Social curiosity in deep multi-agent reinforcement learning

MSc. Thesis in Computing Science
2nd revision
ICA-3720241

Hugo Heemskerk

Supervisors

Centrum Wiskunde & Informatica
Daniel Bloembergen
Michael Kaisers

Universiteit Utrecht
Gerard Vreeswijk

October 2020

Abstract

In Multi-Agent Reinforcement Learning (MARL), social dilemma environments make cooperation hard to learn. It is even harder in the case of decentralized models, where agents do not share model components. Intrinsic rewards have only been partially explored to solve this problem, and training still requires a large amount of samples and thus time. In an attempt to speed up this process, we propose a combination of the two main categories of intrinsic rewards, curiosity and empowerment. We perform experiments in the cleanup and harvest social dilemma environments for several types of models, both with and without intrinsic motivation. We find no conclusive evidence that intrinsic motivation significantly alters experiment outcomes when using the PPO algorithm. We also find that PPO is unable to succeed in the harvest environment. However, for both of these findings we only show this to be the case without hyperparameter tuning.

Acknowledgements

I would like to thank Daan Bloembergen and Michael Kaisers for supervising my internship at the Centrum Wiskunde & Informatica, and for helping me write this thesis. Without the help of you two, I would have never made it this far. Additionally, I am grateful to Gerard Vreeswijk being my supervisor at Utrecht University, and Thijs van Ommen for providing feedback and being my second examiner, however little we got to meet in person. Natasha Jaques and Eugene Vinitsky also deserve my thanks, for writing an open-source version of their experiments, and for providing extensive additional information on their research.

A warm hello, to John, and to the ski, surf, and circuit teams. Much love also goes to Tim and Quiri, my travel companions on the journey of writing a thesis. I have many fond memories of us helping each other, dealing with all the struggles that writing scientific literature entails. Love also goes to the strength sports association SKVU Obelix, for being an awesome and supportive community, and specifically Richard, who introduced me to the Centrum Wiskunde & Informatica.

Finally I would also like to mention Zechariah and Renato, whom I am grateful to, simply for sharing their time with me.

Contents

1 Introduction **4**

2 Background **5**
2.1 Reinforcement learning . . . 5
2.1.1 Markov Games . . . 6
2.1.2 Policies . . . 7
2.1.3 Neural networks . . . 9
2.1.4 Policy learning algorithms . . . 10
2.2 Rewards and motivation . . . 13
2.2.1 Extrinsic rewards . . . 13
2.2.2 Curiosity . . . 14
2.2.3 Implementations of Curiosity . . . 16
2.2.4 Empowerment . . . 21
2.2.5 Auxiliary tasks . . . 25
2.3 Multi-agent games . . . 26
2.3.1 Coordination problems . . . 26
2.3.2 Matrix Game Social Dilemmas . . . 28
2.3.3 N-player Sequential Social Dilemmas . . . 30
2.3.4 Multi-agent games: cleanup and harvest . . . 31

3 Proposed model **32**
3.1 Motivation . . . 32
3.2 Social Curiosity Module (SCM) . . . 32
3.3 SCM loss function . . . 35
3.4 SCM reward function . . . 36

4 Method **38**
4.1 Experiments . . . 38
4.2 Hyperparameters . . . 40

5 Results **42**
5.1 Reproduction comparison . . . 43
5.2 Baseline extrinsic reward . . . 45
5.3 Social influence . . . 45
5.4 SCM . . . 46
5.5 SCM without influence reward . . . 46
5.6 Statistical test . . . 47

6 Related work **48**

7 Conclusion **49**
7.1 Future work . . . 49

1 Introduction

One of the major challenges in Reinforcement Learning (RL) is that of sparse rewards: an agent cannot learn complex behavior if the environment provides too few informative learning cues [1, 5, 17, 40]. However, many environments where automation through RL is highly desirable have this challenging property. To name a few: driving a car, warehouse package picking, automated farm and construction work. All of these environments have fairly well-defined end goals, but the space of possible intermediate states and steps towards the goals is huge, making it hard to define appropriate learning cues.

When we want an agent to learn some task, we provide it with an extrinsic reward. Extrinsic rewards are the goals we generally have in mind when thinking of completing a task: beating the level in Mario, scoring points in your favorite ball game. These goals have proven to be useful, but profoundly inadequate when used alone for learning nontrivial tasks. This is because extrinsic rewards are usually sparse: the event of success consists of a complex sequence of steps leading up to that event. Only upon success the agent is rewarded - this rarely happens. Thus, we want to provide the agent with a reward at every step in time. Unfortunately, the problem spaces for many interesting tasks are impossibly large for humans to manually annotate with reward scores.

To alleviate this problem, the concept of intrinsic motivation has been introduced. Intrinsic motivation is supposed to supply the agent with a continuous stream of rewards. This lets the agent explore both its internal states and its external environment in a somewhat structured, yet environment-agnostic way. Moreover, it has been shown to make learning the extrinsic goal easier [5, 40]. Intrinsic motivation can be subdivided into 2 categories: curiosity and empowerment [24, 42].

Jaques et al. [23] have employed empowerment in Sequential Social Dilemma [28] environments, specifically cleanup and harvest. Respectively, these are multi-agent Commons Dilemma (CD) and Public Goods Dilemma (PGD) scenarios where cooperation is needed for individual success. They demonstrated that their type of empowerment, social influence, can make agents learn cooperation, where agents without social influence fail. However, they state that their intrinsic motivation itself only sparsely doles out rewards. Additionally, they only tested empowerment as an intrinsic motivation, not curiosity. The work of de Abril and Kanai [12] used both curiosity and empowerment, but only sequentially, not simultaneously. Finally, the environment they used was single-agent, not multi-agent.

This gives rise to the question:

> How do agents perform when using a combination of curiosity and empowerment in the cleanup and harvest environments?

Performance is measured in terms of *how high the total reward is* at the end of training.

This work builds upon the existing tactic of dealing with reward sparsity: intrinsic motivation. We attempt to improve upon the models of intrinsic mo-

tivation defined by Jaques et al. [23] and Pathak et al. [40]. To do so, we first reproduce parts of the work of Jaques et al. [23] with the PPO algorithm instead of A3C, which gives surprisingly different results: the choice of model seems not to matter. Following this, our novel contribution is the application of curiosity to the empowerment reward, which we dub the Social Curiosity Module (SCM). We run experiments on the SCM, on a baseline PPO model, and on the social influence model of Jaques et al. [23]. We do not find statistically significant evidence that the mean performance of these models is different after 5e8 environment steps.

The next chapter contains all the background knowledge that we will build upon. After this we describe our proposed method. Subsequently the experiment setup and results are presented. In the penultimate section we compare the experiment results with related work, then we finish up with a conclusion which includes suggestions for future work.

2 Background

Before we dig into our new method, let's flesh out what the previously mentioned terms formally mean. We'll do this step by step with ample examples, where every subsection of this chapter covers one major topic. After reading this chapter, you should be able to understand how our proposed method fits within existing literature and concepts.

2.1 Reinforcement learning

To begin with the general field this topic is situated in: Reinforcement Learning (RL). To visualize this, imagine a computer simulation with an agent (an individual character) in an environment. For example, a computer program that is learning how to play Super Mario Bros. The agent constantly observes its environment, and upon taking in an observation it decides what to do next. This mapping from observation to action is called a *policy*.

How does it decide what to do? We give it a goal. However, we do not give it explicit step-by-step instructions on how to achieve said goal. Instead, we make the agent learn: by running the simulation over and over again, and letting the agent evaluate and update its own behavior over time. The learning is thus done by repeated trial-and-error: all the while reinforcing desirable, and punishing undesirable behavior. We call this Reinforcement Learning (RL) [48].

In RL, the goal of an agent is encoded as a human-created reward function. This function takes in an observation of the agent, and returns a score indicating how well it is doing. In other words, when the agent makes an observation, it can thus determine whether it is on the right track by consulting its reward function. In this way, an agent learns from its past actions and experiences. For example, if you want to make an agent learn how to play Pong, you could provide it with a positive reward for scoring a point, and give it a negative reward for letting the opponent score a point.

2.1.1 Markov Games

RL problems are formalized as Markov Games (MGs) [30], also known as Stochastic Games. Let's look at the components that make up a MG, then we'll go through how they are used. The discrete, finite case of MGs wraps up all the above ideas in the following formal terms: An n-player Markov Game $\mathcal{M}$ has n players, where $n > 0$. $\mathcal{M}$ has a set of possible environment states $\mathcal{S}$. Each player $i \in \{1, \ldots, n\}$ has a set of allowed actions $\mathcal{A}^i$ they can take each turn, and every player must execute a single action from their own set each turn. All possible action combinations of all players combined are called the collective action space, denoted by $\mathcal{A} = \mathcal{A}^1 \times \ldots \times \mathcal{A}^n$.

The transition function $\mathcal{T}$ tells us how to update the game world each turn. To do this, it takes in any combination of a state and player actions: $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$. The output, $\Delta(\mathcal{S})$, is a set of discrete probability distributions over S. In less fancy words: $\Delta(\mathcal{S})$ tells us what the probability of every single possible state is. Thus, for any given state, $\Delta : S \rightarrow [0, 1]$. To obtain the next state, we can then sample $\Delta(\mathcal{S})$. When the state transition contains random elements, we call it a stochastic state transition function. If it contains no such elements, it's called deterministic instead of stochastic.

Upon observing a state, all n players must choose an action a from their respective action sets. Together, they form the joint action $\vec{a} = (a^1 \in \mathcal{A}^1, \ldots, a^n \in \mathcal{A}^n)$. These actions are made independently and simultaneously: only after all players have chosen an action, they observe each other's actions. By default, Markov Games are played with full observability: players can always observe each other's actions at the end of each turn, and every player can see the entire state. This means that in normal Markov Games, every player's observation is identical: think of a board game like chess or Go. Deviating from this rule means the game is a Partially Observable Markov Game (POMG).

POMGs need a way to determine what each player sees: this is done through the observation function in eq. (1). The complete possible observation space for agent i is then given by eq. (2). All agents' simultaneous observations form the joint observation $\vec{o} = (o^1, \ldots, o^n)$. Note that regular Markov Games are specific instances of POMGs, where the observation function always returns the entire state.

$$O : \mathcal{S} \times \{1, \ldots, n\} \rightarrow \mathcal{O} \tag{1}$$

$$\mathcal{O}^i = \left\{ o^i(s) \middle| s \in \mathcal{S}, o^i = O(s, i) \right\} \tag{2}$$

$$O : \mathcal{S} \times \{1, \ldots, n\} \rightarrow \mathbb{R}^d \tag{3}$$

Since we're going to be using a computer to simulate these games, we can make our life easier by redefining the observation function in a slightly stricter way, as shown in eq. (3). This observation function produces a d-dimensional view of the game world for the specified player. In other words, the dimension parameter d determines how many real numbers the agent receives as input data on each observation. For example, a 4×5 pixel observation with a red, green, and blue value for each pixel will have a $d = 4 \cdot 5 \cdot 3 = 60$. Although the strict

definition of POMGs does not require a mapping to $\mathbb{R}$, sticking to this makes computation easier.

Finally, the game needs a goal or win condition, which is provided by a reward function for every player i, namely $r^i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This concludes the environment simulation itself. However, the environment is only one of the components that make up RL! For one, we have yet to discuss how agents choose actions, which is covered in the next section.

2.1.2 Policies

What determines how agents choose actions in Reinforcement Learning? Policies! To select an action, a RL agent i consults their policy π^i. The policy takes in the agent's observation $\mathcal{O}^i$ and produces a discrete probability distribution over i's action set: $\pi^i : \mathcal{O}^i \to \Delta(\mathcal{A}^i)$.

We can write the policy as follows: $\pi^i(a^i|o^i)$. What this notation means in the following: the policy of agent i produces a probability for each of i's actions a^i, conditional on the observation o^i made by i. The joint policy is written as $\vec{\pi}(.|\vec{o}) = (\pi^1(.|o^1), \ldots, \pi^N(.|o^n))$, where . stands for the random action variable.

Just like with the previous distribution, $\Delta(\mathcal{A}^i) \to [0, 1]$, so we can sample it to obtain the next action. Deterministic policies exist, though these are usually indicated by the character μ instead of π [29, 32], and they provide a direct mapping from observation to action, like so: $\mu^i : \mathcal{O}^i \to \mathcal{A}^i$.

It is hard to find the correct policy. We can't simply try to maximize reward at every single step in time, because any action the agent undertakes changes what states it can end up in afterwards. Actions have consequences: only considering the present does not make for good long-term planning. However, looking too far ahead is not feasible either, as this quickly becomes too computationally expensive.

Also, recall that the game is stochastic - there's randomness involved! Any future reward may thus be uncertain. Moreover, some games have a set amount of turns after which they end, other games can go on forever, and then there are the games that may end randomly. How could you possibly choose an optimal policy if you don't even know when the game ends?

To figure out how to choose a good policy, we have to consider the uncertainty of future rewards. We all have the tendency to value immediate rewards higher than rewards that come later. This is not irrational, because our ability to predict the future is far from perfect. It can be uncertain whether we will still need something, whether the reward will still be there, whether others will uphold their word, even whether we are are still alive after a set amount of time. For an agent, the simulation it exists in could end at any moment. Because of this uncertainty, immediate rewards should have a higher priority than delayed ones, as the delayed rewards may fall through. In humans, this is the reason that immediate gratification is generally more salient than delayed gratification.

This is called time discounting, or time preference: a high time preference means liking rewards now much better than rewards later. For instance, when choosing between getting some amount of money now, or twice that amount

in a year, many people will opt for the quicker cash. Conversely, a low time preference is still inclined to devalue rewards that are in the future, but to a lesser degree. Having a lower time preference makes you more willing to plan ahead for rewards later in time. Humans display a positive time preference in very simple black and white situations [13]. There is not a single formula to predict human time discounting for all situations though. For instance, when choosing between mirrored sequences of increasing or decreasing reward, people will often choose the former [31]. This contradicts the model of "reward now good, future reward bad".

It would be nice to give agents some elementary version of time discounting as well, as it is evidently a useful skill. Time discounting in RL is commonly represented as the parameter $\gamma \in (0, 1]$ [48]. The discounted reward function then becomes $\gamma^t \cdot r$. In other words: for any reward r that is t steps in the future, we don't value it fully at $1 \cdot r$, but at $\gamma^t \cdot r$. This reduces the value of rewards more, the further away they are in the future, as long as $0 < \gamma < 1$. Note that, paradoxically, higher time discounting means using a lower γ. What economists call negative time discounting (future rewards are better than instant rewards), implies that $\gamma > 1$.

Note that neither a high nor low time preference is inherently good or bad: they are appropriate for different types of situations. For instance, if your environment is chaotic and violent, it makes sense to have a high time preference. There is no sense in making long-term plans if you die young, so you'd better live fast. Conversely, if you live in a very peaceful, predictable world that allows you to plan for the future with low uncertainty, low time preference allows for more complex behavior with a potentially larger payoff, despite increased sparsity of individual reward moments.

With time discounting in hand, we almost have a way to tell our agents how to learn: by looking for the policy π that returns the maximum expected value for the discounted reward function. Why expected? Because it's still stochastic, we can't be certain that our efforts will be rewarded.

Learning happens based only on the agent's observations $o^i = O(s, i)$ and its received reward $r^i(s, \vec{a})$. The goal that every agent now works towards is maximizing the expected discounted reward, shown in eq. (4) below. In this formula, expressions with the form $x \sim y$ mean x sampled from y.

$$V^i_{\vec{\pi}}(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r^i(s_t, \vec{a}_t) | \vec{a} \sim \vec{\pi}_t, s_{t+1} \sim \mathcal{T}(s_t, \vec{a}_t)\right] \tag{4}$$

If we didn't have $\gamma < 1$, think of what would happen to our optimization function above. The estimated reward could sum to positive or negative infinity, and would therefore be useless for optimization. If time goes on forever, to maximize reward, you only need the tiniest amount of expected reward per turn: eventually it will sum to infinity anyway. The agent would not be able to distinguish between different infinite returns, and might settle on a policy that only returns the tiniest rewards. Ideally we'd like agents that do better than the bare minimum!

