ATCI: Reinforcement Learning

Sample Efficiency II: Exploration, Curriculum Learning and Hierarchical Learning

Mario Martin

CS-UPC

April 25, 2024

Motivation

Sample efficiency

- In previous lecture we saw how to build a model of the world to reduce number of interactions with the environment.
- Other ways to deal with the problem, specially when we have **sparse rewards**.
- We talk about sparse reward when the agent has positive reward only for few states (goal states).
- In this case the agent don't get rewards until it finds that goal state.
- We will focus on three point today:
  - Exploration
  - Hindsight Experience Replay
  - Curriculum learning

Exploration

Exploration

- We already know the importance of exploration in order to improve the policy.
- We have seen at least two methods of exploration, ϵ-greedy and Boltzman exploration
- But let's start from the beginning introducing multi-armed bandits

Subsection 1

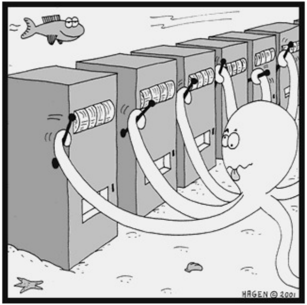
Multi-armed bandits framework

Multi-armed bandits

- Multi-armed bandit is a tuple of $(\mathcal{A}, \mathcal{R})$
  - $\mathcal{A}$: known set of m actions (arms)
  - $\mathcal{R}^a(r) = \mathbb{P}[r \mid a]$ is an unknown probability distribution over rewards
- At each step t the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- Goal: Maximize cumulative reward $\sum_{\tau=1}^{t} r_\tau$

Multi-armed bandits

- Multi-armed bandit is a tuple of $(\mathcal{A}, \mathcal{R})$
  - $\mathcal{A}$: known set of m actions (arms)
  - $\mathcal{R}^a(r) = \mathbb{P}[r \mid a]$ is an unknown probability distribution over rewards
- At each step t the agent selects an action $a_t \in \mathcal{A}$
- The environment generates a reward $r_t \sim \mathcal{R}^{a_t}$
- Goal: Maximize cumulative reward $\sum_{\tau=1}^{t} r_\tau$



How to proceed?

- Obviously selecting the more promising bandit (exploitation)
- But are we sure that the bandit I think is the more promising is the best?
- Greedy can lock onto suboptimal action, forever!

How to proceed?

- Obviously selecting the more promising bandit (exploitation)
- But are we sure that the bandit I think is the more promising is the best?
- Greedy can lock onto suboptimal action, forever!
- We have to try also other bandits to be sure! (exploration)
- Constraint: we want not to explore more than necessary
- Some procedures to balance exploration with exploitation:
  - ϵ-greedy
  - Optimistic
  - Upper Confidence Bound
  - Thomson Sampling

ϵ-greedy

- You know: choose greedy action with probability 1-ϵ and choose random action with prob. ϵ
- You choose *always* suboptimal action with probability ϵ

ϵ-greedy

- You know: choose greedy action with probability 1-ϵ and choose random action with prob. ϵ
- You choose *always* suboptimal action with probability ϵ
- May be this is necessary at the beginning of learning, but no when learning is advanced
- So, may be better to start with high exploration parameter and reduce it with time: decaying $\epsilon(t)$

$$\epsilon(t) = 1/t$$

$$\epsilon(t) = 1/e^t$$

ϵ-greedy

- You know: choose greedy action with probability 1-ϵ and choose random action with prob. ϵ
- You choose *always* suboptimal action with probability ϵ
- May be this is necessary at the beginning of learning, but no when learning is advanced
- So, may be better to start with high exploration parameter and reduce it with time: decaying $\epsilon(t)$

$$\epsilon(t) = 1/t$$

$$\epsilon(t) = 1/e^t$$

- Hmm. So how decrease? At which rate? Not easy to answer.
- In addition, there should be an exploration at the end?
- On the positive side, better than greedy and easy to implement

Optimistic initialization

- Assume deterministic reward function
- Repeat the following procedure:
  - Initialize expected return higher than true return
  - Choose always greedily.
  - Recompute estimated return from result
    $\hat{Q}_t(a_t) = \hat{Q}_{t-1} + \frac{1}{N_t(a_t)}\left(r_t - \hat{Q}_{t-1}\right)$
- Expectation is decreased up to actual reward for each arm
- When an arm has been chosen and it does not decrease expected reward, means that it is optimal.
- Does not work well when reward is a random variable
- But can we do use this intuition of optimistic choosing of actions (greedy) without ending in sub-optimal estimations?

Upper Confidence Bound

- Let's build an estimation of expected reward as a mean and an uncertainty about the mean for each arm
- Now choose actions greedily. You will learn always something
- Two outcomes:
  - Getting high reward: if the arm really has a high mean reward
  - Learn something: if the arm really has a lower mean reward, pulling it will (in expectation) reduce its average reward and the uncertainty over its value

Upper Confidence Bound

- Estimate an upper confidence $U_t(a)$ for each action value, such that $Q(a) \leq U_t(a)$ with high probability
- This depends on the number of times $N_t(a)$ action a has been selected
- Select action maximizing Upper Confidence Bound (UCB)

$$a_t = \arg\max_{a \in A} [Q(a) + U_t(a)]$$

Upper Confidence Bound (UCB)

- *Hoeffding's Inequality*: Let $X_1, \ldots, X_t$ be i.i.d. random variables in $[0,1]$. The sample mean is $\bar{X}_t = \frac{1}{t}\sum_{\tau=1}^{t} X_\tau$ Then for $u > 0$, we have:

$$\mathbb{P}\left[\mathbb{E}[X] > \bar{X}_t + u\right] \leq e^{-2tu^2}$$

- Applying to Bandits: action a, $r_t(a)$ as the random variables, $Q(a)$ as the true mean, $\hat{Q}_t(a)$ as the sample mean, And u as the upper confidence bound, $u = U_t(a)$. Then we have,

$$\mathbb{P}\left[Q(a) > \hat{Q}_t(a) + U_t(a)\right] \leq e^{-2tU_t(a)^2} = p$$

- Let's reorganize and set $U(a)$ in terms of p:

$$e^{-2tU_t(a)^2} = p \ \text{ Thus, } \ U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

Upper Confidence Bound (UCB)

$$e^{-2tU_t(a)^2} = p \;\; \text{Thus,} \;\; U_t(a) = \sqrt{\frac{-\log p}{2N_t(a)}}$$

- One heuristic is to reduce p with time. Set $p = t^{-4}$ we get **UCB1** algorithm:

