Self-Tuning Hyperparameters in Fast RL with Generalized Policy Updates

Isaac Wolverton [*1] **Anisha Agarwal** [*1] **Julia Wagner** [*1]

Abstract

The ability to determine optimal hyper-parameter values efficiently is necessary to solve larger, more computationally expensive Reinforcement Learning problems. We argue that combining meta-learning in the form of self-tuning with other reinforcement learning algorithms leads to an improvement in efficiency and accuracy compared to the baseline formulation without self-tuning. Based on this premise, we propose an upgrade of DQN and GPE/GPI – two existent RL formulations – with self-tuning of the discount factor. We compare the accuracy and efficiency of these upgraded formulations against their original baseline formulations. We also compare the upgraded formulations against models trained with discount factors selected from a number of other parameter searches, namely random search, grid search and Bayesian optimization. We will demonstrate the efficiency of self-tuning parameters over a single lifetime compared to selecting parameters with other algorithms that require many lifetimes. We will also demonstrate the increased performance from self-tuning the discount factor. Thus, we show the value of meta-learning when applied to reinforcement learning for the purpose of improving end to end training complexity. Our code can be found here: `https://github.com/IsaacWolverton/6883_final`

1. Introduction

Deep reinforcement learning is a class of machine learning algorithms for training on large, complex environments. The RL problem space consists of value function(s) and a policy. In our project, we deal specifically with the GPE/GPI algorithm, but the shortcomings of this algorithm that we seek to solve exist across many other non-metalearning RL algorithms as well. In this project, we address one issue: the GPE/GPI algorithm still depends on the computationally expensive task of hyperparameter tuning to identify the discount factor. Thus, although generalized policy updates result in a more efficient algorithm, we will make the process even more efficient by significantly decreasing the number of lifetimes required to determine a discount factor. Our goal is to address the inefficiency that results from determining a discount factor by using self-tuning to determine a discount factor. Incorporating metalearning allows the parameters to be tuned within a single lifetime, and uses metalearning rather than depending on human intervention. Self-Tuning within a single lifetime is more efficient in terms of time and computation. It also removes the need for manually changing parameters. We are implementing a meta-learning extension to Fast Reinforcement Learning with Generalized Policy Updates as described in the paper (Barreto et al., 2020). Our approach to reinforcement learning self-tunes the discount factor using meta-learning (Zahavy et al., 2020). We extend the algorithm described in the paper to further increase its efficiency by using self-tuning rather than hyperparameter search to find the optimal discount factor. We compare the effectiveness of the discount factor that results from self-tuning to the effectiveness of the discount factor obtained with a number of hyperparameter optimizations, namely Bayesian Optimization (Snoek et al., 2012), random search and grid search. We also compare the efficiency of obtaining the hyperparameter with self-tuning against the efficiency of obtaining the hyperparameter with Bayesian optimization, random search and grid search. With these comparisons, we see the benefit of using metalearning to automatically tune hyperparameters. We expect the change to improve the efficiency of training compared to the GPE/GPI baseline, as well as the DQN baseline; rather than taking many lifetimes to determine an effective discount factor, we will take just one, potentially allowing modern RL algorithms to scale up to problems currently out of reach.

1.1. Related Works

We build off of two reinforcement learning algorithms in this work, DQN (Mnih et al., 2013) and the GPE/GPI option keyboard. Previous work has been done to apply meta learning self tuning to the DQN via meta-gradients (Xu et al., 2018). This was later expanded to actor critic models in (Zahavy

[*]Equal contribution [1]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts. Correspondence to: Isaac Wolverton <isaacw@mit.edu>, Anisha Agarwal <anisha24@mit.edu>, Julia Wagner <jnwagner@mit.edu>.

et al., 2020). Other researches have created agents that can combine skills into options (effectively macro actions) (Barreto et al., 2019), increasing their efficiency when training in complex environment. Further additions were made to this strategy by adding learnable preferences that improved the efficiency with which an option keyboard agent could select an option (Barreto et al., 2020). We aim to combine these methods and create a self-tuning option keyboard agent with learned preference for yet another improvement to efficiency within complex environments.

2. Methods

We will begin with two baseline formulations: DQN and GPE/GPI. Then we will proceed to implement the improvements that allow for self-tuning of metaparameters via metagradients on each formulation. This includes adding an meta loss function with user set hyperparameters for balancing and periodically updating based on this outer loss function. An online cross validation approach is used while the inner metaparameters are being tuned to be sure the policy does not change too drastically. At this stage we will ensure that our performance matches or exceeds its respective baseline. Currently, we are in the process of implementing the self-tuning improvements to DQN.

