Novelty-Guided Proximal Curriculum Learning

Jan Malte Töpperwien

Abstract

Reinforcement Learning proves to be able to solve increasingly harder environments, but still has trouble on sparse reward environments like Montezuma's Revenge. One Approach to solve this is by using curriculum learning to accelerate learning of the policy. Finding a good curriculum has often constrained itself to imitation learning or inflexible curricula. Recent work has allowed to create more flexible curricula by dynamically judging states and directly starting from these. This work takes the approach of setting appropriate starting states via information from the agent and combines it with state novelty approaches to simultaneously explore the state-space, leveraging the ability to set starting states. This is demonstrated by combining *Proximal Curriculum Learning* with *Random Network Distillation*.

The project can be found at: `https://github.com/CrunchyFlakes/Novelty-Guided-Proximal-Curriculum-Learning`

1 Introduction

Reinforcement learning (RL) has shown to be able to solve difficult tasks in simulated environments ([Mnih et al., 2015], [Lillicrap et al., 2016], [Silver et al., 2017]). However, to solve real-world problems an agent has to often solve problems which exhibit large action- and state spaces, sparse rewards and changing tasks to get to a goal. This poses the problem of exploring these spaces and, even with good exploration getting close to the goal, the agent often does not get a reward due to the sparsity of these. Additionaly, training data gathering on real-world problems through e.g. simulation is often expensive and exploration therefore has to be done efficiently.

One proposed method to tackle this problem is *Curriculum Learning* (CL) [Bengio et al., 2009]. Curriculum learning applies the pedagogical idea of giving a student increasingly difficult tasks to RL algorithms. This allows the agent to get to the reward more often on easier tasks and only after that to progress to more difficult tasks, which would have previously led to seldom rewards at best. A possibility to implement curriculum learning is by setting the starting state closer to the goal difficulty-wise and increasing this distance over the course of training. One approach to measure this distance is given by Tzannetos et al. [2023], where they measure the difficulty of the task by using the value function of the agent and proceed to give the agent tasks appropriate to its current learning progress. One weakness of this approach is, that at the beginning of learning the value function has not converged to proper values and therefore unsuitable starting states may be chosen.

To explore the state space, *state novelty* methods like random network distillation [Burda et al., 2018] have proven to be successfull in steering the agent towards underexplored states. This is usually done by adding an intrinsic reward to the agent for finding new states.

This work aims to combine curriculum learning as done in Tzannetos et al. [2023] with random network distillation [Burda et al., 2018]. By setting starting states also based on state novelty, the state space can be properly explored from the beginning. This helpts the agent to explore the state space and overcome the random initialization of the value function, leading to tasks with appropriate difficulty.

Project proposal for lecture "Advanced Topics in Reinforcement Learning"

2 Related Work

Reverse Curriculum Generation [Florensa et al., 2017] uses random actions beginning from the goal to sample nearby starting states that are easy enough for the agent to solve. It then proceeds to do some rollouts used for learning, sampling uniformly from these states. The starting states are then rated using the collected rollouts to prune the ones which are too easy or too hard. Then we randomly explore from these states to get new starting states. This is done until learning finishes.

Prioritizing Starting States for Reinforcement Learning [Tavakoli et al., 2018] derives a list of starting states using states already discovered in the training process. A limited length buffer is used to collect these states and is processed in different fashions:

1. Sampling uniformly from these states.
2. Using prioritized sampling similar to the *prioritized experience replay* [Schaul et al., 2016].
3. Sampling from states that are part of trajectories yielding high returns to address sparse reward domains.

Multiple approaches exploit expert knowledge to generate suitable starting states.
Nair et al. [2018] first sample a trajectory from a given set of export demonstrations and then uniformly sample a starting state out of that trajectory.
Similar to the previous approach Peng et al. [2018] teach an agent to animate characters inside a physical simulation by sampling a starting state out of a motion demonstration (e.g. backflip) for a given character.
Salimans and Chen [2018] exploit only one expert trajectory by first setting starting states from the end of the trajectory and then working its way back to the trajectorys start. The agent therefore starts training on easy states near the goal and ends training at the starting state.

Go-Explore [Ecoffet et al., 2019] is divided into two phases. In phase 1, called *Explore until solved*, it explores from the starting state by making 100 steps where each step has a 95% chance to repeat the previous action and 5% to take a random action. For promising trajectories this step is done again starting from the end of the trajectory.
In phase 2 found high-promising trajectories are used as expert trajectories, first setting states at the end of the trajectories and then working its way back as proposed by Salimans and Chen [2018].

These methods either rely on expert demonstration being available and/or do not allow the agent to set the pacing of task-difficulty itself. The next two covered methods are the two which will be combined in this work to cover those weaknesses.

