HYPERPARAMETER TUNING FOR DEEP REINFORCEMENT LEARNING APPLICATIONS [*]

Mariam Kiran*
Scientific Networking Division
Lawrence Berkeley National Laboratory
USA
mkiran@lbl.gov

Melis Ozyildirim*
Department of Computer Science
Cukurova University
Turkey
mozyildirim@cu.edu.tr

ABSTRACT

Reinforcement learning (RL) applications, where an agent can simply learn optimal behaviors by interacting with the environment, are quickly gaining tremendous success in a wide variety of applications from controlling simple pendulums to complex data centers. However, setting the right hyperparameters can have a huge impact on the deployed solution performance and reliability in the inference models, produced via RL, used for decision-making. Hyperparameter search itself is a laborious process that requires many iterations and computationally expensive to find the best settings that produce the best neural network architectures. In comparison to other neural network architectures, deep RL has not witnessed much hyperparameter tuning, due to its algorithm complexity and simulation platforms needed. In this paper, we propose a distributed variable-length genetic algorithm framework to systematically tune hyperparameters for various RL applications, improving training time and robustness of the architecture, via evolution. We demonstrate the scalability of our approach on many RL problems (from simple gyms to complex applications) and compared with Bayesian approach. Our results show that with more generations, optimal solutions that require fewer training episodes and are computationally cheap while being more robust for deployment. Our results are imperative to advance deep reinforcement learning controllers for real-world problems.

***K*eywords** hyperparameter tuning, deep RL, evolutionary approaches

1 Introduction

Deep reinforcement learning applications are increasingly being used to teach systems how to drive a car, control massive power grids or self-playing games [1, 2], or finding novel neural network design search [3, 4, 5]. With an optimal neural network architecture, a robust deep RL solution can replace controllers in complex environments for model-free optimum control of various complex systems [6]. However, a deep understanding of deep RL approaches is required due to the many challenges of discrete versus continuous state representations, exploration variances, and complex reward functions that can learn optimal actions by just observing the environment [7].

Much work on hyperparameter searching, has been extensively studied in applications containing Convolutional Neural networks (CNNs) for image recognition [8] or general deep learning models [5]. This also includes the development of various hyperparameter search libraries such as Ray, Hyperband, DeepHyper, and even hand-tuning to iterate through multiple settings repeatedly until finding fairly tuned hyperparameters such as layer numbers, neurons, and so on. However, deep RL applications, in particular, have comparatively seen less progress in developing methods for optimal hyperparameter search. This is due to many factors - the need for scaling in both the neural network learning and the simulation involved, the multiple RL algorithms developed to handle value-based or policy learning (e.g such as Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG), to name a few [3]) or just simple allowing the deep RL to learn via trial-and-error. However, as we aim to replace complex controllers with deep RL solutions,

[*] *Citation*: **Both authors contributed equally to this research.**

the process of trial-and-error becomes tedious, time-consuming, and computationally expensive, to evaluate and find the best model that is robust. As the problems become more complex, search space grows and resources are limited, there is a need to develop a practical solution targeted specifically for hyper-parameter search in reinforcement learning applications. Within distributed computing research, deep RL research is witnessing an explosion of applications such as controlling data centers for improving the carbon footprint, robotics, optimizing storage demands, or finding optimum traffic engineering solutions to reduce flow completion times [9].

Improving agent actions through multiple interactions was also seen in seminal works by Sim, where creatures used genetic algorithms to evolve limbs to cope with swimming challenges [10]. Without any ‘learning’, evolutionary algorithms are useful for finding solutions to multi-objective optimization problems and proven successful [11]. Researchers have used genetic algorithms to find optimal settings in a 3-layer CNN [12] but argued that they are too slow due to their mutation only nature [8]. In this paper, we argue that this is not the case and genetic algorithms can be a powerful manner to automate the unconstrained multi-objective search for RL applications. We focus on combining variable-length genetic algorithm search with deep RL problems to develop a scalable approach for finding optimal hyperparameter for deep RL challenges.

Although many hyperparameter search libraries use RL techniques to find optimal parameters [13, 5], to the best of our knowledge, these do not use evolutionary approaches to find optimal solutions and also are not specifically designed to target deep RL applications. The reasons may be twofold, the assumption of infinite compute cycles for trial-and-error episodes and deep RL research is still in its infancy when it comes to finding real-world solutions using deep RL. Other hyperparameter tuning libraries such as Ray, HyperBand or Oputuna allow hyperparameter tuning, but not for deep RL applications. In deep RL, the neural network needs to train (number of episodes) to learn the best reward situations for state and action pair mappings. Therefore, not only do we need optimal hyperparameters, but also the training period to let the deep RL evolve. The goal is to find the best hyperparameters that can allow the agent to reach optimal reward mappings in fewer episodes. In our work, we develop HPS-RL, designed as a scalable deployment library, that generates a gene population, uses crossover and mutation to quickly arrive at the best multi-objective optimal hyperparameters for each RL experiment. HPS-RL is designed to work with multiple deep RL algorithms, optimization, and loss functions, to find the best model which needs less training time and produces the most robust model that can be deployed in real-world systems. The main contributions of this work include (1) an automated multi-objective search for hyperparameters using genetic algorithms to evolve the best deep RL solutions that train in fewer episodes and produce robust solutions and (2) demonstrate the scalability of our architecture on multiple processors that can leverage multiple thread and parallel processing to improve the overall search time for hyperparameters.

