Enhancing Reinforcement Learning Using DQN with Distributed Training for Taxi and Cartpole

Liza Trundle, Ann Li, Bailey Phillips, Shelton Kwiterovich
University of Virginia
emt8kdn, al7gc, bmp5qd, vsk9hw@virginia.edu

Abstract

Reinforcement learning (RL) has shown remarkable success in training intelligent agents to play complex games. However, as game complexity increases, so does the computational demand for training high-performing RL models. This challenge motivates our exploration of techniques to accelerate training and optimize resource usage while enhancing model performance and reduce training costs. We propose a distributed RL framework that leverages Deep Q-Networks (DQN) and Ray to efficiently train models for playing games like Taxi and Cartpole. Our hypothesis is that by parallelizing computation across multiple nodes and systematically tuning hyperparameters for Taxi and Cartpole, we can significantly reduce training time, achieve higher average rewards, and improve training stability compared to single-node setups.

1 Introduction

Building upon the success of reinforcement learning (RL) in training agents for complex games, this project aims to extend and generalize the capabilities of RL models to play a variety of games effectively. While our previous work focused on the Lunar Lander environment, we now seek to apply the lessons learned and techniques developed to new challenges, specifically the Taxi and Cartpole games. Taxi and Cartpole present unique challenges compared to Lunar Lander, despite their similar complexity. These games have different inner mechanisms, objectives, and state-action spaces, requiring the adaptation and refinement of our RL approach. By tackling these new environments, we aim to demonstrate the versatility and robustness of our framework.

The key goals of this project are:

- Extend and generalize RL capabilities for playing games like Taxi and Cartpole
- Build models that excel at these games while optimizing resource usage
- Leverage DQN and distributed training with Ray to parallelize computation and accelerate training
- Achieve higher average rewards and improved training stability

To guide our experiments and analysis, we aim to answer the following research questions:

- How does distributed training with Ray impact the training speed and model performance compared to a single-node setup?
- What hyperparameter configurations lead to the best performance for Taxi and Cartpole?
- How does the complexity of the game environment affect the training time and stability?

To achieve these goals, we will build upon the actor-based RL framework used in our previous work on Lunar Lander. This framework, based on the Deep Q-Network (DQN) architecture, will be extended to incorporate distributed training using Ray. By partitioning the training process across multiple nodes, we aim to parallelize computation and accelerate the learning process.

Moreover, we will optimize hyperparameters by exploring various configurations and evaluating their impact on model performance and training. This aims to maximize average rewards and improve training stability. We will conduct experiments to assess the effectiveness of our approach, evaluating key metrics such as average reward and training stability. These results will provide insights into the benefits and limitations of our distributed RL framework and guide further refinements. This project aims to contribute to the advancement of RL techniques for complex game environments. By demonstrating the effectiveness of our approach on Taxi and Cartpole, we hope to facilitate the development of efficient, scalable, and robust RL models for a range of challenges.

2 Design and Implementation

This section contains an introduction to our specific games and description of the steps we took to train the models. For

both the Taxi and CartPole games, the original parameter configuration is described in table 1.

Parameter	Value
Number of agents (num_agents)	4
Number of epochs for agent training	10
Batch size (batch_size)	64
Buffer size (buffer_size)	100,000
Discount factor (gamma)	0.99
Soft update interpolation parameter (tau)	0.001
Learning rate (learning_rate)	0.0005
Update network every N steps (update_every)	4
Max time steps per episode (max_time_steps)	10
Total training episodes (num_episodes)	1000

Table 1: Default Training Parameters

2.1 Taxi

The Taxi environment is a part of the Toy Text environments, where the objective is to navigate a 5x5 grid world and transport a passenger from a randomly chosen starting location to one of four designated pickup/drop-off locations (Red, Green, Yellow, Blue). The taxi starts at a random square, and the passenger spawns at a random location. The player must drive the taxi to the passenger's location, pick them up (action 4), drive to their destination among the four locations, and drop them off (action 5). The observation space consists of 500 discrete states encoding the taxi's row and column position, the passenger's location (0-3 for Red-Blue, or 4 if in the taxi), and the destination (0-3 for Red-Blue). Each step yields a reward of -1, with +20 for a successful drop-off and -10 for illegal pick-up/drop-off actions. The episode terminates after a successful drop-off or after an predetermined maximum number of steps.