Random Network Distillation (RND) approximates state novelty by using two different neural networks. One of them stays fixed while the other one tries to approximate the fixed one. When inputting states in both, the loss steers the approximating one towards the fixed one, resulting in a lower loss for often seen states. The loss is therefore a measure of novelty of the state. This approach and its implementation will be covered more thoroughly in section 3.2.

Proximal Curriculum Learning (PCL) uses the agents value function to estimate the probability of success for a given state. The bigger the value, the higher the probability of success. Out of this, a distribution over the states is created which favors probabilities around 0.5. This approach will be explained in more detail in section 3.1.
Although being self-paced when the generated value approximatioons correspond to the agents learning state, directly after initialization the value function is random and setting starting states purely based on that may result in improper exploration. To mitigate this problem this work will additionally incorporate RND, which should help with exploration and in extension faster value function convergence.

3 Approach

3.1 Proximal Curriculum Learning

Tzannetos et al. [2023] use the probability of success (PoS) to pick starting states which are close to the agents capability. To calculate this probability they actually proposed two methods, using rollouts

or using the value function of the agent. We will only look at the value function approach, because rollouts are expensive and we cover the value functions shortcomings by using state novelty.

The starting state distribution is calculated given the following equations. S_{init} specifies the pool of starting states and PoS_V corresponds to the approximated probability of success given the value function at step t:

$$\mathbb{P}_{Prox}(s_0^{(t)} = s) \propto \exp(\beta_{Prox} * PoS_V(s) * (1 - PoS_V(s))), \quad PoS_V(s) = \frac{V(s) - V_{min}}{V_{max} - V_{min}} \quad (1)$$

β_{Prox} is a hyperparameter which allows us to smooth or exaggerate the distribution.

3.2 Random Network Distillation

Burda et al. [2018] use two neural networks, of which one is fixed while the other learns to approximate the fixed one given the occurring states as input.

The original approach uses the novelty approximation to add an intrinsic reward to the agent, but, since we already have the ability to set starting states, this work only uses it to calculate a *distribution* similar to section 3.1 over the states in S_{init}. To obtain this distribution the following calculations are made. $SN^{(t)}$ denotes the state novelty given the parameters at step t of the approximating network and $J^{(t)}$ denotes the loss given the parameters at step t:

$$\mathbb{P}_{RND}(s_0^{(t)} = s) \propto \exp(\beta_{Nov} * SN^{(t)}(s)) \quad SN^{(t)}(s) = \frac{J^{(t)}}{J^{(t)}_{max}} \quad (2)$$

SN is only normalized using the maximum, to allow the novelty values to encode that each state is similarly well known instead of amplifying noise by scaling it to the whole interval $[0, 1]$.

3.3 Combining both approaches: Novelty-Guided Proximal Curriculum Learning (NGPCL)

To combine both approaches it suffices to lay both distributions (eq. (1), eq. (2)) on top of each other using a hyperparameter γ. Additionally one can update the set of starting states S_{init} on every step t.:

$$\mathbb{P}(s_0^{(t)} = s) = \gamma * \mathbb{P}_{Prox}(s_0^{(t)}) + (1 - \gamma) * \mathbb{P}_{RND}(s_0^{(t)}) \qquad s \in S_{init}^{(t)} \quad (3)$$

Algorithm 1 Training Algorithm

Require: environment e, Agent with policy $\pi^{(0)}$ and value function $V^{(0)}$, Starting states $S_{init}^{(0)}$, number of episodes N, performance threshold η, β_{Prox}, β_{RND}, γ
t = 0
reward = 0
while $t < N \wedge reward < \eta$ **do**
  sample $s_0^{(t)} \sim \mathbb{P}(s_0^{(t)} = s)$ using $S_{init}^{(t)}, \beta_{Prox}, \beta_{RND}, \gamma$ (eq. (1), eq. (2), eq. (3))
  Take step in A using action from $\pi^{(t)}$
  Train $\pi^{(t)}$, $V^{(t)}$ using replay buffer and new observation
  Train $J^{(t)}$ on new observation
  Calculate new $S_{init}^{(t)}$
end while
return policy π

4 Experiments

The approach was evaluated on two environments with sparse rewards and compared against PCL and RND on its own and a baseline that samples starting states uniformly from $S_{(init)}$ called vanilla.

The uniform sampling in vanilla ensures that the approach is not too disadvantaged due to the sparsity compared to the other approaches. Evaluations were done using unmodified environments where the agent starts at the actual starting state.