Note that estimated reward methods other than time discounting exist: if we know at what timestep the simulation ends it does not make sense to discount any reward, so we can set γ to 1. This is called a finite-horizon method [48]. On the other hand, if we assume our simulation will run for an infinite amount of time, yet we still want an optimization target, we can consider the average reward per period. These can be created with Cesaro sums[2]. However, the discounted reward function is mathematically simpler[48], which in part explains its relative popularity compared to other optimization methods.

We can now plug our optimization function into an optimization algorithm, which lets an agent learn. In RL, policy gradient algorithms have proven to be effective optimizers. These are implemented using neural networks, as such these are the focus of the next section.

2.1.3 Neural networks

Now that we have a function to optimize for, let's find out how the actual policy learning process goes. Recall the expectation function in eq. (4). We want to maximize the expected discounted reward so that our agent starts performing some desired behavior.

To do this, we use Neural Networks (NNs). The policy is represented by this NN, and maps observations to actions. For continuous actions, the neural network directly outputs the given actions. For discrete actions, it outputs an action distribution which we can then sample from.

To summarize the learning process: first, we create and initialize an NN with random variables. We let the agent collect experience: tuples of 2 states (at t and $t+1$), the chosen action and the reward value obtained in that state transition. In short: (s_t, s_{t+1}, a_t, r_t). Experience is collected by running the simulation, while the (randomly initialized) policy chooses an action at every step.

The experience is concatenated into a list of experience called a trajectory. From this trajectory, a scalar loss value is calculated. Loss corresponds to model error: the goal of the optimization process is to minimize the loss. In RL specifically, we want to minimize the loss over any future states we might encounter.

With this loss value, Stochastic Gradient Descent (SGD), Adam [25], or a different optimization function is performed on the NN. Then, the updated NN is used to collect more experience, and the loop repeats. This goes on indefinitely until some user-defined stopping criterion is reached. How exactly the loss is calculated varies per model architecture and choice of reward function. We'll get to reward functions in section 2.2, and the main categories of learning algorithms in RL are presented in the next section. Before this however, there's one challenge for NNs we have to discuss: time.

Basic NNs are terrible at remembering previous events. Namely: they don't. However, many RL problems span more than just a few timesteps! Learning a good policy can require you to know what specifically happened some amount of states ago.

To tackle this issue, Recurrent Neural Networks (RNNs) were first used [18]. These networks feed their output back into themselves, allowing them to access data from past evaluations. You can plug an RNN layer into your model between other layers, allowing you to add it to existing model architectures. However, RNNs take extremely large amounts of data to train, and are sensitive to the delay between information storage and retrieval.

To solve these shortcomings, the Long-Short Term Memory (LSTM) was conceived by Hochreiter and Schmidhuber [18]. Like the RNN, the LSTM is a pluggable NN layer. It internally utilizes RNNs, and still requires a high amount of training data, though far less so than RNNs do. Thankfully, in RL we can generate as much training data as we want, which makes LSTM use feasible. Secondly, LSTMs can learn how long they need to store information for - a feature that RNNs lacked. Just like with RNNs, an LSTM component can be plugged in between neural network layers.

With that out of the way - onwards to the main categories of policy-finding algorithms that can be implemented using these NN architectures.

2.1.4 Policy learning algorithms

There are currently two main flavors of learning algorithms (also called learning methods) within RL: policy gradient, and action-value methods [48].

Action-value methods are commonly known as Q-learning. It learns the quality or *Q-value* of state-action pairs. This Q-value represents the expected discounted reward for all states after s_t when action a is taken in state s_t. The Q-value is then used by an algorithm to determine the actual policy. The algorithm ϵ-greedy is commonly used, which at each environment timestep generates a random number v between 0 and 1. If $v \leq \epsilon$, it chooses a random action. Otherwise, it chooses the action with the highest Q-value for that state.

Policy gradient methods directly represent the policy in a NN. These algorithms have the benefit of more efficiently representing continuous action spaces (as opposed to discrete ones), as well as the ability to explicitly learn a stochastic policy [48]. However, action-value methods do not require a NN, and are generally more stable and quick to learn.

Each policy gradient algorithm comes with its own loss function. There exist numerous policy gradient algorithms [50], varying in computational complexity, simplicity of implementation, and performance across different tasks. Both Q-learning an policy gradient methods see use in research, and both are still being improved upon. However, in this section we will mostly focus on policy gradient algorithms, because the research by Jaques et al. [23] that we will be building on later utilizes only those.

To formally define the Q-value, we need the state-value function, denoted as $V(s)$. For any given state s, the state-value function $V(s)$ represents the expected discounted reward of a state (without an action). The Q-value is then:

$$Q(s_t, a) = E\Big[r_{t+1} + \gamma V(s_{t+1})\Big] \tag{5}$$

(Q-)Value functions have an issue - they can have large variance. Imagine a bonus level of a game where an agent is showered with coins, which provide high reward. If the rest of the level contains few coins, the bonus level is likely to greatly distort the learning process, as there is a sudden gigantic influx of reward. Additionally, imagine a situation where an agent is given 101 reward if it chooses the correct action. If it chooses the wrong action, it receives 100 reward. There hardly is a difference, yet we would like the agent to be strongly incentivized to learn the correct action.

To solve this problem, the concept of *advantage* was coined. The advantage does not look at the absolute reward value, but instead at the relative quality. This requires the Q-value, $Q(s, a)$ from eq. (5). The advantage function A is the following:

$$A(s, a) = Q(s, a) - V(s) \tag{6}$$

In the 101 vs 100 reward example above, the 100 reward would be subtracted out. Therefore, the advantage of choosing the correct action would be 1, and for other actions 0. This leads to far smoother policy updates, which benefits learning.

Nearly all policy gradient algorithms use an Actor-Critic approach [48]. Though the Actor-Critic structure predates neural networks, it can be represented in one. To do this, the NN architecture is structured in such a way that both the policy (actor) and value function (critic) are learned. The policy output is then represented as a set of probabilities over a discrete set of actions, or a value per continuous action. The value function output is a real number. The rationale between separating learning the policy and value function is that it becomes easier to explicitly guide the learning process. If only a policy is learned, the value of states is implicitly embedded in the policy: this way we cannot reliably judge how good every individual action is, only how good sets of actions are.

To give an example of how the value can be used: given a trajectory and a policy, the A2C algorithm by Wu et al. [53] multiplies this number by the log probability of the action-state pairs in that trajectory when using the given policy, then flips the sign. The resulting value is the policy loss for that step. The actor and critic are learned in two different networks, though these networks may share (many) parameters. Whether parameter sharing is a good idea or not depends on the learning algorithm used. For instance, when sharing parameters in Proximal Policy Optimization (PPO), the loss generated by the value function must be scaled, otherwise performance degrades [47].

You can choose which value function you want your critic to learn: if it learns the Q-value, it is Q-Actor critic, when choosing the advantage function it's called Advantage Actor Critic (A2C) [53]. The latter algorithm was derived from an asynchronous version called A3C [36], but it was found that the synchronous, deterministic variant A2C achieved equal performance.

But wait, didn't the advantage function A require both the value function V, as well as the Q-value? Learning both V and Q would be inconvenient, we'd

require three networks in total. Fortunately, there is a way around this: recall that we can write Q in terms of V through eq. (5). This way we don't need to learn Q at all, by learning V:

$$A(s,a) = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{7}$$

To reduce variance even further, Generalized Advantage Estimation (GAE) was devised by Schulman et al. [46], which is usable on top of A2C.

When updating a policy, it is wise to make only small changes to the behavior generated by that policy. Doing otherwise destabilizes the learning process, which is a common problem in RL. Schulman et al. [47] created a method that does exactly that: Proximal Policy Optimization (PPO). It clips neural network updates when the change in behavior would be too large. PPO does this in a pessimistic fashion - when the agent experiences states with highly negative advantage the update goes as it normally would, unclipped. In contrast, when an agent is confronted with states with a very high advantage, it cautiously updates the policy, not allowing for any drastic changes.

This pessimism has the following rationale: if you are doing well already, why change your behavior too much? And when doing poorly, you'd better start doing something else fast. This mirrors the Win or Lose Fast (WoLF) algorithm coined by Bowling and Veloso [4], based on the same premise: learn quickly while losing, slowly while winning. In addition to this, PPO is able to re-use the same samples multiple times, which is done by splitting batches of experience into smaller pieces, then repeatedly running the neural net optimizer on these so-called minibatches. Because the updates are small, this does not result in destabilized learning.

A final problem we will visit is that of premature convergence. If an agent ceases to explore too quickly, it gets stuck repeating the same behavior, never reaching potentially better policies. To remedy this, the concept of entropy is often used. Entropy in this context relates to how random a policy is: if all actions an agent can undertake are equally likely for a given state, the entropy for that state is maximal. Conversely, if the likelihood for one action is extremely high compared to the others, the entropy approaches zero.

When you reward an agent for having states with high entropy, it is more likely to randomly explore. Too-high entropy rewards can however destabilize the learning process. Again, a learning schedule is commonly used, where the agent will initially be rewarded handsomely for high entropy. As training progresses, the entropy reward is slowly scaled down to a lower value. Providing an incentive for entropy can be done in various ways: one method is subtracting the entropy from the final loss. Another method is to include it in the reward function, which is part of the Soft Actor-Critic (SAC) algorithm, introduced by Haarnoja et al. [14].

This wraps up the theory on policy learning algorithms. We still haven't explicitly discussed how we can go from an observation to a reward however. That's the terrain we'll be covering in the next section.

2.2 Rewards and motivation

In this section, we explore the two main categories of reward functions: extrinsic and intrinsic motivation. Furthermore, we discuss two types of intrinsic motivation: curiosity and empowerment. Finally, we conclude with a look at how intrinsic rewards resemble auxiliary tasks.

NB: When talking about an RL reward function, it is often shorthanded to reward. On top of that, the concrete step-by-step reward value provided by a reward function is also called reward - fortunately this term occurs much less often, and context should make clear what is meant.

2.2.1 Extrinsic rewards

When a reward function resembles how you and I would describe the goal of a behavior, we call it an extrinsic (or extrinsically defined) reward. It thus falls under the category of extrinsic motivation. Note the subtle difference between goal and reward - extrinsic rewards are a human attempt to capture a human goal in concrete, actionable terms.

However, what the agent does with it is does not necessarily align with what the human intends. For an amusing example, think of an agent that is rewarded for maximizing its score in a videogame, and instead of learning to play well, it finds an exploit that just racks up the points. This exact behavior has been observed by Clark and Amodei [9], and in the work of Chrabaszcz et al. [8]. In the latter, it even managed to produce funky buggy visuals. While some exploits may look like master-level play, others look like a nonsensical pointless sequences of actions. In those cases, what's going is that the sequence of actions exploits bugs that manipulate the game's memory to increase the player's score, therefore giving the agent large amounts of reward. It's up for debate whether this is a desirable outcome.

Bostrom [3] gives us a somewhat less innocent example: picture a machine that is rewarded for maximizing the amount of paperclips in its possession. The machine figures out that humans turning it off would impede its goal of maximizing paperclip production, and thus it kills all humans and eventually turns the world into paperclips, after which it sets its sights on space. It achieves its goal, but with terrible externalities.

As you can see, giving agents an appropriate goal (reward function) is capital H Hard. Imagine attempting to make a single agent learn to play Super Mario Bros. What is the goal? Completing levels? What is a level, and how would the agent recognize the completion of one? Even if you explicitly tell the agent "you have completed the level" when it does so, there would be only one indicator of success in the entire level: the endpoint! How would the agent know that it is making progress towards the end, at any point in time?

None of these are trivial to define. What about collecting coins, or getting a higher score? Well, if the final boss gives you neither score nor coins, the agent will realistically never beat it. Worse, for every game out there, every simulation and real-world environment, for every goal you can imagine, you

would have to painstakingly write down what all possible correct steps towards that goal could be. This requires manual human annotation of impossibly large amounts of observations.

These kinds of problems are so common in RL, that there are names for them. When we talk about the problem of sparse rewards, we mean exactly the Super Mario Bros. level completion problem we just discussed: level completion is not a good indicator for learning, because it only happens after a long and complicated sequence of steps. The rewards are too rare or infrequent to even begin making progress towards those rewards. In other words, when rewards are too sparse, the agent doesn't receive enough feedback to determine whether it is on the right track towards its goal [1, 5, 17, 40]. Imagine doing mathematics assignments, and only receiving teacher feedback on one problem for every million you turn in. Not a great learning environment.

Then there is the problem of exploration: high-reward states may be hidden behind several low-, zero- or negative-reward states. Agents tend to nudge themselves out of negative-reward states because they prefer high rewards over low ones, which means they might never reach the high-reward state. Consider the Super Mario Bros. final boss example from above: even if beating the last level gives a huge coin or score reward, the agent is unlikely to ever see it if the act of fighting the boss dispenses no coins or score.

Reward functions seem to be faced with a minefield of challenges. Is there no escape from poorly defined reward functions outside trivial environments? Never mind that we're talking about a videogame that can be completed by a sufficiently dedicated child, all this without giving them a goal, except maybe telling them to have fun. And *fun*, in fact, is the key word here. Both children and adults seem to be intrinsically motivated to play games for fun, and playful behavior is not limited to humans: it can also be observed in cats, chimpanzees and other animals [51, 52]. Nature seems to be on to something useful here - can we give RL agents the same kind of drive? The answer is: yes!

In RL, the concept of intrinsic motivation has inspired various sub-types of general (and thus not problem-specific) reward functions. Two distinct categories that are often named are the following: curiosity [40, 45] and empowerment [26, 37]. These sub-types are not set in stone, there are various ways of categorizing them. For instance, Oudeyer and Kaplan [38] defined various categories of intrinsic motivation, and make no mention of empowerment at all. However, for our purposes curiosity and empowerment will suffice. We'll dive into them in the next three sections.

2.2.2 Curiosity

To recap, extrinsic motivation creates the following challenges: sparse rewards make learning hard, manual annotation of environments is expensive or impossible, and high-reward states that are hidden behind low-reward states are frequently left unexplored. Moreover, an extrinsic reward has to be defined for every single separate environment - ideally, a reward function works independently of the specific environment or problem that we are trying to solve

through RL.

Curiosity, a type of intrinsic motivation, has exactly the qualities that address these issues. It is a general method of rewarding an agent, encouraging it to explore regardless of the environment and problem it faces. Additionally, a curiosity reward is available at every single step, so the agent can learn at every single step.

What is the intuition behind choosing curiosity as a reward function? Chentanez et al. [7] borrow this term from developmental psychology, which concerns itself with questions such as "what is the purpose of play?". Curiosity directs humans and other animals to explore, play, and exhibit other behaviors when an extrinsic reward is absent. Although curiosity might not always lead to the development of any particular skill, it provides you with experiences and general knowledge. These generate general competence: the ability to reason about, and solve general sub-problems. Acquiring this competence then makes learning specific skills easier [7]. The reasoning to model curiosity in RL then is the following: curiosity should generate general competence in agents, which makes learning to achieve extrinsic goals easier.

In RL, an intrinsic motivation is usually combined with an extrinsic motivation. The intrinsic motivation helps agents to learn the extrinsically defined task more quickly: received extrinsic reward increases more rapidly when the agent also receives a curiosity reward [40]. To facilitate this combination, learning schedules are used. The reasoning behind this is the following: when starting to learn, everything will be novel - at this point essentially all behavior is exploration. Thus, there is no need for an exploration reward - it would likely drown out any extrinsic reward the agent receives at this point. Secondly, when nearing the end of our learning process, we want to focus more on the extrinsic reward. After all, this is usually what we benchmark the performance of our agents on.

