Learning Novel Policies For Tasks

Yunbo Zhang[1] Wenhao Yu[1] Greg Turk[1]

Abstract

In this work, we present a reinforcement learning algorithm that can find a variety of policies (*novel policies*) for a task that is given by a task reward function. Our method does this by creating a second reward function that recognizes previously seen state sequences and rewards those by novelty, which is measured using autoencoders that have been trained on state sequences from previously discovered policies. We present a two-objective update technique for policy gradient algorithms in which each update of the policy is a compromise between improving the task reward and improving the novelty reward. Using this method, we end up with a collection of policies that solves a given task as well as carrying out action sequences that are distinct from one another. We demonstrate this method on maze navigation tasks, a reaching task for a simulated robot arm, and a locomotion task for a hopper. We also demonstrate the effectiveness of our approach on deceptive tasks in which policy gradient methods often get stuck.

1. Introduction

Deep Reinforcement Learning (DRL) has shown great potential in solving problems in various domains such as game playing, maze navigation, and robotic control. Often times it is sufficient to find a single policy that solves the given task. In some cases, however, it may be desirable to find several different policies that solve the problem in different ways. If, for instance the goal is to produce a locomotion policy for a legged robot, there may be several ways in which the robot may coordinate its limbs, leading to various styles of walking. Similarly, there might be several ways in which a robot arm can reach a given target, and some of these reaching motions may prove more useful than others

in different situations.

The goal of our work is to provide reinforcement learning framework for finding not just a single policy for a given task, but instead find a variety of distinct policies, each of which also solves the task at hand. We will refer to each of these various policies for a given task as a *novel policy*. Our method of finding novel policies draws upon the recent success of policy gradient approaches to solving continuous control problems, such as PPO (Schulman et al., 2017), TRPO (Schulman et al., 2015), and DDPG (Lillicrap et al., 2015). A given policy gradient algorithm is guided towards producing a policy by a provided reward function, and the algorithm seeks a policy that maximizes the cumulative reward. If a particular policy that has been produced by a policy gradient algorithm is somehow insufficient for a given task, or if further variation is desired, there are typically two alternatives. One possibility is to run the algorithm again using a different random number seed, and hope that this will result in a different policy. The second option is to modify or augment the reward function so that the policy gradient algorithm is guided towards a different policy. Our method offers a different approach to finding novel policies that does not rely on chance or reward tuning.

We formulate the problem of finding novel policies as a multi-objective optimization problem. First, we create an autoencoder that recognizes the sequence of states that previous policies typically generate. We use this autoencoder to create a function that rewards novelty, that is, that rewards sequences of actions that are significantly different than those of the previous policies. We then solve a reinforcement learning problem in which there are two objectives: 1) solve the given task (guided by the task reward function), and 2) perform sequences of actions that are novel when compared to previous policies (guided by a novelty reward function). There are several ways in which a policy gradient algorithm may solve such a multi-objective problem. One way is to create a new reward term that is simply a weighted sum of the task reward and the novelty reward. Another possibility is to examine the policy gradient with respect to each of these reward functions, and update the policy in a way that attempts to follow both these gradients. We explore both these methods, and in particular we demonstrate that using an angle bisector between the task and novelty gradients is an effective approach across a variety of tasks.

---

[1]School of Interactive Computing, Georgia Institute of Technology, USA. Correspondence to: Yunbo Zhang <yzhang3027@gatech.edu>, Wenhao Yu <wenhaoyu@gatech.edu>, Greg Turk <turk@cc.gatech.edu>.

We demonstrate our approach on several problems, including a maze with multiple goals, a reacher task for a simulated robot arm, and a hopper. We also demonstrate the application of our approach to two deceptive problems (deceptive maze, deceptive reacher), for which most algorithms get stuck in a poor local minimum. Our approach finds policies that avoid the behaviour of earlier failed policies, thus making it easier to learn a policy that achieves the given task.

In this work we make the following contributions:

- We formulate the problem of *finding novel policies*, in which the goal is to sequentially produce policies that both solve a given task and that do so in a manner that is distinct from prior policies.

- We show how to measure the novelty of a given policy's behavior using the output of an autoencoder that examines the recent history of a policy. This measurement of novelty can then be used to find multiple novel policies for a given task.

- We present a method called the Task-Novelty Bisector (TNB) that produces novel policies that solve a given task. We demonstrate TNB when it is used together with PPO, but our method is applicable to any policy gradient method.

2. Related Work

Our method builds on top of the basic policy gradient approach for reinforcement learning (Sutton & Barto, 2018). There have been a number of recent policy gradient algorithms that have been used to create neural network policies for continuous control such as PPO (Schulman et al., 2017), TRPO (Schulman et al., 2015), DDPG (Lillicrap et al., 2015).

