# Informed Sampling Strategies for Multi-Goal Reinforcement Learning

**Balarama Raju Buddharaju**[*]
Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`bbuddhar@andrew.cmu.edu`

**Satyen Rajpal**
Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`satyenr@andrew.cmu.edu`

**Suyash Nigam**
Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
`suyashn@andrew.cmu.edu`

## Abstract

Model-free reinforcement learning is a powerful tool for learning complex behaviors. However, they require large experience buffers and are data-hungry. Methods like Temporal Difference Models provide means for sample-efficient learning and have been found to be effective wrappers on top of, off-policy model free algorithms like Deep Deterministic Policy Gradients. In this project, we explore how using informed sampling strategies can enhance the performance of these models in a multi-goal reinforcement learning setting. Numerical results show that this "informed" sampling helps the off-policy algorithm converge faster than earlier methods. We also provide results from ablation studies we performed to demonstrate the effectiveness of these strategies in goal-based environments with sparse as well as dense rewards.

## 1 Introduction

Reinforcement Learning algorithms when used in conjunction with non-linear function approximations like Deep Neural Networks have been found to be effective on tasks ranging from playing games [3] to controlling robotic arms [2]. However, these deep RL algorithms often require a large amount of experience to arrive at an effective solution, which can severely limit their application to real-world problems where this experience might need to be gathered directly on a real physical system. Part of the reason for this is that direct, model-free RL learns only from the reward: experience that receives no reward provides minimal supervision to the learner. This problem is more pronounced in multi-goal reinforcement learning where learning agent is required to address several tasks at once. To address this problem of poor sample efficiency, several smart-sampling strategies have been proposed. Time-horizon relabeling as done in Temporal Difference Models [1] has been proven to be effective in speeding up learning. Temporal Difference Models evaluate Q-values with an additional input $\tau$, which represents the planning horizon. TDM models describe how close the agent will get to a given goal state $s_g$ after taking $\tau$ time steps, when it is attempting to reach that state in $\tau$ steps. We can resample new goals $s_g$ and new horizons $\tau$ for each tuple $(ts_t;\, a_t;\, s_{t+1})$, even the ones which were not a part of the original data collection. The pivotal idea here is to replay each episode with

[*]Authors listed in alphabetical order.

a different goal and time-horizon than the one with which the agent was trying to achieve. This leads to a natural data augmentation strategy, since each tuple can be replicated many times for many different goals and different time horizons without additional data collection. In this way, TDM can be trained very efficiently, since every tuple provides supervision for every possible goal and every possible horizon. TDMs are applicable whenever there are multiple goals which can be achieved, e.g. achieving each state of the system may be treated as a separate goal. Not only does this strategy improve the sample efficiency but also makes learning possible even if the reward signal is sparse and binary.

In this project, we implement TDMs in Keras RL framework as an extension to a popular off-policy algorithm Deep Deterministic Policy Gradients. We work with the goal-based environments released as part of OpenAI Gym (Brockman et al., 2016 [22]) and use the MuJoCo (Todorov et al., 2012 [23]) physics engine for fast and accurate simulation. These environments take as input not only the current state, but also a goal state. In particular, we work with the Robotics environments where the task is to control the robot in simulation. One aspect where we believe the performance of TDMs could be improved is the way goals are resampled. In the proposed TDM approach, the resampled goals for each step in an episode are randomly sampled from the goals achieved after that step within the same episode. We believe choosing these goals in an "informed" manner can help us achieve convergence earlier than the proposed method. So, we resort to sampling goals based on their value function and demonstrate that when performed with decaying exploration, this indeed helps the algorithm converge faster. Furthermore, we also experimented with automatic goal generation- a network predicts the most valuable goals by itself. Though we were able to see some positive results, we were not able to have a converged solution and hence, call the study inconclusive.

In the following sub-sections, we discuss in brief the following concepts upon which our work is based: Reinforcement learning formalism, goal-conditioned value functions, on-policy and off-policy algorithms, model-free and model-based methods and Deep Deterministic Policy Gradient methods.

