Adaptable Agent Populations via a Generative Model of Policies

Anonymous Author(s)
Affiliation
Address
email

Abstract

In the natural world, life has found innumerable ways to survive and often thrive. Between and even within species, each individual is in some manner unique, and this diversity lends adaptability and robustness to life. In this work, we aim to learn a space of diverse and high-reward policies on any given environment. To this end, we introduce a generative model of policies, which maps a low-dimensional latent space to an agent policy space. Our method enables learning an entire population of agent policies, without requiring the use of separate policy parameters. Just as real world populations can adapt and evolve via natural selection, our method is able to adapt to changes in our environment solely by selecting for policies in latent space. We test our generative model's capabilities in a variety of environments, including an open-ended grid-world and a two-player soccer environment.

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?

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 encompassing a range of skillful behaviors that are adaptable and robust to changes in the task and environment. We name our method of learning a space of *ad*aptable *a*gent *p*olices 'ADAP'.

Previous work has touched on ideas akin to a generative model of policies. In hierarchical RL, the the high-level policy controller can be considered a generator of sub-policies that are 'options' [24, 4, 5]. But these methods are designed with the intent to find decomposable skills that aid in the construction of just one downstream controller policy. A body of prior work that aligns more closely with our goals is that of quality diversity [18], 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, each with their own policy weights. This consumes more time and training resources, and limits the number of potential behaviors. Our work integrates the goals of quality diversity into time and memory efficient deep RL 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? We propose two primary reasons. First, 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. By finding points across this manifold, we avoid 'having all eggs in one basket,' granting robustness and adaptability to environmental changes. In the event of a change,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

we are able to adapt our generated population to select individuals that can still survive given the ablation, much like natural selection drives evolution in the real world. Secondly, using a generative model of policies as a population of agents makes intuitive sense in multi-agent environments, in which different agents should have the capacity to act like they are 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 may reduce the multi-modality of the agent population, resulting in essentially a single 'average' agent.

The rest of the paper is organized as follows. First we introduce our generative model of policies and the diversity objective that guides its learning. Next, we explore the potentials of learning a population of agents by ablating environments and then optimizing generated policies to fit each ablation, directly in latent space. We study two main environments: Markov Soccer [11], and Farmworld, a new environment we have developed for testing diversity in a multi-agent, open-ended gridworld. Included in the supplement are more results on toy environments of CartPole [2], a standard multi-goal environment, and a toy environment of modeling multiple algebraic functions.

2 Method

We learn a mapping $G : \phi, Z \rightarrow \Pi$, from generator weights ϕ and latent distribution Z to a space of policies Π. The generator G_ϕ itself is not a policy, and must be conditioned on a $z \sim Z$ in order to define a learned set of behaviors that map states to actions. The latent vector z is a stochastic parameter of G_ϕ, and is sampled once at the beginning of each agent episode. In our experiments, we sample z from the three dimensional unit sphere, which is low dimensional enough to quickly search the latent space. Note that unlike [6], we do not need to build the weights of each agent policy, allowing memory efficiency when training many agents simultaneously. In practice, to compute agent actions, we provide the current environment observation and the agent's latent z as inputs to G_ϕ. The weights ϕ are differentiable with respect to our diversity regularizer (1) and agent policy actions, and we update ϕ like normal policy weights. Generally, we drop the ϕ from notation for simplicity.

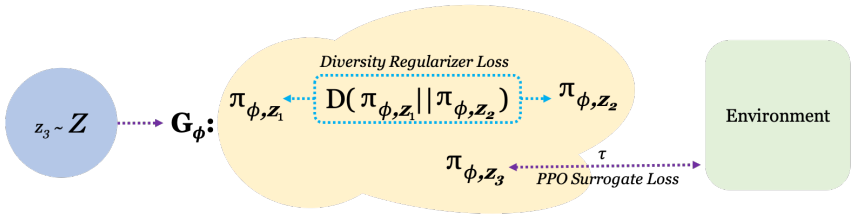

Figure 1: Method Diagram: Upon agent initialization, we sample a latent $z \in Z$, which along with ϕ defines an agent policy for the episode. The diversity regularizer loss is defined over the expected pair (z_i, z_j) and is thus optimized over all possible policies – not just the current policy used on the environment.

