Distance Based Replay Sampling in Hindsight Experience Replay

Aaron Luo
Jacobs School of Engineering
University of California, San Diego
San Diego, CA
aaluo@ucsd.edu

***Abstract*—In this paper we will be building off the Hindsight Experience Replay by sampling new goals based off of their Euclidean distance to the original goal. The aim here is to give the networks a certain direction early in training and speed up convergence.**

I. Introduction

One of the biggest challenges in robotics today is the sparse reward problem. Robots are trained to perform increasingly complex and long-term tasks while being powered by reinforcement learning algorithms that do not provide any feedback should the agent be unable to chance upon the terminal state. One of the most common and intuitive approaches to solving the sparse reward problem is to use reward shaping, where the robot provided with feedback should be able to reach specific short-term goals in a long list of goals. This method, however, is impractical, as it requires the researchers to have a certain level of knowledge about the environment. Furthermore, reward shaping is typically specific to an environment, so training a robot in a different environment would require a new set of shaped rewards each time. Other standard solutions to the sparse reward problem, such as imitation learning (Pomerleau, 1988) and curriculum learning (Bengio et al., 2009), suffer from the same generalizability problem. Hindsight experience replay (Andrychowicz et al. 2017), or HER, is an off-policy reinforcement learning algorithm that is not only able to function in sparse reward environments with long term or multiple goals but is also generalizable, such that the researcher does not need to do anything specific to the training environment to make learning possible. HER takes advantage of the replay buffer used in off-policy reinforcement learning algorithms such as Deep Q-Networks (DQN) (Mnih et al., 2013) or Deep Deterministic Policy Gradient (DDPG) (Lilicrap et al., 2015) by saving the desired goal states of the environments along with other state information. Doing this allows it to randomly sample previously reached states as 'subgoals,' changing the reward function to provide feedback when the agent reaches these subgoals. In this paper, we will be experimenting with sampling these subgoals based on their Euclidean distance to the main goal to provide an improved version of the original HER algorithm.

II. Background

A. Reinforcement Learning

Reinforcement Learning maximizes the cumulative reward an agent receives from interacting with an environment. We consider an agent in with a fully observable environment. An environment contains a set of states S, a set of actions A, a reward function $r_t = r(s_t, a_t)$, and transition probabilities $p(s_{t+1}|s_t, a_t)$. Let $R_t = \sum_{t=0}^{\infty} \gamma^t r_t$ be a return function, where γ is defined as the discount factor, which determines the priority of short-term rewards. The goal of the agent is to find the optimal policy π^* that maximizes the expected return $\mathbb{E}[R_t|s_t, a_t]$, which is also called the Q function. The optimal Q-function Q^* follows the form of the Bellman equation:

$$Q^*(s,a) = \mathbb{E}_{s',a'}[r(s,a) + \gamma \max Q^*(s',a')], \ \ a' \sim \pi(s')$$

This is the fundamental equation we will be using to train our networks to find the optimal policy.

B. Deep Deterministic Policy Gradient (DDPG)

DDPG (Lilicrap et al., 2015) is an off-policy model free actor-critic reinforcement algorithm that is used in environments with continuous action spaces. This method utilizes a replay buffer that stores previously visited states, which it later samples from to train the networks. Neural networks are trained with parameters θ_i to estimate Q values by minimizing the loss functions. The actor network is trained using gradient descent on the loss:

$$L_{actor,i}(\theta_i) = -E_{s,a,r,s' \sim p(\cdot)}\left[Q\left(s_t, \pi(s_t)\right)\right]$$

The negative form of the loss is used as an easy way to perform gradient ascent, as we are trying to maximize the actor loss. This is due to the loss function of the critic network, which is defined as:

$$L_{critic,i}(\theta_i) = \mathbb{E}_{s,a,r,s' \sim p(\cdot)}\left[\left(y_i - Q(s,a)\right)^2\right]$$

where y_i is defined as:

$$y_i = r(s,a) + \gamma \max_{a'} Q_T\left(s', \pi(s')\right)$$

Since the value of y_i is determined by the actor network policy, and the Q value in the critic loss function is determined by the state and action values sampled from the replay data, action loss is maximized to minimize the critic loss. Parameters of each network are calculated by backpropagating each of their corresponding loss equation.

Separate target networks for each network can be used to stabilize the optimization procedure. These target networks change at a slower pace than the main network, and their weights are updated using the polyak-average method (Polyak and Juditsky, 1992), which is defined as:

$$\theta^{t+1} = \tau\theta^{s} + (1-\tau)\theta^{t}$$

where θ^{t} refers to the target network weights, θ^{s} refers to the source (main) network weights, and τ is the polyak value.