## 1.1 RL formalism

We work with the general formalism dealing with decision making problems that consist of a state space $S$, action space $A$, transition dynamics $P(s'|s; a)$, and an initial state distribution $p_0$. The goal of the learner is encapsulated by a reward function $r(s; a; s_0)$. Typically, long or infinite horizon tasks also employ a discount factor $\gamma$ and the standard objective is to find a policy $\pi((a|s))$ that maximizes the expected discounted sum of rewards, $E_\pi[\sum_t \gamma^t r(s_t; a_t; s_{t+1})]$, where $s_0 \sim p_0$, $a_t \sim \pi(a_t|s_t)$, and $s_{t+1} \sim P(s'|s; a)$.

## 1.2 Goal-conditioned value functions

In [6], Somil et al argue that having a task-specific approach - i.e. learning the dynamics of a model specific to the task - is more valuable than learning the entire model. To achieve this, goal-conditioned value functions are conditioned on some task description vector belonging to the goal space. The goal vector induces a parametric reward function $r(s_t, a_t, s_{t+1}, s_g)$ which gives rise to parametric Q-functions. In this project, we deal with goal-oriented Robotics and Mujoco environments provided by OpenAI gym. These environments come with a state space in which each step is characterized by an observation, a desired goal and a goal achieved by the end of the step. The rewards are given away based on the distance of the achieved goal from the desired goal. We can choose to work with sparse rewards as well in which case we get binary rewards based on our achieved goal being within an epsilon-neighborhood of the desired goal.

## 1.3 On-policy and off-policy algorithms

Reinforcement Learning algorithms which are characterized as off-policy generally employ a separate behavior policy that is independent of the policy being improved upon; the behavior policy is used to simulate trajectories. A key benefit of this separation is that the behavior policy can operate by sampling all actions, whereas the estimation policy can be deterministic (e.g., greedy) [1]. Q-learning is an off-policy algorithm, since it updates the Q values without making any assumptions about the actual policy being followed. Rather, the Q-learning algorithm simply states that the Q-value corresponding to state $s_t$ and action $a_t$ is updated using the Q-value of the next state $s_{t+1}$ and the action $a_{t+1}$ that maximizes the Q-value at state $s_{t+1}$. On-policy algorithms directly use the policy that is being estimated to sample trajectories during training. In this project, we work with a popular off-policy algorithm called Deep Deterministic Policy Gradients [20] as a baseline and bring in the ideas of TDMs and informed sampling strategies as extensions to this.

### 1.4 Model-Free Methods Vs Model-Based Methods

In model-based methods, the algorithms obtain a large amount of supervision by keeping track of the state transitions. They do so to learn the system dynamics. Although this leads to higher theoretical efficiency [5], if the system is highly complex then the policy built by planning over this learned model could be highly sub-optimal. This is because of model bias inherent in learned model. In contrast, a model-free method learns the policy by approximating the state-value function or by approximating a policy function from the feature space and sampling through it henceforth. These methods require huge number of training examples and are hence sample inefficient, but as they don't intend to learn a state-transition function; the policy learned is usually bias-free. Given that we intend to show how our ideas can improve sample efficiency and conventional wisdom holds that model-free methods are sample-inefficient, we work with a model-free framework in this project. However, the idea of TDM and our extensions can be extended with few changes to model-based framework as well. Infact, the main point of TDMs is to illustrate a connection between model-free and model-based approaches.

### 1.5 Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradients (DDPG) [20] is a model-free actor-critic RL algorithm for continuous action spaces. DDPG consists of two neural networks: a target policy (also called an actor) $\pi : S \rightarrow A$ and an action-value function approximate (called the critic) $Q : S \times A \rightarrow R$. The critic's job is to approximate the actor's action-value function $Q^{\pi}$. Episodes are generated using a behavioral policy which is a noisy version of the target policy, e.g. $\pi_b(s) = \pi(s) + \aleph(0, 1)$. The critic is trained in a similar way as the Q-function in DQN but the targets $y_t$ are computed using actions outputted by the actor, i.e. $y_t = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}))$. The actor is trained with mini-batch gradient descent on the loss $L_a = E_s Q(s, \pi(s))$, where $s$ is sampled from the replay buffer. The gradient of $L_a$ w.r.t. actor parameters can be computed by backpropagation through the combined critic and actor networks.

## 2 Related work

Lin et.al.[21] introduced the concept of experience replay as an extension to basic RL algorithms for speeding up learning. It gained prominence after it was used in the DQN agent playing Atari [10] and is currently adopted as a standard practice. Prioritized experience replay [11] is an improvement to experience replay which prioritizes those transitions with high magnitude of TD error over the other transitions.