We are evaluating the performance of our methodology on the squares and triangles gridworld environment from the GPE/GPI paper since it allows for a direct comparison of performance. The goal of the environment is to collect all of the squares and triangles. Conveniently the reward function can be changed easily to simulate significantly different problems; for example the paper also demonstrates performance on a slight twist of their environment where: upon picking up an object of a certain type, the agent gets a positive reward only if the number of objects of this type is greater or equal to the number of objects of the other type. Otherwise, the agent gets a negative reward.

We will evaluate the overall performance of our project based upon how many environment episodes it takes for the model to converge to a good value of the hyperparameters. The self-tuning agents should converge in one lifetime whereas Bayesian hyperparameter search will require many lifetimes. The addition of self-tuning should improve data efficiency and overall reduce the training time required. We believe that the addition of self-tuning could be a way to effectively reduce overall training time and put more problems in reach of deep RL. Ultimately we will consider this experiment a success if our algorithm can reach the same performance as the GPE/GPI paper with Bayesian hyperparameter optimization, but with significantly fewer training steps and lifetimes.

2.1. Hyper-parameter Tuning

In order to create a baseline of successful parameter values to compare to our self-tuning method, we first tune the hyperparameters using a variety of other methods. In order to incorporate our hyper-parameter-optimization-selected discount factors, we altered and recombined the existent code base. Rather than hard-coding the discount to be 0.9, we take in a single variable discount factor, pass it into the DQN agent, and output the reward. This is used repeatedly within the three hyperparameter-tuning algorithms below. We are currently in the process of writing the function to additionally evaluate the efficacy of the Option Keyboard paper's regressed GPE/GPI model with specialized hyperparameters selected using the three algorithms (Barreto et al., 2019).

2.1.1. Random Search

We started by implementing a Random Search tuning algorithm. For 100 iterations, we choose a random value uniformly from the range $[.7, 1]$. The model is evaluated with the discount factor set to each of these random values. We keep track of the discount factor that resulted in the highest reward as the output of the search.

2.1.2. Grid Search

We additionally implemented Grid Search. With a step size of about .003, we evaluated the model on 100 discount factors from .7 to 1 inclusive. Once more, we found the highest reward among all of the model evaluations with the given discount factor, and this was the output of the search.

2.1.3. Bayesian Optimization

Finally, as our last hyperparameter-tuning algorithm we used Bayesian Optimization. We approximated the model reward using a Gaussian process based on the discount factor input. We evaluated the model with different discount factors 100 times, the first 10 of which are random initialization points. The rest of the evaluations are guided by the surrogate model in order to make smarter parameter choices. As before, the final output is the discount factor that resulted in the highest reward.

3. Results

In this section we demonstrate our results so far on the path to a self-tuning option keyboard agent. We decided to start with DQN as the baseline, per the protocol followed in the Fast RL paper. All of the agents tested so far have been DQN agents in the Fast RL environment that were trained for 10,000 episodes. We started by using our three different hyper-parameter optimization algorithms to get optimized values for the discount factor. We ran random, grid, and

Table 1. Comparison of the three hyperparameter tuning algorithms. Each algorithm was run for 100 agent lifetimes.

METHOD	DISCOUNT	CUMULATIVE REWARD
GRID SEARCH	0.852	6.8885
RANDOM SEARCH	0.912	6.9245
BAYESIAN SEARCH	0.885	7.0609

Bayesian search for 100 agent lifetimes each and compared the resulting discount factors against each other. The discount of 0.885 from Bayesian search ended up performing the best. Figure 1 shows the data from our random search. We generated 100 discount factors drawn uniformly from 0.7 to 1 and trained an agent on each for 10,000 episodes. The reward shown is the evaluation reward from the final episode. These values were smoothed using a simple moving average with a window of 4 for viewing clarity. Figure 1 clearly shows that the optimal discount factor is somewhere between 0.85 and 0.9. Performance begins to degrade outside this range. Our Bayesian optimization confirmed this when it arrived at a discount of 0.885.

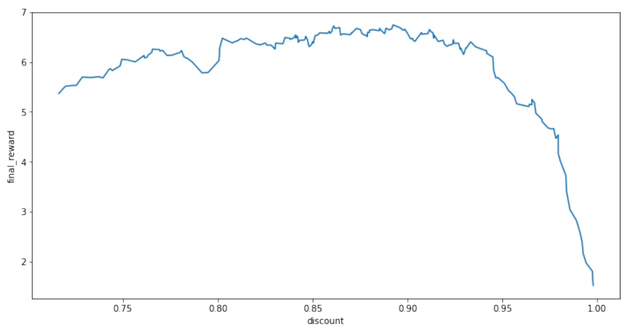

Figure 1. Discount factors and their corresponding rewards after training for 10,000 episodes.