Diversity Regularization In order to learn a diverse space of unique policies, we introduce a diversity regularization objective. 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. To this end, we optimize the objective (1):

$$\min_{\phi} \mathop{\mathbb{E}}_{s \in S} \left[\mathop{\mathbb{E}}_{\substack{z_i, z_j \in Z \\ z_i \neq z_j}} \alpha \exp\left(-D_{KL}(\pi_{\phi,z_i;b}(s) \| \pi_{\phi,z_j;b}(s))\right) \right] \tag{1}$$

in which D_{KL} is the KL-divergence between the two policy action distributions π_{θ,z_i} and π_{θ,z_j}, and b is a smoothing constant over the action distributions. We use the smoothing constant b because otherwise, the KL-divergence may quickly approach infinity as probability mass over a particular action approaches zero.

Model Architecture Similarly to prior work [5], we have found that richer integrations between the latent vector and the context can yield a more multi-modal policy space. We call our baseline model the concatenation model, which concatenates the latent vector z with the environment observation at

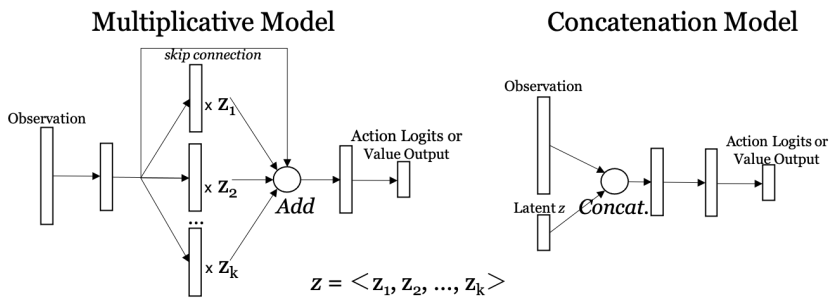

Figure 2: Model Architectures for Latent Integration

each timestep. We additionally introduce a multiplicative model for latent integration. 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 online fashion using gradient descent, in conjunction with a PPO [21] clipped-surrogate objective and entropy regularization objective. We wrote and trained our models and algorithms using the RLLib framework [10]. We trained our models on a collection of 40 CPUs, using an internal compute cluster. We ran three seeds for each experiment, of which the longest single trial ran for 30 million environment steps on CPU, training in under 24 hours using a single policy update worker and two rollout workers.

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 (dimensionality 3 in our experiments).

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 the supplement. By conditioning G via a latent subspace rather than modifying the network weights to the ablated environments (as in transfer learning), we can quickly adapt back and forth to various situations: the search procedure often takes *less than 30 seconds*, or 100 episode rollouts, to find high quality solutions if they exist. In the event that episodes have a high degree of variablility per run – such as in the Markov Soccer environment – it may be necessary to run several episodes per latent vector and average the returns.

Considerations of Impact What are potential drawbacks of learning such a policy space? By blindly incentivizing diverse policy behaviors, we must be cognizant of what each policy in policy space is capable of doing. We would not, for example, want to deploy a system into the real world without understanding fully the actions-given-state modulated by each latent z one may sample. It would be possible that an agent makes a dangerous, or even malicious, action when we would otherwise not expect it to – because it was incentivised to make that action with respect to the diversity objective. Should such systems be one day deployed in real life, we must properly vet high-risk states and ensure safe and predictable actions.

3 Related Work

Option Discovery for Hierarchical RL The option framework introduced by [24] could be thought of as learning a generator of skills, which are temporal abstractions over actions that can be used by a downstream, higher-level controller. Recent works [4, 1, 5] in option discovery learn a fixed set of skills that are discriminable by observed state or trajectory, and oftentimes learn skills without any extrinsic environmental reward. On the other hand, our method learns a continuous space of diverse and stand-alone policies trained to maximize extrinsic environmental reward, in which diversity is defined as difference in actions taken given state. Additionally, requiring discriminability via state or trajectory impose diversity constraints along a and specific task-dependent axis.

Goal-Conditioned Reinforcement Learning Yet another way to induce diverse policy behaviors is through using goal-conditioned policies [8, 15, 20] that use a family of task-defined value or Q functions or expert trajectories [3] to incentivize diversity. These methods require structure in how to define diversity, such as defining a value function family over states [20].

