RL^2 Meta Reinforcement Learning

Matteo Conti
Master in Artificial Intelligence
University of Bologna, Italy
matteo.conti16@studio.unibo.it

Abstract

In this research, we present the results obtained by the application of standard *Meta Reinforcement Learning* techniques (meta-RL, RL^2) to different distributions of environments (with both discrete and continuous action space) using, as a baseline, common Actor-Critic methods such as A2C and A3C. In the final section, we describe some of the possible improvements available in order to enhance our standard meta-learner agent.

1 Introduction

A good *meta-learning*[4] model should be trained over a variety of learning tasks and optimized for the best performance on distribution of environments, including potentially unseen tasks[11].

We considered the concept of *meta-learning*[4] applied in the context of Reinforcement Learning problems. We started by considering one of the basic techniques for developing Meta Reinforcement Learning agents called *meta-RL*[10]. Starting from this one, we have implemented the technique called RL^2[2] (which recalls the previous one *meta-learning*) which is based on the concept that we call *meta memory* (a Recurrent Neural Network for storing agent's past experiences).

Finally, we have tested our meta and non-meta agents both on common environments (unique) instances and multiple environments distribution in order to compare the effectiveness of the *meta-learning* procedure.

2 Actor Critic Methods

Like actor-critic methods, A2C performs online learning: a couple of transitions are explored using the current policy, which is immediately updated. As for value-based networks, the underlying NN will be affected by the correlated inputs and outputs: a single batch contains similar states and actions (e.g. consecutive frames of a video game). The solution retained in A2C and A3C does not depend on an experience replay memory as DQN, but rather on the use of multiple parallel actors and learners[9].

See the reference paper[5] for further details.

2.1 Entropy Regularization

In A3C, the authors added an entropy regularization term to the policy gradient update:

$$H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(s_t, a_t) \log \pi_\theta(s_t, a)$$

$$\nabla_\theta J(\theta) = E_{s_t \sim p^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)(R_t - V_\varphi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))]$$

$H(\pi_\theta(s_t))$ measures the "randomness" of the policy: if the policy is fully deterministic (the same action is systematically selected), the entropy is zero as it carries no information. If the policy is completely random, the entropy is maximal. Maximizing the entropy at the same time as the returns improves exploration by forcing the policy to be as non-deterministic as possible[9].

2.2 Generalized Advantage Estimation (GAE)

Starting from *n-step advantage* formula, what is the optimal value of n? GAE decides not to choose and to simply average all n-step advantages and to weight them with a discount parameter λ. This defines the Generalized Advantage Estimator $A_t^{GAE(\gamma,\lambda)}$[8]:

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda)\sum_{l=0}^{\infty} \lambda^l A_t^l = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

2.3 Implementation

It is necessary to point out that, only for research purposes, our practical implementation differs from the theory in some aspects. In particular, we have summarized all the details of our implementation in the Table 1:

Agent	Advantage Estimate	Entropy	Async	Shared Backbone
A2C	MC[9]	no	no	no
A3C	GAE[8]	yes	no	no

Table 1: agents implementation

3 Meta Reinforcement Learning

A *meta-learning* model is expected to generalize to new tasks (possibly taken from new environments) that have never been encountered during training. The adaptation process, essentially a mini-learning session, happens at test with limited exposure to the new configurations. Even without any explicit fine-tuning (no gradient backpropagation on trainable variables), the meta-learning model autonomously adjusts internal hidden states to learn[11].

Meta Reinforcement Learning, in short, is to do *meta-learning* in the field of reinforcement learning.