C. Hindsight Experience Replay (HER)

The fundamental idea behind HER (Andrychowicz et al. 2017) is the way the humans can learn through trial and error. A human failing at a task receives feedback through that failure, whether to cease performing a certain action or adjust said action to reach a certain goal better. The HER algorithm considers the most basic form of the reward function and adds a goal parameter to it, such that:

$$r(s_t, a_t, g_t) = \begin{cases} 0, & ||s_t - g_t||_2^2 < threshold \\ -1, & otherwise \end{cases}$$

The HER technique also adds a goal parameter to each transition state $\{s_t, a_t, s_{t+1}, r_t, g\}$ of the replay buffer. When the algorithm samples from the replay buffer, the goal states of the sampled transitions are re-labeled with a randomly chosen s_{t+1} state. The reward values are also re-labeled by passing the new goal state into the reward function defined above. Not all the goal states of the sampled transitions are re-labeled; transitions are randomly selected to be re-labeled at a k to 1 ratio. The original paper describes a k range of $4 \leq k \leq 8$ to be optimal, as a k value higher than 8 degrades performance due to a scarcity of normal replay data.

The original paper also describes three strategies used to sample new goal states to replace the main goal states.

- Random: New goal states are sampled from all available goal states in the replay buffer
- Episode: Transition goals can only be replaced by s_{t+1} states that occurred in the same episode
- Future: Transition goals can only be replaced by s_{t+1} states that occurred after it in the same episode

With the sampled transitions re-labeled and their corresponding reward values recalculated, the networks have more feedback to work with, thus alleviating the sparse reward problem.

III. Methods

A. Distance Based Sampling

The proposed method modifies how the HER algorithm samples new goals for re-labeling, taking advantage of the vectorization and total observability of the environment. The original paper describes the 'future' goal sampling strategy as the method that returns the highest performance and converges the fastest. Distance based sampling builds on this, sampling for new goals using the 'future' strategy based on their Euclidean distance to the desired goal state. The motivation here is to give the agent a direction when selecting subgoals, pushing the agent towards the goal rather than in random directions, potentially speeding up convergence. There are a few ways to implement this:

- Method 1: Choose the state s_{t+1} closest to the desired goal
- Method 2: Using an epsilon greedy strategy, choose the state s_{t+1} closest to the desired goal with a chance to choose a random goal instead
- Method 3: Use weighted random sampling to choose a state s_{t+1} to replace the goal. Probability weights are calculated using the normalized distance values from each state s_{t+1} to the desired goal state.

Method 1 is non-stochastic and could get the networks stuck in local optima. Method 2 is better than Method 1 due to the introduction of stochasticity but requires some amount of tuning. Furthermore, the more the networks are trained, the less sense it makes to sample based on distance since all potential subgoals should be about the same distance to the desired goal. This method fails to account for this. Method 3 potentially solves both problems. It is introducing stochasticity due to its use of random sampling as well as a direction early on in training, while becoming more and more random as the network learns. Method 3 also has the advantage that it eventually is the same random sampling, and will perform just as well in the long term as than the baseline performance of the vanilla HER implementation. We experiment with each of these methods in this paper.

B. Environments

Experimentation took place on robotic arm environments simulated by Mujoco and hosted by the OpenAI Gym library. 'FetchPush-v1', where a robot arm has its grippers locked closed and is tasked with pushing an object onto a specific spot on a table, 'FetchPickAndPlace-v1', where a robot arm is tasked with reaching for an object and placing it on a specific spot on a table, and 'FetchSlide-v1', where a robot arm has its grippers locked closed and is tasked with sliding an object to a specific spot on a table that is out of reach of the arm. In these environments, the arm state and object state are separate entities, with each new observation resulting from executing an action consisting of a new arm state and an 'achieved goal' state, which is the current location of the object. The 'desired goal' or terminal state is the spot on the table the object is meant to be on. Because the arm state and object states use different measurements, they need to be normalized before being used as input for the network Q-value estimators. Specifics are discussed in the 3.D *Normalization* section.

C. Action Selection

There are a few ways to select actions when generating training data for the replay buffer. The most intuitive way is to select the output of the actor network, but this could introduce problems when our actor network has little training. To introduce stochasticity, we use the epsilon greedy strategy, sampling a random action from the action space at a rate of $\epsilon = 0.3$, taking from the actor network otherwise. We also introduce some action noise to simulate a real-world environment better, adding a noise value sampled from $N(0,0.2)$ to each action.

D. Normalization

As discussed in section 3.B, the observation space is split up into a robot arm state space and an object state space. Because these two operate on different domains, they must be normalized, especially since they are concatenated together as part of the HER implementation.

The arm and object space were each assigned a normalization class object that keeps track of the current sum of all its feature values, the number of feature values calculated, and each feature's current mean and standard deviation. Normalization takes place by subtracting each new entry by its corresponding feature mean and dividing by its corresponding feature variance, thus giving each entry feature value a zero mean and unit variance. Each normalizer is regularly updated with the values generated each time the agent steps through the environment.