Multi-Agent Roles ROMA [25] has studied how to generate roles in a multi-agent setting. However, these roles are conditional on agent observations, and are learned jointly with other roles via a joint action-value function; wheras we learn fixed policies trained to optimize individual reward only. ROMA learns a space of roles that are meant to work well together on a given multi-agent task, while ADAP can be used in single or multi-agent environments. MAVEN [12] uses a hypernetwork to generate agent policies, using QMIX [19], while also optimize mutual information of joint actions – such that a latent z represents actions of many agents, rather than an individual agent.

Quality Diversity The evolutionary computing community has developed various quality diversity (QD) algorithms that aim to find a balance of novel and high-performing individuals within a population. Novelty is defined by morphological [22, 23] or behavioral characteristics of the individual [9, 14], which must be specified by the user for the task. For example, in a maze navigation task, the behavioral characteristic could be the final resting location of the individual, and agents may be selected based on how far away they end up from an archive of individuals. One issue with these algorithms is that they can be very sensitive to the choice of behavioral characteristic [17], and in many cases requires domain knowledge for understanding which axes of diversity make sense for a given environment. On the other hand, ADAP does not require the construction of a behavioral characteristic, as we instead optimize for diversity directly from the environment action space. Furthermore, QD algorithms optimize individuals via genetic algorithms or evolutionary search, while ADAP is able to directly optimize for diversity and policy credit assignment via gradient descent.

There are several prior works that aim to connect ideas of quality diversity with deep reinforcement learning. Like quality diversity algorithms, these methods attempt to optimize a fixed-size population or archive of policies to be distinct from each other. [13] finds a small set of separate-weight policies that find diverse trajectories by iteratively adding one novel policy at a time to a set of existing and fully-trained policies. [7] uses a KL-divergence over policies; but a policy's diversity is optimized over previous SGD updates of itself, thus limiting the potential multi-modality of solutions. [16] optimizes for diversity of the total population via maximizing the determinant of a population distance matrix, but works best only with small populations of size three or five.

All of these QD methods utilize a fixed-size population of separate-weight individuals. In contrast, our work integrates the goals of quality diversity into a memory efficient deep RL framework by simulating an entire population of agents into the weights of single generative model of policies, in which the number of unique individuals is bounded only by the capacity of the generator.

4 Experiment 1: Can We Learn a Multi-Modal Policy Space?

A good generative model of policies should be able to represent a multi-modal space of behaviors, that is: agent policies that can act like they are different individuals. Our generative model uses a shared parameter set across all agents, but naively using shared weights could result in a 'jack-of-all-trades' or 'average' agent, which is precisely what we wish to avoid. In this experiment, we test whether our policy generator is able to model policies for *unique individuals* in a multi-agent setting.

Environment We test our learning G 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 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. There 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's observation is composed of a flattened representation of the units in a configurable L1 radius centered at its location. Full details of the Farmworld environment are provided in the supplement.

Niche Specialization Experiment To test the multi-modality of G, we set up the Farmworld environment with a hidden rule specific to this experiment: 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. We call this part of the environment the *enforced specialization rule*. This rule checks if agents can behave competently within distinct `tower` and `chicken` niches. In this experiment, and all following experiments, we train agents to only optimize their own expected reward (i.e. no shared value functions or group rewards).

Baselines In addition to ADAP, we test Vanilla PPO (which is ADAP without the diversity regularization, DIAYN [4], and a modification of DIAYN which we will call DIAYN*. Note that we still provide the Vanilla baseline latent integration. Briefly, DIAYN* uses a continuous distribution rather than a categorical distribution, and attempts to minimize the mean-squared error between the predicted context $\phi(s)$ and the actual context, where ϕ is the discriminator and s is the state. We provide more details in the supplement. For both DIAYN and DIAYN*, we augment the intrinsic reward with extrinsic environmental reward, and searched over hyperparameter settings to find the best results.

Results ADAP is able to learn a more multi-modal policy space than any of the other baselines. In other words, the policies generated by different latent vectors allowed ADAP agents to actually act like unique individuals, even though they shared the same generator parameters. Our results also indicate that using a multiplicative model can result in a higher degree of policy space multi-modality. We explain our results in more detail in the following paragraph.

