Automatic Goal Generation for Reinforcement Learning Agents

Acciaro Gennaro Daniele | Midterm **4**

Reinforcement learning is a reward-based ML training method with the goal of performing a single task. However, in real scenarios, many agents have to perform a number of different tasks, not just one.
The idea described in this paper is how to deal with several goals, maximizing the average success rate of the agent over all the possible goals, finding a single policy.
We then discriminate a particular subset of goals to focus on, and in order to improve performance we use, in addition to the goal dataset a GAN to generate new goals automatically. In particular, the GAN used is a modified version of Generative Adversarial Networks, which consists of two components: a (goal) **discriminator** and a (goal) **generator**, like any GAN.
The goal discriminator (noted as D) aims to assess whether a goal has an **appropriate level of difficulty**, given the current policy, and the goal generator (noted as G) aims to generate goals that meet this criterion. Furthermore, the goal generator, at each iteration, generate goals that are slightly more difficult than the goals that the agent already knows how to achieve.
The purpose of the algorithm is to perform training of a policy that maximizes rewards for a set of different goals.

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

where $R^g(\pi)$ is the probability of success given a goal g.
This formula, then, measures the average probability of success on all goals sampled from $p_{g(g)}$.

Automatic Goal Generation for Reinforcement Learning Agents: the algorithm

The algorithm is based on three steps:

1. A set of goals is labeled whether or not it has an appropriate level of difficulty given the current policy.
2. Using the labels found from step 1, a generator G is trained that can generate new goals
3. Using the goals generated by step 2, policy training is performed, maximizing the rewards.

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow$ `initialize_GAN()`
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow$ `sample_noise`$(p_z(\cdot))$
  $goals \leftarrow G(z) \cup$ `sample`$(goals_{\text{old}})$
  $\pi_i \leftarrow$ `update_policy`$(goals, \pi_{i-1})$
  $returns \leftarrow$ `evaluate_policy`$(goals, \pi_i)$
  $labels \leftarrow$ `label_goals`$(returns)$
  $(G, D) \leftarrow$ `train_GAN`$(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow$ `update_replay`$(goals)$
end for

For each iteration i, in order to generate a set of goals, the algorithm starts sampling noise from p_z (the noise distribution). This is the purpose of **sample_noise()**.
This noise is then used by the generator G(z) which generates goals.
These goals are used to perform policy training using Reinforcement Learning techniques using as a reward function the following formula that measures whether the agent has reached the goal.

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

where $\mathbb{I}$ is the indicator function, and

$$S^g = \{s_t : d(f(s_t), g) \leq \epsilon\}$$

where f() is a function that projects a state in goal space G, d is a distance metric in goal space, and ϵ is a threshold that establishes when the goal is reached.

The expected return gained when taking actions sampled by the policy can be seen as the probability of success on that goal within T time-step.

$$R^g(\pi) = \mathbb{E}_{\pi(\cdot|s_t, g)}\mathbb{I}\{\exists\, t \in [1 \dots T] : s_t \in S^g\} = \mathbb{P}(\exists t \in [1 \dots T] : s_t \in S^g | \pi, g)$$

This expected return is then used by the **update_policy**() function to update, indeed, the policy.

Automatic Goal Generation for Reinforcement Learning Agents: the algorithm

The algorithm, then, calculates the returns given the goals and the current policy via the **evaluate_policy()** function.
We now introduce two hyperparameters: R_{min} and R_{max}, whose values were set to 0.1 and 0.9, respectively.
So, now the algorithm needs to set the label of each goal by empirically estimating the expected return, given a goal, under the current policy π_i. The labels for each goal are assigned by the **label_goals()** function, where a single label y_g is defined as:

$$y_g = \mathbb{1}\{R_{min} \leq R^g(\pi_i) \leq R_{max}\}$$

For most goals g, given the current policy at iteration i (π_i), no reward is obtained due to the sparsity of the reward function.

