loss`. The loss from the REINFORCE trainer.
- `cos`. The cosine between gradients of L_f and L_d.
- `nnz`. Number of non-zero components in the model. This is the main metric of success, when combined with zero L_f and L_d losses.
- `fit_loss`. The fit L_f loss
- `rec_loss`. The reconstruction non-degeneracy loss L_d.

The success ratios for different dimensions are shown in Figure 3. It can be seen that the success rates are reasonable for dimensions of 3 and 5, but are zero for dimensions of 2 and 10. We explain it by the fact that for dimension of 2 the loss is too non-convex to converge to a good representation (the learner is stuck in a local minimum), and for the dimension of 10 the l_1 regularization becomes impractical: it selects many close-to-0 components instead of selecting only few with a bit higher value due to the curse of dimensionality.

Figure 7 shows that the results depend crucially on the random seed of the encoder. Some encoders give observations that are easier to disentangle compared to other random seeds. For example, random seed 42 gives non-zero success rate for dimensions of $3, 5$, but random seed 1 gives zero success ratio for both.

Environment models as graphs are shown in details for dimension of 5 in Figure 4. It can be seen that the model fitted on raw observations is quite complex and not sparse: there is no clear separation between zero and non-zero components in the histogram. On the contrary, or method can recover the sparse graph.

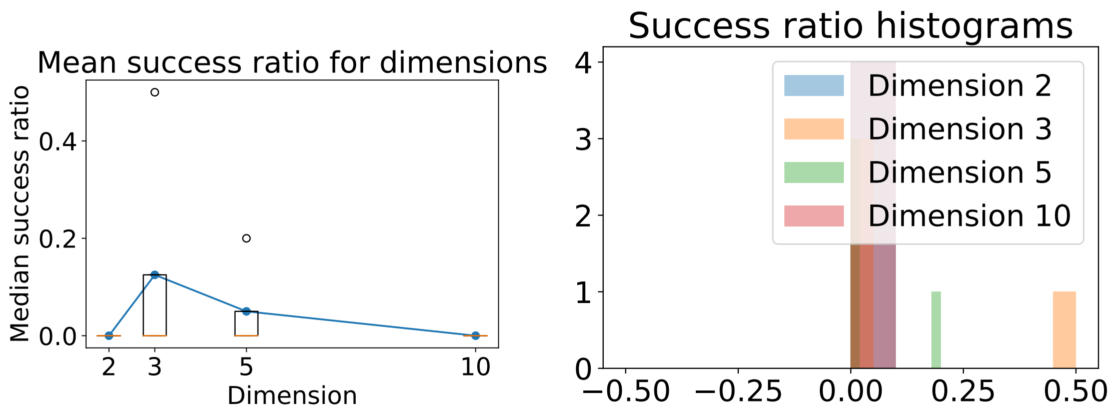


Figure 3: Success ratio for different dimensions. (left) as a line plot, (right) as a histogram

We visualize the descent curves under various versions of the losses. The result is shown in Figure 5. It can be seen that for dimension of 5 it is (subjectively) easier to find the global minimum than for dimension of 3. All the related metrics are shown in Figure 6. It can be seen that the resulting model (heatmap with dark background) is indeed sparse. During training, the cosine between gradients of two loss components does not reach extreme values (which would lead to "stuck" training), in this case. The agent metrics (rightmost chart) shows that the agent achieves near-optimal performance when using the decoder.

Overall, the experiments show feasibility of the proposed approach in a proof-of-concept settings. Using out method, it is possible to learn abstract representation, in principle. However, there are some engineering challenges that need to be overcome for this approach to be practical.

0.5 Conclusion

We presented a framework of learning abstract state representations in linear Reinforcement Learning cases. Even under these simplifications, the problem is NP-hard. Our heuristic algorithm is able to recover a sparse model of a linear environment.