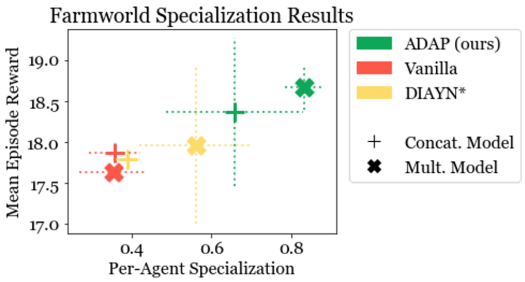

Figure 3: Reward versus Specialization

We can see in Figure 3 that ADAP is able to attain the highest episode reward, defined by the mean reward collected over all agents. This, however, is not necessarily the most interesting point. ADAP also has the highest 'per-agent specialization', which we compute as one minus the per-agent entropy of its chicken and tower attacks. That is to say, if an agent only attacks towers, it will have a specialization of one, whereas if it attacks equal numbers of towers and chickens, it will have a specialization of zero. Now, we can see that ADAP was able to encode niches into the latent space Z, thus partitioning agents into chicken and tower specializers. Indeed, during episode rollouts, ADAP agents that fall on one side of the partition will only attempt to attack their respective unit type. On the other hand, Vanilla PPO and DIAYN* generally failed to specialize in this way, resulting in an 'average' agent that attacked towers and chickens relatively equally, even though half of those attacks were not be able to do damage (because of the enforced specialization rule). In the cases of DIAYN* using a multiplicative model, and ADAP using a concatenation model, the generators learned specialized agents in only one of the three runs.

Note that in later experiments, we optimize the latent distribution of G to fit an ablated environment, but in this experiment, we present results without latent optimization. With that being said, it could be possible to further improve the performance of ADAP and DIAYN* by optimizing the latent distribution to remove 'bad apples' in the population: latents that map to less-optimal policies, or even possibly degenerate policies. Also notice that in Figure 3, we have excluded the results of DIAYN to better visualize the rest of the methods, because DIAYN was an outlier with respect to the rest of the methods. The best performing DIAYN settings scored a maximum reward of 16.25 ± 0.01 and had a specialization of 0.74 ± 0.4. Possibly because intrinsic reward was not normalized across batches, DIAYN learned degenerate solutions that attempted to find states that provided information about agent contexts, essentially wedging agents against borders of the map, or between chickens and towers, without actually interacting with the units themselves.

5 Experiment 2: Adaptation to Environmental Ablations

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, differences between policies can lend robustness to the policy space as a whole.

Experiment We aim to test how having a diverse policy space allows us to search in latent space for policies that better fit unexpected environmental ablations. Doing so would demonstrate the robustness of a population of policies, and simultaneously provide information about different types of diversity that are learned by G.

To this end, we train G on a normal Farmworld environment without the enforced specialization rule from Experiment 1. In this experiment, we use a Farmworld environment containing ten each of agents, towers, and chickens each, randomly initialized on a 10x10 grid.

We then *ablate* the environment, changing features such as map size and features, location of food sources, and even re-spawn times and food-yield of towers and chickens. Lastly, we deploy G into the ablated environment and *without* changing the weights ϕ, optimize the latent distribution for policies that are successful in the new environment, using the search algorithm described in the 2. Ablations and descriptions are available in Table 1.

Ablation	Description
`Far Corner`	The map expands in size, and food spawns in the bottom right. Agents spawn in the top left.
`Wall Barrier`	A 'rift' opens up between agents and their food. Agents must be able to navigate up and around the wall.
`Speedy`	2x2 map. Food health yield is set very low, so agents must be able to rapidly and consistently farm adjacent towers.
`Patient`	2x2 map. Food yield is very high, but food respawn speed is low. Agents must 'ration' their food so it lasts until the next respawn.
`Poison Chickens`	Agents spawn on the same map in which they were trained. However, chickens now yield -15 health, and towers yield 10 health.