Our work has some relation to the problem of finding *options* (Sutton et al., 1999). An *option* is a course of actions that are determined by a policy. Usually options are used in a hierarchical setting, in which a high level policy selects from a variety of possible options over the course of solving a higher level problem. Some prior works have studied on option learning (Gregor et al., 2016; Bacon et al., 2017; Hausman et al., 2018). In particular, Gregor et al. (2016) maximizes the mutual information between options and the final states of a trajectory to encourage the discovery of different intrinsic options. Bacon et al. (2017) uses a hierarchical structure of policies in which options are created by learning a set of intra-option policies, termination functions, and an policy over options simultaneously. However, neither of these methods have been used to solve continuous robotic control problems.

Some other work on options, including DIAYN (Eysenbach et al., 2018) and VALOR (Achiam et al., 2018), address problems in robotics control. DIAYN (Eysenbach et al., 2018) encourages the diversity of a policy by maximizing the mutual information between options and states while minimizing the mutual information between options and actions conditioned on states. VALOR (Achiam et al., 2018) uses an LSTM variational autoencoder for option discovery where a universal policy conditioned on an option serves as the encoder, and a bidirectional LSTM serves as the decoder. Neither DIAYN nor VALOR use a task-relevant reward term, and they solely maximize behaviour discovery. Both methods aim to create a set of policies that can be used as initial policies for later task relevant training, often to be used as a low-level controller in a hierarchical setting. In the same spirit of encouraging diversity over an entire trajectory, work from the evolutionary algorithms community has applied Novelty Search (Lehman & Stanley, 2011a;b; Pugh et al., 2016; Mouret & Doncieux, 2009) to solve various problems.

Our work is also related to encouraging exploration in reinforcement learning. Some research uses the approach of curiosity-driven exploration (Schmidhuber, 1991; Sun et al., 2011; Conti et al., 2018) and applies these techniques to reinforcement learning (Houthooft et al., 2016; Hester & Stone, 2017). For example, to encourage exploration, VIME (Houthooft et al., 2016) uses variational inferences to derives an intrinsic reward objective that maximizes the information gain about the environment dynamics. Haarnoja et al. (2018a; 2017; 2018b) add an entropy term in the objective function so that exploration is encouraged through maximizing entropy for visited states. GEP-PG (Colas et al., 2018) tried to decouple exploration and exploitation by focus in one or the other in stages to encourage exploration. Although these methods encourage policy discovery with more exploration, they are not used to intentionally generate multiply distinct policies that solve a given task.

3. Preliminaries

3.1. Deep Reinforcement Learning

Deep Reinforcement Learning solves problems that are modeled as Markov Decision Processes (MDPs), defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \rho_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto S$ is the transition function, $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, ρ_0 is the initial state distribution, and $\gamma \in [0, 1]$ is a discount factor. The goal of RL is to find a policy $\pi_\theta : \mathcal{S} \mapsto \mathcal{A}$ parameterized by θ that maximizes the expected return:

$$J(\theta) = \mathbb{E}_{s_0, a_0, \ldots, s_T} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \right], \qquad (1)$$

where $s_0 \sim \rho_0(\cdot)$, $a_t = \pi_\theta(s_t)$ and $s_{t+1} = \mathcal{T}(s_t, a_t)$.

3.2. Policy Gradient Methods

Policy gradient methods are a class of RL algorithms that estimate the gradient of the expected return with respect to the policy parameters θ. The gradient estimation of the return $R(\theta)$ can be expressed as:

$$\frac{\partial J}{\partial \theta} = \mathbb{E}_{s_0, a_0, \ldots, s_T} \left[\sum_{t=0}^{T} \nabla \log \pi_\theta(a_t|s_t) Q(s_t, a_t) \right], \quad (2)$$

where $Q(s_t, a_t)$ is the Q function that measures the expected return of taking action a_t at s_t and following the policy π_θ thereafter.

4. Methods

In this work, we propose an algorithm that incrementally build a set of *novel policies* that solve a given task while exhibiting distinct behaviors to each other. At each iteration of our method, we train a new policy for which the learning agent is rewarded for both solving the task and demonstrating novel behaviors compared to the previously trained policies. We define novelty as dissimilarity between state sequences visited by the policies, and this is computed using autoencoders represented by neural networks. To achieve balanced learning progress between solving the task and seeking novel behavior, we propose a two-objective update method for policy gradient algorithms. After the policy is trained, it is expected to solve the task while also being novel in comparison to all of the previous policies. We then train an autoencoder for this newly generated policy to recognize behaviors similar to it and update the autoencoder set. This process is repeated until the desired amount of novel policies have been trained.