3.1 Formulation

Let's say we have a distribution of tasks, each formularized as an MDP (Markov Decision Process), $M_i \in \mathcal{M}$. An MDP is determined by a 4-tuple:

$$M_i = \langle S, A, P_i, R_i \rangle$$

$$\begin{aligned} S &= \text{a set of states} \\ A &= \text{a set of actions} \\ P_i : S \times A \times S \to \mathbb{R}_+ &= \text{transition probability function} \\ R_i : S \times A \to \mathbb{R} &= \text{reward function} \end{aligned}$$

RL^2 paper[2] (starting from the previous research[10]) adds an extra parameter into the MDP tuple to emphasize that each MDP should have, what is called, a finite horizon τ:

$$\tau_t = \langle s_t, a_{t-1}, r_{t-1} \rangle$$

$$\begin{aligned} s &= \text{task space observation} \\ a_{t-1} &= \text{action taken at previous timestep} \\ r_{t-1} &= \text{reward obtained at previous timestep} \end{aligned}$$

Note that common state S and action space A are used above, so that a (stochastic) policy: $\pi_\theta : S \times A \to \mathbb{R}_+$ would get inputs compatible across different tasks. The test tasks are sampled from the same distribution $\mathcal{M}$ or slightly modified version[11].

The Figure 1 shows an illustration of meta-RL[1] formulation containing the two optimization loops.

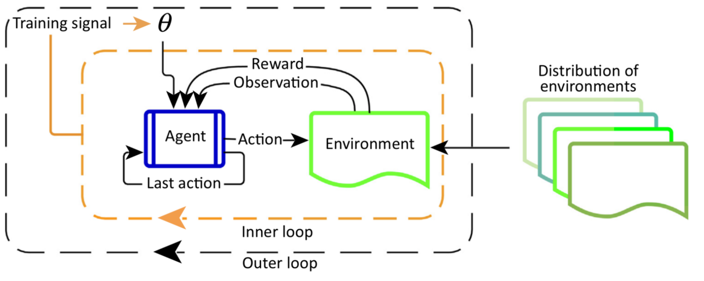

Figure 1: The outer loop samples a new environment in every iteration and adjusts parameters that determine the agent's behaviour. In the inner loop, the agent interacts with the environment and optimizes for the maximal reward[1].

3.2 Differences from Reinforcement Learning

The overall configure of meta-RL[10, 2] is very similar to an ordinary RL algorithm, except that the *last reward* r_{t-1} and *last action* a_{1-1} are also incorporated into the policy observation in addition to the current state s_t.

- In RL: $\pi_\theta(s_t) \rightarrow$ a distribution over A
- In meta-RL: $\pi_\theta(s_t, a_{t-1}, r_{t-1}) \rightarrow$ a distribution over A

The intention of this design is to feed a history into the model so that the policy can internalize the dynamics between states, rewards, and actions in the current MDP and adjust its strategy accordingly. Both meta-RL[10] and RL^2[2] implemented an LSTM policy and the LSTM's hidden states serve as a memory for tracking characteristics of the trajectories[11].

3.3 Training Procedure

The training procedure works showed in Figure 2 as follows:

1. Sample a new MDP, $M_i \sim \mathcal{M}$
2. Reset the hidden state of the model
3. Collect multiple trajectories and update the model weights
4. Repeat from step 1

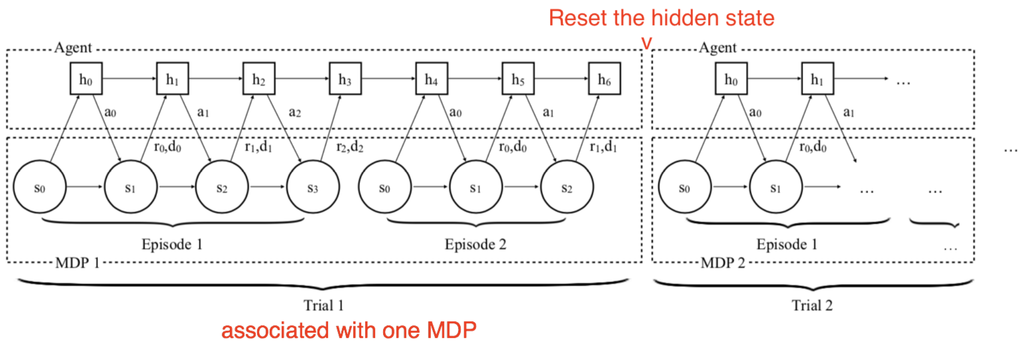

Figure 2: As described in the RL^2 paper[2], illustration of the procedure of the model interacting with a series of MDPs in training time[2].

