Meta-Reinforcement Learning

Raju Tadisetti **Jaishyam**

rajutadi@buffalo.edu **jaishyam@buffalo.edu**

Abstract:

Meta-reinforcement learning algorithms help agents to learn new abilities much more rapidly by exploiting experience to learn how to learn. The main objective of the project is to learn the unseen tasks very quickly. In this project we implemented RL2 (Fast Reinforcement learning via slow reinforcement learning) in Meta world Environments. In this project we implemented RL2 (Fast Reinforcement learning via slow reinforcement learning) in Meta world Environments. We used ML-10 environments for training and testing.

1 Introduction

Reinforcement learning outperforms human knowledge in many areas like beating game Go world champion and playing Dota Game and many more state-of-the-art problems, state of the art methods require substantially more experience than humans to acquire only one narrowly-defined skill. If we want robots to be broadly useful in realistic environments, we instead need algorithms that can learn a wide variety of skills reliably and efficiently. To quickly learn a new task, the environment need to share common structure with previously trained environments. There are many algorithms that solve single task or environment but those algorithms can not solve the task which has not been seen by the agent. In this we used meta-world environments (these environments shared common structure). We used ML-10 environments for training and testing the RL2 algorithm. This ML-10 tasks share the same robot, state space and action space.

2 Related Work

Meta Reinforcement learning methods have been evaluated on a different problems including maze navigation, bandits problems, arcade games and locomotion tasks with varying dynamics.

RL2- Fast Reinforcement via slow Reinforcement Learning:

Rather than designing a "fast" reinforcement learning algorithm, they represent it as a recurrent neural network (RNN) and learn it from data. RL2 , the algorithm is encoded in the weights of the RNN, which are learned slowly through a general-purpose ("slow") RL algorithm. The RNN receives all information a typical RL algorithm would receive, including observations, actions, rewards, and termination flags; and it retains its state across episodes in a given Markov Decision Process

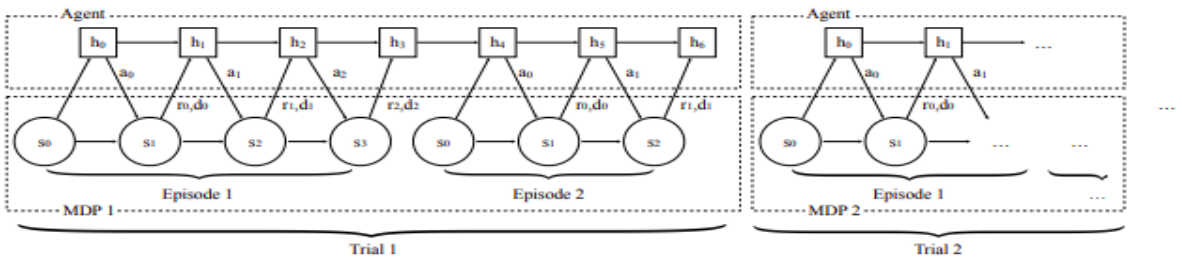


Fig 1: Agent and Environment Interaction

Policy is represented as Recurrent Neural Network Each timestep the next state st+1, action at, reward rt, and termination flag dt are concatenated and provided as input to the policy, which conditioned on hidden state. The policy provides next action and next hidden state. At the end of episode, hidden state is preserved for next episode but not for next trial. In the proposed paper they used TRPO , they implemented RNN using GRU

3 Environment

Meta world consists of 50 shared structure environments like same robotic arm that interacts with different objects. Each environment perform different task like reaching the robot arm to goal position, pushing the object to goal and grasping the object.

The action space is a 2-tuple consisting of the change in 3D space of the end-effector followed by a normalized torque that the gripper fingers should apply. The actions in this space range between −1 and 1. For all tasks, the robot must either manipulate one object with a variable goal position, or manipulate two objects with a fixed goal position. The observation space is represented as a 6-tuple of the 3D Cartesian positions of the end-effector, a normalized measurement of how open the gripper is, the 3D position of the first object, the quaternion of the first object, the 3D position of the second object, the quaternion of the second object, all of the previous measurements in the environment, and finally the 3D position of the goal. If there is no second object or the goal is not meant to be included in the observation, then the quantities corresponding to them are zeroed out. The observation space is always 39 dimensional.