Table 1: Farmworld Ablations

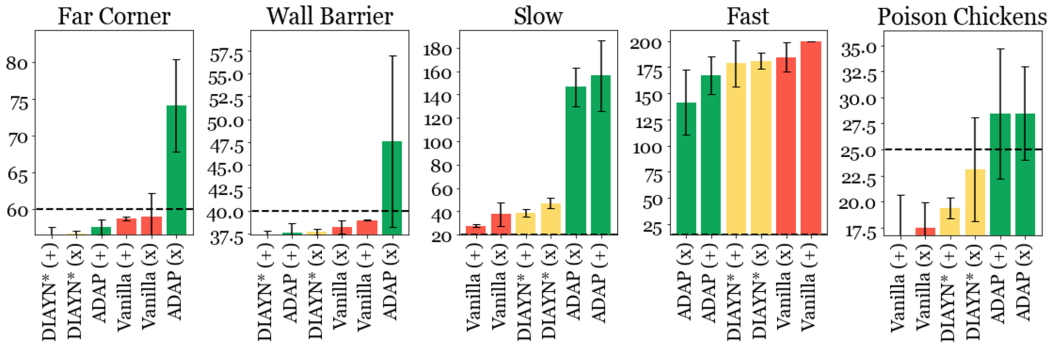

Figure 4: Lifetime After Optimization on Ablations: Plots are of agent health (y-axis) after latent optimization on each ablation, and we plot the agent's initial health as a dashed black line. Error bars reflect standard deviations over three random seeds. It is possible to go below initial health, since agents can do damage to each other or eat poisonous food.

Results Rather to our surprise, in each experiment trial, learning G using ADAP possessed a policy space Π containing 'species' that could thrive in nearly every environmental ablation! In the `Patient` ablation, we see that Π contained an agent that purposefully refrained from eating food until absolutely necessary. Perhaps equally surprising, Π contained an agent that was able to navigate completely around the wall of `Wall Barrier`, without ever explicitly training on an environment in which such behavior was required. The important thing to note is that all of these behaviors are *emergent from the train environment* – a product of optimizing for policy diversity while training G.

How is it possible that ADAP produced a policy space capable of adapting to nearly every ablation? The train environment was relatively abundant with resources scattered about a large map. Thus, there were many degrees-of-freedom in the rules of survival, and by optimizing for diversity, we found a policy space that filled these degrees-of-freedom while still yielding high reward. While these ablations reflect some of the possible degrees-of-diversity, there are certainly more. For example, an agent's direction of 'preference' does not have to be the bottom-right, as in the `Far Corner` ablation.