To give an example of a learning schedule: For a learning process where we train for n environment steps in total, the intrinsic reward could start with a multiplier μ set to 0. Between 0 and $0.1 \cdot n$ environment steps, we linearly scale μ to 1 each environment step, which stays at 1 until $0.5 \cdot n$ environment steps. From there on out until $0.8 \cdot n$ environment steps, μ is linearly scaled down to 0.5. μ then remains at 0.5 until the end of the learning process: upon reaching n environment steps. The amount of steps and values of mu in this example are arbitrary, finding appropriate values for the learning schedule is discussed in section 4.2.

However, curiosity can also be used without any extrinsic reward. This was done by Burda et al. [5], who tested 54 different standard benchmark environments using curiosity alone. This produced surprisingly good results: their purely curiosity-driven agent managed to complete over 11 Super Mario Bros. levels. Guess you didn't need score or coins after all! Now, how would we go about modelling curiosity? The usual method is that of a state prediction: if the agent cannot predict the (immediate) future well, it is rewarded. This may seem counter-intuitive: why reward an agent for bad performance? To understand this, think of how the agent will change its behavior: it is now incentivized

to seek out states that it has not seen often enough yet. As the agent visits the unexplored area and learns to predict it, it obtains less curiosity reward. This then incentivizes the agent to visit new, unexplored areas to obtain more reward.

But what if a part of the environment is just incredibly hard or even impossible to predict? This would get the agent stuck in an endless loop of feeling very good about itself for looking at analog TV static noise. To be precise: it would endlessly get a high curiosity reward for observing pixels randomly flipping between black and white. This was named the Noisy-TV Problem by Burda et al. [5].

So how do we remedy this? You could make agents curious only about the effect of their actions on the environment - but then any environment containing a TV and a remote will generate massive amounts of irresistible randomness for our poor agent, which it induces by its own actions. This is a known issue with curiosity, Savinov et al. [43] dubbed it the Couch-Potato Problem for reasons you can imagine. Note how children don't do this, and are not in the least interested in looking at TV noise. You may however observe couch-potato-like behavior in them when they are provided with an age-appropriate TV-show or movie that does instill curiosity in them. Finally, consider how children will be *highly* resistant to the idea of watching shows made for those younger than themselves. When they have learned what they can, they move on.

Children are somehow able to determine that TV noise is just that: noise. Therefore, it is uninteresting to them. No matter how long they will stare at it, they will never be able to properly predict the next pixel state. This irreducible uncertainty is called aleatoric uncertainty. The other type, epistemic uncertainty, models what we don't know yet, but could theoretically learn if we made sufficient observations [35]. The Noisy-TV Problem thus exists because naive curiosity cannot distinguish between aleatoric and epistemic uncertainty [6].

Thus far we've only covered curiosity on a conceptual level - in the next section we'll discuss some concrete RL implementations.

2.2.3 Implementations of Curiosity

Here we compare two concrete implementations of curiosity: the Intrinsic Curiosity Module by Pathak et al. [40], and the Random Network Distillation model by Burda et al. [6]. These models are general-purpose, and are not specifically tailored to multi-agent learning, nor to social RL problems. To see how they perform in a multi-agent setting, we'll subsequently look the work of Schafer [44].

The Intrinsic Curiosity Module (ICM) by [40] rewards the agent by making a double prediction: first, what is the next state if I choose this action. Second: to achieve a specific change, what action should I choose? The reward then correlates with how wrong the agent is in only the first prediction: more wrong equals more reward. The reason why only the first prediction is used for the reward will be explained after introducing some terminology. Its neural network

architecture is given in fig. 1. The conceptual structure of the ICM can be found in fig. 2.

To give a formal definition, the feature encoding function ϕ has to be introduced. Because high-dimensional observations tend to contain data that is not relevant to solving a given problem, we compress observations with ϕ. This function maps a high-dimensional observation to a lower-dimensional feature set. The features that are learned are not defined by humans - in the learning process, an agent figures out by itself over time what features are relevant, and which ones are not. A state s encoded by ϕ is denoted as $\phi(s)$.

So, to formally define what the ICM predicts: first, given the current encoded state $\phi(s_t)$ at time t, and chosen action a_t, what is $\phi(s_{t+1})$? Second, given $\phi(s_t)$ and $\phi(s_{t+1})$, what is a_t? These then lead to the curiosity reward r_t at time t. The predictions themselves also have a notation: to get predicted state $\hat{\phi}(s_{t+1})$, and predicted action $\hat{a}_t$, we respectively use the following functions f and g:

$$\hat{a}_t = g\Big(s_t, s_{t+1}; \theta_I\Big) \tag{8}$$

$$\hat{\phi}(s_{t+1}) = f\Big(\phi(s_t), a_t; \theta_F\Big) \tag{9}$$

Each of these functions contains a set of parameters, θ_I and θ_F respectively. These are the neural network parameters which have to be learned to make these functions perform well. To optimize the parameters θ_I and θ_F, we minimize their respective loss functions L_I and L_F. These are respectively found in equations 11 and 12, and are weighted as shown in eq. (15). The learned function in eq. (8) is called the inverse (dynamics) model, while eq. (9) is also called the forward model.

To illustrate the reason why the model is structured this way, think of how there is a tension between the forward and inverse model. The forward model's prediction error is lowest when all features go to 0 at all times - when they don't encode anything at all. The inverse model cannot afford to let this happen, because it needs some data to predict what action was undertaken. Thus, the ICM is incentivized to only learn those features that pertain directly to how the agent can influence the state, and to ignore everything else.

$L_I(\hat{a}_t, a_t)$ measures the difference between predictions and actual actions. When the action space is discrete, g outputs a softmaxed distribution over the agent's action space. In this case, L_I outputs the cross entropy between the logits of $\hat{a}_t$, and the actually performed action A_t.

Cross entropy can be seen as a measure of how different two probability distributions are. We have true distribution p and predicted distribution q. Both are distributions over the same discrete set of events $\mathcal{X}$. The cross entropy $H(p, q)$ between p and q is then:

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x) \tag{10}$$

But we had actions, not distributions, right? Correct, and we can transform the former into the latter. As for the predicted distribution: this is the direct

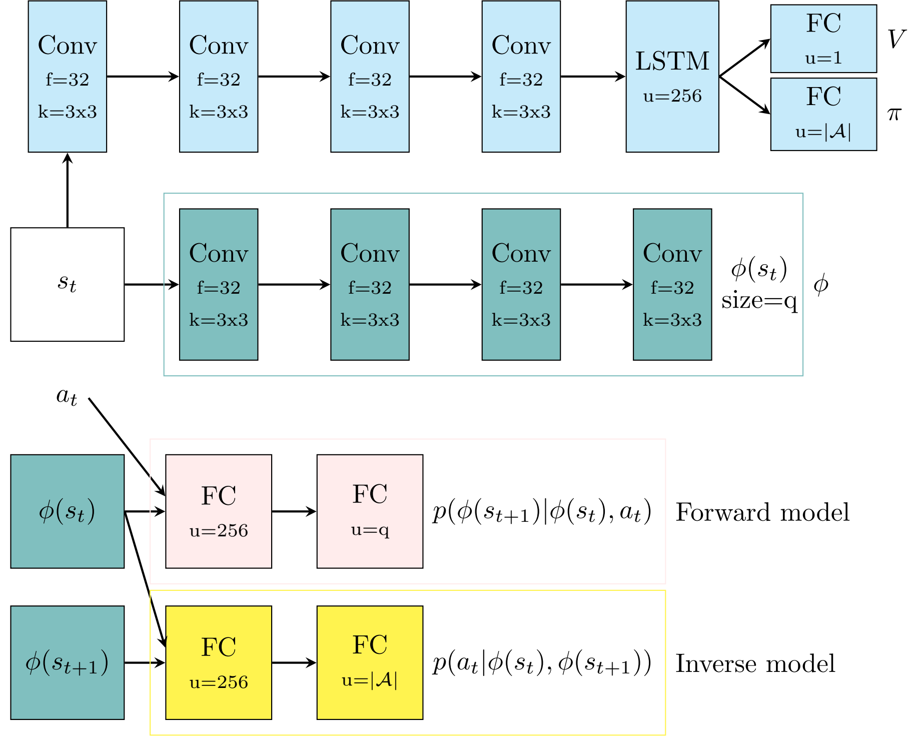


Figure 1: The ICM neural network structure by Pathak et al. [40]. The top, light blue set of layers constitutes the policy gradient network. The encoder network is denoted by ϕ, and $\phi(s_t)$ denotes the encoded state at time t. Encoded states are supplied to the Forward and Inverse models. The Forward model predicts the next encoded state, given the current encoded state and the current action. The Inverse model predicts the current action, given both the current and next encoded state. After each individual convolution layer, an exponential linear unit (ELU) activation function is used [10].

Conv:	Convolutional layer	**FC**:	Fully connected layer
f:	Convolution filters	**u** in FC:	Fully connected neurons
k:	Convolution kernel size	$\boldsymbol{\vert\mathcal{A}\vert}$:	Amount of available actions
LSTM:	LSTM layer	**u** in LSTM:	LSTM units
$\boldsymbol{V}$:	Value function	$\boldsymbol{s_t}$:	Agent's observation at time t
$\boldsymbol{\pi}$:	Policy	$\boldsymbol{a_t}$:	Agent's action at time t
$\boldsymbol{\phi}$:	Encoder network	$\boldsymbol{\phi(s_t)}$:	encoded state at time t
q:	depends on size of s_t		

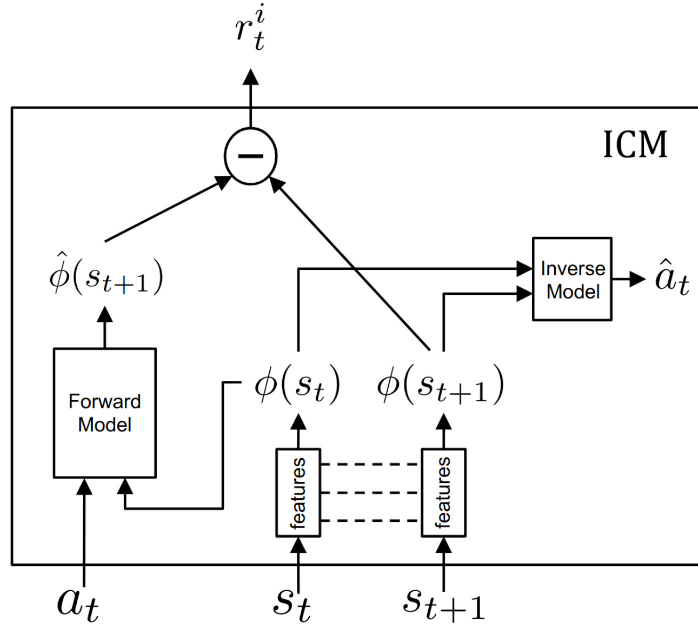


Figure 2: Intrinsic Curiosity Module, image originally from Pathak et al. [40]

output of the inverse dynamics model, so that's one half of the puzzle. As for the other half, to go from a true action to a true distribution, we put the probability of the true action at 1, and 0 for all other actions.

Now the only thing left to do is iterating over our list of true action distribution/predicted action distribution pairs, performing eq. (10). This way, we can determine the cross entropy at each step in time. The inverse dynamics model loss then becomes:

$$L_I(a_t, \hat{a}_t) = H\left(p(a_t), \frac{p(\hat{a}_t)}{1 - p(\hat{a}_t)}\right) \tag{11}$$

Moving on to the next loss function: L_F measures the difference between predicted and actual encoded states, using the (halved) mean squared error as a distance measure. It looks like this:

$$L_F\left(\phi(s_{t+1}), \hat{\phi}(s_{t+1})\right) = \frac{1}{2}\left\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\right\|_2^2 \tag{12}$$

From this, the reward follows - all we have to do is multiply L_F with a scaling factor $\rho_C > 0$:

$$r_t = \rho_C \cdot L_F \tag{13}$$

But wait, isn't the inverse dynamics loss L_I missing from the reward function? The answer is no, we explicitly don't want to use this loss - only to reward

the agent for exploring new states. If we included the inverse dynamics model in the reward function, the agent might be incentivized to seek out areas where it is poor at predicting what effect its actions will have: one of the things we wanted to avoid in the first place!

It should come as no surprise that the agent's policy $\pi(s_t; \theta_P)$ is also encoded in neural network parameters: we call these θ_P. While Pathak et al. [40] use the A3C policy gradient algorithm in their ICM experiments, other algorithms can also be used. The way in which the policy gradient algorithm optimizes θ_P is thus abstracted away: the loss notation only tells us that we need to maximize the expected reward. The following notation is used to represent the expected sum of rewards:

$$E_{\pi(s_t;\theta_P)}[\Sigma_t r_t] \tag{14}$$

To put them all together in one optimization function, we only need two more balancing parameters: the first one, $0 \leq \beta \leq 1$ determines the relative importance of the forward versus the inverse dynamics model. The second one, $\lambda_{ICM} > 0$, determines relative importance of making correct policy gradient predictions versus obtaining the curiosity reward. The full optimization function is thus:

$$\min_{\theta_P,\theta_I,\theta_F} \left[-\lambda_{ICM} E_{\pi(s_t;\theta_P)}[\Sigma_t r_t] + (1-\beta)L_I + \beta L_F \right] \tag{15}$$

While the ICM effectively deals with the Noisy-TV Problem, it seems to fall prey to the Couch Potato problem [5]. To deal with this issue, Burda et al. [6] devised Random Network Distillation (RND). The central idea of RND is that the agent does not learn to predict the future, nor any actions, but instead has to learn the features of a fixed, randomly initialized network f with fixed parameters θ_f. To do this, the agent has an non-fixed network $\hat{f}$ with parameters $\theta_{\hat{f}}$ to learn with. Both f and $\hat{f}$ map the observation space to a k-dimensional space:

$$f : \mathcal{O} \rightarrow \mathbb{R}^k \tag{16}$$

$$\hat{f} : \mathcal{O} \rightarrow \mathbb{R}^k \tag{17}$$

To train $\hat{f}$, gradient descent is used to minimize the expected mean square error:

$$||\hat{f}(x;\theta) - f(x)||^2 \tag{18}$$

To make this more intuitive: imagine an alien giving you a camera that automatically rates any picture it takes, spitting out ratings on a few hundred different properties. However, the ratings are all in the alien's language, and read like "blarp", "flurp" and so on. To find out what these things mean, you start taking pictures.

Soon you find out that all the pictures you are taking are *very* blarp - what could it possibly mean? Any picture not being blarp would be very surprising

indeed. Then you stumble upon the following: when there is a red box in the bottom right corner of the picture, the picture is very much not blarp. This gives you a lot of information: could high blarpness mean the absence of a red box in the bottom right corner of the picture? You should explore further here.

The network f works like the alien camera: it takes in observations, and rates them according to hundreds of different utterly alien concepts. Then, $\hat{f}$ represents you, trying to figure out all these concepts. Getting unexpected results means a high curiosity reward: it's an opportunity to learn more about the alien concepts. However, as you get a better feeling for blarpness in a specific area, the opportunity to learn about it decreases. Hence, over time you get a lower curiosity reward in the same area.

The reason RND dodges the Noisy-TV problem and the Couch Potato problem is because it ignores the time dimension. Recall that stochasticity is the cause of both problems. With RND, all stochasticity of the environment can be ignored, because you're not trying to learn about the environment or its transition function, but about the random initialization of f (the hundreds of blarps and flurps). In doing so, it gives you a good idea about the novelty of any observation, while ignoring its noisy details. This novelty is used as a curiosity reward that is insensitive to noise.