The experiments were run using PPO from Stable-Baselines3 [Raffin et al., 2021] using a multi-layer-perceptron as policy and value network. For every approach hyperparameters were optimized on 100 trials using SMAC3 [Lindauer et al., 2022].
To save on compute, trials were terminated early if they exceeded the performance threshold of 0.95, corresponding to a solved environment. To reward fast learning configurations, the timesteps needed to reach this threshold was incorporated into the function SMAC optimizes. A maximum of one million timesteps was set.
The architecture of the policy and critic network were set the same, while RND got its own set of hyperparameters corresponding to its architecture. RND was trained using a multi-layer-perceptron with the mean-squared-error as loss and ReLU or LeakyReLU as activation function.

Experiments were run on the LUH-cluster using an AMD EPYC 7513, 32GB of RAM and 20 CPU cores. A full run of parallelized HPO with sequential evaluation runs of the best found configuration afterwards did take about 10-14 hours. Evaluations were done on five different seeds for the best configuration found in each approach.

4.1 Environments

All four approaches were evaluated on *MiniGrid-DoorKey-8x8-v0* and *MiniGrid-Unlock-v0* out of the MiniGrid library [Chevalier-Boisvert et al., 2023].

Unlock tasks the agent with finding the key, picking it up and then opening the door. A reward of $1 - 0.9 * (step\_count/max\_steps)$ is given after opening the door. The action space lets the agent choose one of the following actions: turn left or right, move forward, toggle (open/close) the door and 3 unused actions. The observation is given for all states in front and to the side of the agent until reaching a wall or door. For all visible cells it includes the object index, color index (not relevant in these environments) and the state of the object. To relax the problem a single boolean variable was added to the observation, stating if the agent currently posesses a key.

DoorKey adds the additional task of using the pickup action on the key instead of picking it up directly and to get to the goal state which is behind the locked door. Apart from that it is the same as Unlock. This task is significantly harder, because the agent has to actively pick up the key and also find the goal inside the locked room.

To keep the implementation simple S_{init} uses the given environment after a reset and then creates all states over the agents starting position and direction, key carrying status, and door status.

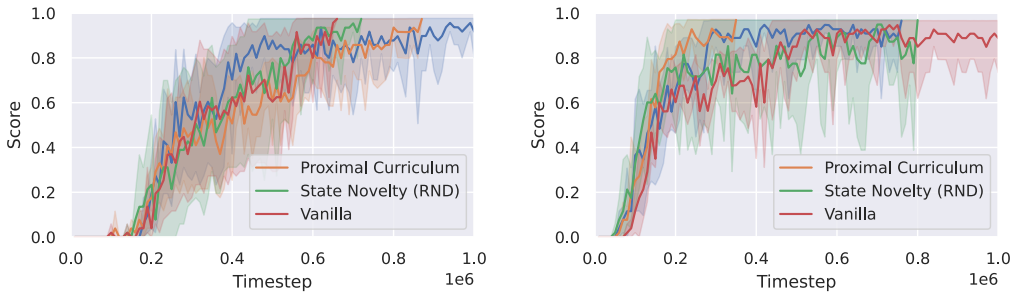
4.2 Results

A typical hyperparameter-configuration used architectures with two to four layers for all networks. Layer sizes where usually picked from starting at 128 and then going to 32 or 64. β_{NOV} was usually close to 20, while β_{PROX} usually was set to 20, but one time got a value of four.
Most of these values are close to the defaults.

Figure 1a shows that NGPCL can lead to worse results than using PCL and RND alone. NGPCL was the fastest to get decent results, but then failed to converge to solving the environment. The best approach here was the baseline, finishing at around 630.000 timesteps on average. RND finished at around 650.000 and PCL finished at about 870.000 timesteps on average. The confidence intervals do not differ by that much, only NGPCL did have significant outbreaks towards the end of learning.

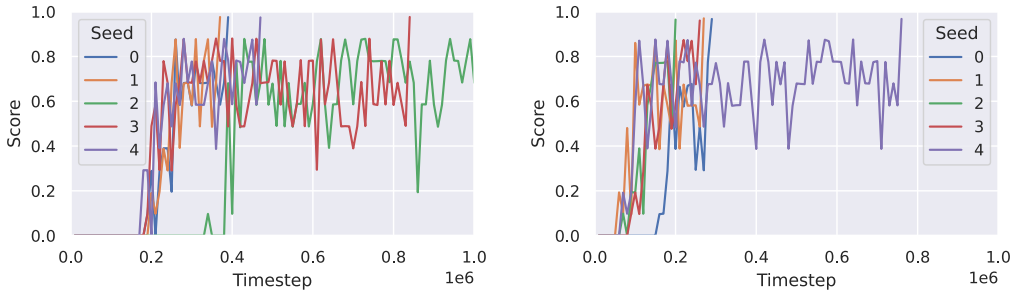
In the Unlock environment (also fig. 1) PCL was the best approach, finishing really fast after about 370.000 steps on average. Vanilla was the worst approach, never being able to converge completely. RND did also not perform well, finishing after 800.000 steps and having significant deviations in performance. NGPCL performed worse than PCL, but was close to terminating after about 370.000 steps.

