Reinforcement Learning : Final Project
Diversity is All You Need

Rémy Deshayes and Olivier Dulcy

May 11th 2021

In the Reinforcement Learning (RL) paradigm one is interested in the study of agents and how they learn by trial and error. The fundamental idea is that rewarding or punishing an agent for its behavior makes it more likely to repeat it or discontinue it thereafter.

Section 1 introduces the standard RL problem and spotlights some of its associated issues before introducing a workaround to the most pressing one. Section 2 delves into the workaround and explore a corresponding method that we later implement[1], namely *Diversity Is All You Need* (DIAYN) [1]. Finally, section 3 briefly lay out where DIAYN stands in the RL literature before wrapping up the project with a few conclusive remarks.

1 Introduction

1.1 The standard RL problem: reward and expected return

Let us further formalize this fundamental idea of reward and punishment. To that end, we introduce the reward function R which is based on the current state of the world s_t, the action just taken a_t, and the next state of the world s_{t+1}.

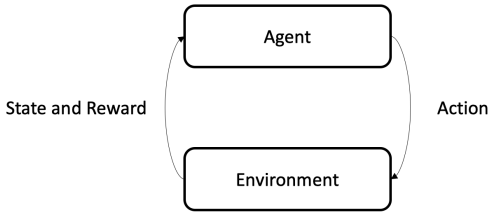


Figure 1: Interaction between the agent and the environment based on [2]

Then, leveraging this reward function, the agent aims at maximizing a return, a form of cumulative reward over a sequence of states and actions in the world called a trajectory - for instance the sum of all yielded rewards. Now, eventually, the RL goal is to select an agent's policy which maximizes the expected return.

Let us formalize those remarks in order to get the RL optimization program when both the environment transitions and the policy are stochastic. This will be useful in later sections and notably in 1.2 and 2.3.

[1]Our GitHub repository can be found here: `https://github.com/remydeshayes/RL_DIAYN.git`

To get our expected return objective function for the RL optimization program, we need to introduce the probability of a T-step trajectory τ which is given by :

$$P(\tau \mid \pi) = \rho_0 (s_0) \prod_{t=0}^{T-1} P (s_{t+1} \mid s_t, a_t) \pi (a_t \mid s_t)$$

where ρ_0 is the start-state distribution and π the policy.
Hence, we get the expected return :

$$J(\pi) = \int_\tau P(\tau \mid \pi) R(\tau) = \underset{\tau \sim \pi}{\mathrm{E}} [R(\tau)]$$

Noting that, when the return is the sum of rewards we have :

$$R(\tau) = \sum_t R(s_t, a_t, s_{t+1})$$

The RL optimization program

$$\pi^* = \arg\max_\pi J(\pi) \qquad \text{(RL)}$$

where $J(\pi)$ is the expected return and π the policy

1.2 Intuition: learning without a reward function

Now, RL has known a wealth of successes with major hits in finance, industry and even gaming with famous implementations such as AlphaGo, outmatching humans in a discipline long-believed to be too complex for a computer program [3]. However, even though RL agents often have the appearance of humans - after all they can drive, walk and even talk! - they generally lack the very human ability to autonomously learn skills by exploring his environment and ultimately to wisely use those learnt skills when faced with an ulterior goal-oriented task.

As a matter of fact, agents are most often designed and taught to perform complex behaviors using task-specific *extrinsic* reward functions, as laid out in 1.1, which can sometimes be limiting. Indeed, as we introduced earlier, it prevents them from having this sort of human ability to autonomously learn skills but also because carefully designing a reward functions is complex and often requires significant effort let it be regarding equations but also material and time related logistics - this can become unreasonable for a large number of tasks.

In this context, the idea of introducing unsupervised RL agents surfaced. The intuition behind it is that the agent now uses an *intrinsic* reward function - such as trying different things in the environment - to generate its own training signals to acquire a broad set of task-agnostic behaviors.
Accordingly, the unsupervised RL paradigm has the desirable property to avoid the complex reward design task while giving the agents the initial human autonomy ability we discussed earlier.

