Report – Mini Project 1

This report is written by Ruyi TANG and Guanyu CHEN.

Part 1 choices we made

In this report, we installed different libraries and chose different parameters for different parts of the expriment, and we will explain our choices of libraries and parameters for each part.

Step 1 Actor-Critic Algorithm

In the Step 1, we establish an Actor-Critic Algorithm and then we provide a plot functions to separately plot the learning curve and heatmaps and finally we test our functions with the given parameters and we slightly made changes.

Libraries Used

In the Actor-Critic Algorithm and plot functions we mainly use the four libraries as follows:

numpy as **np**: Used for initialization (np.zeros, np.ones), norm calculation (np.linalg.norm), and generating random numbers for learning rate sampling in random search (np.random.choice).

matplotlib.pyplot as **plt**: Utilized for plotting the learning curve and heatmaps (plt.imshow, plt.savefig, plt.colorbar).

gym: Used for setting up and managing the environment (gym.make), specifically for the "MazeMDP-v0" environment in this case.

os: Used to handle file paths and directories (`os.makedirs`, `os.path.join`) for saving results and plots.

video and **matplotlib.animation**: To visualize dynamically the steps of learning curve, and save the video in the corresponding path.

Key Parameters

alpha_critic = 0.5, alpha_actor = 0.5: represents the critic's and actor's learning rate, we use 0.5 as a starter since in step 3 we will compare other tuning algorithm with optimized alpha_critic and alpha_actor with (0.5,0.5) from naive actor-critic algorithm.

nb_episodes = **150**: we change the *nb_episodes* from `100` to `150` as suggested in the later steps.

init_v = **0.0**: we set the initial value of the value function in the actor-critic algorithm as 0.

Uniform = **True**: we set the state initialization method in the actor-critic algorithm as *True* to start from a uniformly random state at the beginning.

timeout = **200**, **nb_repeats** = **5**, **render** = **False**: we use these parameters from the original ones in *ac_params.*

log_dir = **./tmp/actor_critics**: we create a directory to save the generated plots of learning curves and heatmaps

save_curves = **True**, **save_heatmap** = **True**: to use *save_learning_curve_plot* and *save_heatmap_plot* function

env = {**name**: **MazeMDP-v0**, **width**: **5**, **height**: **5**, **ratio**: **0.2**, **render_mode**: **"rgb_array"**}

we use the same parameters as in the *ac_params*, but to use it as an environment we use *gym.make* to create one with these parameters.

Step 2 Tuning hyper parameters

In Step 2 we optimize the *alpha_actor* and *alpha_critic* with 3 different methods, including: **Grid Search**, **Random Search** and **Bayes Optimization method**. In this section, we first separately define the three algorithms to get the optimized learning rates (*alpha_actor* and *alpha_critic*) and plot the heatmap for visualization.

Libraries Used

In **Grid Search** and **Random Search**, we didn't install any new libraries and we mainly used: *numpy* as *np*, *matplotlib.pyplot* as *plt*, and *os*. And in Random Search, we use *np.random.uniform* to randomly choose learning rates for actor and critic from 0.01 to 1.0.
In **Bayes Optimization method**, we use the library of optuna, optuna automates the process by intelligently sampling hyperparameter values.
Grid Search explores **a fixed set** of hyperparameter combinations, and **Random Search** explores **random combinations**. Optuna improves on these methods by guiding the search based on past results. It uses Bayesian optimization to find the most promising hyperparameters.

Key Parameters

Grid Search

num_points = **20**: This means that we are testing 20 different actor and critic learning rates values for the grid search in the learning rate range, meaning that we are creating a grid of 20*20 (400 combinations).

learning_rate_range = **np.linspace(0.01, 1.0, num_points)**: Range (0.01, 1.0) is a common choice for learning rate in reinforcement learning, with a value lower than 0.01 being too small to make meaningful updates, and a value higher than 1.0 can be unstable for learning.

best_actor_lr = **None**, **best_critic_lr** = **None**, **best_value_norm** = **-np.inf**: These variables store the best learning rates and the best value norm found during the grid search.
We choose -∞ as the initial value for best_value_norm to ensure that the calculated norm in the Actor-Critic algorithm will only.

value_norms = **np.zeros((num_points, num_points))**: In Grid Search, we initialize the value_norms as a 20*20 grid filled with [0].

mdp = env,
num_runs = ac_params['nb_repeats'],
nb_episodes = ac_params['nb_episodes'],
timeout = ac_params['timeout'],
alpha_critic = ac_params['alpha_critic'],
alpha_actor = ac_params['alpha_actor'],
render = ac_params['render']

These are the parameters created or derived directly from the given parameters, which will be used repeatedly in the other 2 methods.