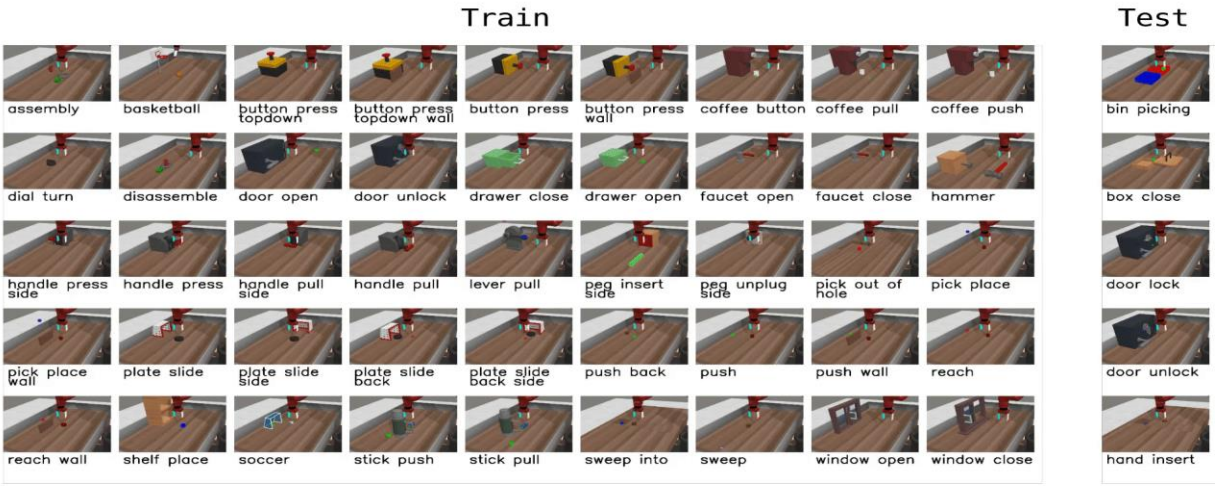


Fig 2: Meta World Environments

Reward values are not directly indicative of how successful a Policy is. There is an evaluation criterion for all the environments. All tasks are involve manipulating one or more objects into a goal configuration

Success metric is typically based on the distance between the task-relevant object and its final goal pose, i.e.‖o-g‖<€ where € is a small distance threshold such as 5 cm.

4 Implementation

We implemented RL2 algorithm in Meta-Learning10 (ML-10), we train on 10 manipulation tasks, and are given 5 new ones at test time.

We used Recurrent Neural Net (GRU) as policy. For Policy optimization we used Proximal Policy optimization algorithm. We used General Advantage Estimation to reduce the variance.

The training Environments are Reach, push, pick and place, Door, Drawer close, Button press, Peg insertion, Window open, sweep Env, Basket ball , and testing environments re are Drawer open, Door close, shelf place, sweep into goal, lever pull.

