Comparison of Deep -Q Network for Reinforcement Learning using Multi-stage Frameworks

Yongju Seong
Arizona State University
Tempe, Arizona
1225467737
yseong3@asu.edu

Tzu Ning Chueh
Arizona State University
Tempe, Arizona
1225456934
tchueh@asu.edu

Aniket Anil Bhosale
Arizona State University
Tempe, Arizona
1225396211
abhosal4@asu.edu

PROBLEM STATEMENT

The problem we investigate is the impact of computational infrastructure, particularly the utilization of deep learning frameworks with different execution modes (eager and lazy) and distributed training capabilities, on the performance of Deep Q Networks (DQN) within multi-stage reinforcement learning frameworks. While DQN has shown considerable success in various reinforcement learning applications, understanding how different deep learning platforms, such as TensorFlow, Apache MXNet, and Google Jax, along with their execution modes and distributed training systems, influence the efficiency and effectiveness of DQN remains an open question. This research is significant as it aims to provide comprehensive insights into the relative strengths, weaknesses, and trade-offs of these frameworks, aiding in the selection of the most suitable platform for reinforcement learning applications. Through systematic experimentation and analysis, we seek to contribute to a deeper understanding of the role played by computational infrastructure in shaping the performance of DQN within multi-stage frameworks.

READING AND BACKGROUND

This research leverages the significant developments in reinforcement learning, particularly the Deep Q-Network (DQN) model, which has demonstrated notable successes across various applications. The study focuses on comparing the performance implications of using different computational frameworks—TensorFlow, Apache MXNet, and Google Jax—on DQN implementation. The literature supports the relevance of these frameworks in enhancing computational efficiency and scalability in machine learning tasks. References such as Caldas et al. (2018) emphasize the importance of computational efficiency in expanding the reach of federated learning, which aligns with the objectives of this study to optimize resource utilization and training efficiency across different frameworks.

Sebastian Caldas et al.'s work provides a foundational understanding of reducing resource requirements in machine learning, which is crucial for evaluating the frameworks chosen for this study. Additionally, framework-specific resources such as "Google JAX" documentation and TensorFlow's reinforcement learning capabilities via TensorFlow Agents (Hafner et al., 2017) provide essential technical insights that help in understanding the operational and architectural distinctions among the frameworks. This comparative analysis aims to identify which framework delivers optimal performance in terms of speed, efficiency, and scalability when running DQN models, guiding future applications in more complex scenarios such as Atari games or real-world tasks requiring robust decision-making models.

METHOD

We investigate the most popular frameworks in machine learning and compare them across different tasks using different execution modes and varied computation resources. Our aim is to identify which of these networks is the most scalable/efficient and find out what tasks each framework is best suited for.

EXPERIMENTAL ENVIRONMENTS

The experimental setup involves comparing the performance of three deep learning frameworks, TensorFlow, Apache MXNet, and Google Jax, in training Deep Q Networks (DQN) within the CartPole environment. The experiments are conducted on ASU's supercomputing infrastructure, Sol, which comprises over 15,000 CPU cores, including 240 A100 and 15 A30 GPUs, as well as high-memory nodes with 2TB of RAM each. The comparison is carried out across three architectures: single CPU core allocation, single GPU allocation (A100), and dual GPU allocation (2 * A100), all performed on a single node. The evaluation metrics include training time, episode length, resource utilization, and scalability. The evaluation is conducted for both lazy and eager execution modes on CPU, GPU, and Horovod (for distributed training with 2 GPUs). For each framework and execution mode, the final score achieved, resource utilization, and training times are compared across different tasks and execution styles to assess their performance comprehensively.

DATA

Q-Learning algorithm is a reinforcement learning algorithm that doesn’t rely on external data. We used the OpenAI gym environ-

ment to set up a training system. OpenAI Gym is a toolkit for reinforcement learning (RL) that provides a range of environments for training RL agents. Users select an environment, like a game or simulation, and their agent interacts with it, learning to make decisions that maximize rewards. The process involves running episodes, balancing exploration and exploitation, and integrating various RL algorithms. Gym enables fair comparisons between algorithms and allows for customization of environments. Overall, it's a foundational tool for RL research and development, offering standardized interfaces and evaluation protocols.

We use this framework with the Cartpole-V1 environment. Our goal is to hit 1000 consecutive steps in this environment. A model that can achieve this score has essentially learned the mechanics of the game and can go on for longer if desired.

The cartpole environment gives out information about the heading and angular velocity of the pole and the coordinates of the cart. This is used as the data per step and the model learns using subsequent steps.

RESULTS

In our evaluation, we present the results of our evaluation across six tasks/execution styles for each framework:

1. Lazy execution on CPU

2. Lazy execution on GPU

3. Lazy execution on Horovod

4. Eager execution on CPU

5. Eager execution on GPU

6. Eager execution on Horovod

We initially assessed the final score attained by each framework across these tasks to determine the effectiveness of the learned policy. Subsequently, we examined resource utilization to evaluate the performance of each framework within a specific task, considering CPU, GPU, and Horovod execution. Finally, we analyzed the training times associated with each task to gauge the performance of each framework in terms of training speed and efficiency. Here are the results per each framework below.