2 Diversity is All You Need - [1]

With the principles we introduced in part 1.2 in mind, we now explore an unsupervised RL framework laid out by Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz and Sergey Levine in [1].

2.1 Meeting the unsupervised RL paradigm challenge

The challenge of such unsupervised method is to yield a great variety of distinct and useful skills that altogether explore a substantial share of the state space.

To meet this challenge and yield useful skills, paper [1]'s authors assume that the skills should be trained in order to maximize the exploration of the possible behaviors' set.

In that regard, their proposed method relies on three main principles :

1. The skills must be distinguishable i.e different skills visit different states
2. States - and not actions - are used in order to distinguish skills, this is important because some actions does not affect the environment and thus are not visible to the observer
3. Finally, as distinguishable does not mean diverse, the method should allow to learn skill that are as random as possible in order to ensure this diversity need

Using the notations introduced in section 1 i.e S is the state, A is the action space and a skill is a policy π conditioned on a latent variable $Z \sim p(z)$, we now explore how the authors succeeded in enforcing those conditions through maximizing the following information theoretic objective with a maximum entropy policy:

Program objective

$$\mathcal{F}(\theta) \triangleq I(S;Z) + \mathcal{H}[A \mid S] - I(A;Z \mid S) \tag{Obj}$$

where $I(\cdot,\cdot)$ is the mutual information and $\mathcal{H}(\cdot)$ the Shannon entropy.

The mutual information quantifies the amount of information obtained about one random variable through observing another random variable. Therefore, the first term $I(S;Z)$ enforces the first principle by indicating that the skill can be inferred from the states visited. Similarly, by subtracting the mutual information between skills and actions given the state, the third term ensures that the second principle is met.
Finally, the entropy of a random variable being the average level of uncertainty inherent in the variable's possible outcomes, the second term is the amount of uncertainty that remains in the action after the state is known thus ensuring that the third condition is met - $p(z)$ is seen as a mixtures of policy, thus maximizing the entropy of this mixture policy.

Acknowledging that the mutual information is generally intractable [4], we recall the following result:

$$I(X;Y) = \mathcal{H}(Y) - \mathcal{H}(Y|X)$$

where X and Y are 2 random variables.

This result yields a new objective form that the authors will leverage in their optimization procedure :

$$\mathcal{F}(\theta) = \mathcal{H}[Z] - \mathcal{H}[Z \mid S] + \mathcal{H}[A \mid S, Z] \tag{Obj V2}$$

Again, second term ensures that the skill can be inferred from the current state, third term ensures that each skill act as randomly as possible.

In the process, we recall another formula:

$$\mathcal{H}(X) = -\sum_i p\left(x_i\right) \log p\left(x_i\right)$$

Using this formula in (Obj V2), this gives us:

$$\mathcal{F}(\theta) = \mathcal{H}[A \mid S, Z] + \mathbb{E}_{z\sim p(z), s\sim\pi(z)}[\log p(z \mid s)] - \mathbb{E}_{z\sim p(z)}[\log p(z)] \tag{Obj V3}$$

We note that $p(z \mid s)$ is not readily available and that we have to find an alternative way to retrieve it. Now, this (Obj V3) encourages us to build on from the work of D. Barber and F. Agakov in paper [4].
Since the mutual information is a measure of information transmission, the aim is to maximise a lower bound on the mutual information.

