École Polytechnique Fédérale de Lausanne

CauseOccam : Learning Interpretable Abstract Representations in Reinforcement Learning Environments via Model Sparsity

by Sergei Volodin

Master Thesis

Approved by the Examining Committee:

Prof. Wulfram Gerstner
Thesis Advisor

Dr. Adam Gleave
External Expert

Dr. Johanni Brea
Thesis Supervisor

EPFL IC/SV LCN
Laboratory of Computational Neuroscience
Bâtiment AAB
offices 135-141
CH-1015 Lausanne

Sergei Volodin
Chemin du Devin 47C
CH-1012 Lausanne

July 19, 2021

The strongest evidence that we can obtain for the validity of a proposed induction method, is that it yields results that are in accord with intuitive evaluations in many different kinds of situations in which we have strong intuitive ideas.

— Ray J. Solomonoff

Acknowledgments

I thank my parents and my sister for the support in my studies, and Switzerland and EPFL in particular for making it possible. Special thank you to the EPFL administration and the IC section administration for processing my visa in time for me to start my studies.

I would like to also thank Nevan Wichers, Le Nguen Hoang, Shalini Ananda, El-Mahdi El-Mhamdi, Louis Faucon, David Krueger, Ivan Vendrov, and Jeremy Nixon for the productive discussions in 2019-2020 on Causality and Machine Learning in general.

I would like to additionally thank Roman Pogodin for introducing me to EPFL and to computational neuroscience. I would like to thank Konrad Seifert, Mattias Berg, Ailin Parsa, Shalini Ananda, Romain Ensminger, Denis Drescher, Michael Pokorny for supporting me. I am grateful for the advice and mentoring from Arshavir Ter-Gabrielyan, Konstantin Lopuhin and Vladimir Vanovsky who supported my passion for science.

Special thank you to my supervisor Johanni Brea for guiding and supporting me during the thesis.

Lausanne, July 19, 2021 Sergei Volodin

Abstract

"I choose this restaurant because they have vegan sandwiches" could be a typical explanation we would expect from a human. However, current Reinforcement Learning (RL) techniques are not able to provide such explanations, when trained on raw pixels. RL for state-of-the-art benchmark environments are based on neural networks, which lack interpretability, because of the very factor that makes them so versatile – they have many parameters and intermediate representations. Enforcing safety guarantees is important when deploying RL agents in the real world, and guarantees require interpretability of the agent. Humans use short explanations that capture only the essential parts and often contain few causes to explain an effect. In our thesis, we address the problem of making RL agents understandable by humans. In addition to the safety concerns, the quest to mimic human-like reasoning is of general scientific interest, as it sheds light on the easy problem of consciousness.

The problem of providing interpretable and simple causal explanations of agent's behavior is connected to the problem of learning good state representations. If we lack such a representation, any reasoning algorithm's outputs would be useless for interpretability, since even the "referents" of the "thoughts" of such a system would be obscure to us.

One way to define simplicity of causal explanations via the sparsity of the Causal Model that describes the environment: the causal graph has the fewest edges connecting causes to their effects. For example, a model for choosing the restaurant that only depends on the cause "vegan" is simpler and more interpretable than a model that looks at each pixel of a photo of the menu of a restaurant, and possibly relies as well on spurious correlations, such the style of the menu.

In this thesis, we propose a framework CauseOccam for model-based Reinforcement Learning where the model is regularized for simplicity in terms of sparsity of the causal graph it corresponds to. The framework contains a learned mapping from observations to latent features, and a model predicting latent features at the next time-steps given ones from the current time-step. The latent features are regularized with the sparsity of the model, compared to a more traditional regularization on the features themselves, or via a hand-crafted interpretability loss. To achieve sparsity, we use discrete Bernoulli variables with gradient estimation, and to find the best parameters, we use the primal-dual constrained formulation to achieve a target model quality. The novelty of this work is in learning jointly a sparse causal graph and the representation taking pixels as the input on RL environments. We test this framework on benchmark environments with non-trivial high-dimensional dynamics and show that it can uncover the causal graph with the fewest edges in the latent space. We describe the implications of our work to developing priors enforcing interpretability.

Contents

Chapter 1

Introduction

Reinforcement learning is a general paradigm in Artificial Intelligence (AI) that considers two entities: an environment and an agent which interact with each other over multiple time-steps. The agent takes actions in the environment, and the environment gives the agent an observation representing the state of the environment, and a reward. The goal of the agent is to execute actions that lead to highest total reward during the interaction.

While RL environments in general can have arbitrary complexity, practically interesting environments usually have a certain low-dimensional structure in them. For example, in the popular CartPole environment, the observation (as image) has 720 000 variables (pixels), while the dynamics of the cart can be explained using only 4 variables with a simple Newtonian update rule. Atari games [16], having 100 800 variables in an observation, can be described using only 128 bytes (of the RAM state). The dynamics in the pixel space would be complex, with one pixel at the next time-step potentially depending on all of the pixels on the previous one. However, the dynamics in the latent space is much simpler. The real world has this property as well [53]: while a camera can capture many megapixels per second observing a drone, the equations describing the dynamics of the drone only have a few variables, and are much simpler.

RL agents could (and sometimes already do) solve real-world problems, such as steering the wheel of an autonomous car, controlling a plant, or determining which content to recommend to people on a social network. However, in many applications there exist highly undesirable outcomes, such as the car crashing due to a mistake in controlling it, or a recommendation engine showing polarizing or misleading content to people. To prevent such cases, we need to have an understanding of what makes an agent take a certain action, why it "thinks" it is optimal, and why it "thinks" it does not violate any sensitive constraints. Most capable agents are based on neural networks (NNs), and, therefore, it is hard to interpret them. Both the large number of parameters in the network, and many intermediate representations at each layer – the distinctive features that allow NNs to be so versatile in fitting different kinds of data, lead to problems with interpretability (or explainability) [84], because a priori there is no structure in the network: all variables depend on all variables of the input.

Even the state-of-the-art interpretability approaches [48] that look at individual neurons and try to determine their functions (such as a neuron responding to the images of dogs) do not allow to obtain a truly concise and reliable explanation, since, while some neurons are more important than others in predicting a particular outcome, all neurons contribute a non-zero amount to the outcome. Because of this, it becomes impossible to determine in advance what the prediction of a network would be in all of the input scenarios.

In order to make an agent interpretable (be able to answer questions of the form "why an action was taken given a particular observation"), the agent needs to grasp the high-level concepts and objects contained in the environment. This is a Representation Learning (or Abstraction Learning) task [14, 79, 31, 92, 108], with an objective of obtaining representations allowing for good explanations. To begin with, regardless of the task the agent is solving, it has to be able to explain in simple terms the effects of its actions on the environment.

While deep neural networks allow to fit various kinds of data and lack interpretability due to the large number of parameters, in contrast, traditional ("good-old-fashioned" or GOFA) Artificial Intelligence (AI) approaches are interpretable (explainable) and lack the versatility of neural networks: they are not applicable to all the tasks. Such GOFA approaches are characterized by discrete symbols that are the basis of a reasoning system, i.e. first-order logic. These symbols correspond to real-world concepts and objects, such as the coordinates of a robot on the floor. The drawback of such systems is that the mapping from real-world "messy" data (e.g. images from a camera) to discrete symbols (e.g. the coordinates) is hand-designed and thus requires effort to develop them. Unifying these two approaches yields the best of both worlds, with the versatility of deep AI in being able to fit almost any dataset and the explainability of traditional AI. To do so, we learn the representation of observations using a deep network. These representations are used by a system involving discrete components to make explainable predictions about the environment.

The quality of explanations given by such an agent would depend on the learned representation. Specifically, a representation of the observation in which every variable depends on every variable would be practically useless: an action potentially changes all of the variables, and, therefore, an explanation of action's effect would require to list all of them. Since the length of an explanation is crucial, the effect of an action should be describable by a change in only a few variables [107, 21, 49].

The current explainable RL agents can be classified into three following categories[1]. First, many projects learn sparse representations in terms of the feature vector sparsity: the predicted learned features associated with an observation are sparse at each time-step as a vector. Such a condition, however, is neither necessary nor sufficient for the *explanations* of the dynamics to be simple. Indeed, any sparse coding technique will yield sparse feature vectors (for example, by

[1]See chapter 6 for detailed references

assigning the first k most sparse vectors to all the k states of the MDP in case of a finite MDP), but a feature at the next time-step would depend on many features at the previous time-step. Indeed, without any assumption on the way the sparse features are obtained, two neighboring states can have drastically different representations. Thus, an action potentially changes all of the variables. This demonstrates that sparse feature vectors are not sufficient to have simple explanations. On the other hand, it is possible to have simple explanations without sparse feature vectors. Indeed, the state of a CartPole environment has all of the components being non-0 most of the time, and, yet, these components comply with a simple explainable kinematics equation. Thus, sparse feature vectors are not necessary for simple explanations.

The second approach for explainable RL [55] is to apply feature attribution to the convolutional networks representing policies, and the analysis of circuits of the network is performed. This approach gives a) the parts of the image most relevant to predict the action and b) the visualization of what each neuron in the network is activated with. While such an approach is extremely robust to the type of the underlying agent (the analysis is performed after the training and does not require any changes in the architecture), the drawback is that it should be performed for every input image: we do not know how a particular new image would affect the outputs, since each output action still depends on all of the pixels.

Finally, a line of work is investigating *causal* explanations of the agents' actions. This means that a causal graph corresponding to the environment is obtained, and an explanation for an action is then a simple path-finding from the action node to the reward node in this graph. The drawback of the current projects is that the graph is either given manually (by hand), or learned with significant manual work, such as designing the feature space. In this project, we extend this line of work by making the graph learnable end-to-end from the raw observations.

We would like to automatically uncover the simple structure in the underlying complex high-dimensional observations that the environment dynamics generates – uncover the succinct "laws of physics" [8] that the environment operates with. This would make the actions of an agent explainable. It makes the actions of the agent explainable in terms of the "laws of physics" that we learn. In terms of Model-Based RL [28, 23, 70] (an RL agent equipped with a model of the environment), we would like to learn a *simple* model of the environment. Specifically, we would like each variable in the model to depend on the fewest possible variables at the previous time-steps. This would make actions explainable in terms of the variables an action changes, and the explanation would be short. The model is the "good-old-fashioned" component, since it includes discrete elements: the variable's causes consist of a set of elements, rather than an array of numbers of weights for all possible causes. Mathematically, we require sparsity of the model's graph of dependencies – the predicted output for a variable depends only on a few variables at the previous time-step. If represented as a graph (with cause-variables at the current time-step as parents and effect-variables at the next time-step as children), this graph needs to have fewest possible edges.

To allow for such a simple *model*, we need to change the representation [73], from raw high-dimensional observations to the features, which the model predicts for the next time-step. Such features are obtained from a deep learning *decoder*[2], a function mapping observations to features. Given a decoder, the model can be obtained by simply fitting it in a supervised way, and only selecting the cause variables that lead to an improvement in the loss. Therefore, the decoder determines the simplicity of the model.

Thus, the decoder needs to be trained jointly with the model, and it is indirectly regularized by the model: the model is directly regularized for sparsity (in terms of the number of edges), and the decoder needs to output data that such model can predict. In our approach, the decoder is a neural network, and the dynamics model is also a neural network with a special mechanism to enforce the sparsity of the architecture.

If the the decoder is not additionally regularized for non-degeneracy, it is possible for the system to always predict constant features. This correspond to closing one's eyes and "predicting" successfully that everything that follows will be dark. The decoder needs to preserve the information that is relevant to the task the system is required to solve. Both in the real world, and in games, not all parts of the observations need to be predicted to achieve goals. For example, the color of an obstacle that a robot encounters is likely not important when it comes to avoiding it. However, this is task-specific, because, for example, a table is much less dangerous as an obstacle compared to water (a robot will be damaged if put into water, while an encounter with a table will likely only result in a loss of time when accomplishing a goal). Therefore, the model needs to predict task-specific features [24].

In this project, we use the two popular types of regularization: we either predict the reward (thus, only the aspects of observations relevant for predicting the reward are kept), or the whole observation (all the aspects of the observation are represented in the latent features). This is achieved by introducing a *reconstructor* – a network that predicts either the reward or the full observation given the latent features.

Our approach allows to obtain "laws of physics" of an environment, or, more specifically, a sparse *Causal Model* [99, 130] representing the environment's dynamics. Our work can be seen as an experimental confirmation of the Consciousness Prior proposal [13], specifically, the part that simple dynamics models learn to interpretable representations [9, 25, 50]. If an environment allows for a complete separation of some parts of its dynamics (for example, termination of the episode only depends on the health of the player, and another variable, such as the number of ammunition does not affect the health), out approach would capture it, because a graph with two components has less edges than a connected graph. This way, we *disentangle* the representation, as it consists of (recurrent) *independent* mechanisms. Such a model is very useful for explanations, because an effect of each action can be described in terms of changes in only a few variables (out of many). For example, for Cartpole with images as observations, our approach is expected to output the 4 features representing cart's and pole's position and velocity.

[2]We call it a decoder rather than an encoder because the observations are already *encoded* by the function mapping the simple environment states such as the cart's position and velocity into the high-dimensional observation

The main challenge when implementing our approach is combining all the requirements (sparsity of the causal model, learning the decoder, and the non-degeneracy of the decoder) into a single training procedure. Specifically, we compare several methods (simple aggregation of losses with coefficients, primal-dual method for optimization under constraints, and an adaptive scheme for choosing the loss coefficients) and employ several useful tricks (such as using a *relative* Mean Squared Error (MSE) loss to have an interpretable magnitude of the loss, and adding the sampled binary mask of selected cause features to the model that predicts the effects), and use different methods to enforce sparsity (from l_1-regularization to discrete variables with various methods to compute the gradients). We present an ablation study to show the effects of various tricks and techniques used.

We design a simple toy environment to illustrate learning of sparse causal models from high-dimensional data, and show that our approach successfully obtains the minimal causal graph on it. Surprisingly, the method finds a simpler (but correct) graph for the toy environment than the authors anticipated. Additionally, we illustrate the same positive behavior on a grid-world environment.

Contribution. This thesis presents, up to our knowledge, the first successful result on learning a sparse causal model jointly with the representation on a challenging high-dimensional problem with our approach CauseOccam. While the parts of the "recipe" used, such as discrete variables to learn a sparse causal graph can be found in existing literature, combining them into a working system that learns the representation and the sparse causal graph, along with the required modifications and the techniques, is a novel main result. The main difference with respect to existing methods of learning abstractions in RL is that our only prior is the sparsity (simplicity) of the learned model in the latent space. We do not add any additional priors or assumptions [45, 133] on the feature space, on the environment, or on the agent. We briefly mention the implications of our work on the Integrated Information Theory [117] of consciousness.

The result of this work can be used to improve significantly the explainability of Reinforcement Learning agents. A learned causal graph opens the possibility to drastically reduce the amount of data required to *ground* [77, 58] the various aspects of the observations with natural-language sentences. In this way, a dataset containing only two sentences "cart going left" and "cart going right" with two data points will likely result in a mapping of the sentences to the learned velocity variable. In contrast, if we ground raw images this way, the model is likely to overfit to the particular details that the image has (spurious correlations), and will not depend on the true cause. Sparse causal graph-based or natural language-based explanations [89, 38, 2] of agent's actions would allow for faster certification of agents for mission-critical applications.

Secondly, since the learned model is the simplest one, it is likely to be less prone to overfitting, and, therefore, more robust to distributional shift [121, 49, 102]. In this way, it would be easier to adjust the agent to a novel environment either without modifications (because the model relies more on a discrete set of concepts, than on the whole vector of features), or with re-training the decoder only (for example, if the shape of an obstacle has changed, only the decoder needs to be updated). For the latter part, the training should take less time, because the bulk of the problem is already solved – the model knows how to avoid obstacles.

Source code. The source code for the project is available at

github.com/sergeivolodin/causality-disentanglement-rl

Chapter 2

Background

In order to proceed, we need to introduce several important concepts: model-based reinforcement learning, causal models, methods for enforcing sparsity and methods for computing gradients for discrete variables.

We use $\mathbb{R}$ to denote real numbers, $[n]$ to denote the first n natural numbers ($[n] = \{1, 2, ..., n\} \subset \mathbb{N}$). For a vector with n dimensions $x = (x_1, ..., x_n) \in \mathbb{R}^n$, we use $\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$ the p-norm of x. The specific cases that we use are the 1-norm $\|x\|_1 = \sum\limits_{i=1}^n |x_i|$, and the 0-norm $\|x\|_0 = \sum\limits_{i=1}^n \mathbb{1}_{x_i \neq 0}$ – number of non-zero components in x ($\mathbb{1}_z = 1$ if z is true, and 0 otherwise). Given a distribution X, we write $x \sim X$ meaning that the random variable x has the distribution X (x "sampled from" X). We write $\mathbb{E}_{x \sim X} x$ meaning the expected value of x when sampled from a distribution X. For two vectors $x, y \in \mathbb{R}^n$, we define the element-wise product as $x \odot y = (x_1 y_1, ..., x_n y_n)$. For $p \in [0, 1]$, $Be(p)$ is the Bernoulli distribution: if $b \sim Be(p)$, $P(b = 1) = p$ and $P(b = 0) = 1 - p$. We write the probability density function of a continuous random variable $x \sim X$ as $f_X(x)$.

2.1 Reinforcement Learning

The standard Reinforcement Learning framework consists of an *environment* μ and an *agent* π (see Figure 2.1). The environment and the agent interact during an *episode* consisting of $T < \infty$ discrete time-steps. First, the agent receives an *observation* $o_1 \in O$ from the environment, where O is the *observation space*. Next, the agent executes an *action* $a_1 \in A$ where A is the *action space*. Next, the environment gives the agent a) a scalar reward $r_1 \in R$ where $R \subseteq \mathbb{R}$ is the *reward space*, and b) the next observation $o_2 \in O$. Next, either the environment terminates the interaction, in this case, $T = 2$, or the cycle repeats (the agent executes an action, etc). We introduce the termination variable d_t ("done") as $d_t = 0, t < T$ and $d_T = 1$.

A *history* (or an *episode*, or a *rollout*) of the interaction up to time-step τ consists of all the

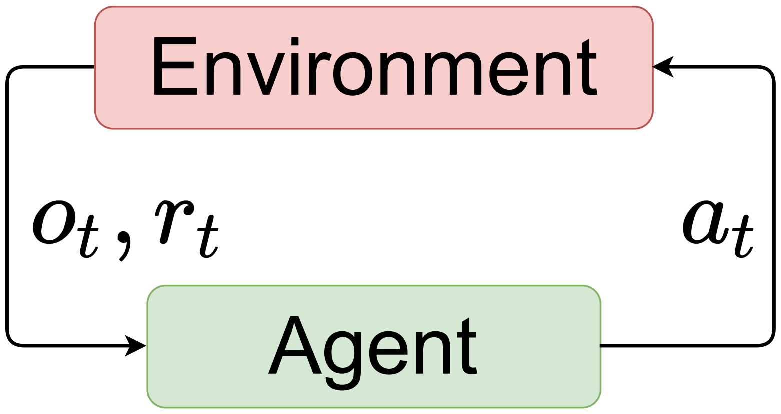


Figure 2.1: The reinforcement learning framework. An environment gives an observation o_t and a reward r_t at time-step t, and the agent responds with an action a_t. Then, the cycle repeats.

variables in the interaction in chronological order: $h_\tau = (o_1, a_1, r_1, o_2, ..., a_{\tau-1}, r_\tau, o_\tau)$.

For each time-step, the following holds: $(o_t, r_t) \sim \mu(a_{t-1}, h_{t-1})$ and $a_t \sim \pi(h_t)$. This means that the observation and the reward at step t is defined by the environment μ given the previous history, and the action at the step t is defined by the agent given the previous history.

We say that $h_T \sim (\mu \leftrightarrow \pi)$ if the history h_T was obtained from an interaction between the environment μ and the agent π.

The *value* is defined as the discounted sum of rewards: $V(h_T) = \sum\limits_{t=1}^{T} \gamma^t r_t$, where $\gamma \in (0, 1]$ is the *discount factor*. The goal of the agent is to find such π that $\mathbb{E}_{h_T \sim (\mu \leftrightarrow \pi)} V$ is maximized.

In our project, A is a finite set (the case of *discrete actions*). For example, it could be $A = \{\text{up}, \text{down}, \text{left}, \text{right}\}^1$. We assume that $O = \mathbb{R}^o$, where o is the dimension of each observation. For example, $o = 210 \times 160 \times 3$ for 210 rows, 160 columns and 3 RGB channels for an Atari [16] observation, or $o = 128$ for the 128 RAM components in the RAM version. The reward space R is bounded: $R = [r_{\min}, r_{\max}] \subset \mathbb{R}$.

Additionally, we consider the *Markov* class of environments: the next time-step only depends on the previous time-step in the history, and not on the ones before it[2]. For this class of environments,

$$(o_t, r_t) \sim \mu(a_t, o_{t-1}),\ a_t \sim \pi(o_t)$$

For example, for CartPole [16], o_t is obtained using the kinematics equation, and a_t is computed based on the velocities and the positions of the system.

[1]Our framework should support real-valued actions, they would be treated the same way as real-valued features without any difference

[2]A typical workaround to make some non-Markov environments to comply with this property is to stack a few observations together

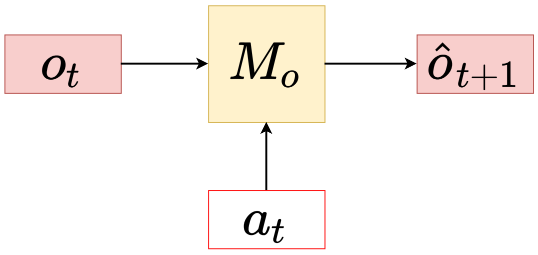


Figure 2.2: A single time-step model M_o predicts the next observations $\hat{o}_{t+1} \approx o_{t+1}$ given the current observations o_t and the actions a_t.