2.1.1 Setup

To adapt the original assignment code for the Taxi-v3 environment from OpenAI Gym, several modifications were made to the starter code provided. The primary changes involved updating the environment configuration, state and action space dimensions, and input preprocessing.

The environment was updated in the taxi-dqn.ipynb code to instantiate the Taxi-v3 environment, with a state size of 500 (one-hot encoded representation) and an action size of 6, corresponding to the available actions in the Taxi game.

Additionally, a new function to get the one hot encoding was introduced to preprocess the states received from the environment. This function converts the integer state representation to a one-hot encoded vector of length 500, suitable for input to the neural network.

The QNetwork architecture remained unchanged, with two fully connected hidden layers of 64 units each and a final output layer of size equal to the action space dimensionality. However, the input layer was modified to accept the one-hot encoded state vectors of size 500.

The Agent class now instantiates the Taxi-v3 environment and uses the one hot encoding function to preprocess the states before passing them to the Learner. The GlobalNet class was modified to initialize the test environment as Taxi-v3 and handle the one-hot encoded state representation during testing.

We also adjusted the visualization code to suit the gym taxi environment. Instead of displaying an animation of images, as done in Lunar Lander, we animated the printing of text frames.

2.1.2 Parameter Configurations

We adjusted the hyperparameters in two different sets: hyperparameters pertaining to the machine learning component of the model, and hyperparameters pertaining to the distributed learning component of the model. The values for both are shown in Table 2.

The machine learning hyperparameters include learning_rate, gamma, and tau, where gamma is the discount factor and tau controls the soft update of parameters. After our experiments with tau, we found that the default of $1e-3$ led to the best performing models. Increasing or decreasing the value did not lead to any improvements. To determine the best combination of machine learning hyperparameters, we chose 3 levels of learning rate and 3 levels of gamma, and trained 9 models for each combination. The best performing model achieved a average game score of 9.7 in the shortest number of epochs and with the shortest elapsed duration of time.

The distributed learning hyperparmaeters are num_agents, num_epochs_actor_train, batch_size, and update_every. We chose three levels of num_agents, num_epochs_actor_train, and batch_size to test systematically. As we tested, we found that the quickest runtime for 4 agents and a num_epochs_actor_train of 5. This optimal parameter set was assessed across different batch sizes and update_every configurations, but no improvement in runtime was observed. Lastly, we explored various num_agents settings. Similar to Cartpole, Taxi is a relatively simple game, so a large number of virtual machines and agents were unnecessary for better performance. This means there were negligible differences in runtime for num_agents = 4 (with num_epochs_actor_train = 5) and num_agents = 1 (also with num_epochs_actor_train = 5). However, as num_epochs_actor_train increased beyond 5, the performance with only one agent vastly decreased.

2.1.3 Results

The best performing machine learning and distributed learning tuned models are visualized in comparison with our untuned baseline model in Figure 1. As the plot indicates, we

	num_agents	num_epochs_actor_train	batch_size	gamma	learning_rate
Default	4	10	64	0.99	5e-4
ML Tuned	4	10	64	0.99	5e-3
DL Tuned	4	5	64	0.99	1e-3

Table 2: Hyperparameters and values for different taxi models.

were able to significantly improve the efficiency of training by tuning either the machine learning hyperparameters, or the distributed learning hyperparameters.

The runtime of the tuning process was drastically reduced from 900s to under 200s. That is a 4.5x speed up. Similarly, the number of episodes decreased from 900 to 200. Looking at the training progression for each model, we can see that the default model's training was highly volatile and its average game score strongly fluctuated throughout training. Even after reaching the positive game score ranges, its performance was unable to hit the cutoff score of 9.7, so it oscillated for approximately 300 seconds (600 episodes) until it did achieve that mark.

