UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Youssef Sherif Mansour Mohamed

Motivating Reinforcement Learning Agents to Control their Environment

Master's Thesis (30 ECTS)

Supervisors: Oriol Corcoll, Raul Vicente

Tartu 2021

Motivating Reinforcement Learning Agents to Control their Environment

Abstract: Exploration lies at the heart of every Reinforcement Learning problem. Sparse environments rarely reward agents, making them extremely hard to explore. Behavioral biases attempt to solve the problem by intrinsically motivating the agent to exhibit certain behaviors. Understanding the controllable aspects of an environment is a popular behavioral bias implemented using intrinsic motivators. It helped many models to achieve state-of-the-art results. However, current methods rely on inverse dynamics learning to identify controllable aspects. Inverse dynamics learning has drawbacks limiting the agent's ability to model the controllable objects. We highlight some of these drawbacks and propose an alternate approach to learning controllable aspects of the environment.

This thesis introduces Controlled Effects Network (CEN), a self-supervised method for learning controllable aspects in a Reinforcement Learning environment. CEN uses causal concepts of blame to identify controllable objects. We integrate CEN in an intrinsic motivation module which improves the exploration behavior of reinforcement learning agents. Agents using CEN outperform inverse dynamics agents in both efficiency learning and the max score achieved in Sparse Environments. CEN-based motivator encourages the agent to do more interactions with controllable objects in an environment. Hence, the agent is more likely to reach events that trigger an extrinsic reward from the environment.

We compare agents using CEN-based intrinsic motivators and others using Inverse dynamics-based motivators. To this end, we create multiple sparse environments to test the exploration behavior of both agents. In an empty grid, CEN agents exhibit uniform exploration visiting numerous grid cells, while Inverse agents tend to stick to corners and walls. In sparse Clusters, CEN agents achieve a max score of 5 while Inverse agents manage to get only 1. Moreover, CEN agents learn to solve the Clusters environment more efficiently, requiring fewer environment steps. We open source our implementation of CEN, the sparse environments, and the Never Give Up (NGU) reinforcement learning agent to ease future research on controllability and exploration.

Keywords: Reinforcement Learning, Causality, Exploration, Deep Neural Networks

CERCS: P176 Artificial intelligence

Õppimisagentide motiveerimine oma keskkonda kontrollima

Lühikokkuvõte: Uurimine on iga tugevdamisõppe probleemi keskmes. Hõred keskkonnad premeerivad agente harva, mistõttu on nende uurimine äärmiselt raske. Käitumuslikud eelarvamused püüavad probleemi lahendada, motiveerides agenti teatud käitumisviisidele. Keskkonna kontrollitavate aspektide mõistmine on populaarne käitumuslik eelarvamus, mida rakendatakse sisemiste motivaatorite abil. See aitas paljudel mudelitel saavutada tipptasemel tulemusi. Praegused meetodid tuginevad aga kontrollitavate aspektide tuvastamiseks pöörddünaamika õppimisele. Pöörddünaamika õppimisel on puudusi, mis piiravad agendi võimet modelleerida juhitavaid objekte. Toome välja mõned neist puudustest ja pakume välja alternatiivse lähenemisviisi keskkonna kontrollitavate aspektide õppimiseks.

See lõputöö tutvustab kontrollitud efektide võrgustikku (CEN), pooljuhitud meetodit kontrollitavate aspektide õppimiseks tugevdamisõppe keskkonnas. CEN kasutab kontrollitavate objektide tuvastamiseks põhjuslikke süü kontseptsioone. Integreerime CEN-i sisemise motivatsiooni moodulisse, mis parandab tugevdavate õppeagentide uurimiskäitumist. CEN-i kasutavad agendid ületavad pöörddünaamika agente nii tõhususe õppimise kui ka hõredates keskkondades saavutatud maksimaalse skoori osas. CEN-põhine motivaator julgustab agenti rohkem suhtlema keskkonnas kontrollitavate objektidega. Seetõttu jõuab agent tõenäolisemalt sündmusteni, mis käivitavad keskkonnast välise tasu.

Võrdleme agente, kes kasutavad CEN-põhiseid sisemisi motivaatoreid ja teisi, kes kasutavad pöörddünaamikal põhinevaid motivaatoreid. Selleks loome mõlema agendi uurimiskäitumise testimiseks mitu hõredat keskkonda. Tühjas ruudustikus on CEN-i agendid ühtsed, külastades paljusid ruudustiku rakke, samas kui pöördagendid kipuvad kleepuma nurkadele ja seintele. Hõredates klastrites saavutavad CEN-agendid maksimaalse tulemuse 5, samas kui pöördagendid saavad ainult 1. Lisaks õpivad CEN-agendid klastrite keskkonda tõhusamalt lahendama, nõudes vähem keskkonnasamme. Avame lähtekoodiga CEN-i juurutamise, hõredad keskkonnad ja NGU (Never Give Up) õppeagensi, et hõlbustada tulevast kontrollitavuse ja uurimise uurimist.

Võtmesõnad: Tugevdusõpe, Põhjuslikkus, Uurimine, Sügavad närvivõrgud

CERCS: P176 Tehisintellekt

Acknowledgments

All my gratitude belongs to Allah for all his blessings.

I would like to thank my supervisors, Oriol and Raul, for their guidance, support, and advice throughout this project. I learned a lot from Oriol's technical expertise. He has been extremely helpful and was always keen to teach me. Raul helped me become a better researcher through our rich discussions. He has always been incredibly supportive, providing me with many opportunities throughout my study. I will always be thankful to both of you.

I express my gratitude to all members of the Computational Neuroscience Lab. The discussions and the reading group helped me develop my understanding and appreciation of the field of computational neuroscience. I extend my gratitude to my professors and friends at the University of Tartu for the beautiful experience.

To my parents, brothers, and lovely sister, thank you for your never-ending love and support.

I dedicate this work to the memory of my beloved late grandmother and Prof. Sherif Sakr.

Youssef Mohamed
August, 2022

Contents

1 Introduction

Reinforcement Learning Agents learn by interacting with their environments to maximize a reward signal. Some environments are sparse with their reward signal, rewarding the agent only after it successfully completes a long sequence of events. For such environments, an agent has to *smartly* explore them. Random exploration has been widely adopted due to its simplicity. It works well enough with dense environments that reward the agent for many events. These rewards help the agent navigate the environment in order to maximize the total reward. However, sparse environments rarely reward the agent. Thus it is harder for an agent, randomly exploring the environment, to learn reward-rich states. This problem is magnified if the agent has a short lifespan since it might not get any reward during its life and consequently learn nothing about the environment.

One might try to make the environment's reward abundant. Although this would work nicely with environments that we can change, most practical environments do not allow for such interventions. On the other hand, the agent is usually under our full control. So, we can introduce behavioral biases to the exploration strategy instead of adopting random exploration. Many strategies imitate humans explorative behaviors such as boredom or control [BVB12, PAED17, BESK18, CGM$^+$18, BSV$^+$20]. Some strategies contributed to achieving state of the art results on hard exploration environments in the Atari benchmark [BSV$^+$20, CGM$^+$18].

These exploration strategies are usually implemented as intrinsic motivators which provide the agent with a reward based on the state of the environment. Hence, they convert sparse environments into dense ones. The intrinsic reward aims to encourage the agent to explore novel states, increasing the likelihood of the agent stumbling upon the environment's external reward.

Boredom is typically used as a behavioral bias for many intrinsic motivators. The agent gets bored from visiting the same state over and over. It gets discouraged to revisit the same state in favor of exploring new ones. The agent usually can not access the internal state of the environment; hence it uses observations as a state approximation. Many practical environments are highly complex with innumerable observations; thus, directly comparing observations can be infeasible. Most intrinsic motivators embed the observations into a lower-dimensional feature representation by dropping redundant and unrelated information. The lower-dimensional representations are then used to approximate the environment's states.

Inverse dynamics prediction is one of the most popular approaches to embed observations into feature representations. It works by training a function approximator, *i.e.* a neural network, to predict the action performed by the agent to transition between two consecutive observations. The latent representations produced by the function approximator should encompass all aspects of the environment related to the action performed. Controlled changes are the most important of these aspects. For example, how agent's avatar location changes is directly related to action performed. Hence, most action prediction methods model the agent's location and ignore other aspects of the environment.

Controlled aspects of an environment besides the agent's location are equally important. Consider the Clusters environment [Bam21] in Fig. 1, the agent has to move all the solid boxes into similar colored hollow blocks in order to solve the environment and get a reward. Intrinsic motivators based on inverse action prediction would model only the avatar and collapse all the

boxes, regardless of color, into a similar representation. Hence, the intrinsic reward module will feel bored with all the boxes after moving the first box discouraging the agent from interacting with other boxes.

In contrast, if the intrinsic motivator learns to model all the controlled objects, *i.e.* boxes, the agent will be intrinsically rewarded for visiting the different boxes, not just the first one. As a result, the agent will have more interactions with the boxes and a higher chance of solving the environment and getting an extrinsic reward. Unfortunately, inverse dynamics prediction is not enough to model all the controlled aspects of the environment. In this work, we discuss the drawbacks of using Inverse Dynamics. Alternatively, we introduce a causal approach to model the controlled aspects of the environment. Our causal approach is analogous to how humans assign a degree of blame to their actions. It compares the normal world to the current observed world and identify controlled aspects accordingly.

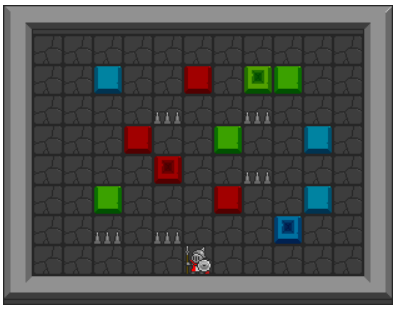


Figure 1. **Clusters Environment.** Intrinsic motivators should model all objects to solve the environment efficiently.

In this work, we propose Controlled Effects Network (CEN), a self-supervised model to identify controlled aspects of an environment. CEN is a specially designed auto-encoder neural network that transforms observations into two latent representations. The first representation models the normal world, while the second models the controlled aspects of the environment. We adopt an intrinsic motivator to leverage the controlled latent representations produced by CEN. This intrinsic motivator encourages the agent to have more interactions with controlled objects resulting in more efficient learning.

Corcoll *et al.* [CMV21] analyzed the modeling capabilities of CEN's latent representations. They showed how these latent representations could identify the controlled aspects of the environment. Meanwhile, inverse dynamics prediction models fail to identify these controlled aspects or identify them with limited capacity. This work aims to extend their analysis to Reinforcement Learning (RL) scenarios. We use CEN as the core of an intrinsic motivation module which we add to an RL agent. We test this agent's performance in very sparse environments and compare it to other agents with inverse dynamics-based intrinsic motivators. In very sparse environments, the CEN-based agent significantly outperforms the competition in both learning efficiency and final score achieved by a factor of **5**.

In summary, our main contributions are as follows:

- We propose CEN, a self supervised model to identify controlled effects.
- We create sparse environments with dense objects that highlight the importance of controlled aspects modelling.
- We test CEN based RL agents on these environments and show the superior qualities of CEN's exploration.
- We provide an open source implementation [1] of the Never Give Up (NGU) RL agent.

[1] https://github.com/Mo-youssef/controllable-IR

2 Background

In the following subsections, we first introduce the reinforcement learning problem and what differentiates it from other types of learning; we discuss the trade-off between exploration and exploitation and the predicament of sparse environments. Secondly, we introduce the notion of causality, the fundamental problem of causal inference, and show how naively applying causal concepts to reinforcement learning does not offer much benefit. Instead, we show how to adopt causal concepts to reinforcement learning to identify controlled effects. Thirdly, we briefly overview deep learning, in particular, Convolutional Neural Networks, Auto-encoders, and deep reinforcement learning.

2.1 Reinforcement Learning

Machine Learning Machine Learning is the science of making programs that can learn from data and experience, in contrast to traditional programs where a developer provides a set of rules for the machine to perform some program. Learning from data results in programs that can discover new patterns and use them to solve a given task. Throughout the last decade, this paradigm shift, learning from data, and the availability of tremendous amounts of data and computation resulted in many technological breakthroughs that changed human societies.

Instead of having a program or algorithm that follows a set of rules, in machine learning, we have a model which maps input data to the desired output. The task of learning/training is to find or create a model that best predicts the desired output. The desired output can vary based on the task, and it usually constitutes the learning signal which iteratively improves the model being trained. Machine Learning can be broadly divided into four categories based on the learning signal: Supervised, Unsupervised, Self-Supervised, and Reinforcement Learning.

Supervised learning is the most common category and has a long history of success. In supervised learning, the desired output is a label attributed to input data. Consequently, the models trained in a supervised fashion use the true labels, commonly named ground truth, as the learning signal. They exploit correlations between the input data and their true label to predict unseen test data's label correctly.

Unsupervised learning, on the other hand, does not have any true labels *i.e.* the model is fed only input data. The task in unsupervised learning is to discover hidden patterns in the input data, usually to segment the training data into semantically meaningful clusters. Unsupervised learning is mostly used in the context of clustering algorithms.

The borderline between Unsupervised and Self-Supervised learning is not very clear since Self-Supervised learning also provides the model with only input data without any labels. However, Self-Supervised learning usually creates a supervised learning task from the input data. For example, in BERT [DCLT18], a very popular language model, the training procedure consists of masking some words in an input signal and then training the BERT language model to predict these masked words. Recently, self-supervised learning achieved many breakthroughs in language [DCLT18, LOG+19, CKG+19], audio [HBT+21], and visual modeling [CKNH20, GSA+20, HCX+22] as well as multi-modal modeling [KSK21], especially in pre-training and transfer learning of highly complex unstructured data. Generally, self-supervised

learning is used to create very powerful representations of raw data that can be used in downstream tasks. These breakthroughs allow for integrating self-supervised learning with other types of learning.

Finally, in Reinforcement learning (RL), the learning signal comes in the form of a reward resulting from an interaction with an environment. The learning in RL aims to maximize the reward signal from the environment. Reinforcement learning is different from supervised learning since the learning process does not include a labeled dataset. Although RL seems similar to unsupervised learning, the difference is that RL aims to maximize a long-term reward signal. Unsupervised learning does not have interactions with an environment and does not maximize a reward. In this work, we are mainly concerned with Reinforcement learning, although we use supervised and self-supervised learning in our approach and experiments.

Characteristics of Reinforcement Learning When we think about learning, the first to come to mind is how we, humans, learn. Usually, humans explore their environment, interact with it, and learn something from these interactions. Humans often have a goal to achieve or a reward they want to maximize, and through trial and error, they learn skills or knowledge which help them achieve their goal. Reinforcement learning is concerned with studying this kind of learning and developing algorithms that learn from interactions with an environment to maximize a long-term reward.

In a broad sense, RL agents map a given situation to an action, perform this action, and observe the consequences of that action on the environment. During the learning process, the agent/learner is not told which action it should take. Instead, the environment reacts to the agent's actions and responds with a reward. This reward can be positive, negative, or neutral. The agent should learn to exploit the environment in order to maximize the positive reward or minimize the negative reward.

A unique characteristic of Reinforcement learning is the dependency of future states on the agent's present actions. For example, an agent walking in a maze finds itself at an intersection where it can either go right or left. Whatever the agent chooses will determine the possible actions and observations in the future. Therefore, the observations and experiences an agent can learn from are highly correlated and dependent on the actions the agent takes. This characteristic differentiates RL from supervised learning where the training data is assumed to be independent and identically distributed (i.i.d).

Another related characteristic of RL is the delayed rewards of present actions where an action performed by an agent might not result in an immediate reward. However, this action is paramount to having a reward in the future. For example, students writing their thesis do not get immediately rewarded for doing so, contrarily, they might face hardships. However, a thesis is essential for graduation, which is ultimately a big reward for the student. This delayed learning signal results in many complexities of reinforcement learning. Assigning the credit for the main causes (actions taken by the agent) is crucial for solving RL problems.

Why is Reinforcement learning important? Reinforcement Learning aims to maximize a numerical reward by acting on an environment. This formulation makes RL more goal-oriented compared to other learning methods. It also introduces very few human biases to what the agent

learns. As a result, the set of skills acquired by the agent during learning are sometimes surprising and lead to the emergence of very complex behaviors. Compare this to supervised learning, where the model tries to minimize a loss on a training set that contains human biases resulting in less surprising behaviors.

Silver *et al.* [SSPS21] postulates *"Intelligence, and its associated abilities, can be understood as subserving the maximization of reward by an agent acting in its environment."*. They argue that maximizing the reward in a sufficiently complex environment leads to the emergence of behaviors that constitute general intelligence. In other words, maximizing a reward is sufficient to create an agent with general intelligence. For example, a squirrel in the real world wants to collect as many nuts as possible. Firstly, it needs to learn how to use its sensors to perceive the world in order to locate the nuts, then it needs the motor skills necessary to get them. As the squirrel collects more nuts, it needs to store them somewhere, so it needs a notion of memory. Finally, to keep the nut storage safe, it needs to understand other squirrels in order to hide the storage adequately and maybe even deceive the other squirrels giving rise to social intelligence. The key factor of this example is the presence of a very complex environment, *i.e.* the real world, and a reward, *i.e.* nuts collected.

Reinforcement Learning agents learn from experience to maximize a reward returned by their environment. Thus, RL naturally fits as the learning system for the reward maximization in Silver's *et al.* [SSPS21] hypothesis. So, one could argue that Reinforcement learning is an essential piece in the general artificial intelligence puzzle.

Components of the Reinforcement Learning Problem Any Reinforcement Learning problem consists of six main components: Agent, Environment, Policy, Reward signal, Value function, and optionally a model of the environment. The agent is an existence that acts on the environment, RL aims to make the agent act in a way to maximize a long-term reward. The environment is the world the agent lives in, it refers to all the objects and dynamics that exist without direct control from the agent. The agent can directly act on the environment, these actions might have consequences governed by the dynamics of the environment. We use the agent's actions to refer to these direct interventions by the agent. On the other hand, all the resulting dynamics are beyond the direct control of the agent and thus are aspects of the environment. The remaining four components are related to the agent itself and are essential for the agent to act properly in its environment.

The policy defines the agent's behavior; it maps observations perceived from the environment to actions performed by the agent. The policy is a core component of an RL agent since it affects not only the actions taken but also the agent's future and thus its learning experience. The policy can be a very simple function, such as random selection from the set of possible actions, or it can be a very complex function implemented with Neural Networks. Finally, the policy can be deterministic *i.e.* maps an observation to a single action, or it can be stochastic, meaning it maps an observation to a probability distribution over the possible actions.

The reward signal defines the goal for the RL agent. The reward is a measure of how good or bad an action is. Although the environment can return the reward due to some actions performed by the agent, it can also be generated intrinsically by the agent to encourage certain behavior. The reward is considered the main training signal for an RL agent as it guides the improvement of the

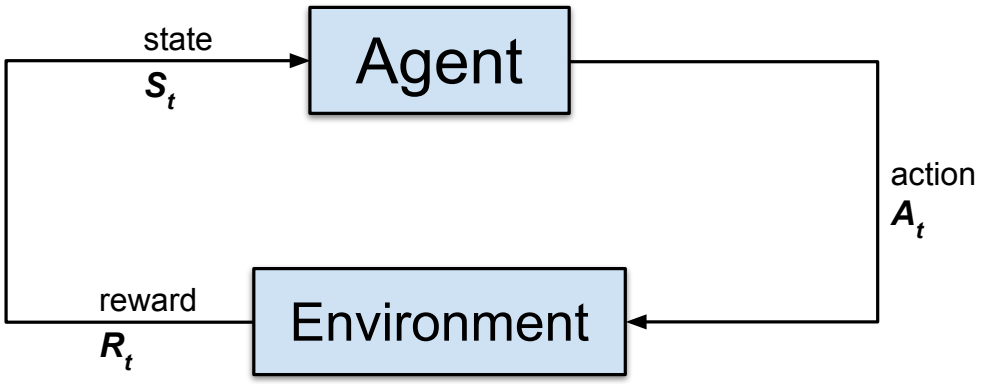


Figure 2. Markov decision process of the Reinforcement Learning problem. The agent interacts with the environment through an action based on current state. The environment responds with a new state and a reward.

policy. The reward signal can be deterministic or stochastic depending on the environment.

Despite the reward function determining what is good and bad for the agent, it is not enough. As we mentioned earlier, the reward does not necessarily follow the action taken by the agent, *i.e.* the agent might perform an action in the present; however, the reward for that action is received in the future after many unrelated actions. For such scenarios, the value function comes to our rescue since it measures how good a state is in the long run. The value function reflects the total amount of reward an agent will receive in the future by following a specific policy. As a result, the value function depends on the agent's policy which determines the future actions. The value function is more tricky to calculate than the reward signal because we must accurately know the agent's future. For complex environments, knowing the future is practically impossible. As a result, the value function is mostly estimated in large environments. It is worth noting that agents are usually trained to optimize the value function, not the direct reward signal, due to the delayed reward problem. However, it should be clear that the value function is just another representation of the reward signal. Ultimately, the reward signal guides the improvement of the policy through the value function.

Finally, the agent can have a model of the environment which simulates the real environment. If the model is accurate enough, it can be used to have better estimations of the value function leading to more efficient learning. If a reinforcement learning algorithm uses a model, it is called model-based RL; otherwise, it is called model-free RL. Policies can exploit good models by using them to plan ahead by predicting the future before it happens, resulting in a better choice of actions.

Formalization of Reinforcement Learning We follow Sutton *et al.* [SB18] in formulating the reinforcement learning problem as a Markov Decision Process (MDP). Fig. 2 shows the interaction between the agent and the environment. The agent is the learner and the decision maker, while the environment is the world the agent lives in and interacts with. The border

between the agent and its environment does not have to be a physical barrier. We refer to the environment as anything outside the agent's direct absolute control. For example, when an animal eats something nice, it is rewarded with a pleasure feeling. Although pleasure is generated inside the animal's brain due to some chemical processes, it is not considered a part of the agent/animal in the MDP formulation since the animal does not have direct absolute control over these chemical process, *i.e.* they are triggered by the environment.

In the MDP in Fig. 2, the agent performs an action on the environment based on some state. In response, the environment responds with a new state as well as a numerical reward. These interactions happen at discrete time steps *i.e.* $t = 0, 1, 2, 3, ...$ At each time step t, the agent observes a state $S_t \in \mathcal{S}$, a set of all the possible states. The agent then perform an action $A_t \in \mathcal{A}$, a set of all possible actions. At time step $t+1$, the environment responds with a reward $R_{t+1} \in \mathcal{R}$, a set of all possible rewards, $\mathcal{R} \subset \mathbb{R}$ the set of all real numbers. The environment also returns a new state $S_{t+1} \in \mathcal{S}$. The interaction between the agent and the environment continues indefinitely giving rise to a sequence of states, actions, rewards, *i.e.* $S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, S_{t+2}, ...,$ this sequence is sometimes referred to as a *trajectory*.