Our work aims to bridge the gap between deep RL solutions and hyperparameter tuning, showing how HPS-RL can significantly impact RL research and applications being designed in both gaming, simulations, and robotics domains, showing further how can be scaled for distributed computations. The project is open-source and can be found (in the supplementary document).

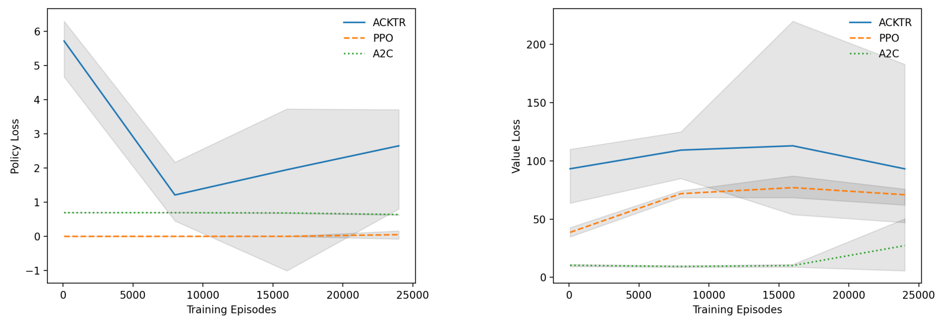
2 Identifying Hyperparameter Impact in Deep RL

A reinforcement learning problem is formulated with an agent situated in a partially observable environment, learning from the environment and past data to make current decisions. The agent receives data as environment snapshots with specific relevant features. Depending on the RL algorithm, the agent computes which action to perform depending on the reward values in the current state. The agent then acts to change its environment, subsequently receiving feedback on its action in form of rewards, until the terminal state is reached. The objective is to maximize the cumulative reward overall actions in the time agent is active [14]. Deep RL research has released a set of environments (or gyms) to develop new deep RL algorithms that can learn optimal actions faster. OpenAI gym experiments are a collection of many control, gaming, and robot problems that users can use to train their deep RL algorithm against [15].

Hyperparameters Impact. As seen in [16], the sensitivity of various deep RL techniques can impact the reward scale, environment dynamics, and reproducibility for the experiments tried. Finding benchmarks for a fair comparison is a challenge, but OpenAI stable baselines can help provide initial comparison across the gym environments chosen. Complex hyperparameters can impact how quickly the RL solutions reach consensus or produce a robust solution that has enough experience to cope in the environment and achieve the maximum reward. Mostly grid search is used to find optimal parameters [3].

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \quad (1)$$

Equation 1 describes the Bellman equation learning optimal action-state pairs from previous states. Exploring the *learning rate* (α) can allow the agent to learn quickly in fewer trial episodes, but not necessarily exploring the space of possibilities. *Gamma* (γ) helps the agent to prioritize immediate rewards versus future rewards in building optimum



(a) Policy loss. (b) Value loss.

Figure 1: Baseline runs in Cartpole Environment showing impact of different RL algorithms on the same problem.

action strategies. Additionally, agents can use a probability *epsilon* (ϵ) to prioritize greedy action searches taken to learn more robust solutions.

Deep RL Algorithms. Most deep RL problems can be framed as Markov Decision Processes (MDPs), consisting of four key elements $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$, taking decisions at each epoch or episode t. The probability that the agent moves into new state s_{t+1} is given by the state transition function $P(s_{t+1}|s_t, a_t)$. In model-free RL, the agent aims to learn a policy to learn how to act. Here, **policy gradient functions** $\pi(a_t|s_t)$ learn a policy mapping from state s_t to action a_t and tells the agent which action a_t to take in-state s_t. **Value function** $V_\pi(s_t)/Q_\pi(s_t, a_t)$ aim to modify the agent's policy based on perceived value of state. Q-learning, deep Q-learning is value-based algorithms, whereas actor-critic, DDPG, leverage both value and policy-based learning to reach a better robust trained agent. Additionally, other examples of algorithms such as TRPO (trust region policy optimization), or PPO (proximal policy optimization), or more, are designed to allow the agent to learn optimal policies quicker, again improving training time. In deep RL literature, most of these algorithms have been developed to tackle discrete versus continuous state and action spaces or improve the training of the agent without compromising the robustness of the solution produced [17]. Figure 1 shows the variability in the rewards learned in multiple deep RL algorithms on the same Cartpole Gym environment from OpenAI. This shows it is important to try out multiple algorithms that is best suited for the application being designed. Various research has developed various versions of deep RL techniques that range from studying and acting in discrete versus continuous states [18], improving the exploration space or the memory buffers which allow the agent to learn from past actions. Combined with variance in methods this poses challenges in reproducibility, where various techniques have shown variable results against baselines, creating a need to investigate how hyperparameters can help build reliable deep RL solutions [16]. Selecting the right algorithms, either value-based or policy-based, can have a significant impact on the results produced [19].