In comparison, the distributed learning tuned model converged on the average game score much faster than the default version. Its training progress also appeared smoother, and it spent a shorter period of time in the -200 score range. However, there was still some level of volatility in its score as it trained.

On the other hand, the machine learning tuned model had a much smoother training curve and it was able to hit the cutoff score without any dips in average game score. It took the fewest number of episodes to finish training and had minimal oscillations once it reached the positive score range.

We tried to find a final tuned model that combined the hyperparameter tuning processes from both sets of hyperparameters, but our final model did not perform as well as our two separately tuned models. Nonetheless, this process illustrates that we can significantly improve the training efficiency with hyperparameter tuning of the distributed learning componenets such that it can compensate for suboptimal machine learning parameter values. While this observation may not hold in all scenarios, and may be especially relevant in the simpler reinforcement games that we worked with, it is still an interesting conclusion to reach. This speaks to the efficient resource usage of distributed learning and suggests that there may be some potential in focusing on distributed learning tuning moreso than machine learning tuning in time-constrained and resource-constrained environments.

Overall, the results demonstrate that the distributed DQN approach with Ray was effective in training an agent to solve the Taxi-v3 environment efficiently, leveraging the parallel computing power of multiple agents and a centralized global model.

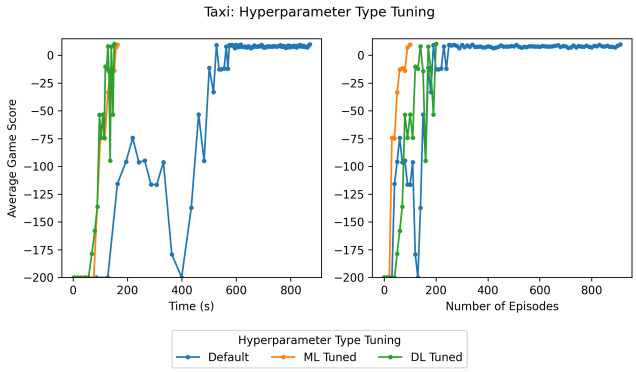

Figure 1: Comparison of training efficiency for different taxi models.

2.2 Cart Pole

The CartPole environment is a part of the Classic Control environments, where the objective is to balance an inverted pendulum by applying forces to move a cart along a frictionless track. The player must prevent the pole, which can deviate up to approximately ±24 degrees (±0.418 radians), from falling over and keep the cart, with a position ranging from -4.8 to 4.8, within the track boundaries by appropriately applying one of two actions: pushing the cart left or right. If the cart is pushed left, the action returned would be 0. In contrast, if the cart is pushed right, the action returned would be 1. The observation space consists of the cart position, cart velocity (with no theoretical limits), pole angle, and pole angular velocity (also with no theoretical limits). The game requires precise control to maintain balance and maximize the episode length, which is capped at 500 steps.

2.2.1 Setup

CartPole has similar adaptations as Taxi-V3 had. The environment was updated in the cartpole.ipynb code to instantiate the CartPole-v1 environment, with a state size of 4 and an action size of 2. Since CartPole was more similar to Lunar Lander, we did not have to preprocess the state in comparison to Taxi.

The play_game method for GlobalNet and train method for Agent had to be edited to ensure the state was passing in an array rather than a tuple. However, these edits may be reverted due to an older version of gym being used on some of the VMs. Once we adjusted the assignment 3 code to work with the CartPole environment, our next steps included adjusting

hyper parameters to improve the learning rate and quality.

2.2.2 Parameter Configurations

For CartPole, we focused on fine tuning the hyper parameters first and then analyzing resource efficiency on training quality and time.

The default parameters as shown in table 1 provided a control training time of 384.22 seconds. After the analysis of different hyper parameter configurations, we decided that the most impactful parameters were num_epochs_actor_train, batch_size, learning_rate, and update_every. With the parameter adjustments shown in table 3, we were able to achieve the quickest training time of 135.52 seconds.

