Robustness and Adaptation via a Generative Model of Policies in Reinforcement Learning

by

Kenneth Derek

B.S. Computer Science and Engineering, Massachusetts Institute of Technology (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2021

Certified by . . . .
Phillip Isola
Assistant Professor
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Robustness and Adaptation via a Generative Model of Policies in Reinforcement Learning

by

Kenneth Derek

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In the natural world, life has found an uncountable number of ways to survive and often thrive. Between and even within species, each individual has a slightly unique way of existing, and this diversity lends robustness to life in general. In this work, we aim to incentivize diversity of agent policies while optimizing for an external reward. To this end, we introduce a generative model of policies which maps a low-dimensional latent space to an agent policy space. In order to learn a broad range of solutions, our generative model uses a diversity regularizer that incentivizes different agent behaviors given the same state. Agents are assigned a specific latent vector through their trajectory, and the generator learns to encode these behavioral preferences in the latent space. Results show that our generator is able to find an array of policies that can express agent individuality through distinct and unique agent policies. Of particular interest, we find that having a diverse policy space allows us to rapidly adapt to unforeseen environmental ablations simply by optimizing generated policies in the low-dimensional latent space. We test this adaptability in an open-ended grid-world, as well as in a competitive, zero-sum, two-player soccer environment.

Thesis Supervisor: Phillip Isola
Title: Assistant Professor

Acknowledgments

I want to give a huge thank-you to my advisor, Professor Isola. Learning how to go about research was no easy task, especially in the midst of a global pandemic. The work in this thesis was only possible thanks to his availability, enthusiasm, and illuminating discussions.

I would like to thank my friends, both near and far, for reminding me that not everything is about research. I look forward to once again having time to picnic in the park, and play chess - once this is submitted!

Finally, I could not include the writing in this thesis on evolution and genes without thanking my family, who have made me into who I am today. I think all those PBS nature videos we watched together found their way into my code.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Quick thought experiment: imagine our world was such that all people acted, thought, and looked *exactly* the same in *every* situation. Would we ever have found the influential dissenters that sparked scientific and political revolutions that now define our humanity - daring to fight for civil rights, venture into space, and far more?

In reinforcement learning (RL), it is common to learn a single policy that fits an environment. However, it is often desirable to instead find an entire array of high performing policies. To this end, we propose learning a generative model of policies. At a high level, we aim to show that purposefully learning a diverse policy space for a given environment can be competitive to learning a single policy, while better simulating the complexity and diversity of life in the real world.

Previous works have touched on ideas akin to a generative model of policies. Under the hierarchical RL framework, the the high-level policy controller could itself be considered a policy generator of sub-policies that are 'options' (Sutton et al. [1999], Eysenbach et al. [2018], Florensa et al. [2017]). But these methods are designed with the intent to find decomposable skills that aid in the construction of just one downstream controller policy, and ultimately could even be integrated into our framework to instead learn generative model of 'high-level' controllers. A body of prior work that aligns more closely with our goals is that of Quality Diversity (Pugh et al. [2016]), which optimizes a population of agents along the axes of both reward and diversity. These methods often use evolutionary search and a require a discrete sized population of separate agents with their own policy weights, which consumes more time and training resources, and limits the number of diverse behaviors. Our work integrates the goals of quality diversity into a time and memory efficient deep RL framework by simulating an entire population of agents via a generative model of policies, with diversity bounded only by capacity of the generator.

Why should we bother with finding more than one policy per environment? There are two primary reasons, elaborated in the following paragraphs.

RL environments are continually approaching greater levels of open-endedness and complexity. For a given environment, there might be an entire manifold of valid and near-equally high performing strategies. Each environment might have several or hundreds of degrees-of-freedom that allow different choices of varying scale. For example, many role-playing games have entirely different stories and outcomes depending on player actions - impacted by decisions as grandiose as whether the player is aligned along the faction of good or evil, or as menial as whether the player stopped and talked to a seemingly inconsequential non-playing character. These forks in the road define an entire set of enjoyable play-styles. So then, learning how to play an environment should not stop at just one possible solution. We want to find policies that lie across the entire manifold of high performing strategies defined by the environment. By learning an entire manifold of policies, we increase the robustness and adaptability of our entire population of learned agents. Using our generative model, we are able to adapt our population to select individuals that can still survive given the ablation, much like natural selection drives evolution in the real world. To this end, we introduce a simple optimization algorithm that can quickly adapt to ablated environments, unseen adversaries, and even novel reward functions without forgetting prior solutions, as would be a problem when using transfer learning.

Secondly, using a generative model of policies as a population of agents makes sense in multi-agent environments, in which different agents should in many cases act like they are actual unique individuals. However, it is common in many multi-agent reinforcement learning settings to deploy the same policy across all agents, such that they are essentially distributed clones. Doing so can reduce the multi-modality of the agent population, resulting in essentially a single 'average' agent, which we will show in Chapter 4. Along a similar thread, we believe using a single policy distributed across agents reduces the complexity of behaviors and interactions to be found in multi-agent environments, which is a driving factor behind emergent cooperation (Bansal et al. [2017]) and learning curriculums (Baker et al. [2019]) that make multi-agent environments so interesting, and which we explore in Chapter 5. The current alternative is to use separate weight policies for each agent, which comes with scalability issues in the number of agents active in the environment, both in terms of memory and sample efficiency. Of greater concern; however, is that using k distinct policies to simulate a multi-agent environment bounds the number of unique agent 'personalities' or 'behavior profiles' to k. Instead, by using a generative model, we are able to find unbounded number of diverse solutions, unlocking the potential of interpolate between solutions, diversify the population as much as is possible, and potentially improve multi-agent auto-curricula.

The path of this thesis is as follows. First, we introduce the generative modelling framework upon which we model a population of agents that learn to solve a particular environment. We then show that our generative model has the capacity to form actual distinct 'species' of agents, without requiring separate policy weights per agent. We then go on to explore the potentials of learning a population of agents by ablating environments and optimizing generated policies to fit each ablation, directly in the latent space. We study two main environments: Markov Soccer (Littman [1994]), and Farmworld, a new environment developed during the course of this thesis which supports multi-agent learning in a open-ended gridworld environment.

Chapter 2

Methods

2.1 Diversity Objective

Our method is simple: all policies in Π share the same weight parameters, but are strongly conditioned on latent variable $z \in Z$. We condition each policy $\pi \in \Pi$ via a latent variable via two methods: a diversity regularizer, and model architecture.

One might think that we could train our generative model like a generative model of images or text; however, unlike methods in those fields, we do not have any training data. So instead of learning a generator that can match a training data distribution, we train it to make diverse but high quality behaviors, defined by the environmental reward function.

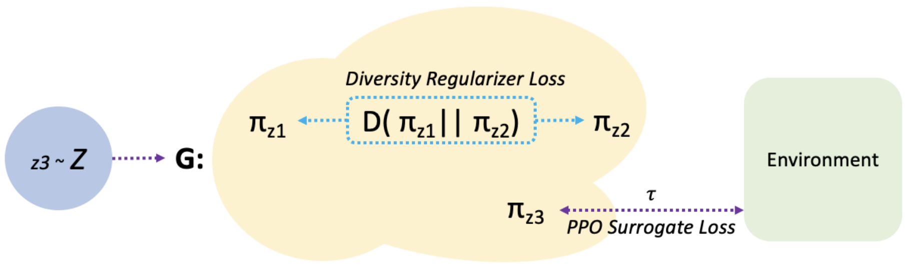


Figure 2-1: Method Diagram
Upon agent initialization, we sample a latent $z \in \mathcal{Z}$. In this case vector $G : z_3 \rightarrow \pi_{z_3}$ which is used to roll out a trajectory in the environment. Diversity regularizer loss is defined over the expected pair (z_i, z_j) and is thus optimized over all possible policies – not just the current policy used on the environment.

Diversity Regularization We introduce a diversity regularization objective as the primary methodology to condition each policy $\pi_{\theta,z}$ on z. In general, we seek to optimize the following objective shown in (2.1), alongside a standard RL reward maximization objective such as in PPO.

$$\max_{\theta} \mathop{\mathbb{E}}_{s \in S} \left[\mathop{\mathbb{E}}_{\substack{z_i, z_j \in Z \\ z_i \neq z_j}} \alpha D(\pi_{z_i,\theta}(s), \pi_{z_j,\theta}(s)) \right] \tag{2.1}$$

Since policies define a space of actions taken over different states, we propose that in order for two policies to be distinct, they must have different action distributions given the same state. In (2.1), D defines some distance between the action distributions of two policies π_{θ,z_i} and π_{θ,z_j}, and α is a scaling constant for the diversity regularizer. While we introduce (2.1) as a general form diversity regularization objective, in our experiments, we particularly optimize

$$\min_{\theta} \mathop{\mathbb{E}}_{s \in S} \left[\mathop{\mathbb{E}}_{\substack{z_i, z_j \in Z \\ c_i \neq z_j}} \alpha \exp\left(-D_{KL}(\pi_{z_i,\theta;b}(s) \| \pi_{z_j,\theta;b}(s)) \right) \right] \tag{2.2}$$

in which D_{KL} is the KL-divergence between the two policy action distributions, and b is a smoothing constant over the action distributions. We use the constant b because in continuous settings and categorical settings, the KL-divergence can quickly approach infinity as probability mass over a particular action approaches zero.

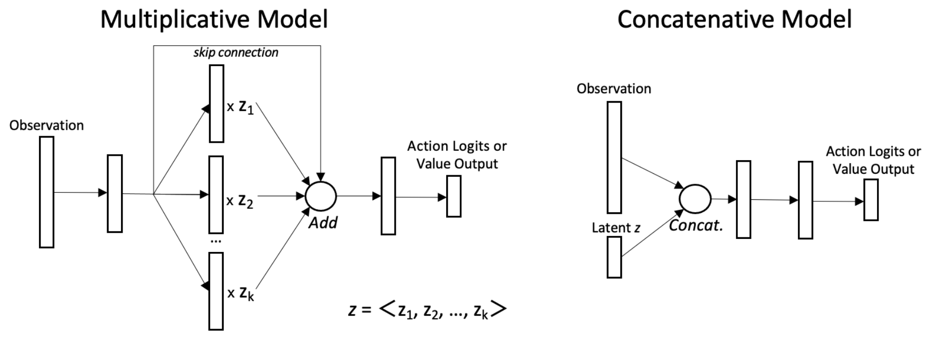


Figure 2-2: Model Architectures for Latent Integration

Model Architecture In our experiments, we have found that richer integrations between the latent vector and the context can yield a policy space that has more multi-modal solutions. Such models are known in literature as Stochastic Neural Networks, but are not commonly used in RL. We call our baseline model the concatenation model, which concatenates the latent vector z with the environment observation. We additionally introduce a multiplicative model, which is inspired by the attention network architecture. Using a latent vector of dimension k, our multiplicative model is able to learn k interpretations of the observation, which are each modulated by a dimension of the latent vector. Using a skip connection allows the model to learn policies faster than without.

Optimization of G In our experiments, we optimize the diversity objective in an on-line fashion using gradient descent, in conjunction with a PPO clipped-surrogate objective and entropy regularization objective. We provide pseudocode for in Algorithm 2.

Adaptation via Optimization in the Latent Space of G By learning an entire space of policies Π, we are able to search our policy space for the highest performing policy on any given environment. In contrast to searching over policy parameters, we are able to quickly search over the low-dimensional latent space (dimension 3 in our experiments). We provide psuedocode for our optimization process in Algorithm 1. In general, we are able to optimize for new conditions in the space of just 100 generations, which takes under a minute and requires no gradient updates.

Chapter 3

Related Work

3.1 Option Discovery

Sutton et al. [1999] introduced the option framework, which provides a temporal abstraction of actions. Under the options framework, a high-level controller policy chooses a latent vector z, which encodes a sequence of actions to be executed by a sub-policy. Frequently, the high level controller is sampled for an action every k steps (Zhang et al. [2021], Haarnoja et al. [2018]), which aids in hard exploration problems by reducing the number of actions required to find a solution. Generally, the sub-policy is a single policy network conditioned on z, and can itself be thought of as a generative model. Nevertheless, works in this field have several distinctions from our work. We detail three main differences.