Random Search

total_training_runs = 400, num_random_samples = total_training_runs: To make comparison with the 400 total evaluations of Grid Search, we continue to use 400, but here it means that we are randomly sampling 400 learning rate pairs (*alpha_actor*, *alpha_critic*) and run the random search process for 400 times.

learning_rate_range = (0.01, 1.0): We still keep the range of the learning rate as (0.01, 1.0), but here we randomly choose the learning rates within this range.

Bayes Optimization

alpha_actor = trial.suggest_float("alpha_actor", 0.01, 1.0)
alpha_critic = trial.suggest_float("alpha_critic", 0.01, 1.0):
Same as before, but here we use the optuna library to suggest the value of *alpha_actor* and *alpha_critic* within the given range.

Step 3 Statistical Test

This is where we compare the performances of different tuning methods.

Libraries Used

The bootstrapped package given in the notebook is unused in the code, therefore we won't expand on this package.

scipy.stats.ttest_ind: Used in the run_test function to perform a Welch's t-test, to compare two sets of performance data to determine if they have significantly different means.

numpy as **np**:
Used for: Random sampling (np.random.randint).
Calculating mean, median, standard deviation, and percentiles (np.nanmean, np.nanstd, np.nanpercentile, etc.).
Array operations and indexing.

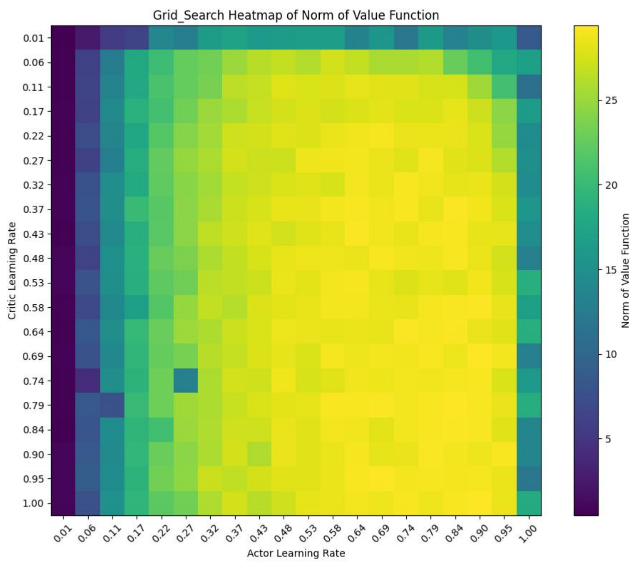
matplotlib.pyplot as **plt**:
Used for: Plotting performance metrics over time (plt.plot).
Adding filled regions for error margins (plt.fill_between).
Creating scatter plots to highlight points where the models performances are significantly different (plt.scatter).

Key Parameters

In the collection of learning curves, we repeatedly used the same set of parameters from previous steps, but to learn the influences of num_runs, we first perform tests when num_runs=5(ac_params['nb_repeats']) then we make more tests with num_runs=15 and num_runs=50.