Looking at fig. 2 shows us, that NGPCL is able to solve the environments fast on most seeds, but sometimes is not able to converge properly.



(a) Doorkey-8x8 (b) Unlock

Figure 1: Rewards over the training steps with 95% confidence intervals over 5 seeds. Later timesteps in early terminated episodes where filled with the value achieved at termination.



(a) Doorkey-8x8 (b) Unlock

Figure 2: Rewards over the training steps for Novelty-Guided Proximal Curriculum Learning

5 Discussion

Novelty-Guided Proximal Curriculum Learning enables to combine the efficiency of Proximal Curriculum Learning and exploration of state novelty approaches like Random Network Distillation. This allows an agent to set its learning pace itself, while exploring the state space efficiently using different starting states. It also provides a way to do curriculum learning without expert demonstrations or environment knowledge, although the success is limited. The main drawback of this approach is, that environments have to support the setting of the starting state.

The experiments showed that this approach may result in faster learning performance, but it is not able to do this robustly on different seeds.
One reason for this behaviour could be, that the direct overlaying of the distributions results in doing neither approach well and therefore picking of unreasonable starting states. This may be improved by interleaving PCL and RND on each environment reset or scheduling the weight used for combination to allow for first getting a good value function via state novelty exploration and then exploit that knowledge to train the agent using PCL.
Another reason may be, that hyperparameter optimization was not run long enough, which is further supported by the fact that a lot of hyperparameters where set as the given default. To save on compute, a multi-fidelity approach would be reasonable.
Environments with dense rewards or local minimas in the reward space would additionally be interesting to judge the performance on significantly different tasks.

In the long term, future work is needed on the actual picking and evolving of the pool of starting states, which would make this approach applicable to large state-action spaces and may improve the performance significantly. Finding interesting trajectories and using them as the pool of starting states, like in Go-Explore, would be one way promising way to go about this.
Different state novelty approaches may also lead to better performance.

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In Andrea Pohoreckyj Danyluk, Léon Bottou, and Michael L. Littman, editors, *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, volume 382 of *ACM International Conference Proceeding Series*, pages 41–48. ACM, 2009. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. *CoRR*, abs/1810.12894, 2018. URL http://arxiv.org/abs/1810.12894.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo de Lazcano, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *CoRR*, abs/2306.13831, 2023.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019. URL http://arxiv.org/abs/1901.10995.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, pages 482–495. PMLR, 2017. URL http://proceedings.mlr.press/v78/florensa17a.html.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL http://jmlr.org/papers/v23/21-0888.html.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas 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. *Nat.*, 518(7540):529–533, 2015. doi: 10.1038/NATURE14236. URL https://doi.org/10.1038/nature14236.

Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 6292–6299. IEEE, 2018. doi: 10.1109/ICRA.2018.8463162. URL https://doi.org/10.1109/ICRA.2018.8463162.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143, 2018. doi: 10.1145/3197517.3201311. URL https://doi.org/10.1145/3197517.3201311.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. *CoRR*, abs/1812.03381, 2018. URL http://arxiv.org/abs/1812.03381.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05952.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nat.*, 550(7676):354–359, 2017. doi: 10.1038/NATURE 24270. URL https://doi.org/10.1038/nature24270.

Arash Tavakoli, Vitaly Levdik, Riashat Islam, and Petar Kormushev. Prioritizing starting states for reinforcement learning. *CoRR*, abs/1811.11298, 2018. URL http://arxiv.org/abs/1811.11298.

Georgios Tzannetos, Bárbara Gomes Ribeiro, Parameswaran Kamalaruban, and Adish Singla. Proximal curriculum for reinforcement learning agents. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=8WUyeeMxMH.

Checklist

1. General points:
   (a) Do the main claims made in the abstract and introduction accurately reflect your contributions and scope? [Yes]
   (b) Did you cite all relevant related work? [Yes]
   (c) Did you describe the limitations of your work? [Yes]
   (d) Did you include a discussion of future work? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [NA]
   (b) Did you include complete proofs of all theoretical results? [NA]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you run at least 5 repetitions of your method? [Yes]
   (d) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (e) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you make sure the license of the assets permits usage? [Yes]
   (c) Did you reference the assets directly within your code and repository? [Yes]