First, the options framework is intended to find an abstracted set of 'options,' or skills that maximize the the ability of a higher level policy. These skills are not intended to be stand-alone policies. In some situations, such as 2D gridworld navigation, it is sufficient to sample just one option for the course of the episode Eysenbach et al. [2018]. However, this is not always the case. For example policy sub-controllers may learn skills such as 'jumping' or 'crawling,' which on their own are not policies that can solve a more complex task, like 'walk over to the object and pick it up.' In order to get such a solution, one would need to compose a sequence of the learned options. On the other hand, in our framework, the pair (G, z): generative model and a corresponding sampled latent vector, is intended to fully define a complete policy for an environment. Generated policies are complete, in the sense that they should fully define a behavior profile that finds high reward in the train environment.

Secondly, our work introduces a novel diversity regularizer in comparison to the objectives used in many options frameworks. In options frameworks, the general optimization problem is a mutual information maximization problem:

$$\max_{\pi,\phi} \mathop{\mathbb{E}}_{z\sim p(z)}\left[\mathop{\mathbb{E}}_{\tau\sim\pi,z}\left[\sum_{t=0}^{T}(\log q_\phi(z|\xi) - \log p(z))\right] + \beta\mathcal{H}(\pi|z)\right] \tag{3.1}$$

where, ξ is an arbitrary aspect of the agent's existence in the environment. Most often, ξ is grounded in the state of the environment (Eysenbach et al. [2018], Achiam et al. [2018]), so that the agent must visit unique states in order to provide information about the latent z. The inner expectation $\log q_\phi(z|\xi) - \log p(z)$ corresponds to an intrinsic reward that is proportional to the discriminability of z given ξ. $\beta\mathcal{H}(\pi|z)$ is the term that maximizes the entropy of the latent conditioned policy, so that learned options are diverse as possible. π is conditioned on z by the manner of providing z as an additional input alongside the state s_t.

Grounding ξ entirely in the state makes assumptions that difference in state is a good metric for diversity. In cases such as 2D navigation, this certainly may be true - since the state varies greatly over the course of an agent's trajectory. Howeever, such an assumption may not apply to environments that have very similar states across a trajectory, as we will see in our experiments. To this end, our novel diversity regularizer discards the use of mutual information maximization, and aims to incentivize action diversity given state. Recent works (Wang et al. [2020], Zhang et al. [2021]), attempt a similar technique by including not just state information in ξ, but also policy action logits, squashed into a sigmoid. This allows the mutual information term to be optimized directly by gradient descent, rather than through reinforcement learning as in Eysenbach et al. [2018].

The third main difference is that most option discovery methods are often learned in an unsupervised manner without regard to actual test-time extrinsic environmental reward Eysenbach et al. [2018]. If reward is used, then it is oftentimes a simple surrogate extrinsic reward designed to elicit behavioral abstractions (Haarnoja et al. [2018], Florensa et al. [2017]). For example, in a 2D point-mass navigation task, one might learn options using an extrinsic reward such as the speed of the point mass. This surrogate reward is simple enough to allow the options method to discover a large number of distinct options, such as directions to move. However, when trained with extrinsic rewards that are more difficult to satisfy, learning a distinct set of options using the mutual information maximization objective becomes a challenging problem of balancing intrinsic option reward and extrinsic environmental reward. On the other hand, our diversity regularizer does not deal with mutual information, and has been easier to optimize alongside test-time extrinsic reward.

Despite the differences between these methods, they are not actually disjoint. One could imagine potentially combining the option framework with our policy generator, to get an array of high-performing yet diverse hierarchical 'controllers.' To do so, one could easily modify our diversity regularizer to measure behavioral difference across potential options rather than environment actions.

3.2 Stochastic Neural Networks

In our work, we find that using a class of models called Stochastic Neural Networks aids in allowing our generator to map latent vectors to highly multi-modal and individualistic agent policies. Stochastic Neural Networks have been used in (Florensa et al. [2017], Fukui et al. [2016], Wu et al. [2016]) to improve optimization and yield more complex interactions between the latent vector and latent-conditioned policies. Although we employ a slight different architecture than prior works, we also find that richer integrations of the latent than concatenation inherently offer some degree of multi-modality, and additionally help optimize directly for diverse modes of behavior.

3.3 Quality Diversity

A field known as Quality Diversity has very similar goals as our work. In general, works in this field aim to find a large number of high performing policies, and find that doing so can avoid pitfalls of objective based optimization. Novelty Search (Lehman and Stanley [2011b]) and MapElites (Mouret and Clune [2015]) use methods inspired from evolution, such as mutation and recombination of the parameter space, to find high performing solutions in an environment. MapElites aims to find a large array of high performing solutions to an environment by partitioning the genotype space into buckets, and having solutions compete only within their respective bucket. Mutations will cause genotypes to slowly spread out and fill many buckets. Novelty Search directly optimizes for diversity of behavior, and selects individuals based on how differently they are in behavior space from an archive of past individuals. A modification, Novelty search with Local Competition (Lehman and Stanley [2011a]), only optimizes for behavioral diversity against other agents that have similar morphologies. However, work has shown that NS and NL-LC is very sensitive to the choice of behavior space, and sometimes requires the very domain knowledge that it seeks to avoid. Overall, these works indicate that incorporating diversity into reinforcement learning can have promising results in finding diverse, robust, and high-performing solutions.

3.4 Learning with a Population of Policies

There are several prior works that aim to connect ideas of Quality Diversity with deep reinforcement learning. These methods attempt to optimize a fixed-size population or archive of policies to be distinct from each other. Hong et al. [2018] augments PPO with an additional objective that enforces that the current version of a policy is different from a fixed size archive of past versions of itself, measured by the distance of the policy action spaces. However, this method ultimately only produces a single policy, and is primarily used as a way to avoid local minimum during optimization.

Since comparisons are made against past versions of the same policy, diversity in the archive is inherently constrained. DIPG (Masood and Doshi-Velez [2019]) instead learns a family of separate-weight policies that are each distinct from one another, using a distance metric to push policies apart from one another. Similarly, Parker-Holder et al. [2020] also optimizes for a diverse fixed-size family of separate-weight policies. They define a behavioral embedding as the expected behavior across states, and optimize for the volume of the region contained by all behavior embedding points of the family of policies. Ultimately, these methods constrain the amount of possible diversity by learning only a fixed size population of diverse policies. Instead, by using a generative model of policies to represent our policy population, we have the potential to learn unbounded number of separate and interesting policies.

Chapter 4

Learning an Effective Generator

One of the motivations behind learning a policy generator is that we want to represent a multitude of diverse agent policies while only using the weights for a single environment. Using shared weights helps with computational costs in training large multi-agent populations, and also helps accelerate the learning process by using data collected across all agents to update their shared weights. However, using shared weights also comes with its potential pitfalls. Naively using shared weights across agents may result in all agents in an environment acting like clones, which is precisely what we wish to avoid. We need to ensure that our policy generator is able to learn a highly multi-modal space of policies.

What is a multi-modal space of policies? At its most extreme, our generator should be able to map a two latent vectors to two policies that act *completely differently* in *every state* they encounter.

In this chapter, we introduce three experiments that aided in testing the ability of our generator G to model a multi-modal space of policies. Our experiments are as follows:

- FunctionWorld: a multi-regression task, in which we attempt to regress three different functions using the same regression model, using RL.
- Niche Specialization: a Farmworld experiment in which to attain high reward, agents must specialize into two distinct niches that have disjoint behavioral

requirements.

- Agent Traits: an experiment in which agents are given different abilities, or *traits*, and must act according to their traits if they want to survive in the environment.

4.1 How Model Architecture and Context Distribution affects Multi-Modality

We train a policy π to maximize 1-step expected rewards given observation of the domain $x \in [0,1]$ and target $f(x) \in [0,1]$. The output of π is μ and σ^2, from which we generate action $a \sim \mathcal{N}(\mu, \sigma^2)$. We use the REINFORCE algorithm, and compute advantage by normalizing the reward across a batch of samples (s, a, r), where s is the (x, z) input pair and $a =$ the y prediction. Crucially, we normalize rewards with across a batch collected using the same latent z, which essentially treats each latent vector as a unique 'bucket,' similarly to how MapElites (Mouret and Clune [2015]) only measures competition with individuals that are in the same bucket. Doing so is important, since otherwise we would hinder specialization and diversity by measuring the reward of a hypothetical latent vector z_i that is currently attempting to diversify, against the reward of a currently high-performing hypothetical latent vector z_j. This would penalize z_i's actions, when in reality, we want to encourage the diversification. In later experiments, we discard this sort of reward normalization in favor of using an actual value function that is able to account for these expected reward differences, conditional on the particular latent z.

Reward is calculated as

$$\min_{f \in \mathcal{F}} \; -(a - f(x))^2$$

where $\mathcal{F}$ is a family of functions in the multi-function environment. We use

$$\mathcal{F} = \{f_1(x), f_2(x), f_3(x)\}$$

where

$$f_1(x) = x^2 + 0.5$$
$$f_1(x) = -x$$
$$f_3(x) = x - 0.75$$

In this experiment we explore the influence of model architecture, and latent distribution, and use of our diversity regularizer on the multi-modality of the policy generator.

Model Architectures

- Concatenation Model as seen in 2-2. A latent context is concatenated to the flattened environmental observation.
- Multiplicative Model as seen in 2-2. The latent context is multiplicatively integrated with a learned embeddings of the environmental observation.

Context Distributions of size 3

- Uniform 3-Categorical (k-dim one-hot)
- $\mathbb{R}^3 \sim$ Unif[0, 1]
- $\mathbb{R}^3 \sim$ Unit Sphere, centered at 0, 0. Thus the magnitude of the latent vector $|z| = 1$.

Results As seen in 4.1, both model architecture and latent distribution are important in learning a multi-modal policy space.

The first thing to note is that using our multiplicative model, a type of Stochastic Neural Network, has a significant impact on the ability of our generator to learn a multi-modal space of policies. Simply using the concatenation model does not provide the generator enough representational flexibility to fit to all three functions in $\mathcal{F}$.

Our second observation is that using continuous latent vectors hinders how well any one latent can fit to a particular function. However, we believe using continuous

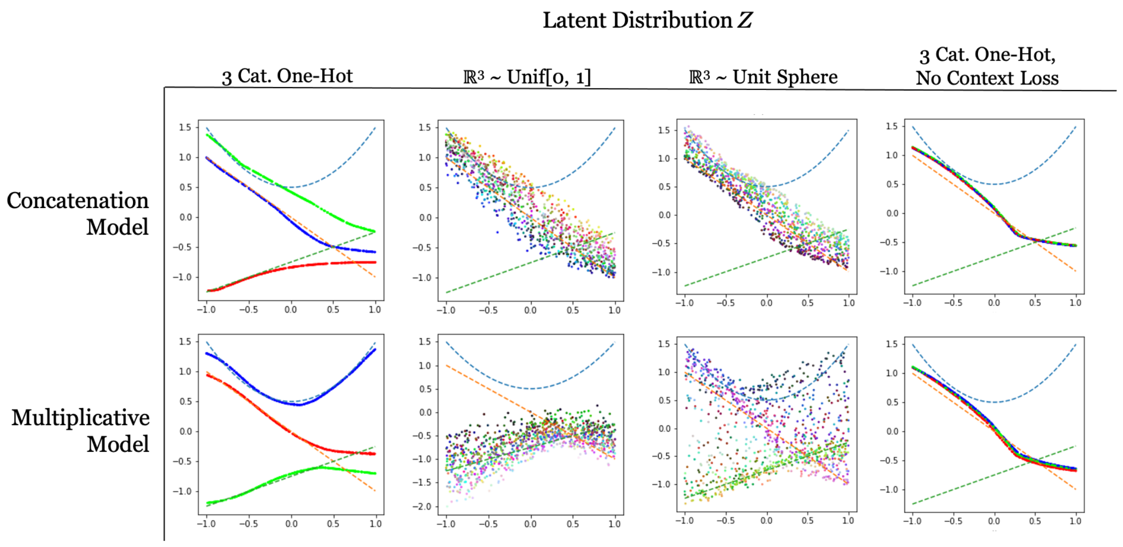


Figure 4-1: Using a Multi-Modal Policy to Model Three Different Functions
Target functions are shown in dotted lines. Policy predictions are shown in solid colored lines, where each (x, y) point is visualized in the RGB color directly proportional to it's latent vector. Importantly, we observe that using a concatenation model does not provide the generator enough representational flexibility to fit to all three functions in $\mathcal{F}$.

latent space still provides major benefits over a discrete latent space. First and foremost, a discrete latent space will only be able to find a limited set of distinct agent policies. On the other hand, a continuous latent space of policies offers the enticing possibility of finding as many policies as the generator G can fit. Additionally, a continuous latent space offers the ability to interpolation and optimization policies in the latent space.