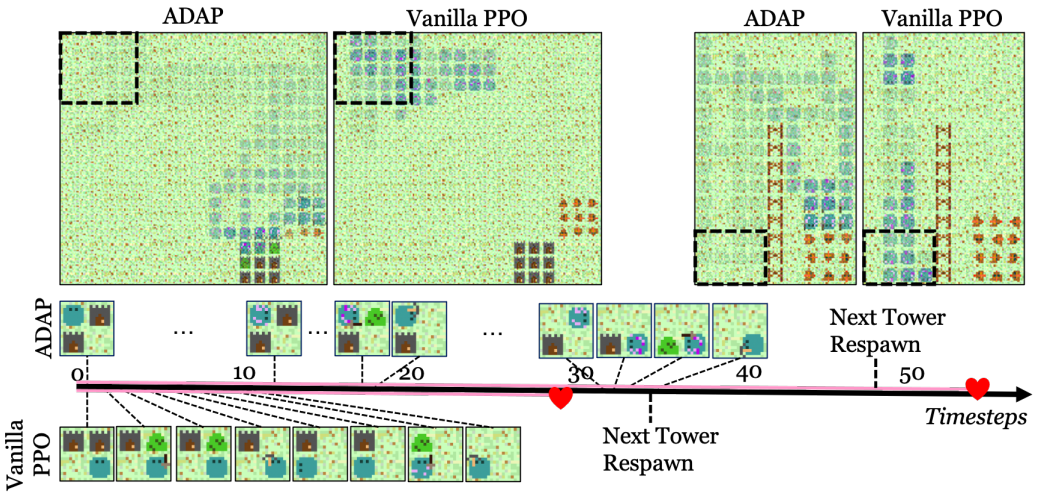

Figure 5: Adapting to Locomotion and Speed Ablations in Farmworld: Randomly selected visualizations from ADAP and Vanilla PPO, using multiplicative models. **Top Row:** `Far Corner` (left) and `Wall Barrier` (right) ablations. Agents are initialized in regions indicated by dotted black outlines, and images of the environment at different timesteps are overlaid to visualize behavior. Earlier timesteps appear more transparent. Episodes are visualized at timestep 60 and 40 for `Far Corner` and `Wall Barrier` respectively. **Bottom Row:** `Slow` ablation, with frames from each episode arranged along a timeline. Elements above and below timeline are related to latent-optimized ADAP and Vanilla PPO agents respectively. The location of the red heart indicates the point at which an agent's health will run out, *if* it does not get any additional food. Notice that the Vanilla PPO agent does not ration the two food towers, and its health will run out before the next tower respawn. Meanwhile, the ADAP agent waits until its health is very low (visualized by purple and pink speckles) before getting food, and is able to survive past the next tower respawn. Ellipses '...' indicate the ADAP agent is waiting during those frames.

Indeed, as a sanity check, we tested placing food locations in various other spots on an enlarged map, and found that for every cardinal location, there was a species of agent in Π that could exploit that new food location. What came as a surprise was that agents also used their health indicator as a degree-of-freedom in which to find diversity: since agents diversify conditional on state, species developed in which agents would prefer to go upwards when their health is high, but downwards when their health is low, thus allowing them to solve the wall barrier ablation. Similarly in the `Patient` ablation, ADAP learned a certain species of agent that waited until its health was low to farm a tower.

The `Poison Chicken` ablation was the one hold-out in which latent optimization could not find a profoundly successful species. While learning a unit preference (as in the niche specialization experiment) could be possible, it is likely that the *cost* of doing so in the train environment was too great. That is to say, in order to specialize, an agent would have had a very difficult time surviving the training environment.

Finally, we should note that the Vanilla baseline performed worse than ADAP in all ablations aside from the `Speedy` ablation. We hypothesize this is because a speedy-strategy is the best suited to quickly gaining high rewards in the train environment. Since Vanilla PPO optimized for solely for expected rewards, this is the only policy that it found. DIAYN* performed similarly to Vanilla PPO, and did not learn to speciate in a manner that was successful on the majority of ablations.

6 Experiment 3: Adaptation to Adversaries in Markov Soccer

Environment This experiment uses Markov Soccer, introduced in [11]. Two agents play on a gridworld and must 'carry' a ball into the opposing goal to score. Agents walk in cardinal directions or `stand` in place. Possession is randomly initialized, and switches if one an agent bumps into the other. The game is zero-sum, and on each timestep ends in a draw with ϵ probability.

Markov Soccer is an interesting environment, because the best policy for one agent depends on the policy of the other agent. As described in [11], 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 p_1 may be optimal against p_2, but not against p_3.

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.

Experiment Can we learn a population of individuals that is holistically strong against all types of opponents? 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 more exploitable than others. Secondly, we evaluate each *G* by playing ADAP (x), ADAP (+), Vanilla (x), and Vanilla (+) in a round-robin tournament against each other. For all evaluation methods, score is determined by `Wins - Losses` over the course of 1000 simulated games.

Against Hard-Coded Bots: Each bot always starts on the left side, and the learned policy starts on the right side (although the environment is coded such that observations are side-invariant). Bot types fall into three categories: offense (bots start with possession), defense (policy starts with possession), and mixed (random starting possession).

Bot	Bot Type	Bot Policy
`Straight`	Offense	Always moves `right`, towards the goal.
`Oscillate 0`	Defense	Oscillates in column 0, blocking both squares adjacent the goal.
`Oscillate 1`	Defense	Oscillates in column 1, leaving a gap in front of the goal.
`Stand`	Defense	`stands` in one square adjacent the goal, leaving the other open.
`Rule-Based`	Mixed	Follows hand-coded heuristics.
`Random`	Mixed	Follows a random policy.

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}.

Training and Baselines We use self-play to train both ADAP and baselines. As a baseline, we use Vanilla PPO with entropy regularization, using both concatenation and multiplicative model types. We found that using a high entropy regularization coefficient of 0.1 was necessary for training the Vanilla PPO baseline, to prevent it collapsing into degenerate solutions. ADAP was less dependent on entropy regularization, and we used a coefficent of 0.05.