Schaul et al. [12] introduced the concept of universal value function approximators (UVFAs) $V(s, g;)$ that generalise not just over states $s$ but also over goals $g$. Many variants of goal-conditioned value functions have been proposed in the literature [12][13][14]. A particularly related UVF extension is hindsight experience replay (HER) [7]. Both HER and TDMs retroactively relabel past experience with goal states that are different from the goal aimed for during data collection. However, unlike TDMs, the standard UVF in HER uses a single temporal scale when learning.

Graves et al. [15] consider a setup where there is a fixed discrete set of tasks and empirically evaluate different strategies for automatic curriculum generation in this settings. Another approach investigated by [16] and [9] uses self-play between the policy and a task-setter in order to automatically generate goal states which are on the border of what the current policy can achieve.

Silver et al. and Jaderberg et al. aim to enhance supervision signals for model-free RL [17][18]. Predictions [18] augment classic RL with multi-step reward predictions, while UNREAL [17] also augments it with pixel control as a secondary reward objective. These are substantially different methods from our work, but share the motivation to achieve efficient RL by increasing the amount of learning signals from finite data.

Plappert et.al. [8] introduced a suite of challenging continuous control tasks (integrated with OpenAI Gym) based on contemporary robotics hardware. The tasks include pushing, sliding and pick  place with a Fetch robotic arm as well as in-hand object manipulation with a Shadow Dexterous Hand. All tasks have sparse binary rewards and follow a Multi-Goal Reinforcement Learning (RL) framework in which an agent is told what to do using an additional input.

## 3 Methods - Temporal Difference Models

Temporal difference models are a class of algorithms which, using goal-conditioned value functions, aim to leverage the rich information in state transitions to learn efficiently as in model-based methods, while still achieving the asymptotic per- formance of model-free methods. TDM's are essentially a combination of the following data augmentation strategies:

**Relabelling goals** One ability humans have, unlike the current generation of model-free RL algorithms, is to learn almost as much from achieving an undesired outcome as from the desired one. Imagine that you are learning how to play hockey and are trying to shoot a puck into a net. You hit the puck but it misses the net on the right side. The conclusion drawn by a standard RL algorithm in such a situation would be that the performed sequence of actions does not lead to a successful shot, and little (if anything) would be learned. It is however possible to draw another conclusion, namely that this sequence of actions would be successful if the net had been placed further to the right. Andrychowicz et.al. were the first to propose a strategy along these lines in [2] and demonstrated that such relabeling allows an algorithm to perform this kind of reasoning and can be combined with any off-policy RL algorithm to efficiently solve environments with sparse, binary rewards. The main idea behind such strategy is to replay each episode with a different goal than the one the agent was trying to achieve, e.g. one of the goals which was achieved in the episode.

**Finite Time Horizon Planning** Relabeling previously visited states with the reward for any goal leads to a natural data augmentation strategy, since each tuple can be replicated many times for many different goals without additional data collection. This kind of relabeling is exploited by [3] in the formulation of temporal difference models and we will emulate the same to boost sample efficiency.

Talking about the choice of reward function for the goal-conditioned value function, a particularly intriguing connection to model-based RL emerges if we set a goal space $G = S$, our state space, such that $g \in G$ corresponds to a goal state $s_g \in S$, and we consider distance-based reward functions $r_d$ of the following form:

$$r_d(s_t; a_t; s_{t+1}; s_g) = -D(s_{t+1}; s_g)$$

where $D(s_{t+1}; s_g)$ is a distance such as the euclidean distance between states $s_{t+1}$ and $s_g$. Building on this interesting idea, a logical connection between model-free and model-based RL for a short-horizon setting can be derived. This can be extended to a long-horizon setting by introducing a new mechanism for aggregating long-horizon rewards. Instead of evaluating Q-values as discounted sums of rewards, an additional hyperparameter $\tau$, which represents the planning horizon, is deployed and an alternative Q-learning recursion is defined as :

$$Q(s_t; a_t; s_g; \tau) = E_{p(s_{t+1}|s_t,a_t)}[-D(s_{t+1}, s_g)\mathbb{1}[\tau = 0] + \max_a Q(s_{t+1}, a, s_g, \tau - 1)\mathbb{1}[\tau \neq 0]]$$

This shows the versatility associated with TDMs and how it can be applied to model-based as well as model-free methods. Furthermore, we can use scalar and vectorized forms of rewards - scalar rewards take into account euclidean/ $l1$-norm distance between achieved goal and desired goal while vectorized rewards deal with distance between achieved goal and desired goal along each dimension. We use scalar rewards throughout this project (though vectorized rewards can provide much better performance - demonstrated to be better than HER by Pong et.al.).