In all following experiments, unless otherwise specified, we use continuous contexts sampled from the 3-dimensional unit sphere.

4.2 Specialization of Policies in Π

Environment We test our generative model of policies in a new open-ended grid-world environment called Farmworld, that supports multi-agent interaction and partially observable observations. The idea behind Farmworld is simple: agents move about on the map to collect various resources. Agents get 0.1 reward per time-step alive, and they can gain health by mining or farming resources from the environment. Their are two types of resources: towers and chickens. Agents can interact directly by attacking each other, or indirectly by competing for shared and limited resources. An agent observation is composed of a flattened representation of the units in a configurable L1 radius centered at its location. In all of our Farmworld experiments, we set this radius to be 2 tiles around the agent.

Niche Specialization Experiment To test the multi-modality of our generator G, we set up the Farmworld environment using a hidden rule: when an agent spawns, it is able harness resources from either towers or chickens. However, once it gets health from one unit type, it becomes 'locked-into' that unit type, and cannot gain health from other unit types. Additionally, information about whether an agent is 'locked-into' a unit, and if so, which unit it is locked into, is not provided as part of the agent observation. For this experiment, we place 8 each of agents, towers, and chickens on a 10×10 grid.

Baselines In this experiment, we test our Diversity Regularization method against a variety of algorithm baselines, which are elaborated in Appendix B.

- Vanilla PPO
- DIAYN
- DIAYN + action: Same as DIAYN, but uses action information alongside with state to train the mutual information discriminator.
- Differentiable Mutual Information (MI): a method used in Zhang et al. [2021], that maximizes the mutual information between state-action-logits pair and the latent z in a differentiable manner.
- Differentiable MI + Noise: Same as Differentiable MI, but with uniform noise applied to the action logits when optimizing the MI discriminator.

For each baseline, we evaluate its performance with Concatenation and Multiplicative model architectures. Additionally, we test Vanilla PPO using a recurrent model, since it could be helpful to learn and remember the initial unit an agent attacked.

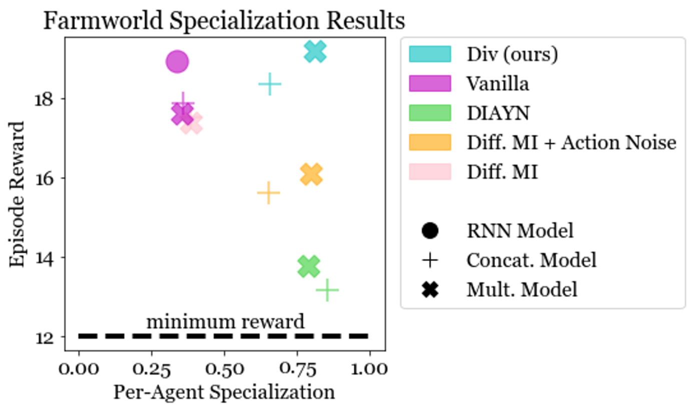


Figure 4-2: Specialization versus Reward in the Niche Specialization Experiment
Per-Agent Specialization is measure by the 1 minus the average entropy between chicken interactions and tower interactions per agent.

Results Figure 4-2 details the outcome of our Niche Specialization experiment. Additionally, we visualize samples of episode trajectories in C-3 and a table of results broken down even further by method in table 4.2.

The thing to take away from Figure 4-2 is that our Diversity method (Div) is able to attain the highest reward in the environment, while maintaining high per-agent specialization. Specialization is measure by the 1 minus the average entropy between chicken interactions and tower interactions per agent. Thus, an agent that only attacks chickens during the course of its trajectory will have a specialization of 1, while a agent that attacks both units equally will have an specialization of zero. Our Diversity method is able to achieve a significantly higher reward than any other method using a purely reactive policy (multiplicative or concatenation), while maintaining having specialization result. The RNN model also has high reward, likely due to better map exploration, but does not learn to specialize between towers and chickens.

This means that our generator G has learned a mapping from $Z \to \Pi$ that partitions agents into one of two highly distinct roles: chicken farmers and tower farmers.

G has actually learned a space of policies, conditional on the latent vector. On the other hand, without our diversity regularizer, Vanilla PPO with Mult and RNN models are not able to learn as multi-modal and diverse a space of policies. As such, they tend to have much higher chicken-tower interaction entropy, and agents will waste time attacking a unit from which they will gain no health.

Furthermore, option discovery baselines also do not learn to balance specialization and reward in an effective manner. Figure C-3 shows that DIAYN generally ignores extrinsic reward, and instead opts to find states that provide information about the agent latent vector z. Differentiable MI also suffers from optimization problems in this task, and we see a tradeoff between specialization and reward.

Model Setting	Mean Reward	C. Hits	T. Hits	C-T Entropy
DIAYN Concat	13.2 ± 0.3	1.0 ± 0.2	0.9 ± 0.2	0.15 ± 0.0
DIAYN Mult	13.8 ± 0.9	1.2 ± 0.4	1.3 ± 0.7	0.21 ± 0.1
Diff. MI Concat	17.5 ± 0.3	3.1 ± 0.1	4.4 ± 0.3	0.61 ± 0.0
Diff. MI Concat + Noise	15.6 ± 0.0	2.1 ± 0.3	3.0 ± 0.2	0.35 ± 0.0
Diff. MI Mult	17.4 ± 0.2	3.1 ± 0.1	4.2 ± 0.3	0.61 ± 0.0
Diff. MI Mult + Noise	16.1 ± 0.0	2.0 ± 0.0	2.6 ± 0.0	0.20 ± 0.0
Div Concat Ctx 0.1	18.4 ± 0.9	2.7 ± 0.3	3.8 ± 0.8	0.34 ± 0.2
Div Mult Ctx 0.1	$\mathbf{19.2} \pm 0.6$	3.1 ± 0.2	3.2 ± 0.0	0.19 ± 0.1
Vanilla Concat	17.9 ± 0.0	3.7 ± 1.4	4.4 ± 0.1	0.64 ± 0.1
Vanilla Mult	17.6 ± 0.0	4.0 ± 1.6	4.3 ± 0.1	0.65 ± 0.1
Vanilla RNN	18.9 ± 0.0	3.0 ± 0.3	4.5 ± 0.2	0.66 ± 0.0

Table 4.1: Specialization and Reward by Model and Algorithm
Notice that while our **Div Mult** model attains the highest reward, it actually has fewer interactions with chickens and towers (C. Hits and T.Hits) than the Vanilla methods. This is because the Vanilla methods tend to waste agent steps attacking towers and chickens, even though they are not able to do damage on them.

4.3 Harnessing Agent Individuality

In the prior experiment, we determined that our generator is able to adapt to a multi-modal policy space as required by our environmental rules. It managed to encode information into the initially meaningless latent space, partitioning agents into two major niches: chicken attackers and tower attackers.

But what if the latent space actually has semantic meaning? In this experiment, we measure if our generative model is able to build a policy space according to implicit semantic meaning. We motivate this experiment by an observation that in nature, species are granted their genes *a priori*, and must act in accordance with how their genes 'program' them. A lion cannot attempt to live like a herbivore, because its biology would not let it. Along this line, we design our experiment such that the latent vector dictates how an agent should act in the Farmworld environment via modulating how much damange it is able to do on each unit type in Farmworld.

Environment We again use the Farmworld environment. We place 10 each of agents, towers, and chickens on a 12×12 grid, with locations randomly initialized each episode.

Agent Trait Experiment As in the prior experiment, we augment the standard Farmworld environment with an additional rule. In this experiment, agents are assigned a latent vector z that is directly proportional to that agent's damage profile. We call the the damage profile a *trait* (each dimension of the latent z could be an allele, of sorts).

This set-up draws inspiration from character creation video games such as MMORPGs, in which agents have a limited set of points to assign to various statistics that make up a given agent's profile. In our experiment, 5 points are distributed randomly amoungst three measures of agent damage: agent vs. chicken damage, agent vs. tower damage, and agent vs. agent damage. For example, an agent with damage profile $< 3, 1.2, 0.8 >$ is able to do 3 units per hit against chickens, but only 1.2 and 0.8 units per hit against towers and other agents respectively. We experiment with two settings:

- *soft traits:* action statistics are sampled from the space of continuous 3-dimensional vectors that sum to 5.
- *hard traits:* agent statistics are randomly sampled from the set $\{< 5, 0, 0 >, < 0, 5, 0 >, < 0, 0, 5 >\}$

In order to generate the a latent vector z given a damage profile, we subtract 2.5 from each dimension of the profile, and then project the profile onto the 3-dimensional unit sphere.

Additionally, we scale up chicken, tower, and agent healths, such that it would be disadvantageous for an agent with a below-average tower damage to attempt to attack towers, for example. Specifically, we scale tower health to 10, chicken health to 6, and agent health to 30. Since even with maximum agent damage, it would take 6 hits for one agent to defeat another agent, we add a 5x multiplier to agent damage when computing how much damage an agent does on another agent. Now, an agent with a maxed-out statistic in one particular unit type will only take at most 2 hits to defeat that unit.

Baselines We test our Diversity Regularization method against Vanilla PPO, using both multiplicative and concatenation models. We do not use the baselines tested in the prior experiment, since they exhibited degenerate performance.

Results Looking at the reward in 4-3 was initially quite surprising. In our soft traits experiment, the Vanilla PPO concatenation model obtained significantly higher rewards than our generative policy model!

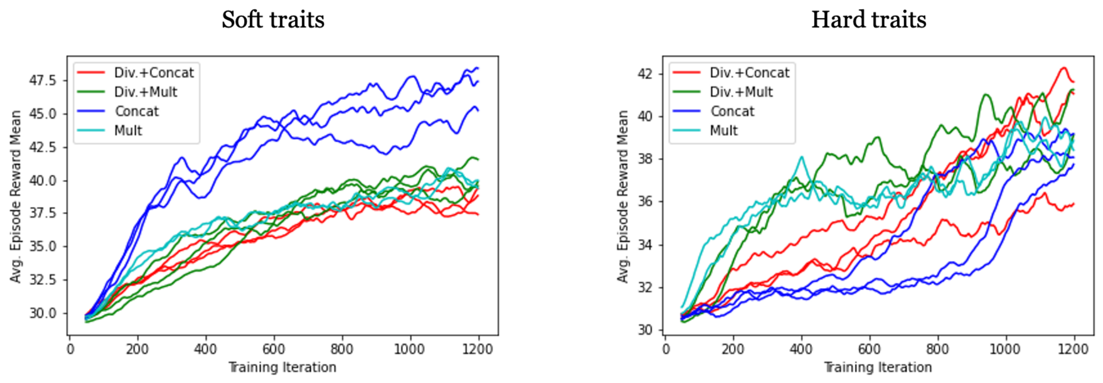
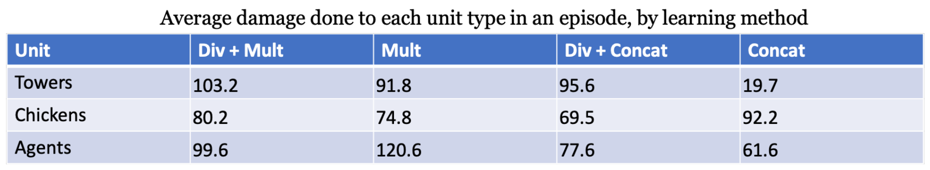


Figure 4-3: Reward in Agent Trait Experiment
Reward over three random seeds, by each method.

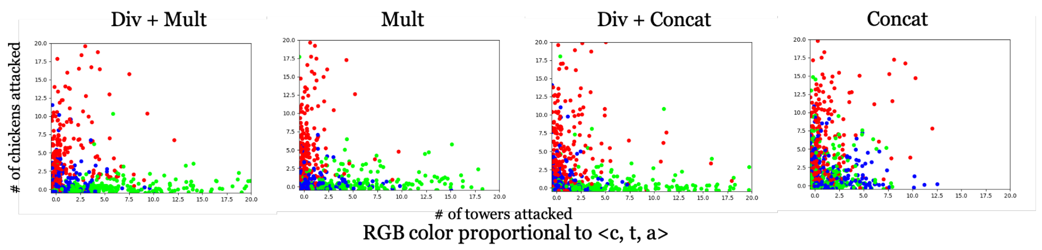
In order to understand what was going on, we calculated the average damage done to each unit type in any given episode. Since agent damage profiles are sampled uniformly, one would expect that in aggregate, agents do approximately equal damage to each of the three unit types in Farmworld during a complete episode. If, on the other hand, one unit type has an inordinately small amount of damage, then we know that agent policies are essentially ignoring that unit type, even though some agents exist that *should* primarily be attacking that unit type.