Results As in the Farmworld adaptability experiment, we see that ADAP is able to learn a G during the *train* phase that emergently contains members that are successful against a variety of unexpected adversaries - including naive bots and other policies.

Compared to Vanilla PPO, the ADAP policy space generalizes better against the 'naive' adversaries `Random`, `Oscillate 0 & 1`, and `Straight`. Going back to the soccer team example, we were able to select individuals from the ADAP population that particularly exploited facets of these naive strategies. For example, against the `Oscillate 1` adversary, ADAP latent optimization found a member of the population that *side-stepped* the oscillating adversary simply by moving to the top row, and then down to the goal. Additionally, against the `Straight` adversary, successful ADAP individuals stole possession by deterministically standing in-front of the opponent to steal possession,

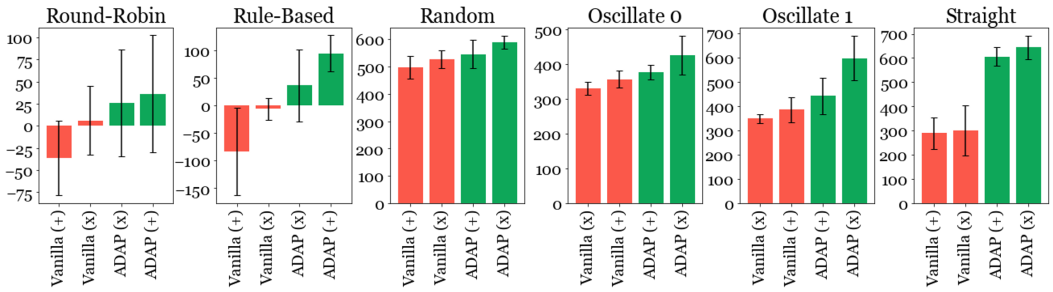

Figure 6: Results of round-robin and bot games. Score is over 1000 runs, some games prematurely end in a draw. We report standard deviation over three random seeds.

and then moving around and into the goal. On the other hand in both of these situations, Vanilla PPO could not find individuals that exploited the naive deterministic nature of their opponents. Instead, Vanilla PPO individuals played as they would against themselves: cautiously and defensively, which resulted in fewer wins against the 'naive' bots.

Using ADAP did not just allow us to optimize against naive opponents. ADAP learned the best G in the round-robin tournament, and was the only method that was able to consistently beat our rule-based bot. Overall, using ADAP allowed us to emergently find a policy space that exhibited strong performance against a gamut of opponents, in a manner that any single policy would not have been able to do (without knowing what type of opponent it is facing).

7 Limitations

Bad Apples When using ADAP, not every member of the policy space is going to be an optimal policy. In fact, some generated policies might be bad apples: policies that were incentivized by the diversity regularizer to take actions that were not rewarding. Naturally, some individuals might be better or worse than others. However, the bad apples may come with a plus side. Even though they do not perform well in the current environment, they might happen to perform well in a future environment!

Continuous-Action Space Environments The results presented so far focus entirely on environments with discrete categorical action spaces, in which we have observed that our diversity regularizer in Equation 1 empirically performs well. However, not all environments in RL use discrete action spaces - continuous action spaces are widely used in RL control tasks. While we believe that our regularizer can work in these environments, we did not rigorously test in these environments. We experiment in a toy environment of simple algebraic functions, included in the supplement.

8 Conclusion

We have presented a framework to learn 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. In 4, we show that ADAP is able to fit to distinct and mutually exclusive niches in the Farmworld environment. Given a highly multi-modal policy space, we explore potential useful properties of agent diversity in 5 and 6 via environmental ablations: ablating the actual Farmworld environment, and introducing new adversaries in Markov Soccer. Even though the ADAP generator was not trained to explicitly solve such ablations, it often turned out that we could find a subset of the ADAP policy space that nevertheless achieved high rewards.

References

[1] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel. Variational Option Discovery Algorithms. *arXiv e-prints*, art. arXiv:1807.10299, July 2018.

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv e-prints*, art. arXiv:1606.01540, June 2016.