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow$ `initialize_GAN()`
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow$ `sample_noise`$(p_z(\cdot))$
  $goals \leftarrow G(z) \cup$ `sample`$(goals_{\text{old}})$
  $\pi_i \leftarrow$ `update_policy`$(goals, \pi_{i-1})$
  $returns \leftarrow$ `evaluate_policy`$(goals, \pi_i)$
  $labels \leftarrow$ `label_goals`$(returns)$
  $(G, D) \leftarrow$ `train_GAN`$(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow$ `update_replay`$(goals)$
end for

The authors therefore limited the set of goals G into a set called **Goals of Intermediate Difficulty (GOID)**, which includes all goals that have a reward between R_{min} and R_{max}, because we want to focus on goals for which there is a minimum reward such that the agent can use to learn, $(R_g(\pi_i) > R_{min})$ but at the same time, we want to force the policy to perform training on goals that still require improvement $(R_g(\pi_i) \leq R_{max})$.
The GOID distribution is thus defined as follows:

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

Automatic Goal Generation for Reinforcement Learning Agents: the algorithm

Once we have obtained the binary labels $y_g \in \{0, 1\}$, which indicates which goals are part of the GOID set, we can use them to perform the training of the two parts that make up the GAN: goal generator and goal discriminator. Training is performed through the **train_GAN()** function.

The GAN used is a modified version that allows us to train the generative model either with positive examples (goals ∈ GOID) and negative examples (goals ∉ GOID).
Another reason for using GANs is that it can generate high-dimensional samples which is important in order to be able to generate goals in high-dimensional goal space.

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow$ `initialize_GAN()`
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow$ `sample_noise`$(p_z(\cdot))$
  $goals \leftarrow G(z) \cup$ `sample`$(goals_{\text{old}})$
  $\pi_i \leftarrow$ `update_policy`$(goals, \pi_{i-1})$
  $returns \leftarrow$ `evaluate_policy`$(goals, \pi_i)$
  $labels \leftarrow$ `label_goals`$(returns)$
  $(G, D) \leftarrow$ `train_GAN`$(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow$ `update_replay`$(goals)$
end for

Now then we can introduce three values **a**, **b** and **c**. **a** and **b** are the labels of the false and real data, respectively. **c** indicates the value that G wants D to believe for the false data.

$$\boldsymbol{a = -1} \qquad \boldsymbol{b = 1} \qquad \boldsymbol{c = 0}$$

At this point then we can introduce the following functions:

$$\boldsymbol{min_D\, V(D)} = \mathbb{E}_{\boldsymbol{g \sim p_{data}(g)}}\left[\boldsymbol{y_g}(\boldsymbol{D(g) - b})^2 + \boldsymbol{(1 - y_g)}(\boldsymbol{D(g) - a})^2\right] + \mathbb{E}_{\boldsymbol{z \sim p_z(z)}}\left[\left(\boldsymbol{D\left(G(z)\right) - a}\right)^2\right]$$

$$\boldsymbol{min_G\, V(G)} = \mathbb{E}_{\boldsymbol{z \sim p_z(z)}}\left[\left(\boldsymbol{D\left(G(z)\right) - c}\right)^2\right]$$

Where:
D(g) is the output for the discriminator on a real goal, G(z) is a fake goal generated using noise z, D(G(z)) is the output for the discriminator on a fake goal, p_z is the noise distribution and p_{data} is the distribution of the real data

Automatic Goal Generation for Reinforcement Learning Agents: the algorithm

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

$$\min_G V(G) = \mathbb{E}_{z\sim p_z(z)}\left[\left(D\left(G(z)\right)-c\right)^2\right]$$

The functions were obtained by modifying those proposed in the LSGAN framework, in order to train the discriminator to be trained even with "negative" goals (goals that are not in GOID).
In general, the discriminator is trained to distinguish goals ∈ GOID ($y_g = 1$) and goals ∉ GOID ($y_g = 0$).