Special important classes are *deterministic environments* – ones where μ is a constant for every history (and not a distribution), and *deterministic agents* – ones where π is a constant given a history (and not a distribution).

In what follows, we consider Markov deterministic environments with deterministic agents[3].

2.2 Model-based reinforcement learning

Model-based reinforcement learning [28, 23, 70] agent additionally includes a *model of the environment* M_o[4] which is a function mapping the current observation o_t and the action taken a_t to the predicted next observation $\hat{o}_{t+1}$ (see Figure 2.2):

$$\hat{o}_{t+1} = M_o(o_t, a_t)$$

This function has the same signature as the environment. It is trained to fit the true dynamics: $\|o_{t+1} - \hat{o}_{t+1}\| \to \min$.

Since predicting the high-dimensional observations is computationally expensive, the observation model is often decomposed into three components: a decoder, a feature model, and a reconstructor:

$$M_o(o_t, a_t) = R(M_f(D(o_t), a_t))$$

Here, the decoder D maps an observation into a low-dimensional latent *feature space* $f_t = D(o_t)$, $f_t \in \mathbb{R}^f$ with f being the dimension of the feature space. The feature model M_f predicts next features from the current ones $f_{t+1} = M_f(f_t, a_t)$, and the reconstructor R maps the features back to the observation space: $\hat{o}_{t+1} = R(f_{t+1})$.

[3]To account for the stochasticity, the decoder and reconstructor should include an additional sampling step, and noise variables should be additionally injected as potential causes into the model

[4]The index o means that the model works in the observation space

This way, the computationally expensive reconstructor can be fit on two kinds of data: end-to-end to predict the next features $\|\hat{o}_{t+1} - o_{t+1}\| \to \min$ where $\hat{o}_{t+1} = M_o(o_t, a_t)$, and to predict the current ones $\|\hat{o}_t - o_t\| \to \min$, where $\hat{o}_t = R(D(o_t)) \equiv R(f_t)$. This way, the convergence is sped up.

Another technique prevents predicting high-dimensional observations altogether by replacing R with a value function predictor [109, 6], which is fit to predict the reward-to-go $R_t(h_T) = \sum\limits_{\tau=t}^{T} \gamma^{\tau-t+1} r_\tau$: $\|\hat{R}_t - R_t\| \to \min$ where $\hat{R}_t = R_{value}(f_t)$.

2.3 Causal modeling and deep causal modeling

Structural Causal Models [106, 130, 51] are a subclass of the usual models fitting data from some distribution, in which we explicitly define the set of variables a given variable could depend on. For example, if we have a dataset with a time series of the number of cancer patients, and one with a time series of the number of smokers, we could just fit an autoregressive model to predict both time series using both values. However, we could explicitly limit the effect of one time series on the other in our model, by simply not including it as the input to the model. In this way, we can discuss the differences in the losses when regressing the number of cancer patients given the time series with the smokers data: if this difference is not 0, it could be that this data suggests that smoking causes cancer.

More formally, suppose we have a set of variables f_i, $i \in [f] = (1, ..., f)$. These variables change with time and form time series $f_i^{t=1}, f_i^{t=2}, ..., f_i^{t=\tau}, ...$ for $i \in [f]$[5].

A *deterministic structural causal model (SCM) with 1-step dependencies* is a set of functions F_i, $i \in [f]$, which output the value of the feature i at the current time-step given a subset of features PA_i at the previous time-step: $f_i^{t=\tau} = F_i(f_j^{t=\tau-1}, j \in PA_i)$. In the following, we write $F_i(PA_i) \equiv F_i(f_j^{t=\tau-1}, j \in PA_i)$. For example, if $PA_i = [f]$ for all $i \in [f]$, each feature i depends on all features j at the previous time-step. This is the most general case – as all SCMs will fall into this category. However, we would like to find structure in the data: some features do not require *all* the features to predict them. For example, in a game the coordinate of the player is unlikely to depend on its ammunition. There could be a correlation (for example, having collected some inventory allows access to another room), but the ammunition is not required to predict the position, as we could just use the previous position, the action taken, and whether or not there are any obstacles around instead.

Granger causality [10] is defined in the way as in the example above: a feature j can be excluded from PA_i if the best function (measured by the Mean Squared Error loss) that predicts f_i from $PA_i \setminus \{j\}$ has the same loss as the best function that predicts f_i from PA_i.

[5]We write time indices below for the whole vector $f_\tau = (f_1^{t=\tau}, ..., f_t^{t=\tau})$, and we write them above, if we additionally specify the index of the feature $f_i^{t=\tau}$

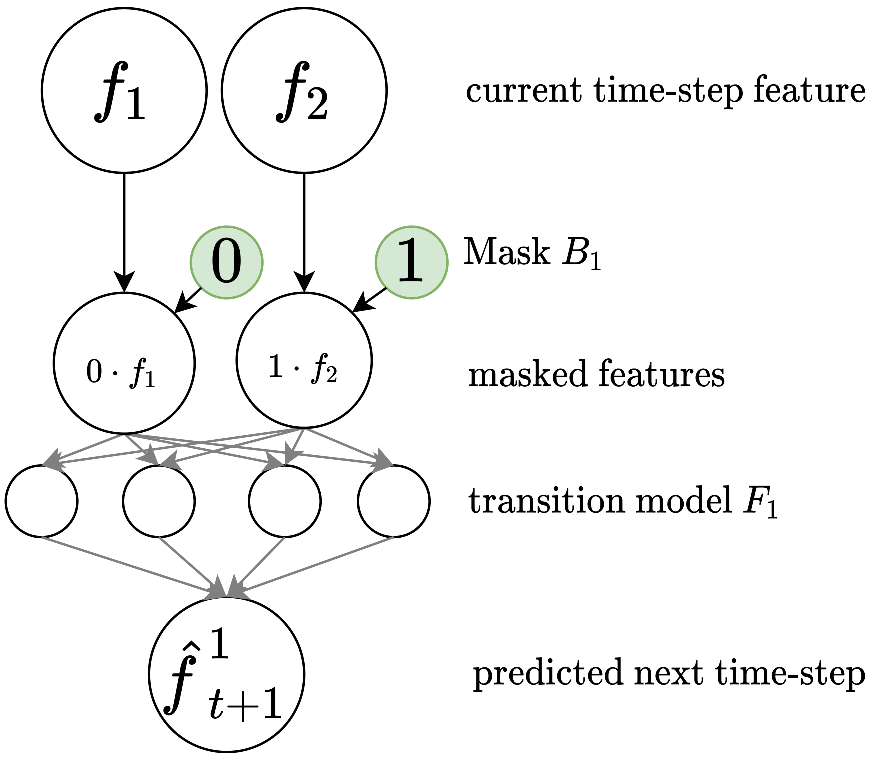


Figure 2.3: The model $F_1(f^t \odot B_1)$ only uses features determined by the mask $B_1 = (0, 1)$ to estimate the value of $\hat{f}_i^{t+1}$

More generally, we set $\delta(PA_i, j) = \inf_{F_i} \mathbb{E}_t \|f_i - F_i(PA_i \setminus \{j\})\| - \inf_{F_i} \mathbb{E}_t \|f_i - F_i(PA_i)\|$. This is the difference in the losses without and with a feature j when predicting the feature i, given the current set PA_i. Given a threshold $\varepsilon > 0$, if $\delta(PA_i, j) < \varepsilon$, we could exclude j from PA_i, and this would only worsen the prediction quality by ε.

We note that while Granger causality does not imply "proper" causality [99], and only imply correlation, we are applying it in the context of Reinforcement Learning. In RL, actions truly cause the observations to change [105], and, therefore, Granger causality can be seen as a way to uncover the structure of the environment.

2.3.1 Neural Granger Causality

In order to find F_i in practice, we model them with neural networks [115, 72, 97] with input size f. Each function is $F_i \colon \mathbb{R}^f \to \mathbb{R}$. In order to account for the sparsity of the inputs PA_i, we pre-multiply the input to each function F_i by a binary mask $B_i \in \{0, 1\}^f$ (see Figure 2.3):

$$\hat{f}_i^{t+1} = F_i(f^t \odot B_i)$$

Next, we fit functions F_i using least squares regression:

$$\mathcal{L} = \frac{1}{T} \sum_{i=1}^{f} \sum_{t=1}^{T} \left(f_i^{t+1} - \hat{f}_i^{t+1} \right)^2 \to \min$$

The gradient with respect to the parameters of F_i is computed in standard way, and we present methods to enforce sparsity of B_i in the next section.

2.4 Enforcing sparsity in deep architectures

Suppose that we are fitting a loss function which depends on a binary variable, like in the previous section:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} (y_t - F(x_t \odot B)) \to \min_{F,B}$$

Here, $y_t \in \mathbb{R}^y$ is the target, $x_t \in \mathbb{R}^x$ is the input data, and $B \in \{0,1\}^x$ is a binary mask.

One (very inefficient) method to solve this problem is to a) go over all possible masks B b) for each of them, find the best model F c) select the most sparse $\|B\|_0 \to \min$, such that $\mathcal{L}(B) \leq \varepsilon$ for some threshold ε. The complexity of such search is $\mathcal{O}(2^x)$, which is not feasible even for $x = 20$. In our setups, even for the simplest environment, $x > 20$. In general, this problem would be NP-hard, as it is the feature selection problem.

We can use gradient descent on B if we relax the problem. There are several methods for it.

1. The obvious solution is to allow $B_i \in [0,1]$ instead of $B_i \in \{0,1\}$ and regularize the model with $\|B\|_1$ [114]. This would enforce sparsity on B. However, since we are using a multi-layer network, this solution would not work: in practice, the coefficients in B are never exactly 0, and the subsequent layers can multiply the incoming value by a large constant. Therefore, while B is sparse, the network could actually still be using all of the features.

2. In the rest of the methods [97, 1, 78, 71, 131], we sample (element-wise) $B \sim Be(P)$ from the Bernoulli distribution, where $P \in [0,1]^x$ is a matrix of probabilities. The computation of the gradient of the loss with respect to the parameters of the distribution is described in the next section

2.5 Methods to compute gradients of discrete variables

This section describes standard methods to compute gradients with respect to a binary variable. We have the same setup as in the previous section.

We sample $B_i \sim Be(P_i(\theta_i))$, where θ_i parameterizes the probability P_i, and, therefore, the distribution $Be(P_i(\theta_i))$.

We want to compute a gradient of $\frac{\partial \mathcal{L}}{\partial \theta_i}$ for one of the elements P_i The list of methods follows [15, 129, 88].

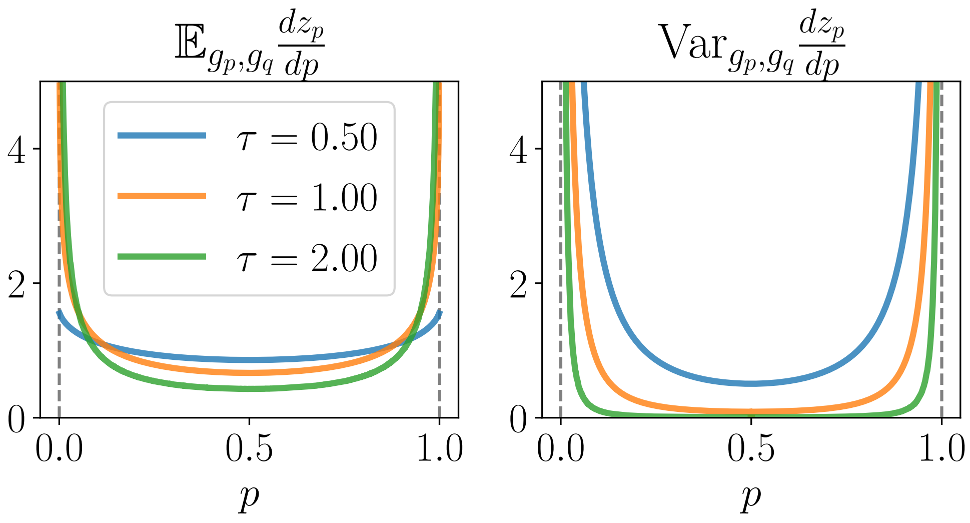


Figure 2.4: Gradient for Gumbel-Softmax for different parameters $\tau \in \{1/2, 1, 2\}$. We obtain it by considering the Gumbel-Softmax estimate $z_p = 1/(1 + \exp((\log(1-p) - \log p + g_q - g_p)/\tau)$ and noticing that $\delta = g_q - g_p$ has a Logistic distribution. We compute the expected gradient for $\tau = 1$ and $\tau = 1/2$ via symbolic integration with Wolfram Mathematica, and using numerical sampling for $\tau = 2$ (the integral as a function of p does not seem to have a symbolic value for $\tau = 2$). Both the expectation and the variance of $\frac{dz_p}{dp}$ have asymptotes at $p = 0, 1$. Specifically, $\lim_{p \to 0,1} \mathbb{E} dz_p/dp = +\infty$, and $\lim_{p \to 0,1} \mathbb{E}\,(dz_p/dp)^2 = +\infty$: the gradient has a high expected value and is also noisy. The gradient is well-defined at $p = 1/2$: its expectation and variance have a limit at $p \to 1/2$, even though the symbolic expression contains $p = 1/2$ in the denominator. For example, for $\tau = 1$, $\mathbb{E} dz_p/dp = (2 - 4p + \log \frac{p}{1-p})/(2p-1)^3$.

1. REINFORCE gradient. Consider $\mathbb{E} L'_{\theta_i} = \frac{\partial}{\partial \theta_i} \mathbb{E}_{B_i \sim Be(p_i),\, i \in [x]} \mathcal{L}(b_1, ..., b_x)$. We split the expectation into two: $\mathbb{E} L'_{\theta_i} = \frac{\partial}{\partial \theta_i} \mathbb{E}_{B_i \sim Be(p_i)} \mathbb{E}_{B_i \sim Be(p_j),\, j \in [x],\, j \neq i} \mathcal{L}(b_1, ..., b_x)$. Next, we write $\mathbb{E}_{B_i}$ explicitly with $\mathbb{E}_{b_{\setminus i}}$ meaning $\mathbb{E}_{B_i \sim Be(p_j),\, j \in [x],\, j \neq i}$:

$$L'_{\theta_i} = \frac{\partial}{\partial \theta_i} \left[p_i \mathbb{E}_{b_{\setminus i}} \mathcal{L}\big|_{B_i = 1} + (1 - p_i) \mathbb{E}_{b_{\setminus i}} \mathcal{L}\big|_{B_i = 0} \right]$$

Now, only p_i depends on θ_i, and the expected loss $\mathbb{E}_{b_{\setminus i}}\mathcal{L}$ does not. Therefore,

$$L'_{\theta_i} = \frac{\partial p_i}{\partial \theta_i} \cdot \left[\mathbb{E}_{b_{\setminus i}}\left(L\big|_{B_i=1} - L\big|_{B_i=0}\right)\right]$$

The second multiplier on the right-hand side in the equation above is the difference between the expected losses with the variable B_i on and off, while the expectation is taken with respect to all the other binary variables.

Now, if we have a batch with sampled masks B_i, in which the variable B_i takes the value of "1" N_i^+ times, and the value of "0" N_i^- times, we can set up a sample mean estimate:

$$L'_{\theta_i} \approx \frac{\partial p_i}{\partial \theta_i} \cdot \left[\frac{1}{N_i^+} \sum_{t=1,B_i^t=1}^{T} L(b_1, ..., b_x) - \frac{1}{N_i^-} \sum_{t=1,B_i^t=0}^{T} L(b_1, ..., b_x)\right]$$

Note that we have to have at least one sample with $B_i^t = 0$ and one with $B_i^t = 1$ for this expression to be valid. Otherwise, the gradient is undefined, and the update is not performed for this component of the gradient.

The drawback of such an approach is high variance, pertinent to REINFORCE estimators. The advantage is its simplicity of tuning (it has no hyperparameters), and the fact that the gradient estimate is unbiased (with increases in batch sizes T, the gradient has a limit of its true value, by the Central Limit Theorem).[6]

In this approach, $\theta_i \equiv p_i$. In order to keep the values of p_i in $[0,1]$, we simply project to $[0,1]$ after each gradient step: if $p'_i > 1$, we set $p_i = 1$, and the same for $p'_i < 0$.

2. Straight-through estimator. We set $\theta_i \equiv p_i$ and simply ignore the fact that there was sampling:

$$\frac{\partial \mathbb{E}\mathcal{L}}{\partial p_i} = \frac{\partial \mathbb{E}\mathcal{L}}{\partial B_i}$$

   Here, we simply compute the gradient with respect to the variable B_i, as if it were a real-valued one.

3. Gumbel-Softmax [63]. In the forward pass (when computing the value of the loss), we sample from the Bernoulli distribution. In the Backward pass, we compute the gradient as follows. Consider the probabilities p_i and $q_i = 1 - p_i$ corresponding to sampling either 1 or 0 (two categories).

   We define $l_p = \log p_i + g_p$ and $l_q = \log q_i + g_q$. Here, g_p and g_q are independent samples from the *Gumbel* distribution: a continuous distribution with a density function $f_{g_p}(x) =$

[6]This is called a REINFORCE gradient because it uses the same technique – log-likelihood trick, as the REINFORCE algorithm for Reinforcement Learning. Indeed, consider the log-likelihood trick: $\partial/\partial\theta_i \mathbb{E}_{B_i}\mathcal{L} = \partial/\partial\theta_i \int p(B_i)\mathcal{L}(B_i)dB_i = \int \partial p(B_i)/\partial\theta_i L(B_i)dB_i = \int \partial \log p(B_i)/\partial\theta_i p(B_i)L(B_i)dB_i = \mathbb{E}_{B_i}\partial \log p(B_i)/\partial\theta_i L(B_i)$. For the Bernoulli distribution, $\partial \log p(B_i)/\partial\theta_i = \frac{\partial p_i}{\partial \theta_i}\frac{1}{p_i}$ if $B_i = 1$ or $-\frac{\partial p_i}{\partial \theta_i}\frac{1}{1-p_i}$ otherwise, and the denominators cancel out with the probability mass function.

$\exp(-(x+\exp(-x)))$. It has the following property: $\arg\max[l_p, l_q] \overset{d}{=} Be(p)$: the distributions are equal.

Now, we replace the $\arg\max$ with soft $\arg\max$:

$$B_i := z_p = \frac{\exp(l_p/\tau)}{\exp(l_p/\tau) + \exp(l_q/\tau)}$$

When $\tau \to 0$, $z_p \overset{d}{\to} Be(p_i)$ (soft $\arg\max$ becomes $\arg\max$). If $\tau \to \infty$, $z_p \to 1/2$.

For $\tau \in (0, \infty)$, We set the (biased) gradient to be:

$$\frac{\partial \mathbb{E}\mathcal{L}}{\partial p_i} := \frac{\partial \mathbb{E}\mathcal{L}}{\partial B_i}\frac{\partial z_p}{\partial p_i}.$$

The advantage is that the gradient is defined even for a single sample. Compared to the previous approach, there is no additional noise due to the fact that a sample estimate is used.

The Figure 2.4 shows the expected gradient of the estimator [5] with respect to the probability. The asymptotes at $p \to 0$ and $p \to 1$ mean that the gradient becomes bigger, as the entropy of the distribution decreases. Therefore, the probabilities never reach these terminal values, as the noise "pushes" them away from these values.

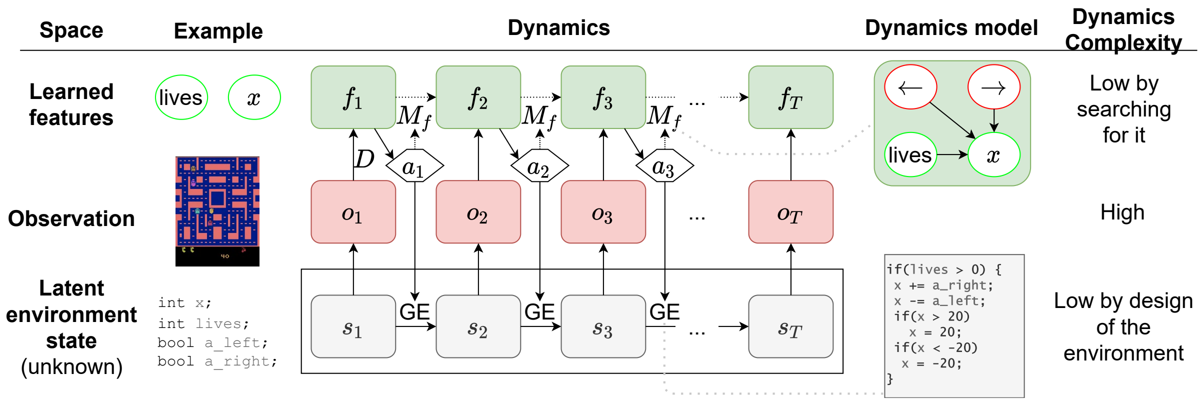
In our project, it is important that the sampling probabilities actually approach the value of $p = 1$ (see subsection 3.2.5). Therefore, we would need to modify Gumbel-Softmax with an additional thresholding mechanism to update the model parameters.

In our setup, we use REINFORCE implementation as the simplest one.

The Gumbel-Softmax trick (without modifications) would result in duplicated variables, as the probabilities never reach $p = 1$, see subsection 3.2.5.