4 Experiments

In order to test properly out Meta and Non-Meta agents, we have grouped our experiments into two sessions.

In the first session we tested all agents only on **one unique** instance of each environment (type) with the goal of showing that, in theory, Non-Meta agents should, in principle, work better. In the second session, we have tested all agents on the **distribution** of each environment (type) with the goal of showing that, in theory, Meta agents should, in principle, learn to adapt to different samples of each distribution and, as a consequence, work better.

4.1 Environments

We have used different environments in our tests. In particular, some of them, have one or more configurable parameters that we have used to test our Meta Agents.

The list environment's name and descriptions have been summarized in Table 2.

Code	Name	Actions	Meta Parameter	Description
10Brf	10-Arm Bandit	discrete	distribution	Bandits with random (fixed) prob.
2Be	2-Arm Bandit	discrete	distribution	Bandits with prob. ([0.1, 0.9]).
2Bm	2-Arm Bandit	discrete	distribution	Bandits with prob. ([0.25, 0.75]).
2Bh	2-Arm Bandit	discrete	distribution	Bandits with prob. ([0.4, 0.6]).
CartPole [6]	CartPole-v0	discrete	$\sim$	Problem described in [6].
LunarLander[6]	LunarLander	continuous	wind_power	Problem described in [6].
Quadrupedal[7]	Quadrupedal	continuous	task	Robot adapting to different terrains.

Table 2: environments descriptions

4.2 Setup

In order to make a proper comparison between each pair of *(agent,environment)* we have fixed, during the training phase, the hyper-parameters values in order to use the same values across all agents. In particular, we have used the *RMSProp* optimizer with a *batch size* equal to 16 and a *learning rate* equal to $5e^{-5}$.

The implementation of our Meta agents consists of attaching to the standard (nn) backbone of ActorCritic methods an additional nn layer (called *Meta-Memory*) which is a Recurrent Neural Network built on top of LSMT-cell layer (as described in the reference papers[10, 2]).

In Table 3 we have listed all the parameter values that we have used for running our experiments.

		training			inference		
Env	Instances	Trials	Episodes	MaxSteps	Trials	Episodes	MaxSteps
10Brf	1x	1	1000	100	1	100	100
CartPole	1x	1	1000	1000	1	100	1000
LunarLander	1x	1	1000	1000	1	100	1000
Quadrupedal	1x	1	1000	10000	1	100	10000
2Bemh	3x	500	20	100	50	2	100
10Brf	5x	500	20	100	50	2	100
LunarLander	4x	500	20	1000	50	2	1000
Quadrupedal	10x	500	20	10000	50	2	10000

Table 3: environments configurations

5 Results

In Figure4 we can see the results obtained at the end of the first session (Section 4).

As expected, considering the results showed in the previous works[10, 2], the average rewards obtained by our meta agents performing actions, on tasks sampled from a distribution of different environments, is higher than the average reward obtain by non-meta agents on same tasks.

