Diversity is All You Need: Learning Skills without a Reward Function

1st Matheus de Oliveira Pereira Paula
M2 Data Science and Artificial Intelligence
Université Côte d'Azur
Valbonne, France
matheusolipaula@gmail.com

2nd Aglind Reka
M2 Data Science and Artificial Intelligence
Université Côte d'Azur
Juan Les Pins, Antibes, France
aglindreka@live.com

***Abstract*—The paper "Diversity is All You Need: Learning Skills Without a Reward Function" [4] presents the "Diversity is All You Need" (DIAYN) approach, which enables the acquisition of useful skills in the absence of a reward function. This method employs an information-theoretic objective alongside a maximum entropy policy to foster the spontaneous emergence of diverse skills, such as walking and jumping, across various simulated robotic tasks. Through rigorous experimentation within reinforcement learning benchmark environments, the efficacy of the method is showcased in learning skills capable of task resolution without explicit reward signals in the original article. In our report, we further showcase the efficacy of the approach by running it on different reinforcement learning environments, and successfully obtaining at least one useful skill - and in other cases, many - as a result of the training process.**

I. Introduction

Deep reinforcement learning (RL) has proven effective in learning reward-driven skills across various domains [6]. However, creatures can learn useful skills without explicit supervision, facilitating rapid adaptation to new goals. Learning skills without reward has practical implications, particularly in environments with sparse rewards or where human feedback for evaluating rewards is costly [2]. Unsupervised skill discovery addresses challenges in exploration and reduces the supervision needed for task learning. The paper proposes an approach for autonomously discovering diverse skills in the absence of rewards by maximizing an information-theoretic objective with a maximum entropy policy [6]. This method enables the emergence of varied skills, such as running and jumping, across simulated robotic tasks. The learned skills are effective in solving benchmark tasks without receiving true task rewards and can be leveraged for hierarchical RL, task adaptation, and imitation learning. Key contributions include the formulation of the discriminability objective, demonstration of skill diversity, hierarchical RL applicability, task adaptation efficiency, and utility in imitation learning.

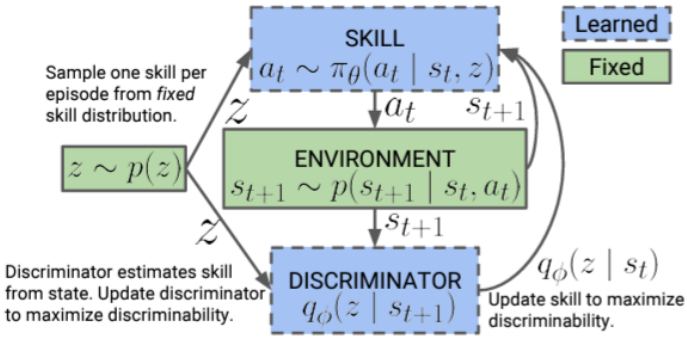

Fig. 1. DIAYN Algorithm : They update the discriminator to better predict the skill, and update the skill to visit diverse states that make it more discriminable. Image taken from [4]

II. How it Works

Algorithm 1: DIAYN Algorithm [4]

Result: Learned skills with DIAYN
while *not converged* **do**
  Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$;
  for $t \leftarrow 1$ **to** *steps_per_episode* **do**
    Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill;
    Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$;
    Compute $q_\phi(z \mid s_{t+1})$ with discriminator;
    Set skill reward
     $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$;
    Update policy (θ) to maximize r_t with SAC;
    Update discriminator (ϕ) with SGD;
  end
end

A. DIAYN: How it Works

DIAYN ("Diversity is All You Need"), is based on three key principles. Firstly, for skills to be valuable, they should influence the states the agent encounters. Each skill should explore distinct states to be distinguishable. Secondly, authors focus on states, not actions, to differentiate skills, as actions may not always visibly alter the environment. Lastly, they promote exploration and diversity by encouraging skills to act as randomly as possible.

The authors formulate their objective by using information theory notation. They maximize the mutual information between skills (Z) and states (S) to ensure skills control the visited states ($I(S;Z)$):

$$\begin{aligned} F(\theta) &= I(A|S,Z) - H(Z|S) + H(Z) \\ &= H(A|S,Z) + E_{z\sim p(z),s\sim\pi(z)}[\log p(z|s)] \\ &- E_{z\sim p(z)}[\log p(z)] \geq H(A|S,Z) \\ &+ E_{z\sim p(z),s\sim\pi(z)}[\log q_\phi(z|s) - \log p(z)] \end{aligned}$$

As exact computation of the posterior distribution is unfeasible, they approximate it with a learned discriminator. Utilizing Jensen's Inequality, they derive a variational lower bound on the objective [1], enabling practical optimization.