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow$ `initialize_GAN()`
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow$ `sample_noise`$(p_z(\cdot))$
  $goals \leftarrow G(z) \cup$ `sample`$(goals_{\text{old}})$
  $\pi_i \leftarrow$ `update_policy`$(goals, \pi_{i-1})$
  $returns \leftarrow$ `evaluate_policy`$(goals, \pi_i)$
  $labels \leftarrow$ `label_goals`$(returns)$
  $(G, D) \leftarrow$ `train_GAN`$(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow$ `update_replay`$(goals)$
end for

Here we can see a major difference with regular GANs, because, in that context we attempt to perform the training of G by **minimizing** $\log(1 - D(G(z))$ while at the same time we perform the training of D by **maximizing** the probability of assigning the correct label for each example: whether the goal is generated by G or is part of the training example.

But, as pointed out by Mao et al. in the introductory paper of Least Squares GANs, regular GANs use a sigmoid cross entropy as loss function and this causes **gradient vanishing** problems during generator training. It is precisely for this reason that Mao et al. introduce LSGAN, **minimizing both the discriminator and the generator**. After that, this idea was slightly modified (from the authors of this paper) to allow training on negative goals as well.

Specifically: For positive goal g, for which $y_g = 1$, the second term disappears and only the first and third terms remain, which are identical to those in the original LSGAN framework. The discriminator is then trained on the distance between (D(g)) and the real goal label (b), as well as the distance between the discriminator's output on the noise-generated fake data (D(G(z)) and the fake goal label (a).
For negative goals ($y_g = 0$), the first term disappears but the main idea remains the same because we train the discriminator to recognize negative example as "fake", due to label a.

Results

In the paper, the results were evaluated by providing an answer to 4 questions.

1. Does our automatic curriculum yield faster maximization of the coverage objective?
2. Does our Goal GAN dynamically shift to sample goals of the appropriate difficulty (i.e. in $GOID_i$)?

To answer these two first questions, the authors used two tasks that, according to Duan et al. (2016), are not feasible to perform for standard RL methods. The first task consists of a quadruped agent must reach the goal, without any constraint, (which is represented as a red sphere in the figure). The second task has the same goal, but the agent is placed in a U-maze. Actuality, the authors further complicated the two tasks, and in both cases **the rewards are very sparse.**

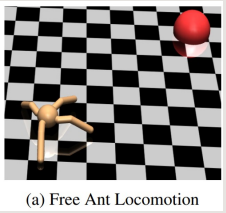
The Goal GAN described in this article was then compared with 4 other methods:

1. Uniform Sampling is a method perform the training at every iteration on goals uniformly sampled from the goal-space
2. Uniform Sampling with L2 loss sample which receives the negative L2 distance to the goal as a reward at every step
3. Asymmetric Self-play (Sukhbaatar et al., 2017)
4. SAGG-RIAC (Baranes & Oudeyer, 2013b)

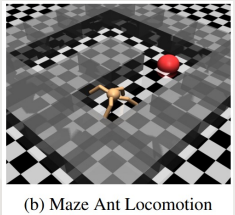
The last two methods were taken from the literature and, in addition, to better analyze the importance of GOID, other two models were used:

- **GAN fit all**, which performs training on both the Goals generated by the GAN and the Goals of the previous iteration.
- **Oracle** performs goal sampling uniformly from the feasible state space but keeping the GOID satisfied.

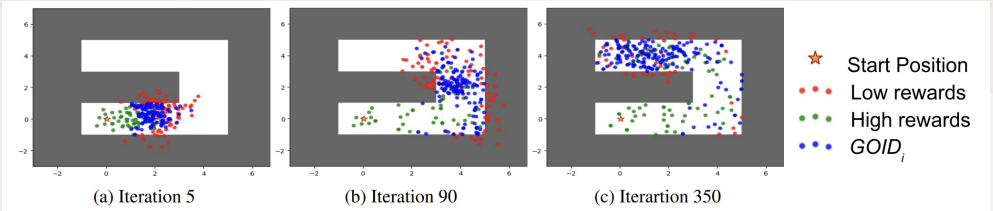
For the first two tasks described above, uniform sampling performs quite poorly because too many samples are wasted perform training on goals that are unfeasible. Uniform Sampling with L2 improves performance but gets stuck on a local minimum. The other two methods perform better than Uniform Sampling, but not as well as the model proposed by the authors of this paper, in fact looking at the figure on the right: it can see that the generated targets move to different parts of the maze, following the parts to be improved.