Environment Gyms. Gym (`https://openai.com/`) is a library that provides various reinforcement learning environments to develop and compare algorithms. It is frequently used as a standard environment with several categories such as Atari games, classical control problems, robotics, etc.

3 Bayesian Optimization for Hyperparameter Search

Bayesian optimization methods emerged as an efficient way of hyperparameter search for the problems when the convexity is not guaranteed. When there is no information for fitness function, approximate function is obtained by using Gaussian Kernels which are one type of surrogate models. The next step is finding a function H which shows the next best point. There are four types of H function, which are generally preferred; expected maximization, maximum probability of maximization, and upper confidence bound, entropy search. Although finding the best point by using

H is another optimization function itself, gradient based optimization is applied. Algorithm 1 shows the Bayesian Optimization process [20, 21].

Algoritmo 1: Bayesian Optimization Using Gaussian Kernels

```
1 Evaluate fitness function $f(x)$ on $n$ random points
2 para $n$ = 1 to N hacer
3     Update Gaussian Posteriors by using all data in $(X_{1:n}, f(X_{1:n}))$
4     Compute next best point function $H(x, X_{1:n})$ using the Gaussian Kernel
5     Evaluate $f(x)$ on $x_n + 1$ which maximizes the $H(x, X_{1:n})$
6     Add $x_n + 1$ to $(X_{1:(n+1)}, f(X_{1:(n+1)}))$
7 fin
8 Return $x_i$ evaluated with the largest $f(x_i)$
```

4 HPS-RL: Hyperparameter Search using Genetic Algorithm (GA)

The design of HPS-RL is a multi-objective optimization problem. In this paper, we use a genetic algorithm (GA) based search method to perform this multi-objective optimization to allow a trade-off between competing variables in the hyperparameter space.

Genetic algorithms have been used as optimization methods in multiple research areas [22]. Modeling the search routines as a Darwinian evolution theory allows optimum (or fittest) solutions to emerge from a group or population of solutions. Experiments in [10] show that evolution was a successful strategy to produce solutions that survive the longest or, in our setting, more robust for RL challenges. For HPS-RL the GA requires:

- Developing a scheme to represent the hyperparameters as a gene, which is part of a population (of many individuals).
- Developing a 'survival of the fittest' method, to measure which gene performs well - the selection process.
- A computationally efficient method to evaluate the individuals for trials - highest rewards achieved in fewer training episodes.
- A method to perform crossover and mutation, to generate new individuals from past well-performing individuals, for a new population generation.

4.1 Selection and Fitness Mechanism

Figure 2 shows the overall process of HPS-RL. We initialize individuals with a block of hyperparameters being explored. For example, in a DDPG case, the basic gene block looks like $(\gamma, \alpha, no.of neurons)$. Additional hyperparameters such as activation functions, ϵ, number of layers, and more, can also be explored. Based on the example, with 4 blocks of individuals, each set is trained with the chosen gym environment and the model is saved. Then the saved model is evaluated for fitness by playing with the gym and recording the cumulative reward playing for 100 steps.

After fitness evaluation, the roulette wheel selection mechanism can allow a random selection among the well-performing parents [23], $p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$, where N is the number of individuals in the population and p_i allows a higher probability of the 'fitter' parents to be chosen for the next generation. The fitness of the parents is calculated using $f_i = \frac{1}{n} + \sum_{k=1}^{n} r_k + \frac{1}{\sum_{k=1}^{n} l_k}$, where n is the number of episodes being played, r is cumulative over n episodes and l represents the total loss incurred during this evaluation.

Crossover and Mutation. The new generation of selected individuals become parents of the new offspring by performing crossover and mutation properties. As shown in Figure 2, at a random point genes are swapped from one parent to the second, to generate two new individuals. For example, represented $(\gamma, \alpha, no.of neuron)$ - Parent 1 (0.01,0.1,10) and Parent 2 (0.5,0.8,50), will become - Child 1 (0.01,0.1,50) and Child 2 (0.5,0.8,10).

In mutation, we select one variable and randomly replace it. For example Child 1(0.01, 0.1, 50) will become - Child 3 (0.01,**0.5**,50). The new generations are then evaluated again via training and fitness measurements to allow better generations (or solutions) to be obtained. The crossover and mutation rates can be altered to either allow a slower change in population dynamics or to allow the system to explore multiple optimal fronts to find the best solutions.