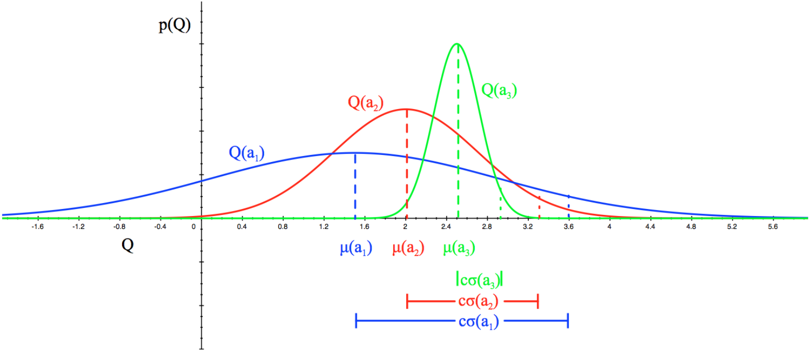
$$U_t(a) = \sqrt{\frac{2\log t}{N_t(a)}}$$

- So, algorithm UCB1 is:
  - Choose each action one time to initialize values
  - Repeat forever: Chose action according to:

$$a_t^{UCB1} = \arg\max_{a \in \mathcal{A}} Q(a) + \sqrt{\frac{2\log t}{N_t(a)}}$$

Upper Confidence Bound (UCB)

- Hoeffding's Inequality works with any distribution (good) but it is not tight (bad)
- If we know kind of reward distribution we can obtain better bounds.
- For instance: Gaussian distribution with $\mu(a_i), \sigma(a_i)$, then $c\sigma(a_i)$ is upper confidence bound, where c is a adjustable.



Upper Confidence Bound (UCB)

- Assuming $\mathcal{R}_a(r) = \mathcal{N}\left(r; \mu_a, \sigma_a^2\right)$:

$$a_t = \arg\max_{a \in \mathcal{A}} \mu_a + c \frac{\sigma_a}{\sqrt{N(a)}}$$

- In Normal distributions, bounds and estimation of parameters is easy.
- In other distributions, update of parameters of the distribution can be done using Bayesian inference

Thomson Sampling

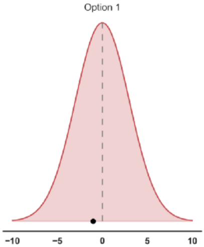
- At each time step, we want to select action a according to the probability that a is optimal:

$$\pi\left(a \mid h_t\right)=\mathbb{P}\left[Q(a)>Q\left(a^{\prime}\right), \forall a^{\prime} \neq a \mid h_t\right]$$

- where $\pi\left(a \mid h_t\right)$ is the probability of taking action a given the history h_t.
- Thomson Sampling: At every time-step, we draw one sample from each distribution and we pick the highest-ranked option.
- Update parameters of distributions accordingly

Thomson Sampling

- Again, use Bayesian inference to update parameters of distribution
- Intuition with normal distributions of reward:



Multi-armed bandits

- Several ways to smartly balance exploration and exploitation
- Applied to a lot of scenarios: Ad-click, Medical treatments, Recommendation systems
- They do not introduce the idea of state.

Multi-armed bandits

- Several ways to smartly balance exploration and exploitation
- Applied to a lot of scenarios: Ad-click, Medical treatments, Recommendation systems
- They do not introduce the idea of state. *Contextual bandits* introduce the idea of state but still they are *one-shot*, i.e., final reward is obtained after one action execution
- Some of them need a guess about distribution
- Need to store number of tries to each arm
- In general not applicable in standard RL
- Lesson of *optimism under uncertainty*: Assume that not optimal actions according to data can be still optimal.
    - Adding a small bonus in selection ($U(a)$ or by sampling) that depends on visits and data

Subsection 2

Exploration in general framework

Issues in Exploration

- Differently that in Bandits we have:
  - States (usually very large space states)
  - Sometimes sparse reward
  - Function approximation
  - Long-term reward (versus one-shot final reward)
- Can we apply some lesson from Bandits? Yes. Bonus idea in selection of actions

Intrinsic reward

- Augment the reward with an additional (vanishing) reward term

$$r_t^+ = \underbrace{r_t^e}_{\text{extrinsic reward (standard)}} + \beta \underbrace{r_t^i}_{\text{intrinsic}}$$

  r^e : extrinsic reward (task reward) r^i : **intrinsic reward** (exploration bonus)

- Run any algorithm using the new reward r_t^+

Intrinsic reward

- How to define the intrinsic reward bonus? Several options:
  - Discover new states
  - Improve knowledge
  - Improve controlability
  - ...

Intrinsic reward

- How to define the intrinsic reward bonus? Several options:
  - Discover new states
  - Improve knowledge
  - Improve controlability
  - ...
- Arbitrary classification of approaches:
  - Count-based bonus
  - Prediction-based bonus

Count-based approaches

- From Bandits we know that number of visits is important to have reliable information
- In large state spaces we cannot count visits to states and action taken there, so we have to estimate a "proxy" for the number of visits $\tilde{N}(s_t)$
- Add an exploration bonus to the rewards

$$\widetilde{r}_t^+ = r_t + \beta_t \sqrt{\frac{1}{\widetilde{N}(s_t)}}$$

so $r_t^e \approx \sqrt{1/\tilde{N}(s_t)}$ is inspired by theory (recall UCB)

- Run any DeepR L algorithm on $\mathcal{D}_t = \left\{\left(s_i, a_i, \widetilde{r}_i^+, s_{i+1}\right)\right\}$

Count-based approaches

- Count by Density Estimation (Bellemare et al. 16) : estimate density of visits on states using any density estimation alg. and moves from density estimation to count to apply intrinsic reward
- Algorithm:
  1. fit model $p_\theta(\mathbf{s})$ to all states $\mathcal{D}$ seen so far
  2. take a step i and observe $\mathbf{s}_i$
  3. fit new model $p_{\theta'}(\mathbf{s})$ to $\mathcal{D} \cup \mathbf{s}_i$
  4. use $p_\theta(\mathbf{s}_i)$ and $p_{\theta'}(\mathbf{s}_i)$ to estimate $\hat{N}(\mathbf{s})$
  5. set $r_i^+ = r_i + \mathcal{B}(\hat{N}(\mathbf{s}))$
  6. Go back to 1

Count-based approaches

- how to get $\hat{N}(s)$? use the equations

$$p_\theta(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i)}{\hat{n}} \quad p_{\theta'}(\mathbf{s}_i) = \frac{\hat{N}(\mathbf{s}_i) + 1}{\hat{n} + 1}$$

- two equations and two unknowns!

$$\hat{N}(\mathbf{s}_i) = \hat{n} p_\theta(\mathbf{s}_i) \quad \hat{n} = \frac{1 - p_{\theta'}(\mathbf{s}_i)}{p_{\theta'}(\mathbf{s}_i) - p_\theta(\mathbf{s}_i)} p_\theta(\mathbf{s}_i)$$

- Density estimation procedure is essential.