Chapter 3

Design of CauseOccam



```
int x;
int lives;
bool a_left;
bool a_right;
```

```
if(lives > 0) {
 x += a_right;
 x -= a_left;
 if(x > 20)
   x = 20;
 if(x < -20)
  x = -20;
}
```

Figure 3.1: Step representations used in the project. At the bottom, the environment's Game Engine (GE) generates the next state s_{t+1} given the previous state s_t and the action a_t. Next, the states are encoded into high-dimensional observations $o_t = E(s_t)$. The true state dynamics s_t, as well as the encoder E are unknown. The proposed methods finds a feature space $f_t = D(o_t)$ by applying the decoder D, in which the dynamics becomes simple. The dynamics in the space of features is modeled by the function M_f. On the right, the parts of the dynamics model are shown for the MsPacman [16] game.

We have an agent π interacting with the environment μ. We start from the model-based Reinforcement learning setup with a decoder and a reconstructor. We would like to obtain the feature space in which the model M_f becomes most simple in terms of a metric K.

We consider a model of the environment $\hat{f}_{t+1} = M_f(f_t, a_t)$ in the sense of section 2.2. Additionally, we would like some components of f_t and a_t to be ignored when predicting some component of the whole feature vector $f_{i,t+1}$. We view the model $M_f = [M_f^1, ..., M_f^f]$ (split into scalar functions predicting each feature) as a causal model in the sense of section 2.3. The set of variables of this causal model contains all features, all actions, and additionally the variable r for

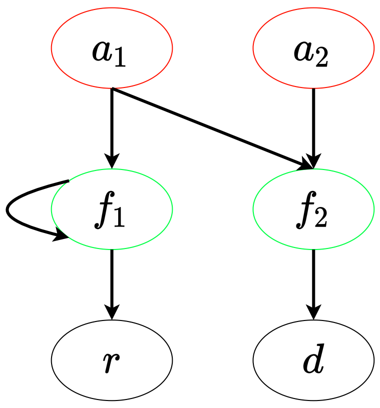


Figure 3.2: An example of a Structural Causal Model we are looking for. Here, the number of features $f = 2$ and the number of actions $a = 2$. The equations can be written as $\hat{f}^1_{t+1} = F_{f_1}(f^1_t, a^1_t)$ (next time-step f_1 depends on itself at the previous time-step, and on whether the action a_1 was taken), $\hat{f}^2_{t+1} = F_{f_2}(a^1_t, a^2_t)$, $\hat{r}_{t+1} = F_r(f^1_t)$, $\hat{d}_{t+1} = F_d(f^2_t)$. The masks corresponding to this graph equal $B_{f_1} = (1, 0, 1, 0)$ (in order of f_1, f_2, a_1, a_2), $B_{f_2} = (0, 0, 1, 1)$, $B_r = (1, 0, 0, 0)$, and $B_d = (0, 1, 0, 0)$

the reward and the variable d for the "done" (see section 2.1 for definitions):

$$\begin{aligned}
X &= \{f_1, ..., f_f, a_1, ..., a_a, r, d\},\\
X_{in} &= \{f_1, ..., f_f, a_1, ..., a_a\}\\
X_{out} &= \{f_1, ..., f_f, r, d\}
\end{aligned}$$

The variables $X_{in} \subset X$ are the input (ones used for prediction), and the variables $X_{out} \subset X$ are the output (ones being predicted). We would like to predict the next features, as well as the reward r and the done d at time-step $t + 1$ given features and actions at the time-step t. The prediction of the model for a variable x consists of pre-multiplying the input features (variables f_t and a_t, written as a vector in what follows as $fa_t = [f_t, a_t] \in \mathbb{R}^{f+a}$) with the binary mask B_x, to select the input features, and then applying the SCM function F_x: $x_{t+1} = F_x(fa_t \odot B_x)$, $x \in X_{out}$.

$$\begin{aligned}
\hat{f}^1_{t+1} &= M^1_f(f_t, a_t) &&= F_{f_1}(\; [f_t, a_t] \; \odot B_{f_1} \;)\\
& \dots\\
\hat{f}^f_{t+1} &= M^f_f(f_t, a_t) &&= F_{f_f}(\; [f_t, a_t] \; \odot B_{f_f} \;)\\
\hat{r}_{t+1} &= M^r_f(f_t, a_t) &&= F_r(\; [f_t, a_t] \; \odot B_r \;)\\
\hat{d}_{t+1} &= M^d_f(f_t, a_t) &&= F_d(\; [f_t, a_t] \; \odot B_d \;)
\end{aligned}$$

The Figure 3.2 shows an example SCM.

3.1 Measure of complexity

3.1.1 Sparse features are neither necessary nor sufficient for interpretability

First, we give a detailed example of why feature sparsity does not imply interpretability, and interpretability does not imply feature sparsity.

Example 3.1.1. *Consider the $2D$ grid-world environment from Figure 3.3 with $H, W > 2$. It has HW states. Consider binary features $f_i \in \{0, 1\}$. Given the feature dimensionality f, we select the first HW most sparse vectors in $\{0, 1\}^f$ to map to the states (this is possible in case if $2^f \geq HW$).*

In case if $f = HW$, we have maximal feature sparsity: each feature vector has at most 1 component. Feature vectors become 1-hot vectors coding for the state numbers. However, the model lacks interpretability because it does not capture the structure of the environment – in truth, it has only 2 independent state components x and y, and actions affect only the corresponding coordinates.

One might argue that the feature dimensionality is too high in this case, and this is the cause of the lack of interpretability.

In case if $HW \approx f + f(f-1)/2$, we use all vectors with 1 component and all vectors with 2 components. However, this model is also not interpretable – it also does not capture the two-dimensional dynamics.

If we continue decreasing f further, some features would have many non-zero elements, and sparsity would be lost.

This shows that feature sparsity in a reasonable example is not sufficient for interpretability.

Now, a coding with two features with $f_1 = x$ and $f_2 = y$ solves the problem – each action corresponds to one feature, and features have a simple dynamics. However, this coding is not sparse – in fact, the values of the features are never 0. This shows that feature sparsity is not necessary for interpretability.

To be fair, sparse features do help interpretability in case if an observation can contain a limited number of objects, one at a time. For example, an observation either contains an enemy, or the health bar. Our approach can be seen as a generalization: these two variables then can be replaced with one and modeled with our framework. We note that our approach also has flaws (see the failure modes below), and the question of defining human "simplicity prior" mathematically in all cases remains open.

In addition, we must clarify the following point. In what follows, we will induce sparsity on the feature vectors, but in a very specific way. First, we consider sparse feature vectors (with some components zeroed out) as an *input to the model predicting the same features*, i.e. a dynamics model. The model takes only certain features as the input. Importantly, the sparsity mask is different for predicting different features, and is not reducible to the sparsity of the feature vector as a whole. In fact, for all of the environments that we use, the feature vector is not sparse as a whole.

Most known interpretable models (natural language inter-sentence dependencies uncovered with attention, causal models used for medical data, physics equations only capturing the neighbors of a particle to describe its dynamic, sparse linear regression) only have few components: they are sparse. However, sparsity alone does not necessarily mean interpretability. In what follows, we will show that some sparse models are not interpretable, and give methods to overcome that.

3.1.2 Sparse fan-in models sometimes are sufficient for interpretability, and usually are necessary

In general, simplicity of a model can be described by Kolmogorov complexity (see Appendix A). Since this quantity is uncomputable, we set the proxy for the Kolmogorov complexity for our class of models as the total number of non-zero components in the mask, or the number of edges in the Structural Causal Model:

$$\tilde{K}(B) = \sum_{x \in X_{in}} \|B_x\|_0$$

Note that we only count the complexity of the mask, and do *not* count the complexity of the functions F_x, or of the decoder D. This stems from two reasons. The first reason is purely practical – there does not exist a measure of complexity for neural networks F_x or D that could be easily optimized for[1]. The second reason is there by design. Namely, This issue is somewhat alleviated by the fact that neural network of fixed architecture can only implement functions of bounded Kolmogorov complexity: it only has a finite number of arithmetic operations. In this way, the total complexity of all learnable parameters can be described as $K(M_o) = K(D) + K(B) + K(M_f)$ where $K(D)$ and $K(M_f)$ are bounded.

However, it is important to set the class of functions that represent the decoder D and the model F_x properly. Otherwise, the following situation can occur.

Proposition 3.1.1. *(Too complex decoder) Consider an arbitrary deterministic environment μ and a deterministic policy π. With a suitable decoder, it is possible to obtain $\tilde{K}(B) = 3$ (minimal value) and $L_{rec} = L_{fit} = 0$*

[1]One candidate is the norm of the Jacobian as a measure of complexity, however, this norm would be lower for a $\sin(x)$ function than for a linear function x for some arguments, because the derivative is lower. Another option is to use the same technique with disabling features [86] to limit the fan-in of the neurons in the network to induce structural sparsity. However, this would result in an even more non-convex and hard-to-optimize problem than this thesis presents. This approach is out of the scope of this thesis.

Proof. The main idea is to map a multidimensional space into a single-dimensional one. There are several ways to do that

- Consider the space of all possible states $S = \mathbb{R}^s$ of the true environment.

  It is possible to bijectively map a unit interval $(0, 1)$ to a unit square $(0, 1)^2$. The mapping is highly non-smooth, like the Peano curve. Using this technique, we map $\mathbb{R}^s$ to $\mathbb{R}^{s-1}$. Repeating the process, we eventually map $\mathbb{R}^s$ to $\mathbb{R}^1$. Now, there is only a single feature describing the state.

- For discrete state spaces, an even simpler approach is possible: since the number of states is finite, it is possible to represent each state using its number. This trick can also work for continuous states with discretization.

Now, $f = 1$ (only 1 feature is required). Given features f_t, we can compute f_{t+1} by first reconstructing the observation (here the bijectivity matters), and then applying the "Game Engine" (the dynamics of the model in its internal state space):

$$f_{t+1} = D(E(GE(E^{-1}(D^{-1}(f_t))), a_t))$$

Here, the model M_f depends on a single feature (because we only have a single feature), the reconstruction is perfect, and prediction is perfect.

The complexity $K(B) = 3$: the only feature is used to predict itself, as well as the attributes d and r.

While the complexity of the binary mask B is small, the complete model is extremely complex: the functions D and D^{-1} compute the non-smooth bijection mapping a unit interval to a unit square. □

While in practice, it would be hard to find such a function with gradient descent, as it is highly non-smooth, there is another example which shows that the complexity of D and M_f should not be exceedingly high:

Example 3.1.2. *Consider a grid-world with 2 components of the player (see Figure 3.3): x (horizontal) and y (vertical) taking discrete values: $x \in [H]$ and $y \in [W]$. The actions $-x$ and $+x$ affect the horizontal coordinate x. The actions $-y$ and $+y$ affect the vertical coordinate y. We can represent this environment by 2 features, one for each of the coordinates.*

Consider the case of H and W being small (for example, $H = W = 2$). In this case, we can design a transition model using only a single feature f_1 which takes the following values:

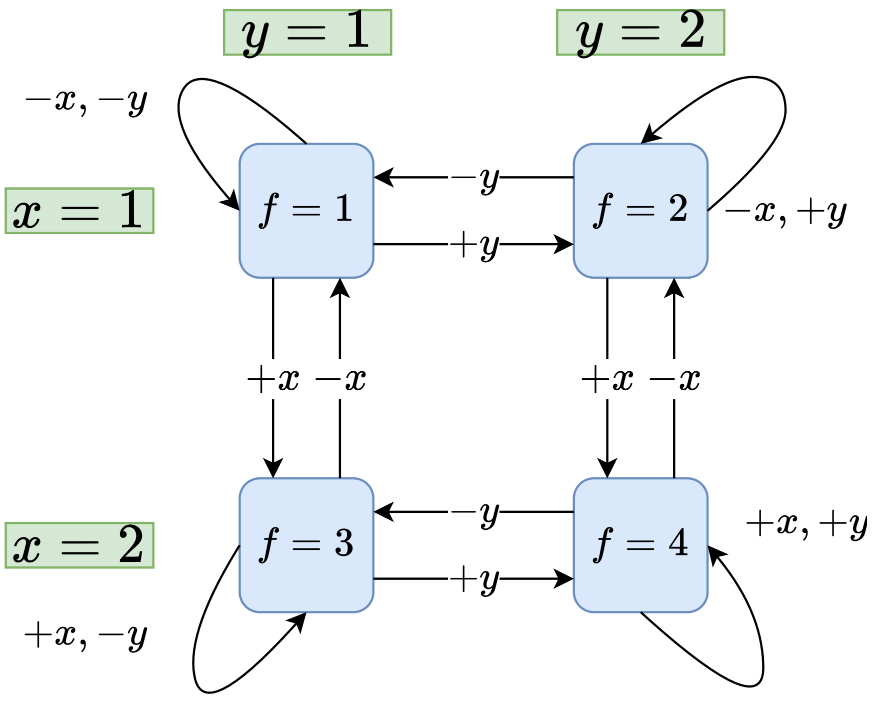


Figure 3.3: 2D grid-world with $H \times W$ cells: H rows and C columns (the image shows $H = W = 2$). Actions $\pm x$ and $\pm y$ change the values of the coordinates. The single feature $f \in \{1, 2, 3, 4\}$ bijectively maps to the state space. This example extends naturally to H rows and R columns with $x \in [H]$ and $y \in [W]$, and the same set of actions $\pm x$, $\pm y$.

f_1	x	y
1	1	1
2	1	2
3	2	1
4	2	2

For example, if $x = 1$ and $y = 2$, we set $f_1 = 2$.

Now, the transition model is shown in Figure 3.3. It is quite complex: each feature variable can potentially result in 3 different states! However, a neural network with enough layers could fit this dynamics.

A much simpler model appears if we map each coordinate to a separate feature: $f_1 = x$ and $f_2 = y$. In this case, the dynamics for each coordinate is much more simple: $f_1^{t+1} = \max(1, \min(2, f_1^t + \mathbb{1}_{a=+x} - \mathbb{1}_{a=-x}))$

Thus, too high model complexity will result in less features used, but the interpretability will be lost.

Luckily, this pathological behavior disappears as $H, W \to \infty$. While the complexity of the *coordinate-wise approach does* not *become more complex, and is always described as* $f_1^{t+1} = \max(1, \min(H, f_1^t + \mathbb{1}_{a=+x} - \mathbb{1}_{a=-x}))$*, the single-feature approach's complexity increases. We show it for the case of "left-to-right" assignment of feature values. For the single-feature model, we set* $f = W \cdot (x-1) + y$*, and we can reconstruct* $y = f \mod W$ *and* $x = (f-y)/W + 1$*. Now, implementing the* $z \mod n$ *function for* $z \in [kn]$ *with a neural network of constant complexity and varying* n *and* z *is not possible (unless using a periodic activation functions [112])*[2].

Therefore, for $H, W \gg 1$*, the model would prefer using two features instead of* 1.

"Reusable" decoders and models. This argument shows a natural setting in which it is not required to count the complexity of the decoder D, and of the SCM functions F_x. Humans do not re-learn the visual cortex when a new game is presented. Instead, the same architecture is reused [12, 122, 61, 34, 49, 90], and only the novel high-level behavior is learned. Additionally, humans do not learn the rules of every game from scratch (for example, learning again that there are x and y coordinates in every shooter game). Instead, the existing abstractions (such as $WASD$ keys used to control the player) are re-used. With this in mind, a natural extension of this work is to re-use the decoder and parts of the model (see section 7 for the concrete future direction proposal). If we reuse some parts of the decoder, or of the model, their complexity (in terms of the shortest program describing them) is greatly reduced, when the "bank" of existing abstractions is given, and only an "index" in this "bank" needs to be specified, instead of learning the complete model from scratch.

Linear models. A reasonable questions might occur: would it be possible to use the simplest possible class of models to define functions F – linear models, and "transfer" all of the non-linear complexity into the decoder? This would be extremely beneficial, as environments would be extremely interpretable in this space. The following example shows that even for one of the simplest RL environments, there is no linear feature space that preserves the dynamics and maps bijectively into observations.

Example 3.1.3. *Consider the class of linear models:* $M_f(f_t, a_t) = M_f f_t + M_a a_t + b$*, where* M_f *is a matrix of size* $f \times f$*,* M_a *is a matrix of size* $f \times a$*, and* b *is the bias vector of size* f*. We are interested in all environments that can be characterized by linear latent dynamics. In other words, what are all the environments, for which there exist a decoder* D *and a reconstructor* R*, such that there exist a bijection between the set of observations and the set of feature vectors, and the feature dynamics is linear?*

We show that this class is very small, even with a non-linear decoder. Indeed, consider the Line-3 environment (navigating on a line with 3 states) – see Figure 3.4. It does not fit into our definition.

Indeed, first, the bias term in the linear model can be "shifted" into the actions M_a*. Suppose*

[2]For example, a ReLU network is a piecewise-linear function, and the number of "switches" is determined by the number of neurons. Note that the $\mod$ function requires at least as many switches as the number of "wraps" of the argument z w.r.t. the n. Therefore, a ReLU network with a fixed architecture can only implement some but not all functions $z \mod n$.

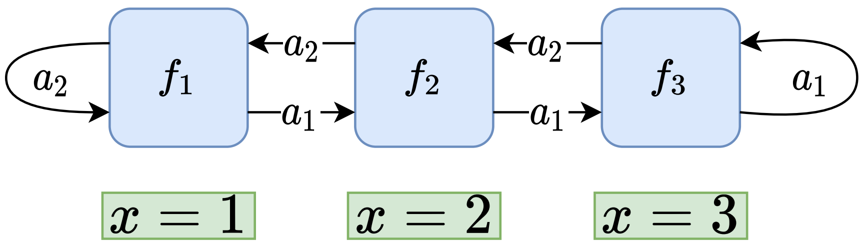


Figure 3.4: Line environment with 3 states. Action a_1 corresponds to moving right, and a_2 corresponds to moving left. Going left from the left-most state does not change the state, and same for the right-most state. The variable x increases when going right and takes integer values starting from 1. Each state with $x = i$ has an associated feature vector f_i

that the decoder gives some features for the states f_1, f_2, f_3 (does not matter how, it's only important that the features are fixed). Multiplying $M_a a_i = A_i$, we get two vectors A_1 and A_2.

Then, $f_1 = M_f f_1 + A_2$ (going left from f_1 gives f_1), and $f_3 = M_f f_3 + A_1$ (going right from f_3 gives f_3). Now consider what happens in f_2. If we go left, we get f_1: $M_f f_2 + A_2 = f_1$. If we go right, we get f_3: $M_f f_2 + A_1 = f_3$. Subtracting these two gives $f_3 - f_1 = A_1 - A_2$. On the other hand, the "stuck at the wall" equations give us $f_3 - f_1 = M_f f_3 + A_1 - M_f f_1 - A_2 = M_f(f_3 - f_1) + A_1 - A_2$. So, equating these two, we get $f_3 - f_1 = M_f(f_3 - f_1) + A_1 - A_2 = A_1 - A_2$. This means that $M_f f_3 = M_f f_1$.

Now, consider the "stuck at the right wall" $f_3 = M_f f_3 + A_1$, and going right from f_1: $f_2 = M_f f_1 + A_1$. But $M_f f_1 = M_f f_3$, which means that $f_2 = M_f f_3 + A_1$. Note the same RHS as for the "stuck at the right wall". Therefore, $f_2 = f_3$, we cannot distinguish between f_2 and f_3, a contradiction with the representation being non-degenerate.

This contradiction shows that for this environment, features are either degenerate *(a single feature vector corresponds to two different observations), or the model should be non-linear.*

Note that a simple non-linear model is applicable for this environment: $f_1 = x$, $f_1^{t+1} = \min(3, \max(1, f_1^t + \mathbb{1}_{a=a_1} - \mathbb{1}_{a=a_2}))$.

These examples show that it is important to set the complexity of the model's neural networks correctly. A too simple model would never fit the environment, and a too complex model can give non-interpretable features.

A practical way to choose the measure of complexity is to first fit a model without sparsity constraints, select one with minimal complexity still fitting the data, and then run CauseOccam.

Examples. We discuss how we expect this approach to work in a wide range of environments, and compare it to another interpretability technique where each action is associated [116] in a change in a single feature. We call the latter the "single-feature action influence framework".

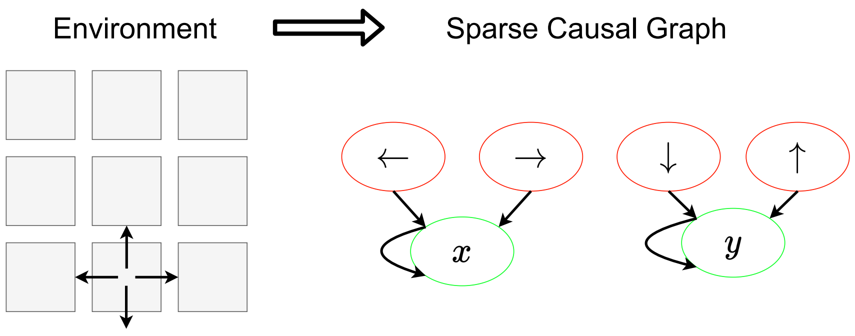


Figure 3.5: The 2D grid-world environment and the SCM that could be extracted from it. The causal model contains 2 feature components, one for the horizontal coordinate (x) and one for the vertical coordinate y. The actions left and right affect x, and the actions down and up affect y.

The single-feature action influence framework fails in case if an action influences two features, and it does not put constraints on features that are not directly affected by agent's actions (such as movements of objects in the environment). Our approach can be seen as an extension of the single-feature action influence framework in a sense that in case if an environment allows for such a parameterization, our approach would find it as well. Indeed, a single action influencing a single feature exactly corresponds to a minimal number of edges from actions to features.

We also note that the Non-Linear Independent Component Analysis [36, 60] would not yield the desired representations in cases where there is significant dependence between features. Indeed, it ignores the influence of actions on the time series, and, therefore, it would not account for the number of edges between action-nodes and feature-nodes.