Variable-length genes for the algorithms. Deep RL research has developed many deep RL algorithms ranging from DQN, DDQN, DDPG to complex algorithms ACKTR. Each algorithm differs by the optimization functions and manners

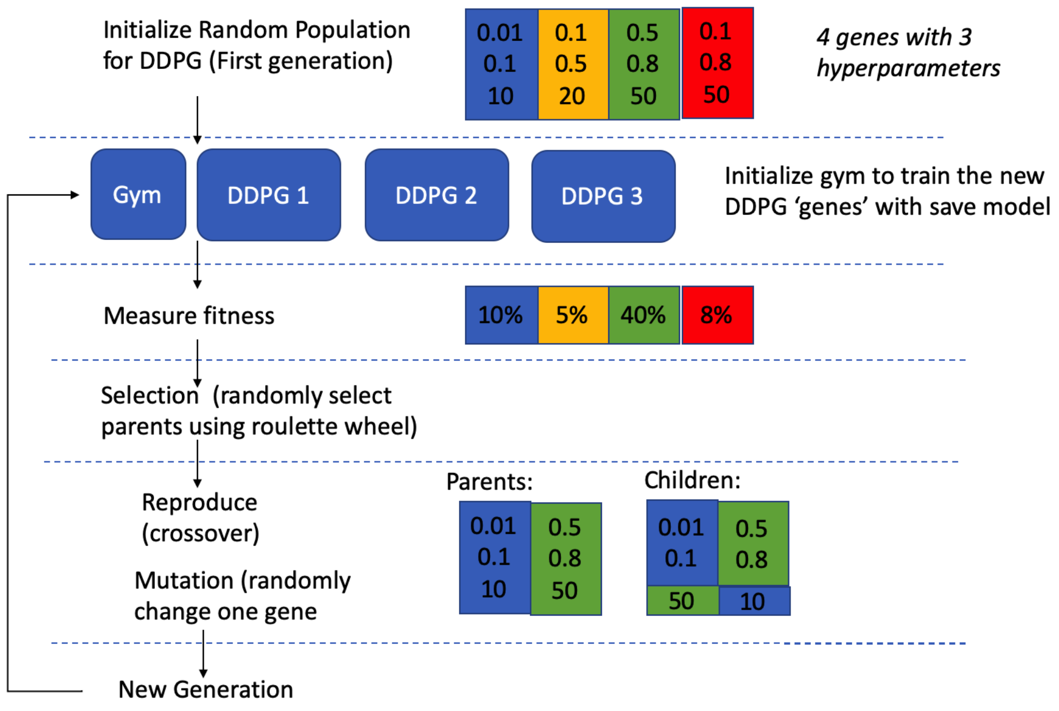

Initialize Random Population for DDPG (First generation)
0.01 0.1 10
0.1 0.5 20
0.5 0.8 50
0.1 0.8 50
4 genes with 3 hyperparameters
Gym
DDPG 1
DDPG 2
DDPG 3
Initialize gym to train the new DDPG 'genes' with save model
Measure fitness
10%
5%
40%
8%
Selection (randomly select parents using roulette wheel)
Reproduce (crossover)
Mutation (randomly change one gene
Parents:
0.01 0.1 10
0.5 0.8 50
Children:
0.01 0.1 50
0.5 0.8 10
New Generation

Figure 2: Example GA search for the best hyperparameters for DDPG in a chosen Gym.

in which the rewards are learned. With this challenge, the hyperparameters that can be tuned in each algorithm are based on the optimization functions as part of their architectures. For example, a DDPG gene ($\alpha, no.layers, neurons$) is different from an ACKTR gene ($\alpha, no.layers, neurons, KL-divergence$). Algorithm 2 shows how GA can help find optimal solutions.

Algoritmo 2: GA for HPS-RL

```
1  Initialize gene population $p$, rate of crossover $c$, rate of mutation $m$, evaluation episodes $e$
2  Generate $p$ possible hyperparameter solutions to explore
3  **para** *episode = 1 to p* **hacer**
4      Train the deep RL chosen with each solution
5      Save the respective deep RL models
6      Evaluate the trained models, by evaluating the saved models for $e$ episodes
7      Determine the fitness of the solution evaluated, reward, training episodes and loss incurred
8      Perform roulette wheel selection
9      Select parents
10     Perform crossover and mutation depending on rates $(c, m)$
11     Generate children and add to new population generation to try $p + 1$
12 **fin**
13 Return best solutions in $p$
```

5 Software Description

Our open-source HPS-RL software contains three Python subpackages: (1) a collection of genetic algorithm functions, (2) benchmark gym environments, (3) benchmark deep RL algorithms with optimization functions, to explore. The current suite can be extended with mpi4py implementations for distributed computations. Currently, we run these models as multiple collections of jobs from a single head node. Figure 3 shows the high-level architecture of how the head node generates the population and maintains a parameter server that collects the fitness among all the individual searches. HPS-RL is designed as a distributed library and can be extended on HPC platforms. Currently, we exploit the multiprocessors. The hardware we used for our testing has Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 64 GB RAM, 12 cores, and Nvidia GeForce RTX 2070 GPU processor which has 2304 Cuda cores and 8 GB RAM.