In this work, we are mainly concerned with finite MDPs which means that the sets $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ have a finite number of elements. This results in an interesting situation where for any t, both S_t and R_t are discrete distributions depending only on the previous state S_{t-1} and action A_{t-1}. In other words, the dynamics of the MDP in Fig. 2 can be defined as

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{2.1}$$

This definition encapsulates the most important attribute of the MDP, that is, the probability distribution of the next state and reward *i.e.* S_{t+1} and R_t depends only on the preceding state S_t and the performed action A_t. This means that the state at any time step has to perfectly summarize all of the interaction history between the agent and the environment. In other words, with a slight abuse of the notation where $Pr\{S_{t+1}, R_{t+1}|S_t, A_t\} = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$,

$$Pr\{S_{t+1}, R_{t+1}|S_t, A_t\} = Pr\{S_{t+1}, R_{t+1}|S_t, A_t, S_{t-1}, A_{t-1}, ...\} \tag{2.2}$$

In the reinforcement learning problem, we aim to maximize the accumulated reward over the entire sequence of interaction between the agent and the environment. To this end, we define the return G_t as

$$G_t = R_t + R_{t+1} + R_{t+2} + ... = \sum_{i=0}^{\infty} R_{t+i} \tag{2.3}$$

However, this definition is problematic for environments which do not end. Even for environments which terminate, *i.e. Episodic Environments*, this definition gives credit to every action performed by the agent since the beginning of the episode till the reward is received. This is not practical since very early action are usually not important to receiving a reward in the distant future. To solve both of these issues, we use the concept of discounted return defined as

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + ... = \sum_{i=0}^{\infty} \gamma^i R_{t+i} = R_t + \gamma G_{t+1} \tag{2.4}$$

where $0 \leq \gamma < 1$, if $\gamma = 0$ the agent optimizes the direct reward and does not care what happens in the future, *i.e.* a greedy agent. On the other hand, the larger γ, the more future rewards the agent optimizes.

Backed by these definitions, we can revisit the value functions and concretely define them. Value functions estimate how "good" a state or an action is for the agent. The "good" is defined in terms of the goal we are trying to optimize in the RL problem, *i.e.* the discounted return G_t. Obviously, the value functions depend on the agent's behavior and, subsequently, its policy. As we elaborated early, the policy maps the states to actions, so we define the policy as $\pi(a|s) = Pr\{A_t = a|S_t = s\} \mid \forall a, s \in \mathcal{A}, \mathcal{S}$. Basically, the probability of performing action a at time step t when the observed state is s.

We define the value function $v_\pi(s)$ as the expected discounted return when the agent starts at state s and follows policy π in choosing the subsequent actions. Formally, v_π is defined as

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i}|S_t = s\right], \qquad \forall s \in \mathcal{S} \tag{2.5}$$

$\mathbb{E}_\pi$ means that the agent follows the policy π when taking actions in the future time steps. For episodic environments, we define the value function of the terminal states as zero. This function is called the *state-value* function for *policy* π.

The value function defined above is iterative and it is more useful to write it in a recursive format as follows:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_t|S_t = s] + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s', S_t = s] \\ &= \mathbb{E}_\pi[R_t|S_t = s] + \gamma \mathbb{E}_\pi[v_\pi(s')|S_t = s] \\ &= \mathbb{E}_\pi[R_t + \gamma v_\pi(s')|S_t = s] \end{aligned} \tag{2.6}$$

We can expand the expectation in Eq. 2.6 to get the following equation for the state-value function in terms of the dynamics distribution (defined in Eq. 2.1):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_t + \gamma v_\pi(s')|S_t = s] \\ &= \sum_{a \sim \mathcal{A}} \pi(a, s) \sum_{s' \sim \mathcal{S}, r \sim \mathcal{R}} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right] \qquad , \forall s \in \mathcal{S} \end{aligned} \tag{2.7}$$

Equation 2.7 is called the *bellman* equation for the *state-value* function v_π for *policy* π.

In many cases, we need to know the value function after performing a specific action a starting at state s. This gives rise to the *Q-function* or *action-value* function. It is defined similar to *state-value* function, however, it takes the action as an input as well. We define it as follows:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i}|S_t = s, A_t = a\right], \qquad \forall s \in \mathcal{S} \tag{2.8}$$

which can be written in the bellman recursive format, using the same derivation, as:

$$q_\pi(s,a) = \sum_{s' \sim \mathcal{S}, r \sim \mathcal{R}} p(s', r|s, a) \left[r + \gamma \sum_{a' \sim \mathcal{A}} \pi(a', s') q_\pi(s', a') \right] \tag{2.9}$$

Solving a reinforcement learning problems means finding a policy π^* which achieves the largest return for all the states in the environment. It is worth noting that there might be more than one optimal policy. However, all optimal policies must have the same *state-value* function for all states in the environment, *i.e.* $v_{\pi_*^1}(s) = v_{\pi_*^2}(s) \quad , \forall \pi_*^1, \pi_*^2 \in \Pi_*, \ and \ \forall s \in \mathcal{S}$, where Π_* is the set of all optimal policies. Consequently, solving a reinforcement learning problem is akin to finding the optimal *state-value* function which is achieved by following an optimal policy, *i.e.*,

$$v_*(s) = \max_\pi v_\pi(s) \qquad , \forall s \in \mathcal{S} \tag{2.10}$$

Not only all optimal policies lead to the same *state-value* function, but they also lead to the same optimal *action-value* function, *i.e.*,

$$q_*(s,a) = \max_\pi q_\pi(s,a) \qquad , \forall s, a \in \mathcal{S}, \mathcal{A} \tag{2.11}$$

The optimal *state-value* and *action-value* functions are related directly through the following equations:

$$v_*(s) = \max_a q_*(s,a) \qquad , \forall s, a \in \mathcal{S}, \mathcal{A} \tag{2.12}$$

$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \qquad , \forall s, a \in \mathcal{S}, \mathcal{A} \tag{2.13}$$

Revisiting the bellman recursive equations for the state-value and action-value functions, we can derive the bellman optimal equations as follows:

$$v_*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \qquad , \forall s, a \in \mathcal{S}, \mathcal{A} \tag{2.14}$$

$$q_*(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma \max_{a'} q_*(s', a')] \qquad , \forall s, a \in \mathcal{S}, \mathcal{A} \tag{2.15}$$

Exploration vs. Exploitation Trade-off For simple environments, calculating the optimal value functions using the above equations is feasible using dynamic programming [SB18]. However, it is rarely possible to keep track of all the states for more complex environments. Due to many reasons, including the number of states being infeasibly large, the states not being observable, or a lack of computational resources to compute the optimal values.

As a result, we use Neural Networks to approximate the value functions for most practical purposes. To get good approximations for the value functions, we collect *experiences* of the agent interacting with the environment. Since we can access the reward in these experiences, we can calculate the actual value function. Then, we can train the neural network to better approximate

the value functions. The more diverse the experiences, the better approximation we get, so it makes sense to include as many experiences as possible.

However, as we stated before, the number of states might be too numerous to count. Many of these states are not useful to explore, so it might be better for the agent to stick to states that it knows have high values, ignoring the unexplored states. This dilemma gives rise to one of the most important trade-offs in Reinforcement Learning: the trade-off between exploration and exploitation. In exploration, we want the agent to explore as many states as possible to know the environment better, hoping that it finds highly rewarding states. Contrarily, in exploitation, the agent uses its knowledge to go to the high values states it knows about while ignoring the less valued states even if they are not well explored. In a nutshell, exploitation maximizes the expected reward from the next time step, while exploration seeks a greater total return in the long run.

Balancing exploration and exploitation is crucial in all RL applications. We want the agent to start learning with more exploration, and after some time, the agent should start exploiting its knowledge and going to more lucrative states. ϵ-*greedy* exploration does exactly that; at each time step, the agent has a non-zero probability of selecting a random action. This probability starts large and then decreases based on a linear schedule making the agent more exploitative. Many approaches were developed to allow better exploration based on desirable attributes such as boredom and curiosity. We will discuss them in detail in Sec. 3.2. Bonus-based exploration methods aim to improve exploration quality by providing the agent with bonuses proportional to the desired quality, such as boredom. These methods were shown to achieve excellent results in solving very complex environments.

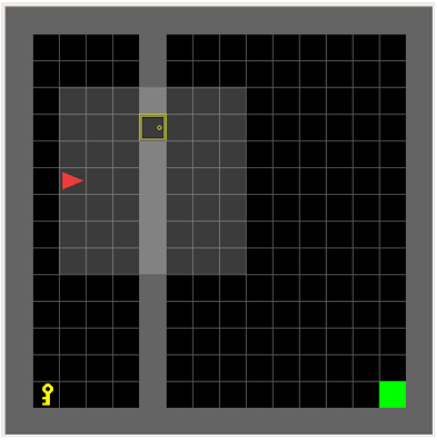


Figure 3. **Minigrid Key Door Environment.** An example of a sparse reward environment. the agent (red triangle) gets a reward only after picking the key, opening the door, and finally, reaching the green goal.

Sparse Environments Although proper exploration is essential for any RL environment, it is increasingly vital in the case of sparse environments. Sparse Environments are environments that sparsely reward the agent, making them very challenging. For example, Fig. 3 shows a sparse environment from the Gym-minigrid suite [CBWP18] where the agent (the red triangle) has first to pick the door and then move to the door and open it using a special action button. Finally, the agent has to reach the green goal at the corner of the map to receive a reward of 1. Despite the environment being very simple for humans, it is very challenging for agents with random exploration strategies such as ϵ-*greedy*. The probability of the agent randomly performing a correct sequence of actions is very low, resulting in the agent taking a very long

Taken Pill	Pregnant	Not Pregnant	Total
True	$^{1}/_{10} = 0.1$	$^{81}/_{90} = 0.91$	$^{82}/_{100} = 0.82$
False	$^{16}/_{80} = 0.2$	$^{19}/_{20} = 0.95$	$^{35}/_{100} = 0.35$

Table 1. **Hypothetical example of Simpson's paradox.** The entries in the table represent the probabilities of having the pain relieved.

time to learn anything. This problem is magnified if the environment allows a limited number of time steps. The extreme case is when the number of time steps available is exactly equal to the number of steps required by an optimal policy, in this case, it would be impossible for an agent exploring randomly to reach the goal. This scenario creates a need for exploration strategies that exhibit certain qualities promoting more efficient exploration. In this work, we propose a bonus-based exploration method that allows the agent to better explore the environment by encouraging interactions with controllable objects in the environment.

2.2 Causality

Our proposed method works by identifying objects over which the agent has control. We then encourage the agent to do more events that involve these objects. To determine whether the agent can control something or not, we need to define a notion of causal effect, *i.e.* the agent causes an object to move. Causality has been studied in multiple domains, including economy, statistics, machine learning, and more. Causal inference goes beyond regular associative relations and determines the causal relationship between different elements of the problem.

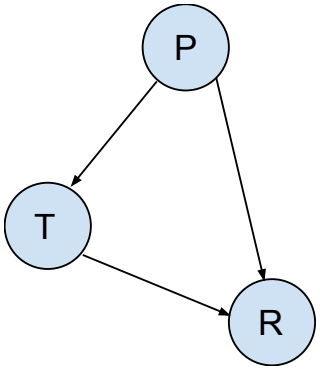


Figure 4. **Causal graph of the Simpson's paradox hypothetical example. P** stands for pregnancy, **T** stands for taking the pill. **R** stands for pain relief. The causal graph shows that the pregnancy affects both taking the pill and pain relief, causing the illusion of the pill relieving the pain.

Simpson's Paradox To give more intuition about associative and causal relations, we proceed with the following example. Imagine a pharmaceutical company producing brand new pain-relieving pills targeted at females. Women try this new product and report whether the pills relieved their pain or not. Assume that 82% of the females who took the pill reported that their pain was relieved *i.e.* $P(R|T) = 0.82$ where R, T are the events of having pain relieved and taking the pill, respectively. On the other hand, 35% of those who did not take the pill reported that their pain was relieved *i.e.* $P(R|\neg T) = 0.35$.

Given these conditional probabilities, one might be tempted to conclude that taking the pill is effective in relieving the pain. However, let us assume that most of the women who did not

take the pill are pregnant because of side-effects concerns. If we break down the probabilities conditioned on pregnancy status, we get the probabilities in Table 1.

Obviously, the pill seems to have no advantage over not taking the pill, but the main cause of the pain is pregnancy, evident from the very low rate of pain relief. The case where women do not take the pill is actually better at relieving the pain. Unfortunately, most of the subjects who did not take the pill are pregnant, making the overall relief probability quite small. This is known as Simpson's paradox, and it shows that a confounding factor can greatly bias the final result. In our example, the confounding factor was the pregnancy and it had an effect over the pain as well as whether subjects decide to take the pill or not. This relation is captured in Fig. 4.

Fundamental Problem of Causal Inference To remove the effect of pregnancy from our problem, one might be tempted to collect more data where more pregnant women take the new medication or stop non-pregnant women from taking the pill. However, this might be impossible due to ethical concerns or the infeasibility of collecting more data. Our best bet is to account for the imbalance in the data by weighting the conditional probabilities with the pregnancy ratio *i.e.*,

$$P(R|T) = P(P) \times P(R|P,T) + P(R|\neg P,T) = \frac{90}{200} \times 0.1 + \frac{110}{200} \times 0.91 = 0.5455$$

Where $P(P)$ is the probability of a subject being pregnant. On the other hand, for the case of not taking the pill, we have,

$$P(R|T) = P(P) \times P(R|P,T) + P(R|\neg P,T) = \frac{90}{200} \times 0.2 + \frac{110}{200} \times 0.95 = 0.6125$$

Now, the probabilities make more sense and are more representative of the task at hand. All we did was marginalize the condition probability based on the confounding factor *i.e.* pregnancy.

In many practical problems, it can be infeasible to collect the data where the confounding factor (pregnancy) is identified for all the scenarios (taking and not taking the pill). For example, if we want to test whether the pill is beneficial for a single woman. This woman can either take the pill or not take it creating two worlds where in one world, she takes the pill while in the other she does not. As a result, it is impossible to compare the two worlds. This dilemma is known as **the Fundamental Problem of Causal Inference**, basically, we can not observe the two worlds and compare them. Consequently, we can at best speculate what could be the outcome in the alternate world. In this work, we use a neural network to approximate the normal world and identify controlled effects as changes introduced as a result of the agents intervention on the environment.

2.3 Deep Neural Networks

Deep Neural Networks (DNNs) are a family of machine learning models that started with a very simple idea of mimicking how neurons in the brain work. DNNs have exploded in recent years (since 2012) mainly due to the availability of computational resources and data. In 2012, Krizhevsky *et al.* [KSH12] used a family of architectures of DNNs called Convolutional Neural Networks (CNN) in the LSVRC-2010 contest to classify input images from the ImageNet

dataset [DDS+09] into 1000 classes. They managed to significantly outperform the state-of-the-art models by decreasing the top-5 error rate by over 10%, sparking the boom of Deep Neural Networks. They also showed the feasibility of successfully training very big DNNs using GPUs (Graphical Processing Units). Since then, researchers have proposed many architectures and models better suited to a variety of applications. We first present a simple Neural Network and then proceed with more advanced architectures.

A simple feed-forward Neural Network (NN), shown in Fig. 5 is divided into layers. Each layer is built using neurons as its atomic building block. A neuron is a simple computation unit that aggregates its inputs through an affine transformation where it multiplies the inputs with a weight matrix W and adds a bias b to the sum. Usually, the affine transformation is followed by a non-linear phase, where a non-linear function is applied to the sum. Many non-linear functions were proposed; a summary can be found here [Ped18]. In our work, we mainly use Relu and Sigmoid functions defined as follows:

$$relu(x) = \begin{cases} x & \text{if } x >= 0 \\ 0 & \text{if } x > 0 \end{cases} \tag{2.16}$$

$$sigmoid(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

The more layers we stack, the more powerful representations we get for the input features. The output of the network is usually defined to suit the task at hand. For example, if we try to predict the probability of some class, we can use a sigmoid function to limit the output between 0 and 1. Next, we choose a loss function based on the task in order to measure the error between the network prediction and the true labels.

To train such a neural network, we use the chain rule of derivatives to measure the gradient of the error with respect to the parameters of the network. These gradients reflect the relation between the parameters and the error, where a small change in the direction of the gradients leads to the biggest increase in the error. Consequently, we update the parameters in the direction opposite to the gradients such that the new parameters lead to the biggest decrease in the error. The training phase is divided into cycles, each cycle has two parts: firstly, the forward propagation, where we run the input through the network producing the prediction; secondly, the backward propagation [LBD+89] where we calculate the gradient of the loss with respect to every parameter and update it. We keep looping over this cycle until we reach a satisfying performance, measured via the loss function or a different metric. More architectures were proposed based on the simple feed-forward neural network. We are particularly interested in two families: Convolutional Neural Networks (CNNs) and Auto-encoders.

CNNs [GBC16, LB+95] are based on the convolution operator where we typically apply grid-like filters to the input. We are mostly concerned with images, so we will focus the discussion on 2D convolutions, but the arguments can be extended to any number of dimensions. The 2D convolution operator takes as an input a 2D matrix X, typically representing an image, and a weight matrix W called kernel or filter, having smaller dimensions compared to the input. The

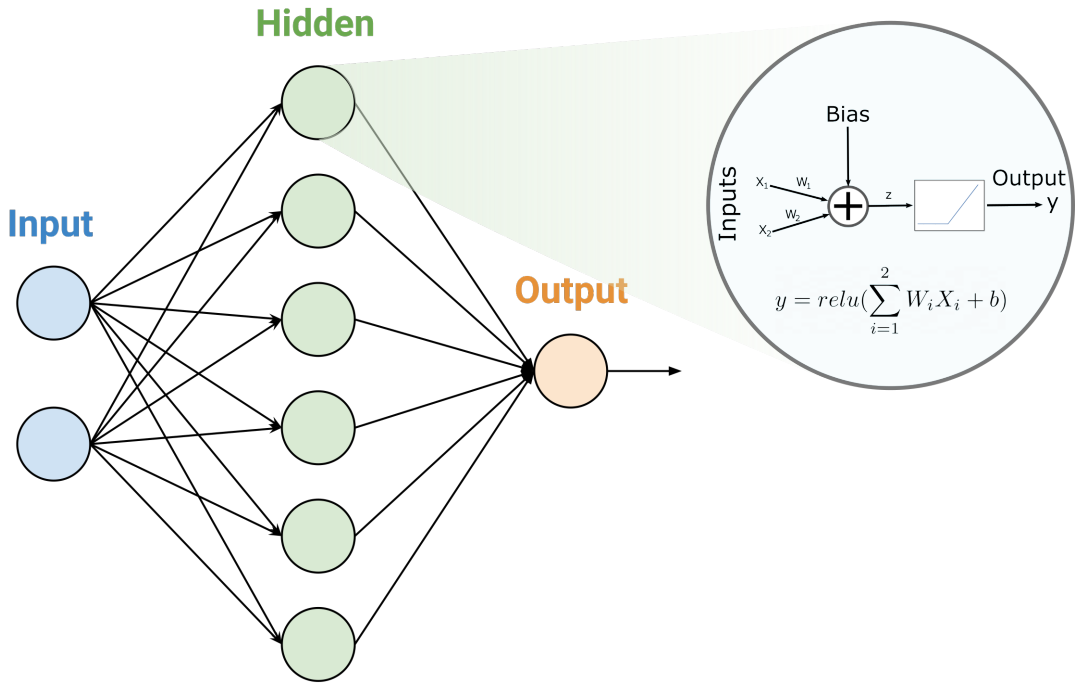


Figure 5. **Simple Feed-Forward Neural Network.** Each neuron multiplies its inputs with a weight vector, adds a bias, and finally apply a non-linear function.

output of the convolution is then defined by y where,

$$y_{i,j} = \sum_m \sum_n X_{i+m,j+n} \times W_{m,n} \tag{2.18}$$

Equation 2.18 can be modified to allow padding, where we add zeros to the input X, and strides, where we skip some parts of the input while applying the filter. For more formalization details, we refer the reader to [GBC16].

Prior to deep learning, the filter's weights were chosen to perform specific tasks such as edge detection [KVB88]. Tuning these weights to achieve the desired effect is an extremely tedious task and very limited for detecting patterns in unstructured data such as images. To mitigate this issue, CNNs use data to learn the weights of the filters through backpropagation, updating them such that the loss function decreases. This result in filter weights performing a wide variety of tasks. LeCun *et al.* [LB$^+$95] showed how to train a CNN to classify handwritten digits. Then, Krizhevsky *et al.* [KSH12] trained a deep CNN having 60 million parameters to outperform all other models in classifying natural images into 1000 classes. Many modifications were introduced to the vanilla CNN architecture leading to much more powerful models that eventually surpassed human-level performance on ImageNet classification [HZRS15]. Notably, Kaiming He *et al.* [HZRS16] presented residual networks, which are based on shortcut connections between the input to a CNN and its output. Ioffe and Szegedy [SIVA17] introduced batch normalization

making the training of deep neural networks less sensitive to the learning rate and parameter initialization. They also showed that batch normalization has a regularization effect similar to dropout [SHK+14].

Recurrent Neural Networks (RNNs) [WZ89, Wer90] used neural networks and backpropagation to tackle the problem of time series analysis such as language, video, and audio signals. RNNs maintain a state vector that stores information from the past of the sequence. The earlier approaches suffered from exploding or vanishing gradients with longer sequences [Hoc98]. To remedy these problems, Hochreiter and Schmidhuber [HS97] proposed LSTM, which uses gates to better maintain the state vector by forgetting less useful aspects of the vector. LSTM stored information over extended time intervals resulting in a very powerful time series model that paved the way for many breakthroughs in language [BCB14, XBK+15], audio [GMH13], and sequence to sequence modelling [SVL14]

Rumelhart *et al.* [RHW85] described a novel feed-forward neural network that learns internal representations by backpropagation. Their network is identical to the framework of autoencoders, where the output has a similar form to the input. They used such a network to learn a good compressed representation of the network's input. Matan *et al.* [MBLD92] applied CNNs to inputs with variable sizes since the same filters are applied to every part of the input. This paved the way for using CNNs in the framework of autoencoders for applications where the output has a similar form to the input, such as image segmentation, image denoising, object detection, and other applications.

Long *et al.* [LSD15] proposed Fully Convolutional Networks (FCN), an autoencoder made fully from convolutional layers. They used FCN to segment natural images. Meanwhile, Ronneberger *et al.* [RFB15] presented U-Nets, which are very similar to FCN. However, they use upsampling operators to increase the spatial dimension of the compressed representation resulting from a backbone CNN while using convolutional layers to produce more precise outputs compared to FCN. Our proposed architecture borrows ideas from CNNs, RNNs, and Autoencoders to identify controllable aspects of the environment; more details will be discussed in the Methodology section.

3 Related work

3.1 Policy Gradient Methods

Model-free Reinforcement learning methods can be roughly divided into two main families: value function learning and direct policy learning. Value function methods learn the optimal value and then use it to derive an optimal policy that chooses actions maximizing the learnt value function. On the other hand, direct policy methods learn an optimal policy directly without the need for any value function. Finally, Actor Critic methods combine both, learning the policy and the value function, to converge faster with lower variance.

We approximate the policy of the agent with a differentiable parametric function $\pi(a|s,\theta)$, *i.e.* a neural network with a softmax output. Optimizing the policy involves calculating the gradient of a return value with respect to the parameters θ of the approximate policy. Then, we use gradient ascent to change the parameters in the direction which maximizes the return value. We define the return value to be the value of the initial state S_0, *i.e.* $J(\theta) = v_{\pi_\theta}(S_0)$.

The Policy Gradient Theorem [SB18] states that

> *for any episodic environment,*
>
> $$\nabla J(\boldsymbol{\theta}) \propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \tag{3.1}$$
>
> where, $\boldsymbol{\theta}$ are the parameters of the policy function approximation, ∇J is the gradient of the initial state value, $\mu(s)$ is a probability distribution over the possible states in the environment and it is determined by the policy π.

The Policy Gradient Theorem directly relates the gradient of the return to the gradient of the policy which is a requirement to perform gradient ascent and optimize the policy. REINFORCE [Wil92] is the simplest policy gradient method which utilize the policy gradient theorem. It uses Monte Carlo sampling [MU49] of episodes to approximate the gradient of the return with respect to the policy function parameters. The gradient can be simplified as follows,

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &\propto \sum_s \mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s,\boldsymbol{\theta}) \\ &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t,\boldsymbol{\theta}) \right] \end{aligned}$$

where the expectation takes into consideration the distribution of all states while following the policy π.

