Off-Policy Reinforcement Learning in the Game of Rocket League

Chanzo Bryan
Department of Computer Science
University of Toronto
Toronto, ON M5S 1A1
chanzo.bryan@mail.utoronto.ca

Ting-Fu Hsu
Department of Statistics
University of Toronto
Toronto, ON M5S 1A1
tingfu.hsu@mail.utoronto.ca

Kevin Covelli
Department of Computer Science
University of Toronto
Toronto, ON M5S 1A1
kevin.covelli@mail.utoronto.ca

Abstract

In this paper we implement two off-policy reinforcement learning methods, Soft Actor Critic and Diversity is All You Need, for training a high-dimensional deep neural network agent in the Rocket League environment. We discuss our implementation of these algorithms, and their shortcomings when it comes to completing the Rocket League shootout task.

1 Introduction

Reinforcement learning (RL) algorithms have been shown to achieve a strong performance over a variety of problems and domains [Mnih et al., 2013, 2015, Schulman et al., 2016]. Model-free deep RL suffers from a key issue in that it is very sample inefficient, requiring millions of environment steps to solve even simple environments. Off-policy algorithms aim to rectify this by allowing algorithms to learn from their previous experiences, reducing the total amount of experiences necessary to achieve a strong performance. Another approach to increasing sample efficiency is to incorporate an unsupervised learning component to learn skills. Using skills as the baseline actions to other RL algorithms have been shown to vastly improve the sample efficiency [Sharma et al., 2020]. In our paper, we will contrast the Soft Actor-Critic (SAC) algorithm [Haarnoja et al., 2018a] using baseline actions, with Diversity is All You Need (DIAYN) [Eysenbach et al., 2018], an algorithm that uses unsupervised learning to learn "skills".

2 Related Works

A central feature of SAC [Haarnoja et al., 2018a,b] is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy. Maximizing entropy has the effect of encouraging exploration of the environment which can accelerate learning later on, and can also prevent the policy from prematurely converging to a bad local optimum. Similar to TD3 [Dankwa and Zheng, 2019], it uses twin Q-functions to prevent the overestimation of Q-values.

Unsupervised learning methods for reinforcement learning problems come in many archetypes [Laskin et al., 2021]. Random Network Distillation [Burda et al., 2019] predicts the outputs of a random network on states and maximizes the prediction error, which has been shown to solve

Montezuma's Revenge, a sparse reward environment. DIAYN [Eysenbach et al., 2018] is a hierarchical reinforcement learning algorithm, attempting to learn higher level actions or *skills*. These skills are learned using unsupervised learning, and then an agent can be trained using them with any reinforcement learning algorithm. Skills are actions that affect the environment in an controlled fashion. They are discovered by maximizing the mutual information between the states and skills. DIAYN specifically uses SAC as its reinforcement learning algorithm, training the agent end-to-end, learning both skills and an optimal policy concurrently.

3 Methods and Algorithms

For our comparison we will implement both version two of SAC [Haarnoja et al., 2018b] and DIAYN [Eysenbach et al., 2018]. As both algorithms are off-policy we will also be using experience replay [Lin, 1992] to allow them to learn from previous experiences. Prioritized Experience Replay (PER) [Schaul et al., 2016] is an enhancement over standard experience replay, allowing experiences to be weighted by the loss on those experiences using the current parameters. This allows experiences to be selected more frequently when they are unfamiliar, reducing the number of steps required to achieve a good performance.

Our DIAYN implementation in Algorithm 1 concatenates the meta-controller outputs with the state as an input to the policy. This allows us to perform gradient descent on the meta-controller using the same SAC policy loss. In the pretraining phase, the meta-controller is trained along with the SAC policy and DIAYN discriminator, but it is unused. Once in the regular training regime, the meta-controller selects the skills that are used by the policy. Our SAC algorithm is detailed in Algorithm 2, we use the same training schema outlined in [Haarnoja et al., 2018a], with a replay buffer. For both algorithms, we elect to take P steps in the environment and store the transitions in the replay buffer before training begins.