4.1. Measuring Novelty

Given a rollout generated by the current policy of interest, we want to measure how novel it is compared to rollouts generated from previously trained policies. To do this, we need to define a metric for measuring novelty between rollouts. One possible approach would be using nearest-neighbour based approch as in (Zhao & Saligrama, 2009; Liao & Vemuri, 2002) to measure the difference between the newly generated rollout and the rollouts from previous policies and aggregate them to obtain a novelty measurement. However, such methods can be computationally expensive as more rollouts are generated for comparison. Alternatively, one can use a count based novelty detection as as in (Bellemare et al., 2016; Tang et al., 2017; Strehl & Littman, 2005) to measure how frequently each state is visited by previous policies and compare that to the new rollout. However, this approach is difficult to scale to problems with high di-

mensional continuous state space such as the ones in our experiments.

To generalize across a variety of problems, using neural network for novelty detection as in (Hawkins et al., 2002; Richter & Roy, 2017; Ruff et al., 2018) becomes a more desired approach. In this work, we train autoencoders to measure the novelty of a rollout given a set of existing rollouts. The key idea is that if an autoencoder is trained on data from a particular distribution, it will be good at reconstructing data from that distribution, while it will perform poorly if the data is from a different distribution, i.e. when the data is novel as compared to the training data. To measure novelty of a rollout, we train one autoencoder for each trained policy in the sequence. There are two potential pitfalls if we were to use just a single autoencoder for all of the policies that have been found so far. First, if we used a single autoencoder, we would need to retain or regenerate the training rollouts for the earlier policies in the sequence when a new policy is trained, in order that the autoencoder does not forget the rollouts from previous policies. Second, when a single autoencoder is trained on data from multiple policies with distinct behaviors, it may generalize to data that has not been seen in the training data, potentially leading to under-estimated novelty for a novel rollout.

To fully capture the characteristics of the policy, it may be tempting to use the entire rollout as input to the autoencoder. However, such model only measures the novelty at the end of each rollout, leading to a sparse and delayed learning signal. In our method, we instead use a sub-sampled partial rollout as input to the autoencoder. Specifically, during the training of autoencoder, we divide each rollout into multiple fixed-length segments of length L and sub-sample them with a stride of G to obtain the training data, each of length L/G. In addition, we use only the states of each segment and ignore the actions, as we are more interested in state sequences that are novel. More details regarding autoencoder training can be found in the supplementary material.

Given a state sequence $\mathbf{s} = (s_t, s_{t+G}, s_{t+2G}, \ldots, s_{t+L})$ and a set of autoencoders $\mathbf{D} = \{\mathcal{D}_i\}$ trained for previous policies, we measure its novelty as:

$$r_{novel} = -exp\left(-w_{novel} \min_{\mathcal{D} \in \mathbf{D}} ||\mathcal{D}_\pi(\mathbf{s}) - \mathbf{s}||^2 \right), \quad (3)$$

where the exponential function bounds the range of the novelty reward and $w_{novel} > 0$ modulates the sensitivity of of the reward to the autoencoder reconstruction error (higher w_{novel} means more sensitive). During the training of a new policy, we use Equation 3 to compute the *novelty reward* for each step in the rollout. Note that for the first L steps we set the novelty reward to zero as there is insufficient data. Once step L has been reached, we can compute the novelty

reward, and we have found this to provide a sufficient signal for learning novel behaviors.

4.2. Two-Objective Optimization

Since we want to learn policies that solves the task and that behaves differently than previous policies, we want to optimize the two reward functions r_{task} and $r_{novelty}$ simultaneously. A straightforward way to do this would be to use a weighted average of the two reward functions as a single reward. However, this requires fine-tuning of the weights for blending the two reward functions.

In this work, we propose a two-objective update method to optimize both rewards at the same time, without the need to tune the relative weights. We will refer to our method as the *Task-Novelty Bisector* (TNB) approach. Using the two reward functions, we can formulate two objective functions $J_{task}(\theta)$ and $J_{novel}(\theta)$:

$$J_{task}(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r_{task}(s_t, a_t)\right]$$

$$J_{novel}(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r_{novel}(s_t, a_t)\right],$$

and use Equation 2 to estimate their gradient with respect to the policy parameters:

$$g_{task} = \frac{\partial J_{task}}{\partial \theta} \tag{4}$$

$$g_{novel} = \frac{\partial J_{novel}}{\partial \theta}. \tag{5}$$

When taking a weighted average of the two reward functions directly, it is similar to taking the weighted average of the corresponding gradients g_{task} and g_{novel}. However, when the two gradients are notably different in magnitude, this may result in updates that are biased toward one reward and thus requires weight tuning. We propose to instead update the policy in the direction of the angular bisector of the two gradients as illustrated in Figure 1 (a). For calculating the magnitude of the resulting update, we project both gradients onto the direction of the angular bisector and take the mean of the projected gradient magnitude. This results in an update direction that is independent of the scales of the two reward functions and is expected to improve the objective functions of both tasks. This scheme works well when the g_{task} and g_{novel} are pointing in the similar direction. However, when the gradients are pointing in opposite directions, i.e. $g_{task} \cdot g_{novel} < 0$, taking the angular bisector direction may result in an update that improves the objective functions little or not at all for either rewards. To address this issue, we propose a second component to our update