Example 3.1.4. *(2D grid-world navigation task, Figure 3.3 and 3.5) Consider an environment giving 2D images of size $H \times W$ as an observation, and taking 4 actions (left (decrease horizontal coordinate), right (increase horizontal coordinate), up (increase vertical coordinate), down (decrease vertical coordinate)). While the actions correspond to coordinates, the coordinate variables are latent (hidden) from the agent, as it receives observations instead. In the latent space, the dynamics is very simple: each action only influences one coordinate, and the transition equation is simple as well.*

If we apply our approach to this environment, it would uncover this latent representation, guided only by the simplicity of the resulting transition model. The model in this space has 6 edges: each of the two features depends on itself from the previous iteration (2 edges), and each action influencing one feature component (4 edges).

For this environment, the simpler approach to associate an action with a feature component works as well. However, the loss would not be 0, as an action does not always influence the state (at the boundaries an action does not do anything).

Example 3.1.5. *(MountainCar [16]) We consider two possible parameterizations. One uses Cartesian coordinates x and y (and their velocities), and the other uses the linear position of the car on*

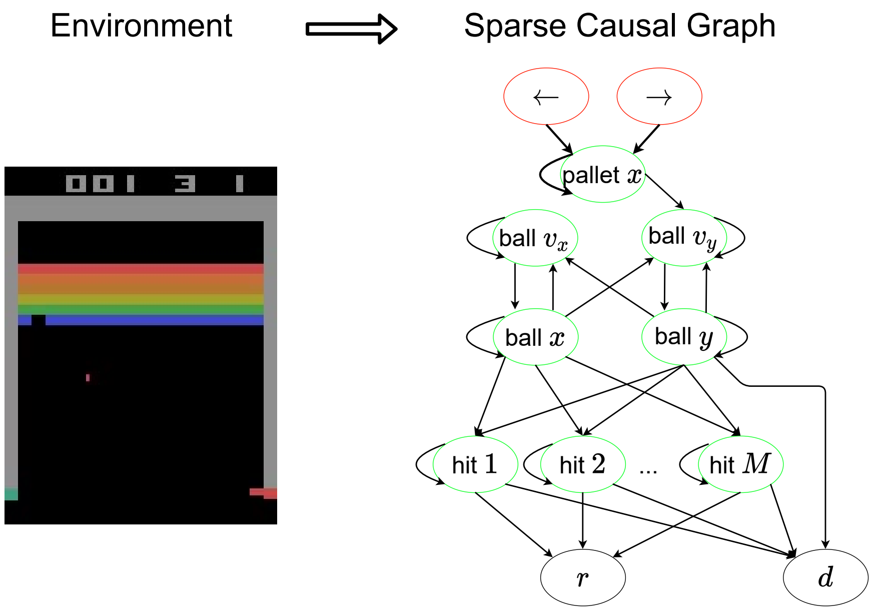


Figure 3.6: A causal graph for the Atari Breakout environment. The actions left and right move the pallet (red line below on the left image). The ball's (a single red dot at the dark space on the left image) coordinates update via the velocity variables: the horizontal coordinate $x_{t+1} = x_t + v_x\delta t$ and the vertical coordinate $y_{t+1} = y_t + v_y\delta t$. If the pallet hits the ball, the ball is reflected, hence v_y depends on x, y and the pallet position: $v_y^{t+1} = -v_y^t$ in case if $x = x_{pallet}$ and $y = 0$. The ball can be reflected from the borders of the environment as well, hence v_x depends on x and y. If the ball hits one of the M targets (colored items at the top on the left image), the target disappears, and a reward is given: $hit_i = 1$ if $x = x_i$ and $y = y_i$. If the ball is below the pallet, the episode terminates, hence d depends on y of the ball.

the track (and its velocity).

The Cartesian coordinates result in an interpretable model with physically realistic equations. However, in this parameterization, an action influences both of the coordinates, as the car moves both in x and y directions.

The linear coordinate is compliant with the single-feature framework, but the dynamics equation becomes more complex (the gravity depends on the angle of the track).

Example 3.1.6. *(CartPole [16] with images as observations.) Consider the CartPole environment from OpenAI Gym. The latent state has 4 components (the positions and the velocities of the cart and the pole). The model in the latent space has 8 edges: features depend on themselves (4 edges), velocity influencing the coordinate (2 edges) and action influence (2 edges changing the coordinate slightly).*

Example 3.1.7. *(Atari [16] Breakout, Figure 3.6) The objects can be grouped into the pallet that is controllable by the player (1 horizontal coordinate), two coordinates for the ball (and its velocity), and one presence feature for each of the individual pieces that need to be destroyed.*

Note that while the single-feature influence model disentangles the coordinate of the pallet from the rest of the features, it does not put any constraints on the representation for the ball or for the pieces to collect: they can be represented in any possible way.

Example 3.1.8. *(MsPacman [16]) The number of lives is a separate feature, the positions of the ghost, of food pallets, of the cherry, of the player are features as well.*

Note that the single-feature action influence approach again would obtain a good representation for the Pacman (because its coordinates are affected by the actions directly). It does not enforce any constraints to obtain a good representation for the ghosts, the food, and the cherry.

Example 3.1.9. *(MuJoCo Humanoid [16]) The representation contains the joint position and velocities, as defined in the MuJoCo environment file. The dynamics equation corresponds to the physics equations from the simulator.*

Since the state components are already corresponding to interpretable quantities (such as coordinates and velocities), applying our approach would not yield novel results for this environment.

If, in contrast, an image is given instead of the state components, our approach is expected to find the representation space of coordinates and their velocities again.

3.2 Optimization problem for CauseOccam

Overall, we would like to obtain a sparse model without loss of fit quality. Our search space consists of continuous parameters for D, R, F_x, and discrete parameters B_x for $x \in X_{in}$ (see above). The figure 3.7 shows the trainable parameters and the losses described in this section.

3.2.1 Relative mean-squared error loss

In order to know how well the model predicts the next state, we use a *relative* mean-squared error loss, instead of the standard "absolute" one. Specifically, if the model predicts $\hat{y}_t$, for $t \in [T]$, and the true values are y_t, $t \in [T]$, the relative MSE error is defined as:

$$\rho(y_t, \hat{y}_t | t \in [T]) = \frac{1}{T} \sum_{t=1}^{T} \left(\frac{y_t - \hat{y}_t}{\sigma[y]} \right)^2 \tag{3.1}$$

Here, $\sigma[y]$ is defined as standard deviation of y in case if it is non-zero, and it is defined as 1 otherwise.

Example 3.2.1. *Suppose that $T = 2$, $y_\cdot = \varepsilon \cdot (0, 1)$ and $\hat{y}_\cdot = \varepsilon \cdot (1, 0)$. We compute $\sigma[y] = 0.5\varepsilon$, and $\rho(y_t, \hat{y}_t) = 4$. This value does not depend on the scale ε of y and $\hat{y}$. In contrast, the traditional mean-*

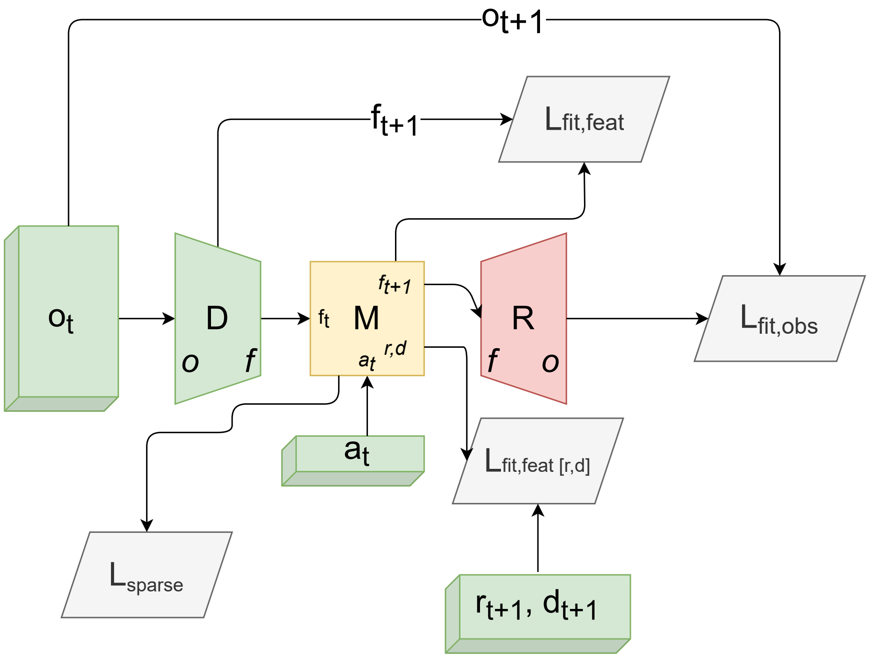


Figure 3.7: The diagram of losses for CauseOccam. The batch of observations o_t (left) is fed into the *decoder* D, which outputs *features* f_t. The next features f_{t+1} are predicted using the model M given the actions a_t. The model is regularized with a sparsity constraint L_{sparse}, and fit using losses $L_{fit,feat}$ in the feature space (predicting the next features) and in the observation space (predicting the next observation). The model also predicts the additional reward (r) and done (d) features. The reconstruction loss is not shown on this figure. See this section for more details on the losses.

squared error would yield $\rho_{mse}(y_t, \hat{y}_t) = \frac{1}{T}\sum\limits_{t=1}^{T}(y_t - \hat{y}_t)^2 = 0.5\varepsilon^2(1^2 + 1^2) = \varepsilon^2$*, which depends on the scale of the features.*

It is important that the quality metric does not depend on the magnitude of the inputs, as the magnitude of the given by the decoder features can be arbitrary high or low.

Example 3.2.2. *Suppose that* $y \sim Y$ *(i.i.d. samples from a distribution* Y*), and* $\hat{y} = c$ *is a constant predictor. Then, the best value of the relative MSE loss* ρ *is* 1 *with* $c = \mathbb{E}Y$*. Since neural networks can fit a constant, we expect that* $\rho < 1$ *in our experiments. However, at the beginning of the training, this value can be larger than* 1.

Example 3.2.3. *(Discrete-valued variables) Suppose that* $y \sim Uniform([n])$ *– a discrete uniform variable taking values in* $\{1, ..., n\}$*.* $\mathbb{E}y = n/2$ *and Var*$y = (n^2 - 1)/12$*. In case if* $\rho(y, \hat{y}) < \varepsilon < \frac{1}{2n}$*, we compute using the Chebyshev's inequality:* $p_{bad} = P(y \neq \hat{y}) = P(|y - \hat{y}| > 1) \leq \mathbb{E}(y - \hat{y})^2/1^2 = O(\rho n^2/12)$*. If we want* $p_{bad} = 1/n$*, we obtain a requirement* $\rho < 12/n^3$*. For a discrete variable with* $n = 20$ *to obtain* $p_{bad} < 1/20$*, we will have to have* $\rho \approx 1.5 \times 10^{-3}$.

However, if we specify $|y - \hat{y}| \sim Uniform([0, a])$*, we have Var*$|y - \hat{y}| = (a^2)/12$*, and the condition* $P(|y - \hat{y}| > 1) \leq 1 - 1/a$*. Setting* $P(|y - \hat{y}| > 1) < 1/n$ *gives* $a < n/(n-1)$*. Since* $\rho = O(a^2/n^2)$*, we* *arrive at a looser condition $\rho < 1/n^2$. For example, for a discrete variable with $n = 20$, to obtain $p_{bad} < 1/20$, we will have to have $\rho \approx 3 \times 10^{-2}$.*

Later, we use this estimate for discrete variables to set the threshold values for the relative MSE loss in the constrained optimization formulation of our problem.

Example 3.2.4. *Suppose that $y = 0$ (a constant), and $\hat{y} \sim N(0, \sigma^2)$ – the normal distribution. In this case, $\sigma[y] = 1$ (because the true variance is 0), and $\rho(y, \hat{y}) = \mathbb{E}_{\hat{y} \sim N(0,\sigma^2)}(0 - \hat{y})^2 = \sigma^2$.*

Example 3.2.5. *Suppose that $y \sim N(0, \sigma^2)$ (normal distribution), and $\hat{y} = 0$ – a constant. In this case, $\sigma[y] = \sigma$ and $\rho(y, \hat{y}) = 1/\sigma^2 \mathbb{E}_{y \sim N(0,\sigma^2)}(y - 0)^2 = \sigma^2/\sigma^2 = 1$.*

3.2.2 Non-degenerate decoder

The decoder needs to be invertible (or, at least, "partially" invertible, in a sense that all relevant information is kept). Note that, technically, it is possible for the decoder to always output 0 for all observations. In this case, there exist a simple constant model that "predicts" the next state perfectly (if prediction for d and r is disabled). However, such a model is useless, as it does not describe any aspect of the dynamics. One way to solve this issue is to introduce a reconstructor predicting observations from features. See section 2.2 for further discussion.

The reconstruction loss is defined in terms of the expected distance between the reconstructed and true observations:

$$L_{rec} = \mathbb{E}_{h \sim (\mu \leftrightarrow \pi)} \rho(R(D(o_t)), o_t | t \in [T]) \tag{3.2}$$

3.2.3 Model fit quality in observation and feature spaces

We also would like the model to predict the future time-steps correctly. This can be achieved in one of two ways.

1. Feature-space loss. We penalize the models' features if they deviate from the features predicted by the decoder at the next time-step:

   $$L_{fit,feat} = \mathbb{E}_{h \sim (\mu \leftrightarrow \pi) | T > 1} \rho(M_f(D(o_t), a_t), D(o_{t+1}) | t \in [T-1]) \tag{3.3}$$

   Here, we only consider episodes that last more than 1 time-step, because otherwise there is no "next" time-step to predict.[3]

[3]We do not model the "initial state" distribution, because it does not affect model simplicity in terms of $K(B)$, and it does not affect the SCM functions F_x, and in this project we are only interested in learning the causal graph.

2. Observation space loss. We penalize the model if the *reconstructed observation* given the predicted features matches the true next observation:

$$L_{fit,obs} = \mathbb{E}_{h\sim(\mu\leftrightarrow\pi)|T>1}\rho(R(M_f(D(o_t), a_t)), o_{t+1}|t \in [T-1]) \tag{3.4}$$

Next, we compare the two approaches. First, if $L_{fit,feat} = 0$, it follows that $L_{fit,obs} = 0$. Indeed, the reconstruction loss L_{rec} ensures that the next-step features result in correct observations. And $L_{fit,feat} = 0$ ensures that the features are predicted correctly.

In contrast, $L_{fit,obs} = 0$ does *not* imply that $L_{fit,feat} = 0$. Indeed, the model can add the 1-st feature to all features: $\hat{f}_i = f_i + f_1$ and indicate that by adding a large enough number to another feature: $\hat{f}_s = f_s + 10\sigma[f_s]$. The decoder first determines if $10\sigma[f_s]$ was added to f_s, and then either subtracts f_1 or keeps it. This way, $L_{fit,feat} = 0$, but the model does not predict the correct features. While this is fine in case if we are using the learned model for planning, it would result in a wrong Structural Causal Model. This means that in practice, we need to use $L_{fit,feat}$ or otherwise, the learned model function does actually predict the next features, and, thus, the causal graph might be incorrect.

Example 3.2.6. *(Incorrect causal graph when only using $L_{fit,obs}$) Consider the Game Engine from Figure 3.3. The features predicted by the decoder are x and y (not the ones in the figure). The model is the following function: $\hat{x}_{t+1} = x_t + 10 \cdot \mathbb{1}_{a=+y} - 10 \cdot \mathbb{1}_{a=-y}$, $\hat{y}_{t+1} = y_t + 10 \cdot \mathbb{1}_{a=+x} - 10 \cdot \mathbb{1}_{a=-x}$. It is possible to obtain $L_{fit,obs} = 0$, because we can reconstruct both the previous state, and the action from $\hat{x}, \hat{y}$. Indeed, note that $|x_t|, |y_t| < 2$. This means that, if $\hat{x} > 10$, an action $+y$ was taken. The rest of the actions are reconstructed in the same way. Next, knowing the actions we obtain x_t and y_t. Next, the reconstructor performs "the role of the model", computes the (true) next x_{t+1} and y_{t+1}, and then performs the reconstruction. In case if both of input arguments to the reconstructor are $|x_t| < 10$ and $|y_t| < 10$, the model does not perform the time-step prediction.*

The causal graph in this case is incorrect: x depends on $+y$ and $-y$, and y depends on $+x$ and $-x$.

Therefore, we have to use $L_{fit,feat}$. However, in practice, even when using the relative loss, $L_{fit,feat} < \varepsilon$ for a small $\varepsilon \in (0,1)$ does *not* mean that the prediction quality is good.

Example 3.2.7. *Consider the Game Engine from Figure 3.3. It has 2 latent variables: x and y. Now, suppose that the features (not on the figure) are $f_1 = x$ and $f_2 = \alpha y + (1-\alpha)x$ for $\alpha \ll 1$. Suppose that $L_{fit,feat} < \varepsilon$. Does it mean that the model "preserves" the information both about x and y? No.*

Indeed, for the variable x, $L_{fit,feat} < \varepsilon$ implies that $\rho(x, \hat{x}) < \varepsilon$, as required. However, for y, it does not. Indeed, we can show that it is possible to obtain any ε bound on the loss given $\hat{y} = \mathbb{E}y$ (the constant prediction giving $\rho(y, \hat{y}) = 1$. Assume that x is predicted perfectly. Next, consider the relative loss

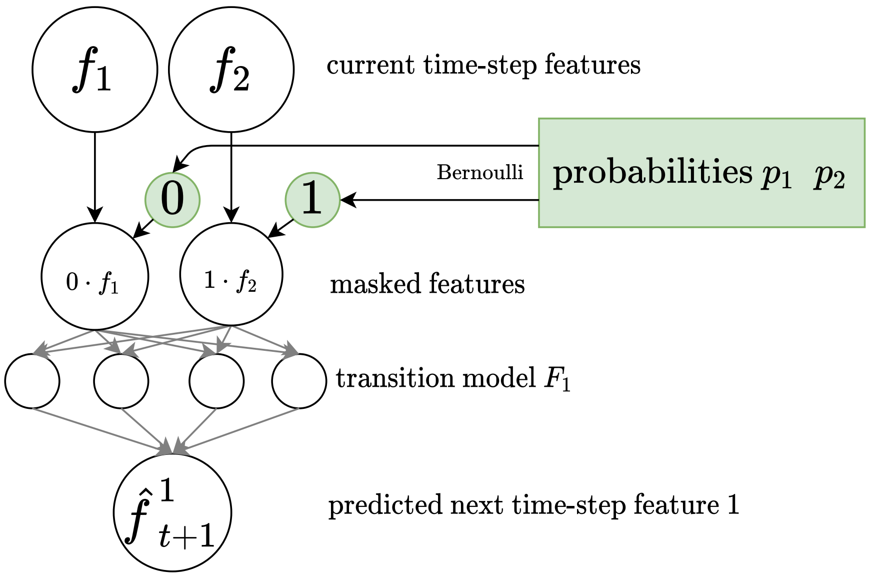


Figure 3.8: The model F_1 predicts the value of the first feature f_t^1 at the next time-step, given the current features f_{t+1}. The model only uses features determined by the binary mask $B_1 \sim Be(P_1)$.

$$\rho(\alpha y + (1-\alpha)x, \alpha\hat{y} + (1-\alpha)\hat{x}) = \frac{\mathbb{E}\left(\alpha(y-\hat{y}) + (1-\alpha)(x-\hat{x})\right)^2}{\sigma[\alpha y + (1-\alpha)x]^2}$$

.

We assume that x and y are independent under π and have the same variance of c. Then we compute $\sigma[\alpha y + (1-\alpha)x]^2 = \sqrt{\alpha^2 + (1-\alpha)^2 c}$.

Next, since $\hat{x} = x$, $\rho = 1/\sigma^2 \alpha^2 \mathbb{E}(y-\hat{y})^2 = \frac{\alpha^2}{\alpha^2+(1-\alpha)^2}$. Since this equation has a limit of 0 when $\alpha \to 0$, and since the decoder can give features with an arbitrarily small α, we can make $L_{fit,feat}$ arbitrarily small, while the model only predicts the value of x, but not the value of y.

Note that, without the relative loss, it is not even required for the model to predict the value of x. Indeed, to make the loss arbitrary small, it is sufficient for the decoder to reduce the magnitude of the features, and for the reconstructor to increase the input weights.

Therefore, we have to use both losses, one in the feature space, and one in the observation space.

3.2.4 Sparsity loss

In section 2.5 we outline techniques for parameterizing $B \sim Be(P)$ and then computing the gradient $\partial B/\partial P$, so it becomes possible to find the values of B using gradient descent.

Now we consider the original objective:

$$K(B) = \sum_{x \in X_{in}} \|B_x\|_0$$

Since B is sampled from a distribution P (see Figure 3.8), we compute the expected value:

$$\mathbb{E}_{B \sim Be(P)} K(B) = \sum_{x \in X_{in}} \|P_x\|_1$$

.

This happens because for each component, $\mathbb{E}_{b \sim Be(p)} b = p$. Thus, we naturally arrive at the l_1-relaxation of our discrete problem [15]

We define this loss as:

$$L_{sparse} = \mathbb{E} K(B) = \sum_{x \in X_{in}} \|P_x\|_1$$

3.2.5 Ranges of edge presence probabilities and exploration

In case if for some of the probabilities $P_{xx'} = 0$, or $P_{xx'} = 1$, that feature is *never* selected and is never active, or is *always* selected, and the model never explores cases where this feature is not enabled.