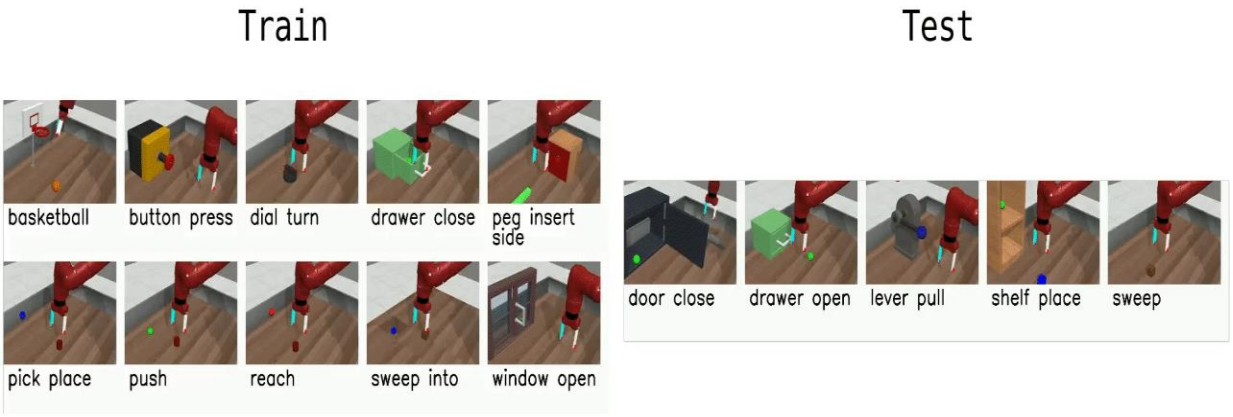
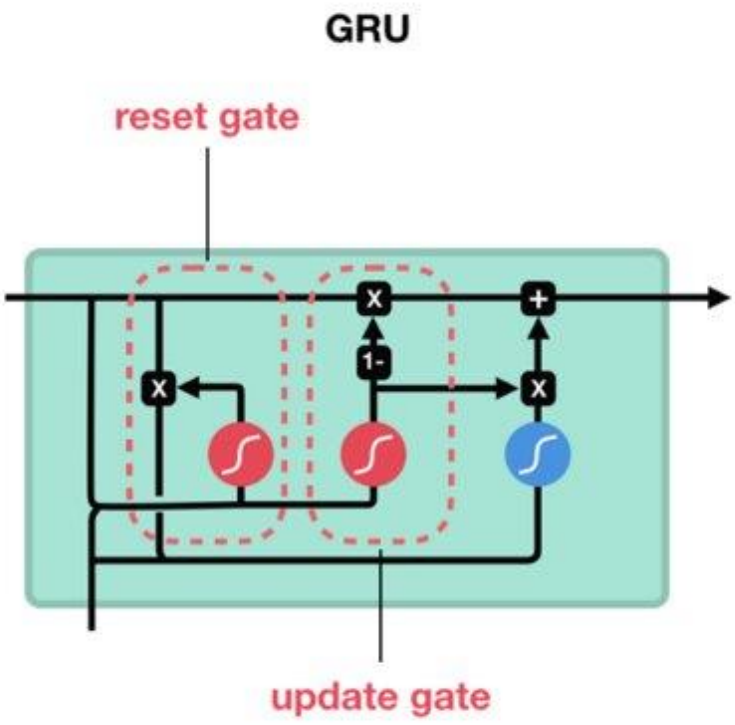


Fig 3 : Meta Learning10 environments

Gated Recurrent Units



GRU's have 2 gates namely "Update" and "Reset"
Update gate decides what information to throw away and what new information to add.
Reset gate is used to decide how much past information to forget.

Proximal Policy Optimization (PPO) :

PPO is on-policy algorithm, it can be used for environments with either discrete or continuous action spaces. PPO is motivated by the question, how can we take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause performance collapse. PPO is a family of first-order methods that use a few other tricks to keep new policies close to old.

There are two primary variants of PPO: PPO-Penalty and PPO-Clip.

PPO-Penalty approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient over the course of training so that it's scaled appropriately.

PPO-Clip doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

Algorithm 1 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Unlike TRPO, which is complicated to implement. PPO is different

PPO uses clipped surrogate objective, which is simpler to implement while retaining performance.

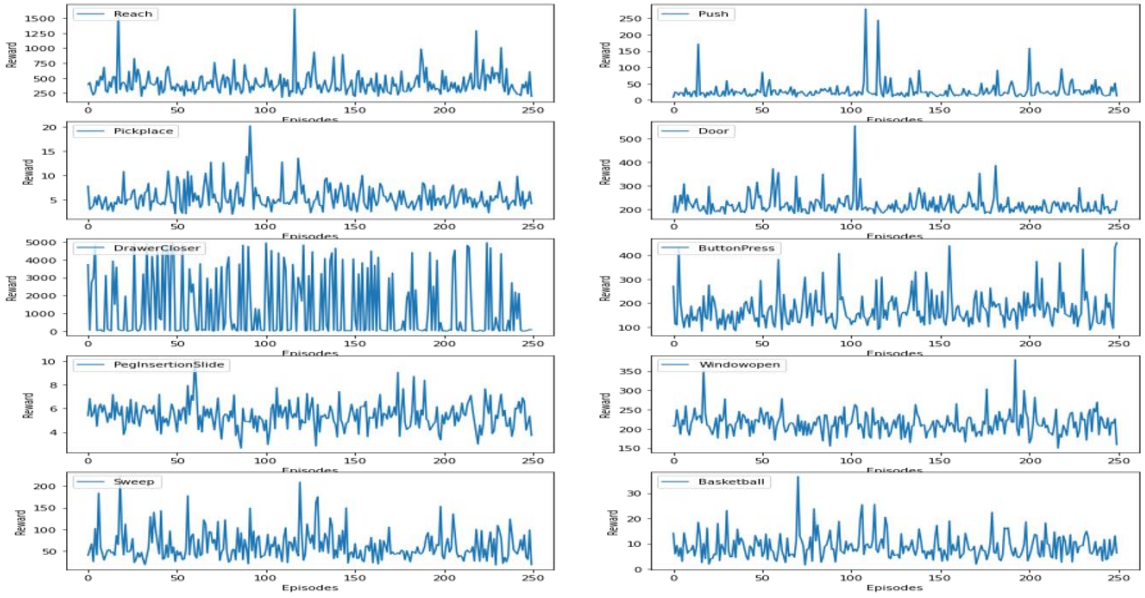
$$J^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\left[\min\left(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a)\right)\right]$$