Parameter	Value
Number of epochs for agent training	2
Batch size (batch_size)	128
Learning rate (learning_rate)	0.0001
Update network every N steps (update_every)	2

Table 3: Updated Training Parameters

After fine-tuning the hyper parameters, we focused on decreasing the number of VMs and agents in order to minimize resource usage. As shown in figure 2, we achieved a training time of 100.59 seconds by using 2 VMs, 1 agent, and the hyper parameter combination from table 3.

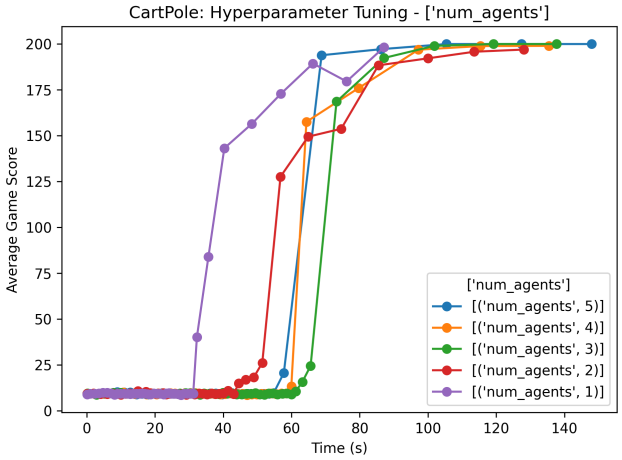

Figure 2: Hyper parameter Tuning: num_agents

2.2.3 Results

The distributed DQN agent successfully solved the CartPole-v2 environment from OpenAI Gym under various parameter configurations and resource allocations. Table 4 summarizes the key results obtained during the experimentation process.

Table 4: Results Summary

Configuration	Results
Default	
Average Score	195.80
Environment solved in	110 episodes
Total duration	384.22 seconds
Average time for training	3.40 seconds
Average time for updating	0.02 seconds
Average time for synchronizing	0.02 seconds
Fine Tuned Hyperparameters	
Average Score	199.0
Environment solved in	440 episodes
Total duration	135.51 seconds
Average time for training	0.26 seconds
Average time for updating	0.02 seconds
Average time for synchronizing	0.02 seconds
Fine Tuned Hyperparameters and VMs	
Average Score	198.60
Environment solved in	410 episodes
Total duration	100.59 seconds
Average time for training	0.21 seconds
Average time for updating	0.01 seconds
Average time for synchronizing	0.01 seconds

Under the default parameter settings (epochs = 10, batch size = 64, learning rate = 5e-4, update every = 4, agents = 4), the agent solved the environment in 110 episodes, with a total duration of 384.22 seconds. The average time for training, updating, and synchronizing was 3.40 seconds, 0.02 seconds, and 0.02 seconds, respectively.

By fine-tuning the hyperparameters (epochs = 2, batch size = 128, learning rate = 0.0001, update every = 2, agents = 4), we observed a significant improvement in training efficiency. The environment was solved in 440 episodes, with a total duration of 135.51 seconds, representing a 2.83x speed increase compared to the default settings. The average time for training decreased to 0.26 seconds, while the updating and synchronizing times remained at 0.02 seconds each.

Further optimization was achieved by reducing the number of VMs and agents in addition to the fine-tuned hyperparameters (epochs = 2, batch size = 128, learning rate = 0.0001, update every = 2, agents = 1, VMs = 2). This configuration yielded the best performance, solving the environment in 410 episodes with a total duration of 100.59 seconds, a 3.81x speed increase compared to the default settings. The average time for training, updating, and synchronizing decreased to 0.21 seconds, 0.01 seconds, and 0.01 seconds, respectively.