Count-based approaches

- (Tang et al. 17) use *locality-sensitive hashing* (LSH) to implement counting
  - We still count states (images) but not in pixel space, but in latent compressed space.
  - Compress s into a latent code, then count occurrences of the code.
  - How do we get the image encoding? E.g, using autoencoders
  - How to count states? Count on discrete hashed-states (LSH)
- There is no guarantee such reconstruction loss will capture the important things that make two states to be similar or not policy wise

Prediction-based Exploration

- Computational Curiosity idea: let's explore to improve skills
- Look for novelty and surprises
- One way to do that is by executing behaviors that reduce uncertainty on how the world works looking for novelty and surprises

Prediction-based Exploration

- Computational Curiosity idea: let's explore to improve skills
- Look for novelty and surprises
- One way to do that is by executing behaviors that reduce uncertainty on how the world works looking for novelty and surprises
- Yes! It implies a world model like we saw in previous lecture

Prediction-based Exploration

- Incentivizing exploration in reinforcement learning with deep predictive models (Stadie et. al 15) proposes to add as bonus the error in prediction
- Given an encoding $\phi(s)$, learn a prediction model

$$f : (\phi(s_t), a_t) \mapsto \phi(s_{t+1})$$

- Use the prediction error

$$e_t = \|\phi(s_{t+1}) - f(\phi(s_t), a_t)\|_2^2$$

as exploration bonus $r_t^i \propto e_t$ (normalized and scaled)

Prediction-based Exploration

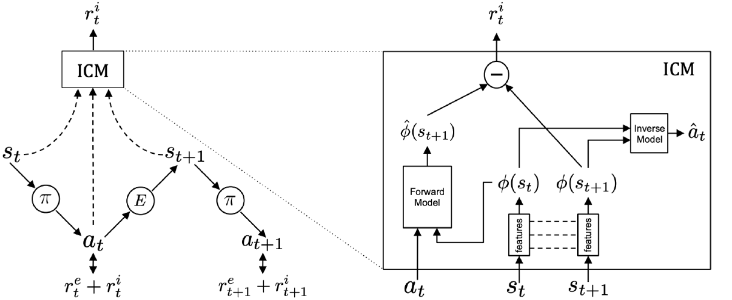
- However is difficult to predict every possible change in the transitions and may be not necessary
- Yes, for instance the predictions that do not depend on agents actions
- Example: The TV problem with random images
  - Agent cannot predict what she will see on TV
  - So TV has a lot of novelty (and error prediction)
  - And the agent gets stuck behind the TV trying to learn a model that it cannot control!

Prediction-based Exploration

- **Curiosity-driven Exploration** (Pathak et al. 17) predict only changes that depend on agent's actions, ignore the rest!

Inverse dynamics: $h : (\phi(s_t), \phi(s_{t+1})) \mapsto \widehat{a}_t$
Forward dynamics: $f : (\phi(s_t), a_t) \mapsto \widehat{\phi}(s_{t+1})$

Intrinsic reward:
$r_t^i = \|\widehat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$

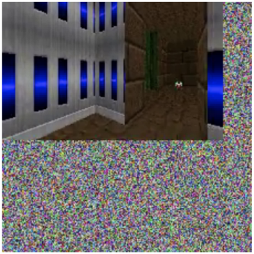


Prediction-based Exploration

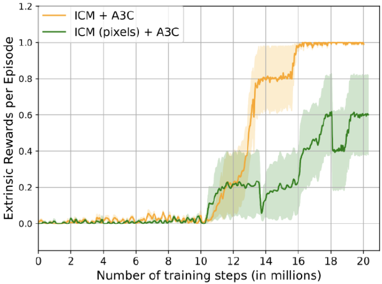
- In ICM the features of the state depend on the inverse model
- Loss function considers both models:

$$\min_{\theta_P,\theta_I,\theta_F} \left[-\lambda \mathbb{E}_{\pi(s_t;\theta_P)}\left[\Sigma_t r_t\right] + (1-\beta)L_I + \beta L_F\right]$$

- As TV is not controllable by the agent, the model will be blind to the features of the TV



(b) Input w/ noise

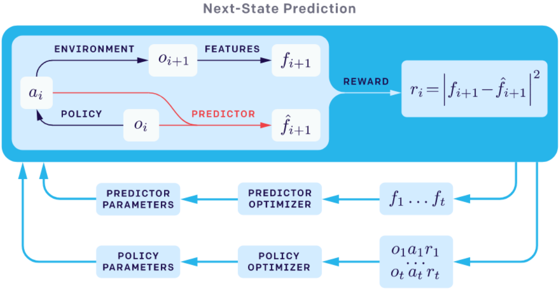


Prediction-based Exploration

- Expl. by **Random Network Distillation** (Burda et a. 18)
- Authors distinguish three kinds or errors in previous models
  1. Prediction error is high where the predictor fails to generalize from previously seen examples. Novel experience then corresponds to high prediction error.
  2. Prediction error is high because the prediction target is stochastic.
  3. Prediction error is high because information necessary for the prediction is missing, or the model class of predictors is too limited to fit the complexity of the target function.
- First is related to exploration, the other no.
- Authors propose to solve the TV problem by comparing difference prediction in results for next state for a learning NN and output of **Random fixed** NN with same architecture and input.

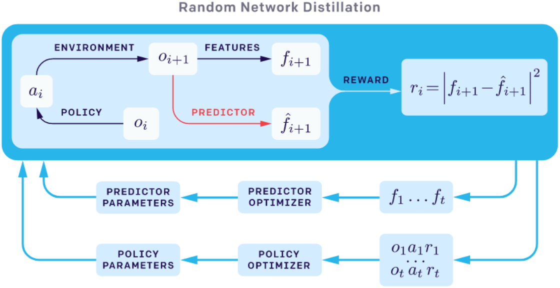
Prediction-based Exploration

- Compare usual setting



Prediction-based Exploration

- With new proposal



Prediction-based Exploration

- Randomly initialize two instances of the same NN (target θ_* and prediction θ_0)

$$f_{\theta_*} : \mathcal{S} \to \mathbb{R}; \quad f_\theta : \mathcal{S} \to \mathbb{R}$$

- Train the prediction network minimizing loss w.r.t. the target network

$$\theta_n = \arg\min_\theta \sum_{t=1}^{n} \left(f_\theta\left(s_t\right) - f_{\theta_*}\left(s_t\right)\right)^2$$

- Build "intrinsic" reward

$$r_t^i = \left|f_\theta\left(s_t\right) - f_{\theta_*}\left(s_t\right)\right|$$

- No model misspecification (f_θ can exactly predict f_{θ_*})

Prediction-based Exploration

- Idea behind is if similar states have been visited many times in the past, the prediction should be easier and thus has lower error
- So measure computes indirectly "pseudo-count" of visits
- In addition, not so hard to learn like a predictive model
- Normalization of bonus is important and tricky (see implementation details in paper)

Conclusions

- Exploration is key for fast and efficient learning. Random exploration is not a good idea.
- Some tasks cannot be solved without smart exploration techniques because of sparse reward and/or large state space
- A lot of imaginative possibilities in RL that can be combined with World Models, and other techniques we will see in next lecture
  - Build a world model and only intinsic reward for better learn the policy
  - Plan2Explore paper (Sekar et al. 20)
- Introductory references:
  - Nice intuitive and complete review of latest exploration methods
  - Survey paper on intrinsic motivation

Conditioned policies and Hindsight

Universal Markov Decision Processes

- Universal Markov Decision Processes:

$$< \mathcal{S}, \mathcal{A}, \mathcal{G}, P, R >$$

- Now policy to learn is *goal conditioned* and also *universal*, that is, able to solve any goal in $\mathcal{G}$

$$\pi : \mathcal{S} \times \mathcal{G} \longrightarrow \mathcal{A}$$

- Q-values are then also dependent of goal

$$Q^{\pi}(s, a, g) = \mathbb{E}_{\pi}\left[\sum_{k=0} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a, g_t = g\right]$$

Universal Markov Decision Processes

- The kind of problems where this can be applied is where reward is positive for getting goal state and 0 otherwise (so sparse)
- Why are them important?