Hard Traits

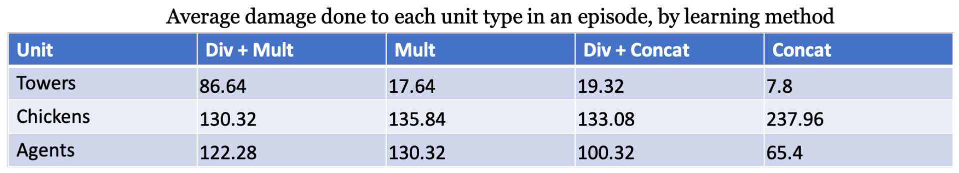


Average damage done to each unit type in an episode, by learning method

Unit	Div + Mult	Mult	Div + Concat	Concat
Towers	103.2	91.8	95.6	19.7
Chickens	80.2	74.8	69.5	92.2
Agents	99.6	120.6	77.6	61.6



Soft Traits



Average damage done to each unit type in an episode, by learning method

Unit	Div + Mult	Mult	Div + Concat	Concat
Towers	86.64	17.64	19.32	7.8
Chickens	130.32	135.84	133.08	237.96
Agents	122.28	130.32	100.32	65.4

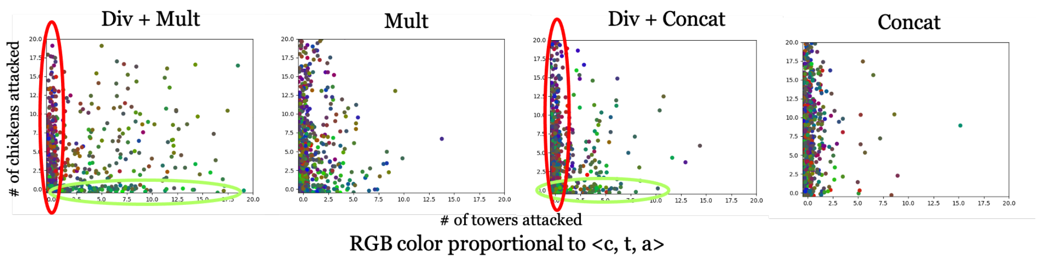


Figure 4-4: Agent Interactions Given Trait
Agent traits are composed of three damage statistics - Chicken, Tower, Agent - that are used as the R, G, and B values respectively of each scatter plot dot. **Left:** it is clear that all phenotypes simply focus on gaining health through chickens. **Right:** there are potentially 3 modes of agent interaction, red chicken-specialized agents on the y-axis focus on chickens, green tower-specialized agents on the x-axis focus on towers, and agents with an average of the skills tend to interpolate their behavior.

Sure enough, it turned out that the concatenation model nearly completely ignored the tower unit type in the soft trait experiment, as well as had significantly less damage against other agents as well. In other words, agents with very weak chicken damage *still attacked only chickens.* On the other hand, our generative policy model was able to learn to partition the policy space into at least three niches of multi-modality, along the axes of the damage profile.

So the Concatenation model with Vanilla PPO was able to attain high rewards in the soft trait environment, largely because agents attacked each other on average less than the other methods. These deficiencies are even more apparent in the hard traits experiment, in which we can observe that agents generally completely ignore towers, even though $\frac{1}{3}$ of agents can only gain health from towers.

Chapter 5

Adaptation and Robustness

In nature, differences between species and even within species lend robustness to life as a whole. It becomes less likely that any single perturbation in the environment will break the overall system. In the same manner, having a space of policies that exist in an environment that each behave differently lends robustness to the policy space as a whole.

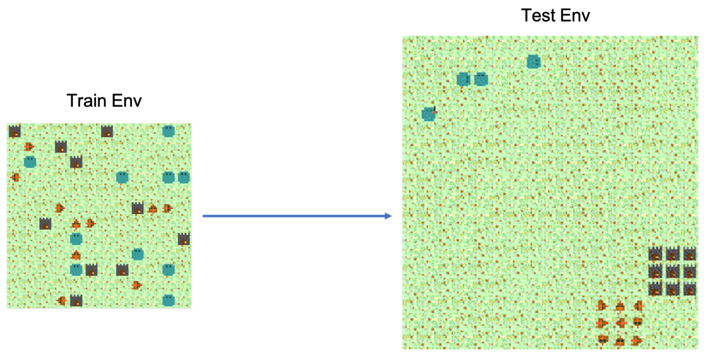


Figure 5-1: Adapting from the Train Environment to the Test Environment

Experiment and Baselines In this experiment, we use a Farmworld environment seen in Figure 5-1, containing eight of agents, towers, and chickens each, on a 10x10 grid. We do not enforce any specialization or grant agents with unique traits, as the point is to see what degrees of diversity might be naturally learned by G.

We test how a diverse policy space allows us to adapt agent policies to better fit an ablated Farmworld environment. As seen in Figure 5-1, we train our policy generator G on the train environment and then evaluate the learned generator on an ablated test environment. We attempt to search the latent space of G for a subset of latent vectors that are able to thrive in the new test environment.

Our search procedure is motivated by the idea of reproduction and mutation in evolutionary search. Over the course of a small number of generations, we evaluate random contexts and keep higher performing ones with greater probability. More details on our evolutionary search are included in Algorith 1. Importantly, the search procedure takes a mere quantity of seconds, and by conditioning G via a latent subspace rather than modifying the weights towards the ablated environments, we avoid problems of the network forgetting its original solutions.

We use the same baselines as in the Niche Specialization environment, but exclude using an model RNN, as our main focus is learning a diverse space of policies via our diversity regularizer. We run each method across three random seeds and report the mean and standard deviation for each result.

Ablations Each ablation targets a potential degree-of-freedom present in the train environment. Particularly, we measure ablations that require a diverse policy space that covers locomotion preference, timing preferences, and unit-target preferences. We describe each ablation in more detail below.

- *Far Corner:* Food has migrated to a new source! The map is expanded to size 18x18. Four agents spawn in the top left of the map, and food is entirely concentrated in the bottom right of the map, far away from the spawn points. Agents have 60 max health in order to reach the food at the bottom right.

- *Wall Barrier:* A rift opens up between agents and their food source! Four agents spawn at the bottom left of the map, and a wall extends up to the middle. Agents have 40 max health in order to reach the food at the bottom right.

- *Speedy Attacking:* Food has a very low health yeild, and agents must constantly work to keep their health from ticking to zero. A single agent spawns in a

2x2 environment, with two towers, so a tower is always adjacent to the agent. Towers yield 3 health, re-spawn immediately, and take 2 turns to defeat, so agents must constantly be farming the towers in order to survive.

- *Slow Attacking:* Food takes a long time to grow, and agents must ration out their food sources! Same map settings as Speedy Attacking. However, this time, towers yield 20 health and take 30 timesteps to re-spawn. Agents are capped at a max health of 20, so agents must be patient before attacking the towers in order to make full use of tower health. Applying the optimal *Speedy Attacking* strategy would mean an agent would only live for 26 timesteps.

- *Poison Chickens:* The title says it all. Agents spawn on the same map in which they were trained. However, chickens now yield -15 health, and towers yield 10 health. Living a long time requires not targeting chickens.

For each ablation above, we do an evolutionary search in the latent space to maximize reward under the ablation. We show a table of the results in Table 1. Using our diversity regularizer, we are able to find policies that can adapt to the environmental ablations without any gradient updates on our policy weights.

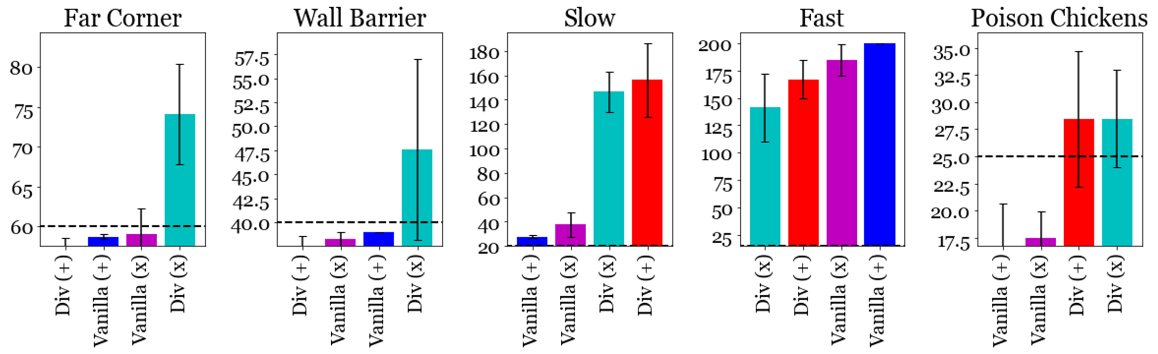


Figure 5-2: Lifetime After Optimization on Ablations
Plots of agent health (y-axis) after latent space optimization on each ablation. We plot the agent's initial health as a dashed black line. Notice that only diversity methods are able to succeed in the Locomotion and Slow Attack tasks, and can be optimized to avoid dying prematurely in the Poison Chicken task.

Results Our results show that by using our diversity regularizer, we are able to adapt to a large variety of environmental ablations with just a single generative model of policies. Performance on these ablations shows that our generator was able to identify and diversify agents along several degrees-of-freedom in the train environment. The most obvious degree-of-freedom might be agent locomotion, in which it would make sense for different agents to explore in different directions given an empty state. We thought that policy diversity would stop here, but much to our surprise, our generative model managed to actually learn a *patient* agent, that delays attacking a tower or chicken until absolutely necessary! We visualize these results in 5-3.

Takeaway Soros and Stanley [2014] hypothesizes that one of the conditions for emergent life is that solutions to life create opportunities for other solutions to exploit or co-operate with – a sort of stair step progression of evolution. For example, algae evolves in a pond, and in doing so now allows fish that eat the algae to evolve. This sort of progression of evolution requires that we actually have species and specialization in our multi-agent population. Certainly, using a naive single policy distributed across agent clones does not grant us this sort of progression. Instead, by applying our diverse policy generator, we may be able to speciate the multi-agent population in a manner that finds ways agents can co-operate and compete in a more natural way. In this section, we have shown that our generative model can emergently find ways to speciate its population, and that this helps in ablations to the train environment.

In the next section, we show that learning a generative model of policies may offer additional advantages in finding a natural training curriculum in multi-agent self-play.

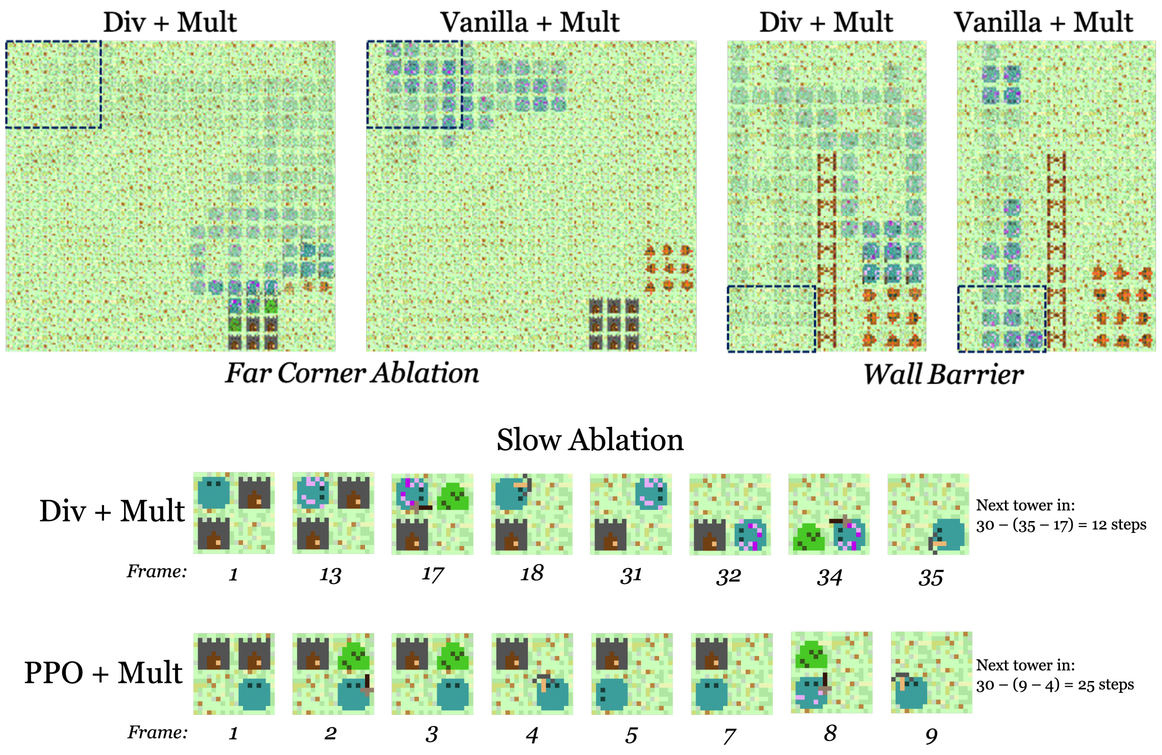


Figure 5-3: Adapting to Locomotion and Speed Ablations
Top: We show two locomotion ablations, Far Corner and Wall Barrier, and the results of latent space search from a randomly selected model from Vanilla and Div methods. The blue dashed square marks the initialization location of agents, and agents must navigate to the food locations located at the bottom right of the maps. We show the Far Corner at frame 60, and Wall Barrier at frame 40. Episode steps are stacked in a faded fashion, such that more recent episodes are darker. Notice that a high-entropy policy does not equate to a diverse policy space, and results in the Vanilla model failing to navigate far from its origin. **Bottom:** We show selected frames from the *slow* speed ablation. Notice how the Div + Mult agent refrains from attacking the tower until absolutely necessary. This allows the agent to harness more food from the tower, and survive until the next tower respawn.