The use of curiosity in multi-agent RL is as of yet a relatively unexplored area. The only publication so far is by Schafer [44], which states that in an environment with partial observability, curiosity can cause learning instability without benefiting exploration. However, in an environment with sparse rewards and full observability, curiosity greatly improves learning stability and final performance. In the work by Schafer [44], the multi-agent curiosity reward is based on the entire environment. Therefore it does not specifically contain a notion of social curiosity. This in contrast to the second type of intrinsic motivation, empowerment. While empowerment can also be based on the entire environment, Jaques et al. [23] developed a type of social empowerment, which we will explore in the next section.

2.2.4 Empowerment

This section contains a brief introduction to empowerment, then covers the formalized notion of empowerment in RL by Klyubin et al. [26] and concludes with the Social Influence model by Jaques et al. [23].

Curiosity is not the only intrinsic motivation we see in nature: humans and animals vie for resources, social status, food, in a seemingly universal drive to have more options for the future. The desire for empowerment is what binds all these together, which was formalized in the context of RL by Klyubin et al. [26].

Empowerment is the concept of *having* different options. Not the actual execution of those options, but merely the potential. Think of a country leader or megacorporation CEO, whose decisions can set thousands or millions of people and machines in motion. They are both highly empowered, and have many options available to them. That doesn't mean they *should* constantly exer-

cise every single high-impact option available to them, unless a nuclear winter sounds cozy to you. More modestly, think of having a full bank account, versus spending everything in it on something ludicrous (and subsequently having a very empty bank account). Empowerment can be seen as the simultaneous presence of two things: the ability to drastically change one's environment, and the changes resulting from different actions being maximally *different* (also called distinct).

Why include distinctness? Consider the following situation: an agent finds a part of the environment that dramatically changes every time step. However, this is just part of the environment, and not induced by the agent's own actions. If there is no distinctness criterium, the agent ends up feeling maximally empowered in this area: every action seems to be followed by some dramatic change. We don't want this - we only want the agent to receive empowerment reward when its actions are the cause of dramatic change. Distinctness helps with this goal, because when there is a very different outcome for every different action, the action actually matters - and vice versa.

To elaborate on the formal definition: Klyubin et al. [26] define empowerment in terms of mutual information, a concept from information theory. Here we give a simplified version - the original is slightly more general, as it can give empowerment over multiple actions over time, but the simplified version maintains the core idea. Given 2 random variables, X and Y, and a probabilistic relationship (also called a *channel*) between the two, defined by a conditional distribution $p(y|x)$. We then want to measure how much more certain you can be about y, given that you know x. This is the mutual information I, measured in bits, and is defined by the following equation:

$$I(X;Y) = \sum_{x,y} p(y|x)p(x) \log_2 \frac{p(y|x)}{\sum_{x'} p(y|x')p(x')} \tag{19}$$

The summation over x' here is really just a summation over x, but denoted with a different variable to distinguish it from the first summation over x, y. With the definition of mutual information I, we can now define the empowerment reward $\mathcal{E}$, for all possible action distributions $p(a_t)$ at time t, the full set of actions A, and the state one timestep later S_{t+1}:

$$\mathcal{E} = \max_{p(a_t)} I(A; S_{t+1}) \tag{20}$$

The reason that we take all possible action distributions, is so that we can ignore what the actual policy is. Only the options the agent has at any given point in time are relevant, not which one it will select!

Control over the environment is not the only valuable type of empowerment: Young children characteristically have very poor fine motor skills - they still lack control over their own bodies. In other words they lack self-empowerment, and through it, they lack environment-targeted empowerment. Unfortunately for them, having half of your meal end up on the floor or on your cheeks is not an efficient allocation of resources. Playing with toys and blocks (and food)

therefore not only teaches them about the world: it also teaches them to control their own bodies. However, it is not just physical, but also emotional control that has to be learned. The ability to push oneself out of harmful or undesirable emotional states is a critical skill for humans to master. Internal states are therefore also a potential target for empowerment.

The final type of empowerment we will discuss here is social empowerment: control over other people (or agents). Jaques et al. [23] made agents learn to predict the actions of other agents, and found that this leads to improved coordination compared to a baseline algorithm without these predictions. But that's not all: when connecting these predictions to the reward by giving agents a desire to influence the behavior of others, even better coordination arises! This is a specific form of empowerment: influence over the behavior of others. They named their specific method Social Influence (SI). It is not empowerment in the usual sense: recall that empowerment originally meant having options rather than exercising them. Social Influence however measures the influence of the actions that an agent *does* take. In their work, you can find the profound insight that through purely self-interested motivation (collect resources, gain power over others), coordination arises.

How exactly do you measure how your behavior influences others? The model of Jaques et al. [23] does this through counterfactual reasoning. This means that the agent asks the question: if I had acted differently, how would the other agents have responded? To be able to answer this question, the agent needs to predict the actions of other agents. This is exactly what the aptly named Model of Other Agents (MOA) does. You can use social influence without a MOA, but doing so requires actors to cheat by looking up the policies directly. In a real-world scenario this would be unrealistic, as the policies of other agents in competitive situations are often hidden.

Using the MOA, an agent k considers all of its own possible actions at time t. We call these possible actions counterfactual actions. Agent k then predicts what the other agents will do at time $t+1$. This process happens independently in each distinct agent: in an environment with N agents, each agent k considers all of its possible counterfactual actions $\tilde{a}_t^k$. For each of these counterfactual actions, for each other agent j, it predicts which action j will take at $t+1$. With this collection of predictions, agent k can calculate a baseline for the behavior of the other agents: the effect of k's actions can be marginalized out.

We want the marginal policy of j, which only reacts to the current observation of j (that is, s_t^j), and does not take into consideration k's action. However, the MOA *does* take into account k's action! Fortunately, the output of the MOA can be further processed to cancel out the effect of k's action. We do this by multiplying the probability of the counterfactual actions by k's policy, given that same action. This process is called marginalization, and through it we obtain the marginal policy of j with the output of the MOA. Think of it as calculating a mean policy for j, which gives the mean reaction of j to k's actions. You can calculate the marginal policy of j as follows:

$$p(a^j_{t+1}|s^j_t) = \sum_{\tilde{a}^k_t} p(a^j_{t+1}|\tilde{a}^k_t, s^j_t)p(\tilde{a}^k_t|s^j_t) \tag{21}$$

Finally, the marginalized policy is compared to the probability distribution of actions taken by j, given the true action of k. The difference between the two measures how much k's actions influence j's policy. As shown below, this influence calculated per agent j, then summed together into the single value i^k_t. The social influence reward i^k_t for agent k is thus:

$$\begin{aligned} i^k_t &= \sum_{j=0, j\neq k}^{N} \left[D_{KL}\left[p(a^j_{t+t}|a^k_t, s^j_t) \middle\| \sum_{\tilde{a}^k_t} p(a^j_{t+1}|\tilde{a}^k_t, s^j_t)p(\tilde{a}^k_t|s^j_t) \right] \right] \\ &= \sum_{j=0, j\neq k}^{N} \left[D_{KL}\left[p(a^j_{t+1}|a^k_t, s^j_t) \middle\| p(a^j_{t+1}|s^j_t) \right] \right] \end{aligned} \tag{22}$$

Here, D_{KL} stands for Kullback–Leibler divergence, a divergence measure that gives an indication of the distance between probability functions. According to Jaques et al. [23], other measures can also be used with similar effect.

To balance the extrinsic vs the social influence reward, 2 scaling parameters are used: ρ_E and ρ_I. Unless otherwise mentioned, the default value for ρ_E is 1. With the extrinsic reward for agent k given as e^k_t, the full reward function is the following:

$$r^k_t = \rho_E e^k_t + \rho_I i^k_t \tag{23}$$

The Social Influence technique was implemented using a neural network, the architecture can be found in fig. 3. It is divided into three parts: a state encoder, an actor-critic, and an action predictor. The state encoder maps observations onto a set of learned features, which then are the input of the other two parts. The actor-critic output consists of the learned value and policy. Finally, the action predictor predicts the actions of other agents at time $t + 1$.

At each step in time, the actor-critic is first evaluated to obtain the current action distribution, which is then sampled to obtain the current action. Then, this action is inserted at the bottom of the model along with other agents' actions, to evaluate the action predictor. This evaluation is repeated for the desired amount of counterfactual actions: this only requires recomputation of the LSTM and the action predictor's output. This gives us the counterfactual predictions needed to calculate the marginal policy of other agents as defined in eq. (22), and thus also the social influence reward.

The loss function is somewhat similar to the one used in the ICM: a combination of a policy gradient loss and action prediction loss (from the MOA). However, it contains no state prediction. While Jaques et al. [23] used the A3C policy gradient algorithm, it can be substituted for another. The MOA loss L_{MOA} is calculated by taking the cross entropy over the true+predicted actions of other agents. This is done in a similar fashion to what happens in the ICM,

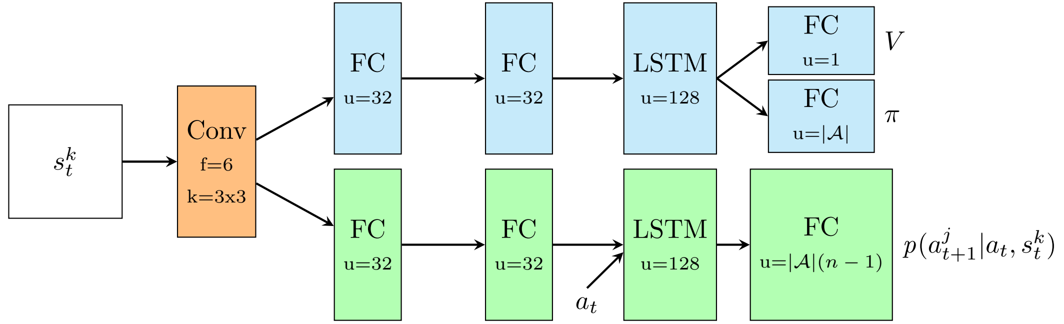


Figure 3: The Model of Other Agents neural network architecture by Jaques et al. [23]. The policy gradient is drawn in light blue, and the MOA in light green. Note how the actions of all agents are fed to the action predictor's LSTM. Stride in convolutions is 1.

Conv: Convolutional layer
f: Convolution filters
k: Convolution kernel size
LSTM: LSTM layer
$\boldsymbol{V}$: Value function
$\boldsymbol{\pi}$: Policy
$\boldsymbol{n}$: Number of agents
FC: Fully connected layer
u in FC: Fully connected neurons
$|\boldsymbol{\mathcal{A}}|$: Amount of available actions
u in LSTM: LSTM units
$\boldsymbol{s_t^k}$: Agent k's observation at time t
$\boldsymbol{a_t}$: All agent actions at time t

as defined in eq. (11). In an n-player environment, given true and predicted actions a^j respectively of agent j, the MOA loss for agent k is then:

$$L_{MOA}^k = \frac{1}{n-1} \cdot \sum_{j=1, j \neq k}^{n} L_I(a^j, \hat{a}^j) \tag{24}$$

Let's call the policy gradient loss L_{PG}. Finally, we need a scaling parameter to balance the two losses, let's call it λ_{MOA}. The social influence loss L_{SI} is then:

$$L_{SI} = L_{PG} + \lambda_{MOA} \cdot L_{MOA} \tag{25}$$

In the next section, we go slightly beyond reward functions, we will cover a neat quirk of NN architectures. Specifically, you will find out how to incentivize agents to learn something, without including it in any reward function!

2.2.5 Auxiliary tasks

Environments can be noisy and contain many superfluous details, which makes tasks harder to learn. To deal with this challenge, agents are usually equipped with a mechanism to encode the environment into a lower-dimensional representation. This encoded representation is then used to calculate the outcomes of the value function and policy. For an example, look at the model in fig. 3.

Encoding is usually done in the form of convolutional layers. Let's look at an example: the Atari experiments in the work of Burda et al. [5]. Say, an observation is a 84x84 observation of RGB pixels, the dimensionality of the observation is thus $84 \cdot 84 \cdot 3 = 21168$. This is reduced to a size of 512 by the use of convolutional layers. This is over 40 times smaller, a huge improvement!

The encoding task itself has to be learned: the agent will have to find a useful representation of the world through experience. However, this depends on receiving a reward signal: the agent can only learn about what is and what isn't a useful encoding as it receives a reward. Thus, this is a point where sparse rewards are especially challenging - conversely, it means intrinsic motivation can shine here! Still, there's more than one way to skin a cat.

As it turns out, both curiosity and empowerment are something called an auxiliary task. These are tasks separate from the main (intrinsic or extrinsic) goal, which still help manage to build a useful representation of the world. This works even when the task itself does not contribute to the reward! Recall how this was the case in the MOA described in section 2.2.4: predicting the actions of other agents led to improved extrinsic reward, even without giving any intrinsic empowerment reward.

What is key here, is the fact that both the auxiliary and the main task *share a representation*. State prediction, as seen in some forms of curiosity, is another example of such a task. In fact, learning to predict any detail of a future (internal or external) state is called an auxiliary task. This holds regardless of whether the task contributes to the reward function.

Jaderberg et al. [21] demonstrated a combination of four auxiliary tasks: to start, it takes the Q-learning loss over 2 types of empowerment. These types are pixel control and network feature control, which respectively correspond to the categories named in section 2.2.4: empowerment over the environment, and self-empowerment. Both of these contribute to the reward function and the loss function. The other two auxiliary tasks are reward prediction, and value function replay. The latter of which recycles experiences several times to learn how to predict the value function output. These only contribute to the loss function. Finally, the model is topped off with an extrinsic A3C reward. That's a lot of different things for a single model to learn, does it even help? The answer is a resounding yes: this model performs far better than the baseline A3C extrinsic model, and each individual component is found to contribute to the final result. This highlights how important it is to build a good representation.

At this point, we still haven't introduced any game theory, and we have yet to cover the concrete games we'll let our agents play. These things are what the next section is all about.

2.3 Multi-agent games

2.3.1 Coordination problems

The problems faced by an agent learning to play Super Mario Bros. were illustrative, but we ultimately want to analyze how multi-agent systems can more

effectively achieve coordination - Mario is only a single agent. We'll need a new environment, so let's ditch Mario for now. To begin, we'll introduce a practical example of a coordination problem.

Let's focus on something mildly catastrophic like overfishing. Overfishing happens when too many fish get caught for a fish population to sustain itself, and both fishermen and fish agree that this is a Bad Thing. Yet, it still happens on a global scale. If no one wants it, why does it still happen?

Individually, any fisherman will agree that inability to catch any fish a year from now is a bad thing. However, if they won't fish today, other fishermen will just catch more. Their own income would go down, while no positive effect on the fish population is achieved. Thus, there is no individual incentive to fish sustainably.

The only way to keep fish populations at a sustainable level is by coordinating with other fishermen: if they all agree to fish less, their incomes will all take a small hit, but in return the fish population can remain stable, allowing them to fish for years to come. Then, the fishermen of the neighboring city/nation/continent notice that there are a lot of fish to be found in the sustainably-fished regions, restarting the process all over again.

Overfishing is a classical example of the commons dilemma (CD). CD-type situations occur when there is a pool of shared resources that can sustain indefinite use up until a certain level. However, when giving everyone unlimited access, the pool is quickly depleted and is exhausted forever. To enable indefinite value, we thus have to coordinate access to the resource pool. Commons dilemmas (CD) are one of the two categories of social dilemmas defined by Kollock [27] - the other type being the public goods dilemma. In commons dilemmas, individuals are tempted by greed, where giving in to greed depletes a shared resource. Public goods dilemmas (PGD) require individual sacrifices to create resources that all can benefit from [27].

Coordination is hard. Really really hard. Unsurprisingly, it's also one of the challenges we face in multi-agent systems. Even in purely cooperative settings, where no agent receives a reward unless everyone coordinates, coordination can be difficult [34]. Never mind the more complex category of CD and PGD problems, where greedy individual behavior leads to higher individual rewards, but the group as a whole suffers and receives a lower collective reward.