This can be further simplified by weighting the $q_\pi(S_t,a)$ with the policy $\pi(a|S_t,\boldsymbol{\theta})$ converting the summation over actions into an expectation over the policy and the action corresponds to any

Algorithm 1: REINFORCE

Input: Parametric policy approximation: $\pi(a|s, \boldsymbol{\theta})$
Reinforcement Learning Environment: Env
Discount factor: γ
Learning rate: $\alpha > 0$

Result:
Policy $\pi(a|s, \boldsymbol{\theta})$ optimized to solve a given environment

```
Initialize the policy parameters θ;
foreach Episode do
    Create a trajectory S_0, A_0, R_1, S_1, ..., S_{T-1}, A_{T-1}, R_T in Env by following
     policy π(.|., θ);
    foreach time step t in [0,T] do
        calculate the return G_t = Σ_{i=t}^{T} γ^{i-t} R_{i+1};
        update the parameters θ = θ + α γ^t G_t ∇ ln π(A_t|S_t, θ)
    end
end
return π(a|s, θ);
```

specific timestep, *i.e.*,

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_\pi \left[\sum_a q_\pi(S_t, a) \nabla \pi(a|S_t, \boldsymbol{\theta}) \right] \\
&= \mathbb{E}_\pi \left[\sum_a \pi(a|S_t, \boldsymbol{\theta}) q_\pi(S_t, a) \frac{\nabla \pi(a|S_t, \boldsymbol{\theta})}{\pi(a|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \boldsymbol{\theta})}{\pi(A_t|S_t, \boldsymbol{\theta})} \right] \\
&= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right]
\end{aligned} \tag{3.2}$$

notice that the expected return at state S_t doing action A_t is $\mathbb{E}_\pi[G_t|S_t, A_t] = q_\pi(S_t, A_t)$. Thus,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_\pi \left[G_t \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta}) \right] \tag{3.3}$$

This gradient can be used to update the policy parameters such that the new policy maximize the return. REINFORCE can be summarized in Algorithm 1 REINFORCE can be further improved by calculating the advantage of an action instead of its value. The advantage of an action A_t is defined as $A_t = q(S_t, A_t) - v(S_t)$. $v(S_t)$ is basically the state-value function at time step t, similar to the policy, we approximate $v(.)$ with a differentiable function usually, *i.e.* neural network in our case. Although, this results in a bias in the return, it greatly reduces its variance

which leads to faster convergence [SB18]. By adding the baseline to equation 3.3, we get,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[(G_t - v(S_t|\boldsymbol{\phi}))\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})\right] \tag{3.4}$$

where $\boldsymbol{\phi}$ are the parameters of the state-value function approximator. $\boldsymbol{\phi}$ are updated using temporal difference (TD) learning [SB18] where the approximate state-value function is used to predict the return at time step t, then this predicted value is compared against the true value which is calculated after each episode. The gradient of this loss is then used to update $\boldsymbol{\phi}$. Formally,

$$\begin{aligned} G_t &= \sum_{i=t}^{T} \gamma^{i-t} R_{i+1} \\ \delta &= G_t - v(S_t|\boldsymbol{\phi}) \\ \boldsymbol{\phi} &= \boldsymbol{\phi} + \alpha_{\phi}\delta\nabla v(S_t|\boldsymbol{\phi}) \end{aligned} \tag{3.5}$$

where α_{ϕ} is the learning rate for the approximate state-value function and $\nabla v(S_t|\boldsymbol{\phi})$ is the gradient of the value function at state S_t with respect to the parameters $\boldsymbol{\phi}$.

A major drawback of REINFORCE is the need to create a full trajectory in the environment for each parameter update. This makes REINFORCE very sample inefficient, meaning that it needs a lot of steps to learn a good policy. REINFORCE needs the full trajectory to calculate the return at each state and accordingly update the policy function parameters and the state-value function parameters in case a baseline is used.

Actor-Critic methods mitigate this problem (the need for full trajectories) by using the state-value function to approximate the return value at each state. Consequently, both the policy and state-value function parameters can be updated at each time step. Although using an approximation for the return introduces a bias to the return value, it was shown that using such approximation results in less variance and better sample efficiency. The most straightforward method that apply this principal is one-step Actor-Critic, where the full return in REINFORCE with baseline is replaced with an approximation as follows,

$$G_t = R_{t+1} + \gamma v(S_{t+1}|\boldsymbol{\phi})$$

Consequently, the gradient in Equation 3.4 becomes,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma v(S_{t+1}|\boldsymbol{\phi}) - v(S_t|\boldsymbol{\phi})\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})\right]$$

which can be simplified into,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\pi}\left[\delta\nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})\right] \tag{3.6}$$

where $\delta = R_{t+1} + \gamma v(S_{t+1}|\boldsymbol{\phi}) - v(S_t|\boldsymbol{\phi})$

One-step Actor-Critic method reaps the benefits of both value learning methods and policy gradient methods, however, it suffers from the drawbacks of both. The drawbacks from the policy gradient method used is usually more prominent since it defines the behavior of the agent. Schulman *et al.* [SLA$^+$15] provided theoretical evidence of a limitation on the policy gradient

methods making it very sample inefficient. They showed that small changes in the policy function approximation parameters leads to drastic changes in the expected return leading to a performance collapse. In order to constraint the changes to the policy parameters, vanilla policy gradient methods use very small learning rates making the training process very long.

Schulman *et al.* [SLA+15] proposed Trust Region Policy Optimization (TRPO) which uses a Kullback–Leibler (KL) divergence loss to force the updated policy parameters to be close to the old parameters. They showed significant gains in sample efficiency with larger learning rates. However, their method is very complicated and require the calculation of the Hessian matrix of the policy parameters. Although they provide many approximations and optimizations to calculate the hessian, it is still inefficient especially in the case of neural networks where the number of parameters is huge.

Schulman *et al.* proposed Proximal Policy Optimization (PPO) [SWD+17] tackled the same problem of big changes in the policy parameters in a much simpler way. It does not require any calculation of second order gradient, *i.e.* the hessian matrix. Schulman *et al.* proposed two versions PPO-penalty and PPO-clip, the latter is widely used due to its simplicity. PPO-clip simply modifies the objective J such that the changes in the policy parameters do not induce large changes in the objective avoiding performance collapse. The new objective function then becomes,

$$J_{clip} = \mathbb{E}_{\pi}\left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}A_{\pi_{\theta_k}}(s_t,a_t),\ g(\epsilon, A_{\pi_{\theta_k}}(s_t,a_t))\right)\right] \tag{3.7}$$

where,

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases}$$

and θ_k are the parameters from the current time step *i.e.* parameters before updating, $A_{\pi_{\theta_k}}(s,a)$ is the advantage of performing action a at state s, it is defined as $A(s,a) = q(s,a) - v(s)$, ϵ is a hyperparameter usually small (~ 0.2).

The intuition behind this constraint is that we measure the ratio between the old policy and new policy, if the absolute value of this number is very big we clip it to $1+\epsilon$ if the advantage is positive and $1-\epsilon$ if the advantage is negative. This directly constraint the effect of changing the policy parameters on the objective function allowing for the usage of larger learning rates which leads to better sample efficiency. PPO-clip adds an entropy bonus [Wil92, MBM+16] to the objective function which encourages the policy to sample actions uniformly resulting in better exploration. The entropy is defined as $H_{\pi_\theta}(s) = -\sum_a \pi_\theta(a,s)\ln\pi_\theta(a,s)$.

We use actor-critic agent as our reinforcement learning method, with the actor (policy gradient method) being PPO and the critic estimates the advantage of action-state pairs. We use the same network (shared parameters) with two heads, the first for the advantage estimation, and the second for the policy output. The final objective function, after including the entropy bonus and critic loss, becomes,

$$J_T = \mathbb{E}_{\pi}\left[J_{clip} + c_1 H_{\pi_\theta}(s) - c_2 J_a\right] \tag{3.8}$$

where c_1, c_2 are hyperparameters set in our implementation to 0.01 and 1 respectively. J_a is the critic objective used to learn a better advantage function approximator, we use eligibility trace temporal difference (TD-Lambda) similar to Mnih *et al.* [MBM+16], the same objective suggested by the PPO authors.

3.2 Exploration

Despite deep reinforcement learning agents outperforming humans in some highly complex environments such as Atari [MKS+13], efficient comprehensive exploration still remains one of the main challenges for RL. The exploration problem is especially important in sparse reward environments which we are mainly tackling in this work. In value function learning, explicit exploration strategies need to be adopted by the RL agent. ϵ-*greedy* is the most commonly used exploration strategy, where with probability ϵ the agent selects an action randomly while with probability $1 - \epsilon$ it selects the best action according to the value function approximation. An obvious drawback of this strategy is its high variance and sample inefficiency, *i.e.* it needs many steps to explore well. This drawbacks are more significant in complex as well as sparse environments where lots of actions needs to be done correctly in order to receive a reward.

Humans, on the other hand, are much more successful in these types of sparse reward environments. For example, a small child in a room where there is a locked cupboard would explore its surroundings until it finds a key and then uses it to open a cupboard and potentially get a reward *i.e.* candy. The child in this scenario does not know if there is candy or not in the cupboard but it intrinsically rewards itself when it explores its surroundings. A notion of novelty or curiosity motivates the child to look for new places and identify new items such as a key. While a controllability aspect motivates the child to use the key in order to change the environment *i.e.* open the cupboard. The combination of both novelty and controllability leads the child to open the cupboard. Bonus based exploration strategies aim at augmenting exploration with desirable attributes such as novelty/curiosity and controllability. The former have been studied extensively while the latter is less studied in the context of Reinforcement Learning. Next, we give a brief overview over the bonus based strategies.

Pseudo-counts [BSO+16, OBOM17] encourages curiosity in exploration by estimating the counts of visited states using density models. The bonus is then added to the reward provided by the environment, to avoid confusion we call the bonus intrinsic reward and regular reward from the environment as extrinsic reward. Pseudo-counts inversely relates the intrinsic reward of a given state with its count, such that the agent is encouraged to visit states which are less visited/explored and vice versa. In complex environments, explicitly counting the number of visits to each state is infeasible due to the enormous number of states. Consequently, Pseudo-counts use different density models $\mu(s)$ to approximate the frequency of visiting a given state s.

Intrinsic Curiosity Module (ICM) [PAED17] uses a combination of an encoder, inverse model, and forward model to promote curiosity in exploration. The encoder $\phi(s_t)$ takes as an input an observed state s_t and embeds it in a learned feature space. The inverse model $I(\phi(s_t), \phi(s_{t+1}))$ uses the embedding of two successive states to predict the action $\hat{a}_t$ done to transition from state s_t to state s_{t+1}. Both the inverse model and the encoder are trained via a cross entropy loss between the predicted and true action, *i.e.* $L = -\sum_{i=1}^{|A|} \mathbb{1}(a_t = A_i) \ln(\hat{a}_t)$ where $|A|$ is the number of actions, and $\mathbb{1}$ is the identity function. Finally, the forward model $f(\phi(s_t), a_t)$ predict the embedding of next state $\hat{\phi}(s_{t+1})$ from the embedding of the current state $\phi(s_t)$ and the current action a_t. The intrinsic reward is then calculated as the Squared Error between the predicted next state embedding $\hat{\phi}(s_{t+1})$ and the true next state embedding $\phi(s_{t+1})$, *i.e.* $r^i(s_t) = (\hat{\phi}(s_{t+1}) - \phi(s_{t+1}))^2$.

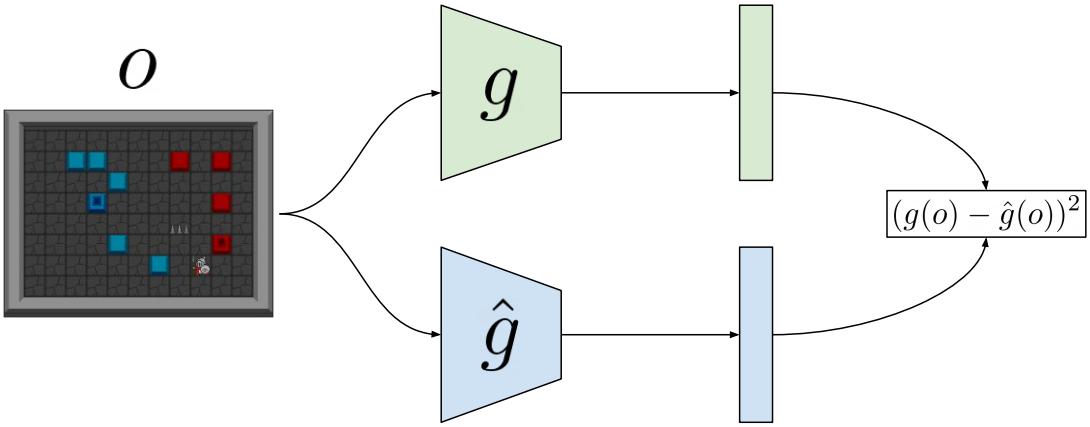


Figure 6. **Random Network Distillation.** The network $\hat{g}$ learns to reproduce the representations from the network g. The more g trains on an observation, the less intrinsic reward is given.

The encoder solves the problem of having innumerable states by embedding them into a learned space removing useless information and noise. The inverse model forces the learned space to be useful in predicting the actions used in a transition. It can also be seen that the inverse model loss teaches the encoder to remove redundant and aspects of the environment which do not help with predicting the action taken. Finally, the forward model is responsible for encouraging curiosity where the forward model is randomly initialized resulting in bad performance over all states and consequently higher intrinsic reward. The more the agent visit a given state, or similar ones in the learned space, the better the forward model gets at predicting such states leading to smaller intrinsic reward.

Random Network Distillation (RND) [BESK18] improves exploration through the concept of boredom, it is very similar to the forward model of ICM. RND has two identical copies of the same deep neural network which takes a state s as an input and output some embedding $g(s)$, shown in Fig. 6. The two networks are initialized differently, one network g produces target distributions and its weights are fixed, *i.e.* they never change. The second network $\hat{g}$ is trained using Mean Squared Error (MSE) to produce similar embeddings to the one produce by the target network, *i.e.* $g(s) = \hat{g}(s)$.
The intrinsic reward is proportional to the squared error between $g(s)$ and $\hat{g}(s)$. The parameters of $\hat{g}$ are trained using the MSE and eventually the resulting embeddings should converge to those produced by g. The intuition for why it works is identical to that of ICM, where initially the embeddings from the two networks for the same state are very different, but as the training progress the loss decreases for more frequent states resulting in a decrease in the intrinsic reward, *i.e.* the agent get bored from frequent states.

Although the aforementioned bonus-based exploration strategies have foundations in cognitive behaviour, Taiga *et al.* [TFM$^+$21] raised concerns about the efficacy of such strate-

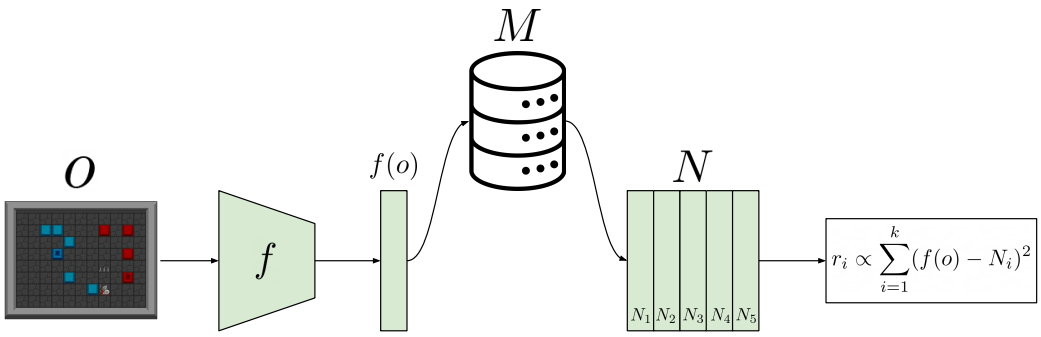


Figure 7. **NGU Episodic Memory.** The encoder f maps the observation o into the representation $f(o)$. The nearest k-neighbors N are retrieved from the episodic memory. Finally, the intrinsic reward is calculated proportional to the euclidean distance of $f(o)$ from its neighbors

gies. They performed extensive benchmarking for these methods by adding them to Rainbow DQN [HMVH$^+$18] and compared it to the naive ϵ-*greedy*. They concluded that these strategies work well on specific hard exploration environments such as Montezuma's Revenge from the atari suite [BSO$^+$16], however they perform worse than ϵ-*greedy* on easy exploration environments. They showed that all these methods are tuned towards specific environments and overall there is no clear "best" strategy. They proposed that these exploration methods might be redundant and complex RL algorithms implicitly explore with curiosity. For example, prioritized experience replay [SQAS15] might select less visited transitions that have high priority, simulating the effect of pseudo-counts. Finally, they proposed having more rigorous evaluation that are more comprehensive.

3.3 Never Give Up

Badia *et al.* [BSV$^+$20] presented Never Give Up (NGU) Reinforcement Learning framework which achieved state-of-the-art performance on the atari suite and performed extraordinarily well on hard exploration games such as Montezuma's Revenge and Pitfall. They noticed that the improvements attained by different exploration strategies are orthogonal, in the sense that they do not impair each other and generally they do not hurt the performance of the learning algorithm. They developed a framework which combines multiple bonus-based exploration strategies with very powerful learning agent.

In particular, they used Recurrent Replay Distributed DQN (R2D2) agent [KOQ$^+$18] with an inverse model and RND for generating the intrinsic reward. They divided the intrinsic reward into two components, the first is a life-long novelty module implemented by the RND, while the second is an episodic novelty module implemented via the inverse model. The life-long novelty module provides the agent with the ability to get bored of states that are visited regularly in every episode such as the starting state. On the other hand, the episodic novelty module is concerned with novelty in each episode and reset when the episode terminates. It mainly encourages the

agent to get bored quickly from states that are visited frequently in a single episode.

The life-long novelty module is implemented exactly the same as RND. We have two identical encoders both initialized randomly and one encoder learns to transform the observations into the same representations produced by the fixed encoder. Figure 6 shows the architecture of RND, we use CNNs as the encoder g and $\hat{g}$. The parameters of $\hat{g}$ are learned using MSE between the embeddings of both g and $\hat{g}$. If the agent visits a state very often, the encoder $\hat{g}$ will learn to produce the same representation as the random encoder g for this specific state. The life-long novelty module rewards the agent proportional to the squared error loss between the two representations $g(o)$ and $\hat{g}(o)$, where $g(o) \in \mathbb{R}^{d_r}$ and $o \in \mathbb{R}^{W \times H \times C}$ W, H, and C are the observations' width, height, and channels respectively. Therefore, the more the agent visits a state, the less reward it gets from visiting it. However, $\hat{g}$ takes time to learn the correct mapping making it suitable for avoiding frequent states across episodes but not within a single episode.

Algorithm 2: NGU Episodic Memory Intrinsic Reward calculation

Input: M; k; $f(o_t)$; c; ϵ; ξ; s_m; d_m^2;

/* c, ϵ, ξ, s_m are all hyperparameters, while d_m^2 is a moving average of previous distances. For more details, see [BSV+20] */

Result: $r_i(o_t)$

$N =$ nearest k-neighbors of $f(o_t)$ from M;
for $i \in$ *range(k)* **do**
  $d_n[i] = \frac{\sum(f(o_t)-N_i)^2}{d_m^2}$;
end
$d_n = max(d_n - \xi, 0)$;
$K = \frac{\epsilon}{\epsilon + d_n}$;
$s = \sqrt{\sum_{i=1}^{k} K[i]} + c$;
if $s > s_m$ **then**
  $r_i = 0$;
else
  $r_i = \frac{1}{s}$;
end
return r_i;

The episodic novelty module is more nimble and reacts to visiting states faster, thus, it is more suitable for encouraging curiosity within a single episode. It consists of an encoder/embedding function f, an inverse model h, and an episodic memory M. The episodic novelty module uses the encoder f to map an observation o to a latent representation $f(o)$, where $f(o) \in \mathbb{R}^{d_e}$, then it stores this representation $f(o)$ into an episodic memory M that is emptied at the beginning of each episode. To calculate the intrinsic reward $r_i \in \mathbb{R}$, the k-nearest neighbors to the representation $f(o)$ are retrieved from the episodic memory. The intrinsic reward is then calculated proportionally to the average euclidean distance between the representation $f(o)$ and all representations in its

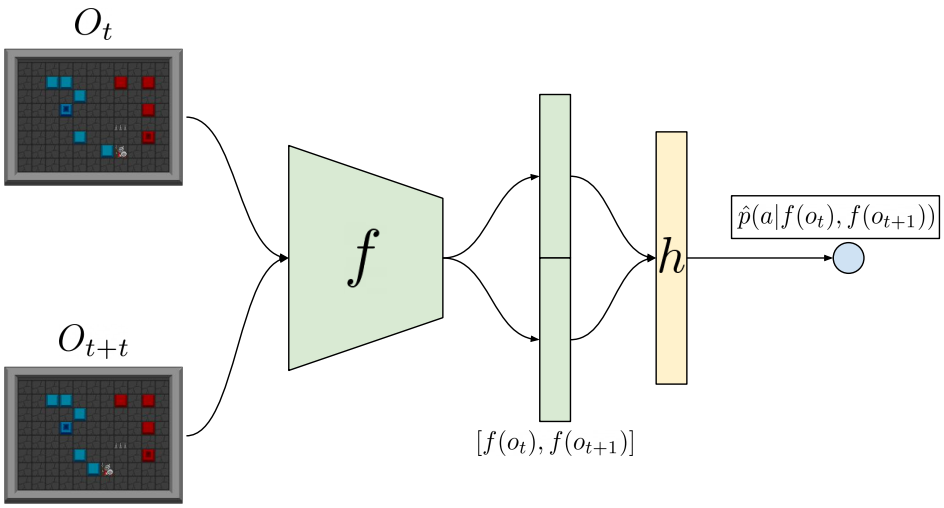


Figure 8. **NGU Inverse Model.** The classifier h predicts the action a used to transition from O_t to O_{t+1}. The encoder learns to discard useless information and model the agent location.

neighborhood N. Figure 7 shows the process of calculating the intrinsic reward for a given observation, *i.e.* $r_i(o_t)$. The exact reward calculation is detailed in Algorithm 2.

The question now is how to train the encoder such that the learnt feature space is beneficial for NGU. The encoder f is trained via the inverse model h where two successive observations o_t and o_{t+1} are mapped through the encoder to their representations $f(o_t)$ and $f(o_{t+1})$ respectively, shown in Fig. 8. The two embeddings are concatenated into $[f(o_t), f(o_{t+1})] \in \mathbb{R}^{2d_e}$ and fed to the inverse model classifier h which predicts the action a used to transition from observation o_t to observation o_{t+1}, *i.e.* $\hat{p}(a|f(o_t), f(o_{t+1}))$. Cross entropy loss L is used to learn the parameters of both the classifier and the encoder, in particular $L = \mathbb{E}\left[-\sum_{i=1}^{|A|} \mathbb{1}(a_t = A_i) \ln(\hat{a_t})\right]$. The dimensionality of the learnt space d_e is much smaller compared to the dimensionality of the input observations, consequently, the encoder has to get rid of useless information and keep only relevant ones needed to predict the action. In most environments, modelling the position of the agent is enough to predict the correct action used to transition between observations.

3.4 Causal Perspective of Exploration

Although NGU with inverse model achieved state-of-the-art results on the atari suite, the inverse model is very limited in its modeling capacity since modeling the agent only is sufficient to predict the action. As a result, the encoder drops useful information which can help solve the environment more efficiently such as the different objects and irregular events. Indeed, aspects like objects under agent control are usually essential in getting rewards. We argue that the inverse model work well with NGU in atari [BSV$^+$20] mainly because most atari environments have limited number of interacting objects and just knowing the agents place is enough for proper exploration.