Chapter 6

Auto-Curricula and Competition

This experiment uses the same environment as in Littman [1994], which introduces an environment called Markov Soccer. In this game, two agents, A and B, are placed on a grid-world soccer field, and each have the objective to 'dribble' the soccer ball into the goal that is guarded by the opposing agent. Possession of the ball is given randomly to one of A or B, and possession only changes upon a 'block' – if one agent attempts to move into a square that is occupied by the other agent. In this case, possession switches, and the move fails for the agent that was blocked. Actions for both agents are to move one block up, down, left or right, or to 'stand' and remain in-place. Actions of A and B occur on the same timestep, and the order of execution is randomized. Additionally, with some probability on each timestep, the game ends in a draw. We keep everything the same as in the original paper, except for reducing the probability the game ends in a draw on each step to $\frac{1}{20}$. We did this so that we could attempt to learn more interesting strategies that took a longer time horizon to execute.

Markov Soccer is an interesting environment, because the best policy for one agent depends on the policy of the other agent. As described in Markov Games for Multi-Agent RL, there exists a worse-case-optimal probabilistic policy for Markov Soccer, which maximizes the minimum possible score against any adversary. This strategy tends to be conservative, preferring to act towards a draw where a different policy could have obtained a higher score. On the other hand, non-worse-case-optimal strategies may be less conservative and may achieve very high scores against some opponents, but very low scores against others. Analogous to real soccer, different players have varying abilities and play styles, and a given player p1 may be optimal against p2, but not against p3.

If any single policy has its drawbacks, can we instead learn an entire space of diverse policies $\Pi := \{\pi_1, \pi_2, ..., \pi_{\text{inf}}\}$, where for any opponent, we can select a policy $\pi_i \in \Pi$ that achieves the maximum score against that opponent? Ideally, this space includes the worse-case-optimal policy, as well as other more aggressive policies. Then, just as a coach might swap out a soccer player, we can mix and match our champion as suited.

We propose that our generative model acts as a mechanism in which to achieve a diverse family of policies, and that navigating in context space lets us quickly swap out policies.

Baselines and Training As a baseline, we train a policy on Markov Soccer, using self-play and Proximal Policy Optimization with entropy regularization. We found that using entropy regularization was necessary to prevent policies trained with PPO using self-play from devolving into degenerate solutions that completely fail against opponents other than themselves. As in previous experiments, we combine the environment observation with a small random latent vector that the policy can use as it sees fit. Additionally, we test the Vanilla PPO method with both Concatenation and Multiplicative model types.

We train both Vanilla PPO and our Diversity method for 30 million time steps. In both situations, latent vectors for agents A and B are sampled uniformly and independently at the beginning of an episode, and held fixed throughout the episode.

Adaptation to Various Adversaries We evaluate adaptability to various adversaries using two methods. First, we test baselines and our method against a set of hand-coded soccer bots. These bots are designed to represent a wide gamut of strategies, some of which are far more exploitable than others. Secondly, we evaluate the learned generative models of policies by playing them in a round robin tournament against each other, by method. For evaluation, score is determined by `Wins - Losses` over the course of 1000 simulated episodes.

Against Hard-Coded Bots

Each bot plays as agent A, and the learned policy plays as agent B. Then, we attempt to optimize reward using our latent space optimization. Each bot is described below:

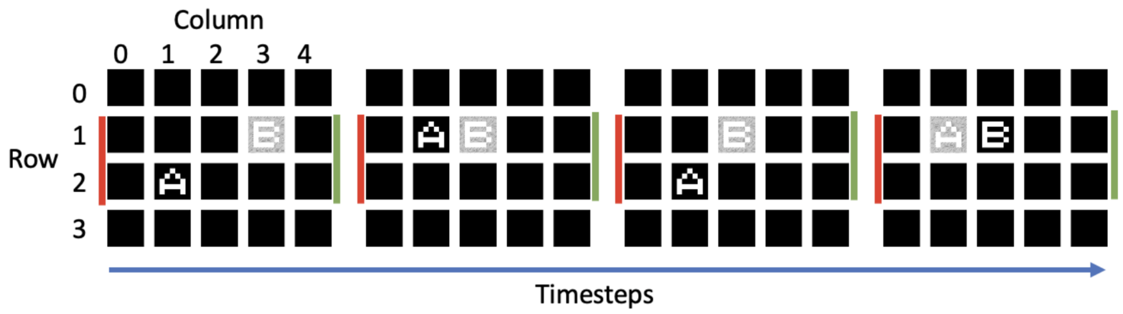


Figure 6-1: Visualization of Oscillating Defense - Column 1
Example of *Oscillating Defense - Column 1* hand-coded policy. Possession is indicated in light-gray, B aims to hit the red goal, and A aims to hit the green goal. B blindly moving forward will result in the ball being stolen half of the time.

- *Straight-Only Offense*: Agent A always starts with possession, and spawns on tile (1, 0). Agent A then always moves right, regardless of B's actions.

- *Oscillating Defense - Column 0*: Agent B always starts with possession. Agent A oscillates between (1, 0) and (2, 0) regardless of B's actions. Visualized in Figure 6-1.

- *Oscillating Defense - Column 1*: Agent B always starts with possession. Agent A oscillates between (1, 1) and (2, 1) regardless of B's actions. This bot is very exploitable, as a policy could just walk above or below the area of oscillation.

- *Stand-Only Defense*: Agent A spawns on (1, 0) and always stands. B always has possession.

- *Rule-Based Heuristics*: Possession starts randomly between A and B. Agent A follows a hand-crafted policy that follows a strong suite of rules.

- *Random*: Possession starts randomly between A and B, and Agent A acts randomly.

Round-Robin Against Each Other

Let the two generative models of policies be G_1 and G_2. Evaluation is tricky, since each generative model could map to an entire space of policies. Our goal is to evaluate the best individual policy from G_1 against the best individual policy from G_2. To compute the score of G_1 against G_2, we select the latent vector z_2 for G_2 that maximizes the expected reward of G_2 against the entire family of policies of G_1. Then, we select the latent vector z_1 for G_1 that maximizes G_1's reward against π_{G_2,z_2}. The score of G_1 vs G_1 is π_{G_1,z_1} versus π_{G_2,z_2}.

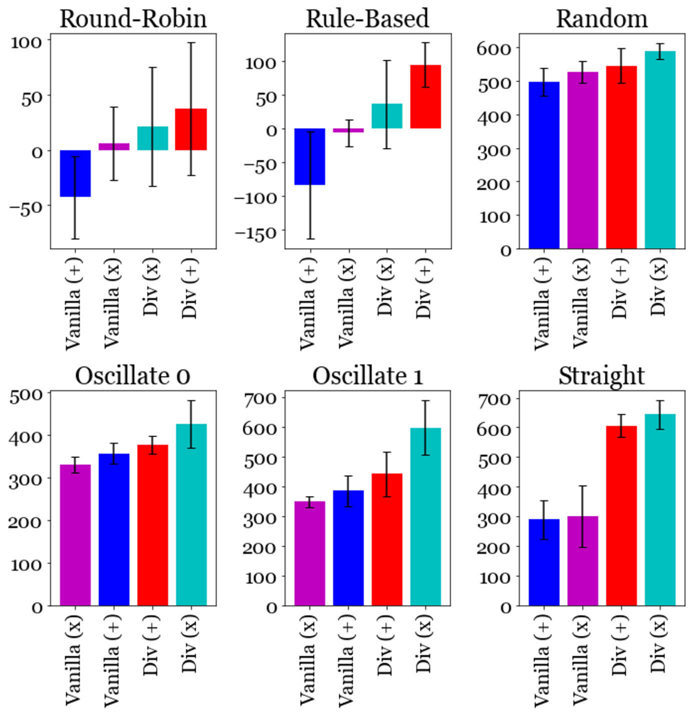


Figure 6-2: Results of learned policy spaces in a Round Robin tournament and against pre-programmed bots

Results We present results in the form of a score bar chart. Most impressively, the diversity regularization methods are able to beat our rule-based preprogrammed bot *and* win the round-robin tournament, in addition to performing the best agaist our 'naive' bots. On the other hand, Vanilla methods have significantly lower generalization across our naive bots.

The ability of our diveristy regularization methods to win the round-robin tournament and beat our rule-based heuristic bot indicates that using a generative model of policies in self-play could be beneficial. It is possible that by essentially training a 'population' of agents via our policy generator, we expose agents to a wide range of strategies that prevents degenerate behavior and enforces that individual agents learn strategies that are robust to a variety of counter-strategies. We believe that using generative models of policies warrants more exploration, as potentially a substitute or augmentation to learning a fixed size set of competitiors, as might be done in Population Based Training (Jaderberg et al. [2017]).

Robustness-Performance Tradeoff We want to understand the relationship between policy space diversity on general adaptation performance, and worst-case-optimal performance. We let general adaptation performance be the score of the G against `{Oscillate-0, Oscillate-1, Straight, Stand, and Random}`, as these are naive and exploitable opponents that may not be encountered in the standard self-play curriculum. We measure worst-case-optimal performance as the score of G against our hard-coded bot `Rule-Based`, which as seen in Figure 6-2, is a strong opponent against all methods.

In our experiment, we use the Div + Mult model and vary only the diversity regularizer coefficient. All other parameters remain the same as in the prior experiment. We use coefficient settings 0.0, 0.1, 0.2, 0.3, 0.4. Note that setting 0.0 is equivalent to Vanilla PPO with entropy regularization. We then test each method against our suite of hand-coded bots.

As seen in the Figure 6-3, there is a strong positive relationship between higher diversity regularizer coefficient, and score against the `{Oscillate-0, Oscillate-1,`

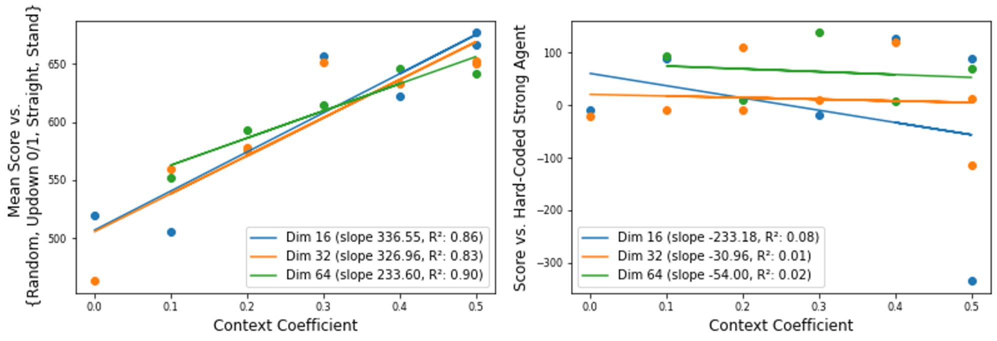


Figure 6-3: Robustness and Performance as a Function of Capacity and Diversity Regularizer Coefficient

`Straight, Stand, and Random}` tasks. On the other hand, as the diversity regularizer coefficient increases, performance against our hand-coded bot remains generally constant, provided there is enough network capacity. We conclude that, in general, it is not necessary to sacrifice adaptability for performance against a single task. For example, if there exists a worse-case-optimal policy in an environment, there is no reason it would be excluded from the learned policy space.