However, we would like to have the case $P_{xx'} = 1$ possible in our system. Indeed the difference in losses $L_{fit}\big|_{B_{xx'}=0}$ and $L_{fit}\big|_{B_{xx'}=1}$ can be drastic, and in the worst case we have $L_{fit}\big|_{B_{xx'}=0} - L_{fit}\big|_{B_{xx'}=1} = f + 2$ (in the worse case, without using a single very important important feature, each output feature can only be predicted as a constant, and the same for the reward and done). Therefore, if $p_{\max} < 1$, we have $L_{fit} = (f+2)(1 - p_{\max})$, and the loss will not go lower than this value.

Another issue that $p_{\max} < 1$ yields is duplicated features. Indeed, if an important feature is only given with probability $p_{\max} < 1$ and $\varepsilon = 1 - p_{\max}$, it would be beneficial for decreasing the loss L_{fit} to duplicate this feature (the decoder uses a spare feature to output the same value). Then, the model would *not* receive the value for an important feature with probability ε^2, instead of ε. This decreases the loss $\varepsilon \ll 1$ times. Therefore, training the model with $p_{\max} < 1$ would likely lead to feature duplication.

In contrast, having a lower bound $p = p_{\min} > 0$ would not have significant effects on the loss, as adding a non-important feature cannot improve the loss significantly, and definitely cannot worsen the loss (because more features are used). Now, since $p = p_{\min} \approx 0$, we have

$$L_{fit} = p \underbrace{L_{fit}\big|_{B_{xx'}=1}}_{L_1} + (1-p) \underbrace{{}_{fit}\big|_{B_{xx'}=0}}_{L_0}$$

Since $L_1 \leq L_0$, $L_{fit} \leq pL_0 + (1-p)L_0 = L_0$, and $L_0 \geq L_{fit}$. On the other hand, $L_{fit} \geq (1-p)L_0$, which gives $L_0 \leq \frac{L_{fit}}{1-p} = L_{fit}(1 + p + O(p^2)) \approx L_{fit}(1+p)$.

This means that $L_0 \in (1, 1/(1-p))L_{fit}$. For $p = p_{\min} \approx 0$, the gap between the two solutions is small: $L_0 \approx L_{fit}$.

It means that the loss with $B_{xx'}$ disabled is upper-bounded by the current loss L_{fit} (where $B_{xx'}$ is only enabled with a small probability). To output the final answer causal graph, we could threshold these probabilities to 0, and the loss given by this answer graph would only be worse than the training loss by a small amount in terms of the fit quality.

Example 3.2.8. *Let's consider different cases, assuming that the model was fitted to convergence on a set of features with a fixed tensor of probabilities. Consider the case of* 2 *input and output features, with features being equal and having a deterministic relationship (from input to output).*

1. $p = 0$ *for all pairs. In this case, no features are given to the model, and the model will learn to predict the mean of the output features. The causal graph is empty. The loss is equal to* 2 *(since standard deviation for each feature is* 1 *w.l.o.g. because of the relative error).*
2. $p = 1$ *for all pairs. In this case, the model uses all features, and the error is minimal. The loss is* 0.
3. $p = 0.5$ *for all pairs. In this case, the model will try to extract as much information as possible when the feature is present. Essentially, with probability* 0.25, *the model will give the mean output feature, and with probability* 0.75 *it will use either one of the input features, or both of them. The loss is equal to* $2 \times 0.25 = 0.5$.
4. *In the general case* $p \in [0, 1]$, *the model has an error of* 2 *with probability* $(1-p)^2$ *and a loss of* 0 *with probability* $1 - (1-p)^2$. *The expectation is* $2(1-p)^2$.

Now consider a similar case when the mutual information between features is 0 *(they are independent). In this case, for a probability* p, *we have the loss being* 2 *with probability* $(1-p)^2$, *a loss of* 1 *with probability* $2p(1-p)$ *and a loss of* 0 *with probability* p^2. *The expectation is* $2(1-p)^2 + 2p(1-p) = 2 - 4p + 2p = 2 - 2p = 2(1-p)$. *This loss is higher than for the case of equal features for all* $p \in (0, 1)$.

To avoid a case when there are no samples with $\xi = 1$ *(with low* p*), we limit the range of* p *to be* $p \geq p_{\min} = 0.01 = 1\%$. *Setting this value higher results in feature duplicates, as the model tries to predict the outputs even from partially present features. A value of* 1% *with a batch size of* $T = 5000$ *gives* 50 *samples with* $\xi = 1$ *in the worse case (on average), which is empirically sufficient to turn features on. To turn the features off, we introduce a sparsity regularizer.*

3.2.6 Adding the mask as the input to the model to adjust for non-stationarity

Consider the data that the functions F_x are fit on, when the probabilities P change:

$$fa_t \odot B_x \to x_{t+1}$$

Since B_x depends on P_x, the dataset clearly depends on P. Since the probabilities change during the training (a feature could be turned on and off repeatedly in search of the best graph), the problem becomes highly non-stationary – effectively, the model has to re-relearn to rely on the new set of features that it did not see before. In addition, in case of discrete features, the model might not be able to distinguish between the case where $fa_t = 0$ or $B_x = 0$. It may happen that in case if $f_1 = 0$ is given, the model should rely on $f_1 = 0$. However, if a feature is not given, the model should rely on another, more noisy feature. However, an input of 0 does not allow to distinguish between these cases, and the model would be forced to "guess" (output the expected value between the two data-sets). A solution for this is to feed the mask to the model as well[4]:

$$\hat{x} = F_x(fa_t \odot B_x, B_x)$$

3.2.7 Separate exploration loss

To speed up the convergence even more for the cases $p \approx 0$ and $p \approx 1$, we temporarily threshold the probabilities to be in the range $[p_{\min}^{expl}, p_{\max}^{expl}]$, and fit the models with masks B sampled from this distribution. This results in the loss

$$L_{fit}\Big|_{p:=p^{expl}} = \mathbb{E}_{B\sim Be(p^{expl})} L_{fit} \tag{3.5}$$

3.2.8 Additional feature transformation ("rotation")

During training, the distribution given by the the decoder is non-stationary, because the feature space changes to optimize for sparsity. This happens because the model affects the decoder, and the reconstructor must match the decoder.

When the feature space changes, the reconstruction loss would increase drastically, as the reconstructor is trained for the "old" feature space. Fitting the reconstructor is costly, and, therefore, we introduce two separate linear transforms right before and after the model (see Figure 3.9). These transforms allow to quickly enable or disable features, or to change the linear

[4]A similar approach can be found in [94, 29], but the ablation study was not done for whether or not it improves the setup

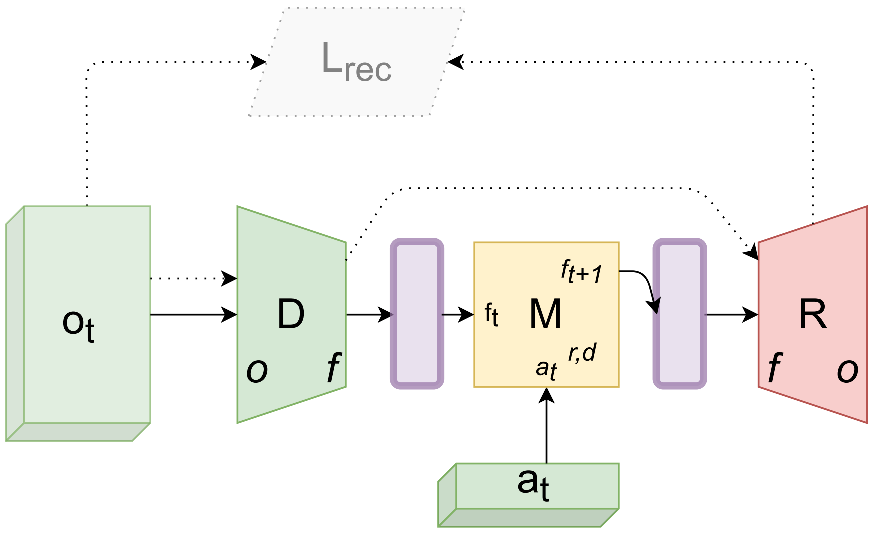


Figure 3.9: Two additional linear feature transformations (purple) applied before and after the model M in order to speed up the convergence. The reconstruction loss path (dotted) bypasses the additional transformations. See Figure 3.7 for the full description of the components

combination of them:

$$\begin{aligned}
L_{fit,obs} &= \rho(R(Z_{post}(M_f(Z_{pre}(D(o_t)), a_t))), o_{t+1}); & R \circ Z_{post} \circ M_f \circ Z_{pre} \circ D &\mapsto o_{t+1},\\
L_{fit,feat} &= \rho(Z_{post}(M_f(Z_{pre}(D(o_t)), a_t)), D(o_{t+1})); & Z_{post} \circ M_f \circ Z_{pre} \circ D &\mapsto Do_{t+1},\\
L_{fit,m-feat} &= \rho(M_f(Z_{pre}(D(o_t)), a_t), Z_{pre}(D(o_{t+1}))); & M_f \circ Z_{pre} \circ D &\mapsto Z_{pre} \circ Do_{t+1}.
\end{aligned}$$

These transforms make it simpler for the model to quickly change the feature space, without affecting the reconstruction loss.

3.2.9 NP-hardness in the general case

In the EPFL semester project at LCN, I considered the linear case of CauseOccam and proved the NP-hardness. Since a non-linear case includes a linear one, learning non-linear sparse causal models jointly with a representation is NP-hard as well.

While it was shown [22] that finding the causal graph without a decoder is not NP-hard, the learned representation makes it NP-hard.

Indeed, consider the linear dynamics $f_{t+1} = M_f f_t + M_a a_t$, where M_f and M_a are matrices of sizes $f \times f$ and $f \times a$. The decoder is also linear: $f_t = Do_t$ where D is a matrix of size $f \times o$. The reconstructor R s.t. $o_t = Rf_t$ is linear as well and has size $o \times f$. In the case where $f = o$ and D is invertible, we can set $R = D^{-1}$. Now, the model for observations is linear as well: $o_{t+1} = D^{-1}(M_f Do_t + M_a a_t)$. Since the dynamics of the environment is fixed, the matrices $M_o = D^{-1} M_f D$ and $M_a^o = D^{-1} M_a$ are determined by the environment. Therefore, our task of finding a sparse model in this case boils down to finding a decoder matrix D that would give the most sparse feature transition matrix DM_oD^{-1} and the action transition matrix DM_a^o:

$$\|DM_oD^{-1}\|_0 + \|DM_a^o\|_0 \to \min_{\exists D^{-1}}$$

Now, if $M_o = 0$ (the environment does not have a state), the problem becomes (exactly) Sparse Dictionary Learning [125], a known NP-hard problem. Since this is a special case of our problem, our problem is NP-hard as well.

Another extreme $M_a^o = 0$ requires to find a basis given by D in which a matrix DM_oD^{-1} is most sparse. If we allow for complex variables, this problem is polynomial-time computable, as we only need to diagonalize the matrix M_o. An approximation in the real case could be the Jordan normal form.

Note that the case of one action corresponding exactly to one feature [116, 23] means that DM_a^o has only a single element in each column. Since D transforms the columns of M_a^o, in this linear example, it would mean that the columns of M_a^o are collinear, and we can find D by finding sets of collinear vectors in M_a^o in polynomial time.

3.2.10 Optimization schemes

Now, we have a number of losses (see Figure 3.7 for a diagram of all components) that we would like to minimize together

$$\begin{aligned} L_{sparse} &\to \min_{P} \\ L_{rec} &\to \min_{D,R} \\ L_{fit,feat} &\to \min_{D,M,P,R} \\ L_{fit,obs} &\to \min_{D,M,P,R} \end{aligned} \tag{3.6}$$

Ideally, we would like to find a point which jointly minimizes all of these losses. In the case of a deterministic environment, it is possible to achieve all of the losses being 0, L_{sparse} having the minimal possible discrete value.

The traditional practical reformulation of this problem in terms of a Machine Learning problem would result in a loss consisting of two components: one to fit the parameters of the posterior to the data, and one to regularize the model for sparsity, with a coefficient λ controlling how "strong" is the posterior [110]. Such an approach, however, requires to "guess" the parameter λ [87, 127]. A much more convenient way is to either guess the target number of edges in the graph $K(B)$, or the losses constraint $L_i \leq c_i$ for $i \in \{fitobs, fitfeat, rec\}$. We present the approaches below:

- Linear combination of losses with constant coefficients:

$$\mathcal{L} = \lambda_{sparse} \cdot L_{sparse} + \lambda_{fit,feat} \cdot L_{fit,feat} + \lambda_{fit,obs} \cdot L_{fit,obs} + \lambda_{rec} \cdot L_{rec} \to \min_{D,M,R,P} \tag{3.7}$$

- Constrained problem with sparsity guess:

$$\begin{aligned} \min_{D,M,R,P} \quad & \lambda_{fit,feat} \cdot L_{fit,feat} + \lambda_{fit,obs} \cdot L_{fit,obs} + \lambda_{rec} \cdot L_{rec} \\ s.t. \quad & L_{sparse} \leq c_{sparse} \end{aligned} \tag{3.8}$$

- Constrained problem with fit guess and a linear combination:

$$\begin{aligned} \min_{D,M,R,P} \quad & L_{sparse} \\ s.t. \quad & \lambda_{fit,feat} \cdot L_{fit,feat} + \lambda_{fit,obs} \cdot L_{fit,obs} + \lambda_{rec} \cdot L_{rec} \leq c_{fit,comb} \end{aligned} \tag{3.9}$$

- Constrained problem with granular fit guess:

$$\begin{aligned} \min_{D,M,R,P} \quad & L_{sparse} \\ s.t. \quad & L_{fit,feat} \leq c_{fit,feat} \\ s.t. \quad & L_{fit,obs} \leq c_{fit,obs} \\ s.t. \quad & L_{rec} \leq c_{rec} \end{aligned} \tag{3.10}$$

- If the target sparsity is known, a projection [26] to the l_1 simplex can be applied, possibly via the proximal gradient descent method [30].

- Adaptive sparsity [35] with a multiplicative update for the λ parameter given the reconstruction losses. Given a target constraint for the linear combination of all losses but the sparsity loss $L_{\setminus sparse} \leq \varepsilon$, we update the parameter λ_{sparse} in Equation 3.7. If $L_{\setminus sparse} \leq \varepsilon$, we increase $\lambda_{sparse} := \lambda_{sparse} \cdot (1 + \delta)$, and otherwise decrease it $\lambda_{sparse} := \lambda_{sparse} \cdot (1 - \delta)$. We note that this approach is ideologically similar to the Lagrangian update, but does not consider the magnitude of the reconstruction loss compared to the threshold.

We compare the versions 3.7, 3.8, 3.9, 3.10 in chapter 5.

In terms of the ease of use, Equation 3.10 requires least knowledge about the problem. The parameters λ, if set manually, have a drastic effect on the quality of the solution. In contrast, in Equation 3.10 we only set the relative thresholds for the losses (values less than 1), and obtain the most sparse causal graph allowing for such a loss.

Solving constrained problems. To solve the constrained problems with gradient descent we use the Lagrange primal-dual method [3]. We consider the Lagrange function for the Equation 3.10:

$$L_{agrange} = L_{sparse} + \lambda_{fit,feat}(L_{fit,feat} - c_{fit,feat}) + \lambda_{fit,obs}(L_{fit,obs} - c_{fit,obs}) + \lambda_{rec}(L_{rec} - c_{rec})$$

We note that solving Equation 3.10 corresponds to solving

$$\min_{D,M,R,P} \max_{\{\lambda_i \geq 0\}} L_{agrange},\ i \in \{fitfeat, fitobs, rec\} \tag{3.11}$$

Indeed, an an optimal point of Equation 3.10 is a saddle point of Equation 3.11 [44]. Roughly speaking, this happens because if a constraint is violated, any $\lambda > 0$ results in an infinite value of the loss.

To solve this problem, a primal-dual descent-ascent method can be applied [44] (here $x = (D, M, R, P)$ – the parameters of all neural networks):

$$\begin{aligned} x_{t+1} &= x - \eta\frac{\partial L_{agrange}}{\partial x} \\ \lambda_i &= \lambda_i + \eta\frac{\partial L_{agrange}}{\partial \lambda_i},\ i \in \{fitfeat, fitobs, rec\} \end{aligned} \tag{3.12}$$

The algorithm has the following intuitive interpretation. Suppose that λ_i is small, and $i'th$ constraint is violated: $L_i > c_i$. Since $\frac{\partial L_{agrange}}{\partial \lambda_i} = L_i - c_i > 0$, the parameter λ_i would increase. If, in contrast, the constraint has a slack (is not violated) and $L_i < c_i$, the parameter λ_i would decrease to allow optimizing for the objective L_{sparse}. Therefore, we expect the following dynamics: first, the algorithm would turn most of the features on in order to satisfy the constraints, and then it would optimize for sparsity.

If we apply this algorithm for the other (symmetric) formulation of our problem (Equation 3.8), it would first disable features, and then, once the target sparsity level was reached, it would care about fitting the reconstructor and the SCM functions.

Chapter 4

Implementation

In this chapter we discuss the specific of our implementation of CauseOccam. We touch such issues as the initialization of the probability values, tips and know-how during training, and the project structure.

4.0.1 Initialization

We initialize the probability matrix to be an identity for the features (a feature only affects itself), and to a full matrix for the rest (actions, reward and done). We compare this method to initializing with all zeros, all ones, or a random initialization in the next section

4.0.2 Metrics

In order to assess the quality of the solution, we track the following quantities:

- Sparsity gap. We compute L_{fit} with the current mask $B \sim Be(P)$, as well as with a fully-enabled mask (or the complete causal graph) $B_{xx'} = 1$. This gives losses $L_p = L^p_{fit}$ and $L_{full} = L^{p:=1}_{fit}$.

  Now, we define $\Delta^{add}_{sparse} = L_p - L_{full}$. This quantity is positive if the model is fit well (adding features should not worsen the prediction quality). The value of this metric shows how well feature selection has worked: if the value is high, it means that some of the required for prediction features are currently disabled (or, that the model is not fit enough).

  A multiplicative version $\Delta^{mult}_{sparse} = \frac{L_p - L_{full}}{L_p}$. This value is positive when Δ^{add} is positive, and this value is less than 1, because $L_{full} > 0$

- Number of edges in the graph, or the number of large components in the matrix P. We compute a k-means clustering[1] in 1-dimensional space of all probabilities $\{P_{xx'}\}_{x\in X_{in}, x'\in X_{out}}$. This gives a threshold $\tau \in [0,1]$. Next, we select the pairs (x, x') such that $P_{xx'} > \tau$. This gives the answer causal graph, and the number of non-zero components.
- Total entropy of distributions $Be(P_{xx'})$. This quantity should be low at the end of training, because all features should be either turned on or off. At the end, we also expect it to be low, as only the important features have $P \approx 1$, and the rest have $P \approx 0$. In the middle of the training, this quantity must increase.

4.0.3 Separate networks to predict pixels and features

Since we would like our features to be disentangled (corresponding to different things), there is (by definition) no benefit of multi-task learning. In our experiments (see chapter 5) we show that multi-task learning harms convergence speed in case if the tasks are not related. We explain it by the fact that a shared model has layers that are used by all tasks. If the tasks are not related, it is hard for gradient descent to "split" the neurons between tasks, as all of them "want" to use the neurons at a layer. The same applies to the reconstructor (predicting weakly correlated pixels in a toy grid world), and the decoder (predicting features that are disentangled)[2]. This approach can be found in [74]

In order to implement the separate networks we use the `einsum` operator in PyTorch to implement "vectorized" fully-connected networks.

4.0.4 Project code structure

The code can be found in our https://github.com/sergeivolodin/causality-disentanglement-rl. The code uses gin config files for modular configuration, ray tune for hyperparameter tuning, Tensorboard for tracking metrics real-time, and Sacred to save all the data from all experiments. Parallel data collection is implemented with ray, and hyperparameter tuning studies are done with ray tune. The code has function-level and class-level documentation, unit- and integration tests, with continuous integration on GitHub.

[1]A simple thresholding with $p = 0.5$ would suffice to obtain the final graph, however, the benefit of K-means is that it works even when the training is not fully complete, and the probabilities are not fully $p_{\min}$ or 1. This allows the see visually the preliminary results while training and quickly understand if the trial is likely to be successful.

[2]In case of a convolutional decoder, the convolutional layers are shared, but the subsequent fully-connected layers are not. Decoding the image is a shared task, but decoding features from that representation is not

4.0.5 Data collection and the true distribution

We run k parallel data collection processes with ray, and aggregate the data in one ring buffer process. Each mini-batch is sampled as random steps from all of the collected episodes and given to the learner class. The ring buffer process maintains the ratio of collected steps to sampled steps to be at least a constant fraction $\zeta = 0.5$ to prevent overfitting to only collected steps.

We note that this does not result in a true distribution $\mathbb{E}_{h\sim(\mu\leftrightarrow\pi)}\mathbb{E}_{t\sim Uniform([T])}$. The true distribution requires us to sample one episode, select a random step, and then discard the rest of the episode. Effectively, it would require to set $\zeta = \mathbb{E}_h T$: we sample T times more steps than "required". This would slow down the convergence significantly. In a mini-batch from the true distribution, all the steps are likely to be from different episodes. This is important for procedurally-generated environments which have a different layout for each new episode.

4.0.6 Number of optimizers

Consider the loss from Equation 3.7. It depends on multiple groups of variables: the decoder D, the model M, the reconstructor R, and the matrix of probabilities P. We test two options for optimizing L:

- One optimizer, updating each group of variables at every iteration.
- 4 optimizers, each updating one of $\{D, M, R, P\}$, and doing multiple steps at a time. This corresponds to finding the optimum a bit more precisely for one group of parameters. The rationale is the following. Since for a single group of parameters (such as M), the problem looks more like supervised learning (the model receives a stationary distribution of inputs, and has to fit a stationary distribution of outputs), the problem becomes more like what neural networks are well-tested at – supervised learning, albeit only for a few iterations.