With the knowledge of time discounting, it becomes apparent why CD-type situations are so hard: we have to fight the urge to exploit the resources with certainty now, rather than conserve them for an uncertain later time. And yet, coordination of this type can be seen everywhere in nature and human societies. We only really notice it when the coordination breaks down: cancer (cells defecting, gobbling up resources), government corruption, unsustainable agricultural practices, just to name a few.

Back to the overfishing scenario: after the fishermen have forged a cooperative pact to prevent overfishing, a single fisherman decides to not heed it, greedily increasing their own income. Other fishermen notice this, then slowly start to defect, until nobody heeds the pact anymore and the fish population is on the brink of collapse again. Then, everyone agrees that the current situ-

ation is bad and a new pact is formed, hopefully with more options to punish defectors.

This situation is an example of non-stationarity: while an agent learns and changes its behavior, all other agents also adapt. This can lead to endless cycles of adaptation, there may exist no stable global optimum when everyone learns! The challenge of non-stationarity is also called moving target [16] or moving goalposts [39]. Even if agents do learn to cooperate, they might not arrive at an equilibrium where agents have learned the optimal behavior. The game itself may not contain a stable optimal behavior: a type of non-stationarity that cannot be solved!

To be able to reason about coordination problems in the space of RL, we'll need a formal mathematical definition. Luckily, game theory provides us with exactly that: Matrix Game Social Dilemmas. These are covered in the next section.

2.3.2 Matrix Game Social Dilemmas

To simulate coordination problems, Matrix Game Social Dilemmas (MGSD) were conceived [33]. In these 2-player games, the goal for players is to maximize their own score. The reason they are called matrix games is that the game rules are structured as a 2x2 matrix: At every turn, players choose to either cooperate or defect.

When both players cooperate, they both get R (reward), when both defect, both receive P (punishment). When one player cooperates and the other defects, they respectively receive S (sucker) and T (temptation). This, along with the three canonical MGSD payoff matrices, is shown in fig. 4. For a game to qualify as an MGSD, it has to satisfy the following four criteria formulated by Macy and Flache [33]:

- $R > P$ Mutual cooperation is better than mutual defection (26)
- $R > S$ Mutual cooperation is better than being exploited by a defector (27)
- $2R > T + S$ Mutual cooperation is better than a 50/50 chance of unilateral cooperation and defection (28)
- At least one of the following: (29)

— $T > R$ Greed: Exploiting a cooperator is preferred over mutual cooperation

— $P > S$ Fear: Mutual defection is preferred over being exploited.

We can see all these criteria appear in the overfishing example, even though MSGDs only describe a 2 player situation. Inequality (26) means that careful organized conservation of the fishing stock is preferable over everyone fishing as much as they can. Furthermore, inequality (27) means that it's preferable to have organized conservation over a situation in which an individual attempts conservation, but others do not. Inequality (28) informs us about an undesirable

situation: having a 50/50 chance of winning or losing in a miscoordinated conservation strategy where on a given day, one fisherman gets to overfish, and the other does not. Again, organized conservation is preferred over this situation.

Finally, with inequality (29) we see that the fisherman can be incentivized by both Greed: "Everyone else abides by the anti-overfishing pact, so my few extra black market tunas won't hurt, also I want a nicer house." and motivated by Fear: "Everyone else is overfishing anyway, so I should catch as much as I can before everything is gone".

	C	D
C	R,R	S,T
D	T,S	P,P

Chicken	C	D
C	3,3	1,4
D	4,1	0,0

Stag Hunt	C	D
C	4,4	0,3
D	3,0	1,1

Prisoners	C	D
C	3,3	0,4
D	4,0	1,1

Figure 4: Matrix Game Social Dilemma payoff matrices. Top: abstract payoff matrix with outcome variables R, P, S, T mapped to cells of the game matrix. Bottom: the three canonical MGSDs. A cell of X, Y represents outcome X for the player in the left column, Y for the player in the top row. Each game has a unique valid combination of the Greed/Fear properties from eq. (29): Chicken features Greed, Stag Hunt features fear, and Prisoner's Dilemma features both greed and fear. Figure taken from Leibo et al. [28].

Note that MGSDs are Markov Games with $|\mathcal{S}| = 1$, with a specific action set that is identical for both players, each containing both C for cooperate and D for defect: $A_1 = A_2 = \{C, D\}$.

MGSDs have some limitations however, Leibo et al. [28] list the following: real world social dilemmas are temporally extended, and cooperate/defect are not atomic actions like in MGSDs. This leads to the following issues: we have to apply the labels of cooperate and defect to policies. These are generally not strictly pure cooperation or pure defection, but a mix of the two. This can be seen as a graded quantity: cooperativeness. In the real world, a decision to cooperate or defect is only made quasi-simultaneously, agents may react to each other's behavior as they observe the start of a cooperative or uncooperative action. In MGSDs, decisions are instead made at exactly the same moment.

Additionally, there is the problem of partial observability: in real-world scenarios, agents cannot observe the entirety of the environment, nor the activities of other agents at all times. In MGSDs, both players can always see the entire environment and activities of the other player. Finally, MGSDs only support 2 players: real-world social dilemmas tend to exceed 2 parties, and we would like to model this as well.

To address these issues, Leibo et al. [28] created Sequential Social Dilemmas (SSDs), and Hughes et al. [20] extended them to n-player games. This extension is the next section's topic.

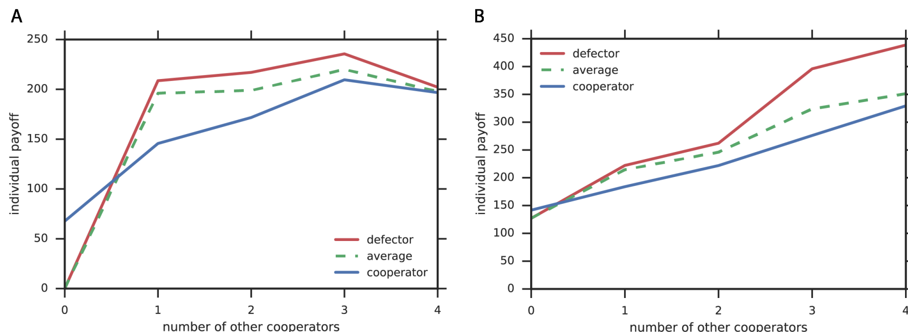


Figure 5: Schelling diagrams derived from Cleanup (A) and Harvest (B). The dotted line shows the average return for all players, were you to choose defection. Figure taken from Hughes et al. [20].

2.3.3 N-player Sequential Social Dilemmas

Sequential Social Dilemmas (SSDs) as defined by Leibo et al. [28] extend MGSDs by introducing properties of Partially Observable Markov Games: that is, the concept of time, and partial observability. Furthermore, in SSDs cooperativeness is a graded quantity, instead of having cooperation as a single atomic action. Intuitively, they resemble the overfishing problem, although variations that only feature either Greed or Fear are possible.

Still, the SSDs Leibo et al. [28] are only 2-player: generalizing them to N-Player SSDs (NPSSDs) was done by Hughes et al. [20]. In the rest of this section we mostly summarize the definitions given by Hughes et al. [20]. We end up with a definition of NPSSDs that is similar to, but does not precisely mirror the MGSD inequalities.

With n players in an MGSD, the payoff matrix becomes n-dimensional, which quickly becomes unwieldy and hard to interpret. Instead of a payoff matrix, we can use a Schelling diagram to visualize the payoff structure [41]. This diagram plots the payoffs for an individual cooperator or defector against the number of other cooperators. Schelling diagrams for cleanup and harvest can be found in fig. 5.

With these diagrams, we can define the n-player sequential social dilemma: a tuple $(\mathcal{M}, \Pi = \Pi_c \sqcup \Pi_d)$ of a Markov Game $\mathcal{M}$ and two disjoint sets of policies $\Pi_c \sqcup \Pi_d$, respectively implementing cooperation and defection. This tuple is a n-player sequential social dilemma if equations 30-32 are satisfied.

To define how many cooperators and defectors we have, we create the strategy profile $(\pi_c^1, \ldots, \pi_c^\ell, \pi_d^1, \ldots, \pi_d^m) \in \Pi_c^\ell \times \Pi_d^m$, where $\ell + m = n$. The average payoff of a cooperating policy is denoted $R_c(\ell)$; for a defecting policy $R_d(\ell)$. When ℓ players cooperate and $m = 1$, the Schelling diagram plots the curves $R_c(\ell+1)$ and $R_d(\ell)$. We can now look at these diagrams to see if the inequalities 30-32 hold:

- Mutual cooperation is preferred to mutual defection: $R_c(n) > R_d(0)$ (30)
- Mutual cooperation is preferred to being exploited by defectors:
  $R_c(n) > R_c(0)$ (31)
- At least one of the following: (32)
  - Greed: Exploiting a cooperator is preferred to mutual cooperation:
    $R_d(i) > R_c(i)$ for sufficiently large i
  - Fear: Mutual defection is preferred to being exploited:
    $R_d(i) > R_c(i)$ for sufficiently small i

Note that these inequalities cannot be used to distinguish between public goods and commons dilemmas! Still, it's a start. In the next section we can discuss two relevant examples of NPSSDs: cleanup and harvest.

2.3.4 Multi-agent games: cleanup and harvest

The concrete testing environments we use are cleanup and harvest. These are the same as the two used by Jaques et al. [23], and were introduced by Hughes et al. [20]. The reason these two specific games are used is that they fall under the two distinct categories defined by Kollock [27]: commons dilemmas, and public goods dilemmas. Specifically, harvest is a commons dilemma, cleanup is a public goods dilemma.

Both cleanup and harvest are gridworlds that obey POMG rules. Gridworlds are discrete rectangle-shaped worlds in which players are placed. Players have an action set that allows them to move through, and/or interact with the world. The observation function in the harvest and cleanup gridworlds allows agents to observe everything happening in a square 15×15 grid surrounding them.

In the public goods game cleanup, reward is earned by collecting apples from a gridworld. Each apple provides 1 reward. However, next to the apple spawn area, there is a river that slowly accrues pollution. When there is no pollution, the apple spawn rate is maximal. It linearly decreases as the river gets more polluted. The game starts with the river being just polluted enough for apples to stop spawning entirely. Players can clean the river with a cleaning beam, but this in itself provides no reward.

The goal of the harvest game, a commons dilemma, is also to collect apples which each provide 1 reward. However, here the apple spawn rate is determined by the amount of uncollected apples. The more uncollected apples are nearby, the more quickly new apples spawn. When all apples are collected, none will ever grow back.

Both games include a punishment beam in each players' action set: at the cost of -1 reward, players can fire a beam that makes any agent it hits lose 50 reward. This is included to emulate punishment mechanisms which are necessary in human sequential social dilemmas [20].

Figure 6: The two n-player SSD environments cleanup (left) and harvest (right). Figure taken from Hughes et al. [20].

To deal with these dilemmas, to overcome greed and fear and to achieve coordination in RL, we are now equipped with curiosity, empowerment, and auxiliary tasks. In the next chapter, you'll find out how we build upon existing intrinsic motivation models which do exactly this.

3 Proposed model

3.1 Motivation

At some point we've all been curious about the inner workings of another's mind - and the behavior stemming from it. In fact, in every interaction people constantly try to predict how others will act: "My partner will appreciate me buying them flowers", or "If I poop on the desk of my superiors, they will be upset". Using these predictions, you decide what, and what not to do. Making a correct prediction is the norm, not an exception. Inversely, think of how scary a wildly unpredictable person can be. If you are acting unpredictably in public, people tend to call emergency mental health services, who will then assist you with a healthy dose of sedatives and antipsychotics.

This demonstrates that learning to predict how others act is vital to healthy functioning in any social relationship. Children are intrinsically motivated to form social connections and learn social behaviors on their own: making friends, inviting other children over to play. No external reward is needed. Though sating curiosity is not the only motivation children have to socialize, learning from others and discovering new things is an important component. In fact, even adults often socialize to sate their desire for social novelty. Think of workplace gossip, where one of the main activities is talking about the actions of other people, while reasoning about their motivations.

Therefore, in RL the learning of social predictions should be rewarded. Specifically, we want agents to be rewarded for learning what effect their behavior has on others. To this end, we introduce the Social Curiosity Module.

3.2 Social Curiosity Module (SCM)

This chapter details our proposed method. In this section we introduce the model architecture of our novel contribution: the Social Curiosity Module. The

next two sections detail its loss and reward functions.

Given the Intrinsic Curiosity Module by Pathak et al. [40] and the social influence reward by Jaques et al. [23] found in respective sections 2.2.3 and 2.2.4, we set out to combine the two. We name this combination the Social Curiosity Module (SCM). Note that the model by Jaques et al. [23] already contains the MOA, which predicts the actions of other agents. However, it does not predict its own internal encoded features, like the ICM. The idea here is to extend the model so that it does.

There are, however, some caveats. To recap: the ICM can neatly isolate the parts of the environment that it can control through its encoding function ϕ. Consider the following two predictions the ICM makes in order to train ϕ:

$$P\left(\phi(s_{t+1}^k)\middle|\phi(s_t^k), a_t^k\right) \tag{33}$$

$$P\left(a_t^k\middle|\phi(s_t^k), \phi(s_{t+1}^k)\right) \tag{34}$$

The former (forward prediction, eq. (33)) produces the curiosity reward. Consider how when ϕ encodes everything to 0, predictions are trivial: predict 0 and you're always correct. Therefore, this prediction will incentivize the neural network optimization process to make ϕ encode as little state information as possible. This is because the encoding function ϕ is on the left side of the conditional probability.

The latter (inverse prediction, eq. (33)) dictates what the agent should be curious about, and forces ϕ to encode some type of information. The ICM makes agents curious about parts of the environment that it can control, by letting ϕ only encode information about those parts.

As for the SCM, we would like to isolate the behavior of others that we can control. This might sound manipulative, but control is inherent in social interaction. For example, dialing a person's phone number (occasionally) causes them to pick up the phone. It is not about manipulation per se, but about the flow of action-reaction in a social interaction. Isolating social control with an encoding function ϕ is much harder, however. This is because the influence relation is indirect: the action of k influences the next state, which then influences the action of j at the next timestep.

It does not seem sensible to predict j's actions directly from k's actions. Although this would isolate the influence, making accurate predictions would be very difficult without the context provided by the state. For instance, an agent collecting an apple right next to it is easy to predict if you can observe the state. If you cannot, then it's very hard.

Why not just combine the action and state, then predict actions from that? The reason is that it would make it hard to isolate the influence. The state has a big influence on j's behavior, and learning moments for these are not sparse - any time we observe another agent, we can learn about them reacting to the environment. It is social interaction and influence that is sparse, that is what we want our agents to be curious about.

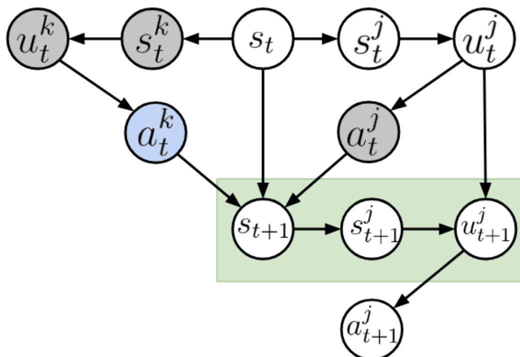


Figure 7: The MOA causality diagram. Gray-shaded nodes are inputs to the MOA. Nodes boxed in green are implicitly modeled by the MOA to predict a_{t+1}^j. Image taken from Jaques et al. [23].

If only we could isolate influence of k on j - but wait, that's what the MOA does! Jaques et al. [23] state that in order to isolate the causal effect of a_t^k on a_{t+1}^j, the MOA of agent k must implicitly model both the environment transition function, as well as the hidden LSTM output u_t^j of agent j. This is shown in fig. 7.