Part 2 Heatmap obtained in the tuning algorithms

Grid Search

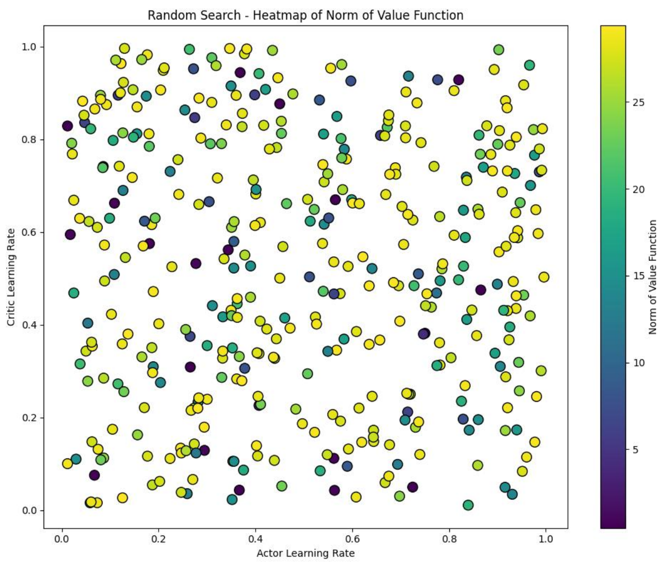


This heatmap for grid search is showing 20*20 grid(400 cells), and with warmer color, the norm of the value function is higher, meaning the performance is higher, and the optimized learning rate is inside the cell with the warmest color.

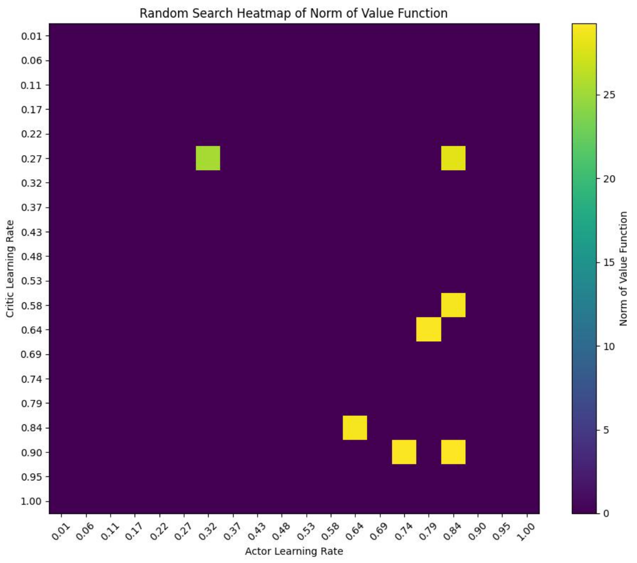
In our test, we find the optimized learning rate pair is (0.58,0.79) which is in a warm-colored square as shown in the heatmap above.

Random Search

Since we are randomly choosing the learning rates within the range of (0.01,1.0), the learning rate pairs are shown as scatter points as below.



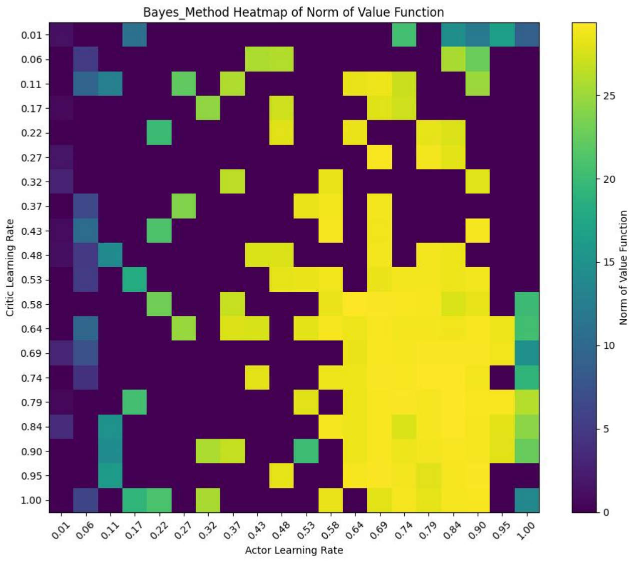
But to better demonstrate and make comparison with other tuning methods, we map the scatter points generated by the random search method to the 20*20 grid. To do this we need to find the nearest point on grid for each scatter point, and then add the norm value to the corresponding grid cell.



Since the most part of the space have not been sampled in the random search method, meaning the rest space has the norm value of 0. Though we can see that the scatter points are multicolored, when they are mapped to the grid, the cell remain purple.

And in this test we got the optimized learning rate pair as (0.75,0.89) which falls in a yellow cell on the heatmap above.

Bayes Method



In the Bayes Optimization method, it uses the probabilistic model to guide the search process. While the first few trials may appear random, but over time, the search becomes more focused and more likely to improve performance. The learning rate pairs are also scatter points, but to make better demonstration, we directly map them on the grid.

Since every search in Bayes Optimization aims for the best results, the heatmap shows a larget amount of warm-colored cells.