0.5.1 Future directions

We would like to try the following options:

1. A closed-loop system tweaking the "temperature" based on the loss
2. SparseSep/Laplace prior for sparsity
3. Non-linear case
4. Harder environments, such as OpenSpiel. We extract causal models from them
5. Trying human studies: giving our extracted model to participants and seeing who performs better

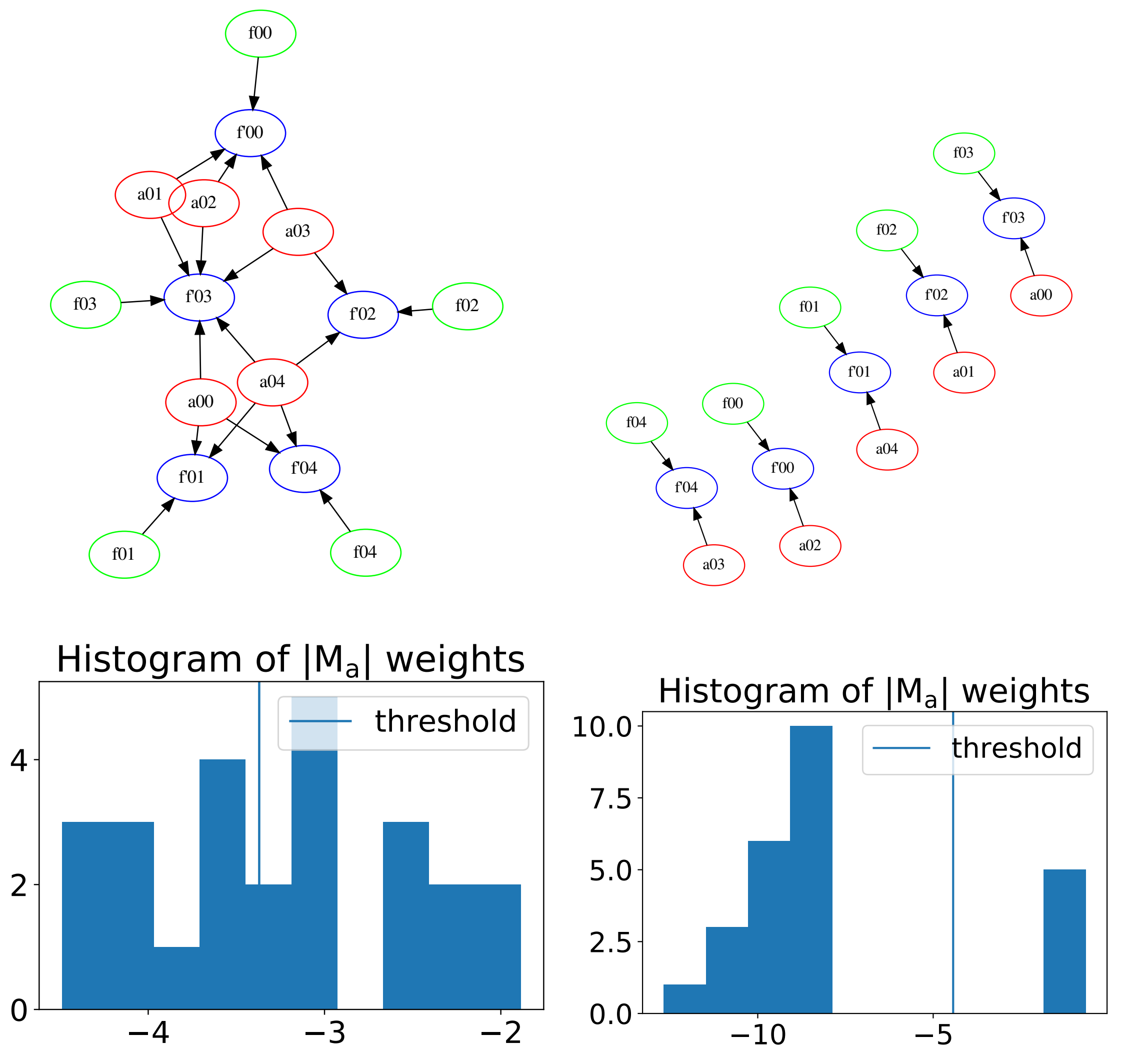


Figure 4: Environment models as graphs for $n = 5$. (top left) the original model learned from observations. Green are