In summary, their approach seeks to autonomously discover diverse skills by maximizing the influence of skills on states, minimizing the predictability of skills from actions, and promoting randomness in skill actions, all within a principled information-theoretic framework.

B. Implementation

The authors implement DIAYN using the soft actor critic (SAC) algorithm [5], learning a policy $\pi_\theta(a|s,z)$ conditioned on the latent variable z. SAC maximizes the policy's entropy over actions, addressing the entropy term in our objective G. Following [5], they scale the entropy regularizer $H[a|s,z]$ by α. Empirically, they found that $\alpha = 0.1$ provided a balanced trade-off between exploration and discriminability.

To maximize the expectation in G, they replace the task reward with the following pseudo-reward:

$$r_z(s,a) \triangleq \log q_\phi(z|s) - \log p(z)$$

The authors employ a categorical distribution for $p(z)$. During unsupervised learning, they sample a skill $z \sim p(z)$ at the start of each episode and act accordingly throughout the episode. The agent is rewarded for visiting states that are easy to discriminate, while the discriminator is updated to better infer the skill z from visited states. Entropy regularization occurs as part of the SAC update.

Regarding stability, unlike prior adversarial unsupervised RL methods [7], DIAYN forms a cooperative game, avoiding many instabilities of adversarial saddle-point formulations. On grid worlds, they can analytically compute that the unique optimum to the DIAYN optimization problem is to evenly partition the states between skills, with each skill assuming a uniform stationary distribution over its partition..

In the continuous and approximate setting, convergence guarantees would be desirable. However, this is challenging; even standard RL methods with function approximation lack convergence guarantees, yet remain useful. Empirically, they find DIAYN to be robust across random seeds; varying the seed minimally impacts the learned skills and has little effect on downstream tasks.

III. Experiments

In order to simulate the idea of training an agent to learn skills without a reward function presented in the paper, we utilized a previously existing implementation of the paper's agent training process on PyTorch, which can be found in GitHub [1]. Unfortunately, due to the repository not having been updated in a few years, an incompatibility between the used PyTorch and CUDA versions and the GPU of our machine happened and we were unable to run the project's code as is.

In order to solve this, we forked the code to our own GitHub repository [2], and slightly changed the requirements file of the project in order to use PyTorch 1.13 with CUDA 11.1 support, which is enough to support a NVidia RTX 3070Ti mobile GPU, and did a few tweaks to the project's code to solve other incompatibilities that showed up afterwards. We also added a few missing packages to the project's requirements that were not present in the original repository's requirements.

In total, we carried out experiments in three environments made available present in OpenAI's Gym package. Those environments are *Hopper*, *BipedalWalker* and *MountainCar-Continuous*. We believe that by testing the approach's efficacy in making agents learn useful skills in is the best and most practical way of showing how successful it is. In the next three subsections, each of the environments are described alongside their characteristics, such as their action space. We also discuss the usual goal of the environment - in terms of the state that leads to a reward in more classical reinforcement learning approaches - and how we adapt it to a "useful skill" in our analysis of the learned skills so we can discern which skills are useful or not and whether the agent learned an useful skill or only non-useful ones.

A. Hopper

The *Hopper* environment is based on the paper *Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts* [3]. The hopper agent is a two-dimensional one-legged figure with four body parts: the torso at the top, the thigh in the middle, and a single foot upon which the entire body rests. You can see a visual representation of the hopper agent and its environment in fig. 2.

Another characteristic here is the presence of a three-dimensional action space, based on the torque applied on the links between each body part - the joints - which we name *rotors*, similarly to the nomenclature in robotics. The torque can be applied on the thigh rotor, leg rotor or foot rotor, and its measurement is given in continuous values between -1 and 1.