Tang *et al.* [CGM$^+$18] used CNNs with spatial attention maps to detect contingent regions in

an observations. They used the attention map to infer the agent location and use this information to formulate an intrinsic reward. Despite their approach being promising in providing additional useful information through the attention map, they use action prediction to learn these attention maps resulting in the same issue as the inverse model where the encoder drops many useful information. Corcoll *et al.* [CMV21] performed a deep analysis and showed the limitations of action prediction losses in modeling important aspects of the environment. In this work, we extend their analysis and conclusions to problems in reinforcement learning.

As we stated earlier action prediction by inverse models results in modeling only the agent location disregarding other important aspects of the environment. An alternate to modeling the agent is to model aspects which can be controlled by the agent such as objects and doors. In order to identify the controllable effects on the environment, *i.e.* changes caused mainly by the agent, counterfactual worlds [P$^+$00] were proposed.

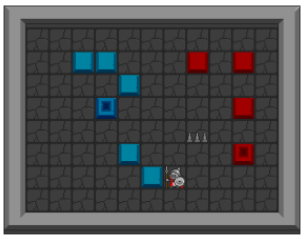


Figure 9. **Example for controllable effects.**

Counterfactual worlds states that an effect is controllable if the change in the environment would have been different if the agent performed a different action. Unfortunately, this is not very helpful since it results in all changes being controllable. For example, Fig. 9 shows an agent standing next to a box and doing nothing, so the box remains in its place. According to the definition of counterfactual worlds, the effect where the box does not move is controllable since the agent could go left and move the box. An alternate approach is to assign blame to actions performed by the agent.

To assign blame to an action, humans compare the consequences of an action with a mental image of a normal world. If the world changed from the normal version, humans assign a causal relation, *i.e.* blame, to the action they performed. If the world stay the same, then the action had no controlled effect on the environment. Back to Figure 9 example, in the normal world, the box stays in its place. If the agent perform the action "do nothing", the box will stay in its place similar to the normal world and no blame will be assigned to the agent action (do nothing). On the other hand, if the agent move left, the box will move as well. The resulting effect on the environment is different from the normal world, as a result, the agent should blame the action "move left" for the motion of the box.

Formally, counterfactual worlds identifies the causal relation between random variable X and random variable Y by measuring the Individual Causal Effect (ICE) of $X = x$ on Y. ICE is defined as,

$$ICE_Y^x \equiv Y^x \neq Y^{\tilde{x}} \tag{3.9}$$

That is the ICE of $X = x$ on Y can be measured by comparing the value of Y when $X = x$ and its value when $X = \tilde{x}$, where $\tilde{x} \in X - x$.

In the context of reinforcement learning, X and Y represent the actions and states/observations, respectively. Usually an action a causes a change to the state s which is then reflected in the observation o, but since we do not have access to internal states in most real problems, we will consider Y to be the observations. We will then define the effect of an action on the environment

as $e_p^a = o_{t+1} - ot$, where p stands for perceived change. Subsequently, Eq. 3.9 can be written as,

$$ICE_e^a \equiv e_p^a \neq e_p^{\tilde{a}} \tag{3.10}$$

So for an action a to cause a perceived effect e_p, the ICE_e^a should not be zero. Unfortunately, in the discussion earlier about counterfactual worlds, we showed that this definition of causal relation make everything trivially controllable by the agent, even when the agent perform the action "do nothing".

To include blame, we define the ICE_Y^x in terms of normal worlds. Instead of comparing the world where $X = x$ happens to all worlds where $X \neq x$, we compare it to the normal world. Accordingly, the ICE_Y^x is now defined as,

$$ICE_Y^x = Y^x - \beta_Y \tag{3.11}$$

where β_Y is the normal world which depends on the choice of normality, we will elaborate next on how to define normality in a useful way. Note here that we made the definition more specific by defining the ICE_Y^x as a difference relation. This specificity is essential for defining the normal world and for calculating the causal effect.

In order to define the normal world in a reinforcement learning context, we follow the same ideas in [SB18] where Sutton and Barto state that generalized value functions, which is an expectation over all actions, integrate general knowledge of the world. So, we define our normal world as an expectation over all actions $\tilde{a}$. Thereby, Equation 3.11 becomes,

$$\begin{aligned} ICE_{e_p}^a &= e_p^a - \beta_{e_p} \\ &= e_p^a - \mathop{\mathbb{E}}_{\forall \tilde{a} \in A} \left[e_p^{\tilde{a}} \right] \end{aligned} \tag{3.12}$$

Equation 3.12 reflects the intuition we built in the discussion about blame and how to define controlled effects. In short, we define the causal effect of action a as the difference between the effect observed in the world e_p^a after doing action a and the normal world β_{e_p} which is defined as the average observed effects corresponding to performing all other actions $\hat{a} \in \tilde{a}$. In the next section we present CEN, our approach to approximate the controlled perceived effects and normal perceived effects, and how to use CEN to promote better exploration.

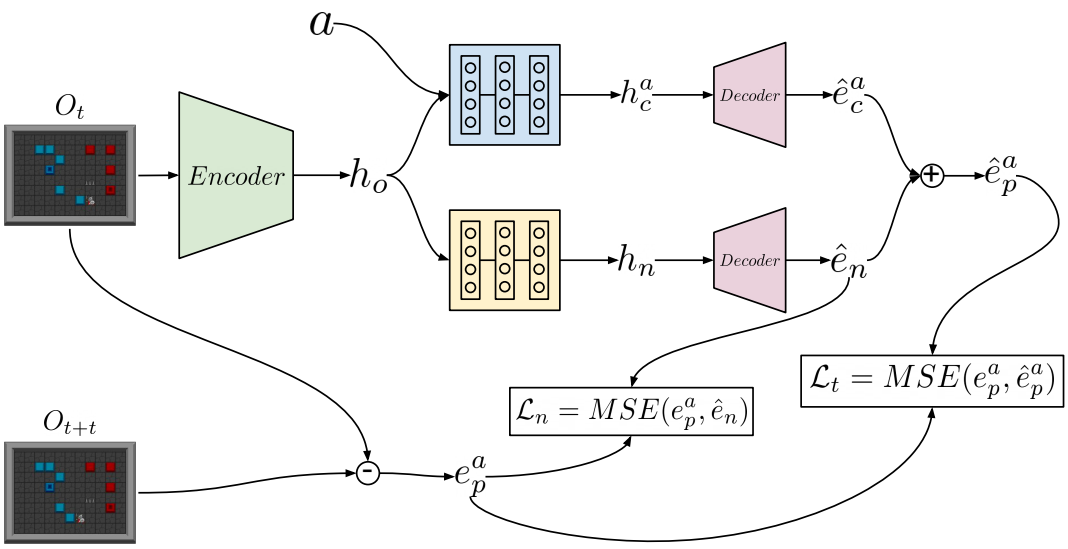


Figure 10. **Controlled Effects Network (CEN).** The encoder maps observations into a latent representation which is process by the two branches of the Latent Processing Layers. One branch transform the latent to model the controlled effects (Top), while the other model the normal effects (Bottom). The decoder maps the latents back to the observation space. The normal loss $\mathcal{L}_n$ teaches the normal branch to predict the expected normal world. While the total loss $\mathcal{L}_t$ encourages the controlled branch to model the controlled effects.

4 Methodology

4.1 Self-Supervised Learning of Controlled effects

We defined in Equation 3.12 the causal effect e_c^a, to simplify notation let $e_c^a = ICE_{e_p}^a$, of an action a on the environment. Unfortunately, generating the normal world is infeasible in most practical cases, for example, in model-free RL we do not have access to a model of the environment, so we can not observe the effect of applying every other action $\hat{a} \in \tilde{a}$ and take the average effect. Even if a perfect environment model is accessible, we still can not measure the normal world if we have a continuous action space or a very huge discrete action space (computationally infeasible). To solve this problem, we propose a self-supervised method that approximates the normal world using deep neural networks and disentangles the controlled effects e_c^a from normal ones e_n, where $e_n = \beta_{e_p}$ to simplify notation.

We have two main requirements for the encoder function. Firstly, it needs to disentangle the causal effect of an action from the normal effect. Secondly, it should not require extra data or information not accessible to the agent normally. Hence, we present **Controlled Effects Network (CEN)** to fulfill the two desiderata.

CEN, illustrated in Fig.10 is a special auto-encoder designed and trained to produce an

encoder which maps observations to learnt representations that exhibit the two aforementioned requirements. CEN consists of an encoder, Latent Processing Layers (LPL), and finally a decoder. The encoder takes an observation O_t as input and produce a corresponding latent representation $p(h_o|o, \boldsymbol{\theta_e})$.

Next, the latent processing layers convert the latent h_o into two representations h_n and h_c^a corresponding to the normal and controlled latent representations, respectively. LPL has two branches of feed-forward layers, each of them producing a transformed version of h_o. The first branch, named *Normal Branch*, transforms the input latent h_o into h_n without any extra input, *i.e.* it calculates $p(h_n|h_o, \boldsymbol{\theta_n})$. On the other hand, the second branch, named *Controlled Branch*, takes as input both h_o and a to produce the controlled latent representation h_c^a, *i.e.* $p(h_c^a|h_o, a, \boldsymbol{\theta_c})$.

Finally, the decoder takes as input a latent representations and outputs an effect similar in dimensionality to the input observations. We define the true ground-truth effect to be $e_p^a = O_{t+1} - O_t$. Since we have two latent representations, h_n and h_c^a, the decoder produces two effects $\hat{e}_n$ and $\hat{e}_c^a$. The two effects are added to give an approximation to the true effect e_p^a, *i.e.* $\hat{e}_p^a = \hat{e}_n + \hat{e}_c^a$.

To teach the normal branch to produce the average normal effect of the world, we use MSE loss

$$\mathcal{L}_n = \frac{1}{N}\left[\sum_1^N (\hat{e}_n - e_p^a)^2\right] \tag{4.1}$$

Where N is the number of samples in a given batch. Intuitively, the normal branch along with the decoder are trained to produce the true effect without access to the action. Consequently, the optimal behavior is to output an effect which achieve the smallest error irrespective of the action performed which converges to the normal world as defined in Eq. 3.12.

We add another MSE loss on the total predicted effect,

$$\mathcal{L}_t = \frac{1}{N}\left[\sum_1^N (\hat{e}_p^a - e_p^a)^2\right] \tag{4.2}$$

Combining the two losses gives,

$$\mathcal{L} = \mathcal{L}_t + \alpha\mathcal{L}_n \tag{4.3}$$

where α is a parameter to balance the magnitude of the two losses. The default value is $\alpha = 0.01$.

The final loss $\mathcal{L}$ forces the controlled branch to output the modifications needed to fix the normal branch produced effect $\hat{e}_n$ in order to have a total predicted effect $\hat{e}_p^a$ as close as possible to the true effect e_p^a. As a result, the controlled branch learns to focus on the controlled aspects of the environment, such as objects. To see why, note the agent in Fig. 9 has 5 actions, move up, left, down, right, or do nothing. The box will only move if the agent move left. In all the other cases, the box will remain in its place. Accordingly, in an optimal scenario, the normal branch should not model the box moving since, in four out of five cases, it does not move, so the loss will be greater if the predicted effect $\hat{e}_n$ models the box moving. Contrarily, the controlled branch has access to the action performed, so it should model the box moving in its predicted effect $\hat{e}_c^a$. By adding the two effects, the final loss is minimized if the normal and controlled branches model the normal and controlled effects, respectively.

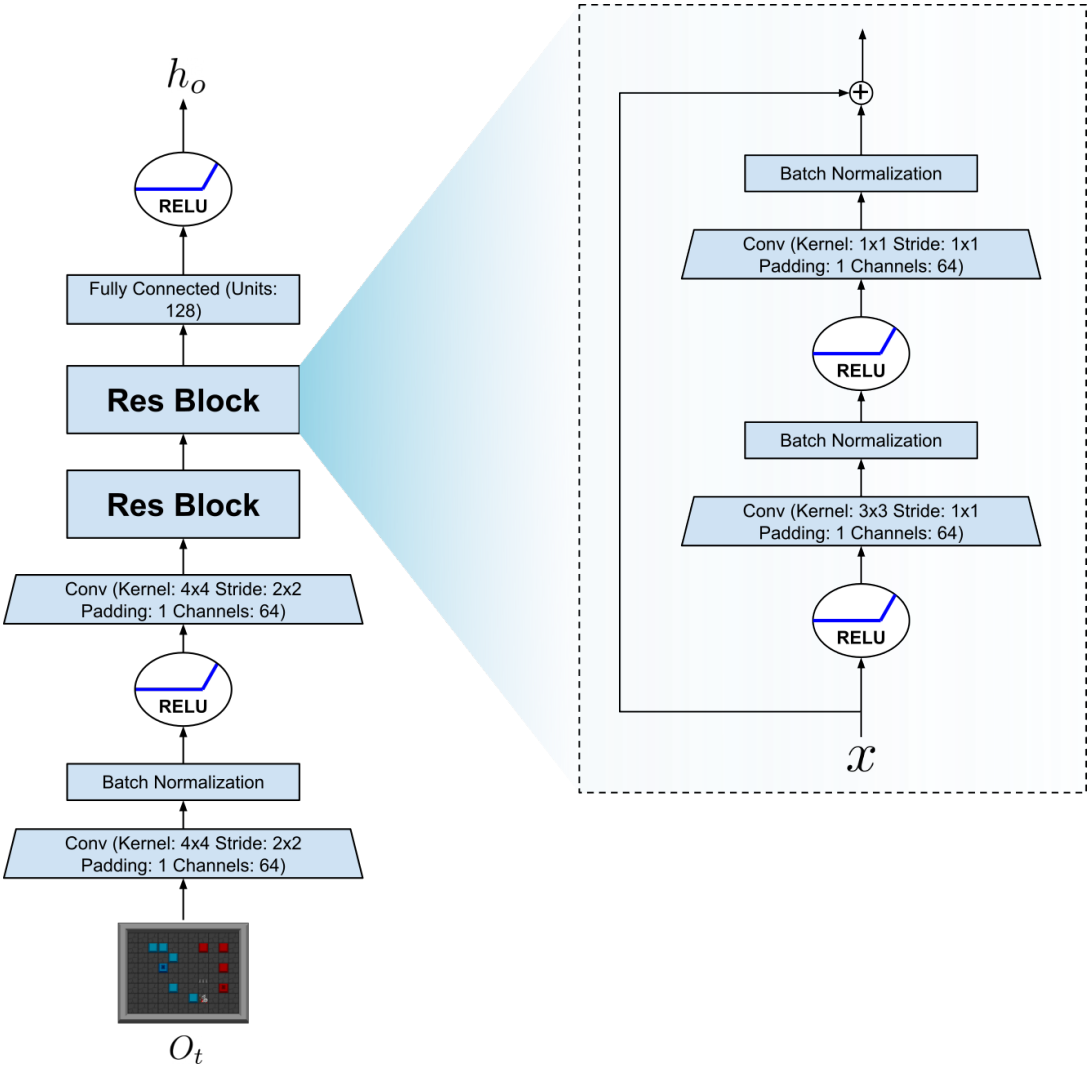


Figure 11. **CEN Encoder.**

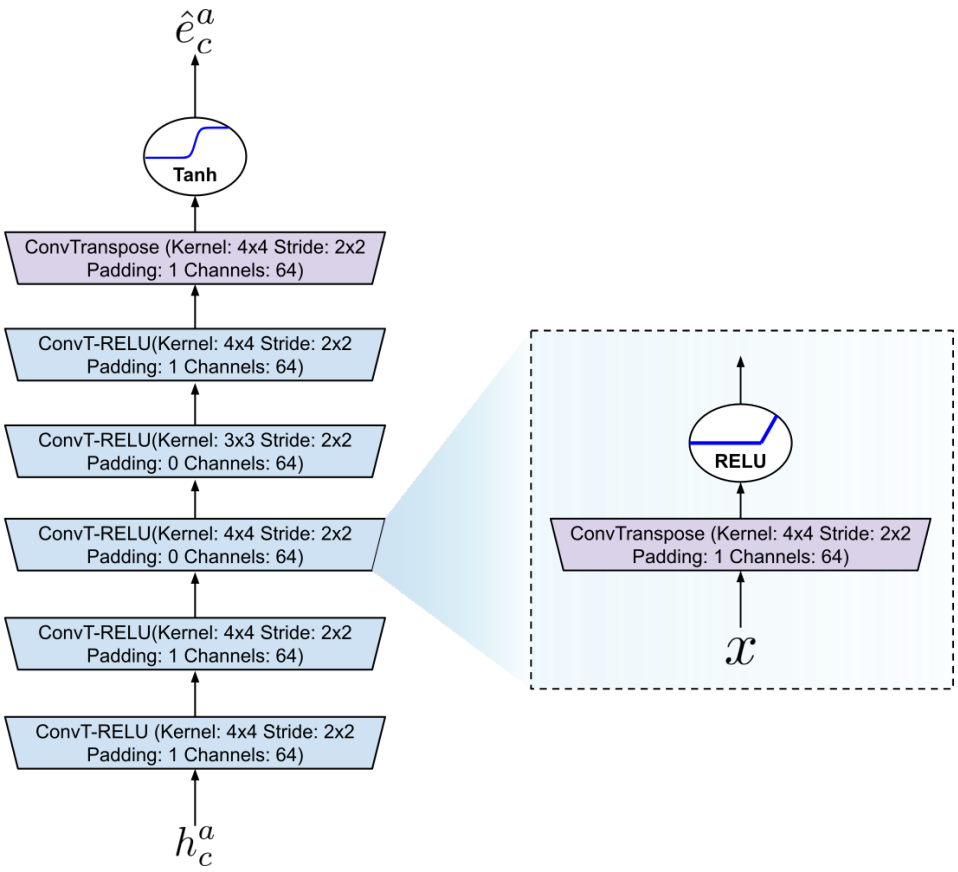


Figure 12. **CEN Decoder.**

The encoder, depicted in Fig. 11, is composed of regular convolutional, batch normalization, and fully connected layers. We use RELU [Aga18] as our activation function. We use residual blocks to improve the representations produced by the encoder. We included the default hyper-parameters we used in Fig. 11, we will state any deviations from the default parameters when needed.

As for the decoder, shown in Fig. 12, we used ConvTranspose layer [2] from Pytorch [PGM+19] that pads the input feature map to make the output's spatial dimensions bigger. It then applies normal convolutional operation to transform the input. We use RELU as activation for all intermediate layers, and hyperbolic tan function as activation for the last layer.

Lastly, for the latent processing units, we keep it simple and use three fully connected layers in each branch. We use RELU activation between the layers. For the controlled branch, we use an embedding layer to convert the action into a feature vector before concatenating it to the input latent representation. The action embedding layer applies an affine transformation to the input action, where the parameters of the weight matrix are learned via backpropagation.

[2] https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

4.2 CEN Agent

In section 4.1, we presented our embedding network, CEN, and how to train it. In this section, we integrate **CEN** with an agent similar to NGU presented in Section 3.3. CEN is mainly responsible for providing the embeddings used to calculate the intrinsic reward. We have already shown CEN's capability of modeling controlled effects which makes the latent representation suitable for exploring aspects of the environment controlled by the agent, such as moving objects.

Our CEN agent consists of a base RL agent augmented with an intrinsic reward module composed of an episodic memory and CEN. For the base RL agent, we require several attributes, it should:

- Provide strong baseline performance.
- Not be very complex containing several modules and engineering tricks.
- Be less sensitive to hyperparameters.
- Allow distributed training to make best use of available resources.
- Allow for adding recurrent layers to maintain a history of states and event.

The first three attributes are essential to allow a fair comparison between using *CEN* and *Inverse model*, named *Inverse* for simplicity, in the intrinsic reward module. Having a strong baseline performance is important since a trivial model will not fully exploit the intrinsic reward module. While being simple with a few modules would not bias the exploration in unexpected ways that can favor either CEN or Inverse. Taiga *et al.* [TFM+21] argued that very complex agents like rainbow DQN [HMVH+18] and similarly R2D2 [KOQ+18] implicitly encourage curious exploration. Finally, being less sensitive to hyperparameters simplifies comparing CEN and Inverse with less worry about hyperparameter tuning.

Distributed training has been a key factor in the performance gains of recent reinforcement learning algorithms [KOQ+18, ABC+20, BMHB+18, BESK18, ESM+18, HQB+18, SHM+16] because it allows for faster training and more hyperparameter tuning. In our case, distributed training allows us to do more extensive experiments.

Lastly, since we provide the agent with an intrinsic reward based on curiosity, the agent needs to keep track of the history of visitation to different states. For example, if the agent visits a state for the first time, the intrinsic reward will be high. However, the more it visits the state, the less the reward it gets. If the agent policy network can not model this history, it might lead to a degradation in performance. Using recurrent layers in the base RL agent allows the agent to keep track of its history.

Accordingly, we base our agent on Proximal Policy Optimization (PPO), introduced in Section 3.1. PPO is an actor-critic method. It has many advantages matching the desiderata mentioned above. It is also simple to implement, allowing for easy adoption and modifications. We use a shared CNN and LSTM backbone with two heads: the first for approximating the advantage and the second as a policy. The architecture of the CNN backbone is shown in Fig. 13.

We create multiple parallel workers, each running a copy of the environment initialized with a different seed. Meanwhile, we have a master training worker where the main neural network

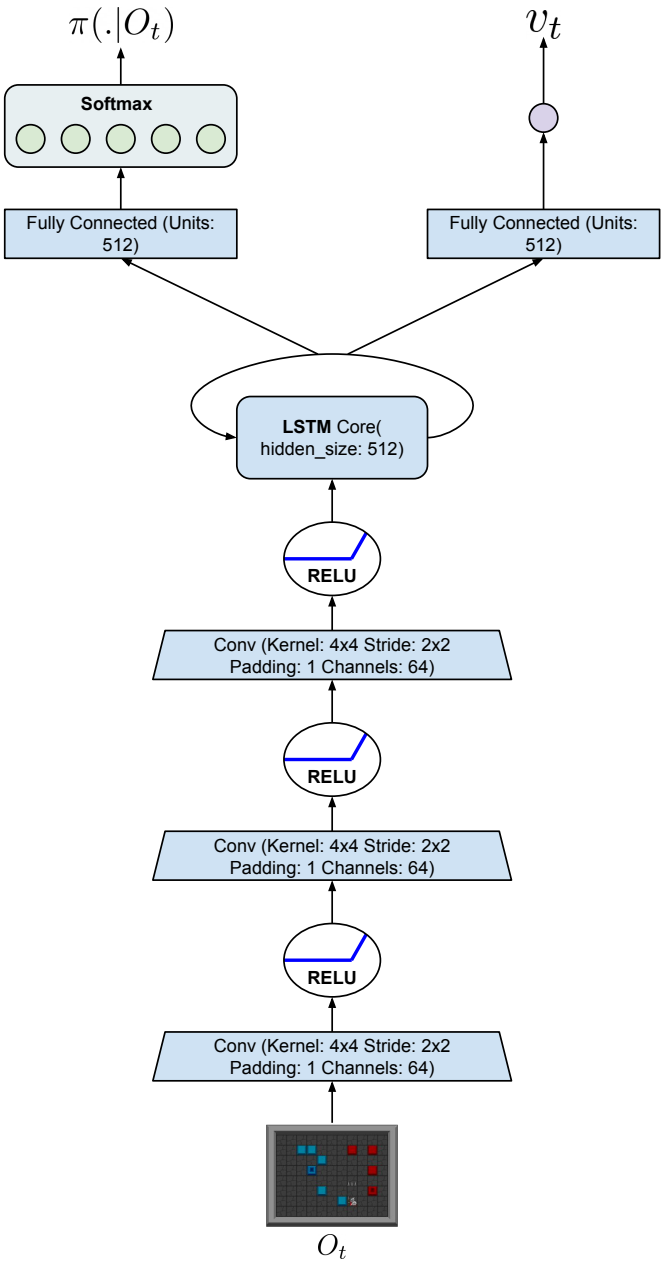


Figure 13. **PPO Model Architecture.**

is maintained and trained. After the workers run a predefined number of steps (roll-out), the master worker uses the collected trajectories to train the main network. The transitions in the collected trajectories are augmented with advantage values calculated using the value branch in the backbone network. The network parameters are then updated using Adam [KB14] optimizer with gradients of losses elaborated in Section 3.1. Our implementation is based on the PFRL package [FNKI21].