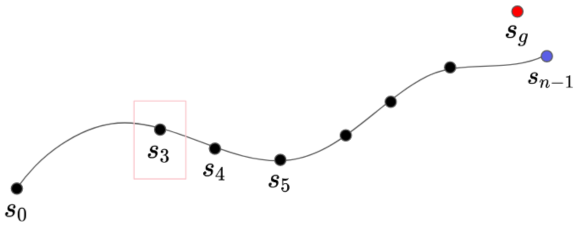
Universal Markov Decision Processes

- The kind of problems where this can be applied is where reward is positive for getting goal state and 0 otherwise (so sparse)
- Why are them important? Idea is that data collected to solve one task may help to solve or speed up the learning of another in the same domain
- But how to transfer this knowledge?
- Sharing Experience Replay? (s,a,s',r)

Hindsight Experience Replay

- In sparse domains, trials usually fail to achieve goal
- So data in experience replay is full of failures and learning is impossible
- But a failure of trial ending in one state for going to a goal state, is a success for a trial going to that final state!
- This is behind Hindsight Experience Replay (HER) (Andrychowicz et al. 17)

HER (Andrychowicz et al. 17)



$$(s, a, g, r, s')$$

- When $g \neq s'$, $r = 0$, but when $g = s'$, then $r = 1$
- So, add some successes in ER with final states of trajectories

HER (Andrychowicz et al. 17)

Algorithm 1 Hindsight Experience Replay (HER)

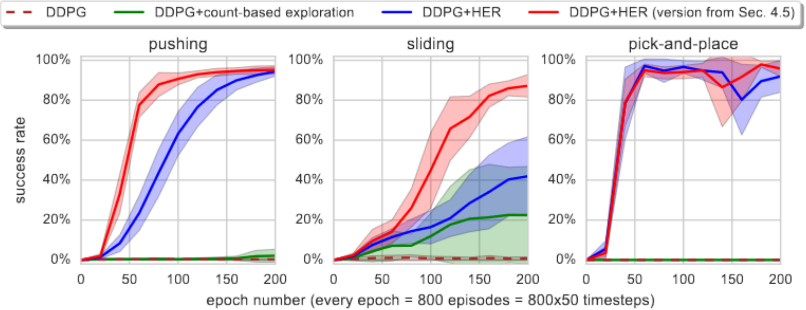
Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode $= 1, M$ **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ || denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g,\ a_t,\ r_t,\ s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}($**current episode**$)$
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g',\ a_t,\ r',\ s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

HER (Andrychowicz et al. 17)

- Very good performance even when goal is fixed
- Concatenation of goals in states allow generalization to similar goals and speed up in learning
- It can be used also when learning several tasks at the same time
- Beats exploration methods

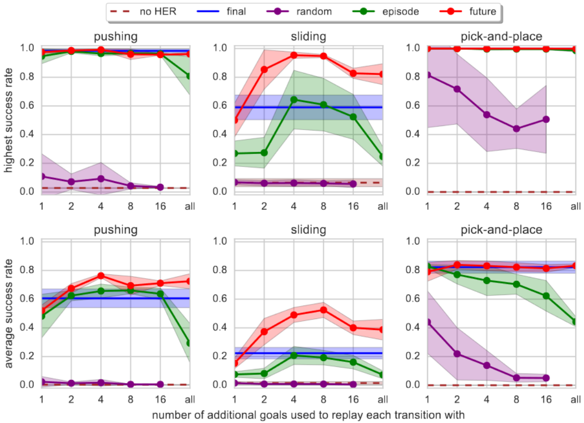


HER (Andrychowicz et al. 17)

- Several strategies to choose additional goals:
  - **final** replay for final state in trajectory
  - **future** replay with k random states which come from the same episode as the transition being replayed and were observed *after* it
  - **episode** replay with k random states coming from the same episode as the transition being replayed,
  - **random** replay with k random states encountered so far in the whole training procedure

HER (Andrychowicz et al. 17)

- Future strategy seems to work better (standard)



HER (Andrychowicz et al. 17)

- Widely used in general sparse reward tasks.
- Not clear about how to generalize to tasks that are not sparse in rewards
- But may be not bad in some cases?
- Nice explanation of HER with examples of universal policy in Lunar Lander
- Notice that reward function of Lunar Lander has changed to be sparse (so no shaping needed!)

Curriculum learning

Intuition and motivation

- Some tasks are hard to solve (sparse and large state actions)
- Curriculum learning propose subgoals to be learnt in sequence (*curriculum*) to solve a complex tasks.
- Introducing gradually more difficult examples speeds up or even allow online training.
- Focus on "interesting" examples that are neither too hard or too easy: You can only learn something that you are ready to learn
- Inspired in animal and human learning
- Example, learn to solve Rubik Cube

Curriculum learning

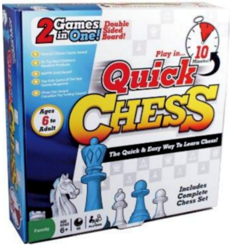
- Usually involves 3 steps:
  - Design of tasks in the curriculum
  - Design a metric to quantify how hard a task is so that we can sort tasks accordingly.
  - Provide a sequence of tasks with an increasing level of difficulty to the model during training.

Curriculum learning

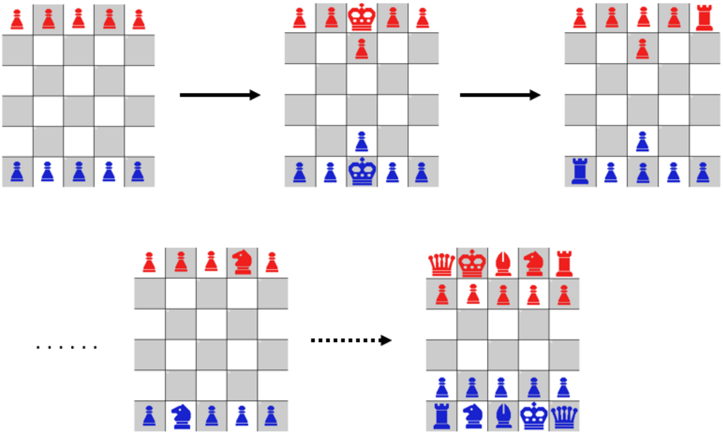
- Usually involves 3 steps:
  - Design of tasks in the curriculum
  - Design a metric to quantify how hard a task is so that we can sort tasks accordingly.
  - Provide a sequence of tasks with an increasing level of difficulty to the model during training.
- It can be done manually but better if we build the curriculum automatically!