The goal in this environment, when using classical reinforcement learning, is to teach the hopper agent to jump to the correct direction (which can be either left or right) by setting a goal and rewards to reinforce this behavior. This carries over to the task here: an useful skill to be learned for the agent is to jump to either the left or the right, and therefore, this

[1] https://github.com/alirezakazemipour/DIAYN-PyTorch
[2] https://github.com/MatheusOPP/DIAYN-PyTorch

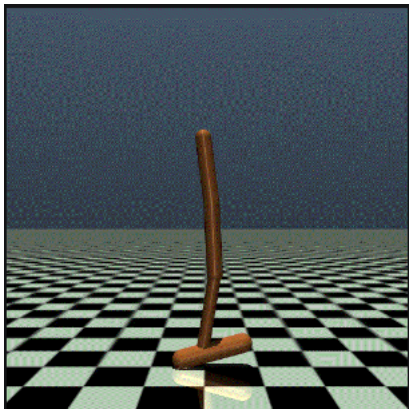

Fig. 2. Visual representation of the Hopper agent in MuJoCo

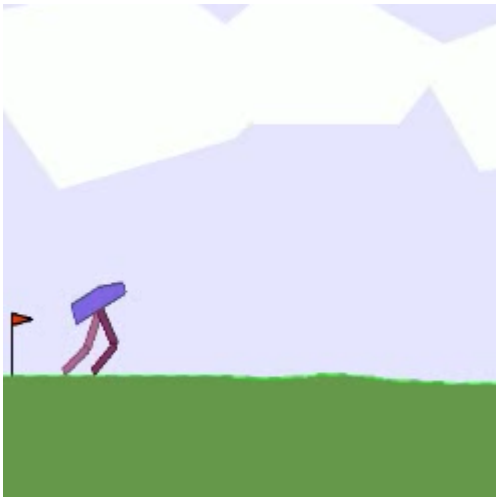

Fig. 3. Visual representation of the BipedalWalker environment in Box2D

behavior should show itself in the skills learned by the agent if we are to consider this approach successful.

We therefore train our agent: we set the number of skills at 20, the learning rate at 3×10^{-4} and train our agent for about 1000 episodes before stopping. We then generate visual representations of the agent's learned behavior as video files of the AVI format, one for each skill. We check each one of them to verify if our agent learned an useful skill by checking if the agent's behavior is similar to the environment's goal, which can be characterized by it jumping to the left or right.

The approach is successful here: the agent learns how to jump to the right in skills 7 and 8, and how to jump to left in skill 9. Skills 7 and 8 also exhibit different behavior: skill 7 is characterized by slow but strong jumps to the right, while skill 8 is characterized by quick, successive jumps that only move slightly to the right. There are also many skills that are not very useful and mostly consist of the agent applying torques to its joints in different ways but not jumping and/or mostly staying still, but this does not matter as the very idea of the *Diversity is All You Need* approach is that we will obtain useful skills by learning a diverse set of skills; therefore a set of emergent behaviors that are not useful is expected.

B. BipedalWalker

The *BipedalWalker* environment is a simple 4-joint walker robot environment, part of the Box2D environments present in OpenAI's Gym package. This environment utilises the Box2D physics engine alongside PyGame for rendering to create a "walker" game, where we have an agent consisting of a hull and two legs, whose goal is to walk to the right as much as possible, being rewarded more points the further it walks. It should be noted that this game is actually implemented in two difficulties in the library: a "normal" one with just slightly uneven terrain, and a "hardcore" one with ladders, stumps and pitfalls. For the sake of this simulation, we used the the "normal" difficulty. We illustrate the BipedalWalker agent and its environment in fig. 3.

The action space here is four-dimensional, consisting of the motor speed values that can be applied to each of the four joints of the walker (which are equivalent to its groin and its knees in human anatomy). When using reinforcement learning through usual methods, the agent is given a reward for "moving forward", and the simulation is ended when it either reaches the far end or its hull touches the ground. In our approach, as no reward is given to the agent for moving forward, we consider that a desirable skill to learn would be to move forward continuously.

We again set out to train our agent: we set the number of skills at 50 this time, the learning rate at 3×10^{-4} and train our agent for about 1000 episodes before stopping. We then generate visual representations of the agent's learned behavior as video files, with one file relating to one learned skill. We check each video verify if our agent learned an useful skill, which here is identified by a continuous movement to the right side.

We have moderate success in this task: the agent learns exactly one skill - skill 11 - that is able to walk continuously to the right, although it does not seem to reach far right end in the maximum time of the simulation (32 seconds). The other skills are generally unsuccessful at completing the task, however: in some the agent gets stuck at awkward positions, and in others, the agent ends up backtracking and falling to the ground or on the pitfall before the starting line, or even tripping while moving forward and hitting the ground with the hull. Nevertheless, one of the skills was similar to the original environment's goal, and therefore the training of the agent for this environment can also be considered a success.