(a) Free Ant Locomotion



(b) Maze Ant Locomotion



(a) Iteration 5 (b) Iteration 90 (c) Iterartion 350

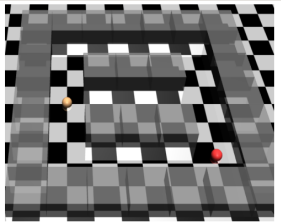
Results

In the paper, the results were evaluated by providing an answer to 4 questions.

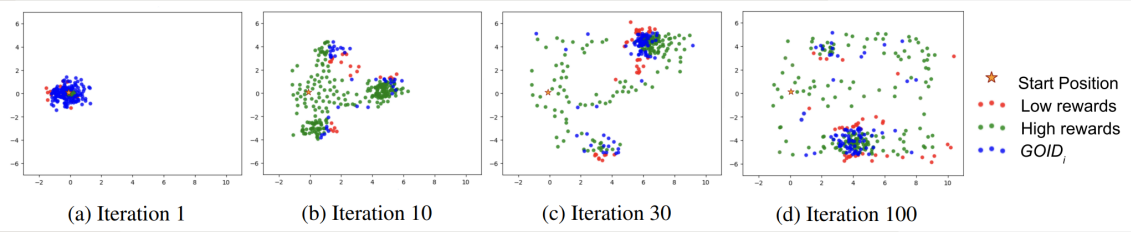
3. Can our Goal GAN track complex multimodal goal distributions $GOID_i$?
4. Does it scale to higher-dimensional goal-spaces with a low-dimensional space of feasible goals?

To answer the third question, we introduce a new maze environment with multiple paths, as can be seen in (d). For simplicity, the agent has been simplified with a single point. As in the other experiments, our aim is to learn a policy that can reach any feasible goal.
Looking at the figure in the center, we also note that the authors' proposed method is able to consider several goals simultaneously by tracking different areas.

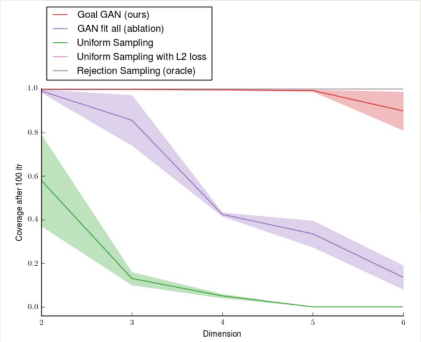
The last question deals with dimensionality, in fact, in real RL problems, feasible goals are in a lower-dimensional subset than the full state space, because there are the constraints of the environment.
So, to answer the question, the authors used a hypercube of dimensions [-5.5]^N and a Point Mass that is only able to move in a small subset of this space. Several experiments were performed, increasing the dimensionality and reducing the volume of the feasible space of the Point Mass, the purpose of this experiment is to evaluate how well the algorithm scales up to goals of higher dimensions.
As shown by the figure on the right, Uniform Sampling has bad performance as soon as the number of dimensions increases because the fraction of feasible states within the full state space decreases as the dimension increases. In contrast, the performance of Goal GAN proposed by the authors does not decay much when the state space dimension increases, because our Goal GAN always generates goals within the feasible portion of the state space.



(d) Multi-path point-mass



(a) Iteration 1 (b) Iteration 10 (c) Iteration 30 (d) Iteration 100



Conclusions

This work has shown us how a Generative Adversarial Network can also be used to generate goals for better policy training, than uniform sampling of goals, and this works even in a sparse reward scenario, without having any knowledge of the environment or the task to be performed.

Personally, I think that the use of a GAN could be a reason for instability, because performing the training of a GAN might already be an unstable operation. Also, it would be interesting to see the results of this methodology on other types of tests.
Finally, all the work is based on the assumptions in Section 3.3, it would have been helpful to know what can happen if one or more assumptions are violated.

Overall, I still think this is a well-written document that does not take almost all knowledge for granted.