Once again, we use the formulation of the mutual information we used earlier to obtain (Obj V2) :

$$I(S; Z) = \mathcal{H}(Z) - \mathcal{H}(Z|S)$$

In general, one wants to optimize the mutual information with respect to the parameters $p(z \mid s)$ from (Obj V3), $p(z)$ is simply the fixed distribution. One needs to bound $H(Z \mid S)$ suitably.
Recalling the the Kullback-Leibler bound in its general form $\sum_{\mathbf{x}} p(\mathbf{x} \mid \mathbf{y}) \log p(\mathbf{x} \mid \mathbf{y}) - p(\mathbf{x} \mid \mathbf{y}) \log q(\mathbf{x} \mid \mathbf{y}) \geq 0$ and applying it to our problem, yields :

$$I(S, Z) \geq H(Z) + \mathbb{E}_{z \sim p(z), s \sim \pi(z)}[\log q_\phi(z \mid s)]$$

where $q_\phi(z \mid s)$ is an arbitrary variational distribution, chosen to be a learnt discriminator in [1].

Finally, the Jensen inequality gives a variational lower bound $\mathcal{G}$ on the objective $\mathcal{F}$

Variational Lower Bound $\mathcal{G}$

$$\mathcal{F}(\theta) \geq \mathcal{H}[A \mid S, Z] + \mathbb{E}_{z \sim p(z), s \sim \pi(z)} \left[\log q_\phi(z \mid s) - \log p(z)\right] \triangleq \mathcal{G}(\theta, \phi) \qquad \text{(VLB)}$$

2.2 DIAYN algorithm

Now to learn (VLB), the idea behind DIAYN's implementation is a two-stage cooperative game.
First, an agent experiments in an environment during *steps_per_episode*. During this experiment, the actions undertaken by the agent are conditioned by a skill, sampled before the experiment. Then, the probability of the sampled skill given by the state is approximated by the discriminator q_ϕ. This is one crucial component of DIAYN : the discriminator attempts to tell skills apart. The *intrinsic* reward is then computed using the probability given by the discriminator.
In a nutshell, the agent is rewarded for visiting states that are easy to discriminate and the discriminator is updated to better infer the skill z from states visited.

Let us formalize those ideas by having a look at the pseudo-code below :

Algorithm 1: Diversity is All You Need (DIAYN) - as given in [1]

Input: fixed skill distribution $\mathbb{P}_{\text{skill}}$ (denoted $p(z)$ in [1]), an environment for the agent
Initialize discriminator weights ϕ
while *not converged* **do**
  Sample skill $z \sim \mathbb{P}_{\text{skill}}$ and initial state $s_0 \sim \rho_0(s)$
  for $1 \leq t \leq$ *steps_per_episode* **do**
    Sample action $a_t \sim \pi_\theta(a_t|s_t, z)$ from skill.
    Step environment : $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$.
    Compute $q_\phi(z|s_{t+1})$ with discriminator.
    Set skill reward $r_t = \log q_\phi(z|s_{t+1}) - \log p(z)$
    Update policy (θ) to maximize r_t with SAC.
    Update discriminator (ϕ) with SGD.

Two things stand out from this pseudo-code:

- Line 8, we note that the expectation term in (VLB) is maximized by replacing the task reward by a pseudo-reward $\log q_\phi(z|s_{t+1}) - \log p(z)$
- Line 9, the policy $\pi_\theta(a_t|s_t, z)$ is learnt with soft actor-critic (SAC) [5] - as we will see in part 2.3, SAC maximizes the policy's entropy over actions which corresponds to the first term in $\mathcal{G}$ (VLB)

2.3 Soft Actor-Critic - [5]

The Soft Actor-Critic (SAC) is an off-policy actor-critic - in a nutshell, actor-critic amounts to a combination of policy-based (actor) and value-based (critic) approaches - deep RL algorithm based on the maximum entropy framework.
Indeed, a crucial feature of SAC is the entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy. This very much recalls the famous exploration-exploitation trade-off. Here increasing entropy produces a more thorough exploration - this can be linked to various comment we made about DIAYN principles in part 2.1 for instance.