---

Algorithm 1 Task Novelty Policy Learning

1: **Input:** Autoencoders $\mathbf{D} = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_n\}$, Learning rate α
2: **Initialize:** Policy weights θ
3: **for** iteration $= 1, 2, \cdots$ **do**
4:     Collect trajectories τ using π_θ
5:     Assign rewards to steps in τ using r_{task} and r_{novel}
6:     **for each** epoch **do**
7:         Compute the gradient g_{final} using Algorithm 2
8:         $\theta = \theta + \alpha g_{final}$
9:     **end for**
10: **end for**

Algorithm 2 Task-Novelty Bisector Gradient

1: **Input:** Task policy gradient g_{task} and Novelty policy gradient g_{novel}
2: **if** $g_{task} \cdot g_{novel} > 0$ **then**
3:     $g_{finalDir} = \frac{bisector(g_{task}, g_{novel})}{\|bisector(g_{task}, g_{novel})\|}$
4:     $g_{final} = \left(\frac{g_{task}+g_{novel}}{2} \cdot g_{finalDir}\right) g_{finalDir}$
5: **else**
6:     $g_{final} = g_{task} - \frac{g_{task} \cdot g_{novel}}{\|g_{novel}\|} g_{novel}$
7: **end if**

---

scheme. When $g_{task} \cdot g_{novel} < 0$, we project g_{task} onto the hyperplane that is perpendicular to g_{novel} and use the projected vector as the final gradient, as shown in Figure 1 (b). The idea is that when the two gradients do not agree with each other, we want to prioritize solving the task over seeking novel behaviors. Algorithm 2 shows how we compute the final policy gradient g_{final} from the two gradients and Algorithm 1 describes how we use the proposed method for learning a new policy.
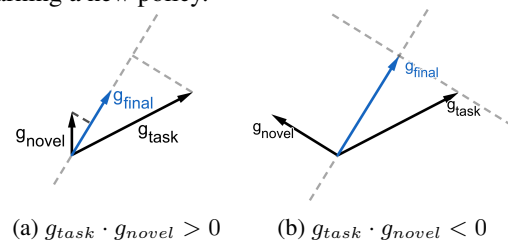

(a) $g_{task} \cdot g_{novel} > 0$ (b) $g_{task} \cdot g_{novel} < 0$

Figure 1. Final update gradient that improves both the novelty and task objectives.

5. Experiments

We use five environments to evaluate our method: 4-Way Maze, Reacher, Hopper, Deceptive Maze (D-Maze), and Deceptive Reacher (D-Reacher). The visualizations of each environment are shown in Figure 2. We provide videos of these environments at https://sites.google.com/view/learningnovelpolicy/home.
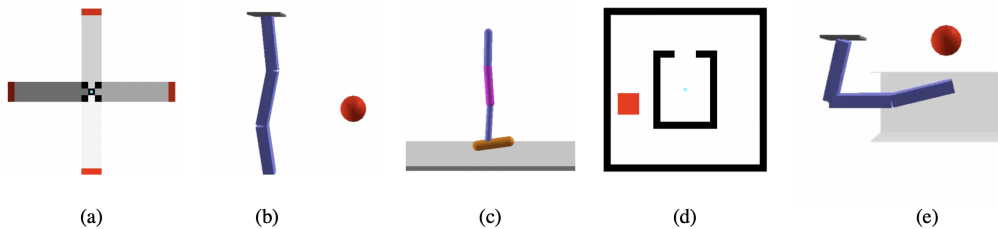
Figure 2. (a) - (e) are 4-Way Maze, Reacher, Hopper, D-Maze, and D-Reacher environments

5.1. Implementation Details

Our TNB method of two-objective update is built on top of the PPO (Schulman et al., 2017) implementation in OpenAI Baselines (Dhariwal et al., 2017). We implement both the 4-Way Maze and D-Maze environments in OpenAI Gym (Brockman et al., 2016). For the physics simulation of the Reacher, Hopper and Deceptive Reacher, we use the DART physics engine (Lee et al., 2018). Each rollout for each of the environments has a horizon of 500 control steps unless it triggers an early termination criterion.

For each newly trained policy, we used the final policy and a few policies before convergence to generate the data for the autoencoder. Details are given in the supplementary material.