Algorithm 1 Diversity is All You Need

Initialize DIAYN discriminator with parameters ϕ, meta-controller with parameters ψ, SAC with parameters θ, skill distribution $p(z) \sim \mathcal{U}$
for pre-training iteration $= 1, 2, \ldots, T$ **do**
  Sample skill $z \sim p(z)$
  for skill step = $1, 2, \ldots K$ **do**
    Sample action a_t from SAC with skill z
    Take action a_t and receive transition tuple (s_{t+1}, d_t)
    Set $r_t \leftarrow$ DIAYN skill reward
    Store transition tuple $(s_t, a_t, s_{t+1}, r_t, d_t, z)$ in PER buffer
    Sample experience $(s', a', r', s'', d', z')$ from PER
    Update θ, ψ with SAC on transition (s', a', r', s'', d)
    Update ϕ with DIAYN on transition (s'', z')
  end for
end for
for training iteration $= 1, 2, \ldots, N$ **do**
  Choose skill z_t for state s_t from the meta-controller
  Sample action a_t from SAC with skill z_t
  Take action a_t and receive transition tuple $(s_{t+1}, r_t d_t)$
  Set $r_t \leftarrow r_t +$ DIAYN skill reward
  Store transition tuple $(s_t, a_t, s_{t+1}, r_t, d_t, z_t)$ in PER buffer
  Sample experience $(s', a', r', s'', d', z')$ from PER
  Update θ, ψ with SAC on transition (s', a', r', s'', d)
  Update ϕ with DIAYN on transition (s'', z')
end for

Algorithm 2 Soft Actor Critic

Initialize SAC with parameters θ
for iteration $= 1, 2, \ldots, N$ **do**
  Sample action a_t from SAC
  Take action a_t and receive transition tuple (s_{t+1}, r_t, d_t)
  Store transition tuple $(s_t, a_t, s_{t+1}, r_t, d_t)$ in PER buffer
  Sample experience (s', a', r', s'', d') from PER
  Update θ with SAC on transition (s', a', r', s'', d')
end for

3.1 Training Setup

Both models are trained on a single GPU and CPU, with a 1:1 ratio of update steps to training samples gathered. The RLGym [RLGym, 2020] environment consists of a car and ball on a pitch in a shootout scenario. An episode ends when a goal is scored. The environment provides a 42-dimensional input representing the in-game memory state including values such as the car and ball positions, velocities, and the previous action. The actions are a concatenation of a 5-dimensional continuous input of the throttle, steer, yaw, pitch and roll of the car, and a 3-dimensional discrete input of whether the car should jump, boost (nitrous) or handbrake. For the purposes of these experiments, we will represent the actions as a single 8-dimensional continuous output, with a threshold value for the discrete actions. Under normal conditions, the RLGym environment ticks 120 times per second, so a frameskip of 8 will be used to produce a reduced 15 actions per second. Training will be done in an environment that is sped up by 100 times, however due to heavy computation requirements, it often ran at less than a 100 times speedup.

4 Experiments and Discussion

Table 1: Final experiment hyperparameters

Hyperparameter	Value
Optimizer	Adam [Kingma and Ba, 2014]
Learning Rate	10^{-4}
Discount	0.995
PER buffer size	10^6
Buffer initialization size	10^4
Hidden layer size	256

Hyperparameter	Value
Number of hidden layers	2
Minibatch size	1024
Polyak coefficient	$5 \cdot 10^{-3}$
Number of skills (DIAYN)	20
Skills pretrain steps	10^4

In these experiments, we aim to evaluate the robustness of SAC and DIAYN to less conventional control problems, where only a subset of the state is controllable. On the Rocket League shootout task, we contrast the two algorithms on their performance and training curves. Starting from the hyperparameters outlined in [Haarnoja et al., 2018b], we perform a grid search over multiple random seeds, keeping the best performing hyperparameters across the two algorithms. Our hyperparameters are listed in Table 1.

Plotting the DIAYN discriminator and skill distribution in Figure 1, we can see that the discriminator was able to converge, being able to discern skills. With the ball being at (1.5, 0), our plot of skills reveals that each skill starts with the car turning to the right, before diverging in paths. Most of the skills perform spirals of differing widths, while others drop to either side of the pitch. None of the skills correctly drive towards the goal, or towards the opposition goal.

Training both algorithms to 300k steps, we see that the two loss curves are very similar. With DIAYN using SAC as its underlying reinforcement learning algorithm, this is expected. SAC was able to train quicker, being able to reduce its critic and value losses after the initial explosion due to newly explored states being trained on.