The results demonstrate that fine-tuning the hyperparameters and optimizing resource allocation can significantly reduce training time and improve the efficiency of the distributed DQN agent in solving the CartPole environment. The simplicity of the CartPole environment allowed for efficient

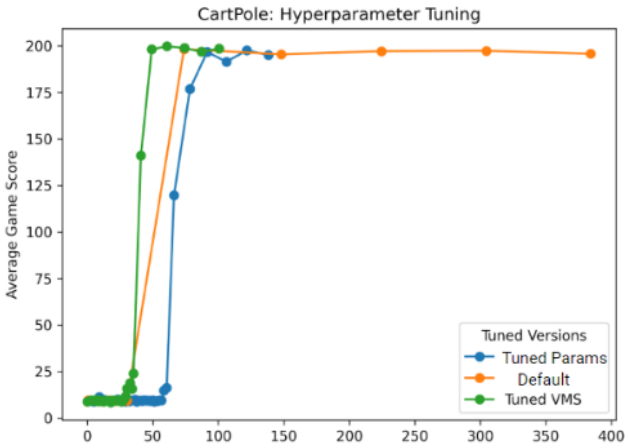

Figure 3: CartPole Visualization

training with minimal resources, as evidenced by the optimal performance achieved using only 2 VMs and 1 agent.

The reduced training time can be attributed to several factors:

1. Decreasing the number of epochs for agent training from 10 to 2 reduced the number of training iterations per agent, leading to faster convergence.
2. Increasing the batch size from 64 to 128 allowed for more efficient gradient updates, as larger batches provide a better estimate of the true gradients.
3. Lowering the learning rate from 5e-4 to 0.0001 helped stabilize the learning process by making smaller updates to the network weights, preventing overshooting the optimal solution.
4. Reducing the update frequency (update_every) from 4 to 2 decreased the number of network updates per episode, striking a balance between computational efficiency and maintaining sufficient update frequency for effective learning.
5. Minimizing the number of agents and VMs reduced communication overhead and resource contention, allowing for faster training and synchronization.

These factors enabled the distributed DQN agent to learn more efficiently, resulting in a significant reduction in training time while maintaining high performance with Cartpole.

3 Conclusion

We utilized distributed training with Ray to parallelize computation for Taxi and CartPole. We identified a cost-effective and efficient hyperparameter configuration for CartPole, and improved training efficiency through distributed learning and machine learning tuning for Taxi. Overall, we successfully optimized solutions for both Taxi and CartPole using DQN and Ray, with a focus on resource usage efficiency.

3.1 Taxi Conclusion

We conducted a thorough evaluation of both tuned machine learning parameters (learning rate, gamma, tau) and tuned distributed learning parameters (epochs, agents, and batch size). We observed a significant improvement in training time and smoothness after tuning either the distributed learning or the machine learning hyperparameters individually. However, we did not find a combination of both sets of parameters to be effective in reducing runtime or improving convergence. While similar performance improvements can be achieved by tuning either distributed learning parameters or machine learning model parameters separately, the same cannot be said for tuning both concurrently. This is an important takeaway from our project: in certain scenarios, tuning the distributed learning hyperparameters can compensate for suboptimal machine learning hyperparameters used in the DQN. Therefore, under resource-limited scenarios, it can be beneficial to prioritize tuning the distributed learning hyperparameters to improve training efficiency.

3.2 CartPole Conclusion

We achieved optimal results in CartPole using only 2 virtual machines and 1 agent. By fine-tuning the hyperparameters alongside the virtual machines, we significantly improved our training runtime, even with an increased number of episodes. Compared to the default parameters, our hyperparameter configuration made training 3.81 times faster. CartPole's simplicity allowed us to train it efficiently without requiring extensive resources, thereby enhancing cost-effectiveness and reducing resource usage.

4 Metadata

The presentation of the project can be found at:

`https://drive.google.com/file/d/1EAmzH5xfCH_LIzY8_3mFU9kNPALB0iNX/view?usp=sharing`

The code/data of the project can be found at:

`https://github.com/lizatrundle/DS5110-Final-Project`

5 Sources

We used help from Open AI Chat GPT in configuring game code, as well as information from Gym library Docs:
`https://www.gymlibrary.dev/`