4.0.7 Time-scale separation for Lagrange multipliers

Sometimes the order of magnitude of λ_i in Equation 3.10 or Equation 3.9 needs to be changed significantly. For example, the value of the sparsity has recently decreased significantly, which leads to a much bigger emphasis on the constraints. We need to decrease the Lagrange multiplier, or, otherwise, too little emphasis is put on decreasing sparsity even more. With standard parameterization, λ_i would decrease exponentially like in standard Gradient Descent.

If we reparameterize $\lambda_i = \sqrt{\mu_i}$, or $\lambda_i = \mu_i^2$ or $\lambda_i = \exp(\mu_i)$, the parameter λ_i would change slower (in the first two cases) or faster (in the last two cases). Such a difference cannot be achieved by simply changing the learning rate for λ_i, as such modifications would only change the "slope of the line", while reparameterization changes the asymptotic behavior (for example, a line vs. a quadratic function).

A result shows that a sufficient time-scale separation is crucial [42]. [104] considers a similar bi-level problems as a Stackelberg game (where the follower adapts to the leader). The takeaway is to update the follower faster. We achieve this by doing more optimization steps for the decoder and the reconstructor, and less for the model.

Given a single batch of data at epoch i, we try the following optimization schemes:

1. Updating all parameters at every epoch i
2. Updating the model for τ_m epochs, then the decoder for τ_d epochs, then the reconstructor for τ_r epochs.

Empirically, the last approach works better.

Chapter 5

Evaluation

In this section, we present the results of running CauseOccam on a number of benchmark environment that we develop, as well as on established baselines (such as the pure-navigation grid-world). We denote the resulting graph given by the algorithm as $\hat{G}_{x'x}$ for whether there is an edge from a cause variable (one at the current time-step) $x \in X_{in} = \{f_1, ..., f_f, a_1, ..., a_a\}$ to an effect variable (one at the next time-step) $x' \in X_{out} = \{f_1, ..., f_f, r, d\}$ (see chapter 3). The true causal graph for an environment (if exists) is denoted as $G_{x'x}$. The matrices $G_{x'x}$ and $\hat{G}_{x'x}$ have binary values $G_{x'x} \in \{0, 1\}$. 1 means that the edge is present, and 0 means that the edge is absent. The total number of edges is $K(G) = \sum\limits_{x \in X_{in}} \sum\limits_{x' \in X_{out}} G_{x'x}$. To obtain $\hat{G}$ from P, a thresholding algorithm (K-Means) is applied (see chapter 4 and chapter 3).

5.1 Environments

We describe the environments that we develop for this project, as well as established benchmarks. Each environment tests a particular aspect of learning causal graphs. The SparseMatrix environment considers a case of correlated features in which each action can affect multiple features. VectorIncrement has a simple dynamics, but the observations are encoded in pixel space. KeyChest environment is a grid-world with variable complexity with both frequently changing and hard-to-change features.

5.1.1 SparseMatrix(n, k) environment

The simplest environment we consider is SparseMatrix(n, k). Given the dimensionality of the space, we randomly sample a matrix $A \in \mathbb{R}^{n \times n}$ at the initialization of the environment (once per training session, i.e. the matrix is the same after a `reset()`).

The matrix A is sampled to have $k \geq n$ non-zero component and that it is non-degenerate. We first permute the numbers $[n]$, and then set $A_{ij} := \pm 1$. After that, elements $k - n$ are added to random "free" (zero) spots $A_{ij} := \pm 1$.

At each episode, the state of the environment is initialized as $s_1^i \sim N(0,1)$ for $i \in [n]$. At each time-step, the state is multiplied by the transition matrix:

$$s_{t+1} = As_t$$

The actions are ignored for this environment, and the reward is always 0. An encoder is not applied, i.e. the environment's observation is the state.

The true causal graph (without a decoder) for this environment always equals to non-zero components of the transition matrix: $G^{true}_{f_i f_j} = \mathbb{1}_{A_{ij} \neq 0}$.

Additionally, we add a sparse random matrix influencing actions. To make the effect of actions non-linear, we only apply actions to positive state components:

$$\tilde{s}_{t+1} = As_t,\ s^i_{t+1} = \mathbb{1}_{\tilde{s}_{t+1} > 0} \cdot \sum_j B_{ij} a^j_t$$

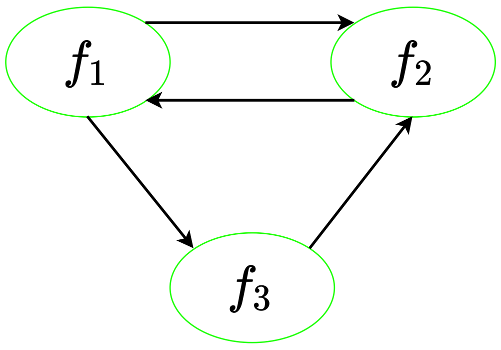


Figure 5.1: SparseMatrix$(3, 4)$ environment for the given A

Example 5.1.1. *If we take $n = 3$ and the following matrix:*

$$A = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$

we obtain the causal graph shown in Figure 5.1.

Example 5.1.2. *We test our approach on the following matrices for $n = 5$:*

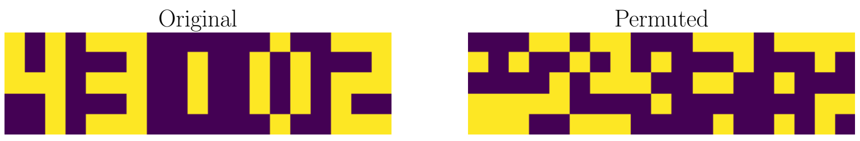


Figure 5.2: The 5 components of the VectorIncrement environment displayed as 5×19 images. The left image shows the digits, and the right image shows the same observation with a fixed permutation applied.

$$A = \begin{pmatrix} 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix}, B = \begin{pmatrix} 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 \\ -1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

5.1.2 VectorIncrement(n) environment

The state has n components, and there are n actions. At the beginning, the state is a zero vector $s_1 = 0 \in \mathbb{R}^n$. Executing action i results in an increment in the i'th component of the state vector:

$$s^j_{t+1} = \begin{cases} s^j_t, & a \neq j \\ s^j_t + 1, & a = j \end{cases}$$

The reward is given when the lowest component is incremented. A somewhat optimal strategy is to stay close to the diagonal of the environment:

$$a_t = \arg\min\{s^i_t | i \in [n]\}$$

If no encoder is applied, the ground-truth graph (without an encoder or a decoder) is given by $G_{f_i f_j} = \mathbb{1}_{i=j}$, $G_{f_i a_j} = \mathbb{1}_{i=j}$, $G_{d,f_i} = 1$, $G_{d,a_j} = 0$, $G_{r,f_i} = 1$, $G(r, a_i) = 1$ for all but one action (we can always determine the action taken by all but one one-hot variables, since at every step exactly one action is taken). See Figure 5.3.

The environment additionally supports rendering observations as images using pixelized digits (see Figure 5.2), as well as optionally permuting the pixels of the observation to make it hard for humans to play.

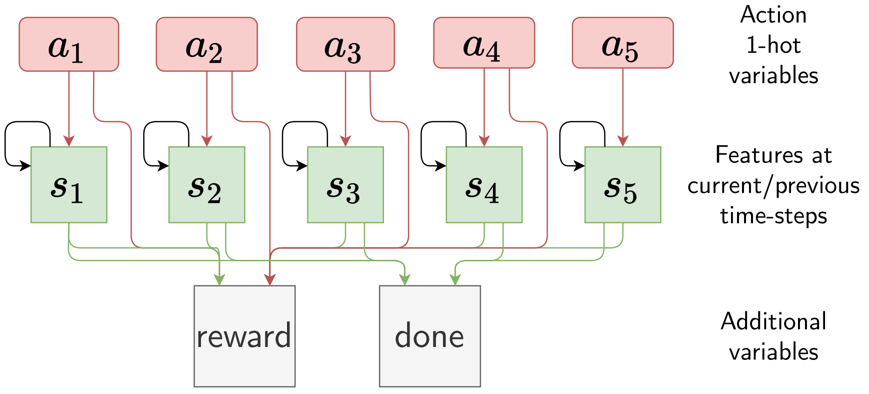


Figure 5.3: The causal graph for VectorIncrement(5) without a decoder or an encoder. First, each state component determines its own value at the next time-step (black self-loops for s_i). Secondly, a corresponding action increments the state component (red line from a_i to s_i). Finally, all state components affect reward and done (green lines from s_i to reward and done), and all but one action affect the reward (red lines from a_i to reward). This one of 5 possible causal graphs for this environment (one of the actions can be disconnected from the reward node). The number of edges is equal to $K(G) = 24$ (out of 120), or $K(G) = 5n - 1$ out of $2n^2 + 4n$ in the general case of VectorIncrement(n). This family of graphs is not minimal in terms of the number of edges for this environment.

5.1.3 KeyChest(h, w, f, c, k) environment

The environment is described fully in my previous publication [126] at ICLR CLDM 2020. h is the height of the grid-world, and w is the width. f is the number of food items, c is the number of chests, and k is the number of keys. The correct causal graph is shown in Figure 5.4

5.2 Learned causal graphs

5.2.1 SparseMatrix$(5, 7)$ environment

First, we run the algorithm on the environment without an encoder. The task then simply corresponds to selecting the relevant features. The algorithm uncovers the correct graph (see Figure 5.6).

The results can be replicated by running the following command (about 3 minutes):

```
python -m sparse_causal_model_learner_rl.learner \
 --n_gpus 1 --nofail \
 --config sparse_causal_model_learner_rl/configs/rl_const_sparsity_obs_space.gin \
 --config sparse_causal_model_learner_rl/configs/env_sm5.gin \
```

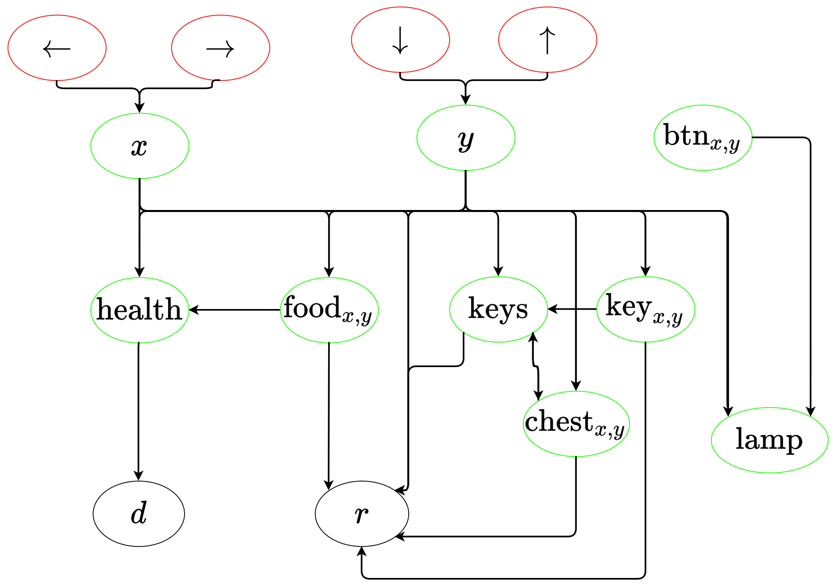


Figure 5.4: The causal graph for the KeyChest environment. The actions influence the coordinates. At each time-step, the health variable decreases, and if it is 0, the episode ends. Collected food (player coordinates equal to the food coordinates) increases health, and food disappears. Keys can be collected as well. A chest can only be collected in case if there are keys already collected, and in that case the number of collected keys decreases. A lamp is toggled in case if the player is at the button position.

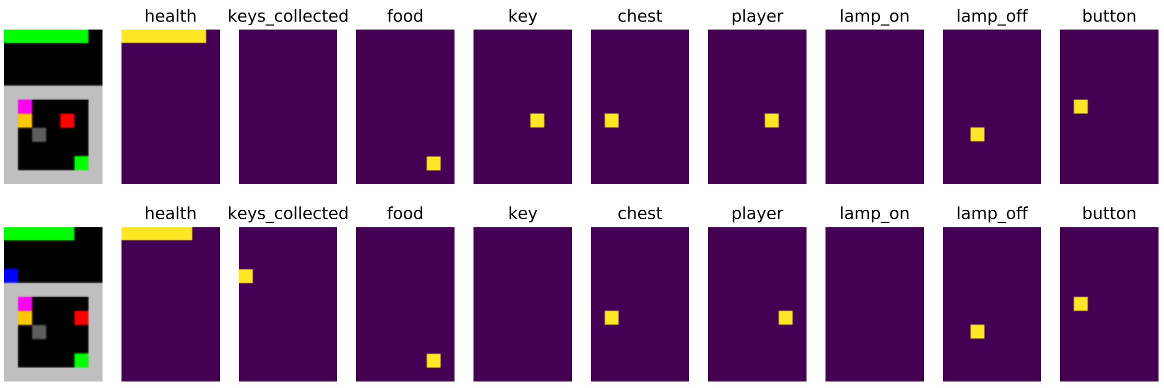
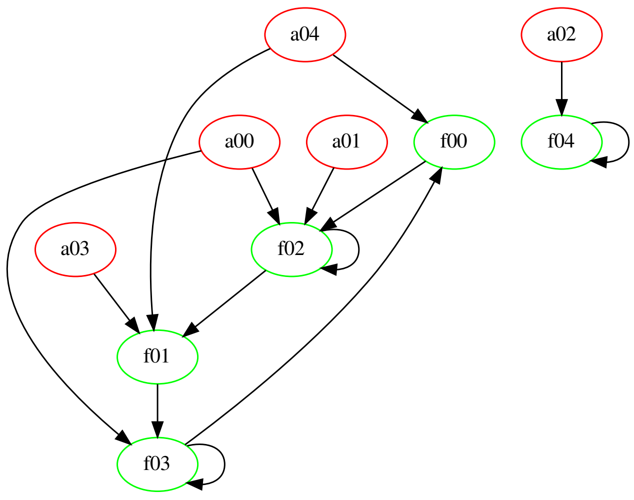


Figure 5.5: The two consecutive time-steps (top and bottom rows) of the KeyChest environment. The rows show the RGB render (leftmost image) and the binary object masks given as an observation. The player moved to the right and has collected a key. The health bar has decreased by 1 element.



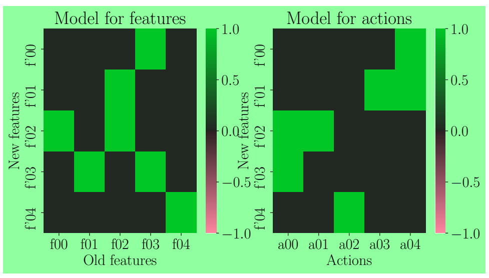


Figure 5.6: The learned graph for SparseMatrix$(5, 7)$ without an encoder. Left: the resulting causal graph. Right: the causal graph as a node adjacency matrix.

```
 --config sparse_causal_model_learner_rl/configs/with_lagrange_dual_sparsity.gin
```

5.2.2 VectorIncrement(5) environment

Given the permuted pixels, the method successfully uncovers a correct causal graph, with even fewer edges than anticipated. The results can be reproduced by running the following command (about 7.5 hours):

```
python -m sparse_causal_model_learner_rl.learner \
  --n_gpus 1 --nofail \
  --config sparse_causal_model_learner_rl/configs/rl_const_sparsity_obs_space.gin \
  --config sparse_causal_model_learner_rl/configs/env_ve5_with_rew_done.gin \
  --config sparse_causal_model_learner_rl/configs/with_lagrange_dual_sparsity.gin
```

The correlation matrix between the decoder features and the digits is shown in Figure 5.7. The step variable can be obtained from one of the features with $R^2 = 0.989$ and the p-value $p < 10^{-4}$.

The Figure 5.9 shows the learned graph for the environment. It can be seen that the graph is simpler than expected (Figure 5.3). Indeed, while a_3, a_1, a_4, a_0 are associated with features f_1, f_3, f_4, f_0, the action a_2 does not correspond to any feature. In addition, only the feature f_2

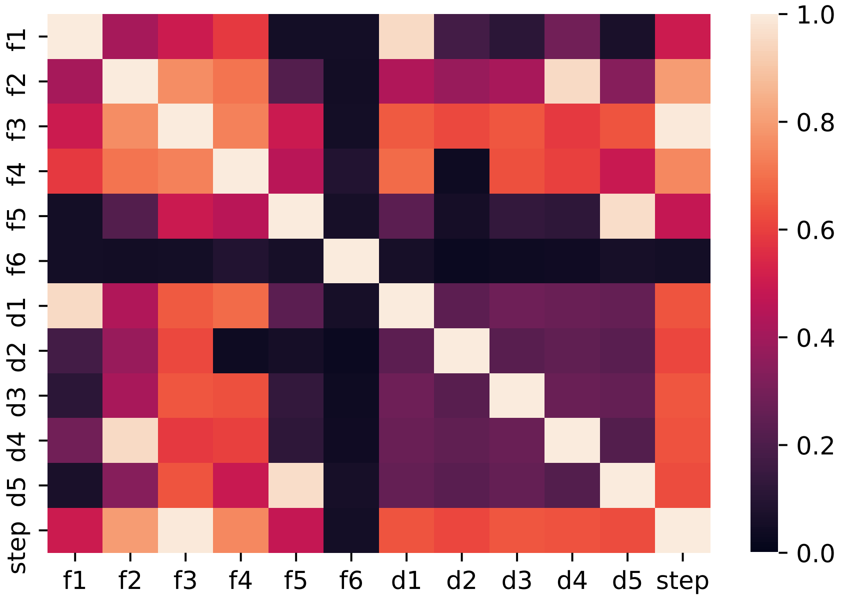


Figure 5.7: The feature correlation matrix for the learned model for the VectorIncrement(5) environment with pixels as observations. $f_1, ..., f_6$ are the learned features (f_6 is a constant feature and can be ignored), $d_1, ..., d_5$ are the ground truth values of the state components, and `step`is the time-step. The block-diagonal entries show that there is weak correlation between digits, and no strong correlations between features. In contrast, the correlations between features and digits are high: d_1 is correlated with f_1, d_4 with f_2, `step` with f_3, and d_5 with f_5.

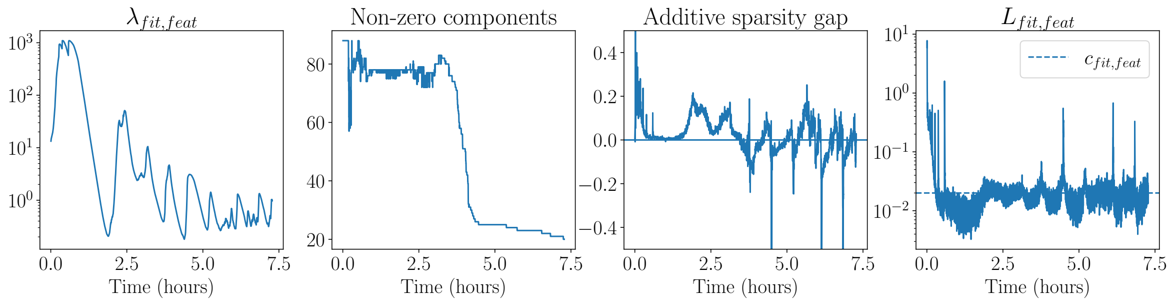


Figure 5.8: The losses and parameters when running CauseOccam on VectorIncrement(5). The leftmost chart shows the Lagrange multiplier associated with the fit constraint $L_{fit,feat} \leq c_{fit,feat}$. The second chart shows the number of non-zero components in the node adjacency matrix. The third chart shows the adaptive sparsity gap, and the final chart shows the fit loss $L_{fit,feat}$.

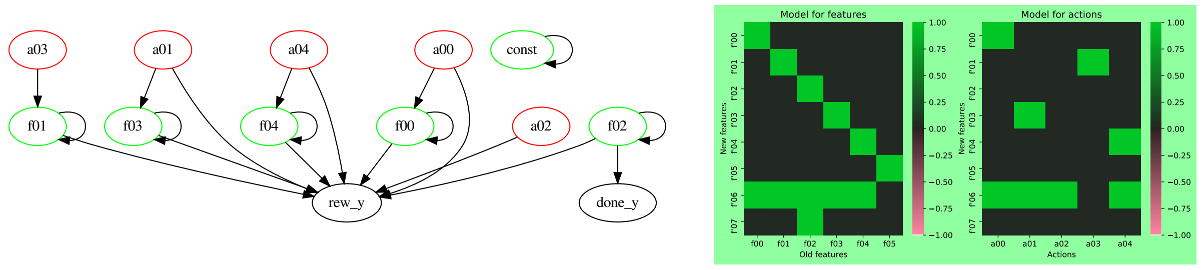


Figure 5.9: The learned graph for VectorIncrement(5). Left: the resulting causal graph. Right: the causal graph as a node adjacency matrix.

determines `done`. This is actually correct: indeed, `done` can be computed via the current time-step, regardless of the values of the state components. The value of the state component associated with a_2 can be computed by subtracting all the other features from the time feature. Additionally, a_3 does not affect the reward. This is also correct, because at each time-step at least one action is taken. Therefore, we can determine that a_3 is taken by noticing that no other action is taken.

The Figure 5.8 shows the training curves during the run of CauseOccam. First, the Lagrange multiplier increases up to its maximal value of 10^3, which corresponds to the phase where the initial (non-sparse) model is learned. Next, the parameter decreases and oscillates with decreasing magnitude. This corresponds to selecting the right edges in the graph. The oscillation in the Lagrange multiplier cause the values of P_i to oscillate as well, which leads to an oscillation in the fit loss. Note that the sparsity gap is also oscillating, which means that the oscillation is caused solely by the change in the probabilities (and not in the quality of the model given all the features).