In short, SAC builds off of the regular RL optimization program (RL) introduced in 1.1 but adds an entropy term to (RL) yielding the following augmented objective :

Maximum Entropy RL objective

$$J(\pi) = \underset{\tau\sim\pi}{\mathrm{E}}[R(\tau)] + \alpha \sum_t \mathcal{H}\left(\pi(\cdot|s_t)\right) \qquad \text{(MERL)}$$

where α is called the temperature parameter and controls the relative importance of the entropy term compared to the usual reward term

The Maximum Entropy framework has a wealth of advantages such as improved training stability and exploration phase, that we review in more details in part 2.5. In addition, having a look at the (MERL) objective, we understand the motivations behind the choice of SAC in the DIAYN framework. SAC maximizes the policy's entropy over actions which corresponds to the first term in $\mathcal{G}$ (VLB).

SAC can initially be derived from a maximum entropy variant of the policy iteration method - a general framework to learn optimal maximum entropy policies that alternates between two steps: a policy evaluation step and a policy improvement step.
The policy evaluation step boils down to computing a soft Q value iteratively (details can be found in [5]) and the improvement step amounts to update the policy towards the new Q-function while constraining the updated policy in the set of tractable policies.

This method has the desirable property of converging to the optimal policy within a parametrized set of tractable policies. The major issue is that the procedure relies on a tabular setting. The intuition behind SAC is to extend it to the continuous setting that we are studying in this project.

The idea is to approximate the Q-function and the policy and to swap the initial evaluation-improvement dynamic for a concurrent optimization process through stochastic gradient descent.

We now need to introduce three crucial functions :

- A parametrized state-value function $V_\psi(s_t)$ which approximate the soft value and is learnt through the minimization of the squared difference between the prediction of the value function and the expected prediction of the Q-function with the entropy of the policy, π :

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\psi\left(\mathbf{s}_t\right) - \mathbb{E}_{\mathbf{a}_t\sim\pi_\phi}\left[Q_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right) - \log\pi_\phi\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right]\right)^2\right]$$

where $\mathcal{D}$ is a replay buffer.
The gradient of the previous equation can be estimated with the following unbiased estimator:

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t)\left(V_\psi(\mathbf{s}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t) + \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t)\right)$$

Using this state-value function, we can now introduce the remaining two central functions:

- A soft Q-function $Q_\theta(s_t, a_t)$ parametrized by θ and which is learnt through the minimization of the following soft Bellman residual

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t)\right)^2\right]$$

where $\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}\sim p}\left[V_{\bar{\psi}}(\mathbf{s}_{t+1})\right]$
Again, in a gradient-based learning fashion:

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t, \mathbf{s}_t)\left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) - \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1})\right)$$

where $\bar{\psi}$ is the exponentially moving average of the value network weights

- A tractable policy function $\pi_\phi(a_t|s_t)$ parametrized by ϕ which training centers on :

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}, \epsilon_t\sim\mathcal{N}}\left[\log \pi_\phi(f_\phi(\epsilon_t; \mathbf{s}_t) \mid \mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t; \mathbf{s}_t))\right]$$

where $\mathbf{a}_t = f_\phi(\epsilon_t; \mathbf{s}_t)$
Finally, still in a gradient-based learning fashion:

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t) + \left(\nabla_{\mathbf{a}_t} \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t) - \nabla_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t)\right)\nabla_\phi f_\phi(\epsilon_t; \mathbf{s}_t)$$

That said, the whole procedure to update actor and critic are described in the full pseudo-code below:

Algorithm 2: Soft Actor-Critic - as given in [5]

Input: initialize parameter vectors $\psi, \bar{\psi}, \theta, \phi$
for *each iteration* **do**
  for *each environment step* **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  for *each gradient step* **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$

2.4 Our DIAYN implementation

Our implementation can be found on GitHub here : `https://github.com/remydeshayes/RL_DIAYN.git`

As described in Appendix C of [1], the authors implemented DIAYN building off of the work done for SAC in [5]. Regarding the discriminator, a look at the paper's GitHub repository [6] shows that they used a two

layer perceptron with 300 hidden units.

