AUTOMATIC GOAL GENERATION FOR REINFORCEMENT LEARNING AGENTS

IOMMI ANDREA 578212 - M.SC. IN ARTIFICIAL INTELLIGENCE

PROBLEM INTRODUCTION

- A weak aspect of classical reinforcement learning consists into train the agent by maximizing its reward function in order to execute well a certain single task. The **real-word** environment is more **complex** and a smart agent has to perform well **multiple tasks** as navigating to varying position in a room or moving object to varying location. The classical approach may be not enough for specific kind of situations.
- ***A Goal*** is reward functions defined with respect to target subsets of the state space.
- ***Automatic Goal Generation*** considers the problem of maximizing the average success rate of the agents over all possible goals, where success is defined as the probability of successfully reaching each goal of the current policy.

Goal Generative Adversarial Network (Goal G.A.N)

- The preliminary phase of each training step iteration "i" is composed of two components: ***Goal Generator*** which creates a goal with a pertinent level of difficulty of the current policy of the agent a time "i" and ***Goal Discriminator*** that is trained to evaluate if a goal is at the suitable for the policy of the agent.

- This approach ***incrementally*** creates a curriculum, in which at each step, the generator generates goals that are only slightly ***more difficult*** than the goals that the agent already knows how to achieve.

MODEL DEFINITION

Goal-parameterized Reward Functions

- In the classical RL we have an agent that lies on the state $s_t \in S \subset R^n$ and take an action $a_t \in A \subset R^m$ according to some policy $\pi(a_t|s_t)$ and receive a reward r_t r(s_t, a_t, s_{t+1}).

Instead to have a deal with only single reward function, we try to learn optimizing a set of reward functions r^g where $g \in G$ correlated to a set of state $S^g \subset S$. We want create a policy that $\forall g \in G$ perform optimally with respect to r^g.

- We define a S^g as set of states that can be achieve a goal g with ε tolerance , $S^g = \{s_t : d(f(s_t)) < \varepsilon\}$ where d is a metric function that measure the distance to the goal, f is a function that "transform" the states into a goal in goal-space.
- We define a reward function respect to g : $r^g(s_t, a_t, s_{t+1}) = \begin{cases} 1 \; if \; s_t \in S^g \\ 0 \; otherwise \end{cases}$
- Moreover the ***return*** is defined as $R_\pi^g = P(\exists\, t \in [1 \dots T] : s_t \in S^g \,|\, \pi, g)$. The probability of success on that goal within T time steps given the policy and goal g.

Key objective (a.k.a Coverage)

$\pi^*(a_t|s_t, g) = argmax\{\pi\}\left(\mathbb{E}_{g\sim p(.)} R^g(\pi)\right)$. We want to find a policy π^* that maximize the expectation on total rewards. For now we suppose that $p(g)$ samples uniformly from G.

METHOD

1. Label a set of goal based on the difficulty for the current policy.
2. Train the Goal Generator (with previous labels) to produce a new goal with a suitable level of difficulty.
3. Use the new goal to train the policy.

In few words, at first step we create a sort of database, that it's used in step two in order to create new goal, the latter it's used to train the policy.

(Iterate until converge)

Goal of Intermediate Difficulty (G.O.I.D)

- To improve the training instead of using $p_g(g)$ we sampling the goal over $Goid_i = \{g : R_{min} \leq R^g(\pi_i) \leq R_{max}\} \subset G$
- For some goals, due to the sparsity of the reward function, the policy at iteration *i* doesn't obtain reward.
- Hence we select goals that the agent a time "i" is able to receive some minimum expected return $R_{min} \leq R^g(\pi_i)$ and we also demand that $R^g(\pi_i) \leq R_{max}$ to impose the policy to train that still need improvement.
- *Train the policy with GOID maximize the Coverage.*
- Unusually $R_{min} \in (0,\ 0.25)\ and\ R_{max} \in (0.75,1)$

Ok but, How do we sample new goal *g* uniformly from GOID?

- We introduce an adversarial training procedure Goal GAN (modified version of GAN).
- The latter allows us to train the Generator with positive and negative examples, more precisely the positive examples are used for approximate the distribution that we want to obtain.
- Goal GAN (more in general the GAN) has the ability to generate very high dimensional samples for example an image. Others techniques e.g. *Stochastic neural networks* don't manage the negative examples and high dimensional space.