Curriculum learning

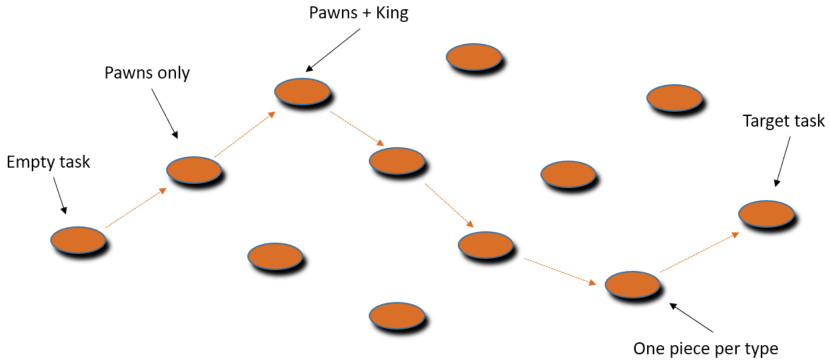
- (Narvekar et al. 16) proposes to formalize a curriculum as graph of tasks sorted by difficulty
- For instance the *Quick chess* game for humans:



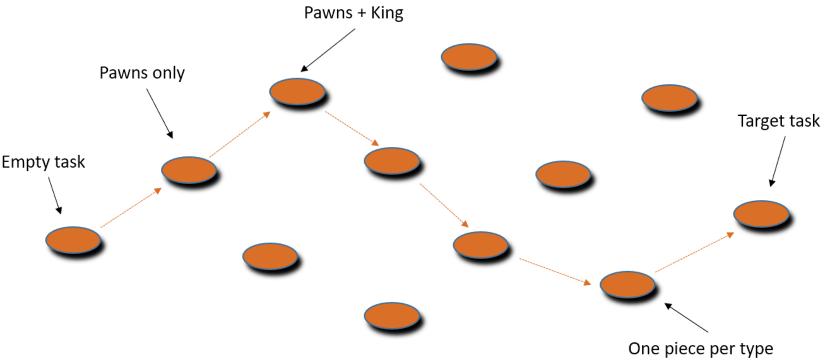
Curriculum learning



Curriculum learning



Curriculum learning

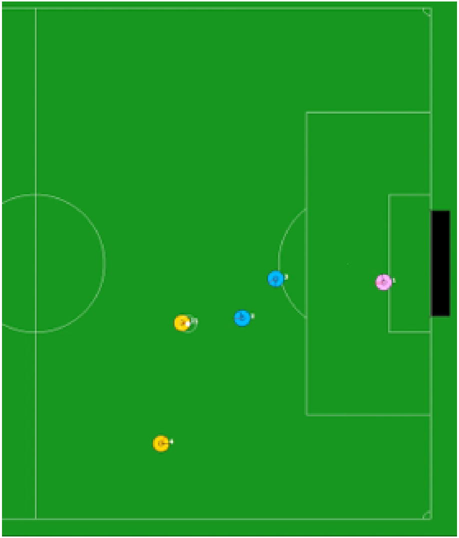


- In order to work for machines we need:
  - Create the tasks
  - Propose a sequence in the graph
  - Agent be able to do transfer of learning

Curriculum learning

- How to create tasks?
- Each task is an MDP with $< S, A, R, P >$
- A modification of these parameters is a different tasks
- (Narvekar et al. 16) propose simplification of target tasks using the following methods
  - Task Simplification: It consists in reducing the number of states, actions or transitions to make the problem easier
  - Promising Initializations: Change distribution of initial or final states to make the problem easier
  - Mistake Learning: Create subtasks to avoid or correct mistakes
  - Option-based Subgoals
  - Task-based Subgoals
  - Composite Subtasks

Half Field Offense

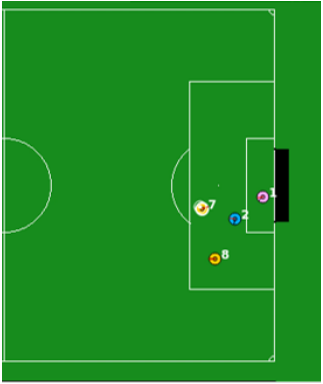


Curriculum proposal 1: Shoot Task

- Initially, goal scoring episodes are rare

Curriculum proposal 1: Shoot Task

- Initially, goal scoring episodes are rare
- Promising initialization: Start close to the goal
- Task simplification: Start with only 2 opponents



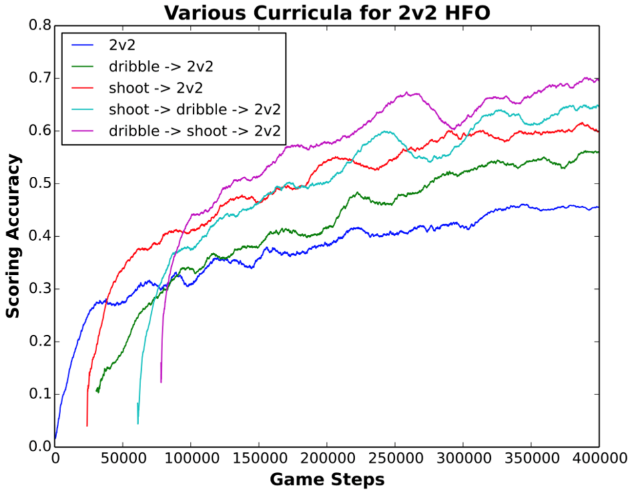
Curriculum proposal 2: Dribble Task

- Agent takes too many shots from far away

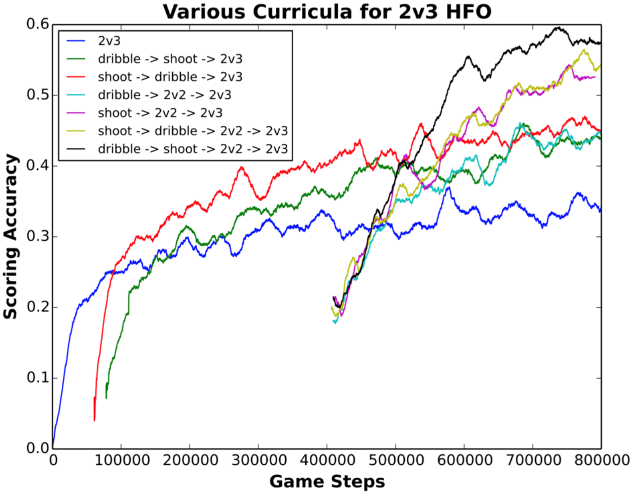
Curriculum proposal 2: Dribble Task

- Agent takes too many shots from far away
- Skill proposal: move the ball up the field while maintaining possession, until a shot is likely to score
- Task simplification: Start with only 2 opponents

Half Field Offense: Results 2 vs 2



Half Field Offense: Results 2 vs 3



Curriculum learning

- We have seen what a curriculum is
- We have seen how to create tasks
- We have seen that curriculum learning is effective