5.2. Algorithm Comparisons

In the five experiments below, we compare the Task-Novelty Bisector (TNB) algorithm of Section 4 to four other methods: 1) regular PPO with different random seeds, 2) Weighted Sum of task and novelty rewards (WSR), and 3) a simplified Task-Novelty Bisector that always uses bisector, and does not perform projections when g_{task} and g_{novel} are pointing in opposite direction (TNB-NoProj). Specifically for the WSR method, we evaluated it on a set of weights and report the best performing one. The detailed results for different weights are given in the supplementary material. We compare to plain PPO with random seeds because, as discussed in Henderson et al. (2017), different random seeds can produce significantly different policies. In addition, for both deceptive problems and the 4-Way Maze problem, we compare our method with Soft-Actor-Critic (SAC) using different random seeds.

In this section, we use the term *trial* to denote k sequential runs of a given algorithm, i.e. one trial produces k distinct policies. Regardless of method, the first policy is created using regular PPO, without any novelty reward. For WSR, TNB, and TNB-NoProj, the second policy is trained using a novelty reward based on the first policy. The third policy is trained using a novelty reward from the first two policies, and so on. For these three algorithms, we use the same

Table 1. Experiments in 4-Way Maze measure the average number of path explored in each trial, and the number of trials that explore the paths in the order of total rewards.

METHODS	AVG # PATHS FOUND	# TRIALS IN ORDER
SAC	1	0/5
PPO	1.4	0/5
WSR	3.8	3/5
TNB-NOPROJ	**4**	3/5
TNB	**4**	**4/5**

random seed to initialize all of the k policies for a single trial. In this way, the amount of novelty in a trial cannot be attributed to a difference in random seeds. For a given PPO or SAC trial, we use different random seeds to initialize each of the k policies.

5.2.1. 4-WAY MAZE

Our first experiment is the 4-Way Maze, a simple 2D navigation environment for a point mass. The agent is placed in the center, and has easy access to the four arms of the maze that each lead to a goal. The observation space of the agent contains the position and velocity of the point mass, and the action space is the 2-dimensional force applied on the point mass.

For calculating the reward, the map is discretized in grid cells, and different color of the cell represents different reward when the agent steps on. Floors are gray-scaled cells and goals are red cells with various brightness. The magni-
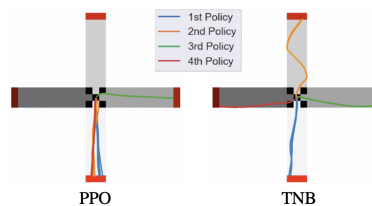

Figure 3. **4-Way Maze:** Trajectories from a trial of PPO (left) and TNB (right). The colors of the trajectories indicate where in the sequence the corresponding policy is from (first to last): blue, orange, green, red.

tude of reward signals for a cell depends on the brightness of the color on the cell. Brighter cells will provide more rewards, and darker cells will provide less rewards. The detail of the reward function design are described in the supplementary material. The agent receives a one-time small reward when stepping on any floor cells, and it receives a large reward when it steps on a goal cell, followed by the termination of the rollout.

In the case of the 4-Way Maze, we set $k = 4$ so that four policies are created per trial. We run five such trials using different random seeds in order to evaluate each algorithm, and thus create $5 \times 4 = 20$ policies for one algorithm. We evaluate the diversity of policies using two criteria: 1) The number of paths out of four that are explored in a trial, and 2) Whether the paths in a trial are discovered in descending order of rewards receiving on the path. The first criterion shows whether novel policies are found during training, and the second criterion evaluates whether each subsequent policy is the next best possible solution.

Figure 3 gives a visual comparison of PPO and TNB, showing several rollouts for each of the four policies that were created using PPO or TNB. For the trials shown, PPO with random seeds only explores two paths, whereas the TNB policies explore all four paths in order.

Table 1 shows the results of the 4-Way Maze trials. The values for average number of paths explored shows that both plain PPO and SAC had difficulty finding paths along multiple maze arms. Each of the three variations that use the novelty reward term was able to find all four paths in each trial. The rightmost column shows how many of the five trials found the four possible paths in increasing order of difficulty. Since PPO and SAC never produced all four paths, they scored zero out of five. In most of the trials, WSR, TNB-NoProj and TNB found all four paths in order, with TNB scoring the highest.

5.2.2. REACHER

To test our method on a continuous robotic control problem, we use a variant of the classic 3D-Reacher environment. In this problem, a 3-linked robot arm attempts to move its end effector to a fixed target. The reacher has total of 5-DOFs, one universal joint at the base, one revolute joint connecting the first and second links, and one universal joint connecting the second and the end effector. The observation space is a 26D vector consist of the position of the target, a vector pointing from the arm tip to the target, the sine and cosine of each DOF, and the angle q and angular velocity $\ddot{q}$ for all DOF's. The action space is a 5D vector representing the torque on each of the DOF. The reward function is designed as the distance from the end effector to the target minus the scaled sum of torques on each DOF. In addition, a rollout will be terminated, and a large reward will be added when
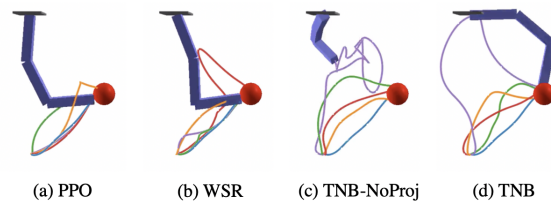

(a) PPO   (b) WSR   (c) TNB-NoProj   (d) TNB

Figure 4. **Reacher:** End effector trajectories from a trial for each of the four methods. Policies were trained in this order: blue, orange, green, red, purple.

the reacher touches the target.