GOAL LEARNING

- Goal generator ***G(z)*** generate goals *g* from a noise vector *z* and we train it to return output goals g using Goal discriminator ***D(g)*** as adversarial.
- The Goal discriminator is trained to recognize if a given goal $g \in Goid_i$.
- $y_g = 1 \leftrightarrow g \in Goid_i$ otherwise 0
- $\min_D V(D) = \mathbb{E}_{g\sim p_{data}(g)}\left[y_g(D(g) - b^2) + (1 - y_g)(D(g) - a)^2\right] + \mathbb{E}_{z\sim p_z(z)}\left[(D(G(z)) - a)^2\right]$
- $\min_G V(G) = \mathbb{E}_{z\sim p_z(z)}\left[(D(G(z)) - c)^2\right]$
- Negative examples when $y_g = 0$. With hyperparameters $a = -1, b = 1\ and\ c = 0$
- For positive examples we have $\mathbb{E}_{g\sim p_{data}(g)}\left[y_g(D(g) - b^2) + \mathbb{E}_{z\sim p_z(z)}\left[(D(G(z)) - a)^2\right]\right.$ the discriminator is trained to discriminate between goals from $p_{data}(g)$ with a label $y_g = 1$ and the generated goals *G(z)*.
- For negative $\mathbb{E}_{g\sim p_{data}(g)}[(D(g) - a)^2] + \mathbb{E}_{z\sim p_z(z)}\left[(D(G(z)) - a)^2\right]$ the generator is trained to "fool" the discriminator, i.e. to output goals that match the distribution of goals $g\sim p_{data}(g)$ for which $y_g = 1$.

Policy Optimization (for each iteration i)

1. Generate the goals with G(z) where z is a noise vector sampled from $p_z(.)$
2. Use these goals to train the policy, with the reward function on slide 3
3. Use policy's empirical performance to determine each goal's label y_g
4. These labels is used to train the Goal GAN.
5. Compute the MC estimate with generated goal
6. Finally, the method finds a policy that optimizes the objective by training on goals within $Goid_i$.

The goal generator is updated along with the policy to generate goals in $Goid_i$, for which our current policy "i" obtains an intermediate level of return. Hence goals are always at the suitable level of difficulty.

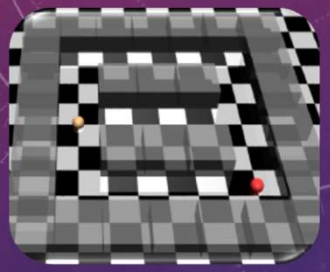
Algorithm 1 Generative Goal Learning

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

Experimental Results (robotic locomotion tasks)

No constraint

U Maze



Multi path maze

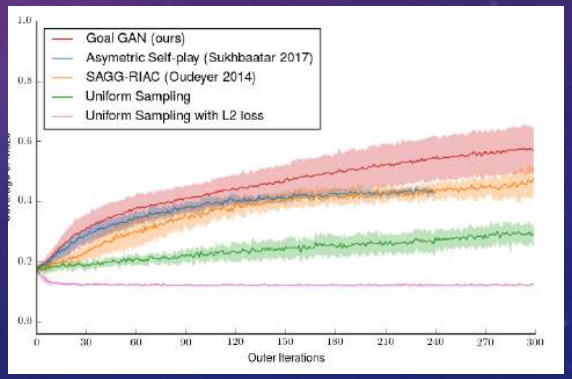
- We use three environment, 1)the agent with no constraint 2) U-maze, 3) Multi path maze.

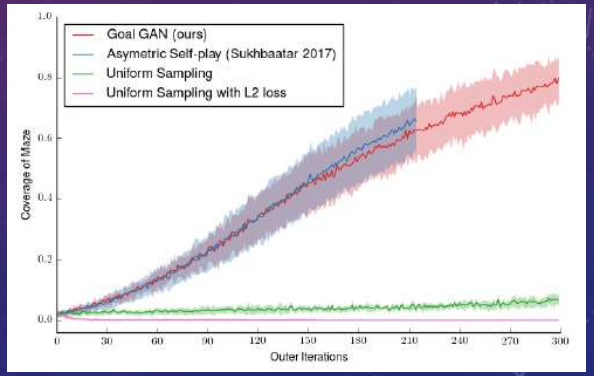
We analyze the performance of the agent (policy) using different methods

- ***Goal GAN*** (using a Generative Adversarial Network)
- ***Uniform sampling*** (doesn't use curriculum but at every iteration sample uniformly from the goal-space)
- ***Uniform Sampling with L2 loss*** (as before but receives the negative L2 distance to the goal as a reward at every step)
- ***Asymmetric Self-play and SAGGRIAC***

  - The **Uniform Sampling** baseline **doesn't perform** well because too many samples are wasted attempting to train on goals not reachable by the current policy (not receiving any learning signal).
  - **The L2 agent falls** into a poor **local optima** of not moving to avoid further negative rewards.
  - The two other methods perform better, but still do not surpass Goal GAN.