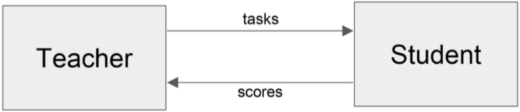
Curriculum learning

- We have seen what a curriculum is
- We have seen how to create tasks
- We have seen that curriculum learning is effective

- But, can we create tasks of the curriculum automatically?
- Can we propose tasks to the learner agent effectively?
- Topic of research. We will see now several approaches

Teacher-Guided Curriculum idea

- In (Graves, et al. 2017), the Teacher-Guided Curriculum idea is proposed
- It consists on TWO agents that should learn together: The teacher and the Student
  - **Teacher** agent has to propose the right task to the student
  - **Student** try to solve the task proposed by the teacher

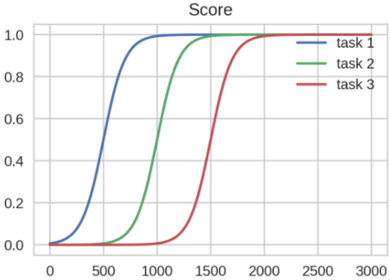


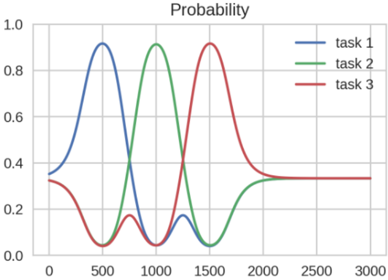
Teacher-Guided Curriculum idea

- In this approach, **task are previously created**, so we focus on how to propose tasks to the learner at each time step
- Authors proposed Teacher agent to be trained using a *n-armed bandit* algorithm where *n* is the number of tasks
- Return of the armed-bandit is based on the success of the student

Teacher-Guided Curriculum idea

- TGCL idea was formalized in (Matiisen, et al. 2017)
- Teacher should propose tasks that the agent still not learn but is ready to learn





Teacher-Guided Curriculum

- Training the teacher model is to solve a POMDP problem:
  - The unobserved s_t is the full state of the student model.
  - The observed $o = \left(x_t^{(1)}, \ldots, x_t^{(N)}\right)$ are a list of scores for N tasks.
  - The action a is to pick on subtask.
  - The reward per step is the score delta. $r_t = \sum_{i=1}^{N} x_t^{(i)} - x_{t-1}^{(i)}$
- Teacher can trained using Thomsom sampling or similar (one-shot learning)

Teacher-Guided Curriculum

- Training the teacher model is to solve a POMDP problem:
  - The unobserved s_t is the full state of the student model.
  - The observed $o = \left(x_t^{(1)}, \ldots, x_t^{(N)}\right)$ are a list of scores for N tasks.
  - The action a is to pick on subtask.
  - The reward per step is the score delta. $r_t = \sum_{i=1}^{N} x_t^{(i)} - x_{t-1}^{(i)}$
- Teacher can trained using Thomsom sampling or similar (one-shot learning)

- Nice idea of two agents learning!
- But still no proposal of task (they are proposed at the begining)

Self-Play Curriculum (Sukhbaatar, et al. 2018)

- In *Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play* authors propose again two agents with different goals
- Alice (teacher) challenges Bob (student) to achieve the same state and Bob attempts to complete it as fast as he can.
- Two kinds of episodes
  - *self-play episode*: Alice alters the state from s_0 to s_t and then Bob is asked to return the environment to its original state s_0 to get an internal (intrinsic) reward.
  - *target task episode*: Original goal of the agent

Self-Play Curriculum (Sukhbaatar, et al. 2018)

- Policies of both agents are goal-conditioned
- Intrinsic rewards are defined as follows:

$$\begin{aligned} R_B &= -\gamma t_B \\ R_A &= \gamma \max(0, t_B - t_A) \end{aligned}$$

  where t_B is total time for Bob to complete the task, t_A is the time of Alice, and γ is constant to rescale the reward with external reward.
- If B fails, t_B is set to $t_{\max} - t_A$ so we penalize Alice
- Looses try to propose hard tasks to Bob but solvable in $t_{\max}$ time

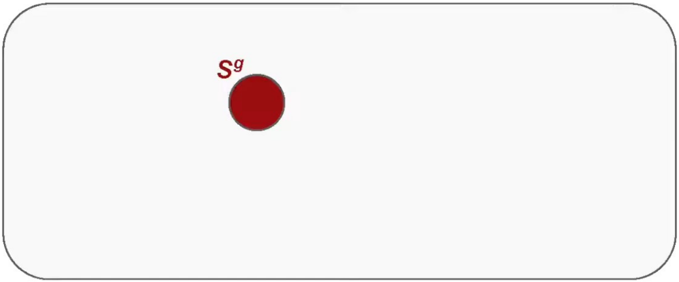
Self-Play Curriculum (Sukhbaatar, et al. 2018)

- It generates a curriculum for exploration that accelerates learning of difficult sparse tasks.
- Tasks for Bob are learnable because Bob and Alice use same architectures, sensors and actions (but just different rewards)
- Learning is asymmetric because Alice does not try to maximize reward of Bob explicitely
- Can only be used in reversible environments!

Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)

S^g: goal states we want to reach from everywhere.

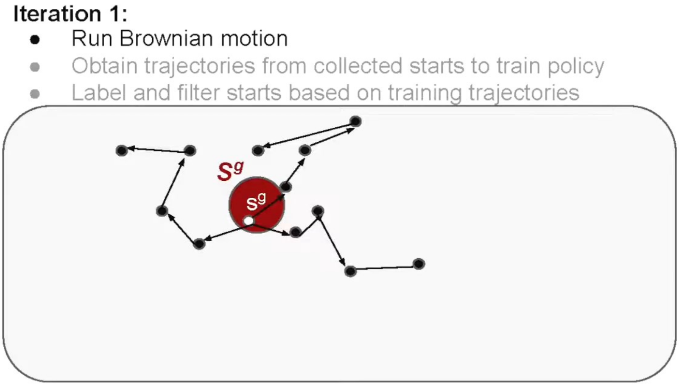


Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)

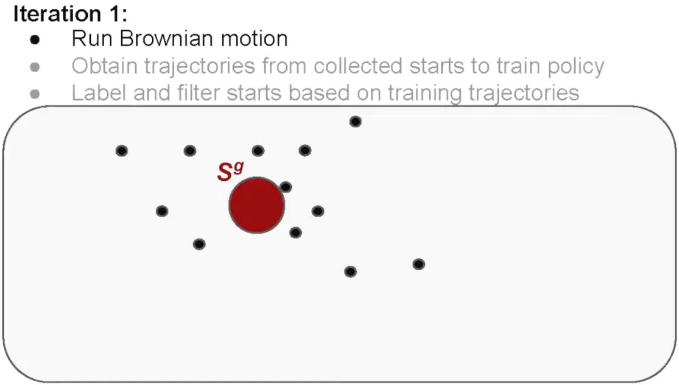
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