For evaluating the reacher environment, we set $k = 5$ in each trial for each of PPO, WSR, TNB-NoProj and TNB. As shown in Figure 4, we can judge the novelty of policies visually. To produce this figure, we plot the trail of the reacher's end effector. For different random PPO trials, we see few variations on the trajectories, and in fact three of the policies end up finding nearly the same solution. WSR and TNB both generate policies that are novel solutions to the reaching task. For TNB-NoProj, on the other hand, some policies fail to solve the task of reaching the target (e.g. the purple curve in Figure 4 (c)). This is possibly due to the relative gradient directions between the task reward and the novelty reward being too different. Always using the bisector as gradient update (in TNB-NoProj) may have the effect of too little improvement in task performance, hence leading to a policy that fails to solve the task. See the provided video to view the reacher motions.

5.2.3. HOPPER

We show that our method is useful for complex locomotion control problems by evaluating our method on the hopper environment. The observation space is a 11D vector containing the linear position and velocity, and all joint angles and velocities. The action space is a 3D vector representing the torque exerted on the three joints.

We run five trials for each of the methods with $k = 5$, which gives us 25 policies. As a result, most policies trained with plain PPO generate trajectories that show very little variation. With novelty reward, on the other hand, the hopper is able to hop forward with variation of styles such as bending the torso backwards. Figure 5 shows three policies in a sequence trained using TNB, and all three policies are clearly distinct to one another. More results of different seeds are shown in the video.

5.2.4. DECEPTIVE MAZE

In Deceptive Maze environment (D-Maze), the same observations and actions are used as in the 4-Way Maze problem. In D-Maze, the agent is initially placed inside of an open box, and the goal is placed outside the box. The reward function penalizes the distance between the agent and the
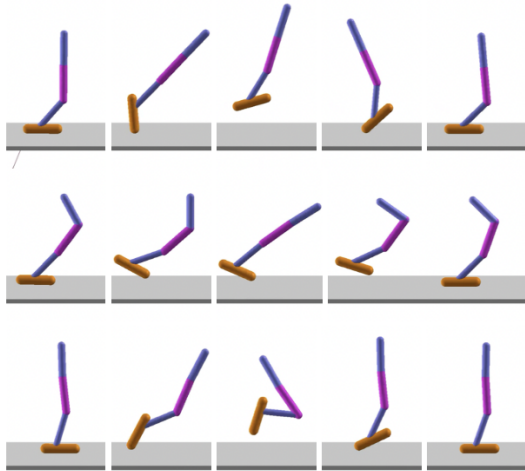
Figure 5. **Hopper:** The first three policies in a trial sequence when trained using TNB.

goal as well as penalizing the agent for being alive. An episode is terminated early and a large reward is added if the agent reaches the goal. If the agent follows the direction that minimizes distance penalty, the agent will run into a wall and will be blocked from the goal.

Policies for tasks with deceptive rewards have a clear measure of successes or failures. Thus, we quantitatively measure the performance of this method through two criteria: 1) The average number of successes of a trial, and 2) The number of trials that contain successful policies.

For D-Maze, we run five trials with $k = 4$ polices in sequence in each trial for each method. Each policy is trained over 3M samples, and that gives a sample budget of 12M for each trial. As shown in Table 3, when running PPO and SAC with different random seeds, very few or none of the policies succeed in driving the agent to the target. In most of the cases, the agent will run straight into the wall on the left shown as the first policy in Figure 6. When the novelty reward is applied in the training, we see a clear boost in the success rate. The number of policies succeeded are still low when just a weighted sum of two rewards is used (WSR). The poor performance of the weighted sum is possibly due to changes in relative importance of the two rewards for different policies in the sequence, thus making it difficult to
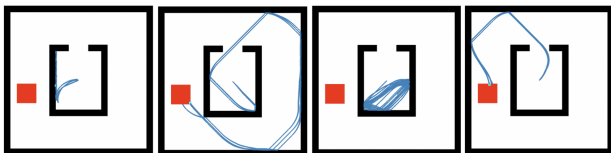
find a fixed weight that performs well. With both TNB and TNB-NoProj, on the other hand, both methods are able to give a high success rate and a number of successful policies.