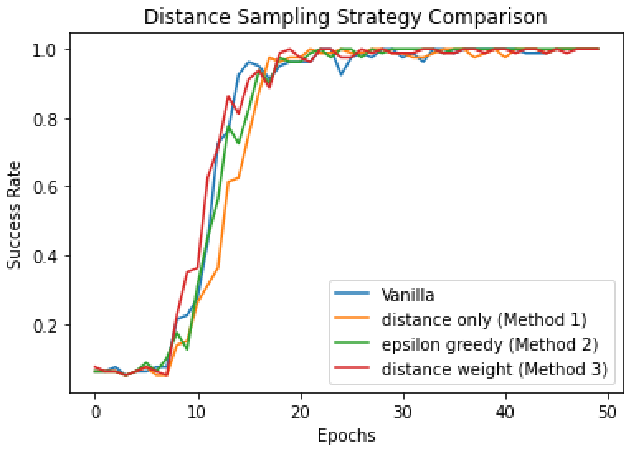
E. Multiprocessing

Due to the complexity of the environments used in this paper, multiprocessing is necessary for learning. It is also possible to increase the size of the replay buffer and the sample batch to accomplish the same thing but doing so increases the training time by several orders of magnitude. For the sake of optimization, multiprocessing is used instead. The 'FetchPush-v1' environment was trained using 8 processes, the 'FetchSlide-v1' environment was trained using 8 processes, and 'FetchPickAndPlace-v1' environment was trained using 16 processes.

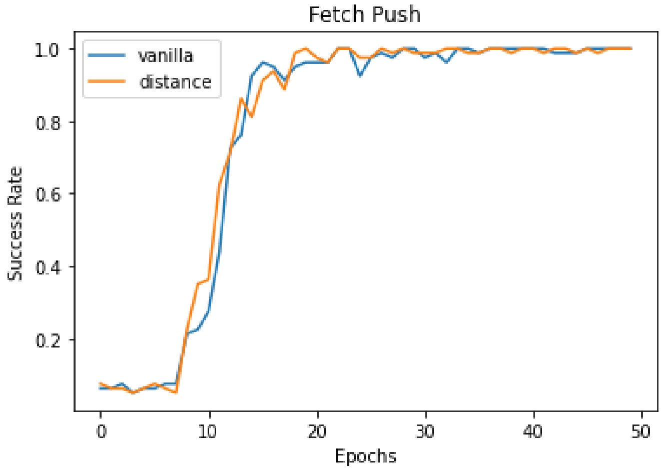
IV. Results

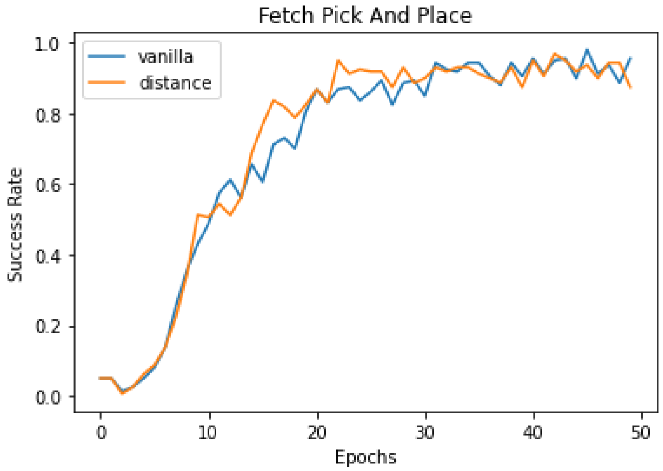
Both vanilla HER and distance sampling HER were trained together on the same seed so that a direct comparison could be made. Training parameters were also kept the same for the same reason. Each environment was trained over 50 epochs, except for 'FetchSlide-v1' which was trained over 200 epochs. Each epoch had 50 cycles of training. Each cycle ran through two episodes, which generated timesteps of samples, after which the networks were updated 40 times. The replay buffer had 10^6 entries and had a sample batch size of 256 samples. Both actor and critic networks consisted of 4 linear layers each with 256 hidden units, and a ReLU activation function between each layer. The actor network passed the output of the last layer through a tanh activation function. Both networks were optimized using an Adam optimizer with a learning rate of 0.001. Lastly, during the network update sequence we passed the actor loss through the L2-loss function with $\lambda = 1$ to prevent overfitting.