We integrate the intrinsic loss explicitly by adding it to the extrinsic reward returned by the environment on each transition,

$$r_t = r_t^e + \beta r_t^i \tag{4.4}$$

where r_t^e, r_t^i are the extrinsic and intrinsic rewards, respectively, and β is a weight to control the relative significance of the intrinsic reward compared to the extrinsic one. The intrinsic reward is calculated using Algorithm 2 with the help of an episodic memory per worker.

CEN is used to provide the embeddings for observations before appending them to the episodic memory, *i.e.* $f(o_t)$ in Algorithm 2. Since we want the embeddings to model the controlled aspects of the environment, we use the latents h_c^a from Fig. 10. Per our previous discussion, these latents h_c^a are used to generate the controlled effects of the corresponding transition. Therefore they need to "know" the location of objects and other controlled aspects of the environment. The intuition is that the controlled latents will collapse uncontrolled observations into a common representation while keeping the representations for interactions with different objects unique since each object should be uniquely represented. On the other hand, if we use an encoder trained using an inverse model, it will not model any object since there is no need for the objects to predict the agent's action. As a result, it will collapse similar representations into fewer modes regardless of the objects and the agent's interaction with them.
The full algorithm of PPO with CEN based intrinsic reward module can be found in Algorithm 3

Algorithm 3: PPO Agent with CEN intrinsic reward module

Initialize N workers with randomly seeded environments;
Create an episodic memory for each worker;
foreach *episode in range(max_episodes)* **do**
    $obs_t \leftarrow$ Reset the N environments;
    Reset the N episodic memories;
    foreach t *in range(trajectory_length)* **do**
        foreach *worker in workers* **do**
            Sample action a_t using PPO policy $\pi(.|obs_t, \boldsymbol{\theta})$;
            $obs_{t+1}, r^e_{t+1} \leftarrow$ Apply action a_t to the environment;
            Use CEN encoder to get the embedding $h_c^{a_t}$ corresponding to obs_t, a_t;
            $r^i_{t+1} \leftarrow$ Calculate Intrinsic reward according to Algorithm 2;
            Append the embedding $h_c^{a_t}$ to the worker's episodic memory;
            $r_{t+1} = r^e_{t+1} + \beta r^i_{t+1}$;
            Append the transition $(obs_t, a_t, r_{t+1}, obs_{t+1})$ to the worker's trajectory;
        end
        Update the parameters of CEN network according to the scheduler using Adam optimizer;
    end
    Add the advantage of each transition to the transitions in each trajectory;
    `/* The advantage is calculated using the value head of the PPO network */`
    Update the parameters of PPO network using Adam optimizer;
end

5 Experiments and Results

The main goal of our experiments is to show how **CEN** (our model) improves the exploration of RL agents compared to an **Inverse** model. To this end, we compare two versions of our proposed agent: the first with CEN as the encoder for the intrinsic reward module, while the second uses an Inverse model as the encoder for the intrinsic reward module.

Our experiments are designed to highlight the importance of modeling the controlled aspects of the environment in the embeddings of the intrinsic reward module, as well as the advantages of bonus-based exploration in hard exploration environments. Hence, we choose environments that sparsely reward the agent and have objects that are indispensable to solving the environment. Moreover, we create custom environments to investigate specific questions raised through our experiments. For example, we compare the exploration quality of CEN and Inverse in an empty grid with no reward to better understand their differences.

We start by describing our experimental setup in section 5.1. We then divide the rest of the results based on the research question that we aim to answer. We provide a brief description of every environment and what aspects it evaluates. MiniGrid is presented in section 5.2, followed by Griddly's Clusters in section 5.3, and finally, task-specific environments in section 5.4, where each environment is created to test a certain exploration aspect.

5.1 Experimental Setup

Vanilla Agent Our baseline is a vanilla PPO agent with no intrinsic reward module. The architecture is illustrated in Figure 13, while the algorithm is the same as Alg. 3 without the intrinsic reward steps. We use this baseline to highlight the difficulty of the environments and how bonus-based exploration is essential for reasonable learning.

CEN Agent This is our proposed architecture. We mentioned the implementation details in Section 4.2. To summarize, we add to the vanilla PPO an intrinsic reward module that uses our CEN encoder. We hypothesize that this intrinsic reward module should encourage the agent to interact more with objects while exploring the environment leading to better sample efficiency.

Inverse Agent We add the intrinsic reward module from the NGU agent [BSV$^+$20] to a vanilla PPO agent. This intrinsic reward module uses an encoder trained via inverse dynamics prediction. Since the encoder is trained for an action prediction task, it models the agent only, ignoring the other controlled aspects of the environment. As a result, the Inverse intrinsic reward module does not reward new interactions with objects since the Inverse encoder collapses their representations leading to less interactive exploration.

Observation Pre-processing We train our agents in environments where the observations are RGB images. We resize the input observation to have a shape of 84×84 before feeding it to the CNN. We chose to keep the RGB channels in contrast to the standard practice [MKS$^+$13] of converting observations into grayscale and stacking them because we use some environments where color information is essential in achieving the goal.

Name	Value	Sweep
hidden size	32	[16, 32, 64, 128]
latent size	128	[16, 32, 64, 128, 256]
channels	64	[16, 32, 64, 128]
learning rate	0.0001	[0.0001, 0.0005, 0.001, 0.005]
α	0.01	[0.001, 0.01, 0.1, 1, 5, 10, 20, 30, 50]
T	0.01	-

Table 2. CEN hyperparameter sweeps and final values used.

Name	Value	Sweep
encoder channels	32	[32, 64]
encoder hidden	32	[16, 32, 64, 128]
latent size	128	[64, 128, 256]
learning rate	0.0001	[0.0001, 0.0005, 0.001, 0.005]

Table 3. Inverse model hyperparameter sweeps and final values used.

Name	Value	Sweep
batch size	512	[64, 128, 512, 1024]
latent size	32	[8, 16, 32]
CEN encoder output size	128	[16, 32, 64, 128]
learning rate	0.0005	[0.00005, 0.0001, 0.0002, 0.0005, 0.001]
IR Beta	0.001	[0.0001, 0.001, 0.002, 0.005, 0.01, 0.1]
Rollout size	2048	[1024, 2048, 4096]
discount	0.95	-
epochs	10	-

Table 4. PPO hyperparameter sweeps and final values used.

Random Seeds Reinforcement learning is notorious for the high variance of its results and their dependency on the random seed used [HIB+18, Pic21]. To mitigate this, we run every agent on each environment at least with 3 different seeds. We also plot the standard error in the results to show the variance in the results making the comparisons more meaningful.

We implemented the three agents with the help of the PFRL package [FNKI21], PyTorch [PGM+19], and NumPy [HMvdW+20]. We used Weights and Biases (WandB) [Bie20] for experiment monitoring and visualizations. We open source [3] our implementation for reproducing the experiments. To the best of our knowledge, this is the first open-source implementation of the episodic novelty module from Never Give Up [BSV+20]. The most important hyperparameters are included in Tables 2, 3, 4. The rest is provided as a JSON file in the GitHub repository to save space. We train our models on a single RTX2080Ti GPU machine.

5.2 What are the advantages of Intrinsic Motivators?

Environment Minimalistic Gridworld (MiniGrid) [CBWP18] is a simple, easy-to-use, and lightweight package to create different grid-like environments with minimal graphics. It is used extensively to test Reinforcement Learning methods [PDPG21, WWS21, ZLL+21, GLH+19]. It allows for easy customization in order to test certain behavior of RL agents. We mainly use the Key-Door environment, shown in Fig. 14; there are 16×16 grid cells, and the agent receives the whole grid as an observation. The agent, the red triangle, must collect the yellow key, then open the door, and finally reach the green square goal in order to get a reward of 1. Clearly, this environment is very sparse and difficult to solve using vanilla PPO. We compare the three agents to show which model learns to solve the environment in the fewest number of environment steps, *i.e.* which agent is the most sample efficient?

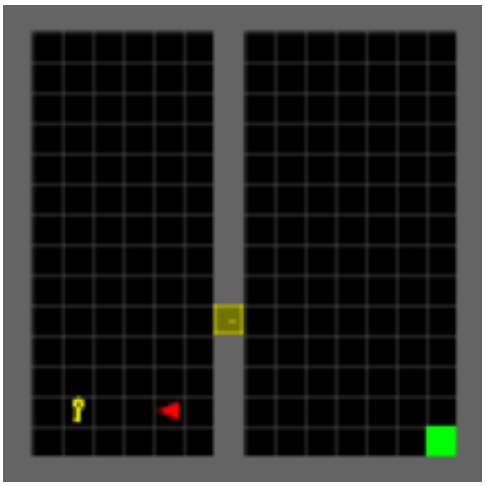


Figure 14. **Minigrid Key Door Environment.** The agent should get the key, use it to open the door, and reach the green goal.

Results We show the results of training the three agents on the minigrid key-door environment in Fig. 15. The results of CEN and Inverse agents were almost identical, so we decided to group them and name them NGU. We use three different seeds per agent and show the standard error as a measure of variance.

[3] https://github.com/Mo-youssef/controllable-IR

Figure 15. **MiniGrid Results.** We plot the reward value (max=1) on the y-axis and the number of environment steps on the x-axis. We group CEN and Inverse Agents and name them NGU because their performance is almost identical. NGU is significantly more efficient compared to Vanilla PPO reflecting the advantages of using the intrinsic reward module.

NGU vs. Vanilla Agents NGU agents utilizing the intrinsic reward module are significantly more efficient compared to the vanilla PPO. They solve the environment and achieve a reward of 1 with less than $1M$ environment steps, while it takes Vanilla agents more than $6M$ steps. Vanilla's results have higher variance reflecting a large dependency on the initial seed due to the environment's sparsity. The IR module makes both NGU agents more robust to the initial seed by encouraging more useful interactions.

CEN vs. Inverse Agents We observed no difference between CEN and Inverse Agents. The Key-Door environment is simple when it comes to the number of objects and events interacting with them. Both "Picking up the key" and "Opening the Door" events are equally novel to both CEN and Inverse, making them assign similar intrinsic rewards to these events. Hence, there is no significant difference between their performance. Our results on Clusters (Sec.5.3) support this claim. We observe a difference in performance between CEN and Inverse Agents due to Clusters' object density.

5.3 Is CEN better than Inverse in Object-Dense Environments?

Environment Griddly [Bam21] is a package for creating environments to test RL agents. It is similar to MiniGrid, where the environments have a grid-like structure. However, it has more advanced graphics, and it allows for more complex dynamics. It is suitable to test our agent in more complex environments. We use the environment **Clusters**, depicted in Fig. 5.3, because it has many controlled objects, *i.e.* boxes. The abundance of boxes should highlight the advantage of CEN over the Inverse model. Clusters have two sets of boxes and blocks of different colors. The blocks are hollow and fixed in place, while the boxes have solid colors and can be pushed by the agent. The goal is to push boxes to blocks of similar color. Once a box is pushed against a block, the box turns into a block and gets fixed in place.

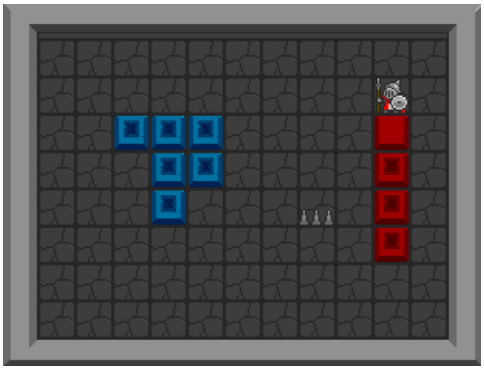


Figure 16. **Griddly Clusters.** The agent has to push all the solid color boxes to the hollow blocks converting the boxes to blocks. The agent is rewarded only after pushing all the boxes of a certain color. The reward is proportional to the number of boxes converted to blocks.

The default Clusters environment is not sparse since it provides a reward each time a box is pushed into a block. Hence, we changed the default dynamics and rewarded the agent only when all the boxes of a single color are converted into blocks, *i.e.* pushed into existing blocks. This change created a problem where a box gets stuck into the wall, and the environment becomes unsolvable. To alleviate this problem, we introduced another change where we removed the boxes stuck in the wall. To encourage the agent to push more boxes towards blocks, we scaled the reward by the number of converted boxes. The agent receives the reward upon converting the boxes of a single color to blocks, *i.e.* in Fig. 5.3 the agent is rewarded 5 for the blue boxes and will be rewarded 3 after pushing the red box down to the red block.

Results We show the results of training the three agents on Griddly Clusters in Fig. 17. Clearly, the sparse Clusters environment is very challenging to the vanilla PPO agent, which could not solve the environment even after 5 million steps. On the other hand, the Inverse agent managed to get some reward by pushing a single box on average against the block, showing that the Inverse model does not encourage interaction with controlled objects. Finally, our model CEN outperforms the other models by a wide margin, revealing its ability to promote interactive events with different objects leading to better exploration in object-rich environments such as Clusters.

Embedding Network Evaluation The reported results, in Fig. 17, might be caused by poor training of the embedding network in the Inverse agent. To investigate further, we evaluate both

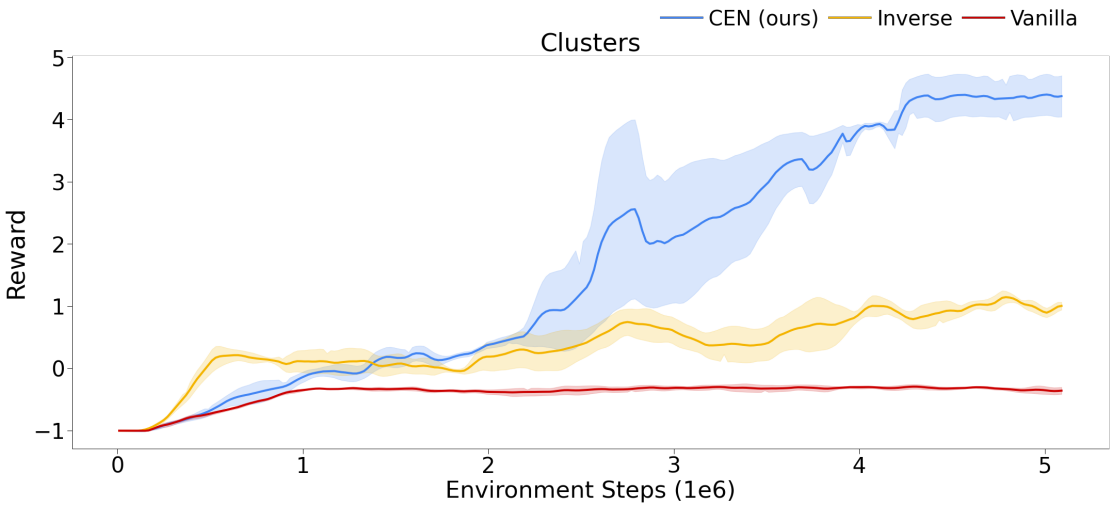
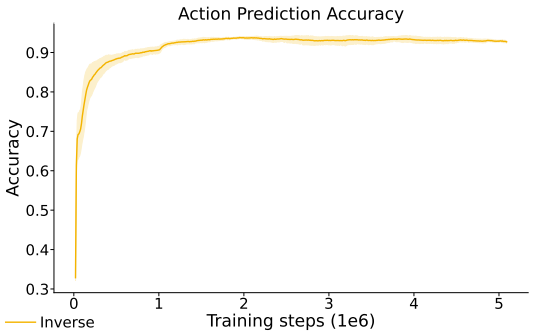
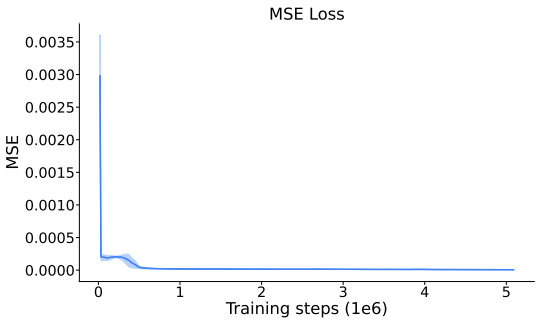


Figure 17. **Clusters results.** We plot the reward on the y-axis against the environment steps on the x-axis. Each agent is run for 3 different seeds. CEN significantly outperforms Inverse. Vanilla can not solve the environment due to its sparsity.



(a) **Clusters - Inverse Accuracy.** The action prediction accuracy is plotted against the environment steps. The Inverse model learns how to predict the action correctly reaching $\sim 90\%$ very early. This indicates that the performance gain of CEN is not due to poor training of Inverse model.



(b) **Clusters - CEN MSE Loss.** The MSE between the predicted and true total effect. CEN learns quickly to generate the total effect indicating proper training.

the Inverse and CEN embedding networks. In particular, we show the Inverse's action prediction accuracy and CEN's MSE loss between predicted and true total effects in Figs. 18a and 18b, respectively. Evidently, both networks are performing well, reflecting that the difference in reward is not due to poor embedding network training. We showcase some videos of the agents solving the environment in the GitHub repository [4]

5.4 How Do CEN and Inverse Agents explore Empty Environments?

Environment We created an *Empty Grid*, using Griddly, in order to study the exploration behavior of CEN compared to Inverse agents. We provide no extrinsic reward and use only the intrinsic reward. We let each agent run freely in the environment and train the agent for $2e6$ environment steps.

Results The CEN agent explores its environment much differently from the Inverse agent. We notice that CEN, in Fig. 19, exhibits a more uniform exploration where the agent regularly visits different cells in the grid. Inverse, on the other hand, tend to stick to corners and walls. It seems to care less about exploring different cells. These behaviors can be explained through the embedding network in each agent and what they model.

The Inverse embedding network predicts the action taken to move between two frames. Therefore, predicting the action when the agent is in a corner or beside a wall becomes more difficult because multiple actions lead to the same result. Since the embedding network's modeling capacity is limited, the representations for non-corner cells get collapsed into similar ones to assign more capacity to the near-corner cases. As a result, the embeddings of the near-corner frames become more different. In response, the IR module assigns a higher reward for these cases since the reward is inversely proportional to the embeddings' similarity.

On the other hand, CEN's embedding network can mitigate this problem by having access to the action. The network needs to model only the location of the agent in order to predict its next location using the input action. Thus, the corners do not hold any special value and are thus ignored in the embeddings. As a result, the IR module assigns a similar reward to all the cells promoting more uniform exploration.

[4] https://github.com/Mo-youssef/controllable-IR

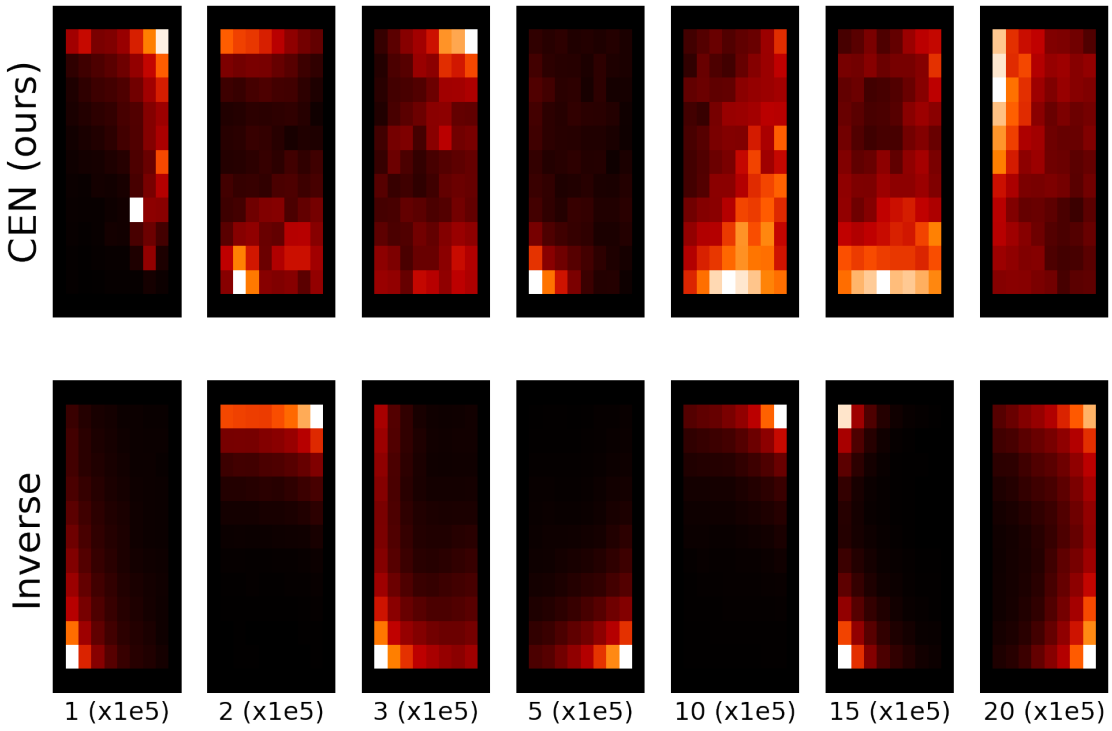


Figure 19. **Empty Grid results.** We plot the reward on the y-axis against the environment steps on the x-axis. Each agent is run for 3 different seeds. CEN significantly outperforms Inverse. Vanilla can not solve the environment due to its sparsity.

6 Conclusion

In this work, we introduced CEN, a self-supervised model to learn controlled aspects of RL environments. We showed how CEN compares the current observed world to the normal world to identify the controlled effects. We used CEN as the embedding network in an NGU intrinsic motivation module. We augmented a PPO agent with this module and analyzed its exploration behavior in carefully designed environments. We created these environments to be sparse, *i.e.* provide few extrinsic rewards, and dense, *i.e.* contain many objects. These environments exposed the need to model controlled aspects in the intrinsic reward module to solve them efficiently.

CEN-based agent outperforms the inverse dynamics agent by a wide margin in the sparse Clusters environment. CEN agent achieved a score ~ 4 after 4 million environment steps, maxing out the score at ~ 5. On the other hand, the Inverse agent achieved only ~ 0.5 after the same 4 million steps, eventually setting at a max score of ~ 1. Moreover, we compared the exploration behavior of CEN and Inverse agents in an empty grid. CEN exhibits a more uniform exploration, visiting many states. On the other hand, Inverse has a hard time exploring the entire grid since it tends to stick to the walls and corners of the grid. We hypothesize that Inverse collapses the different non-corner agent's locations into similar representations different from the representations for the corner case due to the high loss from the action prediction task. CEN does not work with action prediction, hence it does not suffer from this limitation and explores more uniformly.

Although CEN identifies controlled aspects, it is less efficient to train compared to inverse dynamics models. More efficient auto-encoder networks can be proposed that perform similar to CEN. Another limitation is collecting the data to train CEN. Ideally, observations from a random policy are needed to train CEN. But, in practical applications, CEN is trained using observations collected from a trained policy, limiting the diversity of the observations. Modifications to the policy can allow the agent to collect more diverse observations resulting in better CEN training.

Causal inference can provide valuable tools to address fundamental problems in Reinforcement Learning. The temporal dependence of actions and events in an episode raises many causal questions. Exploiting such causal relations can lead to more efficient RL. This work highlights one advantage of adopting causal concepts to the exploration problem. Credit Assignment and Model-based RL are two domains that can benefit directly from the causality literature. Credit assignment is concerned with identifying actions or events that result in rewards, a direct application of causality. Model-based RL can adopt causal models to accurately predict the future of an environment resulting in better planning and decision making.