### 3.1 Informed sampling Heuristics

The present implementation of TDM relies on random sampling for hindsight goal generation at the terminal of each episode. As mentioned in [8], sampling valuable hindsight goals could be critical for improved performance of the algorithms. We contribute and evaluate the following heuristics as methods to sample valuable experiences.

**Value Function Sampling** One heuristic which can fulfill this criteria, is sampling goals based on their value function. If the network samples goals with higher value of value function, then the algorithm will 'replay' experiences which contain better information to reach the goal. We use an unbiased, sum of future discounted rewards as our value function representation. Furthermore, the network should also see some "bad" and "mixed" experiences; to do so we add a component of exploration to our algorithm at the beginning of training, which decays as the training proceeds, to sample states with entire range of value function. Once, the network has seen enough experiences, we make it replay only those ones in which it was closer to achieving the goal. We were surprised to see that the value function sampling performs only marginally without initial exploration but performs very well with exploration.

**Automatic Goal Generation** Another possible avenue for improving the TDM algorithm is to make the actor network learn which goals to relabel; we call this automatic goal generation. A goal feeder network is placed upstream of the actor network in DDPG-TDM implementation. This sub module takes the achieved goal (after taking an action), the number of steps left in the planning horizon and the previous state observation as in out. The intuitive understanding of this approach could be explained in a two-fold manner. Firstly, by seeing the state achieved after taking some action, the previous observation and knowing the planning horizon steps left; the network can learn the trajectory of the policy and sample a goal state. Secondly, after knowing the goal state to reach; the actor network can output an efficient action which will enable the agent to reach the goal state in the remaining time horizon.

**Algorithm 1** Temporal Difference Model with Heuristics

```
1:  procedure TRAINING(TDM $Q_w(s, a, s_g, \tau)$, replay buffer $\mathcal{B}$, local buffer $\mathcal{D}$)
2:      for $n = 0, \ldots, N-1$ episodes do
3:          Sample goal $g_o$ and initial state $S_o$
4:          $s \sim p(s_0)$
5:          for $t = 0, \ldots, T-1$ do
6:              Sample an action from $\pi(s_t, s_{g,t}, T-t)$
7:              $a_t = AddNoise(a_t^*)$
8:              $g_{t+1}, s_{t+1} \sim p(s_t, a_t)$
9:              Store $(s_t, a_t, s_{t+1}, r_t)$ into $\mathcal{D}$
10:         for $t = 0, \ldots, T-1$ do
11:             Calculate expected discounted reward
12:             $G_t = \mathbb{E}[\sum_{n=t}^{T-1} \gamma^{n-t} r_t]$
13:             Store $(s_t, a_t, s_{t+1}, G_t, n, t)$ into $\mathcal{B}$
14:         for $i = 0, \ldots, I-1$ iterations do
15:             Sample M transitions $\{s_m, a_m, s_{m_t+1}, G_{t,m}, n, t\}$ from $\mathcal{B}$
16:             Relabel time horizons $\tau_m$                                  ▷ As in [3]
17:             Relabel goal $s_{g,m} = \Gamma(s_m, s_{m_t+1}, \tau_m)$          ▷ Automatic Goal Generation
18:                                     (OR)
19:             Relabel goal $s_{g,m}$ based on $G_{t,m}$ with probability $\epsilon$   ▷ Value Function Sampling
20:             $D = \phi(s_m || s_{g,m})$
21:             $y_m = -D\mathbb{1}[\tau_m = 0] + max_a Q'(s_{m_t+1}, s_{g,m}, a, \tau_m - 1)\mathbb{1}[\tau_m \neq 0]$
22:             $\mathcal{L}(w) = \sum_m (Q_w(s_m || s_{g,m}, a_m, \tau_m) - y_m)^2 / M$
23:             Minimize$(w, \mathcal{L}(w))$
24:
```

## 4 Experiments

We conduct all our experiments on OpenAI Gym's FetchReach environment. In this environment a goal position is randomly placed in 3D space, and the task of the end-effector is to reach this position as quickly as possible. We perform experiments using vanilla DDPG and TDM-DDPG as baselines. Furthermore, we experiment with our introduced smart sampling heuristics and compare them with the above mentioned algorithms. All our experiments are carried out for both dense and sparse reward functions. We also perform ablation studies by a) Using different values for exploration and b)Using different values of time horizons. The network architectures and hyper-parameter values are catalogued in the appendix.