Ideally, we would eliminate having to model the environment transition function in the curiosity function, as this is not what we'd like to be curious about. Doing so would make for better isolation of the social influence prediction task. Luckily, in the SCM we can do this to some extent! This is because the inverse model doesn't need to supply real-time reward information, it is only used to train the encoding function ϕ. This means we can supply s_{t+1}^k to the inverse model, and eliminate part of the environment transition function prediction. Note that the simultaneously deployed MOA cannot do this (theoretically it could, but then it would be leaking information).

There is a caveat here: we are still working under partial observability. The state of j (and the overall environment state) thus still has to be inferred to the degree that it is not visible. According to eq. (22), all information used to compute i_t^k is the set $\{s_t^k, u_t^k, a_t\}$. Here, the hidden LSTM output of agent k at time t is denoted u_t^k. Implicitly, $\{s_{t+1}^j \in s_{t+1}, u_{t+1}^j\}$ are modeled, as seen in fig. 7. We supply s_{t+1}^k, and this value can overlap with s_{t+1}^j. This means we only have to model those parts of s_{t+1}^j that are not already in s_{t+1}^k, expressed as $s_{t+1}^j - s_{t+1}^k$. Therefore, we only have to implicitly model the following:

$$P\left(s_{t+1}^j - s_{t+1}^k, u_{t+1}^j \middle| s_t^k, s_{t+1}^k, u_t^k, a_t\right) \tag{35}$$

Why do we condition on u_t^k? According to fig. 7, future actions and states depend on the agent's hidden LSTM output u_t^k, but also on other agents' hidden LSTM outputs u_t^j. To ensure that the SCM does not get tripped up by this

temporal dependency, we supply u_t^k output to the forward model. Even though we don't have access to other agents' internal states, the MOA should model at least part of them, as that is its function. We need to be careful here: when neural network gradients are computed for the MOA, the connection between the hidden LSTM output and the forward/inverse models should be deliberately ignored. Otherwise, the forward and inverse model would distort the MOA's behavior of learning to predict future agent actions, which is not something that we want to happen.

The action predictions a_{t+1}^j are used by the MOA to calculate the influence i_t^k, as demonstrated in eq. (22). Because we want the agent to be curious about the sparse social influence reward, let's also make a social influence prediction:

$$P\left(i_t^k \middle| \phi(s_t^k), \phi(s_{t+1}^k), a_t, u_t^k\right) \tag{36}$$

This is the SCM inverse prediction. Now, ϕ will only encode information pertaining to k influencing agent j. This is a similar to the ICM only encoding information that pertains to parts of the state that agent k can control. It is not exactly the same: c is not an action, and thus has no direct causal relationship between the two states. It is, however, based on the action that has been chosen.

There is another way to reason about this prediction: because the neural network is supplied with a_t and u_t^k, it is implicitly trying to learn the algorithm of calculating social influence. This entails inferring the action distributions of other agents, and learning eq. (22). However, the agent's own distribution will have to be inferred as well, as we only supply the action, not the action distribution.

This prediction also resembles the reward prediction by Jaderberg et al. [21], although we don't skew the samples to over-represent rewarding events, and we only predict the intrinsic reward.

Now that we know the predictions on a conceptual level, we can define our neural network architecture and loss functions. This is the territory of the next section.

3.3 SCM loss function

The complete SCM loss function includes a policy gradient loss L_{PG}, and the MOA loss L_{MOA} defined in eq. (24). The curiosity component requires a forward model loss L_F as defined in eq. (12) and an inverse model loss function L_I - we only have to newly define the last one.

Let's look at the loss function L_I. Recall that the social influence reward i_t^k is the sum of agent k's influence over all other individual agents, shown in eq. (22). Therefore, it outputs a single value at every timestep t. As this is a single real value, and not an action distribution, this loss becomes the the squared error. Given social influence i_t^k and predicted social influence $\hat{i}_t^k$ for agent k at time t, we can calculate the loss L_I:

$$L_I(i_t^k, \hat{i}_t^k) = (i_t^k - \hat{i}_t^k)^2 \tag{37}$$

As for the forward model, it is not that different from the ICM - we only have to add two variables to the right side of the conditional probability equation: the hidden LSTM output u_t^k and social influence i_t^k. With these, agent k can make the following prediction:

$$P\left(\phi(s_{t+1}^k)\middle|\phi(s_t^k), a_t, u_t^k, i_t^k\right) \tag{38}$$

Then, let ϕ be the true encoded state representation, and $\hat{\phi}$ be the predicted encoded state representation. We can then use L_F in eq. (12) to calculate the forward model loss. With the forward loss, the curiosity reward can then be given in the same way as in the ICM, see eq. (13).

The SCM neural network architecture found in fig. 8 is a combination of the ICM (fig. 1) and the MOA (fig. 3). In order to make ϕ only encode features relating to influence, we cannot join all encoding networks. Instead, there are two: the policy gradient/MOA encoder, and the curiosity encoder ϕ.

Just like in the ICM and MOA, the loss functions need to be balanced. To do this, we use three parameters: $\lambda_{MOA} > 0$ and $\lambda_{SCM} > 0$ determine the relative importance of respectively the MOA and curiosity module loss. Just like in the ICM, $0 \leq \beta \leq 1$ determines the relative importance of the forward versus the inverse dynamics model.

$$L_{SCM} = L_{PG} + \lambda_{MOA} \cdot L_{MOA} + \lambda_{SCM}\big(\beta \cdot L_F + (1-\beta)L_I\big) \tag{39}$$

We parameterize these functions in the neural network displayed in fig. 8, with the goal of minimizing the above function. Now that we have this network and its associated loss function, all that is left is to flesh out the reward function, which we will do in the next section.

3.4 SCM reward function

Ultimately we would like to use both empowerment and curiosity at the same time. After all, these drives are simultaneously present in humans as well. To do so, we create a reward function r_t^k containing the extrinsic reward e_t^k, the curiosity reward c_t^k found in eq. (13), and the social influence reward i_t^k found in eq. (23). To balance the intrinsic motivations, we use two parameters: $\psi \geq 0$ and $\rho \geq 0$. These respectively scale the curiosity and influence rewards. Correct values for these parameters will be have to experimentally discovered. The final reward function is then:

$$r_t^k = e_t^k + \psi \cdot c_t^k + \rho \cdot i_t^k \tag{40}$$

By setting ρ to 0, we create a version of the SCM that only rewards agents for curiosity about social influence, while not rewarding them for taking influential actions. From here on out we will call this version of the model the "SCM without influence reward".

When using social influence, the influence reward is only given when other agents are in view. This is done because the MOA implicitly models both other

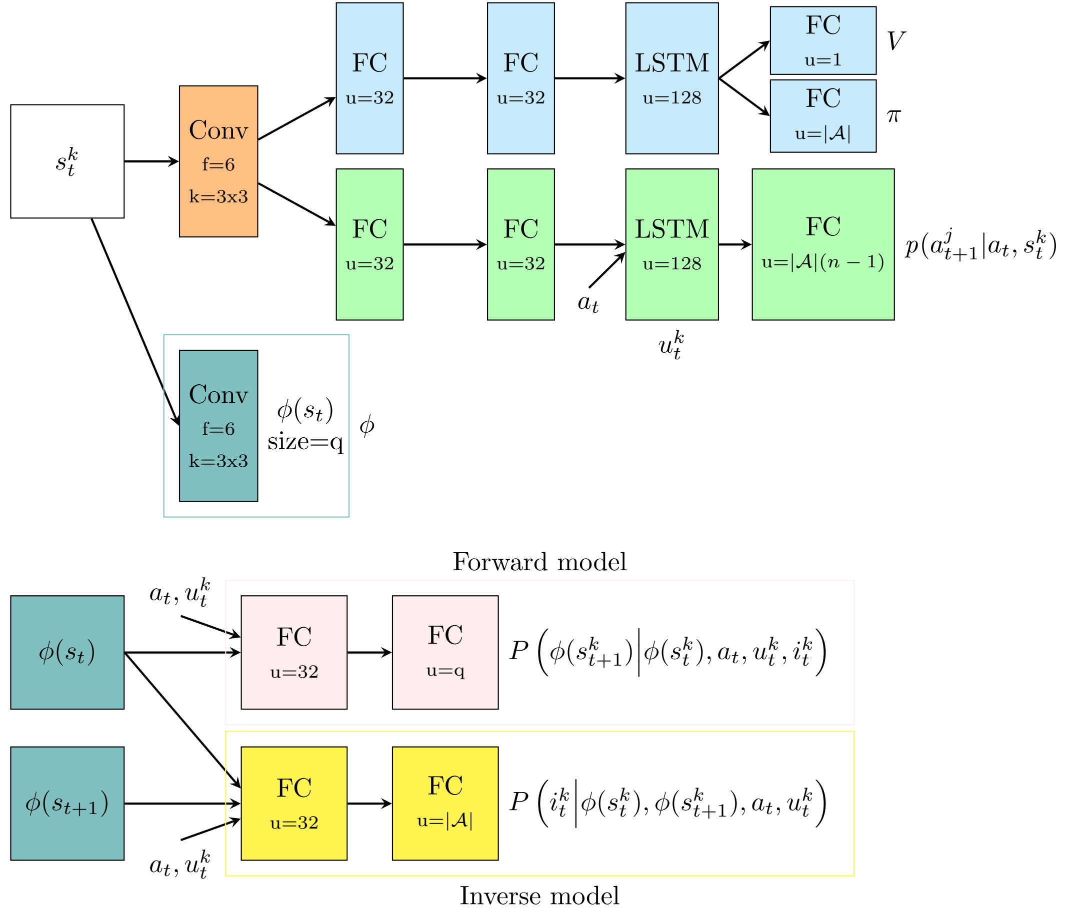


Figure 8: The Social Curiosity Module (SCM), a combination of the MOA and ICM. The policy gradient is drawn in light blue, and MOA in green. The encoder is drawn in teal, the forward model in pink, and the inverse model in yellow. The total amount of neurons used for the curiosity component are greatly reduced compared to the ICM, as the ICM worked with a higher-dimensional state than SSDs. Note that u_t^k is the hidden *output* of the MOA LSTM, which is equivalent to the hidden MOA LSTM *state* at $t+1$. Stride in convolutions is 1. The activation function for convolutional layers is relu, for dense layers it is tanh. The model outputs are exceptions: the activation function for the value function and policy layers is linear, and the last layers of both the inverse and forward model use a relu activation function.

Conv:	Convolutional layer	**FC**:	Fully connected layer
f:	Convolution filters	**u** in FC:	Fully connected neurons
k:	Convolution kernel size	$\boldsymbol{|\mathcal{A}|}$:	Amount of available actions
LSTM:	LSTM layer	**u** in LSTM:	LSTM units
$\boldsymbol{V}$:	Value function	$\boldsymbol{s_t^k}$:	Agent k's observation at time t
$\boldsymbol{\pi}$:	Policy	$\boldsymbol{a_t}$:	All agent actions at time t
$\boldsymbol{\phi}$:	Encoder network	$\boldsymbol{\phi(s_t)}$:	encoded state at time t
q:	depends on size of s_t^k	$\boldsymbol{u_t^k}$:	LSTM hidden output at time t
$\boldsymbol{n}$:	Number of agents	$\boldsymbol{i_t^k}$:	SI reward for agent k at time t

agents' internal states and behavior, in addition to the environment transition function [23]. An inaccurate model would lead to noisy estimates of the influence reward. This reduces what we need to implicitly model: accurately predicting the full state s_{t+1} is less important if we receive no influence reward from agents that are far away. This goes for both the MOA and the SCM, which is a good outcome, as it makes the model easier to learn.

Some caveats: making the influence reward depend on proximity could have the side effect of encouraging agents to cluster together, which was already noted by Jaques et al. [23]. They deemed it a reasonable trade-off, because humans like to spend time with each other as well. Furthermore, the Couch Potato Problem should prove to be less of a problem: after all, agents do not have a "remote control" that generates unpredictable behavior, so there should be no way to get stuck endlessly watching an agent over which you have perceived influence.

This concludes the proposed model - in the next chapter we lay out the experiments we will perform using this model, and the reasoning behind them.

4 Method

This chapter contains two sections: in the first section we describe and motivate the choice of experiments. In the second section we discuss and explain the choice of hyperparameters.

4.1 Experiments

Every combination of model $\{\text{baseline}, \text{MOA}, \text{SCM}\}$ and environment $\{\text{cleanup}, \text{harvest}\}$ is tested five times each. We also test the SCM without influence reward in the cleanup environment five times. This makes for a total of $(3 \cdot 2 + 1) \cdot 5 = 35$ runs, where each run goes on for 5e8 environment steps. Each individual run has a different random seed, and features 5 distinct agents.

In order to verify the foundation of the SCM, several of the experiments found in Jaques et al. [23] are reproduced. A substantial amount of work and effort went into this, spanning over 500 git commits, over 7500 changed lines of code and extensive testing. To be specific, we reproduce 4 of their experiments: using two of their models, each on the harvest and cleanup environments. The first model is the baseline model, a version of fig. 3 where the bottom row has been removed, and only the convolution on the left, and top policy gradient layers are kept. This model only takes extrinsic reward into account, as such the loss of this model is purely its policy gradient loss. The second model is the MOA with SI, as detailed in section 2.2.4. After these experiments have been reproduced, the SCM is tested on both the cleanup and harvest environments. As with the MOA, each individual agent has their own full SCM model, and therefore their own separate neural network. There are some notable changes: Instead of A3C we use the PPO algorithm, as it performs better than, and exhibits more stable learning behavior than A3C [47].

Our goal is to make agents cooperate, despite the presence of social dilemmas. Cooperation leads to a higher collective reward (the sum of individual agent rewards), as demonstrated with the Schelling diagrams in fig. 5. Therefore, we will look at the collective reward as a measure of progress. The agents individually optimize for their own reward, not for a collective reward. They are hence expected to be less able to effectively learn cooperation when using a model that cannot deal with social dilemmas. This should be the case with the baseline model.

Results will be shown as graphs of mean extrinsic collective episode reward over time, with the mean of all 5 experiments as a darker line, and individual experiments as lighter lines. Episodes last 1000 steps, the same as the episodes of Jaques et al. [23]. Each light line is an individual experiment run, showing the average collective reward. Each data point of the light lines is an average over 96 episodes, and thus 96000 environment steps. This number stems from the training batch size described in the following section. This gives us no proof that the model always performs this way under the same circumstances, but it is an empirical sample.

To plot the dark-lined mean, the mean of 1 iteration is calculated over 5 experiments. Each of these experiments reports the mean extrinsic reward over 96 episodes. Therefore, each data point represents the mean of $5 \cdot 96 \cdot 1000 = 480000$ environment steps, distributed over 5 experiments. For these data points, the unbiased variance estimator of the collective reward is also calculated. Using this, we calculate a confidence interval and plot it as bands with the same color as the model. For 4 degrees of freedom, two-sided confidence level 95%, the critical t-value is 2.776.

Because we will compare the mean performance of different models by using samples, we need to establish whether these samples are sufficient evidence to explain any differences we might find. In order to do so, the collective reward scores of the final iteration are compared using the student's t-test. Specifically, we will perform independent two-sample t-tests for each combination of two models per environment. Note that the degrees of freedom for these tests are 8 instead of 4 in the plots, because we use data from two experiments simultaneously. This brings the critical t-value for these tests to 2.306.