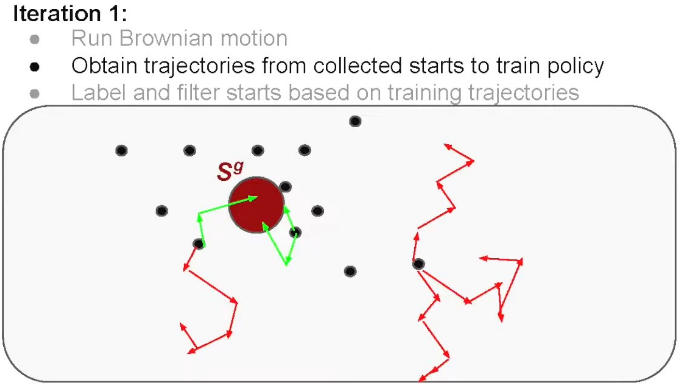
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



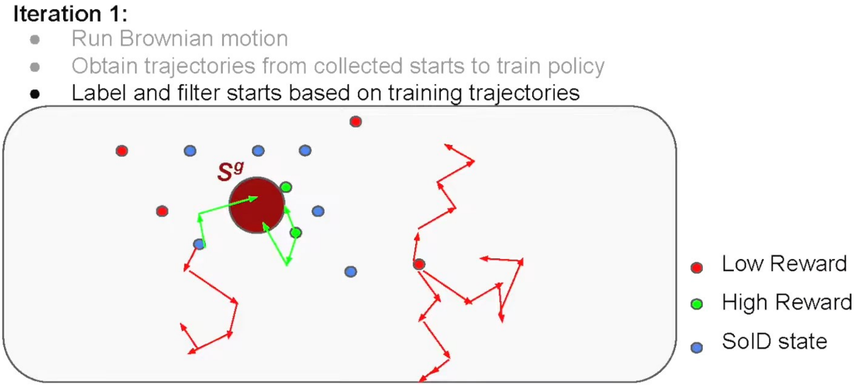
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



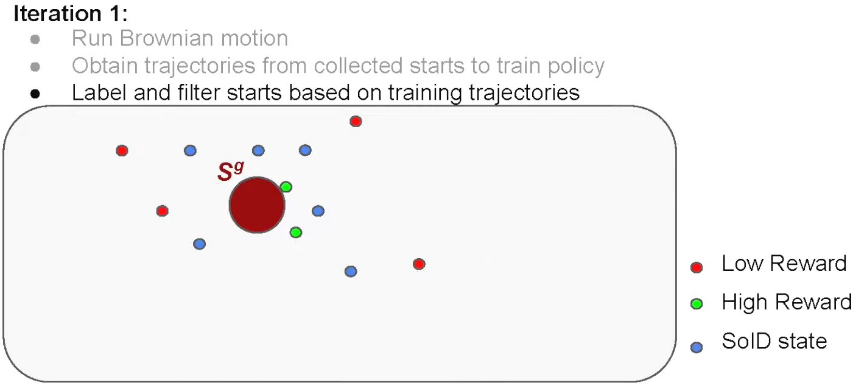
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



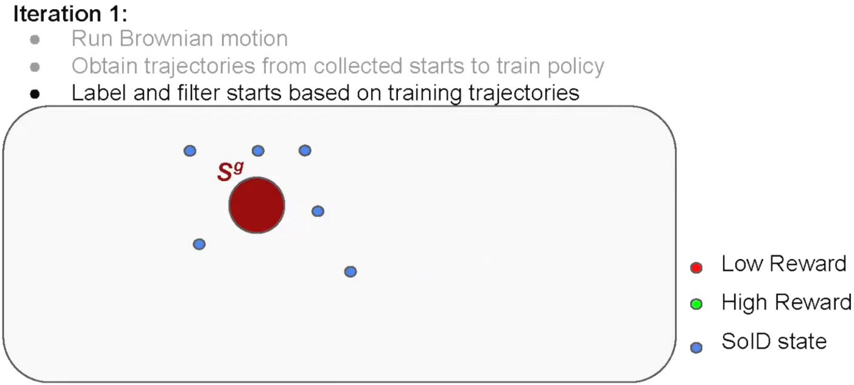
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



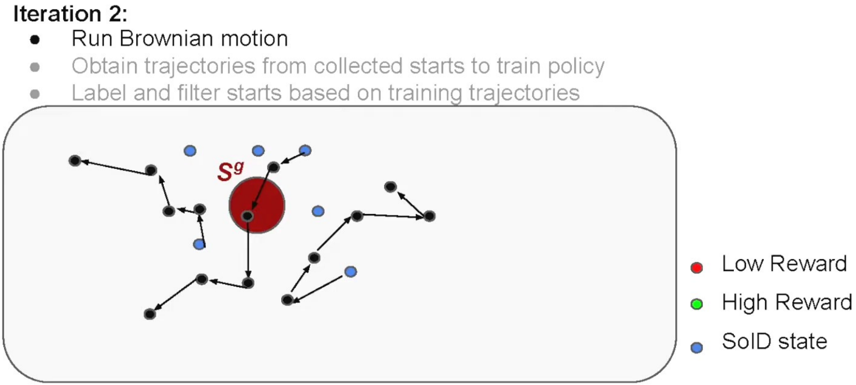
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



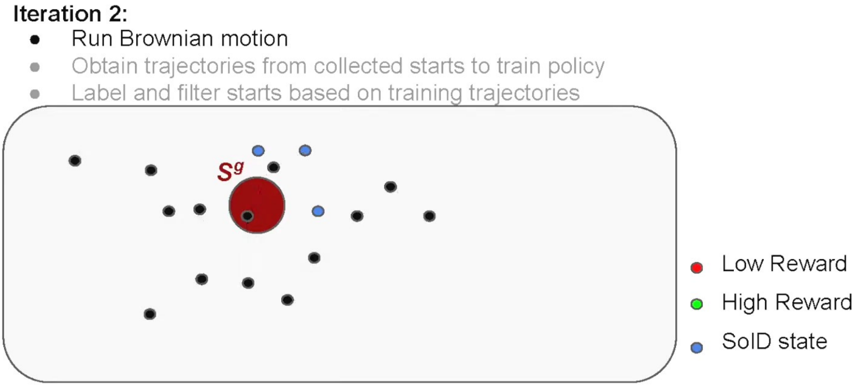
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



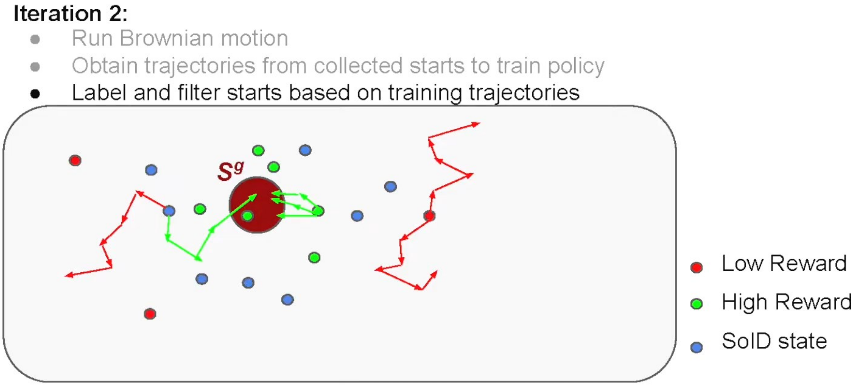
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