## 5 Results

The figures represent the performance of the agent under different algorithms. An episode is considered to be successful if the end-effector reaches within 5cm of the target position. We performed these experiments under different hyperparameter settings and report the best ones.

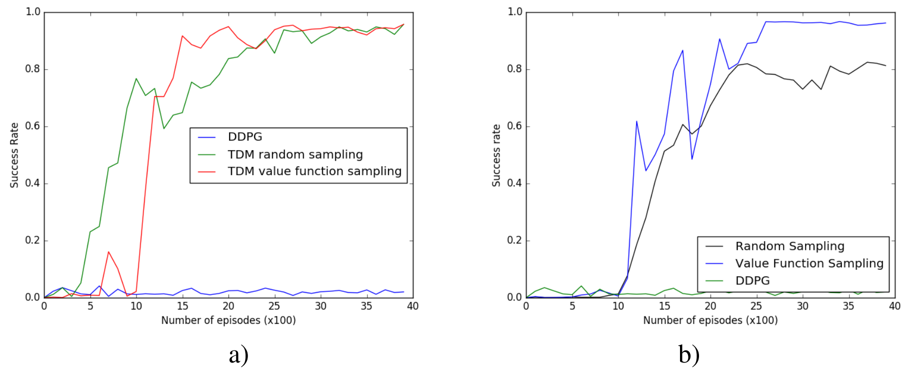


Figure 1: TDM-DDPG and vanilla-DDPG with and without value function sampling a) Dense rewards b) Sparse rewards

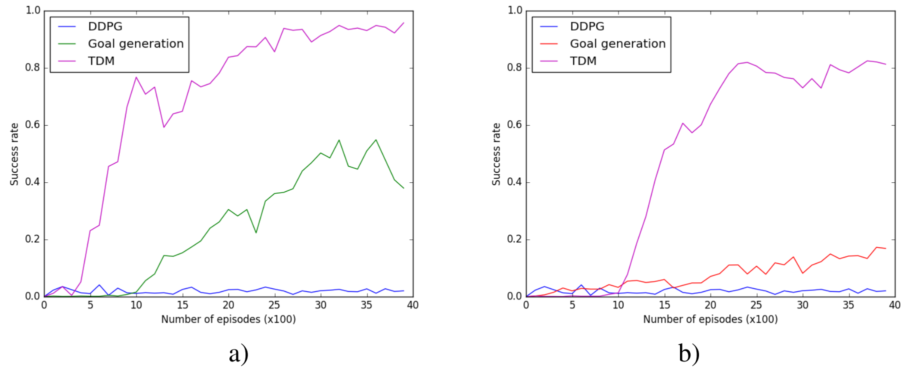


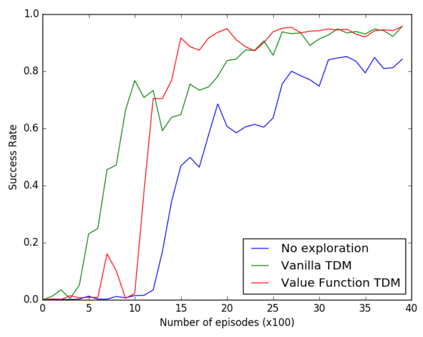
Figure 2: TDM-DDPG and vanilla-DDPG with and without automatic goal generation a) Dense rewards b) Sparse rewards

## 6 Ablation Studies

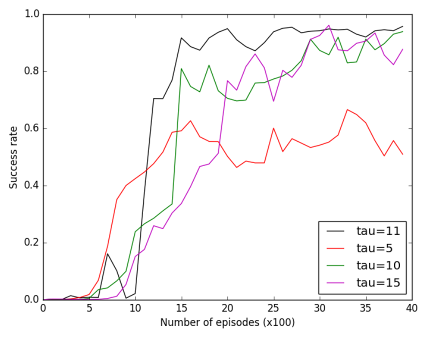
We perform ablation studies using the following design choices.