And the optimized learning rate pair (0.44,0.23) also falls in a yellow cell.

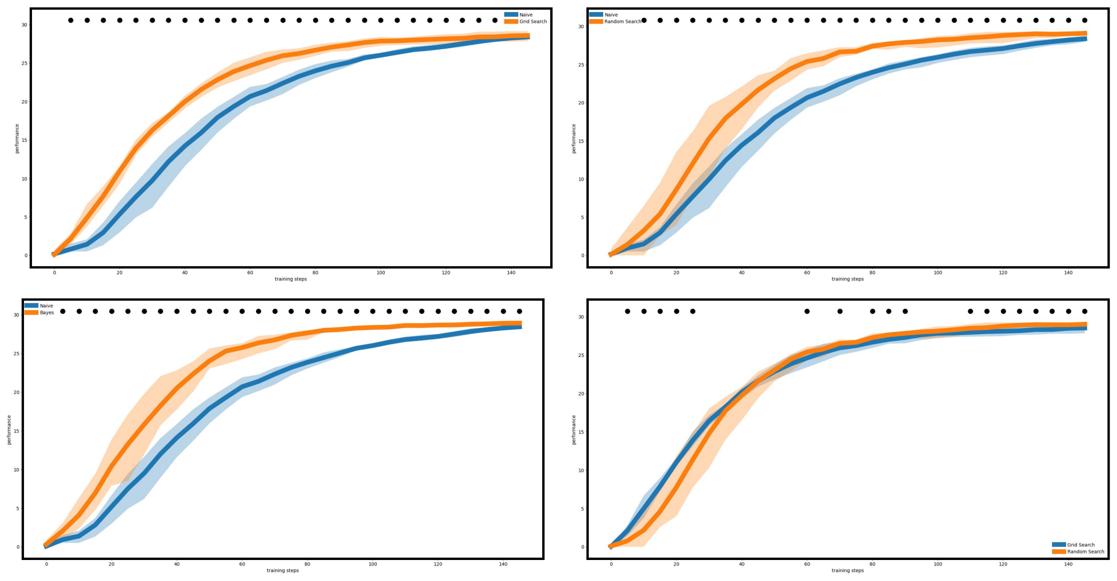
Part 3 Welch's T-test

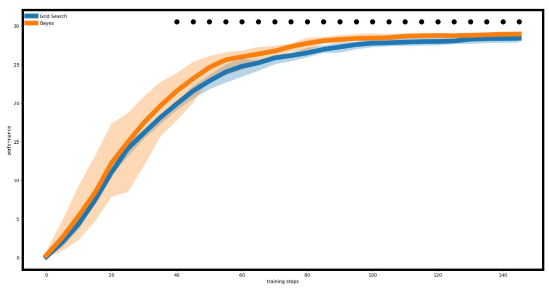
In this part we compare the performances of Grid Search, Random Search and Bayes Optimization by plotting the learning curves, in contrast to the original Naive Actor-Critic algorithm.
We compare the performances 2 by 2 and in 3 cases: when number of runs is 5 (original parameter), 15 and 50 to see if the number of runs influences the performances of each method.

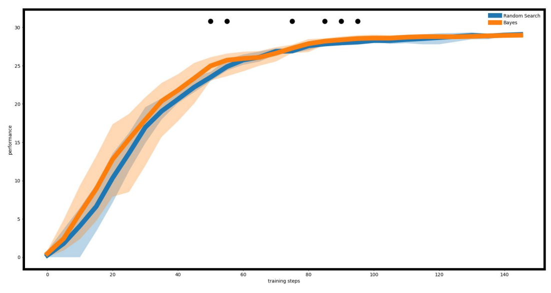
Num_runs = 5

When num_runs=5, we can see that both Random Search and Bayes Optimization performs better than the Grid Search, and these three methods are all better than the Naive algorithm.

From the distribution of time steps, it can be seen that the three algorithms are greatly distinguished from the original Naive Actor-Critic algorithm. But for the Grid Search, from training steps 0-20, it seems to be collapsed with the Naive algorithm.