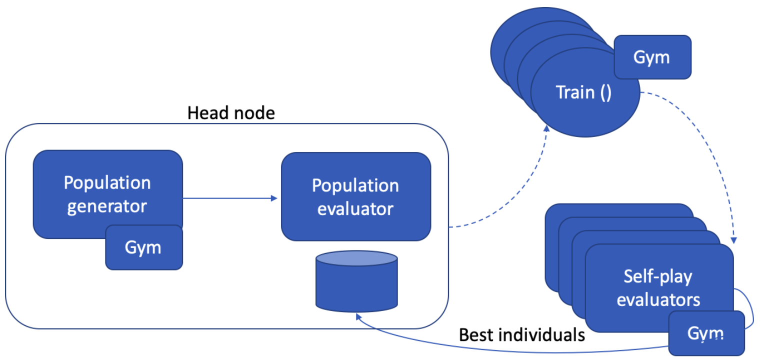


Figure 3: Distributed HPS-RL architecture. The head node can set up multiple nodes to run multiple training and evaluation nodes for each individual. All workers will update the parameter server in the head node to find the optimal hyperparameter for the deep RL application.

5.1 HPS-RL Deep RL algorithms

HPS-RL packages its deep RL algorithms for researchers to explore. HPS-RL provides more deep RL algorithms listed in Table 1.

5.2 HPS-RL Optimization functions

HPS-RL packages its optimization function variants, to allow researchers to explore more approaches, other than the standards stochastic gradient descent [24]. Gradient descent is a common optimization algorithm for training neural networks. Based on the idea of updating the tunable parameters to minimize the objective function, and uses the learning rate to converge the loss function. Conjugate gradient works on finding an optimal coefficient β to prevent fluctuation. CG leads to a fast convergence, it often results in poor performance. Approaches like Broyden-Fletcher-Golfarb-Shanno Algorithm use Quasi-Newton methods approximate the Hessian value to solve unconstrained optimization problems. BFGS algorithm is one of the efficient Quasi-Newton approaches. Levenberg - Marquardt Algorithm is an optimization method for solving the sum of squares of non-linear functions. It is a combination of gradient descent and the Gauss-Newton method. It starts with gradient descent and as it comes close to the solution, it behaves like a Gauss-Newton method, which it achieves by using a damping factor. While a larger damping factor results in gradient descent, a small damping factor leads to the Gauss-Newton method.

6 Experimental Results

In this section, we evaluate HPS-RL and its capability to find optimal parameters for deep RL applications. We evaluate the experiment on three deep RL applications (gym environments).

6.1 Gym Environments

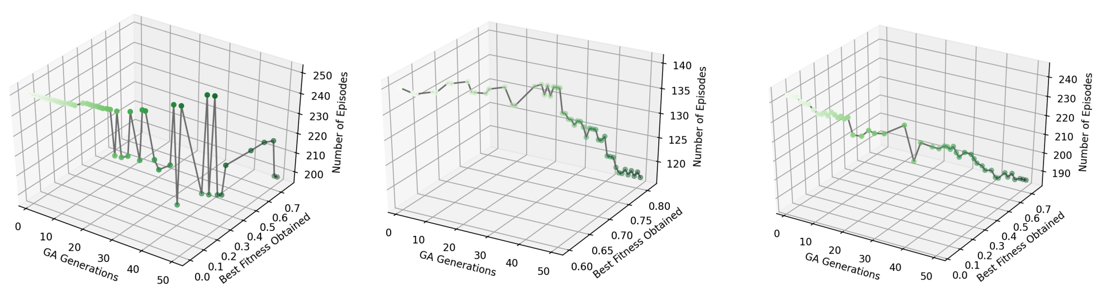
Cartpole Environment. Part of the OpenAI Gym environments, CartPole is a classical control problem of an inverted pendulum problem which is connected to a pole, to balance the pole for as long as possible [25]. The pole is linked to a cart, which moves along a track. The state, action details are: **State:** The agent reads in 4 continuous variables which represent the current environment, position of the pole, cart, angle, and velocity. **Action:** The action is continuous to move left or right by a certain amount of degrees. **Reward:** A reward of +1 is obtained every time step the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical.

Lunar landing. Another environment from OpenAI gym, this game involves trying to control the landing of a ship on a landing pad. The state, action, and reward are as follows: **State:** The state observation space is the current image of the environment represented as 8-dimension continuous variables. **Action:** The action space is discrete, *do nothing, move left,* or *move right*. **Reward:** The agent receives a reward every time it moves if it crashes the episode ends and the game restarts. Additionally, there is +100 or -100 depending on the success of the landing.