U Maze





No constraint

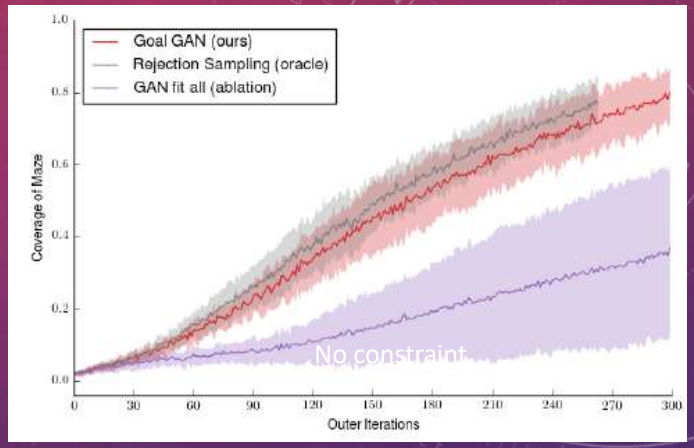
*To experimental result was added an **ablation** and an **oracle** for the method to better understand the importance of sampling goals with consistent level of difficulty* $g \in Goid_i$.

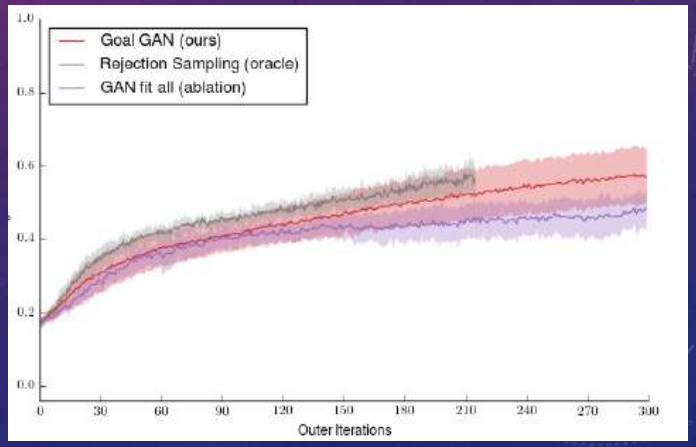
- **The ablation GAN fit all** consists into training the GAN also with goals that are attempted in the previous iteration.
- **Rejection sampling (oracle)**, instead, consists in sampling goals uniformly from the feasible state-space, that satisfy $Goid_i = \{g : R_{min} \leq R^g(\pi_i) \leq R_{max}\}$. It serves to estimate an upper-bound for the method in terms of performance.

The ablation performs worse, because the expansion of the goals is not related to the current performance of the policy.

Rejection Sampling this method is orders of magnitude more sample inefficient, but gives us an upper bound on the performance of our method, the showed charts demonstrate that Goal GAN performance is quite close to the performance of this other approach.

No constraint

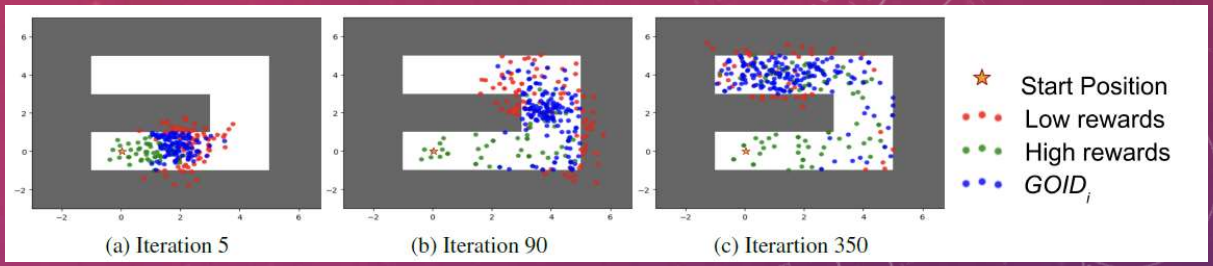




U Maze

Multi-path point-mass maze

The Goal GAN were tested in a more complex maze environment with multiple path achieving as well as before.



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

It can be observed that the method track all the areas where goals are at the appropriate level of difficulty.

CONCLUSION

- We have defined a new approach to RL where the agent will be not train to execute a single task but through the new method for automatic curriculum generation, it will learn to perform more task a the same time.
- The curriculum is created without any prior knowledge of the environment. We use Goal GAN to automatically generate goals for the policy that are always at the consistent with the level of difficulty.