1. Exploration or no exploration in Value Function Sampling.

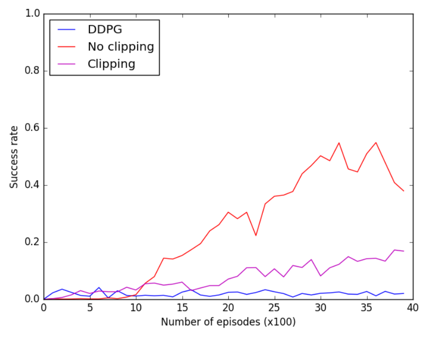
2. Different values of maximum time horizon $\tau$.
3. Clipping or not clipping the generated goal in Automatic Goal Generation.



a)



b)



c)

Figure 3: Ablation studies a) Exploration or no exploration in Value Function Sampling b)Different values of planning horizon tau c)Automatic goal generation with and without clipping of generated goal

The following section explores the results and the ablation studies.

## 7 Discussion

From Figure 1 it is clear that DDPG alone is unable to solve the environment, in both dense and sparse rewards structure. TDM on the other hand is able to solve it efficiently. In the dense reward structure we observe that the heuristic of sampling goals with higher value function coupled with high initial exploration performs at par with random future sampling. The main observation is the difference in the performance of the two algorithms in sparse reward structure. While random sampling stagnates at around 0.8 success rate, this heuristic manages to get convergence; making it a critical component of the algorithm.

A sparse reward structure allows for only minimal supervision, with positive signal being achieved if and only if the agent reaches the goal state. This makes it incredibly difficult to differentiate and sample valuable experiences for relabeling. The hypothesis that using discounted sum of future rewards as a metric allows for resampling the most valuable goals has been reinforced by the above experiment. It's readily easy for TDMs to work with a dense reward setting as the agent is already getting a strong supervision signal at every step of the episode. Our extension helps it perform better or at least as good as vanilla TDM.

From Figure 2, the second Heuristic of automatic goal generation under-performs when compared with a vanilla-TDM implementation, but still improves upon the baseline DDPG implementation. As expected a dense reward structure scheme enables better performance than a sparse one. It is interesting to see that even without explicitly providing a goal to resample, the network is able to learn a relatively learn a well-performing policy from the goal states it produces on its own.

However, this behavior could not be sustained in the long-run and could possibly be improved by providing better supervision than what the network receives just from the backpropagation of loss of the actor's network. A possible avenue for improvement could be to use an independent network for goal prediction. This would provide an enhanced supervision and would decouple actor network's losses with the goal generation one.

From Figure 3, we observe the following from the ablation studies: -

- Exploration is critical to the performance of value function sampling. We observed that relabeling states on the sole basis of sum of future discounts makes the learning process slower, furthermore it stagnates after showing promising initial trends. One intuitive understanding of this could be thought of as - by making the network "see" uniformly sampled goal states in the initial steps of the episode, we enable it to distinguish "good" and "bad" states better as the training progresses.
- The value of the maximum planning horizon $\tau_{max}$ is the most critical hyperparameter to the performance of the algorithm. It is observed that increasing its value makes the training process slower, while still enabling convergence. A low value of $\tau_{max}$ shows promising initial trends, but the learning process stagnates there after. We observe that a $\tau_{max}$ value of 11 performs most efficiently for this particular environment.
- Clipping the value of generated goal dramatically deters the performance of the network. Interestingly, when we clipped the generated goal value within the possible range specified by OpenAI documentation, the performance of the network falls drastically. While the exact reasons for this anomalous behavior needs to be investigated, we noticed that beyond a point, the network continuously predicts goals along the clipping boundary. This could possibly be dealt with better ways of scaling down the outlying goals to the permissible range.

## 8 Conclusion and Future Work

We implemented TDMs for FetchReach robotics environment in OpenAI gym which proved to be quite tricky given the number of hyperparameters involved. We demonstrated the value addition associated with relabeling goals and time-horizons as an extension to vanilla DDPG. We then explored our ideas of informed goal sampling with value function as a qualifying metric and automatic goal generation.