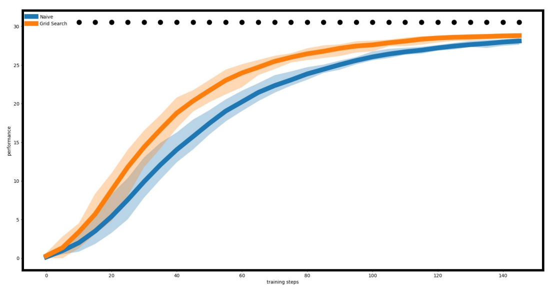
From Both the curves and the time steps we can see that the Grid Search is greatly distinguished from the other two methods of worse performances.

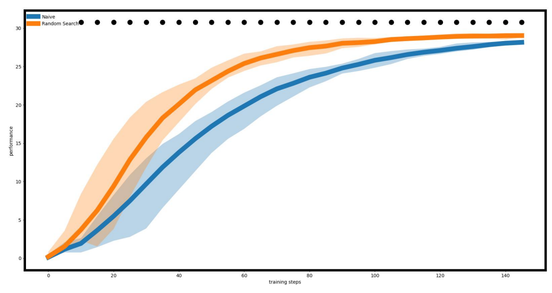
Between Bayes method and Random Search, the curves are relatively close.

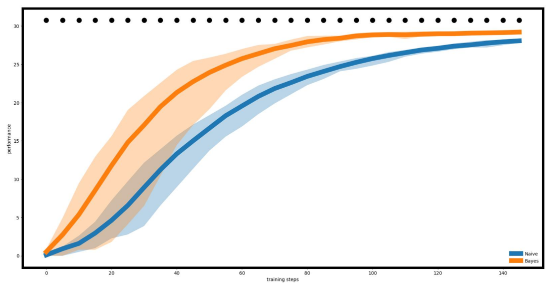
Num_runs = 15

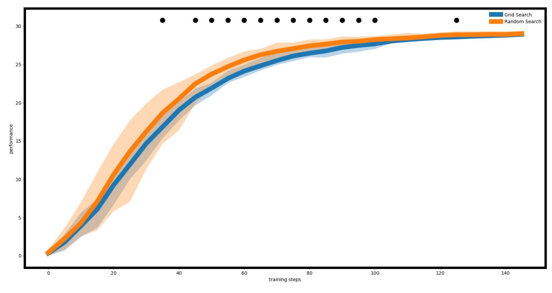
The curves are basically the same as the case when num_runs=5 but for the curves of Naive and Grid Search, it is shown that they are more closely attached, it is also indicated by the time step.

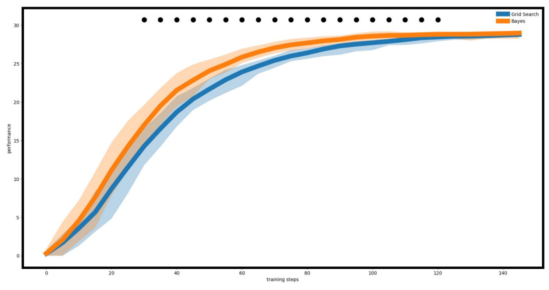
The main difference is that the shaded areas for all algorithms are smaller, meaning that uncertainty in performance at each step is decreased.