At first, we tried to use the code provided by the authors [6] to run some experiments. Unfortunately, the Docker build was broken, and even after some trials, we have not succeeded in debugging the Dockerfile. Furthermore, the authors used a SAC framework which is no longer maintained [7].

As a result, we decided to implement DIAYN using Gym, Stable Baselines 3 and PyTorch.
We used a SAC implementation from Stable Baselines 3 and developed a Gym wrapper to perform the necessary steps to update the discriminator. The wrapper overwrites functions such as "step" or "reset" from Gym to change the reward and perform necessary updates. It also adds some neat features to follow the training such as a progress bar with useful information to observe the discriminator loss.
In addition, our implementation allows us to use alternative algorithms instead of SAC to train the agent if needed.

As for SAC optimizes a stochastic policy in an off-policy way, the discriminator update procedure rests on a replay buffer. On another note, we used Adam to update the discriminator. Like the authors, we augmented the observation by concatenating a one-hot encoded version of z to the current state s_t.

The algorithm has been run on two Gym environments named Pendulum-V0 and MountainCarContinuous-v0. We used an MLPolicy for both environments.

Following are the hyperparameters we used:

- $\alpha = 0.1$ (temperature parameter as seen in 2.3)
- number of skills : 5
- total timesteps : 10000
- hidden layers dimension : 32
- learning rate for the discriminator : 1e-3
- learning rate for the actors and critics : 3e-4

For the sake of simplicity, throughout our implementation, we arbitrarily used 5 as for the number of skills to learn.

2.4.1 Results

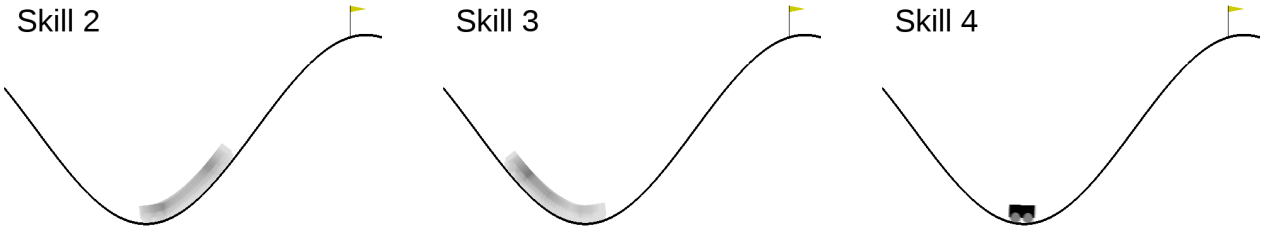
Each and every skill learnt both in the pendulum and mountain car environments of our DIAYN implementation are available in the following videos we uploaded on Youtube:

- link to the pendulum video : `https://youtu.be/scjX7YhNthM`
- link to the mountain car video : `https://youtu.be/XRDxTBMpc8g`

In what follows, we cherry-pick some of the skills to spotlight the diversity and distinguishability yielded by DIAYN. First, we were able to balance the pendulum slighlty to the right on Figure 2a - this would seem like a rather interesting skill to use as an initialization for the classical pendulum upper balance task. We discuss the idea of using DIAYN as unsupervised pre-training in the following part 2.5. We were also able to make almost symmetrical spins to the right - Figure 2b - and to the left - Figure 2c - at various speeds and even to make full revolutions - this can be seen in the pendulum video.
Regarding the mountain car, we witnessed various car speeds and directions - we present some of them in Figure 3. It is interesting to note that, although our car climbed the left hill to the top, it never crossed the

(a) Stationary (b) Spinning to the right (c) Spinning to the left

Figure 2: Our pendulum developed various skills during training



(a) Climbing to the right (b) Climbing to the left (c) Standing still

Figure 3: Our mountain car developed various skills during training

finish line on the right. Again, this can be seen in the mountain car video.

We noticed that our implementation was robust to random seeds. Indeed, we observed very similar results for each skill even after a change in the initialization. We discuss this empirically noted stability and more in the following part 2.5.