Table 2. D-Maze, 12M Sample Budget

METHODS	AVG # SUCC POLICIES	# SUCC TRIALS
SAC	0	0/5
PPO	0.2	1/5
WSR	0.6	3/5
TNB-NOPROJ	**1.2**	3/5
TNB	**1.2**	**4/5**

A result of a successful trial using TNB is shown in Figure 6. We can see that the trails of the policies are significantly different from each other. Although the first policy fails by running straight towards goal, the second policy is encouraged to perform in a different manner and finds a way out the box to reach the goal. Though the the third policy fails, it explores a clearly different set of states from the first two. The fourth policy again finds a solution that is significantly different from the one found in the second policy.

5.2.5. DECEPTIVE REACHER

The Deceptive Reacher (D-Reacher) is a variant of the 3d-Reacher with obstacles and misleading rewards. The end effector of the reacher is initially placed inside a box, and the target position is placed outside the box. The D-Reacher shares the same rewards as the Reacher for the distance to goal and the torque consumption terms. In addition, the D-Reacher has an extra reward component for being alive, and it does not receive a final large reward when finishing the task. By strictly following the reward, the reacher will be trapped by the walls of the box.

We run 5 trials with $k = 5$ sequential policies in each trial. Each policy is trained over 6M samples, and that gives 30M sample budget to get five policies in the trial. The performance for D-Reacher is shown in Table 3. There is a clear boost of performance after using the novelty reward over plain PPO. There is also a larger number of successful policies in both methods that use gradient bisectors (TNB and TNB-NoProj) in contrast to weighted sums.

The visual quality of the policies from D-Reacher are shown by plotting the trail of the reacher's end effector. In Figure 7,

Figure 6. **Deceptive Maze:** From left to right: four policies sequentially trained using TNB.

Figure 7. **Deceptive Reacher:** From left to right: five policies sequentially trained using TNB. The first two policies fail, but they guide future policies 3, 4 and 5 to be successful.

we show five sequential policies from a trial of TNB. The first and second policies fail to move the end effector out of the box, but both explore different states. The remaining three policies all succeed in moving out of the box and solving the task. Moreover, they solve the tasks in three different ways.

Table 3. D-Reacher, 30M Sample Budget

METHODS	AVG # SUCC POLICIES	# SUCC TRIALS
SAC	0	0/5
PPO	0.2	1/5
WSR	1.2	**4/5**
TNB-NOPROJ	**2.6**	**4/5**
TNB	2.4	**4/5**

6. Discussion

The above experiments demonstrate using the Task-Novelty Bisector and the Weighted Sum of Rewards approaches to produce novel policies. Both of these methods proved to be more effective at producing novel policies than simply initializing the policy using different random seeds. Notably, however, the TNB approach successfully learned novel policies for all five tasks with very few changes for the parameters of the algorithm. The Weighted Sum of the task and novelty rewards required tuning the weights between these two terms in order to produce successful novel policies. When the weights were not tuned properly, the WSR approach often produced policies that could only achieve one objective over the other. For this reason, the Task-Novelty Bisector approach is the superior method.

It may be tempting to consider using option discovery approaches such as DIAYN (Eysenbach et al., 2018) or VALOR (Achiam et al., 2018) to learn novel policies. There is certainly a similarity between option discovery and finding novel policies, since in both cases the goal is to find a collection of policies that exhibit different sequences of states from one another. However, current option discovery approaches are not guided by a task reward function. The policies (options) that they produce do not necessarily satisfy a given task. For example, only some of the hopper policies produced by DIAYN move forward, and often produce policies that hop in place or even move backwards. In contrast, the Task-Novelty Bisector method produces hopper policies that always move forward due to the encouragement of forward motion from the task reward function.

It is worth reflecting on why the Task-Novelty Bisector approach is often able to find policies that solve a given task, despite the fact that it does not directly follow the task gradient vector. The key is that the TNB algorithm modifies the policy parameters using a vector that is still in an uphill direction with respect to the task objective function, just not in the *steepest* uphill direction. Even when the task and novelty gradients are pointing away from each other, TNB does not move downhill with respect to the task objective. Nevertheless, it is important to recognize that no policy gradient algorithm can be *guaranteed* to solve a given problem. Even state-of-the-art algorithms such as PPO and SAC can still get caught in local minima and fail to solve a given task. This can be seen in the results of the deceptive reacher and the deceptive maze problems. As described in the Experiments section, often PPO and SAC fail to find a policy that solves these deceptive tasks.