To be specific about how the t-values are calculated: for each model, we collect the mean collective episode extrinsic reward over the last 96 episodes. We do this for 5 experiments each, giving us 5 mean values per model. Using the mean values, the unbiased estimation of the variance is also calculated. With two model means μ_0 and μ_1, and respectively two unbiased variance estimators s_0 and s_1, we can estimate the pooled standard deviation S_p, which is then used to calculate the t-value t:

$$S_p = \sqrt{\frac{s_0 + s_1}{2}} \tag{41}$$

$$t = \frac{\mu_0 - \mu_1}{S_p * \sqrt{\frac{2}{5}}} \tag{42}$$

According to Colas et al. [11], Welch's t-test would be a better choice, because it makes no assumptions about the sample variance, unlike the student's t-test, which assumes them to be equal. However, the sample size per experiment is only 5, whereas Welch's t-test requires more than 5 samples per population. The additional recommendation of Colas et al. [11] to use 20 samples or more was computationally infeasible for us. In retrospect, their final recommendation of using a confidence interval with higher certainty than 95% is not relevant, because even at a 95% confidence level we find no rejection of null hypotheses.

4.2 Hyperparameters

Hyperparameters are values that control aspects of the learning algorithm such as learning rate, episode length, loss coefficients and learning schedules. It is known that the models are sensitive to hyperparameter selection [23], changing one hyperparameter can drastically alter an experiment's outcome. It is hard to find well-performing hyperparameters. The method employed by Jaques et al. [23] to deal with this is to perform a grid search over the hyperparameters. This however, is prohibitively computationally expensive for us. Selecting the same hyperparameters does not give any guarantees about the outcome. The hyperparameters provided by Jaques et al. [23] point in a general direction of acceptable hyperparameters for the MOA, and as such we copied them. Instead of grid search, the more efficient Population Based Training as proposed by Jaderberg et al. [22] was attempted, but this still proved to be too computationally expensive to find better hyperparameters than the ones copied from Jaques et al. [23]. Another weakness of this approach is that we employ PPO, not A3C, a different training algorithm. A different training algorithm is almost guaranteed to require different values for the hyperparameters. However, as PPO has shown to be less sensitive to hyperparameter tuning, we opted for PPO over A3C.

Training algorithms come with their own sets of hyperparameters that are recommended by their respective authors [47]. Because the value function and policy share parameters, PPO requires that we provide a coefficient to scale the value function, otherwise one can end up heavily dominating the other's loss. The default provided value of 1 turned out to cause unstable learning, making the value function loss far too high. We can assume this is because the social influence reward is never easy to predict. At a value of 1e−4 it seemed to stabilize, hence this is the value we used for all experiments. The social influence work by Jaques et al. [23] made no mention of LSTM sequence lengths, hence we used the default value provided by the library used: 20. As seen in the model diagram in fig. 8, the LSTM cell size is 128, the same as in the work of Jaques et al. [23].

As for the SCM hyperparameters, the work of Pathak et al. [40] gives us a clue: they multiply the forward versus inverse loss with respectively 0.8 and 0.2. They also multiply the policy gradient loss by 0.1 - in other words, the curiosity reward is weighed 10 times heavier than the policy gradient loss. However, no reasoning is given for these numbers. A more thorough study of different

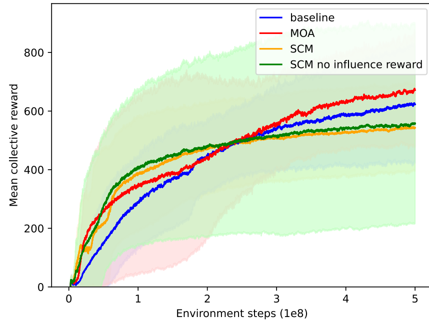
hyperparameters should be conducted to truly evaluate the performance of the SCM.

At the start of the learning process the behavior of all agents is essentially random. We cannot predict anything, does the Noisy-TV Problem appear again? Fortunately no: since we're only curious about influence this is not a problem, just like with the ICM. The SCM avoids this by ignoring noise in other agents' behavior. Still, at the start of training, both the social influence calculation and inverse model will essentially emit noise, until they have learned something after a few training iterations. This is why we use a learning schedule, where the curiosity reward is multiplied by a scaling factor σ_r. Until environment step 10^5 it is set to 0. From step $10^5 - 10^6$ it is linearly increased to 1, where it remains for the rest of the learning process. This schedule is based on the MOA learning schedule by Jaques et al. [23].

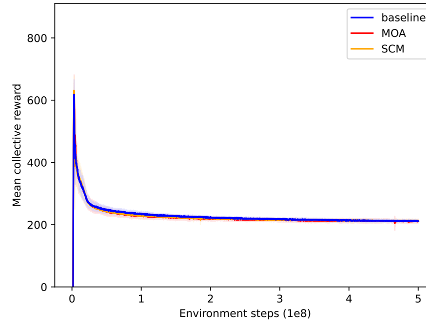
Two final hyperparameters that influence the outcome of the result are the training batch size and specifically for PPO, the minibatch size. Training batch size determines how many environment steps are made with the current model before the loss is calculated and the model is updated. With PPO, the minibatch size determines the size of smaller experience batches that are repeatedly used for model optimization. Larger batches tend to converge more slowly, but make learning more stable. The training batch size was not provided by Jaques et al. [23], as such we could not copy it. In all our experiments the training batch size is set to 96000 and the PPO minibatch size is set to 24000. The training batch size was made as large as possible while still allowing the experiments to fit in memory.

The other hyperparameters used for our experiments can be found in the appendix, in figures 10, 11 and 12. A legend for the hyperparameter tables can subsequently be found in fig. 13.

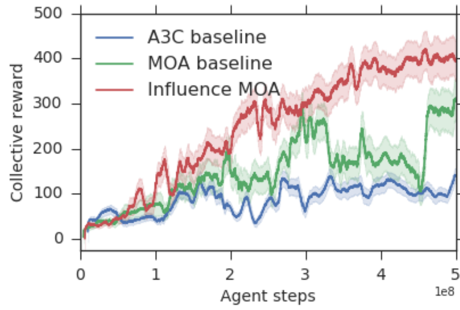
5 Results



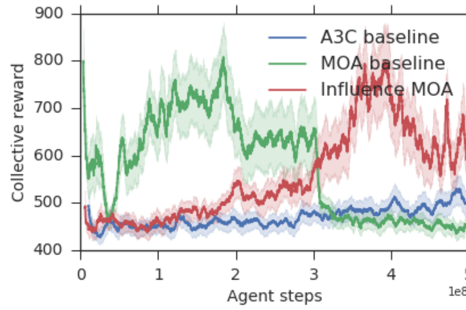
(a) Cleanup mean collective reward



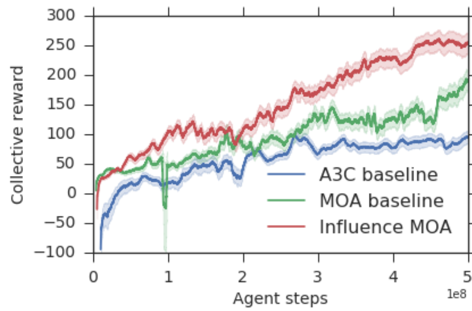
(b) Harvest mean collective reward



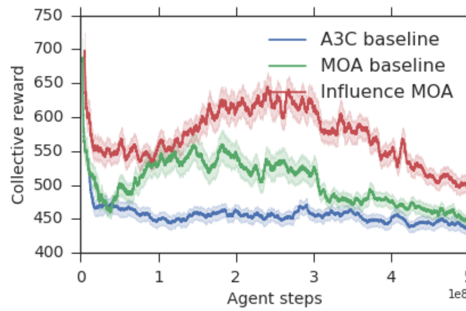
(c) Cleanup, results from Jaques et al. [23]



(d) Harvest, results from Jaques et al. [23]



(e) Cleanup with distinct top 5 hparams, results from Jaques et al. [23]



(f) Harvest with distinct top 5 hparams, results from Jaques et al. [23]

Figure 9: Mean collective extrinsic reward. (a) and (b) are our experiments, showing 5 experiments with 5 random seeds, with unoptimized hyperparameters. (c) through (f) are taken from Jaques et al. [23] for comparison. (c) and (e) depict 5 experiment runs with 5 random seeds each with the set of best found hyperparameters per model, a total of 5 run per model. (e) and (f) depict the 5 experiments with 5 random seeds with the best 5 sets of hyperparameters, a total of 25 per model.

This chapter details the experiment results. A comparison is made between our results, and the results of Jaques et al. [23]. This is followed by the individual experiments. We conclude with a statistical test of our results, comparing our different models.

5.1 Reproduction comparison

Striking differences are immediately visible in fig. 9. Cleanup performance is far better and harvest performance is far worse for all models when compared to the results of Jaques et al. [23]. Our confidence bands in fig. 9 (a) are far wider than those of Jaques et al. [23] in fig. 9 (c), despite being more lax (ours being 95% instead of 99.5%). However, there are some issues with directly comparing our results to those of Jaques et al. [23]. First of all, we use PPO instead of A3C, without tuning our hyperparameters. Secondly, it is not explicitly mentioned what the confidence intervals in the graphs by Jaques et al. [23] depict. In private correspondence, Jaques mentioned that it depicts the 99.5% confidence interval that the rolling mean falls in that range. However, the exact procedure of calculating these confidence intervals is not known, as some parameters are missing. In the work by Jaques et al. [23], no statistical testing was mentioned aside from plotting the confidence intervals, as such we cannot compare our results with statistical testing.

For each individual experiment, we also rendered videos of the agents' behavior with one of the last models obtained (training iteration 5200 out of 5204). By inspecting these videos, we found that in cleanup, usually only one agent would consistently clean the river aside from collecting apples, which is consistent with the distribution of agent rewards found in our experiment data. In the highest-performing models for cleanup, we saw two agents both cleaning the river and collecting apples, while the others always only collected apples. No more than 2 agents cleaning the river in one experiment were seen. As for harvest, no matter the model, the agents would always collect all apples from the get-go and leave none to respawn, which also matches with our experiment data.

The manner in which our confidence bands were calculated can be found in section 4.1. To expand on why we cannot directly compare them to the work of Jaques et al. [23]: the sliding window used by Jaques et al. [23] is reported to be over 200 environment steps versus 96000 in ours, but it is unknown what the training batch size of Jaques et al. [23] is, nor is it clear whether this actually concerns 200 training iterations, or steps over 5 or 25 experiments. We inquired in private correspondence, but the data was no longer accessible. Therefore, the confidence bands cannot be directly compared. We also attempted a sliding window approach over our data for a window size of 5, 10, 100, 200 and 1000 iterations, but this did not change the results in a significant way. In fig. 9, (e) and fig. 9 (f) depict the results of Jaques et al. [23], of 5 distinct sets of top performing found hyperparameters per experiment, with 5 experiment runs with different random seeds each. As such, fig. 9 (e) and fig. 9 (f) depict 5 times more runs in total than fig. 9 (c) and fig. 9 (d), explaining the relatively

narrower confidence bands.

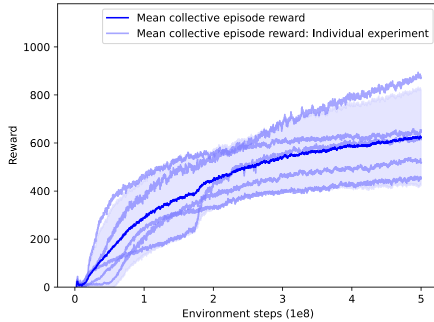
With harvest, our results in fig. 9 (b) show a stark difference compared to the top hyperparameters found by Jaques et al. [23] in fig. 9 (d), as all of the tested models perform in the same way, far worse than even the A3C baseline by Jaques et al. [23]. Here the most likely explanations are either the use of PPO instead of A3C, or a very bad choice of hyperparameters. The latter is mentioned as a likely explanation because it is easy to choose bad hyperparameters, and hard to choose good ones. However, because the results show such an exceedingly narrow confidence band across models, the former explanation is preferred. Because performance does not seem to differ between models, we will not discuss harvest performance individually in later sections.

The only situation in which harvest agents performed better was found during an experiment featuring exceedingly high social influence reward and MOA loss (not depicted). The agent would receive roughly 40x more MOA reward than extrinsic reward, causing the value function loss to balloon. This happened because the value function would be mainly trying to predict MOA rewards which cannot be reliably learned when all agents condition their own behavior on their social influence. This is an extreme version of the moving goalposts problem. The total loss would hover around 15 - enough to learn to move randomly and not shoot, but not enough to overfit on a greedy strategy, causing a high collective reward in harvest. Unlike in the work of Jaques et al. [23], the situation in which one agent fails to learn, enabling all other agents to harvest sustainably, did not occur. As such, no pruning of random seeds was necessary.

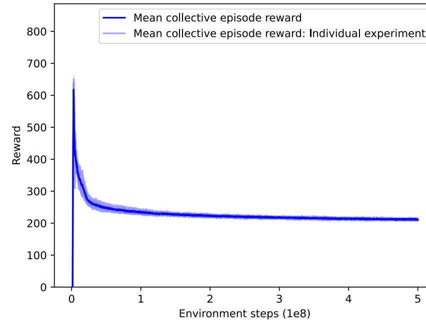
You could argue why comparing fig. 9 (a) with fig. 9 (e), and fig. 9 (b) with fig. 9 (f) would make more sense: despite fig. 9 (e) and fig. 9 (f) showing 5 times more experiments than fig. 9 (a) and fig. 9 (b), they do not work exclusively with the optimal found hyperparameter set, but instead the top 5 distinct sets. Because the experiments in fig. 9 (a) and fig. 9 (b) received no hyperparameter tuning, it is more fair to compare them to a distinct set of hyperparameters, not just the optimal set. This is still a far cry from a fair comparison, but it is better than comparing our results with fig. 9 (c) and fig. 9 (d).

Now that we have seen the big lines of the results and compared our reproduction to the results of Jaques et al. [23], the next sections will dig into the individual experiments a bit more.

5.2 Baseline extrinsic reward



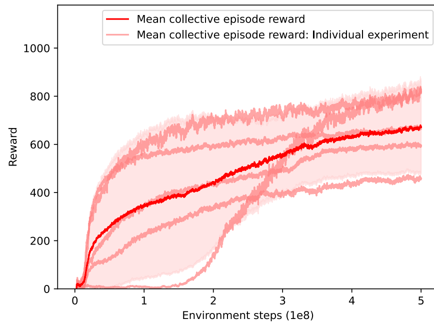
(a) Cleanup baseline



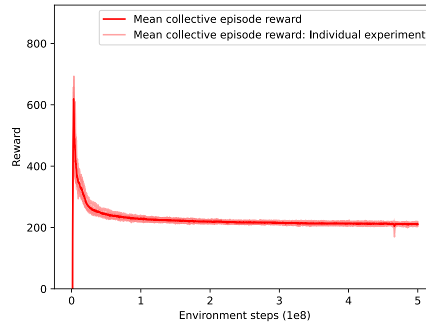
(b) Harvest baseline

In the work of Jaques et al. [23], the baseline model did not learn to perform well in the cleanup environment. In contrast, our experiments show it to work quite well, though still marginally worse than in the other models. A likely potential explanation for this is the use of PPO. It is very unlikely, though possible, that we randomly stumbled on exceedingly good hyperparameters.

5.3 Social influence



(a) Cleanup Social Influence

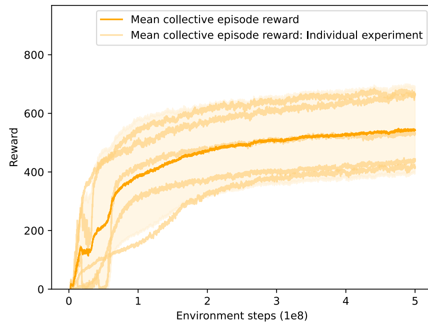


(b) Harvest Social Influence

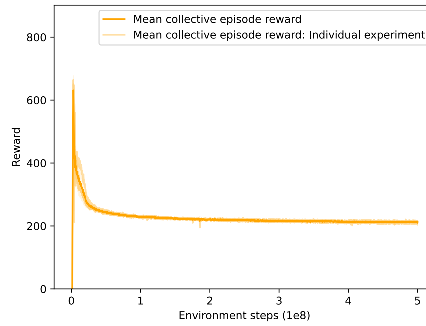
The distribution of extrinsic reward in the cleanup situation are interesting to mention: when looking at the distribution of scores, a less equitable division leads to a lower collective score. This clearly displays the properties of a public goods dilemma: the more cooperators, the higher the collective reward. Additionally, in none of the experiments all agents receive roughly the same amount of reward. In 3/5 experiments there is one agent, and in 2/5 experiments there

are two agents that receive substantially less reward than the other agents. Cooperation is learned, but the distribution of reward is not necessarily fair. This problem has been addressed in the work Hughes et al. [20].

