Automatic Goal Generation for Reinforcement Learning Agents

Alessandro Trenta - 566072

ISPR - Dip. of Informatics
Università di Pisa

27 maggio 2022

The problem

- In general RL frameworks the main objective is to find a policy $\pi(a_t|s_t)$ that maximizes the expected future return.
- In this problem, instead of maximizing the return over a single reward function we want to analyze the situation in which we have a range of reward functions r^g indexed with a goal $g \in \mathcal{G}$.
- A goal is defined as a set of states $\mathcal{S}^g \subset \mathcal{S}$. The reward function associated to this goal is

$$r^g(s_t, a_t, s_{t+1}) = \mathbb{1}\{s_{t+1} \in \mathcal{S}^g\}$$

- We will consider $S^g = \{s \in \mathcal{S} : d(f(s), g) \leq \epsilon\}$ where f projects the states in the goal space $\mathcal{G}$ and d is a distance in this space.

- Given g we consider a Markov Decision Process that terminates whenever $s_t \in \mathcal{S}^g$. We then consider the return to be $R^g = \sum_{t=0}^{T} r_t^g$. This is actually a binary random variable and indicates whether the agent is able to reach a state close enough to the goal in at maximum T time steps.
- The policy is also g dependent: $\pi(a_t|s_t, g)$ and the expected return for a goal is

$$\begin{aligned} R^g(\pi) &= \mathbb{E}_{\pi(\cdot|s_t,g)}\left[\mathbb{1}\{\exists t \in [1, \dots, T] : s_t \in \mathcal{S}^g\right] \\ &= \mathbb{P}\left(\exists t \in [1, \dots, T] : s_t \in \mathcal{S}^g\right) \end{aligned}$$

- We then assume to have a test distribution over goals p_g so that out objective is to maximize the expected mean return over goals obtaining the policy

$$\pi^*(a_t|s_t, g) = \arg\max_{\pi} \mathbb{E}_{g \sim p_g(\cdot)}[R^g(\pi)]$$

which is indeed the average probability of success over all possible goals (w.r.t. p_g).

Three main assumptions:

- A policy learned from enought goals in specific area of $\mathcal{G}$ can learn to interpolate well for other goals in this area.
- A policy trained on some set of goals is a good initialization for other goals that are close enough.
- If a goal is reachable, there is a policy that can reach it consistently.

The main catch: goals of intermediate difficulty generation

- We want to train our agent gradually. At each iteration of policy training we don't want the agent to work on goals that it can barely reach or that are too easy (other than avoiding catastrophic forgetting).
- How do we define hard and easy goals? Introduce the set of **Goals of intermediate difficulty** (for the i-th iteration):

$$GOID_i := \{g : R_{\min} \leq R^g(\pi_i) \leq R_{\max}\}$$

  that is the goals which probability to be reached in at most T time steps with the current policy π_i is in the range $[R_{\min}, R_{\max}]$.
- We then want a way to generate new goals in this set in a efficient way. This is done using a Generative Adversarial Network called the goal GAN.
- To train the GAN we first give a label y_g to the goals used in the last iteration of training set to 1 if they are in $GOID_i$ and 0 otherwise. This is done by policy evaluation.

- The goal GAN is responsible for one of the key parts of the model: generate new goals of the right difficulty for current iteration. The generator G aims to generate goals g from a noise vector $z \sim p_z(\cdot)$. The discriminator has to distinguish goals in $GOID_i$ from those that are not.
- The two losses are defined as:

$$
\begin{aligned}
V(D) = {} & \mathbb{E}_{g \sim p_{\text{data}}(g)} \left[y_g (D(g) - b)^2 + (1 - y_g)(D(g) - a)^2 \right] \\
& + \mathbb{E}_{z \sim p_z(z)} \left[(D(G(z)) - a)^2 \right] \\
V(G) = {} & \mathbb{E}_{z \sim p_z(z)} \left[(D(G(z)) - c)^2 \right]
\end{aligned}
$$

- We chooose $a = -1, b = 1, c = 0$ so that goals used for previous iteration steps $\sim p_{\text{data}}$ that are in $GOID_i$ (with $y_g = 1$) have positive score $D(g) \mapsto 1$, while those too hard or too easy $y_g = 0$ have negative scores $D(g) \mapsto -1$.
- The last term in the discriminator loss is the usual discrimination term for data generated by the generator G. The G loss let as usual the generator train to fool D.

The procedure

The algorithmic procedure is the following: starting from an initial policy π_0 and an empty set of $goals_{\text{old}}$ we iterate N times this steps

- Start by sampling the noise vectors z from $p_z(\cdot)$.
- Generate the set of current goals $goals = G(z) \cup sample(goals_{\text{old}})$: $\frac{2}{3}$ are obtained by feeding the generator with z and the other $\frac{1}{3}$ are sampled from the old goals we already trained on to avoid catastrophic forgetting.
- Update the policy to π_i. This can be done with any RL policy based method (in this case a variant of TRPO).
- Calculate the empirical returns for each current goal w.r.t. the new policy π_i and obtain their labels $y_g \in \{0, 1\}$ based on its difficulty (or belonging to $GOID_i$).
- Train the GAN with the current goals (the p_{data} above) and labels to be able to generate new goals with appropiate difficulty. Update the set $goals_{\text{old}}$ with the new goals used for training this iteration.

Experimental Results

- The model was tested in 4 main settings. An ant locomotor moving in a free bidimensional space and a U shaped maze, a single point mass moving in a single multi-path maze and in a N dimensional setting with decreasing reachable states as $N \to \infty$.
- In all cases the goals were points in the space that the agent has to reach within ϵ radius of precision in T steps.
- Models in which the current goals are sampled uniformly from $\mathcal{G}$ suffer from being trained on currently not reachable goals and performed worse than this model. Training on $GOID_i$ goals gives far better results.
- The goal GAN samples new goals efficiently. In the free setting they cover circles of increasing radius on iterations while in the maze settings they follow the paths.
- The percentage of newly generated goals that are in $GOAL_i$ is every iteration around 20%, giving the agent new goals to work on even as the policy improves. In the end the agent can reach with high success rate any given goal.

- To label a goal it is tested around 4 times. This allows to choose $R_{\min} \in (0, 0.25)$, $R_{\max} \in (0.75, 1)$ without significant performance changes.
- To show the efficiency of the goal GAN the model was also tested against two goal selection models: in the first one the GAN is trained every goal attempted in last iteration, including the ones with $y_g = 0$ (GAN fit all), the second samples uniformly in the set of feasible goals keeping only the ones in $GOID_i$ (oracle).
- The first method performed worse as new goals are not based on current performance. The second one generates perfect goals but is much more inefficient as it has to label lots of goals every iteration. It gives un upper bound on performance. The main model is really close to it.
- In the N dimensional setting where the reachable area is $[-5, 5] \times [-1, 1] \times [-0.3, 0.3]^{N-2}$ and gets narrower as N increases (the goal has to be reached within $\epsilon_N = 0.3\frac{\sqrt{N}}{\sqrt{2}}$) the model performs very close to the oracle one, while the others suffer a lot from the decreasing feasible area

Bibliografia I

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel.
Automatic goal generation for reinforcement learning agents, 2017.