Chapter 7

Conclusion

In this thesis, we have presented what is, to our knowledge, a novel way of thinking about learning policies in a reinforcement learning setting. We present a framework of learning a generative model of policies. Rather than learning just one policy, we aim to find as many high-performing and individually distinct policies as possible, all compressed within the parameters of our generator. By learning a generative model of policies, we are essentially learning an entire population of agents that fit a particular environment.

Like populations of individuals or species in the real world, our policy populations contain individually unique solutions that attempt to find reward in correspondingly unique ways. In Chapter 4, we discuss how learning unique agent policies enables us to solve the problem of fitting to distinct and mutually exclusive niches in our Farmworld environment. We show how without our diversity regularization objective, policies fail to recover truly distinct agent personalities, resulting an a 'jack-of-all-trades' agent that is cloned across the Farmworld multi-agent environment. By learning a generative model of policies, we have the power to harness agent individuality, just as organisms in the natural world are unique in some way.

Also as in the natural world, the diversity of our solutions provide robustness and the capacity to adapt to changes in the training environment. Though one generated policy might fail, we can likely find a different generated policy that does not. We demonstrate in Chapters 5 and 6 how we can adapt to a surprising array of changes in an environment simply by optimizing our policy latent space in a fast and efficient manner. In Farmworld, we are able to emergently discover agent policies that have locomotion preferences, in addition to agents that exhibit a sense of patience and moderation. During our exploration of Markov Soccer, we discussed how any single deterministic policy has its weaknesses, but by learning a learning a ‘team’ of policies via our policy generator, we are able to adapt to and exploit a variety of unexpected adversaries.

We have certainly have only scratched the surface of the possibilities unlocked by thinking about reinforcement learning as a policy generation problem. Exploring new, larger, and even more open-ended environments is certainly essential to testing the limits of policy generators. Yet another research direction could be integrating high-level policy generators into hierarchical reinforcement learning, potentially combining our diversity regularizer with option discovery methods. This could provide benefits, since our method is not inherently an exploration strategy, and could be difficult to optimize in very sparse reward settings or when action spaces are too high dimensional. Additionally, in Chapter 6, we hypothesized that learning a population of agents improved the self-play learning curriculum during Markov Soccer training. Nevertheless, there is a lot more work to be done in understanding how a policy generator could impact emergent multi-agent cooperation and competition. Ultimately, by grounding reinforcement learning in the processes that sparked true biological intelligence, we can hope to get one step closer to true silicon intelligence.

Appendix A

Algorithm Pseudocode

Algorithm 1 Latent Space Optimization

```
1: Input: $g$ the number of optimization generations
2: Input: $E$ an environment
3: Input: $G$ a policy generator
4: Input: $Z$ a latent distribution with dimension $k$
5: Initialize: best ← descending sorted array
6: for $i = 1, 2, ..., g$ do
7:     explor $\sim$ Unif([0, 1])
8:     $r \sim$ Unif([0, 1])
9:     if (explor $\leq 0.5$ and $i \leq \frac{3}{4}g$) or len(best) $\leq 10$ then
10:        if $r \leq 0.5$ or len(best) $\leq 10$ then
11:            $z \sim Z$                                          ▷ Random Sampling
12:        else
13:            $z \leftarrow$ sample(best[0:10])
14:            $z \leftarrow z +$ project$_Z$(Unif[-0.1, 0.1]$^k$)            ▷ Mutation
15:        end if
16:    else
17:        if $r \leq 0.5$ then
18:            $z \leftarrow$ sample(best[0:10])                          ▷ Replication
19:        else
20:            $z \leftarrow$ pop(best[0:10])                               ▷ Pruning
21:        end if
22:    end if
23:    $score \leftarrow$ Reward from running $\pi_{G,z}$ on $E$
24:    best.push($z$) with key $score$
25: end for
26: Return best[0]
```

Algorithm 2 PPO with Diversity Regularization

```
1: $m$ the number of sampled latents in diversity estimation
2: $b$ the number of sampled states in diversity estimation
3: $k$ latent vector size
4: for for iteration = 1, 2, ... do
5:     Let $B$ be an empty batch of (s, a, r) tuples
6:     for actor $a$ = 1, 2, ..., $N$ do
7:         Sample latent $z$ from latent distribution
8:         $B \leftarrow$ Run policy $\pi(\cdot|\theta_{old}; z)$ in environment for $T$ steps
9:         Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
10:    end for
11:    Sample $M \in \mathbb{R}^{m \times k}$ from the latent distribution        ▷ latent matrix
12:    Sample a batch $S$ of $b$ states from $B$
13:    $L_{div} \leftarrow 0$
14:    for $i = 1, 2, ..., m-1$ do
15:        for $j = i+1, i+2, ..., m$ do
16:            $L_{div} \leftarrow L_{div} + \frac{1}{b}\sum_{s \in S} D(\pi(s|\theta_{old}, M^{(i)}), \pi(s|\theta_{old}, M^{(j)}))$
17:        end for
18:    end for
19:    $L_{div} \leftarrow \frac{2}{m(m+1)} L_{div}$        ▷ Scale by number of policy-distance pairs
20:    Optimize surrogate $L_{surr} + L_{ctx}$ wrt $\theta$
21:    $\theta_{old} \leftarrow \theta$
22: end for
```

Appendix B

Benchmarks and Toy Environments

B.1 Benchmarks

In this work, we compare our method against several existing option discovery methods. These benchmark methods, and any modifications we made during the course of training are reported here.

DIAYN DIAYN (Eysenbach et al. [2018]) originally attempts to maximize the mutual information between the state and a discrete categorical latent vector, as shown in Equation 3.1. On the other hand, our method uses a continuous latent in order to find a potentially unbounded number of diverse skills. In order to get DIAYN working with continuous latent vectors, we train the discriminator to regress the latent, rather than predict the latent category. We add this intrinsic reward to the extrinsic environmental reward, giving us the new reward function r':

$$r'_t = err_t + r_t$$

where

$$err_t = -(q_\phi(s_t) - z)^2 - \text{mean}(err_{batch})$$

z is the latent vector, $\text{mean}(err_{batch})$ is the mean discriminator error across the update batch. We subtract by this mean so that on average, expected agent reward equals only what is provided by the extrinsic environment. Otherwise, DIAYN stuggled with balancing dense extrinsic environmental rewards from the experiment with the intrinsic discriminator reward. Finally, since we are attempting to learn a generative model of policies, and not options, we keep z fixed throughout an agent episode.

Additionally, we experiment with providing the discriminator action information as well which we label in our experiments DIAYN + action. This implementation is the same, except that we modify err_t to be

$$err_t = -(q_\phi(s_t, \sigma(\pi_z(s_t))) - z)^2 - \text{mean}(err_{batch})$$

where σ is the softmax operator. We optimize both of these via PPO with entropy regularization. We use the same regression strategy for continuous latents as in DIAYN + action.

Differentiable Mutual Information (MI) Zhang et al. [2021] and Wang et al. [2020] introduce a variant of a mutual information maximization objective as in option discovery that is differentiable through the action logits of a policy. In our implementation of this variant, we attempt to maximize the following objective via gradient descent:

$$\max_{\pi,\phi} \mathop{\mathbb{E}}_{z\sim p(z)} \left[\mathop{\mathbb{E}}_{\tau\sim\pi,z} \left[\sum_{t=0}^{T} (q_\phi(s_t, \sigma(\pi_z(s_t))) - z)^2 \right] \right] \tag{B.1}$$

alongside the standard clipped surrogate PPO objective, and an entropy regularization term where applicable.

B.2 Experiments in Toy Environments

To test that our baseline comparisons were working, we evaluated all methods on two toy environments: CartPole (Brockman et al. [2016]) and a Multi-Agent MultiGoal.

B.2.1 CartPole

Environment CartPole is a standard control task in RL, with a continuous four-dimensional observation space and a two-dimensional discrete action space for pushing the cart left or right. The objective of CartPole is to move the cart such that the pole remains standing up. If the pole sways too far to the left or right from the 90-degree perpendicular from the ground, then the episode terminates. Otherwise, we set reward to be 0.1 per timestep, and episodes can last for 200 timesteps.

Experiment We perform a similar analysis of each method as in the Farmworld and Markov Soccer experiments. Our procedure is to train each method on the normal CartPole task, and then evaluate performance under ablated conditions, and modified reward functions. In particular, we evaluate each method using:

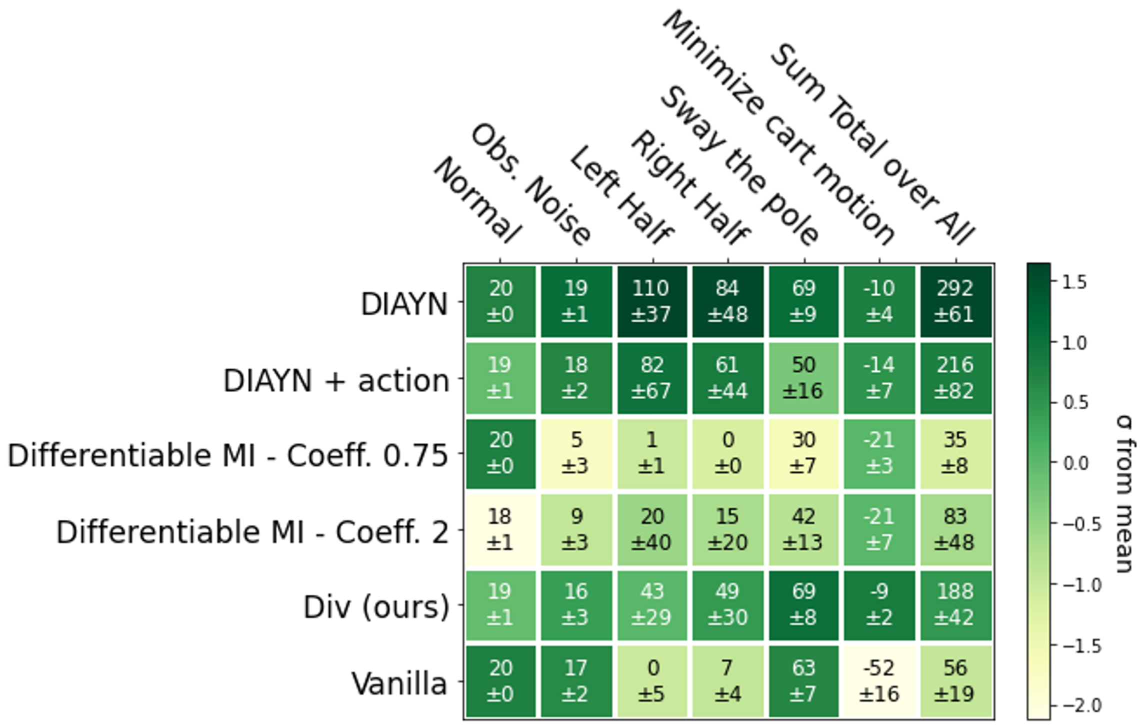
Environmental Ablations

- Normal: The standard CartPole task
- Observational Noise: Uniform [-0.25, 0.25] noise is added to the observations.

New Reward Functions

- Left: the policy should stay on the left side of the x-axis: $r(\tau) = \sum_{s \in \tau} -s_{\texttt{x-pos}}$
- Right: the policy should stay on the right side of the x-axis: $r(\tau) = \sum_{s \in \tau} s_{\texttt{x-pos}}$
- Sway the pole: the policy should maximize the angular velocity of the pole, without ending the episode: $r(\tau) = \sum_{s \in \tau} |s_{\texttt{angular-velocity}}|$
- Minimizing cart motion: the policy should keep the cart as still as possible during the course of the episode: $r(\tau) = -\sum_{s \in \tau} s_{\texttt{x-velocity}}$

Results In Figure B-1, we report numerical scores by method, and by ablation or modified reward function. Additionally, in Figure B-2 we visualize each ablated environment or new reward function. We leave out *Minimizing cart motion*, because all methods except for Vanilla PPO managed to find a hack around this method: they



	Normal	Obs. Noise	Left Half	Right Half	Sway the pole	Minimize cart motion	Sum Total over All
DIAYN	20 ±0	19 ±1	110 ±37	84 ±48	69 ±9	-10 ±4	292 ±61
DIAYN + action	19 ±1	18 ±2	82 ±67	61 ±44	50 ±16	-14 ±7	216 ±82
Differentiable MI - Coeff. 0.75	20 ±0	5 ±3	1 ±1	0 ±0	30 ±7	-21 ±3	35 ±8
Differentiable MI - Coeff. 2	18 ±1	9 ±3	20 ±40	15 ±20	42 ±13	-21 ±7	83 ±48
Div (ours)	19 ±1	16 ±3	43 ±29	49 ±30	69 ±8	-9 ±2	188 ±42
Vanilla	20 ±0	17 ±2	0 ±5	7 ±4	63 ±7	-52 ±16	56 ±19

Figure B-1: Reward in CartPole under Various Conditions
Reward over three random seeds, by each method.

found a latent z that immediately terminated the episode by swaying the pole too much. It turned out that since the τ was terminated so quickly, this minimized the sum of cart velocity over the course of τ.