C. MountainCarContinuous

Lastly, we test our approach in the MountainCarContinuous environment. This environment, once again present in OpenAI's Gym package, is based on a Markov Decision Process, consisting of a car agent placed at the bottom of a sinusoidal valley. The action space here is unidimensional: the only actions that we can take is accelerating the car in either direction - left or right - with continuous values between -1 and 1. In fig. 4 we once again illustrate the environment.

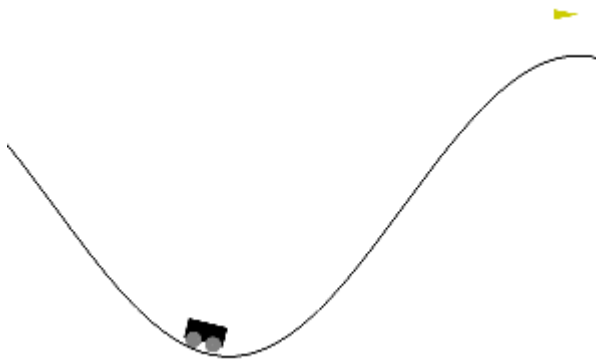

Fig. 4. Visual representation of the MountainCarContinuous environment

The goal in this environment is to accelerate the car strategically to reach the goal state on the top of the right hill. In a typical RL scenario, we would reward the agent for reaching that goal state. Here, as we do not use a reward function, we consider the training to be a success if the agent learns at least one skill that reaches said goal state. With this in mind, we once again train our agent, setting the number of skills to 20 and the learning rate at 3×10^{-4}, for a total of 1000 episodes before stopping. We once more generate a visual rendering of the the agent's learned behavior in the form of video files for each learned skill.

We once again obtain success in the training of the agent: one skill is able to reach strategically accelerated to reach the goal through the learning of diverse skills - in this case, skill 3. Many other skills, such as skill 14, also learn how to strategically accelerate just enough to almost reach the goal, but don't reach it. Other skills have the car get stuck at the bottom of the valley due to not accelerating enough or taking advantage of momentum, and are not useful. But as we are able to extract an useful skill from this approach, we can ascertain that the learning of diverse skills is once again able to successfully teach the agent to reach its goal even without a goal and a reward function set in the environment.

IV. Conclusion

In this report, we were able to summarize the contents of the article "Diversity is All You Need: Learning Skills without a Reward Function" [4], as well as run experiments using this new reinforcement learning approach introduced by it successfully.

As evidenced by the three experiments we carried out, this approach can be replicated under different environments, attesting to its efficacy as a reinforcement learning approach. In some cases, such as in the case of the *Hopper* environment, the diversity of the learned skills is in fact enough to produce more than one useful skill, with different and sometimes even inverted goals. In other cases, we obtain a much bigger number of non-useful skills, but are nonetheless able to obtain at least one useful skill that reaches the goal of the environment despite the lack of a reward function.

The concept of learning skills without supervision and a reward function leads to a number of interesting repercussion for reinforcement learning. For one, human feedback can be mostly or completely omitted from the learning process in the cases where this is applicable. There would be less or no need to rely on human supervision in certain reinforcement learning contexts where it is key. One such case is imitation learning: there is no more need to record games played by a human to use as a basis to train an agent. You could instead have the agent learn diverse skills, which act as policies, and use any useful skills either as the model's policy, or as a starting checkpoint for another reinforcement learning approach (which would work as some kind of pre-training).

Nonetheless, during our experiments, it was possible to also verify some drawbacks and limitations of this approach. In order to effectively train the agent to obtain useful skills, a considerable number of training episodes is needed, and in many cases, this training process can take a considerable amount of time - in our case, training each of the models for about 1000 iterations took between six and eight hours in a setup with an Intel i7-12700H CPU and a 3070Ti mobile GPU and 32GB of RAM. Moreover, as there is no guarantee of convergence to the goal as we, there is also no guarantee that any of the learned skills will be useful. Moreover, as our experiments were not extensive and did not cover all kinds of environments available reinforcement learning - which could be potentially infinite - it is entirely possible that this approach could fail in more complex environments where the diversity of skills is unable to make up for the lack of a proper reward function.

Nevertheless, this approach has proven to be effective under certain reinforcement learning environments and it has potential to be used in different applications. In particular, its applications in hierarchical reinforcement learning are a smaller focus of the original article itself and could be of interest to future researchers of this topic.

References

[1] David Barber and Felix Agakov. The im algorithm: a variational approach to information maximization. *Advances in neural information processing systems*, 16(320):201, 2004.

[2] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

[3] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. 06 2011.

[4] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

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

[7] Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.