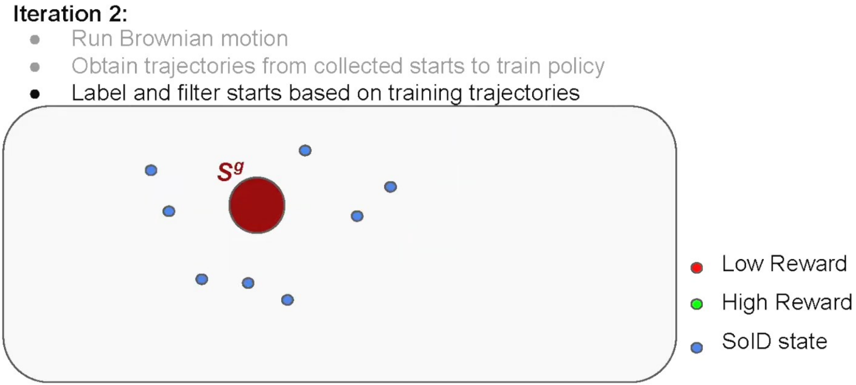
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



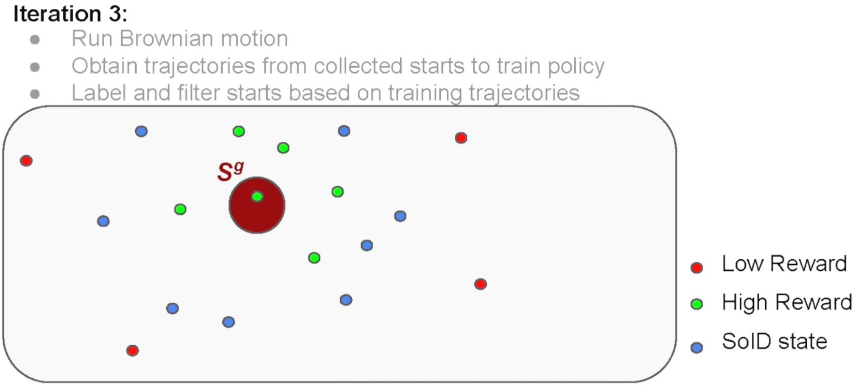
Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)

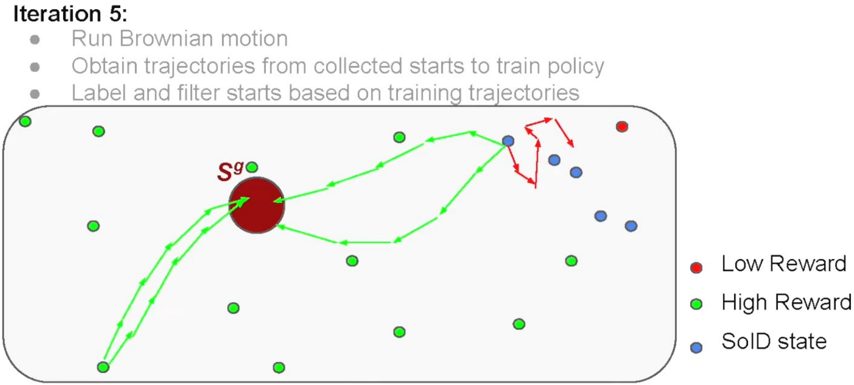


Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)

Reverse Curriculum Generation (Florensa, et al. 2017)

- Idea of generating goals automatically using the criteria of promising initialization from final states backwards (see blog here)



Reverse Curriculum Generation (Florensa, et al. 2017)

- States that are interesting for curriculum are those that are not very easy neither too complex from current policy
- Starts of Intermediate Difficulty (SoID) at iteration i characterized as:

$$S_i^0 = \{s_0 : R_{min} < R(\pi_i, s_0) < R_{max}\}$$

- In robot insert task, iteration 0, 1 and 2.

Reverse Curriculum Generation (Florensa, et al. 2017)

- States that are interesting for curriculum are those that are not very easy neither too complex from current policy
- Starts of Intermediate Difficulty (SoID) at iteration i characterized as:

$$S_i^0 = \{s_0 : R_{min} < R(\pi_i, s_0) < R_{max}\}$$

- In robot insert task, iteration 0, 1 and 2.
- Again, just for reversible environments.

Generative Goal Learning (Florensa, et al. 2018)

- As in the previous paper, they define states of interest as those of intermediate difficulty.
- Algorithm is based on Generative Adversarial Networks (GAN):
  1. Label a set of goals based on whether they are at the appropriate level of difficulty for the current policy.

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

     where $R_{\min}$ and $R_{\max}$ can be interpreted as a minimum and maximum probability of reaching a goal over T time-steps.
  2. Train a Goal GAN model using labelled goals from step 1 to *generate* new goals
  3. Use these new goals to train the policy, improving its coverage objective. Go to 1

Generative Goal Learning (Florensa, et al. 2018)

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow \texttt{initialize\_GAN()}$
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow \texttt{sample\_noise}(p_z(\cdot))$
  $goals \leftarrow G(z) \cup \texttt{sample}(goals_{\text{old}})$
  $\pi_i \leftarrow \texttt{update\_policy}(goals, \pi_{i-1})$
  $returns \leftarrow \texttt{evaluate\_policy}(goals, \pi_i)$
  $labels \leftarrow \texttt{label\_goals}(returns)$
  $(G, D) \leftarrow \texttt{train\_GAN}(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow \texttt{update\_replay}(goals)$
end for

Generative Goal Learning (Florensa, et al. 2018)

Algorithm 1 Generative Goal Learning

Input: Policy π_0
Output: Policy π_N
$(G, D) \leftarrow \texttt{initialize\_GAN()}$
$goals_{\text{old}} \leftarrow \varnothing$
for $i \leftarrow 1$ **to** N **do**
  $z \leftarrow \texttt{sample\_noise}(p_z(\cdot))$
  $goals \leftarrow G(z) \cup \texttt{sample}(goals_{\text{old}})$
  $\pi_i \leftarrow \texttt{update\_policy}(goals, \pi_{i-1})$
  $returns \leftarrow \texttt{evaluate\_policy}(goals, \pi_i)$
  $labels \leftarrow \texttt{label\_goals}(returns)$
  $(G, D) \leftarrow \texttt{train\_GAN}(goals, labels, G, D)$
  $goals_{\text{old}} \leftarrow \texttt{update\_replay}(goals)$
end for

- This algorithm can be used in non invertible domains

Conclusions of Curriculum Learning

- Highly related to *exploration* and *conditioned policies*
- Topic of research with a lot of other approximations
- Some surveys on the topic (Portelas et. al 20) and (Narvekar et al. 20)
- Also blog from Lilian Weng