As one can see in Figure B-1, DIAYN performed quite well over all ablations and tasks, validating our implementation of that method. The Differentiable Mutual Information methods did not perform nearly as well, and we see a sharp decrease in performance on the train task (`Normal`) when we attempt to increase the coefficient of the MI loss. As expected, Vanilla PPO performs optimally on the train task, but fails to generalize to our ablated reward functions via latent space optimization alone.

B.2.2 Multi-Agent MultiGoal

Environment Our MultiGoal task is a variant of the standard 2D gridworld navigation task. Agents start in the center square [0, 0] of the 20 × 20 grid, and must

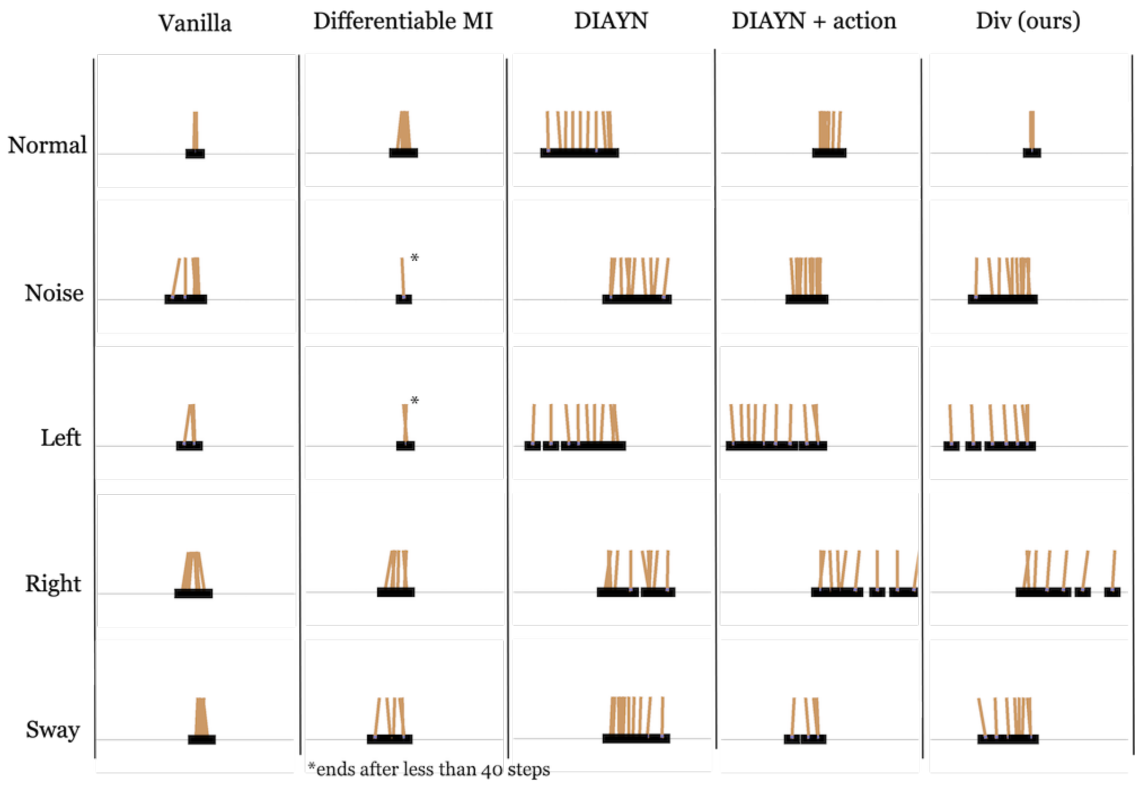


Figure B-2: Visualization of Stacked CartPole Trajectories

navigate to one of four goals placed at [8, 8], [8, -8], [-8, 8], [8, 8]. Agents get 0 reward for all actions that do not take them to a goal, and 1 reward if they reach a goal. Additionally, episodes terminate immediately after reaching the goal. Observations are the one-hot encoding of the discrete agent x, y position, and an additional 3-dimensional latent vector z, integrated in the concatenation or multiplicative fashion.

The caveat is that each MultiGoal episode contains 40 agents that spawn at the center, and each goal has ‘food support’ for only 10 agents. So if all agents act ‘the same’ and navigate to the same goal, then the expected reward will only be 0.25 per agent. However, if agents learn to specialize based on the additional latent z, then they can achieve a significantly higher expected reward. Episodes terminate after 20 steps, so agents can only attempt to find one goal.

Thus, the Multi-Agent MultiGoal experiment is both a challenging exploration task, and a multi-agent specialization and coordination task.

Experiment We do not run any ablations on this environment, as our experiment simply aims to measure how many goals each method is able to find.

Results The methods that are able to consistently reach at least 4 goals are our Diversity Policy Generator (Div), DIAYN, and Differentiable Mutual Information with action noise. Vanilla PPO, DIAYN + action, and Differentiable Mutual Information without any action noise generally fail to find more than 2 goals. We visualize agent trajectories in C-1, and show graphs of reward per method in C-2.

Appendix C

Figures

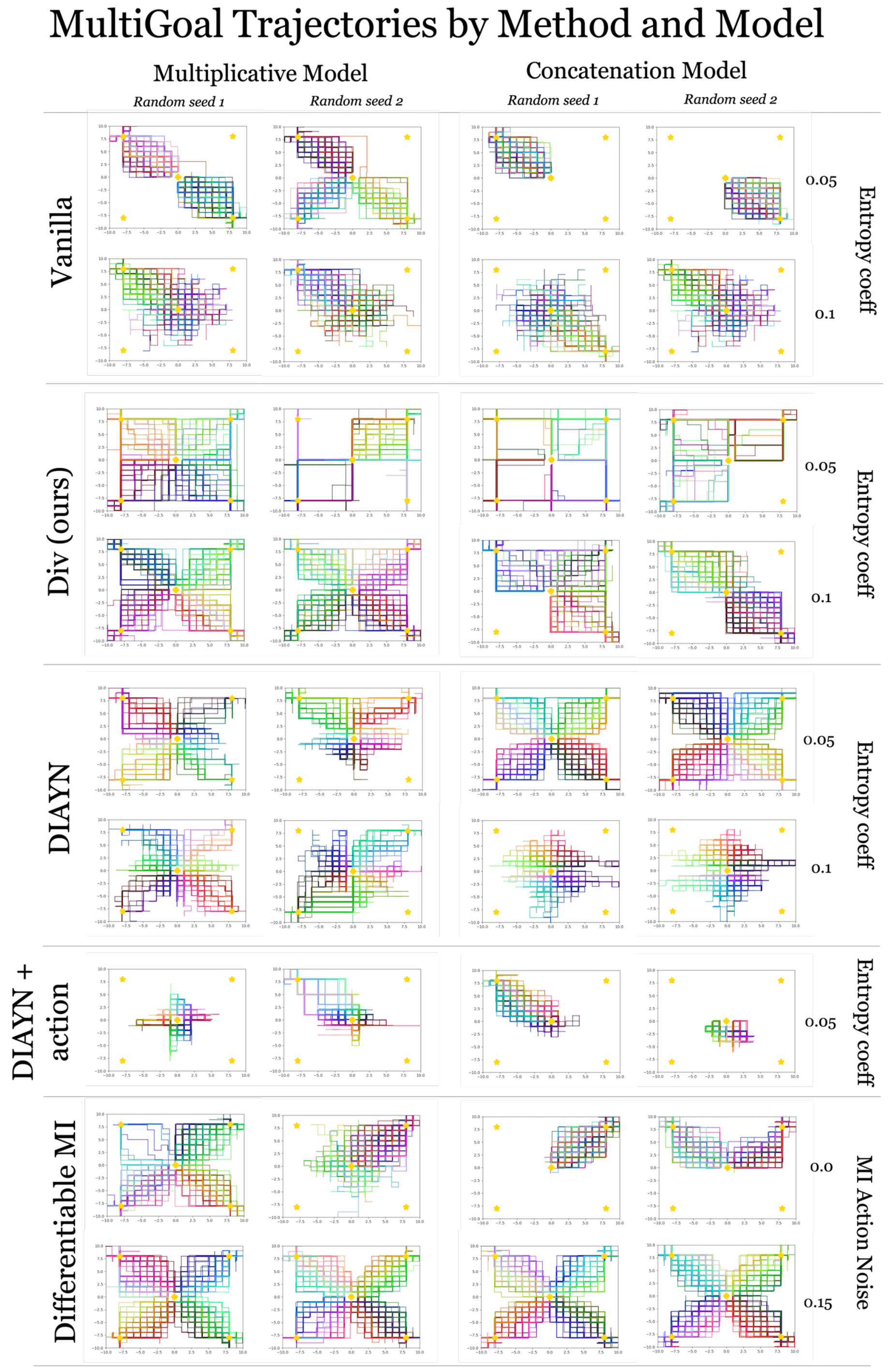


Figure C-1: Visualization of Trajectores of Agents in the Multi-Goal Environment Each agent trajectory is reprsented as a connected sequence of points starting from the origin. Additionally, we color each agent trajectory with its corresponding 3-dimensional latent z, represented proportionally as RGB values.

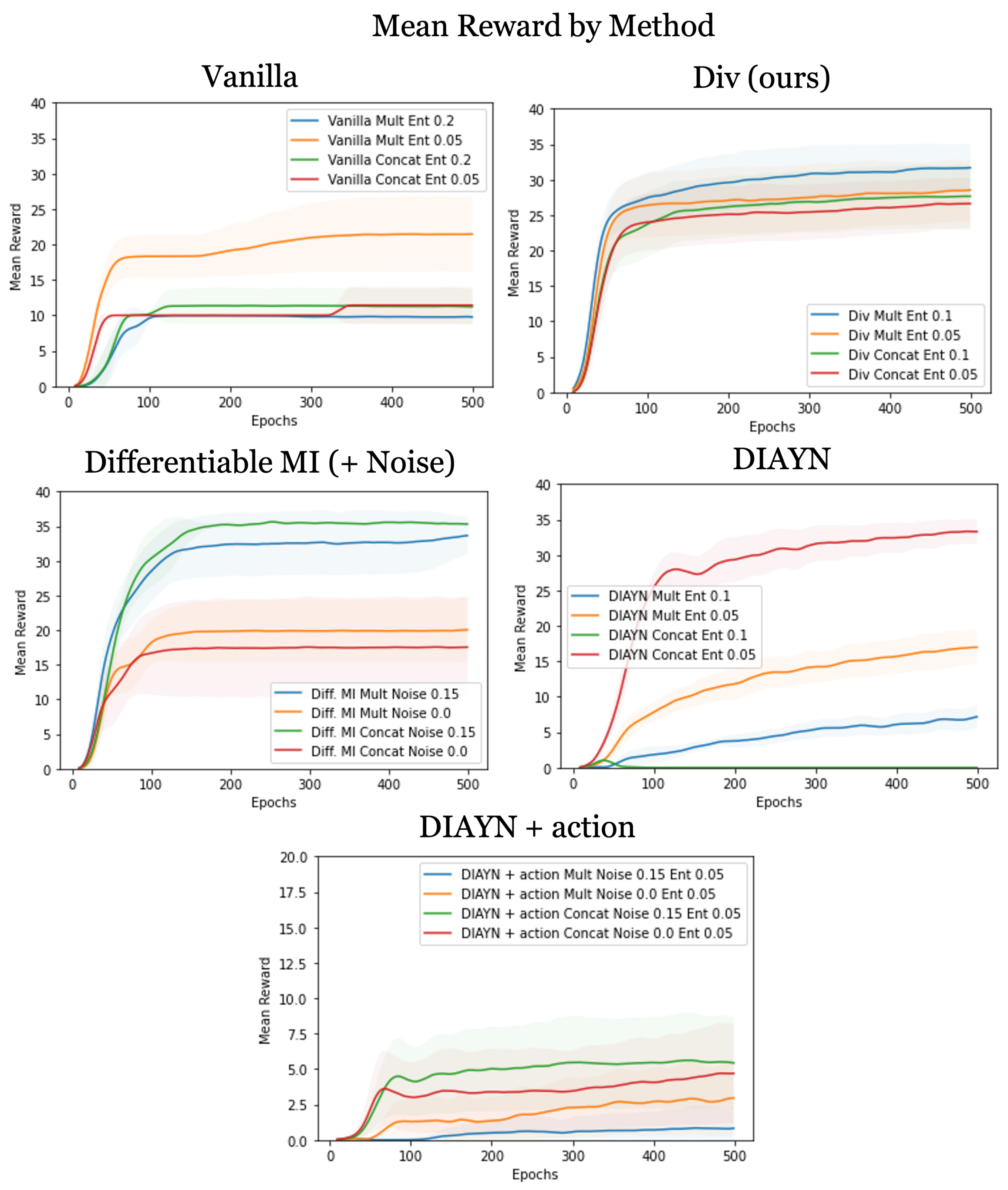


Figure C-2: Reward Multi-Agent MultiGoal over Epochs
Reward over three random seeds, by each method. Shaded region represents standard deviation across seeds.