the feature vector components at the current iteration, red are the actions, and feature values at the next iteration are shown in blue; (top right) the model learned with our method on features; (bottom left) histogram of model weights for the observation model; (bottom right) histogram of model weights for the features model.

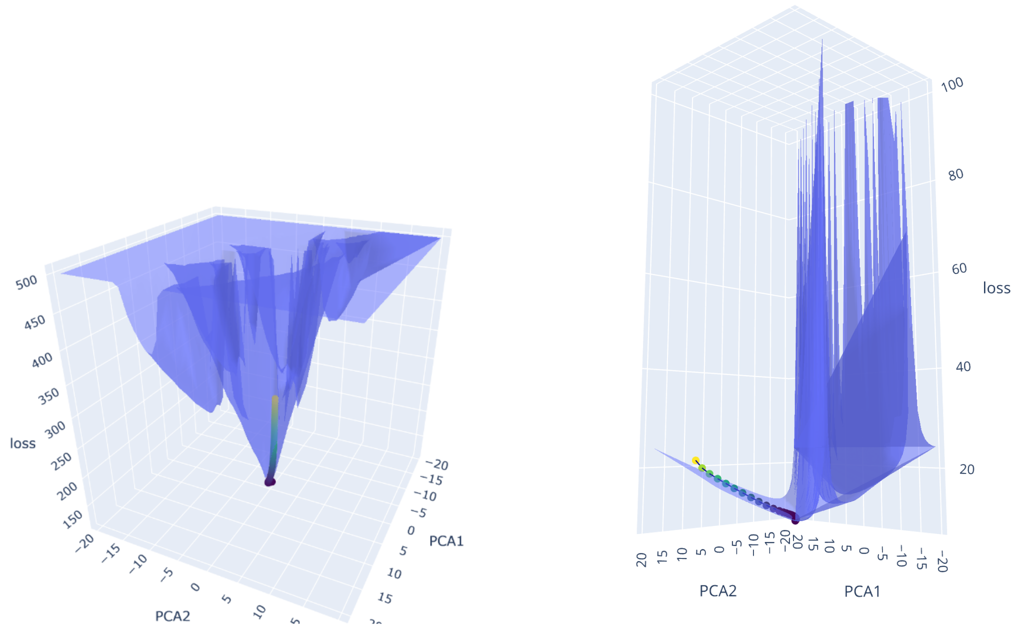


Figure 5: Loss landscape with descent curve for no-data objective for (left) $n = 3$, $\varepsilon = 1$, $p = 0.8$, (right) $n = 5$, $\varepsilon = 1$, $p = 1$