Autonomous laser control. Taken from a real experiment, that is using deep RL to control high power lasers for advanced particle accelerators, this experiment aims to perform coherent beam combining (CBC) [26]. The state, action is as follows: **State:** The state observation space is a 5x5 matrix of continuous variables representing the intensity of

Parameter	DQN	DDPG	TRPO	A2C
Episodes (50, 200, 500)	✓	✓	✓	✓
Gamma γ (0.01,0.1,0.5,0.99)	✓	✓	✓	✓
Learning rate (0.1, 0.01, 0.001)	✓	✓	✓	✓
Batch Size	✓	✓	✓	✓
Neurons No.	✓	✓	✓	✓
Layers	✓	✓	✓	✓
Optimization function (Adam, CG LBFGS, LM)	✓	✓	✓	✓
Activation function (tanh, relu)	✓	✓	✓	✓
Trajectory Size (10, 20, 50, 100, 1000)				✓
KL value (0.001, 0.01, 0.1)				✓

Table 1: Hyperparameters for deep RL algorithms.



(a) Cartpole.

(b) Lunar Landing.

(c) Autonomous Laser.

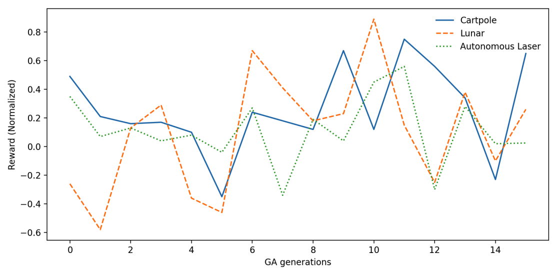
Figure 4: More generations find optimal deep RL solutions, better fitness and less training episodes.

the refracted beams. **Action:** The action is a 3x3 continuous space, tuning 9 beams to achieve maximum combining efficiency. **Reward:** Every time the beam moves it receives a +1 reward and a major +100 when the combining has achieved maximum values.

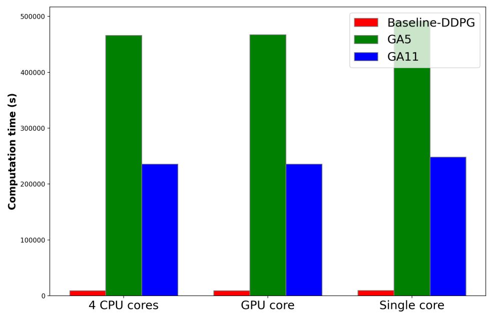
6.2 Baseline Comparisons

With many RL algorithms, OpenAI gym also released stable baseline implementations of many algorithms DQN, DDPG, TRPO, A2C, and more. Working on discrete and continuous observation and action spaces, the algorithms are aimed to allow the research community to test algorithms quickly to see the impact of the deep RL in the environment being built [27]. Figure 1 are stable baseline outputs for Cartpole, showing the variance among the different deep RL algorithms chosen for the same problem. These graphs show why it is important to have a hyperparameter search library that can find the best hyperparameters for the deep RL application.

As deep RL research matures, having more control of hyperparameters and internal functions is needed to develop new deep RL algorithms. HPS-RL is released with its library of deep RL algorithms that allow researchers more flexibility to try multiple function enhancements, potential leading to new RL algorithms. These implementations are based on [28]. Table 1 shows the hyperparameters that can be controlled via GA in HPS-RL in these algorithms. ACKTR and A2C use formulas that allow KL convergence to find trust regions to explore and find optimal values. These formulas are not part of other deep RL algorithms and thus not tuning parameters for those.



(a) Rewards show an upward trend as new generations are produced.



(b) Computational time.

Figure 5: HPS-RL results.

6.3 Evaluation of GA search

In this section, we describe the results of the search for the best hyperparameters for the 3 gym environments chosen. We evaluate the results to find how many generations does it take to find the best fitness and eventually best performing deep RL neural network architectures.

Figure 4 shows the comparison of generations, with the number of episodes and fitness achieved. We wee that in all gym environments, the more generations (up to 50) the solutions found can converge to better fitness and also less number of episodes required for training the algorithms. This shows the solutions have evolved to compute hyperparameters that can build robust deep RL solutions, that take fewer episodes for trial-and-error to learn about the system. These produced solutions are more robust for the problems. Figure 5a shows different GA structures and normalized reward values for the tested gym environments. We ran the experiment for 44 generations, and best individuals in each generation was captured in Table 2.

Generation	LR	Gamma
GA0	0.001	0.025
GA1	0.01	0.025
GA2	0.1	0.025
GA3	0.001	0.25
GA4	0.01	0.25
GA5	0.01	0.25

Table 2: Best performing individual in each generation.

6.4 Comparison of Multi-core Run Time

The proposed approach is compatible with GPU, multicore CPUs and single core implementations. During the tests, 4 of 12 CPU cores and all CUDA cores were utilized and Python concurrency library is used for parallelism. The graph shows Cartpole baseline implementation, the tested GA structure GA-5, and GA structure GA-11. The baseline implementation does not include any HPS-RL to find optimal parameters, it is just one run and the fastest approach. However, when we start looking for optimal hyperparameters using HPS-RL, GA-5 shows that this takes more time than GA-11 which requires a fewer number of episodes. Moreover, these comparisons show that running on GPU or 4 CPUs performs similar, but increases the computational time of GPU. With extensions using mpipy, this processing time can be further improved with parallelism, to be extended in future.