References

[ABC+20] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[Aga18] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[Bam21] Chris Bamford. Griddly: A platform for ai research in games. *Software Impacts*, 8:100066, 2021.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BESK18] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[Bie20] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[BMHB+18] Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

[BSO+16] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

[BSV+20] Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

[BVB12] Marc G Bellemare, Joel Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[CBWP18] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[CGM+18] Jongwook Choi, Yijie Guo, Marcin Moczulski, Junhyuk Oh, Neal Wu, Mohammad Norouzi, and Honglak Lee. Contingency-aware exploration in reinforcement learning. *arXiv preprint arXiv:1811.01483*, 2018.

[CKG+19] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[CKNH20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[CMV21] Oriol Corcoll, Youssef Mohamed, and Raul Vicente. Did i do that? blame as a means to identify controlled effects in reinforcement learning. *arXiv preprint arXiv:2106.00266*, 2021.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[ESM+18] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[FNKI21] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14, 2021.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GLH+19] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[GSA+20] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[HBT+21] Wei-Ning Hsu, Benjamin Bolte, Yao-Hung Hubert Tsai, Kushal Lakhotia, Ruslan Salakhutdinov, and Abdelrahman Mohamed. Hubert: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3451–3460, 2021.

[HCX+22] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[HIB+18] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[HMvdW+20] Charles R Harris, K Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

[HMVH+18] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[HQB+18] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KOQ+18] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*, 2018.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[KSK21] Wonjae Kim, Bokyung Son, and Ildoo Kim. Vilt: Vision-and-language transformer without convolution or region supervision. In *International Conference on Machine Learning*, pages 5583–5594. PMLR, 2021.

[KVB88] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[LB+95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LBD+89] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[MBLD92] Ofer Matan, Christopher JC Burges, Yann LeCun, and John S Denker. Multi-digit recognition using a space displacement neural network. In *Advances in neural information processing systems*, pages 488–495, 1992.

[MBM+16] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MU49] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

[OBOM17] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. Count-based exploration with neural density models. In *International conference on machine learning*, pages 2721–2730. PMLR, 2017.

[P+00] Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: Cambridge-UniversityPress*, 19, 2000.

[PAED17] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2778–2787. PMLR, 06–11 Aug 2017.

[PDPG21] Simone Parisi, Victoria Dean, Deepak Pathak, and Abhinav Gupta. Interesting object, curious agent: Learning task-agnostic exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

[Ped18] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task. *arXiv preprint arXiv:1804.02763*, 2018.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[Pic21] David Picard. Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*, 2021.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SHM+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[SIVA17] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[SLA+15] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[SQAS15] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[SSPS21] David Silver, Satinder Singh, Doina Precup, and Richard S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[TFM+21] Adrien Ali Taïga, William Fedus, Marlos C. Machado, Aaron Courville, and Marc G. Bellemare. On bonus based exploration methods in the arcade learning environment. *International Conference on Learning Representations*, 2021.

[Wer90] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[Wil92] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[WWS21] Akifumi Wachi, Yunyue Wei, and Yanan Sui. Safe policy optimization with local generalized linear function approximations. *Advances in Neural Information Processing Systems*, 34, 2021.

[WZ89] Ronald J Williams and David Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection science*, 1(1):87–111, 1989.

[XBK+15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[ZLL+21] Mingde Zhao, Zhen Liu, Sitao Luan, Shuyuan Zhang, Doina Precup, and Yoshua Bengio. A consciousness-inspired planning agent for model-based reinforcement learning. *arXiv preprint arXiv:2106.02097*, 2021.

Appendix

I. Paper

This paper was published in Transactions on Machine Learning Research (TMLR) in July 2022

Did I do that? Blame as a means to identify controlled effects in reinforcement learning

Oriol Corcoll *oriol.corcoll.andreu@ut.ee*
Institute of Computer Science
University of Tartu

Youssef Mohamed *youssef.mohamed@ut.ee*
Institute of Computer Science
University of Tartu

Raul Vicente *raul.vicente.zafra@ut.ee*
Institute of Computer Science
University of Tartu

Reviewed on OpenReview: *https://openreview.net/forum?id=NL2L3XjVFx*

Abstract

Affordance learning is a crucial ability of intelligent agents. This ability relies on understanding the different ways the environment can be controlled. Approaches encouraging RL agents to model controllable aspects of their environment have repeatedly achieved state-of-the-art results. Despite their success, these approaches have only been studied using generic tasks as a proxy but have not yet been evaluated in isolation. In this work, we study the problem of identifying controlled effects from a causal perspective. Humans compare counterfactual outcomes to assign a degree of blame to their actions. Following this idea, we propose Controlled Effect Network (CEN), a self-supervised method based on the causal concept of blame. CEN is evaluated in a wide range of environments against two state-of-the-art models, showing that it precisely identifies controlled effects.

1 Introduction

The recent success of reinforcement learning (RL) methods in complex environments such as Hide & Seek (Baker et al., 2019), StarCraft II (Vinyals et al., 2019), or Dota2 (OpenAI et al., 2019) has shown the potential of RL to learn complex behavior. Unfortunately, these methods also show RL's inefficiency to learn (Espeholt et al., 2018; Kapturowski et al., 2019; Gulcehre et al., 2020), requiring a vast amount of interactions with the environment before meaningful learning occurs. Consequently, environments with sparse rewards are known to be extremely difficult, making imperative a good exploration strategy. A popular approach to exploration is to introduce behavioral biases in the form of intrinsic motivators (Chentanez et al., 2005; Mohamed & Rezende, 2015). This technique aims to facilitate the learning of task-agnostic behavior by producing dense rewards, driving the agent to discover novel states and by doing so increase the chance of discovering the environment's reward.

Numerous motivators have been developed taking inspiration from humans, e.g. curiosity or control (Bellemare et al., 2012b; Pathak et al., 2017; Burda et al., 2018; Choi et al., 2019; Badia et al., 2020b). Recent work (Choi et al., 2019; Song et al., 2019; Badia et al., 2020a;b) has achieved State-of-the-Art on the Atari benchmark (Bellemare et al., 2012a) by rewarding agents for the discovery of novel ways to control their environment. A typical design principle among these methods is using an inverse dynamics model to predict the chosen action from two consecutive observations with the hope that the latent representation learned encloses aspects of the environment controlled by the agent. This approach assumes that the action can be recovered from consecutive observations, which is not always the case.

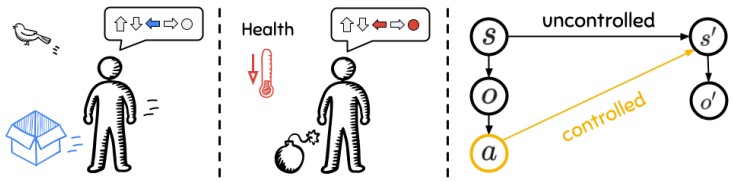


Figure 1: Left) Blame compares an imagined normative world to what actually happened to attribute the movement of the agent and the box to the action. Middle) Using a do-nothing action as normative world is not enough since do-nothing has an effect. Right) Controlled effects are the effects of an action on the environment.

A more causal approach is to compare counterfactual worlds (Pearl, 2009), i.e., an effect is controllable if the effect would have been different had the agent taken another action. A caveat of this approach is that things become trivially controllable. Fig. 1 (left) shows an scenario where an agent moves a box by moving left. Here the box becomes controllable even when the agent performs the action "do-nothing" since there is an action, "move-left", that would move the box. Contrarily, it is believed that humans identify controlled effects by assigning a degree of blame to their actions. In particular, humans compare what happened to what normally would happen (Halpern, 2016; Morris et al., 2018; Langenhoff et al., 2019; Grinfeld et al., 2020). If what happened is normal, humans would not assign blame to their actions, e.g. when performing "do-nothing", the box's effect would not be controlled since normally the box would not move. However, it would be considered controlled when moving left since its normative state is to not move.

This work proposes Controlled Effects Network (CEN), a completely unsupervised approach to identify controlled aspects of the environment based on the human notion of blame. CEN can be incorporated to any forward model and works by dividing its latent representation into two branches, a normal and a controlled view of the world. Our experiments show that CEN can disentangle effects precisely, outperforming state-of-the-art approaches to detect controlled effects.

2 Identifying controlled effects using blame

Our goal is to identify changes in the environment that were controlled by the agent. This section introduces Individual Causal Effect (ICE), a fundamental measure in causal literature, and frames it in the context of RL. We show how this measure can be used to identify controllable effects but argue that these are not suitable for RL. In contrast, the human perception of causality is associated with the concept of blame (Gerstenberg & Lagnado, 2014). For example, if lightning hits a forest tree and starts a fire, humans would point to the lightning as the cause of the fire, not the oxygen or wood since they are normally present in the forest. Consequently, we expand the idea of blame to identify controlled effects by using measures of normality and counterfactuals.

2.1 Controllable effects

What does it mean to cause something? Pearl et al. (2016) provide an intuitive definition of cause-effect relations: "A variable X is a cause of a variable Y if Y, in any way, relies on X for its value". This kind of formulation tries to answer questions like "does smoking causes cancer?". Actual causality, proposed in Halpern (2016), studies causal relations between individual events of X and Y. It aims to answer questions like, "did smoking for 30 years caused David's cancer?". The individual nature of Actual causality makes this framework especially suited to RL problems. In the following, we introduce the concept of causal effect in the context of RL.

The individual causal effect (ICE) of an event $X = x$ on a variable Y_i can be measured by comparing counterfactual worlds

$$ICE^x_{Y_i} \equiv Y_i^x - Y_i^{\tilde{x}} \ , \tag{1}$$

where Y_i^x reads as "what would the value of an individual Y_i be if X is forced to be x". Similarly, $Y_i^{\tilde{x}}$ describes the value of Y_i when X is forced to not be x. Note that the sub-index i refers to an individual, and hence in the following, we use Y_i^x and Y^x interchangeably.

The *fundamental problem of causal inference* states that we can only observe one of these counterfactual worlds and the other needs to be imagined. Intuitively, Eq. 1 compares the world where the event x happened to an alternative world where event x had not happened. Consequently, we say that x has a causal effect on Y if there is an $\tilde{x} \in X$ that makes Eq. 1 nonzero. In the context of RL, X and Y take the form of actions, states and/or observations. Fig. 1 (right) illustrates the causal relations present in a typical RL setting, where a state s has an effect on both the next state s' and the produced observation o which, in turn, has an effect on the agent's choice of action $a \in \mathcal{A}$. Similarly, an action has an effect on the next state. Since states are typically not accessible by the agent, we do not use states as variables; nevertheless the same principle can be applied. We define the perceived effect e_p^a as the difference between consecutive observations when taking action a, i.e. $e_p^a \equiv o' - o$. As in Eq. 1, we say that a perceived effect is controllable by the agent's action when

$$\exists \tilde{a} \in \mathcal{A} \colon \left(e_p^a - e_p^{\tilde{a}}\right) \neq 0 \ . \tag{2}$$

Since we want to know what elements of the perceived effect are controllable, the difference is an element-wise operation. It is important to notice that Eq. 2 has far-reaching consequences, for example, an agent next to a box would have a causal effect on it even when doing nothing since there is a counterfactual world where that box would have moved. If we use Eq. 2 as reward, the agent would be rewarded for almost every action at every state! Note that taking $\tilde{a}$ as a special "do-nothing" action would not work since even doing nothing does something, e.g., Fig. 1 (middle) shows a scenario where doing nothing has an effect on the agent's health. Taking do-nothing as $\tilde{a}$ would not attribute the effect to the agent. Instead, we would want a more human-like definition of what is controlled where an agent controls a box if moved or its life if a bomb could have been easily avoided.

2.2 Blame

It has been shown that the human notion of causality is affected by what is normal (Kahneman & Miller, 1986; Cushman et al., 2008; Knobe & Fraser, 2008; Hitchcock & Knobe, 2009). Here, we resort to concepts of normality from actual causality to find if the agent's action is to blame for what happened. Halpern & Hitchcock (2014) propose to compare what actually happened with what normally would happen. Following this idea we build a normative world to replace $Y^{\tilde{x}}$ in Eq. 1

$$ICE_Y^x = Y^x - \beta_Y \ , \tag{3}$$

where β_Y is the value Y would normally take. Such a value is of course contingent to the notion of normality used, which is for us to define. We can reformulate Eq. 3 to compute the controlled effect of an action e_c^a as

$$\begin{aligned} e_c^a = ICE_{e_p}^a &= e_p^a - \beta_{e_p} \\ &= e_p^a - \mathop{\mathbb{E}}_{\tilde{a},o'}\left[e_p^{\tilde{a}}\right] \ , \end{aligned} \tag{4}$$

where β_{e_p} represents what normally would have happened to the environment. This quantity not only depends on the environment's dynamics but also on the agent's typical behavior.

In this work, we compute β_{e_p} as the expectation over all possible futures o' of each action in $\mathcal{A}$. Note that stochastic environments may have be multiple next observations for each action. To simplify notation, the following sections use normal effect as $e_n = \mathbb{E}_{\tilde{a},o'}\left[e_p^{\tilde{a}}\right]$. Intuitively, Eq. 4 builds a normal world by observing every alternative e_p produced by each action creating an average perceived effect. Consider the example in Fig. 1 (middle), moving left or doing nothing would make the agent's health decrease. Eq. 4 would indicate that what is normal is for health to not change since in average health stays the same; thus, the loss of health when moving left or staying would be attributed to the agent. On the other hand, moving right would only attribute the change in the agent's position as controlled. Note that the explosion would never be credited to the agent since it would have happened no matter what action is taken.

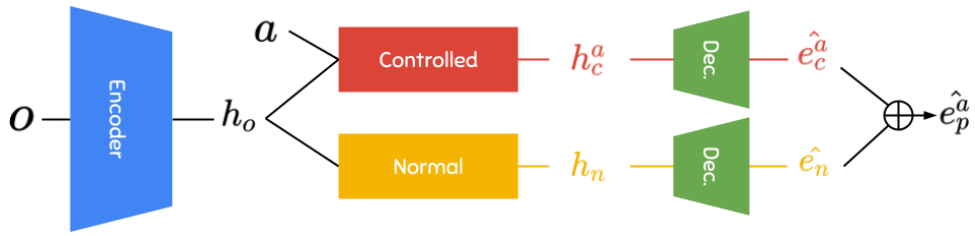


Figure 2: CEN divides the latent space of a forward model into controlled and normal branches. Each branch disentangles controlled and normal effects and decodes each into pixel space independently.

Special care needs to be taken when constructing the normal world β_{e_p} for continuous or large action spaces. Computing counterfactuals on an infinite number of possibilities cannot be done and some approximation needs to be implemented. Although our experiments use discrete actions, the proposed method in the following section is equipped to handle continuous action spaces since it does not compute counterfactuals for each possible action but approximates the normal world directly. It is also important to notice that the controlled effects that Eq. 4 can identify in a partially observable setting ($o \neq s$) are constrained to those observed by the agent. Nevertheless, humans cannot perceive every change in state but can identify relevant controlled effects for their survival and joy.

3 Unsupervised learning of controlled effects

In practice, we do not have access to every world and cannot compute Eq. 4 directly. We propose **Controlled Effects Network (CEN)**, an unsupervised method depicted in Fig. 2 that disentangles controlled and normal effects only using perceived effects as a self-supervised training signal. CEN is modeled as a neural network and it is based on a forward model, where observation and action are used to predict the outcome of performing such action on the environment. In contrast to conventional forward models, CEN divides its latent space into controlled and normal representations; similarly to Dueling Networks (Wang et al., 2015). These two representations approximate the controlled and normal effects in latent space. A decoder converts these latent representations into pixel space allowing to estimate $e_c^a + e_n = e_p^a$ as in Eq. 4.

The **controlled branch** has privileged access to the action, consequently, having only this branch would make CEN a regular forward model, i.e., the controlled branch alone can predict the perceived effect resulting from the action. Then, why do we need the **normal branch**? The role of the normal branch is to force the controlled branch to predict only what is not predictable from the observation alone and hence, modeling what is controlled by the agent. In a way, the normal branch acts as a distillation mechanism where only what can be controlled will be represented by the controlled branch. To promote the controlled branch to model only controlled effects, we use the following objective

$$\mathcal{L} = \text{MSE}\left(\hat{e_c^a} + \hat{e_n}, e_p^a\right) + \alpha \text{ MSE}\left(\hat{e_n}, e_p^a\right), \tag{5}$$

where the first term is the reconstruction loss in which the predicted target is compared to the perceived effects provided by the environment. The second part of the loss enforces the network to use the normal branch to model the world as much as possible. Since this branch cannot predict everything without the action the model will converge to the expected effect due to the MSE loss. Additionally, a hyperparameter α regulates how much the normal branch should model the environment. In practice, we found that this hyperparameter creates an agreement between branches on uncertain futures which seemed to be critical in environments with stochastic entities. Note that we combine normal and controlled effects in pixel space, this is because we would like to know the pixels that are controlled by the agent. Nevertheless, if the downstream task CEN is part of only needs the latent representation, the combination of controlled and normal representations could be done in latent space i.e. $\hat{h_c^a} + \hat{h_n}$.

Let us look again at the example in Fig. 1 (middle) and assume the agent picks the do-nothing action. The normal branch is encouraged to model the bomb since it does not depend on the action. Furthermore, it

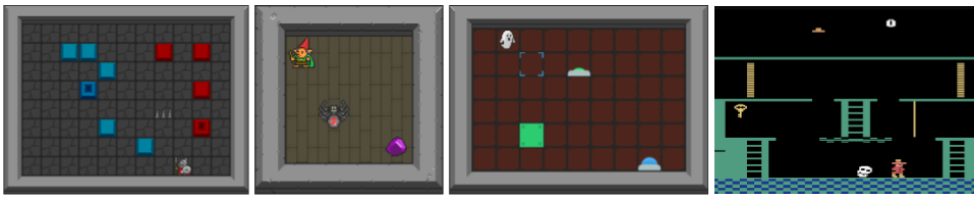


Figure 3: Suite of environments used for the experiments. From left/right top/bottom: Clusters, Lights, Spiders and Montezuma's Revenge (MZR).

should not predict any change in health since what is normal is for health to not change. Thus, the controlled branch must model the change in health.

4 Experiments

This section evaluates CEN[1] on the following main questions: 1) Can CEN identify controlled effects at pixel-level? i.e. can it produce an accurate segmentation mask? 2) Some applications may not require pixel-level precision; we asses CEN on predicting controlled effects at attribute-level from both pixel masks and latent representations. Every experiment reports a 95% confidence interval[2] using three different seeds.

Environments: we use multiple environments (Fig. 3) to answer the above questions, each showcasing a different aspect of what can be controlled. These environments are based on Griddly (Bamford et al., 2020) and Atari ALE (Bellemare et al., 2012a), both using the Gym interface (Brockman et al., 2016). More details of these environments are given in each experiment and appendix.

Baselines: We use Attentive Dynamics Model (ADM) (Choi et al., 2019) in experiments 1) and 2). ADM is an action-prediction model based on an spatial attention mechanism. It works by predicting the action performed on individual image patches. Then, a spatial attention mechanism selects a sparse set of patches to use when making a final prediction of the action. The masks produced by the attention mechanism are considered controlled aspects of the environment. Although ADM does not produce pixel level masks, only at patch level, to our knowledge ADM is the most competitive method to provide pixel-level information about controlled aspects. For 2) we rely on the inverse modeled proposed in Never Give Up (NGU) (Badia et al., 2020b), the current SOTA for exploration in RL.

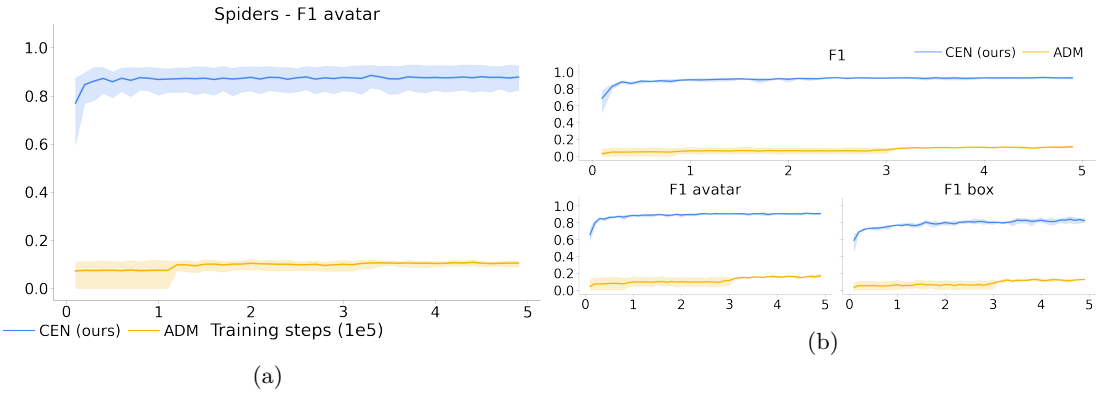
Implementation: CEN is implemented as an encoder-decoder architecture with 2D convolutional layers and ReLU activation functions; the normal and controlled branches are implemented with linear layers. Additionally decoder weights are shared. Throughout the experiments we use the same neural networks and hyperparameters unless specified otherwise. Our implementation of ADM uses the same architecture and hyperparameters proposed in Choi et al. (2019). See appendix for more details on the architecture and hyperparameters.

4.1 Controlled effects at pixel-level

This set of experiments explores CEN's ability to identify pixels corresponding to controlled entities. Although CEN computes the magnitude and direction of the effects, we create a binary mask by setting a threshold for the predicted controlled effects (see exact details in appendix D). We report pixel F1 scores between ground truth and predicted binary masks. We explain how ground truth masks are computed below, but it should be clear that CEN is a fully unsupervised method that does not use this ground truth in any way, ground truth is only used to evaluate the produced masks. The network is trained to minimize Eq. 5 using the ADAM optimizer (Kingma & Ba, 2015) and 500K samples of the form (o, a, e_p^a) collected using a random policy.

[1]Networks, training and evaluation have been implemented using PyTorch (Paszke et al., 2019), NumPy (Harris et al., 2020) and PFRL (Fujita et al., 2021); our experiments are managed using W&B (Biewald, 2020).

[2]CI are computed using bootstrap resampling as per the Seaborn (Waskom, 2021) package



(a) (b)

Figure 4: a) CEN can correctly disentangle the agent from the randomly moving objects. b) Clusters environment where CEN is able to model not just the agent but also the movement of boxes.

Ground truth masks: Griddly provides access to each entity's state (x, y, light is on/off, etc); a binary mask is produced for each controlled entity with any of its attributes changed when transitioning between two time steps. x and y coordinates are projected into pixel space and a bounding box is generated using the size of that entity. The resulting masks for each entity are combined into a single mask m by taking the maximum value among them. Since we want to know what pixels were actually controlled, the final ground truth mask is produced as: $m \cdot e^p_a$.

In case of MZR, Atari ALE allows us to actually compute counterfactual worlds by saving and loading the state of the game (RAM) multiple times when taking different actions. For this, we directly compute Eq. 4 using ALE's special calls *cloneSystemState* and *restoreSystemState*. More precisely, we compute every possible perceived effect reachable from the current state and build a normal effect using the mode over all possible effects. Then, we compare the perceived effect for the agent's chosen action against the normal effect. The ground truth mask will have 1s where these two effects are different.

CEN and ADM masks: CEN's controlled masks are generated using the encoder, controlled branch and decoder. The predicted controlled effect is binarized using a threshold T as $(-T < \hat{e^a_c})|(\hat{e^a_c}c > T)$. In the case of ADM, its attention mask is thresholded in the same way and the mask is resized to the size of the effect.

4.1.1 Controlled vs uncontrolled effects

Here we use the Spiders environment to evaluate CEN's ability to disentangle controlled from uncontrolled effects. This environment has two main entities, the agent and a spider. The controlled masks must only focus on the agent and ignore the spider.