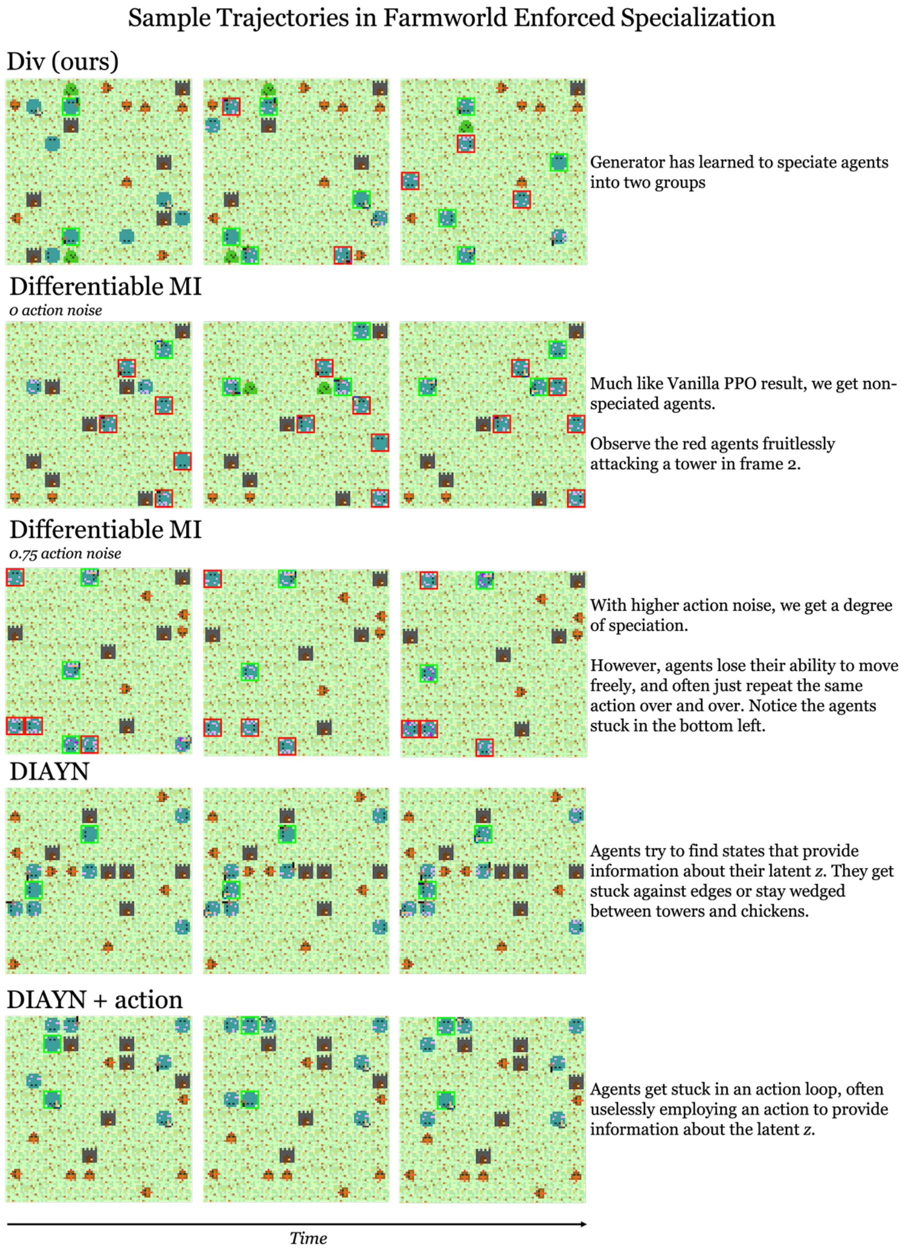

Sample Trajectories in Farmworld Enforced Specialization
Div (ours)
Generator has learned to speciate agents into two groups
Differentiable MI
0 action noise
Much like Vanilla PPO result, we get non-speciated agents.
Observe the red agents fruitlessly attacking a tower in frame 2.
Differentiable MI
0.75 action noise
With higher action noise, we get a degree of speciation.
However, agents lose their ability to move freely, and often just repeat the same action over and over. Notice the agents stuck in the bottom left.
DIAYN
Agents try to find states that provide information about their latent z. They get stuck against edges or stay wedged between towers and chickens.
DIAYN + action
Agents get stuck in an action loop, often uselessly employing an action to provide information about the latent z.
Time

Figure C-3: Sample Trajectories in Farmworld Niche Specialization
For visualization, we have annotated each agent with a green box if it is now a tower-agent, and a red box if it is now a chicken-agent. These annotations are not visible as part of the agent state, and are just for demonstrating how in many cases, a red chicken-agent will fruitlessly attack a tower, or vice versa with a green tower-agent.

Appendix D

Model Architectures and Experimental Hyperparameters

Unless otherwise mentioned, we used optimized our policies using a clipped PPO surrogate objective with learning rate 3e-4. Advantages were computed using Generalized Advantage Estimation, with a γ discount factor of 0.99 and a λ smoothing parameter of 1. We use the RLLib (Liang et al. [2018]) framework for training, using their default PPO configuration. For all experiments, model architectures are as seen in Figure 2-2. Importantly, we always use *separate value and policy networks*. Attempts to combine these networks generally resulted in non-diverse policy spaces, which we believe is a result of the importance of the value function in recognizing the differing expected rewards conditional on each latent from the latent space.

For multi-agent environments, batch sizes are always in *agent* steps, rather than in *environment* steps. Thus, if there are 40 agents in an environment, then 1 environment step is 40 agent steps.

To optimize our diversity regularization objective, we use parameters $m = 10, b = 30, k = 3$, as detailed in 2. However, preliminary investigation into the effect of these hyperparameters indicates that it is possible to get away with even smaller samples of latent vectors and states, while still effectively optimizing for a diverse policy manifold.

Additionally, we explore various coefficients to weigh the importances of our diversity regularizer, and, where applicable, the differentiable MI objective. Unless otherwise specified we use a diversity regularizer coefficient on our novel objective of coefficient of 0.1 in Farmworld, 0.1 in CartPole, 0.2 in Markov Soccer, and 0.5 in MultiGoal. We use a differentiable MI coefficient of 0.75 and 2.0 in CartPole, 0.75 in MultiGoal, 1 and 2 in Farmworld. In Farmworld and MultiGoal, we additionally augment the differentiable MI action probabilities with uniform noise in the range [0, 0.15], which we report as differentiable MI + Noise.

For all methods, we generally use an 0.05 entropy coefficient, except in Markov Soccer in which we also run PPO with 0.1 entropy coefficient and were able to achieve slightly stronger performance. In our Markov Soccer experiment, we report the average of these two PPO results.

Batch size	4000
Minibatch size	400
SGD iterations per batch	10
Training epochs	200
Hidden dimension	16
Value Activations	ReLU
Policy Activations	Tanh

Table D.1: CartPole

Batch size	4000
Minibatch size	400
SGD iterations per batch	10
Training epochs	500
Hidden dimension	32
Value Activations	Tanh
Policy Activations	Tanh

Table D.2: Multi-Agent MultiGoal

Batch size	8000
Minibatch size	8000
SGD iterations per batch	10
Training epochs	10 thousand
Hidden dimension	64
Value Activations	Tanh
Policy Activations	Tanh

Table D.3: Enforced Specialization and Farmworld Ablation Experiment

Batch size	4000
Minibatch size	2000
SGD iterations per batch	10
Training epochs	1200
Hidden dimension	128
Value Activations	Tanh
Policy Activations	Tanh

Table D.4: Agent Trait Experiment

Batch size	8000
Minibatch size	8000
SGD iterations per batch	10
Training epochs	10 thousand
Hidden dimension	64
Value Activations	Tanh
Policy Activations	Tanh
GAE lambda	0.95
GAE gamma	0.9

Table D.5: Markov Soccer

Bibliography

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational Option Discovery Algorithms. *arXiv e-prints*, art. arXiv:1807.10299, July 2018.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent Tool Use From Multi-Agent Autocurricula. *arXiv e-prints*, art. arXiv:1909.07528, September 2019.

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent Complexity via Multi-Agent Competition. *arXiv e-prints*, art. arXiv:1710.03748, October 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv e-prints*, art. arXiv:1606.01540, June 2016.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. *arXiv e-prints*, art. arXiv:1802.06070, February 2018.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic Neural Networks for Hierarchical Reinforcement Learning. *arXiv e-prints*, art. arXiv:1704.03012, April 2017.

Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal Compact Bilinear Pooling for Visual Question Answering and Visual Grounding. *arXiv e-prints*, art. arXiv:1606.01847, June 2016.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent Space Policies for Hierarchical Reinforcement Learning. *arXiv e-prints*, art. arXiv:1804.02808, April 2018.

Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, and Chun-Yi Lee. Diversity-Driven Exploration Strategy for Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1802.04564, February 2018.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks. *arXiv e-prints*, art. arXiv:1711.09846, November 2017.

Joel Lehman and Kenneth Stanley. Evolving a diversity of creatures through novelty search and local competition. pages 211–218, 01 2011a. doi: 10.1145/2001576.2001606.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011b. doi: 10.1162/EVCO_a_00025.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, 10–15 Jul 2018. URL `http://proceedings.mlr.press/v80/liang18b.html`.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157–163, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. ISBN 1558603352.

Muhammad A. Masood and Finale Doshi-Velez. Diversity-Inducing Policy Gradient: Using Maximum Mean Discrepancy to Find a Set of Diverse Policies. *arXiv e-prints*, art. arXiv:1906.00088, May 2019.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv e-prints*, art. arXiv:1504.04909, April 2015.

Jack Parker-Holder, Aldo Pacchiano, Krzysztof Choromanski, and Stephen Roberts. Effective Diversity in Population Based Reinforcement Learning. *arXiv e-prints*, art. arXiv:2002.00632, February 2020.

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016. ISSN 2296-9144. doi: 10.3389/frobt.2016.00040. URL `https://www.frontiersin.org/article/10.3389/frobt.2016.00040`.

L. Soros and Kenneth Stanley. Identifying Necessary Conditions for Open-Ended Evolution through the Artificial Life World of Chromaria. volume ALIFE 14: The Fourteenth International Conference on the Synthesis and Simulation of Living Systems of *ALIFE 2020: The 2020 Conference on Artificial Life*, pages 793–800, 07 2014. doi: 10.1162/978-0-262-32621-6-ch128. URL `https://doi.org/10.1162/978-0-262-32621-6-ch128`.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL `https://www.sciencedirect.com/science/article/pii/S0004370299000521`.

Tonghan Wang, Heng Dong, Victor Lesser, and Chongjie Zhang. ROMA: Multi-Agent Reinforcement Learning with Emergent Roles. *arXiv e-prints*, art. arXiv:2003.08039, March 2020.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan Salakhutdinov. On Multiplicative Integration with Recurrent Neural Networks. *arXiv e-prints*, art. arXiv:1606.06630, June 2016.

Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical Reinforcement Learning By Discovering Intrinsic Options. *arXiv e-prints*, art. arXiv:2101.06521, January 2021.