5.2.3 **KeyChest**$(5, 5)$ **environment**

We first run the algorithm on a vanilla grid-world (just the coordinate dynamics). The results can be reproduced via running the following command (takes about 3 minutes):

```
python -m sparse_causal_model_learner_rl.learner \
 --n_gpus 1 --nofail \
 --config sparse_causal_model_learner_rl/configs/rl_const_sparsity_obs_space.gin \
 --config sparse_causal_model_learner_rl/configs/env_gw.gin \
 --config sparse_causal_model_learner_rl/configs/with_lagrange_dual_sparsity.gin
```

The Figure 5.10 shows the resulting learned causal graph which is correct. Indeed, features correspond to the x and y coordinates, since actions are grouped into (left, right) and (down, up).

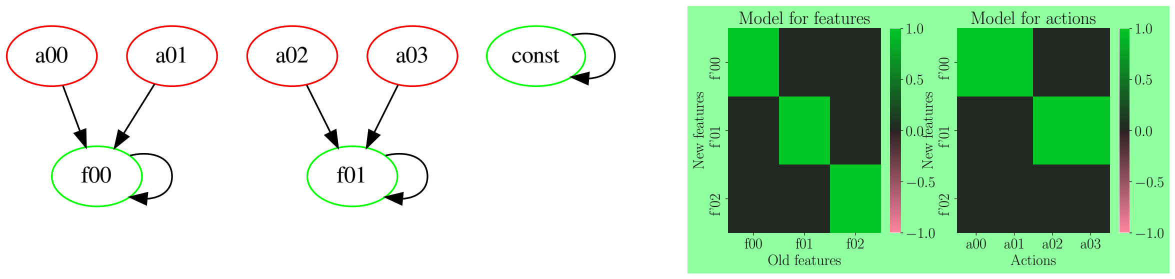


Figure 5.10: The learned graph for KeyChest without any objects or indicators (just the coordinate). Left: the resulting causal graph. Right: the causal graph as a node adjacency matrix.

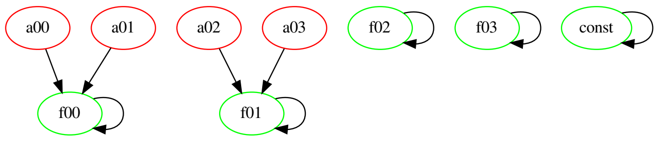


Figure 5.11: The learned graph for KeyChest with only the health bar given along with the player position mask.

The method also succeeds in recovering the health variable (Figure 5.11). The results can be reproduced in 6 hours via

```
python -m sparse_causal_model_learner_rl.learner \
 --n_gpus 1 --nofail \
 --config sparse_causal_model_learner_rl/configs/rl_const_sparsity_obs_space.gin \
 --config keychest/config/5x5_onlycoord_obs.gin \
 --config sparse_causal_model_learner_rl/configs/env_kc_5x5_onlycoord_obs.gin \
 --config sparse_causal_model_learner_rl/configs/with_lagrange_dual_sparsity.gin
```

5.3 Ablation study for VectorIncrement(2)

We run a hyperparameter study with 140 trials on VectorIncrement(2) on pixels. The search is over the following modes (with respect to the `original` configuration):

1. `no_mask`. The binary mask B_i is not given to the model
2. `no_relative`. Instead of the relative MSE loss (chapter 3), the regular Mean-Squared Error loss is used

3. `no_rotation`. The additional transforms (see chapter 3) are not applied
4. `single_model_dec`. The decoder is a single network, instead of a separate network for each feature.
5. `single_model_rec`. The reconstructor is a single network instead of a separate network for each pixel.

The results are shown in Figure 5.12. First, it shows that removing the additional transformation ("rotation") improves the results (losses are lower when comparing `no_rotation` to `original`) for this problem (in contrast, for the KeyChest environment, the version without rotation converges much slower). Secondly, not giving the mask to the model, or not using the relative loss increases all the target metrics, compared to the `original` configuration. Using a multiple networks instead of a single network is generally improving the metrics, except for the additional loss.

Initializing the probabilities with 0 or 1 leads to similar results, but with more time required to fit the data.

5.4 Failure mode analysis

The main issues are that either the model did not reach the target quality fit, or the search is stuck in a local minima. For example, running the method even on the `sm5` environment with a linear encoder does not always provide satisfactory results: the answer graph has more edges than possible.

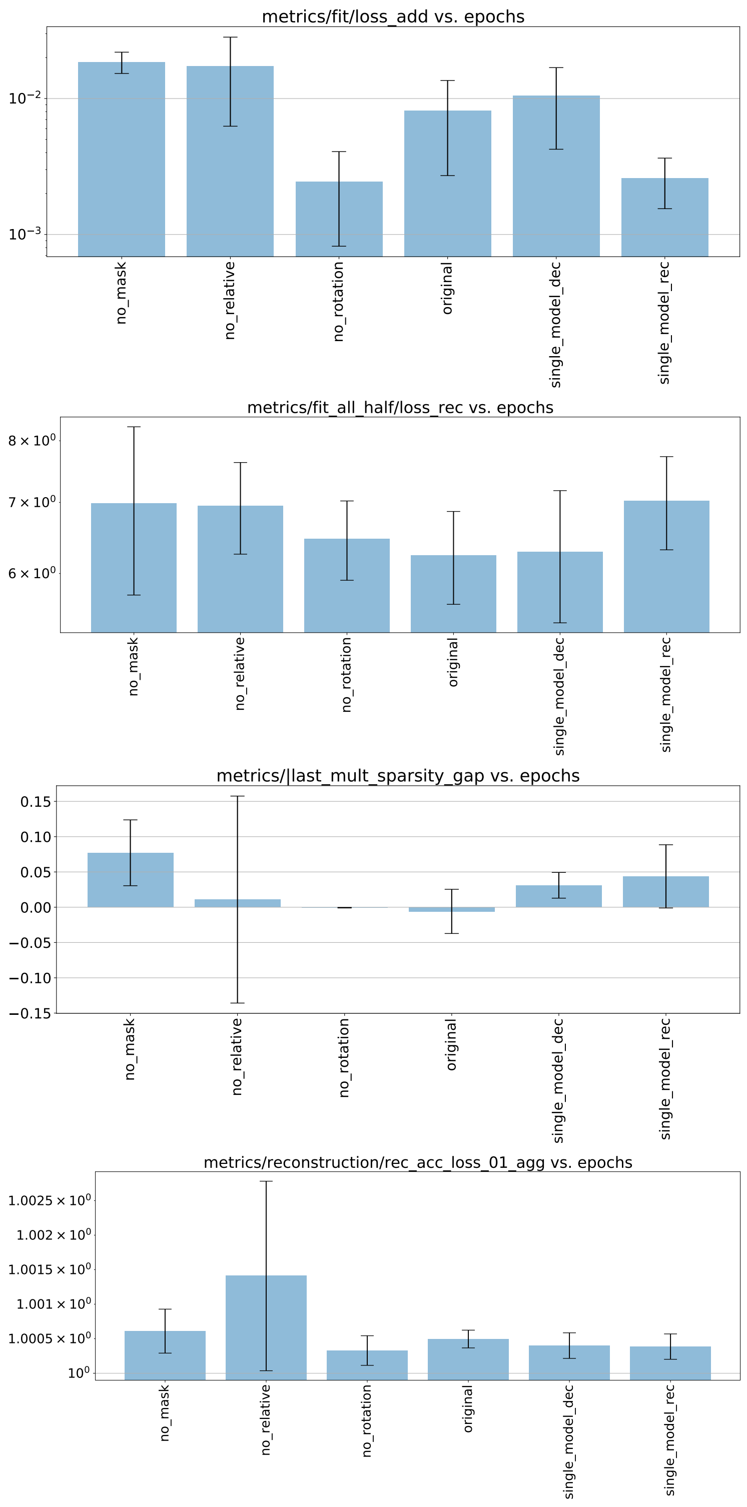


Figure 5.12: The results of the ablation study. Each bar chart represents the metrics gathered during the 140 trials with difference modes. The modes are on the x-axis (see text for description), and the quantity is on the y axis. The charts (from top to bottom) represent: 1) The relative MSE quality of predicting r and d given the selected features 2) The observation fit exploration loss 3) Multiplicative sparsity gap 4) Reconstructor prediction accuracy

Chapter 6

Related Work

Symbolic approaches. [33, 32, 39] considers a symbolic reasoning approach. [69] considers grouping states into "meta states". [23] defines "prototype" states. [132] cluster the environment's state space. [43] clusters time series and fits a Markov model in this space.

The main difference between symbolic approaches and causal graphs is that while the structure of the graph is discrete, the transition functions that determine the nodes' values can be arbitrary neural networks. Our work can be seen as complementary to the case when the transition dynamics is more complex than switching between prototype states (for example, continuous state-spaces), or states cannot be clustered using traditional approaches.

Other approaches to RL or time series interpretability. [116] consider a model where one action corresponds to a change in one feature. Our work generalize this approach (see chapter 3).

[40, 91, 89] use the tools of causal reasoning to interpret agents. Our work can be seen as complementary, as [40, 91, 89] use an existing causal graph of the environment to provide explanation. In contrast, our work focuses on learning the graph.

A number of approaches involve sparse feature vectors for interpretability [128, 136, 120, 18, 31, 85, 65, 103, 100, 83, 101, 41, 64]. See chapter 3 for discussion.

[134] introduce an attention module inside a Reinforcement Learning agent (inside the policy and the value networks), allowing it to solve tasks not solved by a simple MLP, because the task requires relational reasoning. A similar technique can be seen in [52] to solve visual reasoning tasks (similar to IQ tests). Using an attention mechanism could be the next step for our project (see chapter 7).

Natural language for explanations. [89, 38] generate natural language explanations given a causal model of the environment. Our work can be seen as complementary to [89]: we learn the causal graph that can be used as input for that method.

Causal modeling. [10] apply a *linear* decoder and learn a sparse linear model in the context of Granger causality. In our project, we focus on the non-linear case.

A number of approaches learn the sparse graph, but without a learned representation (without a decoder network) [72, 47, 74, 19, 64, 11]. In this thesis, we focus on learning the representation as well.

[72, 97] propose to learn a distribution over causal graph in a mean-field way (all edges are independent and modeled as Bernoulli distribution). Our project uses this technique (see 2).

[17] extend this approach to modeling interventions, and this line of work can be used to design better exploration techniques, when combined with learning sparse graphs (see chapter 7).

[66] learn causal graphs in the context of RL. Our work can be seen as an extension of this project with a learned decoder, and the cases where sparsity is important.

[95] implicitly code for the edges of a causal graph in terms of a neural network, i.e. the network predicts which edges to add or remove. In contrast, we use a matrix of probabilities which determines the node adjacency matrix. Using the approach of [95] could yield potential improvements in our framework.

[68, 113, 76, 20] learn a causal model jointly with a decoder, but do not evaluate the sparsity. In our project, we focus on the sparsity as well as on jointly learning the model with a decoder.

Graph neural networks. [11] proposes Graph Neural Networks [135] (GNNs) based on the idea of thoughts as graphs of connections between abstract concepts. [123] GNNs by allowing a linear (in the number of nodes) number the edges to be learned.

[25] use GNNs to discover unknown laws of dynamics. They apply symbolic regression to the node and edge models of the GNN, to extract the symbolic expression. The model is used for pairwise forces in a particle environment, and for discovering the physical laws of Dark Matter in an automated way. The symbolic expression framework (Eureqa) generalizes better than the non-symbolic model it was extracted from. l_1 sparsity is used in the GNN to ease the problem for the symbolic regression.

One important difference between GNNs and Structural Causal Models (SCMs) used in this project is that the graph itself (as a node adjacency matrix) is usually fixed for GNNs. In contrast, to learn the SCM is to learn the graph of dependencies.

[119] uses contrastive losses to obtain a representation and a model in the latent space. [75] work on learning abstract representations from pixels in 3D environments and in Atari using Graph neural networks, leading to more interpretable world (environment) models. Using the contrastive loss in our approach could yield faster convergence of the world model, and using sparsity constraints along with the approach of [75] could lead to better interpretability.

Chapter 7

Conclusion

We present the first (up to our knowledge) system that learns a sparse causal graph of a Reinforcement Learning environment with a learned feature representation, from raw pixels. We test our approach on a set of benchmarks and show a success of the proposed approach: it outputs an interpretable causal graph and allows to understand the environment. Using this approach could yield increased interpretability of Reinforcement Learning agents if used together with already existing methods requiring a causal graph as an input.

To obtain the meaning of the features learned by our system, existing interpretability techniques based on feature attribution and analyzing neural network circuits [48] can be used.

The main drawback of the system is a higher train the time to fit a model of the environment, and then adjust the decoder to "sparsify" the model. Better (more adaptive) training procedures could speed up the convergence and increase the range of applicability of our approach.

7.1 Future work

A natural extension is to consider more complex environments, and multiple time-step dependencies, either by stacking several subsequent observations together, or by using recurrent neural networks with structured sparsity, or by using an attention mechanism in the time axis [96]. It would be interesting to apply our approach to natural languages and other challenging not-yet-fully-understood time series and environments such as the protein folding problem. This could yield novel insights into the latent nature of these problems.

Combining the abstract causal concept discovery with doing interventions using policies [126] using an RL agent could yield more structured exploration via *interventions* on the causal graph [54, 46, 98, 27], and, thus, faster convergence time.

To reduce training time, it might be beneficial to contrastive losses [119, 75, 93] instead of the reconstructor. However, this did not work as well as the reconstruction loss in the observation space in this project.

7.1.1 Sparse causal models and the problem of consciousness

The connection to the problem of consciousness stems comes from two directions. First, this work can be seen as an implementation of the [13] hypothesis. Indeed, we are learning the most sparse model in the latent space, along with learning the representations. The paper explicitly defines latent feature model simplicity as one of the components of the consciousness prior [13].

The fact that our approach works (albeit for simple environments) proves the merit of this hypothesis and shows that it could be used to explain consciousness: humans indeed can be learning abstract representations (at least partially by the virtue of) learning a space in which the dynamics becomes simple. For example, it is beneficial to learn a variable "time", because this single variable allows to predict if the sun rises, if the metro is open, or if a predator is likely to be sleeping.

Our learned representations can be used to ground the agent in natural language. Indeed, if we already have learned that there are several abstract concepts in the game such as "number of keys collected" or the "health of the player", it would suffice to only give two sentences associated with the player collecting a key or food. In contrast, a vanilla CNN given the same sentences at the same time would likely associate them with raw pixel state, and a policy network would likely associate them with the current action taken.

In this way, we shed (a bit of) light onto the easy problem of consciousness: the abstract concepts can come *before* language, in an unsupervised way, and the first words of language are simply mapped to already existing concepts using Hebbian learning (word and the stimulus occur together). Next, this process could be bootstrapped, since presence of words becomes new variables in the causal model, and the process is repeated, with a more powerful model and a more powerful set of words learned. Asking questions like "what is X?" can be seen as performing interventions on the causal graph: we know that asking "what is X" is likely to result in novel data (a learned way to optimize for an objective), and an X is a novel variable which is still largely unused. Therefore, the agent intervenes setting do(ask 'what is X') and samples data from a novel distribution.

While the grounding technique is known for a long time [57], my work provides a proof that abstract concepts can be learned before language from raw pixels, just by optimizing for the inner model simplicity – a simple and beautiful enough principle to be innate to the brain [7]. Other projects are usually involving more sophisticated (and less simple) priors, such that the action only increments one component, or that the dynamics in the latent space is linear, or that the environment is a $2D$ world.

Secondly, I express a hypothesis that a learned representation is crucial to the Integrated Information Theory, or any other causal theory of consciousness [117]. First, the theory does not seem to specify at which level of abstraction it should be applied – at the level of atoms, or at the level of molecules? Or, maybe, at the level of individual electrical components of a computer that we examine? Specifically, it is unclear what should be the "nodes", "factors" or "variables" in the model, the dependencies between which we are to examine to compute the φ. This leads to the value of φ potentially being arbitrarily high, if a bijective transformation is applied before the analysis [37].

If we declare that we need to first apply a transformation in which the graph becomes most sparse (which intuitively is connected to the value of φ: a sparse graph will likely have less edges [111, 67], or even becomes two independent components instead). Next, we compute φ. This seems to resolve the issue raised in [37], albeit, in a bit of an ad-hoc fashion. More importantly, this resolves the ambiguity of the level of abstraction the theory should be applied: if we apply a decoder first and optimize for sparsity (or, maybe, low φ), the raw wave-function can be mapped into the state of molecules – and the two representations are (somewhat) bijective.

Because of this connection between φ and sparsity, our approach can be seen as approximately computing the φ for the *environment*, in other words, computing how conscious it is. With this in mind, the VectorIncrement environment, for example, is not conscious: its causal graph is a union of two connected components (the "done" and the "time" is not connected to everything else), but can be seen as two "conscious" components. In contrast, the KeyChest environment's graph has only one connected component, and, thus, potentially has a non-0 φ as a whole.

7.1.2 Sparse causal models and the AIXI framework

A quick overview of the AIXI framework is given in Appendix A. First, we note that while in theory, penalizing for model's complexity is inevitable (see section A.1), for practical cases the total model complexity is limited by the hardware, and, therefore, we are free to favor more complex models over simpler ones.

Having the simplicity prior arguably helps the convergence speed. It is an interesting direction for future work to potentially prove a bound similar to the Solomonoff's bound, but for computable environments. Indeed, we are using the simplicity prior, and our algorithm can be seen as doing approximate Bayesian inference.

"Vanilla" neural networks arguably do not have such a simplicity prior "built-in", as they are likely to learn spurious correlations [62]. Therefore, adding the simplicity prior explicitly (in terms of structural sparsity) could give interesting theoretical results.

It is left for future work to test if using our approach leads to better sample-efficiency of learning dynamics model of RL environments. Our framework can be seen as another approximation

[124] to the true uncomputable AIXI framework, in a sense that we are defining a proxy for the model complexity in terms of its structure as $K(M) = \|B\|_0$.

7.1.3 Other future directions

The prior to have a simple model could make the agent require less data to train: since it expects the underlying dynamics to be simple, it will converge to the true model faster, because a more complex model (that a regular agent is likely to take as a first guess) will be discarded.

Next, having a disentangled representation with independent mechanisms allows to reuse components in the model [12, 122, 34, 49], and learn it even faster. Many environments share the same dynamics – for example, navigating in a 2-dimensional maze with 4 actions (up, down, left, right). The learned graph would be the same for all such environments (up and down actions affecting the vertical coordinate variable, and the left-right actions the horizontal one). We can store the commonly-occurring patterns like this one (another example is the velocity of an object being mirrored when colliding with an obstacle), we could represent every environment as a number of "stock" components, or recurrent independent mechanisms (coordinate variables, colliding objects) with parameters, with a small number of custom ones. Such an approach would lead to faster learning, as the agent would "guess" a component much faster, compared to learning it from scratch.

Finally, the method can be applied to challenging environments interesting in their own right. For example, the AlphaFold [4] algorithm was applied to the protein folding problem. The model of protein folding that the agent learns is (a prori) not interpretable. However, using out method, we could obtain a sparse causal graph representing the dynamics of protein folding, which could be useful, because the problem becomes simpler. Indeed, our approach could discover interesting regularities in the way proteins fold, in terms of the final state depending only on the few variables in the initial state – automating the process of discovery of the scientific laws of the natural world.

Appendix A

Optional background: Solomonoff induction and the AIXI framework

In this section, we consider the theoretical Solomonoff induction framework, which gives a convergence bound for Supervised learning for sequence prediction, along with the AIXI framework [59, 80], which extends this approach to RL. The frameworks described in this section do not provide direct practical results, as they are uncomputable [59] (there is no algorithm implementing them exactly). There exist computable approximations [124].

A.1 Solomonoff induction

We consider the class of all computable (having a probabilistic Turing machine that successively prints the string onto the tape) infinite distributions over the sequences of binary strings $\{\mu\}$. We assume that the true distribution is a member of this class. We are given symbols $x_t \in \{0, 1\}$ one-by-one, and the task is to predict the next symbol given the previous ones. We apply the Bayesian approach: given a prior $\xi(\mu) \in [0, 1]$ over the sequences μ, we would like to find the posterior $\xi|(x_1, ..., x_t)$ over the sequences given the observed data $x_1, ..., x_t$. We denote as $K(\mu)$ the length of the shortest program that generates μ[1].

It can be shown [56, 118] that any possible prior has to decay as $2^{-\alpha K(\mu)+\beta}$ as $K(\mu) \to \infty$. Indeed, while the prior distribution has to sum up to 1, the number of programs with length n increases exponentially.

Solomonoff has shown in 1964 that with a prior defined as $\xi(\mu) \sim 2^{-K(\mu)}$, Bayesian inference results in a fast convergence to the true distribution. The squared expected distance in the space of all sequences, between the posterior $\xi|(x_1, ..., x_{t-1})$ given the data from the first $t-1$ time-steps

[1]See [59] for a fully formal definition

$x_{<t} = (x_1, ..., x_t)$, and the true distribution μ is bounded via the complexity (See [59], Equation 18):

$$\sum_{t=1}^{\infty} \sum_{x_{<t}} \mu(x_{<t}) \left(\xi(x_t = 0, x_{<t}) - \mu(x_t = 0, x_{<t})\right)^2 \leq \text{const} \cdot K(\mu)$$

Roughly speaking, the algorithm will make approximately $K(\mu)$ mistakes, when learning a sequence of complexity $K(\mu)$.