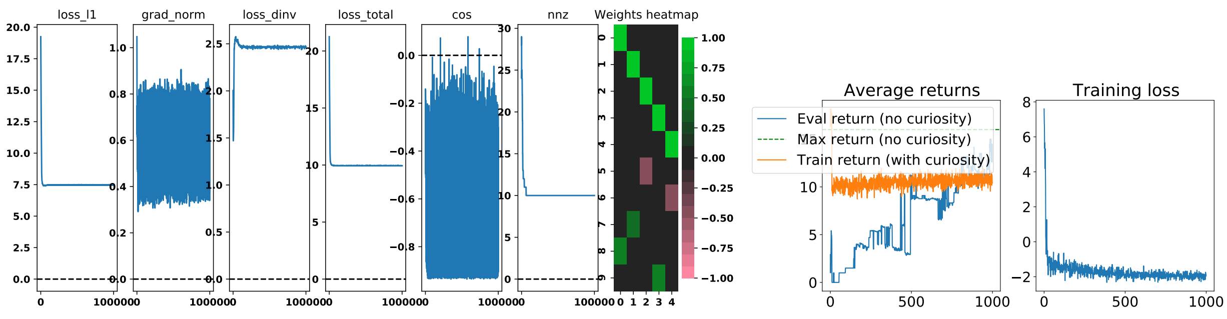


Figure 6: Details of convergence, $n = 5$

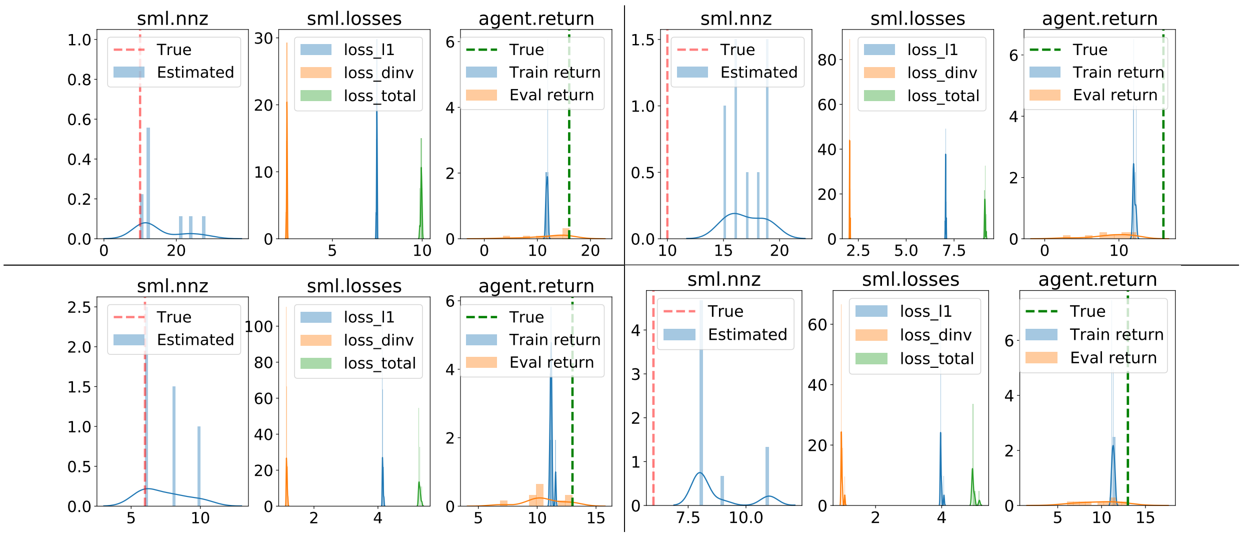


Figure 7: Histograms for dimensions (top left) dimension $n = 5$, encoder seed $s_e = 42$; (top right) $n = 5,\ s_e = 1$; (bottom left) $n = 3,\ s_e = 42$; (bottom right) $n = 3,\ s_e = 1$

6. Compare transfer learning and exploration capabilities of an agent regularized for sparse model vs. an agent that is not

0.5.2 Related work

This project is connected to studies on disentangled representations and to Causal Reinforcement Learning.

0.5.3 Acknowledgements

We thank Nevan Wichers for feedback on this draft, and El-Mahdi El-Mhamdi for introducing the author to the Consciousness Prior paper.

Bibliography

[1] Yoshua Bengio. “The consciousness prior”. In: *arXiv preprint arXiv:1709.08568* (2017).

[2] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. “Curiosity-driven exploration by self-supervised prediction”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2017, pp. 16–17.

[3] Sergei Volodin, Nevan Wichers, and Jeremy Nixon. “Resolving Spurious Correlations in Causal Models of Environments via Interventions”. In: *arXiv preprint arXiv:2002.05217* (2020).