Fig. 4a shows the pixel F1 score for our model and the baseline. CEN is able to correctly disentangle controlled effects and can produce accurate masks. Although our implementation of ADM can predict the agent's action with 88% accuracy, it is not capable of modeling the agent's controlled pixels. We conjecture that this is due to ADM's sparse softmax mechanism; nonetheless this behavior persisted when increasing its entropy weight which should produce more dense masks.

4.1.2 Nearby controlled effects

Models based on action prediction are expected to work well on aspects related to the agent. For example, if an agent moves a box due to moving right; the box's movement is also controlled. It is unclear why these models would pay attention to the box since just knowing where the agent is, suffices to predict the chosen action. CEN's controlled branch, on the other hand, is motivated to model the box's effect since the normal branch would predict that the box stays where it is. We call "nearby" controlled effect to an effect that happens adjacent the agent, like the box's movement. To evaluate CEN on this kind of effects we use the Clusters environment where an agent needs to move colored boxes to their corresponding fixed colored blocks.

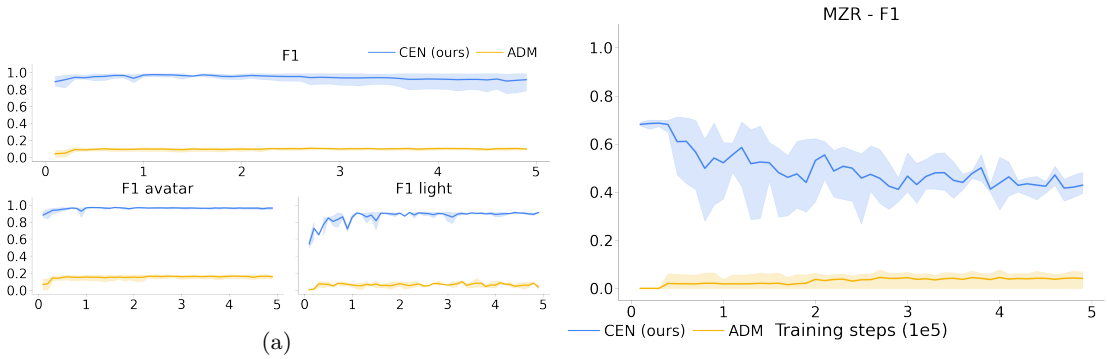


Figure 5: a) F1 score on the Lights, and b) MZR environments. CEN outperforms the baseline even in an environment with more complex dynamics and features.

Fig. 4b shows that CEN can precisely model effects on the agent and boxes. We breakdown individual effects to account for the class imbalance between the agent's movement and the boxes. CEN seems to make more mistakes with boxes than the agent but nonetheless, it can consistently model both.

4.1.3 Faraway controlled effects

In contrast to the previous experiment, we want to evaluate if CEN can model distant effects, i.e. effects that are reasonably far away from the agent's location. In this case, we use the Lights environment. Here the environment presents two buttons of different color that, when pressed, turn on their corresponding light. Lights are relatively far away from their corresponding buttons thus making it difficult to model them. As show in Fig. 5a, CEN is able to model this kind of effects.

4.1.4 Controlled effects in Montezuma's Revenge

This last pixel-level experiment evaluates CEN on Montezuma's Revenge (MZR) environment. Although agents in Atari environments have limited control over the environment, the relatively complex dynamics of Montezuma's Revenge makes it a challenging test-bed. Results shown in Fig. 5b indicate that CEN can also model controlled effects.

Unlike previous environments, CEN's F1 score behaves differently due to two main reasons: 1) *F1 decreases over time:* the F1 score is extremely sensitive for empty ground truth, a single pixel marked as controlled takes the score from 1 to 0. At the beginning of training, CEN produces empty masks correctly but performs poorly at controlled effects, which makes the F1 score artificially high. Over time it improves at controlled effects but starts marking very few pixels as controlled for effects where nothing is controlled. 2) *F1 is lower:* the sensitivity of the F1 score also affects CEN's performance on controlled effects since pixels outside the ground truth are penalized the same no matter how close to the ground truth they are. Appendix B.3 includes additional masks showing that CEN fails by pixels close to the controlled object instead of identifying the wrong object as controlled.

4.2 Controlled effects at attribute-level

In some cases requiring pixel-level precision can be excessive. The following experiments analyze how different representations can predict effects on attributes from the environment's state, e.g. changes on the agent's (x, y) location or if a light turned on. To this end, we use a probing technique (Alain & Bengio, 2016) similar to the one described in Anand et al. (2019). This approach trains a classifier per each attribute of interest in the environment's state using frozen versions of trained networks to produce the classifiers inputs. More specifically we use two different sources for the probing classifiers, pixel-level masks (as in the previous section) or the model's latent representation.

		F1 Pixels		F1 Latent	
Environment	Attribute	CEN (ours)	ADM	CEN (ours)	Inverse
Spiders	Agent	1.0±0.00	0.47±0.23	0.97±0.01	0.67±0.05
	Spider ↓	0.35±0.03	0.25±0.03	0.41±0.01	0.44±0.02
Clusters	Agent	0.76±0.41	0.28±0.08	0.97±0.01	0.56±0.09
	Box	0.78±0.37	0.32±0.19	0.95±0.02	0.77±0.00
Lights	Agent	0.97±0.01	0.33±0.15	1.0±0.01	0.84±0.08
	Button	0.93±0.05	0.33±0.01	0.99±0.0	0.99±0.0
	Light	0.93±0.04	0.41±0.14	1.0±0.0	0.99±0.0
MZR	Agent	0.66±0.08	0.42±0.23	0.91±0.02	0.88±0.02
	Skull ↓	0.19±0.03	0.20±0.08	0.61±0.03	0.61±0.04

Table 1: F1 score when predicting attributes from pixel or latent space. Lower is better for Skull and Spider.

For the first case, we produce a binary mask, as in the previous experiments, to occlude perceived effects and use these to train each probing classifier. Classifiers have to predict if there was a positive, negative or none effect. Note that the classifiers need to predict any effect, not just controlled. Thus, the classifier should only be able to predict accurately controlled attributes such as the agent's position but should fail at predicting uncontrollable effects like the location of the spider or skull. We use a random policy to collect a dataset of 35K samples of the form $(m * e_p, y)$ where m is the mask produced by the model and y is the ground truth class. Each dataset is split into a typical 70/20/10, allowing a 20% class imbalance. We report F1 score of each attribute on the test set.

4.2.1 Attribute probing from pixels

The results in Table 1(first block) indicate that CEN consistently outperforms the baseline when predicting controlled effects for state's attributes, and thus modeling controlled effects accurately. Furthermore, for both Spiders and Montezuma's Revenge environments the model cannot predict the uncontrolled effects, as expected. Even though ADM's action prediction accuracy was high ($\sim 88\%$) on every environment, it is not able to consistently predict controlled effects at attribute-level.

4.2.2 Attribute probing from latent representations

In this case, we train classifiers using a latent representation instead of pixels. We use the latent representation from CEN's controlled branch (h_c). It is unclear how to create a latent representation from ADM, so we use an inverse model. The features of current and next observations are concatenated to create a latent

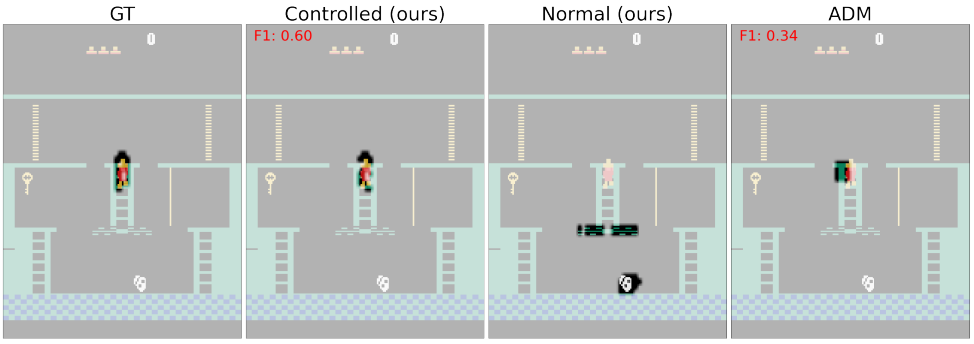
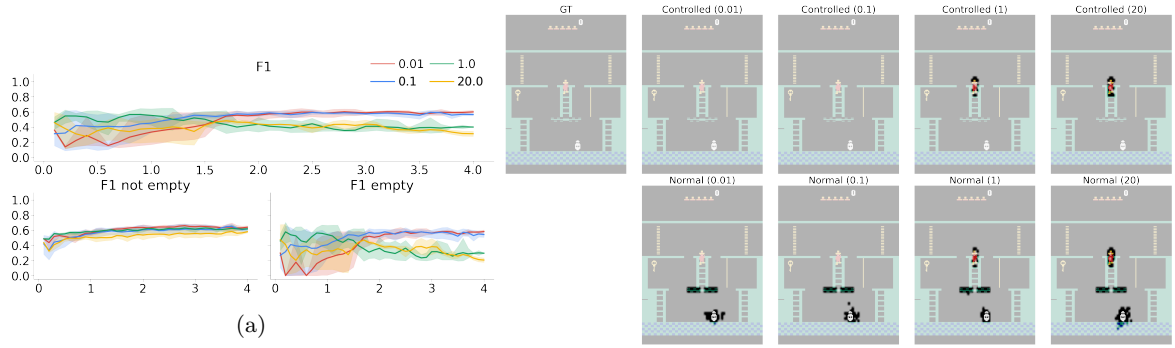


Figure 6: Example masks for Montezuma's Revenge. Additional masks are included in appendix B.3.



(a) (b)

Figure 7: Ablation study on the effect of α from Eq. 5 on the learned representations of the normal branch. X axis of a) indicate training steps (1e5)

representation of what is controlled. As before, we train a linear probe to predict controlled attributes from the latent space of these models. The probe predicts changes on the x, y and direction (if applies) for agent, spider, skull and boxes; and on/off for lights and buttons.

Table 1 (second block) show that CEN improves on the baseline's performance. Although the inverse model is closer to CEN's performance than ADM, it still has difficulties predicting the agent and box changes in location. The reason of having high score in Skull is that the Skull is still only when the agent dies, making it easy to predict from the agent's position. Removing this event leads to a score of ~ 0.35 for both models.

4.3 How does the alpha hyperparameter affects CEN?

Here we study how α affects the loss in Eq. 5. This experiment uses Montezuma's Revenge and the same setup as in Experiment 4.1. We analyze the effect of alpha on CEN using values ranging from 0.01 to 20.

As can be seen in Fig. 7a, the different alphas do not impact the performance when the ground truth or controlled effects are not empty. On the other hand, the performance degrades the higher the alpha. We hypothesize that this behavior happens when the normal branch is forced to model controlled effects too strongly and the controlled branch needs to counter those bad predictions. In this case, the controlled branch will produce wrong masks, especially for empty ground truth. Fig. 7b shows that the higher the α the more the controlled branch needs to remove pixels modeled by the normal branch.

5 Related work

Intrinsic motivators: A popular way of introducing behavioral biases in RL agents is the use of intrinsic motivators (Singh et al., 2005; Mohamed & Rezende, 2015). These motivators can promote different types of exploration, from observational surprise (Burda et al., 2018) to control seeking agents (Pathak et al., 2017; Choi et al., 2019). Methods in the latter category have shown extremely good results achieving SOTA in important benchmarks. Choi et al. (2019) proposed Attentive Dynamics Model (ADM), an attention based method that discovers controlled elements in the environment and rewards the agent for discovering them. This method and its extension (Song et al., 2019) showed SOTA in Montezuma's Revenge. Badia et al. (2020b) combined control and observational surprise to promote exploration. Their method uses an episodic memory with an inverse model to promote the discovery of controlled effects and Random Network Distillation (Burda et al., 2018) to promote long term progress; again achieving SOTA in Atari's hard exploration environments. These methods show the importance of identifying what an agent can control.

Causality in deep reinforcement learning: Causality is central to humans; we think in terms of cause-effect. A similar method to Blame was proposed in Chattopadhyay et al. (2019), where they use causal attribution methods to analyze the effect of inputs on a neural network's outputs. Recent work has introduced

causality into deep reinforcement learning (Schölkopf et al., 2021; Ke et al., 2021; Foerster et al., 2018; Buesing et al., 2018; Jaques et al., 2018; Dasgupta et al., 2019; Goyal et al., 2019; Nair et al., 2019; Madumal et al., 2020) showing that this is a promising avenue for the training of agents. Corcoll & Vicente (2020) proposed an attribution method to learn temporal abstractions for object-centric hierarchical RL. Bellemare et al. (2012b) compute controllable aspects of the environment by generating a mask with all possible controllable areas of an image and uses it as part of the policy's input. In this work, we identify the controlled effects of individual actions using causal concepts of normality and blame. A similar concept to normality has been explored in control theory by Todorov (2009) called "passive dynamics". Passive dynamics compute how a dynamical system would evolve without interventions.

6 Conclusions and limitations

This work provides a causal perspective to the problem of identifying controlled aspects of the environment and proposes Controlled Effect Network (CEN), a fully unsupervised approach to this problem. CEN creates a normative world using counterfactuals and compares what actually happened with what normally would happen to attribute changes on the environment to the agent. The presented experiments show that, despite being unsupervised, this method precisely identifies controlled effects. In future work, we will explore CEN as intrinsic motivator or as a way to discover skill in a hierarchical reinforcement learning setting.

6.1 Limitations

Normality: although we propose a measure of normality in Eq. 4, this is far from ideal. For example, the resulting normal world may not be real/possible under the environment dynamics, a more realistic solution would be to use the mode instead of expectation. This could be achieved by using a critic, as in GANs (Goodfellow et al., 2014). We believe the way humans see normality is context dependent and should be learned instead of a fixed function. This is an active research area in the field of psychology where researchers look to underpin human causal judgment.

State instead of observations: our current formulation of ICE uses perceived effects as opposed to states. When CEN indicates that some change is controlled this is not (necessarily) equivalent to stating that the objects represented by those pixels are controlled. An exciting avenue to explore is combining CEN with a state representation model (e.g. LSTM or RSSM by Hafner et al. (2019)).

Reliance on policy: since Eq. 4 does not depend on the policy, CEN needs diverse data for each action to approximate it, ideally from a random policy. This important problem is not specific to CEN, any forward model needs a policy that explores multiple actions to provide accurate predictions.

Multi-step controllability: CEN assumes that objects are controlled immediately after performing an action. A valuable extension is to identify the consequences of an action multiple steps ahead. Possibly by incorporating Blame to works like Mohamed & Rezende (2015); Gregor et al. (2016).

Acknowledgments

The authors would like to thank Jaan Aru and Daniel Majoral for insightful comments on the manuscript. This work was supported by the University of Tartu ASTRA Project PER ASPERA, financed by the European Regional Development Fund.

References

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes, 2016. URL `https://arxiv.org/abs/1610.01644`.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised State Representation Learning in Atari. 2019.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark, 2020a.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies, 2020b.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019.

Chris Bamford, Shengyi Huang, and Simon Lucas. Griddly: A platform for ai research in games, 2020.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, Vol. 47:253–279, 2012a. doi: 10.1613/jair.3912. URL http://arxiv.org/abs/1207.4708. cite arxiv:1207.4708.

Marc G. Bellemare, J. Veness, and Michael Bowling. Investigating contingency awareness using atari 2600 games. In *AAAI*, 2012b.

Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL http://arxiv.org/abs/1606.01540. cite arxiv:1606.01540.

Lars Buesing, Theophane Weber, Yori Zwols, Sebastien Racaniere, Arthur Guez, Jean-Baptiste Lespiau, and Nicolas Heess. Woulda, coulda, shoulda: Counterfactually-guided policy search, 2018.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov Openai. Exploration by Random Network Distillation. 2018.

Aditya Chattopadhyay, Piyushi Manupriya, Anirban Sarkar, and Vineeth N Balasubramanian. Neural network attributions: A causal perspective. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 981–990. PMLR, 09–15 Jun 2019. URL http://proceedings.mlr.press/v97/chattopadhyay19a.html.

Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. In L. Saul, Y. Weiss, and L. Bottou (eds.), *Advances in Neural Information Processing Systems*, volume 17, pp. 1281–1288. MIT Press, 2005. URL https://proceedings.neurips.cc/paper/2004/file/4be5a36cbaca8ab9d2066debfe4e65c1-Paper.pdf.

Jongwook Choi, Yijie Guo, Marcin Moczulski, Junhyuk Oh, Neal Wu, Mohammad Norouzi, and Honglak Lee. Contingency-Aware Exploration in Reinforcement Learning. *ICLR*, pp. 1–20, 2019.

Oriol Corcoll and Raul Vicente. Disentangling causal effects for hierarchical reinforcement learning, 2020.

Fiery Cushman, Joshua Knobe, and Walter Sinnott-Armstrong. Moral appraisals affect doing/allowing judgments. *Cognition*, 108(1):281–289, July 2008. doi: 10.1016/j.cognition.2008.02.005. URL https://doi.org/10.1016/j.cognition.2008.02.005.

Ishita Dasgupta, Jane Wang, Silvia Chiappa, Jovana Mitrovic, Pedro Ortega, David Raposo, Edward Hughes, Peter Battaglia, Matthew Botvinick, and Zeb Kurth-Nelson. Causal reasoning from meta-reinforcement learning, 2019.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018. URL https://ojs.aaai.org/index.php/AAAI/article/view/11794.

Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14, 2021. URL http://jmlr.org/papers/v22/20-376.html.

T. Gerstenberg and D. A. Lagnado. Attributing responsibility: Actual and counterfactual worlds. In Joshua Knobe, Tania Lombrozo, and Shaun Nichols (eds.), *Oxford Studies in Experimental Philosophy*, volume 1, pp. 91–130. Oxford University Press, 2014.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Anirudh Goyal, Shagun Sodhani, Jonathan Binas, Xue Bin Peng, Sergey Levine, and Yoshua Bengio. Reinforcement Learning with Competitive Ensembles of Information-Constrained Primitives. pp. 1–21, 2019.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control, 2016. URL https://arxiv.org/abs/1611.07507.

Guy Grinfeld, David Lagnado, Tobias Gerstenberg, James F. Woodward, and Marius Usher. Causal responsibility and robust causation. *Frontiers in Psychology*, 11:1069, 2020. ISSN 1664-1078. doi: 10.3389/fpsyg.2020.01069. URL https://www.frontiersin.org/article/10.3389/fpsyg.2020.01069.

Caglar Gulcehre, Tom Le Paine, Bobak Shahriari, Misha Denil, Matt Hoffman, Hubert Soyer, Richard Tanburn, Steven Kapturowski, Neil Rabinowitz, Duncan Williams, Gabriel Barth-Maron, Ziyu Wang, Nando de Freitas, and Worlds Team. Making efficient use of demonstrations to solve hard exploration problems. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SygKyeHKDH.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels, 2019.

Joseph Y. Halpern. *Actual Causality*. The MIT Press, 2016. ISBN 0262035022.

Joseph Y. Halpern and Christopher Hitchcock. Graded Causation and Defaults. *The British Journal for the Philosophy of Science*, 66(2):413–457, 04 2014. ISSN 0007-0882. doi: 10.1093/bjps/axt050.

Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'ıo, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Christopher Hitchcock and Joshua Knobe. Cause and norm. *Journal of Philosophy*, 106(11):587–612, 2009. doi: 10.5840/jphil20091061128.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro A. Ortega, DJ Strouse, Joel Z. Leibo, and Nando de Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning, 2018.

D. Kahneman and D. T. Miller. Norm theory - comparing reality to its alternatives. *Psychol Rev Psychol Rev*, 93(2):136–153, 1986.

Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lyTjAqYX.

Nan Rosemary Ke, Aniket Didolkar, Sarthak Mittal, Anirudh Goyal, Guillaume Lajoie, Stefan Bauer, Danilo Rezende, Yoshua Bengio, Michael Mozer, and Christopher Pal. Systematic evaluation of causal discovery in visual model based reinforcement learning, 2021. URL https://arxiv.org/abs/2107.00848.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Joshua Knobe and Benjamin Fraser. *Causal judgment and moral judgment: Two experiments*, pp. 441 – 447. The MIT Press, United States of America, 2008. ISBN 9780262195690.

Antonia F Langenhoff, Alex Wiegmann, Joseph Y Halpern, Joshua Tenenbaum, and Tobias Gerstenberg. Predicting responsibility judgments from dispositional inferences and causal attributions, Sep 2019. URL psyarxiv.com/63zvw.

Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Explainable reinforcement learning through a causal lens. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2493–2500, Apr. 2020. doi: 10.1609/aaai.v34i03.5631. URL https://ojs.aaai.org/index.php/AAAI/article/view/5631.

Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning, 2015.

Adam Morris, Jonathan S Phillips, Thomas Icard, Joshua Knobe, Tobias Gerstenberg, and Fiery A Cushman. Causal judgments approximate the effectiveness of future interventions, Apr 2018. URL psyarxiv.com/nq53z.

Suraj Nair, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Causal induction from visual observations for goal directed tasks, 2019.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction, 2017.

J Pearl, M Glymour, and N P Jewell. *Causal Inference in Statistics: A Primer*. Wiley, 2016. ISBN 9781119186847.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009. ISBN 052189560X.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Towards causal representation learning, 2021. URL https://arxiv.org/abs/2102.11107.

Satinder Singh, Andrew G Barto, and Nuttapong Chentanez. Intrinsically Motivated Reinforcement Learning. *IEEE Transactions on Autonomous Mental Development*, 2(2):70–82, 2005. ISSN 19430604. doi: 10.1109/TAMD.2010.2051031.

Yuhang Song, Jianyi Wang, Thomas Lukasiewicz, Zhenghua Xu, Shangtong Zhang, Andrzej Wojcicki, and Mai Xu. Mega-reward: Achieving human-level play without extrinsic rewards, 2019.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning - An Introduction*. 2018. ISBN 9780262039246.

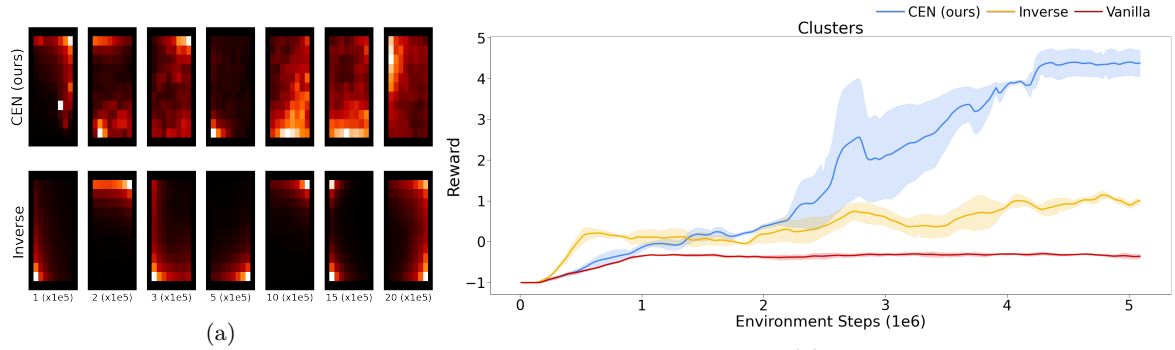
Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup. Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *In Proceedings of 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS), May 2– 6*, 2011.

Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences*, 106(28):11478–11483, 2009. doi: 10.1073/pnas.0710743106. URL `https://www.pnas.org/doi/abs/10.1073/pnas.0710743106`.

Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575, 11 2019. doi: 10.1038/s41586-019-1724-z.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

Michael L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL `https://doi.org/10.21105/joss.03021`.