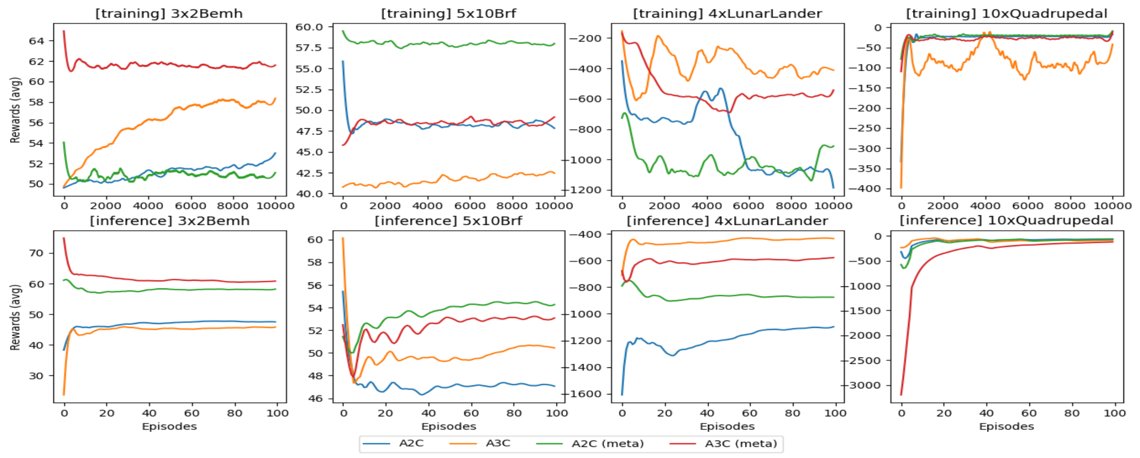

Figure 3: Average Rewards obtained by each agent both during training and inference phases. This results refer to the **second** (distribution of environments) session of tests described in the Section 4.

Moreover, we can see from the Figure 4 that also that non-meta agents have higher performances (in terms of reward outcome), compared to meta agents, when we consider standard Reinforcement Learning problems (we are tasks are sampled not from a distribution of different environments).

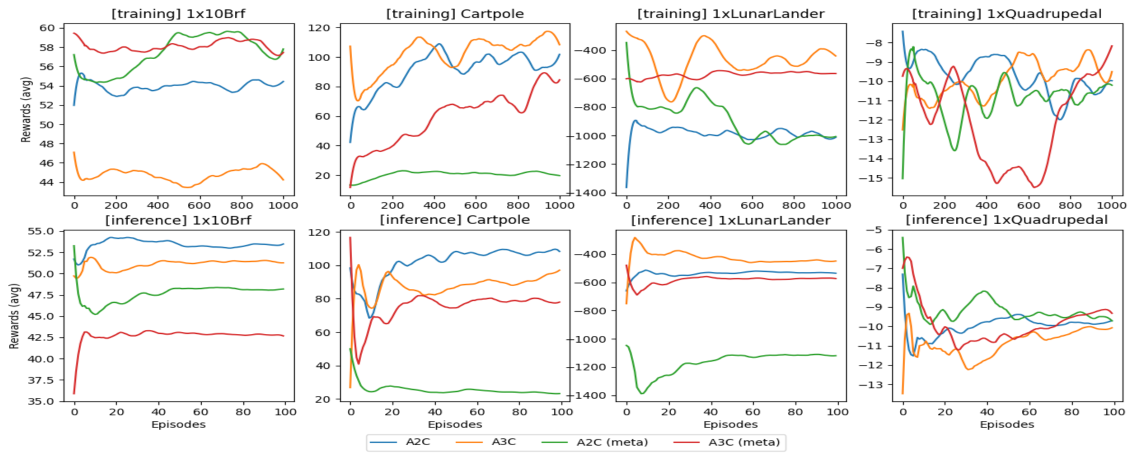

Figure 4: Average Rewards obtained by each agent both during training and inference phases. This results refer to the **first** (one instance of each environment) session of tests described in the Section 4.

Finally, our results highlight also the difficulties of our agents in learning the right policy for environments with continuous action spaces (like *LunarLander*[6] and *Quadrupedal*[7]). This is the symbol of how, probably, the *exploration vs exploitation* strategy that we have used is too naive.