5.4 SCM



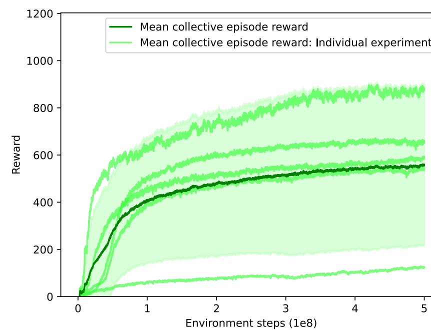
(a) Cleanup SCM



(b) Harvest SCM

We can see that in cleanup, the SCM initially outperforms the MOA but is then overtaken. Given the very wide confidence bands, it is not possible to say whether this is the true mean case. However, it would be in line with our expectations: curiosity should make learning cooperation faster. However, later on it hinders the agent from getting a higher score by adding noise to the reward signal when all it picks up is irreducible uncertainty. To attenuate this, a learning schedule similar to the MOA could be implemented, where the value of the intrinsic reward decreases after a certain amount of environment steps.

5.5 SCM without influence reward



(a) Cleanup SCM no influence

The mean performance on cleanup is similar to that of the SCM with influence reward, although the higher variance make us more uncertain about the mean outcome, causing a far wider confidence band. The reason that this model was not tested on harvest was that previous results were extremely similar, irrespective of model choice. Therefore, we expected no different results on harvest with this model, and thus decided to not run this experiment.

5.6 Statistical test

As mentioned in the method chapter, we will now perform the student's t-test on the final scores of all models, comparing different models within the same environment. The final score per model consists of the mean episode reward over 96 episodes. Our null hypothesis is that the means of the populations from which we drew our samples are equal. Conversely, the alternative hypothesis then is that they differ significantly. Since each model-environment combination has been tested 5 times, which gives us $2 * 5 - 2 = 8$ degrees of freedom. For the confidence level of 95%, the two-sided region critical value is 2.306. The results are shown in table 1.

cleanup	baseline	MOA	SCM	SCM WIR
baseline	-	0.493	0.888	0.451
MOA	0.493	-	1.461	0.802
SCM	0.888	1.461	-	0.111
SCM WIR	0.451	0.802	0.111	-

harvest	baseline	MOA	SCM
baseline	-	0.396	0.808
MOA	0.396	-	0.848
SCM	0.808	0.848	-

Table 1: Independent two-sample t-test results, absolute (non-negative) values. The values represent the t-values for the mean reward scores between two models. The mean reward score for each model is the mean collective extrinsic reward over the last 96 episodes of 5 experiments. To see how this is calculated exactly, see section 4.1. The SCM without influence reward is denoted "SCM WIR".

None of the absolute t-values exceed the critical value of 2.306. Therefore, we find no evidence to reject the null hypothesis in any case with the given degree of certainty. Our results thus give insufficient evidence that the means of the models are significantly different at a confidence level of 95%. This concludes the results. To see how this work fits into the current state of reinforcement learning, we will compare it to other works of research in the next chapter.

6 Related work

In this chapter, we briefly describe the work that has been done in this area, and how our research differs from it. We have already extensively discussed the Social Influence technique by Jaques et al. [23] and the ICM by Pathak et al. [40], so this chapter focuses on more tangentially related work.

Using influence as an intrinsic motivation in a multi-agent setting was also done by Wang et al. [49], which resembles the work by Jaques et al. [23]. They present two methods of influence, the first of which uses mutual information instead of counterfactual actions. Analogous to the Social Influence method, their second method utilizes a counterfactual action-value, which not only marginalizes actions but also the state.

Predicting the influence reward resembles the *UNREAL* model by Jaderberg et al. [21], in which reward prediction is used as an auxiliary task. There are three main differences: firstly, the SCM only predicts the influence reward, whereas the *UNREAL* model predicts the extrinsic reward. Secondly, Jaderberg et al. [21] use this reward prediction purely as an auxiliary task that helps shape the agent's CNN features, it provides no reward to the agent. In contrast, the SCM provides reward, but does not help shape the CNN features, as the policy gradient and reward prediction do not share layers. Lastly, the *UNREAL* model is trained using biased reward sequences to hide the sparsity of rewards, and the SCM is not.

The *DRPIQN* by Hong et al. [19] also models the policies of other agents as an auxiliary task, but does not provide any reward. In that sense it resembles the MOA by Jaques et al. [23] without Social Influence reward. The main differences there are that the *DRPIQN* model not only shares a convolutional layer between modeling the other agent and the agent's own value function - they are connected with a multiplication layer as well. Additionally, this is an action-value model, not a policy gradient model like the MOA and SCM. The way in which *DRPIQN* resembles the SCM, is that policy features (the output of the MOA LSTM) are used to do learn about its own reward. In the case of the work by Hong et al. [19] this concerns the agent's own Q-function, in the case of the SCM it's about intrinsic reward prediction.

This work is not the first to utilize curiosity in a multi-agent setting: Schafer [44] used the Intrinsic Curiosity Module (ICM) by Pathak et al. [40] to encourage exploration in a multi-agent setting. However, the observation function in the work of Schafer [44] provided high-level features, not visual frames. The used environments were also very simple: the games' transition function included very little in the way of complex environment dynamics, so that the only change in observations came from other agents' actions. Finally, they did not pertain to social dilemmas.

7 Conclusion

Here follow a recap of the research question, method, and results, finally wrapping up with potential directions for future research. The research question was the following: How do agents perform when using a combination of curiosity and empowerment in the cleanup and harvest environments? We set out to answer this question by attempting to reproduce previously found results by Jaques et al. [23], then devising our own multi-agent curiosity technique. By combining the Social Influence technique with the Intrinsic Curiosity Module, we created the Social Curiosity Module (SCM). The SCM was found to perform roughly equivalently to the other models, which answers our research question.

The reproduction of previous research was done using the PPO algorithm instead of A3C due to computational constraints, as hyperparameter searches are computationally expensive, and PPO is less sensitive to hyperparameter settings. However, compared to previous work by Jaques et al. [23] who used A3C, PPO turned out to give substantially different results in both the harvest and cleanup environments. Across the different models utilizing PPO, cleanup and harvest show very similar results. Specifically in harvest, the results are almost identical with very narrow confidence bands. With cleanup, the confidence bands are far wider, and the models could very well perform differently, but this cannot be said with any certainty, confirming this would require more experiment runs. In the cleanup environment, we also saw relatively good performance of the PPO baseline model compared to the A3C baseline model in the work of Jaques et al. [23].

7.1 Future work

To expand on this work, a more extensive hyperparameter search could be performed, as this is not something we have done. It could significantly alter the outcome of the experiments, given that RL algorithms can be very sensitive to hyperparameter settings. Performing this hyperparameter search can be done by using the provided code found in the archived repository[15].

In section 2.2.4 we discussed how the Social Influence technique by Jaques et al. [23] is not empowerment in the traditional sense: it measures whether influential actions are exercised, rather than measuring whether influential actions are possible. Implicit in exercising these options is entering states that allow you to exercise them - but exerting influence is not always a good thing. Instead, you can envision a technique in which the estimated social empowerment is given as a reward, not the actual influence that was exerted. This technique could then be a substitute for the social influence technique used in the SCM, either with or without influence reward.

Other auxiliary tasks can be imagined to try curiosity on: the *UNREAL* model by Jaderberg et al. [21] could be a starting point. Conversely, their use of biased reward sequences could be used to train the SCM or the Social Influence model more quickly.

References

[1] Joshua Achiam and Shankar Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

[2] Haurie Alain, Zaccour Georges, et al. *Games and dynamic games*, volume 1. World Scientific Publishing Company, 2012.

[3] Nick Bostrom. Ethical issues in advanced artificial intelligence. *Science fiction and philosophy: from time travel to superintelligence*, pages 277–284, 2003.

[4] Michael Bowling and Manuela Veloso. Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence*, volume 17-1, pages 1021–1026. Lawrence Erlbaum Associates Ltd, 2001.

[5] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

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

[7] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.

[8] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. Back to basics: Benchmarking canonical evolution strategies for playing atari. *arXiv preprint arXiv:1802.08842*, 2018.

[9] Jack Clark and Dario Amodei. Faulty reward functions in the wild. *Internet: https://blog.openai.com/faulty-reward-functions*, 2016.

[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[11] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. How many random seeds? statistical power analysis in deep reinforcement learning experiments. *arXiv preprint arXiv:1806.08295*, 2018.

[12] Ildefons Magrans de Abril and Ryota Kanai. A unified strategy for implementing curiosity and empowerment driven reinforcement learning. *CoRR*, abs/1806.06505, 2018. URL `http://arxiv.org/abs/1806.06505`.

[13] Shane Frederick, George Loewenstein, and Ted O'donoghue. Time discounting and time preference: A critical review. *Journal of economic literature*, 40(2):351–401, 2002.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[15] Hugo Heemskerk, Eugene Vinitsky, and Natasha Jaques. Sequential social dilemma games, 2020. URL `https://doi.org/10.5281/zenodo.4056580`.

[16] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[17] Pablo Hernandez-Leal, Bilal Kartal, Matthew E Taylor, and AI Borealis. Is multiagent deep reinforcement learning the answer or the question? a brief survey. *learning*, 21:22, 2018.

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

[19] Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, and Chun-Yi Lee. A deep policy inference q-network for multi-agent systems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1388–1396. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[20] Edward Hughes, Joel Z. Leibo, Matthew G. Philips, Karl Tuyls, Edgar A. Duéñez-Guzmán, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin R. McKee, Raphael Koster, Heather Roff, and Thore Graepel. Inequity aversion resolves intertemporal social dilemmas. *CoRR*, abs/1803.08884, 2018. URL `http://arxiv.org/abs/1803.08884`.

[21] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[22] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[23] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Çaglar Gülçehre, Pedro A. Ortega, DJ Strouse, Joel Z. Leibo, and Nando de Freitas. Intrinsic social motivation via causal influence in multi-agent RL. *CoRR*, abs/1810.08647, 2018. URL `http://arxiv.org/abs/1810.08647`.

[24] Tobias Jung, Daniel Polani, and Peter Stone. Empowerment for continuous agent—environment systems. *Adaptive Behavior*, 19(1):16–39, 2011.

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

[26] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. Empowerment: A universal agent-centric measure of control. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 128–135. IEEE, 2005.

[27] Peter Kollock. Social dilemmas: The anatomy of cooperation. *Annual review of sociology*, 24(1):183–214, 1998.

[28] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems*, pages 464–473. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

[29] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[30] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[31] George Loewenstein and Drazen Prelec. Negative time preference. *The American Economic Review*, 81(2):347–352, 1991.

[32] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

[33] Michael W Macy and Andreas Flache. Learning dynamics in social dilemmas. *Proceedings of the National Academy of Sciences*, 99(suppl 3):7229–7236, 2002.

[34] Maja J Mataric, Martin Nilsson, and Kristian T Simsarin. Cooperative multi-robot box-pushing. In *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, volume 3, pages 556–561. IEEE, 1995.

[35] Hermann G Matthies. Quantifying uncertainty: modern computational representation of probability and applications. In *Extreme man-made and natural hazards in dynamics of structures*, pages 105–135. Springer, 2007.

[36] 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, 2016.

[37] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 2125–2133, 2015.

[38] Pierre-Yves Oudeyer and Frederic Kaplan. What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*, 1:6, 2009.

[39] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[40] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[41] Julien Perolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *Advances in Neural Information Processing Systems*, pages 3643–3652, 2017.

[42] Christoph Salge, Cornelius Glackin, and Daniel Polani. Empowerment–an introduction. In *Guided Self-Organization: Inception*, pages 67–114. Springer, 2014.

[43] Nikolay Savinov, Anton Raichuk, Raphaël Marinier, Damien Vincent, Marc Pollefeys, Timothy Lillicrap, and Sylvain Gelly. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

[44] Lukas Schafer. Curiosity in multi-agent reinforcement learning. Master's thesis, The University of Edinburgh, 2019.

[45] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[46] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[48] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press, 2011.

[49] Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. *arXiv preprint arXiv:1910.05512*, 2019.

[50] Lilian Weng. Policy gradient algorithms. *Internet: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html*, 2019.

[51] Meredith West. Social play in the domestic cat. *American Zoologist*, 14(1): 427–436, 1974.

[52] Robert W White. Motivation reconsidered: The concept of competence. *Psychological review*, 66(5):297, 1959.

[53] Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems*, pages 5279–5288, 2017.

Figure 10: PPO Baseline hyperparameters

	cleanup	harvest
entropy_coeff	0.00176	0.000687
lr_schedule_steps	[0, 2e+07]	[0, 2e+07]
lr_schedule_weights	[.00126, .000012]	[.00136, .000028]

Figure 11: PPO MOA hyperparameters

	cleanup	harvest
entropy_coeff	0.00176	0.00223
moa_loss_weight	0.06663557	0.091650628
lr_schedule_steps	[0, 2e+07]	[0, 2e+07]
lr_schedule_weights	[0.00126, 0.000012]	[0.0012, 0.000044]
influence_reward_weight	1.0	2.521
influence_reward_schedule_steps	[0, 1e+07, 1e+08, 3e+08]	[0, 1e+07, 1e+08, 3e+08]
influence_reward_schedule_weights	[0.0, 0.0, 1.0, 0.5]	[0.0, 0.0, 1.0, 0.5]

Figure 12: PPO SCM hyperparameters

	cleanup	harvest
entropy_coeff	0.00176	0.00223
moa_loss_weight	0.06663557	0.091650628
lr_schedule_steps	[0, 2e+07]	[0, 2e+07]
lr_schedule_weights	[0.00126, 0.000012]	[0.0012, 0.000044]
influence_reward_weight	1.0	2.521
influence_reward_schedule_steps	[0, 1e+07, 1e+08, 3e+08]	[0, 1e+07, 1e+08, 3e+08]
influence_reward_schedule_weights	[0.0, 0.0, 1.0, 0.5]	[0.0, 0.0, 1.0, 0.5]
scm_loss_weight	1.0	1.0
scm_forward_vs_inverse_loss_weight	0.5	0.5
curiosity_reward_weight	0.001	0.001

Figure 13: Hyperparameter legend

entropy_coeff	PPO entropy reward coefficient. Policy entropy is multiplied by this value, which is subtracted from the PPO loss.
moa_loss_weight	MOA loss scaling parameter λ_{MOA} from section 2.2.4
lr_schedule_steps	Learning rate schedule environment steps between which weights are linearly interpolated.
lr_schedule_weights	Learning rate schedule weights accompanying lr_schedule_steps, between which the lr coefficient is interpolated.
influence_reward_weight	Influence reward weight ρ_I defined in eq. (23) and eq. (40).
influence_reward_schedule_steps	Influence reward schedule environment steps. Functions identically to lr_schedule_steps, but is instead applied to the influence reward.
influence_reward_schedule_weights	Influence reward schedule weights.
scm_loss_weight	SCM loss scaling parameter λ_{SCM} defined in eq. (39)
scm_forward_vs_inverse_loss_weight	Scaling parameter β defined in eq. (39), which determines the relative importance of the SCM forward versus inverse loss.
curiosity_reward_weight	SCM reward coefficient ψ defined in eq. (40).