θ is the policy parameter
$\hat{E}_t$ is the empirical expectation over timesteps
r_t is the ratio of the probability under the new and old policies
$\hat{A}_t$ is the estimated advantage at time t
ϵ is a hyperparameter, usually 0.1 or 0.2

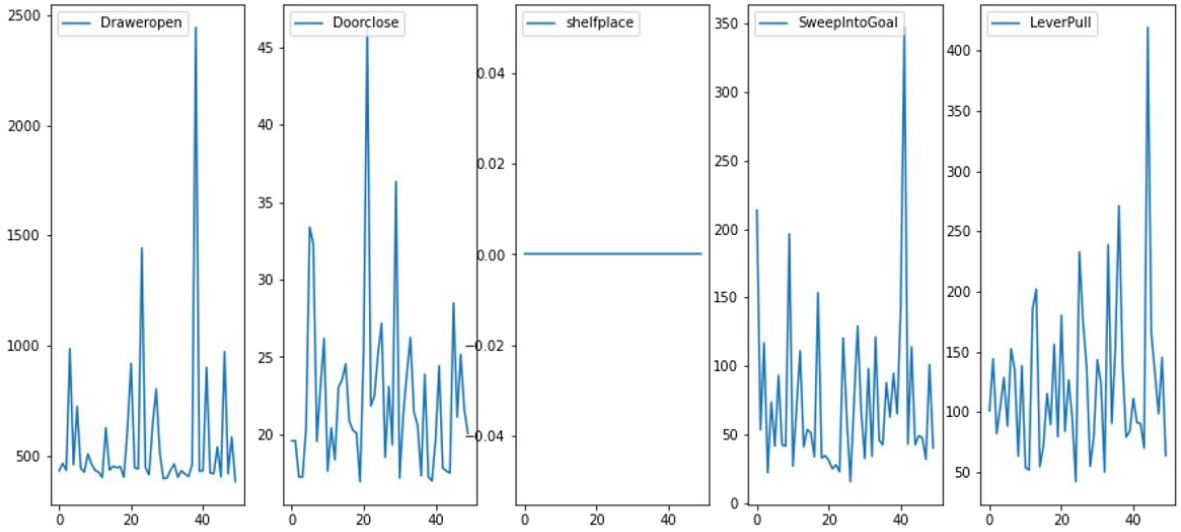
The update for good actions in a state is small, but penalty is high. This makes PPO use the samples better than TRPO

5 Training and Evaluations

The agent is trained in 10 different environments, and the reward is as follows:



After the training we did a fewshot training in 5 new environments which the agent had not seen before. The result of evaluation is as follows:



6 Contribution

Raju – 50%

Jaishyam – 50%

7 Reference

1. Duan, Yan, et al. "Rl $^ 2$: Fast reinforcement learning via slow reinforcement learning." arXiv preprint arXiv:1611.02779 (2016).
2. Wang, Jane X., et al. "Learning to reinforcement learn." arXiv preprint arXiv:1611.05763 (2016).
3. Yu, Tianhe, et al. "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning." Conference on Robot Learning. PMLR, 2020.
4. Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
5. Schulman, John, et al. "High-dimensional continuous control using generalized advantage estimation." arXiv preprint arXiv:1506.02438 (2015).
6. Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).