2.5 DIAYN's practical benefits and purposes

One great advantage of DIAYN is its robustness to random seed, its stability. Indeed, as introduced in part 2.2, DIAYN is a cooperative game which bypasses a great deal of issues associated to adversarial games namely non-convergence, model collapse and diminished gradient which for instance frequently arise in the 2-player minimax game framework of GANs.

The task-agnostic aspect of DIAYN that we introduced in section 1 is especially beneficial to a handful of RL problems that can build off of DIAYN's outcomes:

- Unsupervised pre-training: the idea is to select DIAYN's skill that has the highest reward for a specific task and use it as an initialization to further fine tune it in a task-oriented context.
- Hierarchical RL (HRL): the HRL idea is to break up a complex task in multiple reusable easier tasks - motion primitives. In practice, those methods have faced numerous problems such as having all motion primitives trying to do the entire task.
- Imitation learning: the idea behind imitation learning is to follow expert demonstrations or hard-coded agents.

3 Final remarks and related work

Before wrapping up our project, we briefly lay out where DIAYN stands in the unsupervised RL literature by introducing its most similar work, namely the *Variational Intrinsic Control* - [8].

3.1 Variational Intrinsic Control - [8]

DIAYN aspires to be the offspring of the *Variational Intrinsic Control* (VIC) introduced by K. Gregor, D. Jimenez Rezende and D. Wierstra in [8] (2016). VIC's purpose is very similar to DIAYN : providing an algorithm to discover as many skills as possible using an information theoretic learning criterion. However, DIAYN's authors chose to fix the prior $p(z)$ instead of learning it and allowed their discriminator to compute probabilities at each state whereas VIC's discriminator only do so in the final state.

3.1.1 Learning $p(z)$

Learning $p(z)$ boils down to choosing a distribution which maximizes mutual information between states and skills:

$$I(S, Z) = \mathcal{H}(Z) - \mathcal{H}(Z|S)$$

Let $p_z^t(s)$ be the distribution over states induced by skill z at epoch t and $l_t(z)$ be an approximation of $\mathbb{E}\left[\log p(z|s)\right]$ - See part 2.1 for detailed discussion on $p(z|s)$.
Using the method of Lagrange multipliers, DIAYN's authors show in Appendix E of [1] that:

$$p(z) \propto e^{l_t(z)} \tag{1}$$

Thus, using (1), one can learn $p(z)$. However, the authors argue that the exponential of the entropy of skills $e^{\mathcal{H}(Z)}$, which is also the effective number of skills, drops by a factor of 10 when $p(z)$ is learnt.
Learning $p(z)$ instead of fixing it leads to a poorer skill diversity.

3.2 Final remarks

DIAYN is a genuinely engaging paper as it offers a general framework to train an agent in an unsupervised manner with a reward not depending on the environment but only on the observations.
Moreover, any RL algorithm with entropy maximizing objective can be used as an alternative to the SAC method.

Unfortunately, the paper lacks of theoretical results as underlined by the authors. Indeed, there is no guarantee that there is convergence [1]. Still, the authors provide an extensive review of their algorithm through their q&a format Experiments section.
Finally, further thoughts could be given to decide upon the number of skills one will use as an hyperparameter.

References

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

[2] Josh Achiam. Openai, spinning up in deep rl. `https://spinningup.openai.com/en/latest/spinningup/rl_intro.html`, 2018.

[3] David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 01 2016.

[4] D. Barber and F. Agakov. The im algorithm: a variational approach to information maximization. In *NIPS 2003*, 2003.

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

[6] Benjamin Eysenbach. Official github repository for diversity is all you need. `https://github.com/ben-eysenbach/sac/blob/master/DIAYN.md`, 2018.

[7] Tuomas Haarnoja. Official github repository for soft actor-critic. `https://github.com/haarnoja/sac`, 2018.

[8] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control, 2016.