After implementing our brute force hyper-parameter searches, we began to implement a self-tuning version of DQN. This aspect of our project is still a work in progress and there is likely an error in our implementation given its performance. The self-tuning DQN is modeled after the Meta-Gradient Reinforcement Learning paper which alternates between updating normally and updating a meta objective with respect to the meta-parameters (in this case just the discount factor). Figure 2 shows the comparison in performance between this self-tuning DQN and two fixed discount factors, 0.9 (from the original Fast RL paper) and 0.885 (from our Bayesian search). We took 10 samples of each method and the shaded area around each line is the 95% confidence interval. Although the discount factor of 0.9 was close to optimal, its confidence interval does not contain the average performance of 0.885 for the most of the last 5,000 episodes, indicating that 0.885 is likely better for this particular environment. Clearly the performance of the self-tuning DQN does not come close to either of the fixed discount DQN agents. We believe this is likely due to an bug in our implementation. Our evidence for this is Figure 3, a chart of the discount factor of the self-tuning agent over one lifetime. We initialize the discount factor to 0.9 and expect to see it stay in this general area with some perturbation during learning. Instead we see it decline steadily decline each episode until it reaches near 0, at which point the agent is so shortsighted that it cannot learn the fully complexity of the environment.

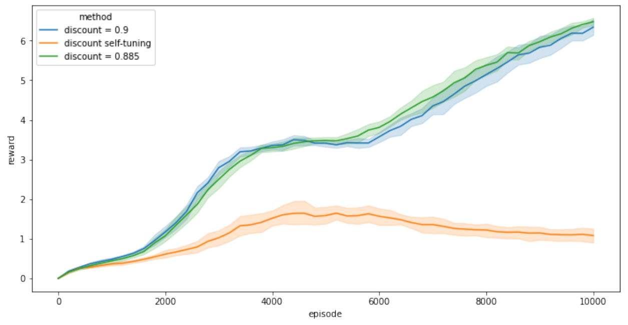

Figure 2. Comparison of two discount factors against our implementation of meta-gradient self-tuning. The shaded region around each line is the 95% confidence interval after ten runs. The 0.9 discount factor comes from the DeepMind work we are building off of. The 0.885 discount factor comes from a one hundred iteration run of Bayesian optimization.

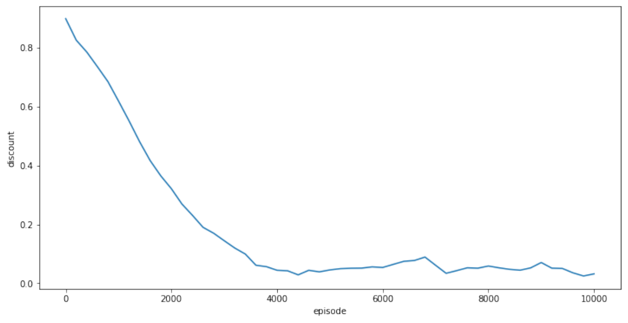

Figure 3. The self tuning agent's discount factor during its lifetime. The initial value provided to the agent is 0.9.

In order to gauge the comparative efficiency of self-tuning compared to parameter search, we analyze wall clock time. A DQN agent takes roughly 7.5 minutes to train fully on an Intel Core i9-9880H CPU. Correspondingly, performing 100 iterations of search (random, grid, or Bayesian) took slightly over 12 hours. In comparison, our self-tuning version of DQN takes 9 minutes to train fully, a 20% increase over the original DQN, but a 98.8% decrease over the total search time. We anticipate that once we have corrected our implementation of the self-tuning DQN that it will continue to have this advantage in overall clock time and thus be a large improvement in efficiency. Once the same self-tuning is added to the option keyboard agent we hope to see a similar improvement, ultimately increasing the complexity of environments to which deep RL can be applied.

References

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., hunt, J., Mourad, S., Silver, D., and Precup, D. The option keyboard: Combining skills in reinforcement learning. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 13052–13062. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9463-the-option-keyboard-combining-skills-in-reinforcement-learning.pdf.

Barreto, A., Hou, S., Borsa, D., Silver, D., and Precup, D. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1907370117. URL https://www.pnas.org/content/early/2020/08/13/1907370117.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms, 2012.

Xu, Z., van Hasselt, H., and Silver, D. Meta-gradient reinforcement learning, 2018.

Zahavy, T., Xu, Z., Veeriah, V., Hessel, M., Oh, J., van Hasselt, H., Silver, D., and Singh, S. A self-tuning actor-critic algorithm, 2020.