(a) (b)

Figure 8: a) State visitation maps at different points of training of the Empty environment. CEN values different locations similarly, and consequently, the agent learns to explore states more uniformly. The inverse model encourages going to walls where predicting the action is hard. b) CEN promotes the movement of boxes and consequently faster learning

A Appendix

B Additional experiments

B.1 CEN as intrinsic motivator

We have shown that CEN can learn controlled effects in an unsupervised manner. Here we showcase the use of this ability as an intrinsic motivator of a reinforcement learning agent. We consider two tasks, an empty environment without any extrinsic reward where the agent can only control itself and Clusters. The RL agent is implemented using PPO (vanilla). Additionally, PPO is augmented with the exploration bonus proposed in Never Give Up (NGU + Inverse) (Badia et al., 2020b). NGU is composed of two modules for computing episodic and life-long rewards. For simplicity the following experiments only use the episodic module which in NGU consists of a count-based method using an episodic memory to approximate the number of times an agent visited each state, and an inverse model to identify controlled states. We replace the inverse model with CEN (NGU + CEN) and use the latent representation of the controlled branch to compute NGU's episodic reward. In this experiments, CEN is trained along the PPO policy using experiences collected with the same policy.

Empty environment: the goal of this environment is to showcase how each NGU variant rewards controlled events. In this environment, only the agent's movement is controlled; thus, rewarding for controlling the agent's location should promote a uniform exploration of the environment since once a location is visited, it should not be rewarded as much as visiting a new location. We hypothesize that an inverse model will create similar representations for the same action disregarding the location where it was taken. This should impair exploration since the reward will be similar regardless of where the action is performed.

Fig. 8a shows that agent trained with the inverse model hogs walls and corners. This is because is hard to predict the action near unmovable objects, leading to higher value. Contrarily, CEN promotes different locations more uniformly and the agent learns to explore better the environment.

Clusters environment: A more challenging environment is Clusters, where the agent needs to move colored boxes to their respective colored blocks. This environment provides a reward at the end of the episode corresponding to the total number of boxes correctly placed. Results are provided in Fig. 8b. Due to the sparsity of the reward PPO does not learn a correct behavior in the given time. Similarly, NGU + Inverse learns to place one box but fails to learn a general behavior to solve the task. Conversely, NGU + CEN quickly learns to move boxes leading to a high extrinsic reward.

B.2 Additional baselines for attribute prediction

The following table is an extension of Tab. 1 with additional simpler baselines. The **No mask** baseline does not filter any pixel from the image, i.e., the mask is all ones. We would expect close to perfect prediction of the attributes since all the information is in the input of the probing network, our results confirm that this is the case.

The **CEN (rand.)** and **Normal (rand.)** baselines are untrained, randomly initialized, CEN networks where one provides the controlled hidden representation and the other the normal hidden representation to the probing network. This experiment aims to isolate the benefits of the architectural biases introduced by CEN.

Interestingly, when there is no uncontrolled effects (Clusters and Lights), the controlled branch provides a good representation. Even with random weights the network is able to encode the action in the representation, making it easier for the linear probe to predict the agent attribute. Action and agent are correlated making the task of the linear probe easier. If the relation between object and action is more complex (e.g. boxes, lights or buttons), the linear probe has more difficulties. This indicates that the architectural biases are enough for simple relations but the more complex the relation the more learning is needed. When there are uncontrolled effects (Spiders and MZR) the representation is entangled and the performance of the linear probe is low.

		F1 Pixels		F1 Latent		
Environment	Attribute	CEN (ours)	No mask	CEN (ours)	CEN (rand.)	Normal (rand.)
Spiders	Agent	1.0±0.00	1.0±0.00	0.97±0.01	0.56±0.06	0.16±0.01
	Spider ↓	0.35±0.03	1.0±0.00	0.41±0.01	0.29±0.01	0.17±0.01
Clusters	Agent	0.76±0.41	1.0±0.00	0.97±0.01	0.95±0.01	0.17±0.01
	Box	0.78±0.37	1.0±0.00	0.95±0.02	0.84±0.04	0.20±0.02
Lights	Agent	0.97±0.01	1.0±0.00	1.0±0.01	0.95±0.01	0.18±0.01
	Button	0.93±0.05	1.0±0.00	0.99±0.0	0.59±0.04	0.49±0.14
	Light	0.93±0.04	1.0±0.00	1.0±0.0	0.56±0.06	0.47±0.05
MZR	Agent	0.66±0.08	0.97±0.02	0.91±0.02	0.62±0.02	0.17±0.01
	Skull ↓	0.19±0.03	1.0±0.01	0.61±0.03	0.48±0.02	0.36±0.00

Table 2: F1 score for CEN and additional baselines when predicting attributes from pixel or latent space.

B.3 Masks

Here we provide masks for both CEN and ADM. Mask are based on effects and extend over two frames, for visualization purposes here we only show the next observation with the full mask. It can be seen that CEN fails for a few pixels in MZR, which may explain the low F1 score for empty GT.

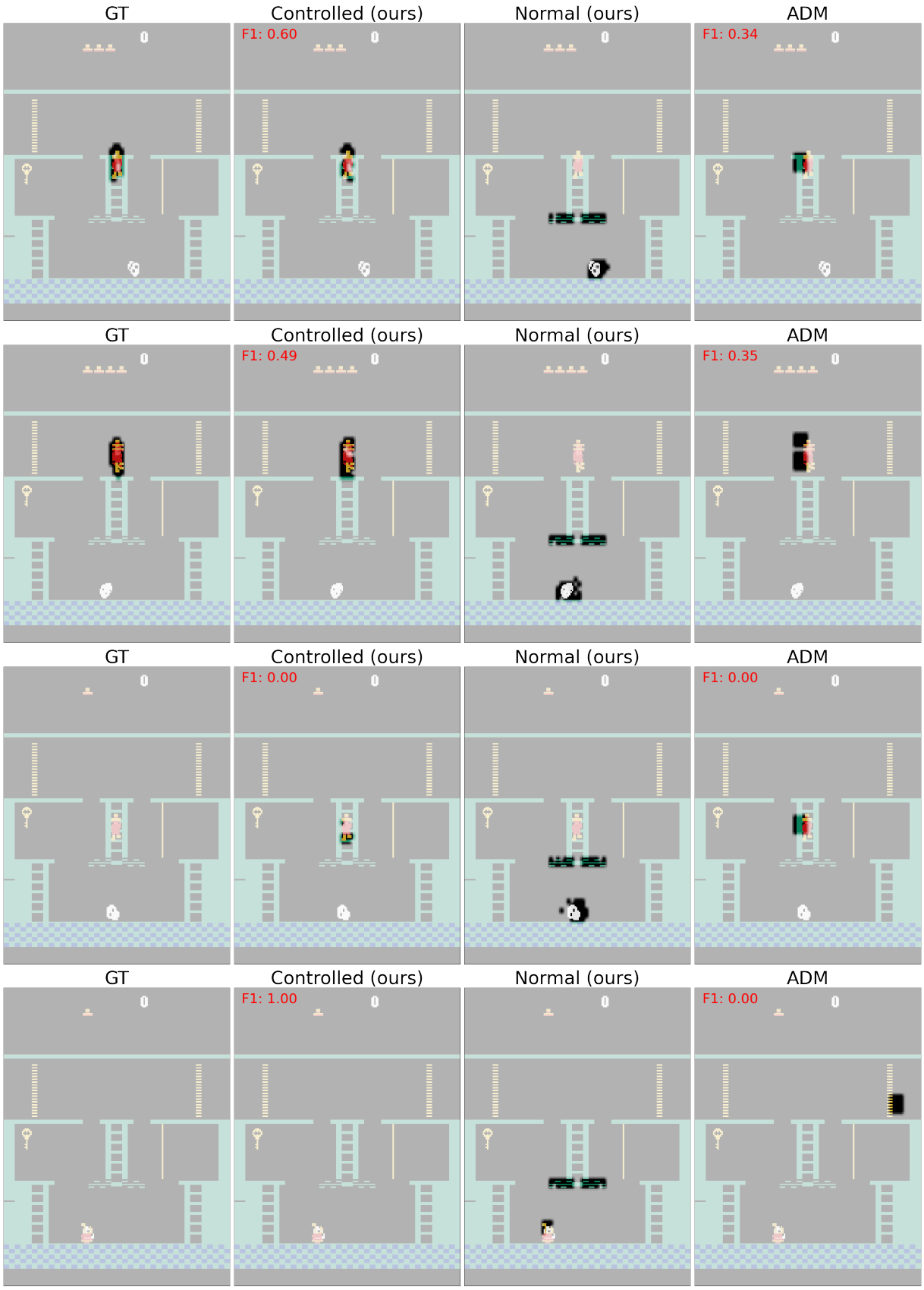


Figure 9: Examples of success and failure cases for CEN and ADM in MZR.

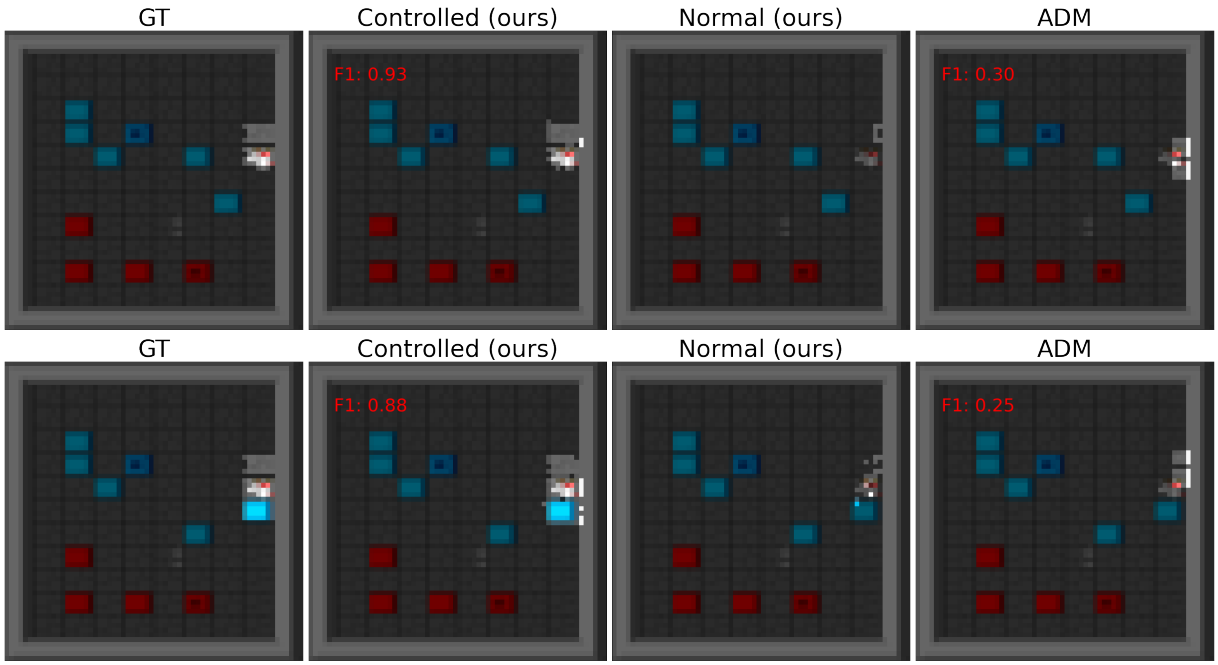


Figure 10: Examples of masks for Clusters.

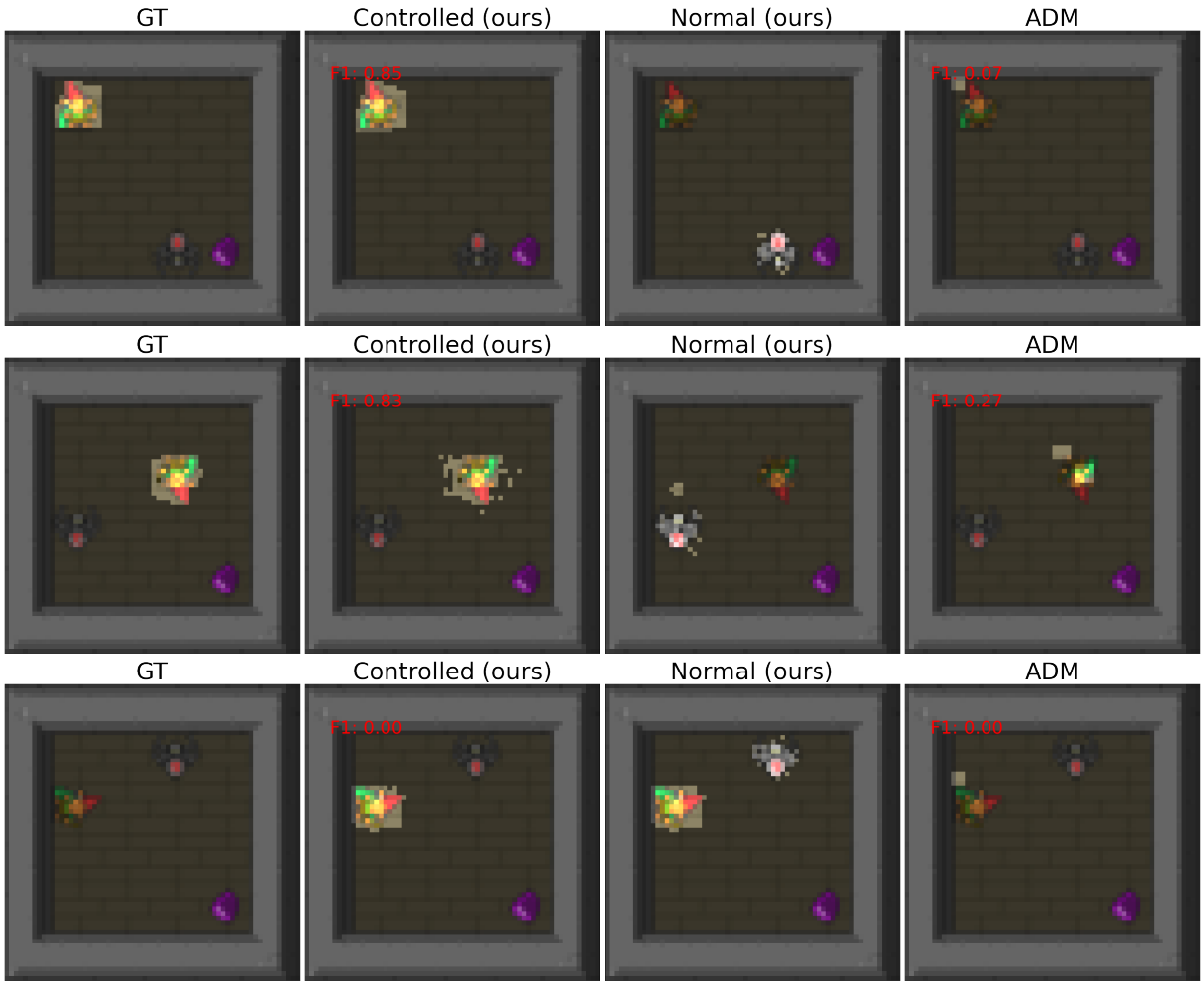


Figure 11: Examples of masks for Spiders.

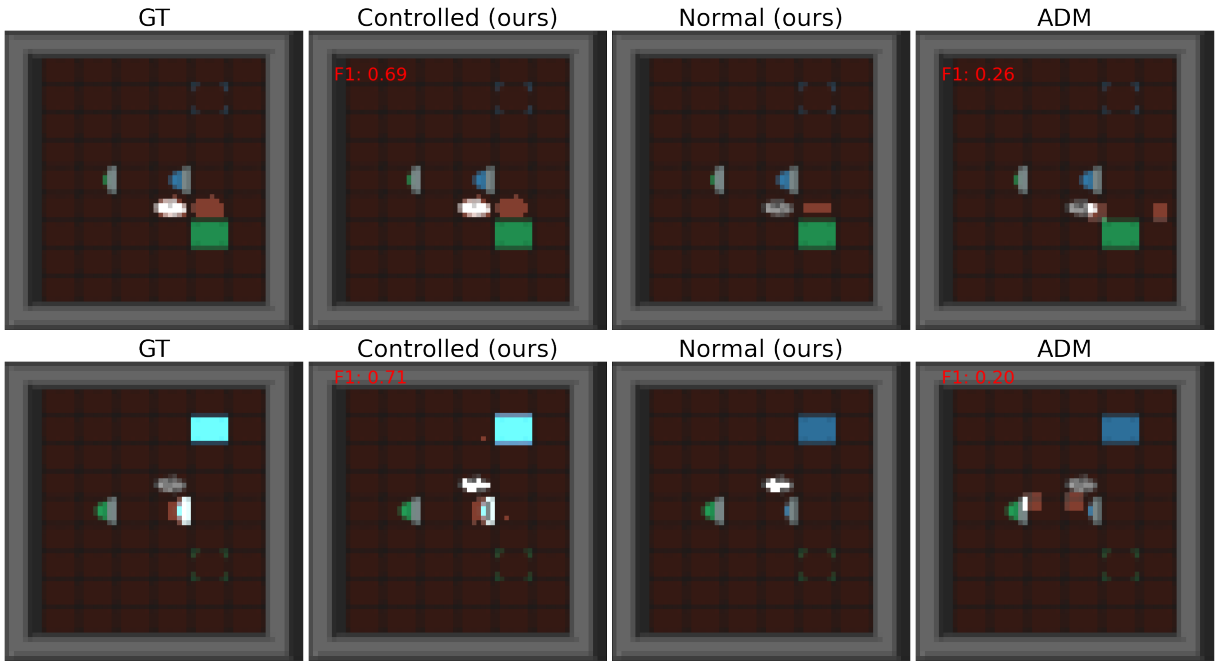


Figure 12: Examples of masks for Lights.

C Relation of Blame to the advantage function

A typical use of the formulation introduced in section 2 is to compute the causal effect of an action on the return G relative to a policy π as

$$\begin{aligned} ICE^a_G(s) &= G^a(s) - \beta_G(s) \\ &= Q(s,a) - V(s) \\ &= A(s,a). \end{aligned} \tag{6}$$

$G^a(s)$ is the return the agent would get if action a were to be taken at state s and is typically estimated using a state-action value function $Q(s,a)$. The choice of normality for $\beta_G(s)$ is to estimate the expected return with the state-value function $V(s)$, giving us the advantage function $A(s,a)$. Inspired by Generalized Value Functions (Sutton et al., 2011; Sutton & Barto, 2018) which aim to integrate general knowledge of the world; Blame takes a more general approach than the advantage function by leaving return as special case.

D Environments

D.1 Griddly

Griddly (Bamford et al., 2020) is a highly optimized grid-world based suite of environments. Environments used in this work based on Griddly generate 64×64 pixel observations, although the size of the grid-world may vary. Griddly supports multiple rendering formats, this work uses the 2D rendering of sprites.

Spiders: is a 6x6 arena where a Gnome (the agent) has to grab a Gem without being killed by a Spider. The agent dies if it collides with the spider. In this environment, the agent can move *left*, *right*, *up*, *down* or *stay*. The spider takes an action randomly from the following: rotate left, rotate right or move forward. This environment's controlled entities are: Agent.

Clusters: is a 13x10 arena where a Knight has to move boxes of the same color to their corresponding colored-block without touching the spikes. There are two different colors, blue and red. The

agent is rewarded with +1 whenever a box is pushed towards a similar colored block. The agent dies if it collides with spikes or if a box is destroyed by spikes. The agent can move *left*, *right*, *up*, *down* or *stay*. This environment's controlled entities are: Agent and Boxes.

For the RL experiments, we made the environment more sparse by removing all intermediate rewards and only rewarded the agent after all the boxes of the same color are pushed to blocks. Since the agent can not get any reward whenever a box is stuck to a wall. We removed boxes that touches the wall and punished the agent with -0.01. We then scaled the reward of solving a color with the number of boxes pushed to the block. This modifications encourage the agent to solve the environment by pushing the maximum number of boxes into blocks while preventing it from getting deprived from reward by accidentally pushing boxes to walls.

Lights: is a 11x8 arena where a Ghost (the agent) has to turn all the lights on by pressing each button. Buttons and lights are colored either blue or green. Pressing a button of one color turns the light of the same color on. The agent can move *left*, *right*, *up*, *down* or *stay*. This environment's controlled entities are: Agent, Buttons and Lights.

Empty: this environment is a copy of the clusters environment where all the boxes, blocks, spikes and rewards were removed.

D.2 Atari Montezuma's Revenge

The ALE (Bellemare et al., 2012a) provides access to Atari 2600 games to learning methods like RL. As it has been a popular choice by methods using inverse models, in this work we use the game Montezuma's Revenge to evaluate CEN. This environment provides uncontrolled as well as controlled effects with more complex entities. The environment typically generates observations of 210x160 pixels which we downscale to 64x64 pixels. Additionally the action space is of size 10.

E Training

E.1 Architecture

Encoder: is composed of two 2D convolutional layers with 4x4 kernels, stride 2 and padding of 1. Additionally, we have 2 residual blocks each with two 2D convolutional layers with stride 1 and padding of 1. The first layer has a kernel of 3x3 and the second layer of 1x1. ReLU is used as activation function throughout the network; BatchNorm is used between each layer; and 64 channels on every convolutional layer. We project the resulting maps into a flatten vector of size 32 using a linear layer with ReLU activation function.

Decoder: this module is composed of six 2D transposed convolutional layers all having 4x4 kernels, stride 2 and padding of 1. Each layer uses ReLU as activation function but the output layer which uses Tanh activation. Every layer uses 64 channels with the exception of the last layer which outputs a 1 channel prediction of the perceived effects. Parameters are shared among the controlled and normal branch decoders.

Controlled and normal modules: both modules are composed of three linear layers with 32 hidden units, each with a ReLU as activation function. The input to the controlled branch are the encoded observation and an embedding of size 8 of the chosen action.

PPO Agent: uses an encoder consisting of 3 convolutional layers with (channels, padding, strides) equal to (32, 8, 4), (64, 4, 2), (64, 3, 1) respectively. The encoder is followed by two linear layers of sizes 512 and number of actions respectively, to transform the feature map to the environment's number of actions.

E.2 Hyperparameters

Name	Value	Sweep
hidden size	32	[16, 32, 64, 128]
latent size	128	[16, 32, 64, 128, 256]
channels	64	[16, 32, 64, 128]
learning rate	0.0001	[0.0001, 0.0005, 0.001, 0.005]
α	0.01	[0.001, 0.01, 0.1, 1, 5, 10, 20, 30, 50]
T	0.01	-

Table 3: CEN hyperparameter sweeps and final values used.

Name	Value	Sweep
entropy	0.05	[0.01, 0.05, 0.1, 0.5, 1, 5]
hidden size	64	[16, 32, 64, 128]
attention size	128	[32, 64, 128, 256]
learning rate	0.0001	[0.0001, 0.0005, 0.001, 0.005]
T	0.01	-

Table 4: ADM hyperparameter sweeps and final values used.

Name	Value	Sweep
encoder channels	32	[32, 64]
encoder hidden	32	[16, 32, 64, 128]
latent size	128	[64, 128, 256]
learning rate	0.0001	[0.0001, 0.0005, 0.001, 0.005]

Table 5: Inverse model hyperparameter sweeps and final values used.

Name	Value	Sweep
batch size	512	[64, 128, 512, 1024]
latent size	32	[8, 16, 32]
CEN encoder output size	128	[16, 32, 64, 128]
learning rate	0.0005	[0.00005, 0.0001, 0.0002, 0.0005, 0.001]
IR Beta	0.001	[0.0001, 0.001, 0.002, 0.005, 0.01, 0.1]
Rollout size	2048	[1024, 2048, 4096]
discount	0.95	-
epochs	10	-

Table 6: PPO hyperparameter sweeps and final values used.

II. Source Code

Source code for the implementation of the different methods proposed in this thesis is located in the following GitHub repository:
`https://github.com/Mo-youssef/controllable-IR`

III. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Youssef Mohamed**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Controllable Effects Intrinsic Reward with Proximal Policy Optimization,

   supervised by Oriol Corcoll and Raul Vicente.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Youssef Mohamed
08/08/2022