6 Possible Improvements

Meta-learning Hyperparameters The return function in an RL problem, $G_t^{(n)}$ or $G_t^{(\lambda)}$, involves a few hyperparameters that are often set heuristically, like the discount factor γ and the bootstrapping parameter λ. Meta-gradient RL[12] considers them as *meta-parameters*, $\eta = \{\gamma, \lambda\}$, that can be tuned and learned *online* while an agent is interacting with the environment. Therefore, the return becomes a function of η and dynamically adapts itself to a specific task over time[11].

$$\begin{aligned} G_\eta^{(n)}(\tau_t) &= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v_\theta(s_{t+n}) && \text{;n-step return} \\ G_\eta^{(n)}(\tau_t) &= (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_\eta^{(n)} && \text{;mixture of n-step returns} \end{aligned} \tag{1}$$

During training, the policy parameters is updated with gradients as a function of all the information in hand, $\theta' = \theta + f(\tau, \theta, \eta)$, where θ are the current model weights, τ is a sequence of trajectories, and η are the meta-parameters[11]..

Meta-learning the Loss Function In policy gradient algorithms, the expected total reward is maximized by updating the policy parameters in the direction of estimated gradient[8]. The corresponding surrogate loss function for the policy gradient can be reverse-engineered[11]:

$$L_{pg} = E[\sum_{t=0}^{\infty} \Psi_t \log \pi_\theta(a_t|s_t)]$$

This loss function is a measure over a history of trajectories, $(s_0, a_0, r_0, \ldots, s_t, a_t, r_t, \ldots)$. Evolved Policy Gradient [3] takes a step further by defining the policy gradient loss function as a temporal convolution (1-D convolution) over the agent's past experience, L_ϕ. The parameters ϕ of the loss function network are evolved in a way that an agent can achieve higher returns[11].

7 Conclusion

We can see from the results (Figure 3) presented in the Section 5 that, as we expected from the previous works[10, 2], the average rewards obtained by our meta agents performing actions (on tasks sampled from a distribution of different environments) is higher than the average reward obtained by non-meta agents on same tasks.

Moreover, we can note that the total average rewards obtained (for each environment) are not outstanding, demonstrating that the low number of training episodes and the lack of a proper hyperparameter tuning phase significantly affect the results obtained during the inference phase. Indeed, the improvements presented in the Section 6 go in the direction of tackling these types of issues.

References

[1] Mathew Botvinick et al. "Reinforcement Learning, Fast and Slow". In: *Trends in Cognitive Sciences* 23 (Apr. 2019). DOI: 10.1016/j.tics.2019.02.006.

[2] Yan Duan et al. *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning*. 2016. DOI: 10.48550/ARXIV.1611.02779. URL: https://arxiv.org/abs/1611.02779.

[3] Rein Houthooft et al. *Evolved Policy Gradients*. 2018. DOI: 10.48550/ARXIV.1802.04821. URL: https://arxiv.org/abs/1802.04821.

[4] Brenden Lake, Ruslan Salakhutdinov, and Joshua Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350 (Dec. 2015), pp. 1332–1338. DOI: 10.1126/science.aab3050.

[5] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: (2016). DOI: 10.48550/ARXIV.1602.01783. URL: https://arxiv.org/abs/1602.01783.

[6] OpenAI. *Gym*. https://www.gymlibrary.ml. 2022.

[7] PaddlePaddle. *MetaGym*. https://github.com/PaddlePaddle/MetaGym. 2020.

[8] John Schulman et al. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2015. DOI: 10.48550/ARXIV.1506.02438. URL: https://arxiv.org/abs/1506.02438.

[9] Julien Vitay. *Actor Critic*. https://julien-vitay.net/deeprl/ActorCritic.html. 2020.

[10] Jane X Wang et al. *Learning to Reinforcement Learn*. 2016. DOI: 10.48550/ARXIV.1611.05763. URL: https://arxiv.org/abs/1611.05763.

[11] Lilian Weng. *Meta Reinforcement Learning*. https://lilianweng.github.io/posts/2019-06-23-meta-rl/. 2019.

[12] Zhongwen Xu, Hado van Hasselt, and David Silver. *Meta-Gradient Reinforcement Learning*. 2018. DOI: 10.48550/ARXIV.1805.09801. URL: https://arxiv.org/abs/1805.09801.