We proved that using value function as a metric with decaying exploration indeed helps the algorithm converge faster. Our study with automatic goal generation is inconclusive at the time of submission. We intend to work on this further. We provide a set of ablation studies for the most critical hyperparameters: $\tau_m ax$, exploration and clipping helping us draw interesting inference about training TDMs and our proposed extensions.

It would be interesting to see how these algorithms perform under more complicated Robotics environments released by OpenAI. Moreover, a vectorized form of TDMs could be explored to see better results.

## References

[1] Vitchyr Pong* and Shixiang Gu* and Murtaza Dalal and Sergey Levine. Temporal Difference Models: Model-Free Deep RL for Model-Based Control. International Conference on Learning Representations. https://openreview.net/forum?id=Skw0n-W0Z

[2] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. Journal of Machine Learning Research, 17(39):1–40, 2016.

[3] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016.

[4] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, pp. 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[5] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. Foundations and Trends in Robotics, 2(1–2):1–142, 2013.

[6] Somil Bansal, Roberto Calandra, Ted Xiao, Sergey Levine, and Claire J Tomlin. Goal-driven dynamics learning via bayesian optimization. arXiv preprint arXiv:1703.09260, 2017.

[7] Andrychowicz, Marcin et al. Hindsight Experience Replay. NIPS (2017).

[8] Matthias Plappert et al. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research https://arxiv.org/pdf/1802.09464.pdf

[9] Held, D., Geng, X., Florensa, C., and Abbeel, P. (2017). Automatic goal generation for reinforcement learningagents. arXiv preprint arXiv:1705.06366.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[11] Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015a). Universal value function approximators. In Proceedings of the 32nd International Conference on Machine Learning (ICML-15), pages 1312–1320.

[12] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized experience replay. arXiv preprint arXiv:1511.05952.

[13] Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

[14] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. arXiv preprint arXiv:1611.01779, 2016.

[15] Graves, A., Bellemare, M. G., Menick, J., Munos, R., and Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. arXiv preprint arXiv:1704.03003.

[16] Sukhbaatar, S., Kostrikov, I., Szlam, A., and Fergus, R. (2017). Intrinsic motivation and automatic curricula via asymmetric self-play. arXiv preprint arXiv:1703.05407.

[17] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. International Conference on Learning Representations, 2017.

[18] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. International Conference on Machine Learning, 2017.

[19] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[20] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

[21] Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine learning, 8(3-4):293–321.

[22] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. arXiv preprint arXiv:1606.01540.

[23] Todorov, Emanuel, Erez, Tom, and Tassa, Yuval. Mujoco: A physics engine for model-based control. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5026–5033. IEEE, 2012.

## 9 Model details and hyperparameters

In this section, we detail the experimental setups used in our results. TDM: We used a network with 2 hidden layer with 300 neurons and ReLU activations for both the actor and the critic. For the automatic generation of goals, we add 2 more layers on top of the critic network. We performed episodic rollouts with maximum 50 steps per episode and the episode was considered successful if the goal state was achieved at an arbitrary timestep during the episode. We use a discount factor of 1, $\tau_{max}$ of 11, reward scale of 8, batch size of 128, euclidean distance metric for reward computation and learning rates of 1e-3 and 1e-4 for the actor and critic network respectively for all the results submitted. Furthermore the actions predicted by the network are clipped between -5 cm and 5 cm.