A.2 The AIXI framework

AIXI [59] is an extension of Solomonoff induction to the problem of Reinforcement Learning. We set $x_t = (o_t, r_t)$ – the observation and the reward given by the environment. The action is selected greedily to optimize for the expected reward given the posterior (see Equation 23 in [59]):

$$a_{t+1} = \arg\max_a \mathbb{E}_{x_{t+1}, x_{t+2}, \ldots \sim \xi | (x_1, \ldots, x_t)} V_{\geq t}$$

Roughly speaking, this equation means that we first select the simplest environments (because the prior ξ penalizes for sequence's complexity), that fit the observed data well. Having such a model, we simulate the future using tree search, and select an action optimizing for the discounted value.

While the AIXI agent has certain optimality guarantees, they are weak [81], due to the hardness of exploration in RL. Approaches [82] exist to deal with this issue.

Bibliography

[1] A. Abid, M. F. Balin, and J. Zou. Concrete autoencoders for differentiable feature selection and reconstruction. *arXiv*, 2019.

[2] J. Abramson, A. Ahuja, A. Brussee, F. Carnevale, M. Cassin, S. Clark, A. Dudzik, P. Georgiev, A. Guy, T. Harley, F. Hill, A. Hung, Z. Kenton, J. Landon, T. Lillicrap, K. Mathewson, A. Muldal, A. Santoro, N. Savinov, V. Varma, G. Wayne, N. Wong, C. Yan, and R. Zhu. Imitating Interactive Intelligence. pages 1–96, 2020.

[3] A. A. Adegbege, S. Member, and M. Y. Kim. Saddle-Point Convergence of Constrained Primal-Dual Dynamics. 5(4):1357–1362, 2021.

[4] M. AlQuraishi. Alphafold at casp13. *Bioinformatics*, 35(22):4862–4865, 2019.

[5] E. Andriyash, A. Vahdat, and B. Macready. Improved gradient-based optimization over discrete distributions. *arXiv preprint arXiv:1810.00116*, 2018.

[6] A. Ayoub, Z. Jia, C. Szepesv, and W. Lin. Model-Based Reinforcement Learning with Value-Targeted Regression. 2020.

[7] B. Babadi and H. Sompolinsky. Sparseness and Expansion in Sensory Representations. *Neuron*, 83(5):1213–1226, sep 2014.

[8] A. Bakhtin, L. van der Maaten, J. Johnson, L. Gustafson, and R. Girshick. PHYRE: A new benchmark for physical reasoning. *arXiv*, (NeurIPS), 2019.

[9] P. Barceló, M. Monet, J. Pérez, and B. Subercaseaux. Model Interpretability through the Lens of Computational Complexity. (NeurIPS):1–12, 2020.

[10] L. Barnett and A. K. Seth. Granger causality for state-space models. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 91(4):1–5, 2015.

[11] P. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *Advances in Neural Information Processing Systems*, pages 4509–4517, 2016.

[12] Y. Bengio. Evolving Culture vs Local Minima. 2006:1–28, 2012.

[13] Y. Bengio. The Consciousness Prior. pages 1–7, 2017.

[14] Y. Bengio, O. Delalleau, N. L. Roux, H. Larochelle, P. Lamblin, D. Popovici, A. Courville, C. Simard, and J. Louradour. Deep Architectures for Baby AI Summary for Day 1. 2007.

[15] Y. Bengio, N. Léonard, and A. Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. Technical report, 2013.

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

[17] P. Brouillard, S. Lachapelle, A. Lacoste, S. Lacoste-Julien, and A. Drouin. Differentiable causal discovery from interventional data. *arXiv preprint arXiv:2007.01754*, 2020.

[18] M. Cha, Y. Gwon, and H.-T. Kung. Multimodal Sparse Representation Learning and Applications. *Journal of AI Humanities*, 2:39–64, 2018.

[19] K. Chalupka, F. Eberhardt, and P. Perona. Causal feature learning: an overview. *Behaviormetrika*, 44(1):137–164, 2017.

[20] K. Chalupka, P. Perona, and F. Eberhardt. Visual causal feature learning. *Uncertainty in Artificial Intelligence - Proceedings of the 31st Conference, UAI 2015*, pages 181–190, 2015.

[21] S. Chari, D. M. Gruen, O. Seneviratne, and D. L. McGuinness. Foundations of explainable knowledge-enabled systems. *arXiv*, (March), 2020.

[22] T. Claassen, J. M. Mooij, and T. Heskes. Learning sparse causal models is not NP-hard. *Uncertainty in Artificial Intelligence - Proceedings of the 29th Conference, UAI 2013*, pages 172–181, 2013.

[23] D. Corneil, W. Gerstner, and J. Brea. Efficient model-based deep reinforcement learning with variational state tabulation. *arXiv*, 2018.

[24] A. Cortese, A. Yamamoto, M. Hashemzadeh, P. Sepulveda, M. Kawato, and B. D. Martino. Value Shapes Abstraction During Learning. *PsyArXiv*, pages 1–36, 2020.

[25] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho. Discovering symbolic models from deep learning with inductive biases. *arXiv*, (NeurIPS):1–25, 2020.

[26] M. Das Gupta. Non-Convex P-norm Projection for Robust Sparsity *. Technical report, 2013.

[27] I. Dasgupta, J. Wang, S. Chiappa, J. Mitrovic, P. Ortega, D. Raposo, E. Hughes, P. Battaglia, M. Botvinick, and Z. Kurth-Nelson. CAUSAL REASONING FROM META REINFORCEMENT LEARNING. Technical report, 2019.

[28] T. De Bruin, J. Kober, K. Tuyls, and R. Babuska. Integrating State Representation Learning into Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, 2018.

[29] P. de Haan, D. Jayaraman, and S. Levine. Causal Confusion in Imitation Learning. (NeurIPS):1–17, 2019.

[30] T. Deleu and Y. Bengio. Structured Sparsity Inducing Adaptive Optimizers for Deep Learning. 2015.

[31] F. Deng, J. Ren, and F. Chen. Abstraction Learning. Technical report, 2018.

[32] M. DesJardins. PAGODA: a model for autonomous learning in probabilistic domains. *AI Magazine*, 14(1):75–76, 1993.

[33] M. DesJardins. Representing and Reasoning With Probabilistic Knowledge : A Bayesian Approach. 1993.

[34] A. Didolkar, A. Goyal, N. Rosemary, K. Charles, B. Philippe, N. Heess, M. Mozer, and Y. Bengio. Neural Production Systems. 2021.

[35] S. Dieleman, C. Nash, J. Engel, and K. Simonyan. Variable-rate discrete representation learning. 2021.

[36] L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear independent components estimation. *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, 1(2):1–13, 2015.

[37] A. Doerig, A. Schurger, K. Hess, and M. H. Herzog. The unfolding argument: Why IIT and other causal structure theories cannot explain consciousness. *Consciousness and Cognition*, 72(April):49–59, 2019.

[38] U. Ehsan, B. Harrison, L. Chan, and M. O. Riedl. Rationalization: A Neural Machine Translation Approach to Generating Natural Language Explanations. *AIES 2018 - Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 81–87, 2018.

[39] R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July:5598–5602, 2018.

[40] T. Everitt, P. A. Ortega, E. Barnes, and S. Legg. Understanding Agent Incentives using Causal Influence Diagrams. Part I: Single Action Settings. 2019.

[41] K. Fallah, A. A. Willats, N. Liu, and C. J. Rozell. Learning sparse codes from compressed representations with biologically plausible local wiring constraints. (NeurIPS), 2020.

[42] T. Fiez and L. J. Ratliff. Gradient descent-ascent provably converges to strict local minmax Equilibria with a finite timescale separation. *arXiv*, pages 1–70, 2020.

[43] V. Fortuin, M. Huser, F. Locatello, H. Strathmann, and G. Rätsch. SOM-VAE: Interpretable discrete representation learning on time series. *arXiv*, pages 1–18, 2018.

[44] L. Franceschi, M. Niepert, M. Pontil, and X. He. Learning discrete structures for graph neural networks. *arXiv*, 2019.

[45] V. Francois-Lavet, Y. Bengio, D. Precup, J. Pineau, V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau. Combined Reinforcement Learning via Abstract Representations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:3582–3589, sep 2019.

[46] M. Frisch and M. Frisch. Causation and intervention. *Causal Reasoning in Physics*, pages 77–110, 2014.

[47] F. Fu and Q. Zhou. Learning Sparse Causal Gaussian Networks With Experimental Intervention : Regularization and Coordinate Descent With Experimental Intervention : Regularization. 1459, 2013.

[48] G. Goh, N. Cammarata, C. Voss, S. Carter, M. Petrov, L. Schubert, A. Radford, and C. Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

[49] R. Gómez, R. Bootzin, and L. Nadel. HALMA: humanlike abstraction learning meets affordance in rapid problem solving. page 358, 2021.

[50] A. Goyal. Inductive Biases for Deep Learning of Higher-Level Cognition. pages 1–42, 2021.

[51] R. Guo, L. U. Cheng, P. R. Hahn, H. Liu, L. Cheng, and J. Li. A Survey of Learning Causality with Data: Problems and Methods. 2020.

[52] L. Hahne, T. Lüddecke, F. Wörgötter, and D. Kappel. Attention on abstract visual reasoning. *arXiv preprint arXiv:1911.05990*, 2019.

[53] R. W. Hamming. The Unreasonable Effectiveness of Mathematics. *The American Mathematical Monthly*, 87(2):81, 1980.

[54] Y. B. He and Z. Geng. Active learning of causal networks with intervention experiments and optimal designs. *Journal of Machine Learning Research*, 9:2523–2547, 2008.

[55] J. Hilton, N. Cammarata, S. Carter, G. Goh, and C. Olah. Understanding rl vision. *Distill*, 5(11):e29, 2020.

[56] L. N. Hoang. *The Equation of Knowledge.* Chapman and Hall/CRC, 1st edition, 2020.

[57] D. Hofstadter and M. Mitchell. The copycat project: A model of mental fluidity and analogy-making. *Advances in connectionist and neural computation theory*, 2(31-112):29–30, 1994.

[58] D. Y.-t. Hui, M. Chevalier-boisvert, D. Bahdanau, and Y. Bengio. BabyAI 1.1. 1:1–9, 2020.

[59] M. Hutter. A gentle introduction to the universal algorithmic agent AIXI. *Artificial General Intelligence*, 2003.

[60] A. Hyvärinen, H. Sasaki, and R. E. Turner. Nonlinear ICA Using Auxiliary Variables and Generalized Contrastive Learning. Technical report, 2019.

[61] M. Ibrahim. Addressing the topological defects of disentanglement via distributed operators. pages 1–46, 2020.

[62] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Madry. Adversarial examples are not bugs, they are features. *arXiv preprint arXiv:1905.02175*, 2019.

[63] E. Jang, G. Brain, S. Gu, and B. Poole. CATEGORICAL REPARAMETERIZATION WITH GUMBEL-SOFTMAX. Technical report, 2017.

[64] K. Javed. Learning Online-Aware Representations using Neural Networks. 2020.

[65] K. Javed and M. White. Meta-Learning Representations for Continual Learning. *arXiv*, (NeurIPS):1–15, 2019.

[66] K. Javed, M. White, and Y. Bengio MILA. Learning Causal Models Online. Technical report, 2020.

[67] M. E. Johnson. Neural Annealing: Toward a Neural Theory of Everything. Technical report, 2020.

[68] M. J. Johnson, D. Duvenaud, A. B. Wiltschko, S. R. Datta, and R. P. Adams. Composing graphical models with neural networks for structured representations and fast inference. *Advances in Neural Information Processing Systems*, pages 2954–2962, 2016.

[69] A. Jonsson and A. Barto. Causal graph based decomposition of factored MDPs. *Journal of Machine Learning Research*, 7:2259–2301, 2006.

[70] Ł. Kaiser, M. Babaeizadeh, P. Miłos, B. Osiński, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski. Model based reinforcement learning for atari. *arXiv*, 2019.

[71] D. Kalainathan. Generative Neural networks to infer Causal Mechanisms: Algorithms and applications. Technical report, 2020.

[72] D. Kalainathan, O. Goudet, I. Guyon, D. Lopez-Paz, and M. Sebag. Structural agnostic modeling: Adversarial learning of causal graphs. *arXiv*, pages 1–49, 2018.

[73] P. Kamalaruban. Environment Shaping in Reinforcement Learning using State Abstraction. Technical report, 2020.

[74] N. R. Ke, O. Bilaniuk, A. Goyal, S. Bauer, H. Larochelle, C. Pal, and Y. Bengio. Learning Neural Causal Models from Unknown Interventions. oct 2019.

[75] T. Kipf, E. V. D. Pol, M. Welling, E. Van Der Pol, and M. Welling. Contrastive learning of structured world models. *arXiv*, pages 1–21, 2019.

[76] T. Kurutach, S. Russell, and P. Abbeel. Discrete Predictive Representation for Long-Horizon Planning. *ICLR openrevivew*, pages 1–10, 2021.

[77] S. Lahlou. Baby AI: a platform to study the sample-efficiency of grounded language learning. pages 1–19, 2019.

[78] A. Lamb, M. Mozer, and P. Beaudoin. Neural Function Modules with Sparse Arguments : A Dynamic Approach to Integrating Information across Layers. 2020.

[79] C. Learning and Y. Need. Do Generative Models Know Disentanglement? Contrastive Learning is All You Need. 2014.

[80] S. Legg. *Machine super intelligence*. PhD thesis, Università della Svizzera italiana, 2008.

[81] J. Leike and M. Hutter. Bad universal priors and notions of optimality. In *Conference on Learning Theory*, pages 1244–1259. PMLR, 2015.

[82] J. Leike, T. Lattimore, L. Orseau, and M. Hutter. Thompson sampling is asymptotically optimal in general environments. *arXiv preprint arXiv:1602.07905*, 2016.

[83] T. Lillicrap and J. Ba. Mastering Atari with discrete world models. pages 1–24, 2019.

[84] Z. C. Lipton. The Mythos of Model Interpretability. Technical report, 2017.

[85] V. Liu, R. Kumaraswamy, L. Le, and M. White. The utility of sparse representations for control in reinforcement learning. *arXiv*, 2018.

[86] C. Louizos, M. Welling, and D. Kingma. Learning sparse neural networks through l0 regularization. (1):1–5, 2019.

[87] C. Luneau, N. Macris, and J. Barbier. Information theoretic limits of learning a sparse rule. *arXiv*, (NeurIPS), 2020.

[88] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pages 1–20, 2017.

[89] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere. Explainable Reinforcement Learning Through a Causal Lens. may 2019.

[90] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *7th International Conference on Learning Representations, ICLR 2019*, pages 1–28, 2019.

[91] G.-m. M. Martic, T. Genewein, and T. Mcgrath. Causal Analysis of Agent Behavior for AI Safety. 2020.

[92] M. Mitchell. Conceptual Abstraction and Analogy in Artificial Intelligence. pages 7–7, 2020.

[93] S. Mohammadi, A. K. Uhrenholt, and B. S. Jensen. Odd-One-Out Representation Learning. 2020.

[94] J. M. Mooij, S. Magliacane, and T. Claassen. Joint Causal Inference from Multiple Contexts. 2016.

[95] S. Nair, Y. Zhu, S. Savarese, and L. Fei-Fei. Causal Induction from Visual Observations for Goal Directed Tasks. pages 1–13, 2019.

[96] M. Nauta, D. Bucur, and C. Seifert. Causal Discovery with Attention-Based Convolutional Neural Networks. *Machine Learning and Knowledge Extraction*, 1(1):312–340, 2019.

[97] I. Ng, Z. Fang, S. Zhu, and Z. Chen. Masked gradient-based causal structure learning. *arXiv*, 2019.

[98] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven Exploration by Self-supervised Prediction. 2017.

[99] J. Pearl. Causality: models, reasoning and inference. 2020.

[100] Y. Pei and X. Hou. Learning Representations in Reinforcement Learning: An Information Bottleneck Approach. *arXiv*, 2019.

[101] G. Petros, P. Aggelos, and C. Yannis. NEURAL DISCRETE ABSTRACTION OF HIGH-DIMENSIONAL SPACES : A CASE STUDY IN REINFORCEMENT LEARNING Petros Giannakopoulos Yannis Cotronis National and Kapodistrian University of Athens University of Piraeus. pages 1517–1521, 2020.

[102] R. L. Priol and S. Lacoste-julien. An Analysis of the Adaptation Speed of Causal Models. 130, 2021.

[103] J. Rafati and D. C. Noelle. Learning sparse representations in reinforcement learning. *arXiv*, pages 1–20, 2019.

[104] A. Rajeswaran, I. Mordatch, and V. Kumar. A game theoretic framework for model based reinforcement learning. *arXiv*, 2020.

[105] D. J. Rezende, I. Danihelka, G. Papamakarios, N. R. Ke, R. Jiang, T. Weber, K. Gregor, H. Merzic, F. Viola, J. Wang, et al. Causally correct partial models for reinforcement learning. *arXiv preprint arXiv:2002.02836*, 2020.

[106] J. Runge, S. Bathiany, E. Bollt, G. Camps-Valls, D. Coumou, E. Deyle, C. Glymour, M. Kretschmer, M. D. Mahecha, J. Muñoz-Marí, E. H. van Nes, J. Peters, R. Quax, M. Reichstein, M. Scheffer, B. Schölkopf, P. Spirtes, G. Sugihara, J. Sun, K. Zhang, and J. Zscheischler. Inferring causation from time series in Earth system sciences. *Nature Communications*, 10(1):1–13, 2019.

[107] J. Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5499 LNAI(April 2009):48–76, 2009.

[108] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, and N. Kalchbrenner. Towards Causal Representation Learning. pages 1–24, 2021.

[109] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *arXiv*, nov 2019.

[110] M. Seeger, S. Gerwinn, and M. Bethge. Bayesian inference for sparse generalized linear models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4701 LNAI:298–309, 2007.

[111] A. K. Seth, A. B. Barrett, and L. Barnett. Causal density and integrated information as measures of conscious level. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1952):3748–3767, 2011.

[112] V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit Neural Representations with Periodic Activation Functions. jun 2020.

[113] J. Tang, W. Hu, X. Gao, and Z. Guo. JOINT GRAPH AND FEATURE LEARNING IN GRAPH CONVOLUTIONAL NEURAL NETWORKS A PREPRINT. Technical report, 2019.

[114] A. Tank, I. Cover, N. J. Foti, A. Shojaie, and E. B. Fox. An Interpretable and Sparse Neural Network Model for Nonlinear Granger Causality Discovery. (Nips), 2017.

[115] A. Tank, I. Covert, N. J. Foti, A. Shojaie, and E. B. Fox. Neural granger causality for nonlinear time series. *arXiv*, pages 1–14, 2018.

[116] V. Thomas, E. Bengio, W. Fedus, J. Pondard, P. Beaudoin, H. Larochelle, J. Pineau, D. Precup, and Y. Bengio. Disentangling the independently controllable factors of variation by interacting with the world. pages 1–9, 2018.

[117] G. Tononi, M. Boly, M. Massimini, and C. Koch. Integrated information theory: From consciousness to its physical substrate. *Nature Reviews Neuroscience*, 17(7):450–461, 2016.

[118] K. Ullrich. UvA-DARE (Digital Academic Repository) A coding perspective on deep latent variable models Ullrich, K. 2021.

[119] A. Van Den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv*, 2018.

[120] A. Van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *arXiv*, (Nips), 2017.

[121] S. van Steenkiste. Learning Structured Neural Representations for Visual Reasoning Tasks Sjoerd van Steenkiste under the supervision of. *2020*, (November), 2020.

[122] R. Veerapaneni, J. D. Co-Reyes, M. Chang, M. Janner, C. Finn, J. Wu, J. B. Tenenbaum, and S. Levine. Entity abstraction in visual model-based reinforcement learning. *arXiv*, (CoRL):1–18, 2019.

[123] P. Veličković, L. Buesing, M. C. Overlan, R. Pascanu, O. Vinyals, and C. Blundell. Pointer Graph Networks. *arXiv*, 2020.

[124] J. Veness, K. Siong Ng, M. Hutter, and D. Silver. Reinforcement learning via AIXI approximation. *Proceedings of the National Conference on Artificial Intelligence*, 1(3):605–611, 2010.

[125] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48(1-3):165–187, 2002.

[126] S. Volodin, N. Wichers, and J. Nixon. Resolving spurious correlations in causal models of environments via interventions. *arXiv*, 2020.

[127] H. Wang, Y. Yang, Z. Bu, and W. J. Su. The Complete Lasso Tradeoff Diagram. *arXiv*, (NeurIPS), 2020.

[128] A. J. Weinstein. INFERENCE AND LEARNING IN HIGH-DIMENSIONAL SPACES. Technical report, 2013.

[129] G. Weiß and C. Meine. END-TO-END INPUT SELECTION FOR DEEP NEURAL NETWORKS. *Dbw*, 70(1):83–87, 2011.

[130] J. C. Wong. Computational Causal Inference. 2020.

[131] M. Yang, F. Liu, Z. Chen, X. Shen, J. Hao, and J. Wang. CausalVAE: Disentangled Representation Learning via Neural Structural Causal Models. pages 1–22, 2020.

[132] D. Yarats, R. Fergus, A. Lazaric, and L. Pinto. Reinforcement Learning with Prototypical Representations. 2016.

[133] J. Ying, D. P. Palomar, D. Analytics, C. W. Bay, and H. Kong. Nonconvex Sparse Graph Learning under Laplacian Constrained Graphical Model. (NeurIPS):1–13, 2020.

[134] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[135] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph Neural Networks: A Review of Methods and Applications. *arXiv*, pages 1–22, 2018.

[136] X. Zhu, X. Li, S. Zhang, C. Ju, and X. Wu. Robust Joint Graph Sparse Coding for Unsupervised Spectral Feature Selection. *IEEE Transactions on Neural Networks and Learning Systems*, 28(6):1263–1275, 2017.