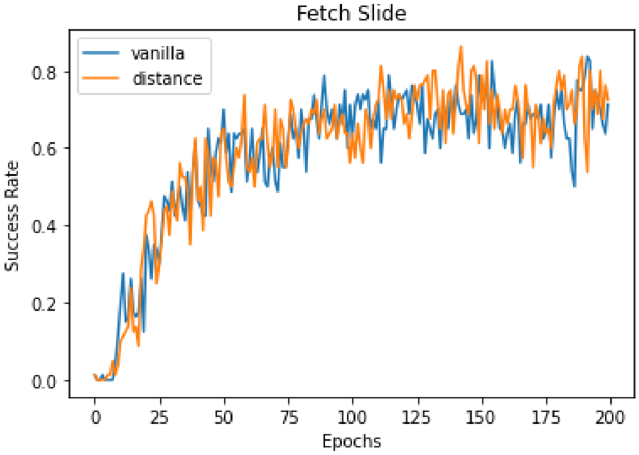
First, we show a comparison graph comparing the various distance sampling strategies mentioned before:



The rest of the environments are trained with the assumption that this is the best distance sampling method. The results are below:







V. Discussion

A. Analysis

Upon cursory inspection of the graphs, it looks as if the behavior of the distance and vanilla HER training curves across each environment is about the same. However, as discussed before, the purpose of distance sampling is to provide the network a direction early in training. As the network has had some time to learn, the curves should look more and more similar. We can see this in the comparison graph, where we test all the methods on the ‘FetchPush-v1’ environment; Method 3 appears to converge at the fastest rate, even when compared to the vanilla HER implementation. The hypothesized issues with Methods 1 and 2 appear to be reflected in the convergence speed as well, with Method 1 performing the worst, and Method 2 performing second worst.

Following the outcome of the comparison graph, the rest of the graphs only compare the vanilla HER implementation with Method 3. If we look at the graph for the ‘FetchPush-v1’ environment, we can see that the agent that uses distance sampling starts convergence two to three epochs sooner and reaches converges about five epochs faster than the agent that uses random sampling. Given that a lot of training time occurs within five epochs, this is a notable improvement in the short term. The same can also be for the ‘FetchPickAndPlace-v1’ environment. From the graph, distance sampling HER converges about ten epochs faster than vanilla HER does. The only environment that doesn’t see noticeable improvement is the ‘FetchSlide-v1’ environment.

One interesting thing to discuss is how for all these environments, the robot arm must first find the object before it can move it, and that distance sampling in these environments doesn’t help the robot arm accomplish this. This is fine, however, since before the object is found, the object remains at a fixed position. Since there is no difference between the ‘achieved goal’ samples that are chosen to replace the ‘desired goal’, the outcome should be equivalent to simple random sampling. We can see this behavior on the graphs; for the first few epochs in both the ‘FetchPush-v1’ environments, and the ‘FetchPickAndPlace-v1’ environments, both algorithms return the very similar performance values. It is only after a certain point that the distance sampling algorithm breaks off and starts to converge faster than the vanilla sampling algorithm. This breakoff point could be where the agent is starting to be able to find the object consistently and move it somewhere.

An explanation as for why the vanilla HER and the distance sampling implementations don’t see much difference in convergence when trained on the ‘FetchSlide-v1’ environment likely has to do with the terminal condition of said environment. As per the description on the OpenAI website, this environment requires that the object come to a rest on the goal position. Because the goal is always out of reach of the arm, it needs to hit the object into the goal spot. Because distance sampling doesn’t account for the physics of the object itself, the agent cannot differentiate between the object moving closer to the goal at high speed or moving closer to the goal at low speed. Thus, distance sampling isn’t very useful in this environment, and ends up returning about the same convergence speed as vanilla sampling does.

B. Conclusion

In conclusion, it appears that modifying the way the HER algorithm samples new goals such that it samples by Euclidean distance does result in notably faster convergence time in environments that only account for the position of the object. The implementation does not work in environments where reaching the terminal state is dependent on variables other than position, such as the velocity of the object. In the future, re-experimenting using different seeds would be useful to make sure that this particular result didn’t occur by chance. This was not done in this paper due to time constraints and the large amounts of time needed to train each agent on certain environments. A good expansion to investigate would be to find a way to generalize the distance sampling method, as the current implementation described in this paper looks to be specific to environments where position is the only relevant measurement.

References

[1] Andrychowicz, Marcin, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. "Hindsight experience replay." *arXiv preprint arXiv:1707.01495* (2017).

[2] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).

[3] Plappert, Matthias. “Ingredients for Robotics Research.” OpenAI. OpenAI, September 4, 2020. https://openai.com/blog/ingredients-for-robotics-research/.

[4] Openai. “Baselines/Baselines/Her at Master · Openai/Baselines.” GitHub. https://github.com/openai/baselines/tree/master/baselines/her.

[5] Dai, Tianhong. “Tianhongdai/Hindsight-Experience-Replay: This Is the PYTORCH Implementation of Hindsight Experience Replay (Her) - Experiment on All Fetch Robotic Environments.” GitHub. https://github.com/TianhongDai/hindsight-experience-replay.