We consider the task solved if an algorithm is able to achieve a reward of 75 consistently. Neither SAC or DIAYN were able to achieve this. One apparent problem with exploration was due to the nature of the controls in the environment. When jumping, a car can perform another jump in any

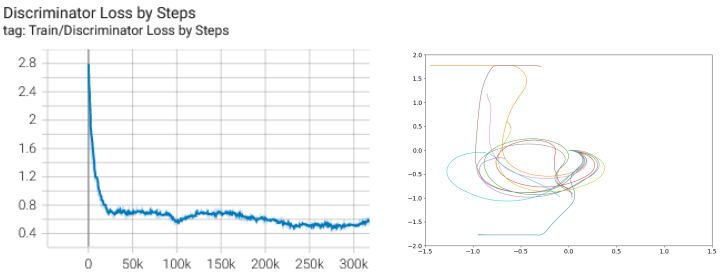

Figure 1: Discriminator loss and skill trajectories (20 skills)

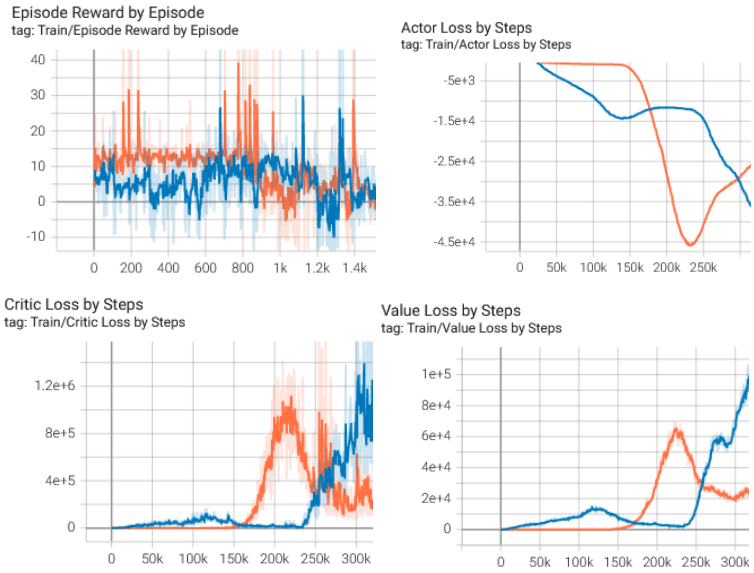

Figure 2: Training Curves for SAC and DIAYN

direction. If in an opposing direction, this second jump overrides the initial velocity, so slight input errors in a single timestep would stop agents from being able to reach the goal. We reduced the problem, removing the jump and handbrake controls, making exploration easier. Even under this reduced action set, neither algorithm was able to solve the environment. Our results confirm the findings in [Laskin et al., 2021], stating that competence-based unsupervised learning algorithms such as DIAYN underperform knowledge approaches such as RND [Burda et al., 2019].

A large portion of the algorithm failures can be attributed to the sparsity of the main reward, scoring a goal. The agent was able to score in less than 1% of all episodes. Without the main reward of scoring, there was not a major difference in the agent actions as long as it was driving in the correct direction. The true action space being a hybrid action space of both discrete and continuous actions was also pitfall in the models. We abstracted the action space to a pure continuous space to accommodate both algorithms, which removes some of the intricacies that could have been exploited by better suited algorithms.

5 Conclusion

We showcased SAC and DIAYN with the SAC backend on the Rocket League shootout environment. It was demonstrated that neither algorithm was able to solve the environment. The sparsity of the trajectories that lead to solutions as well adapted hybrid action space were the two noted factors that held the algorithms back. Approaching this problem in the future, a more robust algorithm as well as more computational power will be required to overcome the sparsity of the goals.

References

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *7th International Conference on Learning Representations (ICLR 2019)*, pages 1–17, May 2019. URL https://iclr.cc/. Seventh International Conference on Learning Representations, ICLR 2019 ; Conference date: 06-05-2019 Through 09-05-2019.

Stephen Dankwa and Wenfeng Zheng. *Twin-Delayed DDPG: A Deep Reinforcement Learning Technique to Model a Continuous Movement of an Intelligent Robot Agent*. Association for Computing Machinery, New York, NY, USA, 2019. ISBN 9781450376259. URL https://doi.org/10.1145/3387168.3387199.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018. URL https://arxiv.org/abs/1802.06070.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018a. URL https://arxiv.org/abs/1801.01290.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, G. Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, P. Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018b.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark, 10 2021.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

RLGym. The rocket league gym, 2020. URL https://rlgym.github.io/index.html.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2016.

Archit Sharma, Shixiang Shane Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *ArXiv*, abs/1907.01657, 2020.