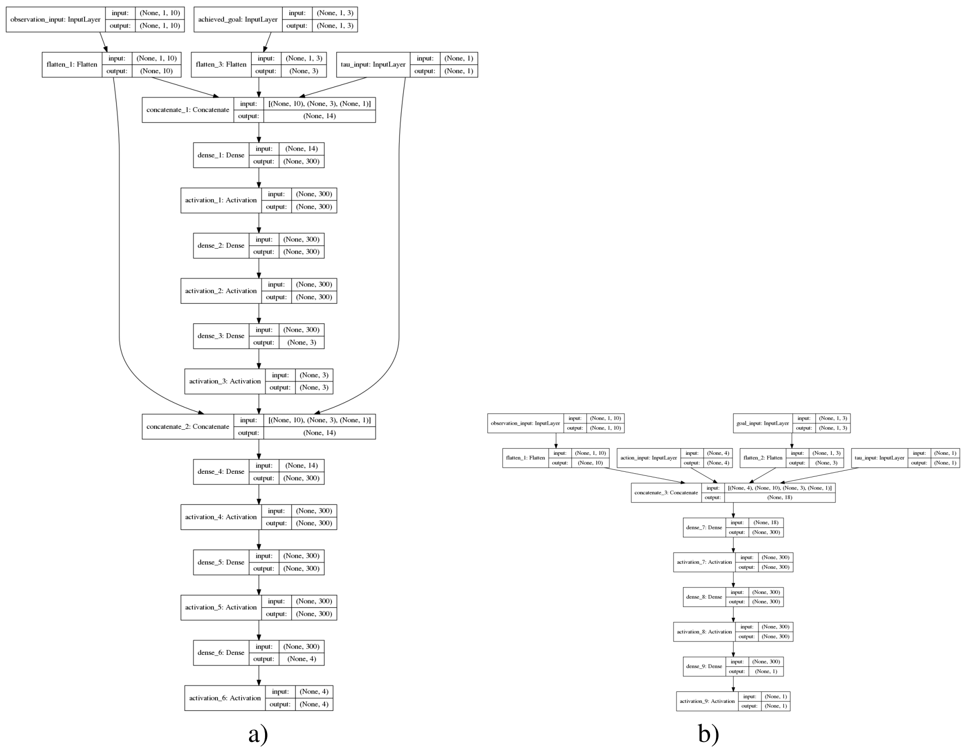

observation_input: InputLayer | input: (None, 1, 10) | output: (None, 1, 10)
achieved_goal: InputLayer | input: (None, 1, 3) | output: (None, 1, 3)
flatten_1: Flatten | input: (None, 1, 10) | output: (None, 10)
flatten_3: Flatten | input: (None, 1, 3) | output: (None, 3)
tau_input: InputLayer | input: (None, 1) | output: (None, 1)
concatenate_1: Concatenate | input: [(None, 10), (None, 3), (None, 1)] | output: (None, 14)
dense_1: Dense | input: (None, 14) | output: (None, 300)
activation_1: Activation | input: (None, 300) | output: (None, 300)
dense_2: Dense | input: (None, 300) | output: (None, 300)
activation_2: Activation | input: (None, 300) | output: (None, 300)
dense_3: Dense | input: (None, 300) | output: (None, 3)
activation_3: Activation | input: (None, 3) | output: (None, 3)
concatenate_2: Concatenate | input: [(None, 10), (None, 3), (None, 1)] | output: (None, 14)
dense_4: Dense | input: (None, 14) | output: (None, 300)
activation_4: Activation | input: (None, 300) | output: (None, 300)
dense_5: Dense | input: (None, 300) | output: (None, 300)
activation_5: Activation | input: (None, 300) | output: (None, 300)
dense_6: Dense | input: (None, 300) | output: (None, 4)
activation_6: Activation | input: (None, 4) | output: (None, 4)
a)
observation_input: InputLayer | input: (None, 1, 10) | output: (None, 1, 10)
goal_input: InputLayer | input: (None, 1, 3) | output: (None, 1, 3)
flatten_1: Flatten | input: (None, 1, 10) | output: (None, 10)
action_input: InputLayer | input: (None, 4) | output: (None, 4)
flatten_2: Flatten | input: (None, 1, 3) | output: (None, 3)
tau_input: InputLayer | input: (None, 1) | output: (None, 1)
concatenate_3: Concatenate | input: [(None, 4), (None, 10), (None, 3), (None, 1)] | output: (None, 18)
dense_7: Dense | input: (None, 18) | output: (None, 300)
activation_7: Activation | input: (None, 300) | output: (None, 300)
dense_8: Dense | input: (None, 300) | output: (None, 300)
activation_8: Activation | input: (None, 300) | output: (None, 300)
dense_9: Dense | input: (None, 300) | output: (None, 1)
activation_9: Activation | input: (None, 1) | output: (None, 1)
b)

Figure 4: Architectures for Goal Generation Network a) Actor b) Critic

## 10 Acknowledgement

We would like to thank Vitchyr Pong, author of Temporal Difference Models: Model-Free Deep RL for Model-Based Control, for providing fruitful comments and insights which greatly enhanced our understanding of the entire TDM technique. We would also like to thank William Guss, for his valuable feedback and encouragement.