Limitations: Despite the success of TNB in the environments from the previous section, there are situations where TNB may fail to find suitable novel policies. One such situation is where two successful policies share a common behavior at the start of their rollouts. After the first of these policies is discovered, our novelty reward function will discourage the second policy from being discovered. An example of this would be a maze in which two different successful paths overlap near the start. It is possible that a larger L might be helpful with this limitation since this would give a larger window when determining the similarity between two segments. However, this could potentially give a longer delay to the novelty reward, making it more difficult for RL algorithms to pick up the reward signal.

Our approach to discovering novel policies is only suitable in the case where there are likely to be a small number of different successful policies. For a non-deceptive problem where there is just a single successful policy to be found (e.g. a maze with one correct path), our approach may not be helpful. At the other extreme, when there are numerous possible successful policies, it may not be necessary to encourage novelty and using different random seeds may be sufficient.

7. Conclusion

In this work, we have investigated the problem of finding novel policies in a reinforcement learning setting. We demonstrate that the Task-Novelty Bisector approach is an effective method of finding a collection of novel policies for a given task. Novel policies may prove useful in settings where different styles of motion are required, or when a task is deceptive. Because the autoencoder training for TNB is fast, this algorithm provides a compute-efficient alternative to PPO when such variations are desired.

8. Acknowledgements

References

Achiam, J., Edwards, H. A., Amodei, D., and Abbeel, P. Variational option discovery algorithms. CoRR, abs/1807.10299, 2018.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In AAAI, pp. 1726–1734, 2017.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In Advances in Neural Information Processing Systems, pp. 1471–1479, 2016.

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

Colas, C., Sigaud, O., and Oudeyer, P.-Y. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. arXiv preprint arXiv:1802.05054, 2018.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K., and Clune, J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In Advances in Neural Information Processing Systems, pp. 5027–5038, 2018.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. Openai baselines. GitHub, GitHub repository, 2017.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning diverse skills without a reward function. 2018.

Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. arXiv preprint arXiv:1611.07507, 2016.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. arXiv preprint arXiv:1702.08165, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. arXiv preprint arXiv:1801.01290, 2018a.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. arXiv preprint arXiv:1812.05905, 2018b.

Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. 2018.

Hawkins, S., He, H., Williams, G., and Baxter, R. Outlier detection using replicator neural networks. In International Conference on Data Warehousing and Knowledge Discovery, pp. 170–180. Springer, 2002.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560, 2017.

Hester, T. and Stone, P. Intrinsically motivated model learning for developing curious robots. Artificial Intelligence, 247:170–186, 2017.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In Advances in Neural Information Processing Systems, pp. 1109–1117, 2016.

Lee, J., Grey, M. X., Ha, S., Kunz, T., Jain, S., Ye, Y., Srinivasa, S. S., Stilman, M., and Liu, C. K. Dart: Dynamic animation and robotics toolkit. The Journal of Open Source Software, 3(22):500, 2018.

Lehman, J. and Stanley, K. O. Abandoning objectives: Evolution through the search for novelty alone. Evolutionary computation, 19(2):189–223, 2011a.

Lehman, J. and Stanley, K. O. Evolving a diversity of virtual creatures through novelty search and local competition. In Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 211–218. ACM, 2011b.

Liao, Y. and Vemuri, V. R. Use of k-nearest neighbor classifier for intrusion detection. Computers & security, 21(5): 439–448, 2002.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

Mouret, J.-B. and Doncieux, S. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In Evolutionary Computation, 2009. CEC'09. IEEE Congress on, pp. 1161–1168. IEEE, 2009.

Pugh, J. K., Soros, L. B., and Stanley, K. O. Quality diversity: A new frontier for evolutionary computation. Frontiers in Robotics and AI, 3:40, 2016.

Richter, C. and Roy, N. Safe visual navigation via deep learning and novelty detection. 2017.

Ruff, L., Görnitz, N., Deecke, L., Siddiqui, S. A., Vandermeulen, R., Binder, A., Müller, E., and Kloft, M. Deep one-class classification. In International Conference on Machine Learning, pp. 4390–4399, 2018.

Schmidhuber, J. Curious model-building control systems. In Neural Networks, 1991. 1991 IEEE International Joint Conference on, pp. 1458–1463. IEEE, 1991.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In International Conference on Machine Learning, pp. 1889–1897, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

Strehl, A. L. and Littman, M. L. A theoretical analysis of model-based interval estimation. In Proceedings of the 22nd international conference on Machine learning, pp. 856–863. ACM, 2005.

Sun, Y., Gomez, F., and Schmidhuber, J. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In International Conference on Artificial General Intelligence, pp. 41–51. Springer, 2011.

Sutton, R. S. and Barto, A. G. Reinforcement learning: An introduction. MIT press, 2018.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2): 181–211, 1999.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In Advances in neural information processing systems, pp. 2753–2762, 2017.

Zhao, M. and Saligrama, V. Anomaly detection with score functions based on nearest neighbor graphs. In Advances in neural information processing systems, pp. 2250–2258, 2009.