7 Related Work

RL Applications: Reinforcement learning applications are being studied for investigating self-learning or self-optimizing behaviors in complex decision-making scenarios, e.g. self-driving cars [17]. The authors also discuss the impact hyperparameters can have on the baseline performance of an RL solution. These include neural network architecture, reward scale, and random seeds. Raising interesting concerns on reproducibility, making an important case for benchmark RL solution as a generalizable solution. In [29], different RL algorithms such as value-function approximation, actor-critic policy gradient, and temporal-difference were compared on the Cartpole benchmark system. The authors also proposed a method for integrating RL and swing-up controllers.

Hyperparameter search: RL itself can be used to find optimum parameters such as in CIFAR and other complex CNN applications [30]. However, these searches often assume infinite compute resources to find optimum solutions over millions of hours, which is always not the case.

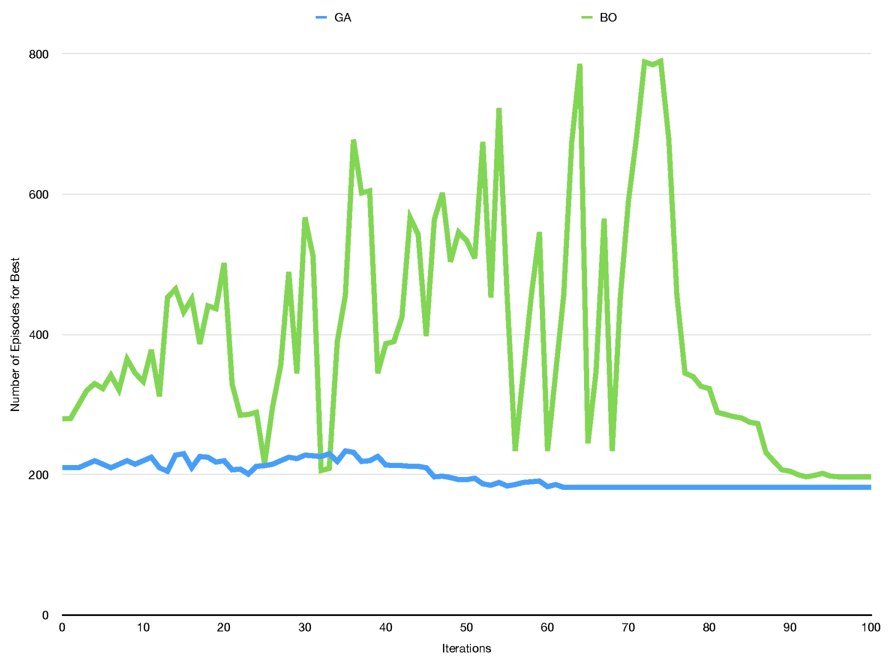


Figure 6: Number of Episode comparison of GA and BO for achieving best value.

Similar to HPS-RL, population-based searches have been used to tune hyperparameters using a population of agents, and shown to be effective for RL applications. The authors also argue about the computational limitations to find optimum solutions in small research labs [31]. Wang et al [32] tuned loss functions to optimize the search space based on reward versus random. This was also seen in [33], where they tune validation loss to yield good models in fewer training episodes. Further, researchers [34] combined Bayesian optimizations with bandit solutions to produce solutions better than available hyperparameter tuning libraries such as HyperBand.

Hyperparameter tuning Libraries. There are many libraries defined for hyperparameter tuning. Each of them has different strengths, compatibility, and weakness. Some of the frequently used libraries are Ray-Tune which supports state of the art algorithms such as population-based training, Bayes Optimization Search (bayesOpSearch); Optuna which provides easy parallelization; Hyperopt which works both serial and parallel ways and supports discrete, continuous, conditional dimensions; ml machine utilizing Bayesian optimization; Polyaxon which is for large scale applications [35]. In this work, however, HPS-RL is specifically designed to target deep RL applications to find optimal and robust deep RL solutions for control or gym environment problems, which is, to be best of our knowledge, still missing in current implementations.

7.1 Comparison with Bayesian Optimization

The larger number of hyperparameters to be searched, the lower efficiency is obtained from Bayesian optimization [36]. In this section, we compared the number of iterations required to reach the optimal solutions with GA and Bayesian Optimization approach on Cartpole env. Figure 6 shows the comparison.