[3] Y. Ding, C. Florensa, M. Phielipp, and P. Abbeel. Goal-conditioned Imitation Learning. *arXiv e-prints*, art. arXiv:1906.05838, June 2019.

[4] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is All You Need: Learning Skills without a Reward Function. *arXiv e-prints*, art. arXiv:1802.06070, Feb. 2018.

[5] C. Florensa, Y. Duan, and P. Abbeel. Stochastic Neural Networks for Hierarchical Reinforcement Learning. *arXiv e-prints*, art. arXiv:1704.03012, Apr. 2017.

[6] D. Ha, A. Dai, and Q. V. Le. HyperNetworks. *arXiv e-prints*, art. arXiv:1609.09106, Sept. 2016.

[7] Z.-W. Hong, T.-Y. Shann, S.-Y. Su, Y.-H. Chang, and C.-Y. Lee. Diversity-Driven Exploration Strategy for Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1802.04564, Feb. 2018.

[8] L. P. Kaelbling. Learning to achieve goals. In *IN PROC. OF IJCAI-93*, pages 1094–1098. Morgan Kaufmann, 1993.

[9] J. Lehman and K. Stanley. Evolving a diversity of creatures through novelty search and local competition. pages 211–218, 01 2011. doi: 10.1145/2001576.2001606.

[10] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In J. Dy and A. 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`.

[11] M. 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.

[12] A. Mahajan, T. Rashid, M. Samvelyan, and S. Whiteson. MAVEN: Multi-Agent Variational Exploration. *arXiv e-prints*, art. arXiv:1910.07483, Oct. 2019.

[13] M. A. Masood and F. 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.

[14] J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites. *arXiv e-prints*, art. arXiv:1504.04909, Apr. 2015.

[15] S. Nasiriany, V. H. Pong, S. Lin, and S. Levine. Planning with Goal-Conditioned Policies. *arXiv e-prints*, art. arXiv:1911.08453, Nov. 2019.

[16] J. Parker-Holder, A. Pacchiano, K. Choromanski, and S. Roberts. Effective Diversity in Population Based Reinforcement Learning. *arXiv e-prints*, art. arXiv:2002.00632, Feb. 2020.

[17] J. K. Pugh, L. B. Soros, P. A. Szerlip, and K. O. Stanley. Confronting the challenge of quality diversity. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, page 967–974, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450334723. doi: 10.1145/2739480.2754664. URL `https://doi.org/10.1145/2739480.2754664`.

[18] J. K. Pugh, L. B. Soros, and K. 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`.

[19] T. Rashid, M. Samvelyan, C. Schroeder de Witt, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *arXiv e-prints*, art. arXiv:1803.11485, Mar. 2018.

[20] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL `http://proceedings.mlr.press/v37/schaul15.html`.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv e-prints*, art. arXiv:1707.06347, July 2017.

[22] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. URL `http://nn.cs.utexas.edu/?stanley:ec02`.

[23] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks. *Artificial Life*, 15(2):185–212, 04 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202. URL `https://doi.org/10.1162/artl.2009.15.2.15202`.

[24] R. S. Sutton, D. Precup, and S. 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`.

[25] T. Wang, H. Dong, V. Lesser, and C. Zhang. ROMA: Multi-Agent Reinforcement Learning with Emergent Roles. *arXiv e-prints*, art. arXiv:2003.08039, Mar. 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We claim to have developed a method to generate a large population of policies via a generative model, and specify the environments in which we conduct our tests. We substantiate these claims through through experiments in Sections 4, 5, and 6, and describe limitations in Section 7.
   (b) Did you describe the limitations of your work? [Yes] See section 7
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Refer to Section 2, Model Architecture regarding increased computational cost, and the last paragraph, Considerations of Impact.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We do not include theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code and instructions is included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We address the choice of certain hyperparameters in the Section 7, as well as specify all training details and hyperparameters in the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We ran 3 trials for every experiment, and report standard deviation of the trials in all of our figures.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See the optimization portion of section 2.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use RLLib [10] under the Apache-2.0 license, and the OpenAI Gym [2] under the MIT License.
   (b) Did you mention the license of the assets? [Yes] Apache-2.0 and MIT License.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We developed the code for our new Farmworld Environment, released in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We are not using any personal data.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]