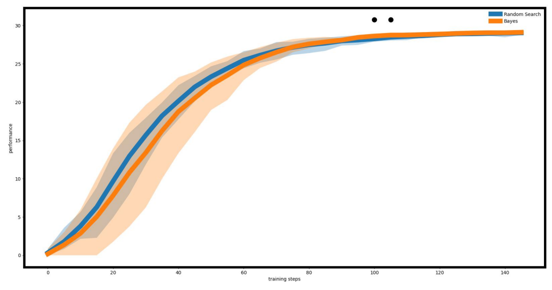








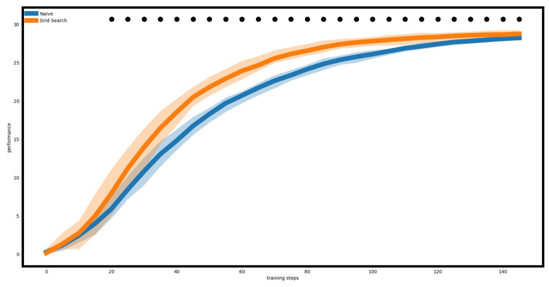


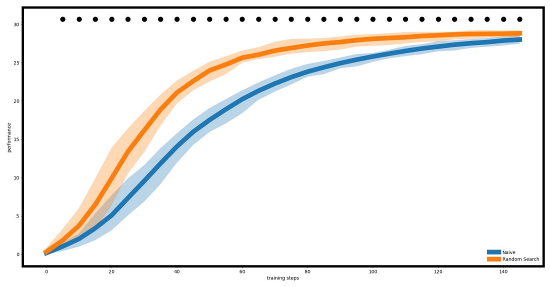


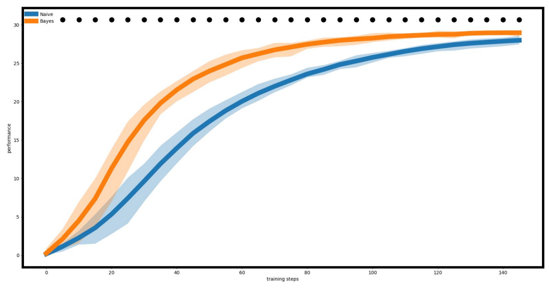
The results are basically the same as the nums_runs=5 case, except for the image of Grid Search VS Random Search. Though the two curves are still very close, they are distinguished better compared to the case of num_runs=5. It is also indicated by the increase of time steps.

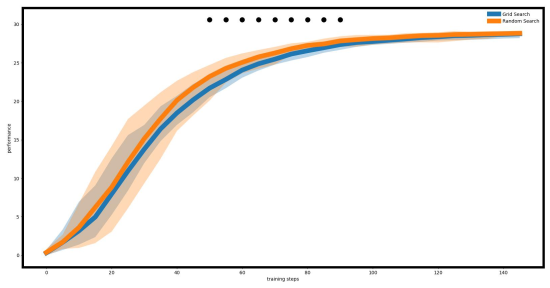
Num_runs = 50

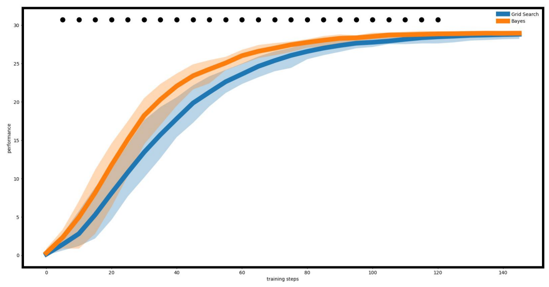
In this case, in all three images the curves are better distinguished, with the indication of time steps it means that the algorithms are significantly differentiated. The Bayes method and Random Search still performs better than Grid Search which performs better than the Naive algorithm.

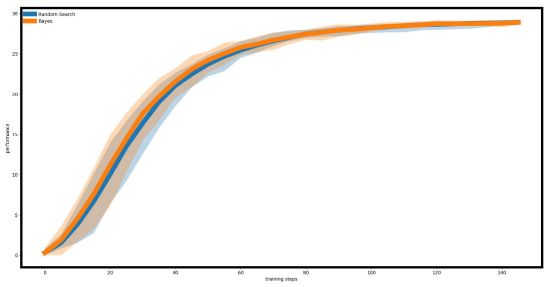












We can see that in this case, the curves of Bayes method and Random Search are basically collapsed and there are no time step indicating significant differences, meaning the models don't have distinctly different behaviors in these steps.

Conclusion

To conclude and compare the performances of these three tuning algorithms, we consider the following indicators:

time of search:
By comparing the running time of the codes of these three tuning algorithms, we find that the Bayes gets the results the quickest by 1 minute, then it is Random Search by 4 minutes and lastly the grid search by nearly 6 minutes.

norm value:
Since in our algorithm, the goal is to maximize the norm value. We find that in our tests, Bayes method wins by the best norm value of 29.40, then it is Grid Search with the value of 29.39, and finally by a slight gap, the Random Search with the value of 29.23.

learning curve:
In our experiments, with 3 different number of runs, we find that the Bayes method and the Random Search are the best. And the Naive algorithm performs distinctly worse than the other three methods.

So we can conclude from the experiment that the best tuning algorithm is Bayes method. We can see that, while the performance on the value function is similar between Grid Search and the Bayes method, and the learning curve is similar between Random Search and the Bayes method, the Bayes method is significantly faster than the other two in terms of runtime. Overall, the Bayes method demonstrates the best performance.