8 Conclusions

We develop a scalable genetic algorithm-based hyperparameter search for deep RL applications. HPS-RL packages its deep RL algorithms, optimization functions, and gym environments to allow researchers to explore deep RL benchmarks and also develop their algorithms. More importantly, using GA for multi-objective optimization, multiple hyperparameters are allowed to evolve over some generations to produce optimal robust deep RL solutions. We experimented with three gym environments, to show that over more generations, HPS-RL can find optimal RL architectures which can achieve higher rewards in fewer episodes. The main advantage of using GA over Bayesian Optimization is that they can be implemented in parallel while original Bayesian optimization process cannot. The only way of paralelizing Bayesian optimization is that implementing acquisition function for multiple points [37]. Also, Bayesian approaches work well on continuous hyperparameters however in RL environments early episodes rely on randomness. Bayesian optimization requires large number of episodes to get optimized results. For early episodes, it behaves like random search. Moreover, the number of parameters is very important to get efficient result from Bayesian approach. As a conclusion, due to the random nature of RL problems in early stages, GA will provide more exploration and provide better parameters than that of Bayesian approach.

Our future work will include extending HPS-RL with mpipy to allow distributed processing over large supercomputing to allow researchers to quickly find optimal solutions. Using GAs allows them to define the number of generations, which also allows researchers to work with a limited number of compute hours. HPS-RL has the potential to accelerate deep RL research. With a scalable search and easy-to-understand solution, researchers can automate the search and produce more reliable and robust deep RL architectures for RL applications.

9 Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science Early Career Research Program for 'Large-scale Deep Learning for Intelligent Networks' Contract no FP00006145.

References

[1] Steven Spielberg, Aditya Tulsyan, Nathan P. Lawrence, Philip D. Loewen, and R. Bhushan Gopaluni. Toward self-driving processes: A deep reinforcement learning approach to control. In *Process Systems Engineering*, 2019.

[2] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[3] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. RLlib: Abstractions for distributed reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062, StockholmsmÃ€ssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[4] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association.

[5] P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild. Deephyper: Asynchronous hyperparameter search for deep neural networks. In *2018 IEEE 25th International Conference on High Performance Computing (HiPC)*, pages 42–51, 2018.

[6] Yan Li, Kenneth Chang, Oceane Bel, Ethan L. Miller, and Darrell D. E. Long. Capes: Unsupervised storage performance tuning using neural network-based deep reinforcement learning. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, New York, NY, USA, 2017. Association for Computing Machinery.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[8] Xueli Xiao, Ming Yan, Sunitha Basodi, Chunyan Ji, and Yi Pan. Efficient hyperparameter optimization in deep learning using a variable length genetic algorithm, 2020.

[9] Junjie Zhang, Zehua Guo, Minghao Ye, and H. Jonathan Chao. Smartentry: Mitigating routing update overhead with reinforcement learning for traffic engineering. In *Proceedings of the Workshop on Network Meets AI and ML*, NetAI '20, page 1–7, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, page 15–22, New York, NY, USA, 1994. Association for Computing Machinery.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[12] Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M. Patton. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[13] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2018.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, USA, 2018.

[15] Ankit Choudhary. (technical report) a hands-on introduction to deep q-learning using openai gym in python. `https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/`, 2019.

[16] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[17] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[19] Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control, 2017.

[20] Roman Garnett. Bayesian optimization.

[21] Stathis Kamperis. Bayesian optimization for hyperparameter tuning.

[22] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[23] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford Univ. Press, 1996.

[24] Buse Melis Ozyildirim and Mariam Kiran. Do optimization methods in deep learning applications matter?, 2020.

[25] (technical report) getting started with gym. `https://gym.openai.com/docs/`,year=2019.

[26] Bashir Mohammed, Mariam Kiran, Dan Wang, Qiang Du, and Russell Wilcox. Deep reinforcement learning based control for two-dimensional coherent combining. In *Laser Congress 2020 (ASSL, LAC)*, page JTu5A.7. Optical Society of America, 2020.

[27] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[28] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Packt Publishing, 2018.

[29] S. Nagendra, N. Podila, R. Ugarakhod, and K. George. Comparison of reinforcement learning algorithms applied to the cart-pole problem. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 26–32, 2017.

[30] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2017.

[31] Jack Parker-Holder, Vu Nguyen, and Stephen Roberts. Provably efficient online hyperparameter optimization with population-based bandits, 2021.

[32] Xiaobo Wang, Shuo Wang, Cheng Chi, Shifeng Zhang, and Tao Mei. Loss function search for face recognition, 2020.

[33] Hadi S. Jomaa, Josif Grabocka, and Lars Schmidt-Thieme. Hyp-rl : Hyperparameter optimization by reinforcement learning, 2019.

[34] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018.

[35] Sivasai Yadav Mudugandla. 10 hyperparameter optimization frameworks. `https://towardsdatascience.com/10-hyperparameter-optimization-frameworks-8bc87bc8b7e3`, year=2020.

[36] Kevin Jamieson Giulia DeSalvo Afshin Rostamizadeh Li, Lisha and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. 2016.

[37] Rodolphe Le Riche Ginsbourger, David and Laurent Carraro. Kriging is well-suited to parallelize optimization. *Computational intelligence in expensive optimization problems*, page 131–162, 2010.