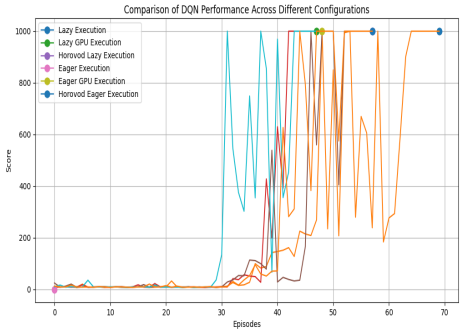

Figure 1: Final score attained by Jax

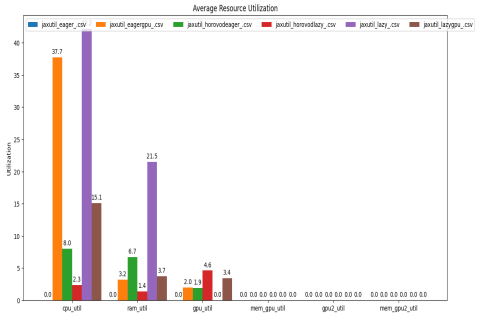

Figure 2: Average resource utilization by Jax

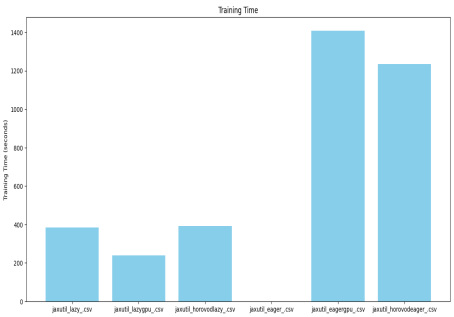

Figure 3: Training times by Jax

As we can see in Figure 1, each task with Jax got the goal score (1000) except for one task (eager execution on CPU). When using JAX with eager execution on CPU, it runs too slow and fails to even give an output.

Also, The CPU showed the highest utilization during lazy execution on CPU, followed closely by eager execution on GPU. The lowest utilization was observed during lazy execution with Horovod. RAM utilization was highest during lazy execution on CPU and lowest during lazy execution with Horovod. GPU utilization was highest during lazy execution with Horovod and lowest during eager execution with Horovod. Additionally, GPU memory utilization and the utilization of the second GPU were both 0%.

Lastly, the highest training time was observed for eager execution with GPU, while the lowest was observed for lazy execution with GPU.

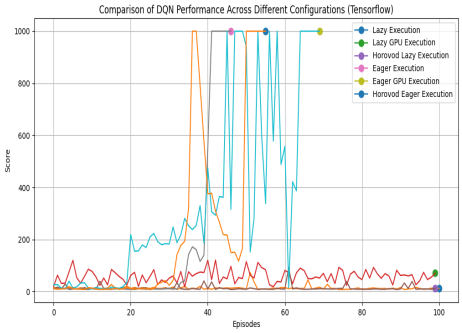

Figure 4: Final score attained by TensorFlow

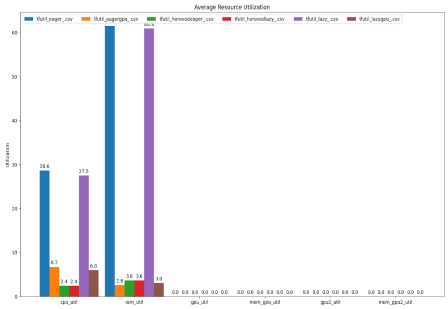

Figure 5: Average resource utilization by TensorFlow

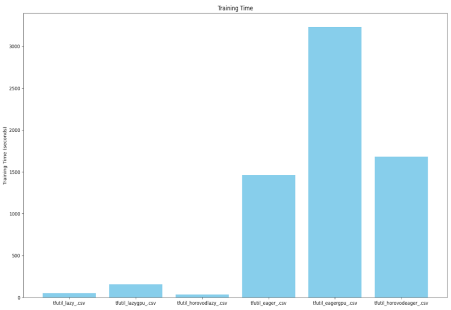

Figure 6: Training times by TensorFlow

Similarly, as we can see in Figure 4, models trained using lazy execution with TensorFlow did not achieve our goal score (1000), while those trained using eager execution did.

Moreover, the CPU utilization performance of the eager execution with CPU is the highest, while the Horovod with eager and GPU execution shows the lowest performance. The RAM utilization performance of the eager execution with CPU is the highest, while the eager execution with GPU shows the lowest performance.

Lastly, the eager execution with GPU shows the longest training time, and Horovod with lazy execution shows the shortest training time as we can see in Figure 6.

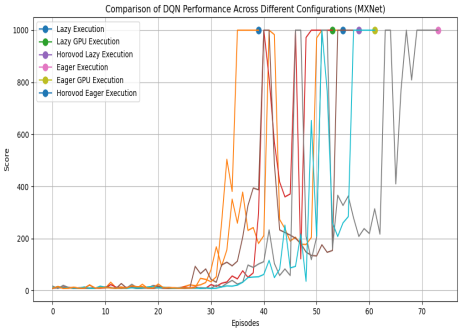

Figure 7: Final score attained by MXNet

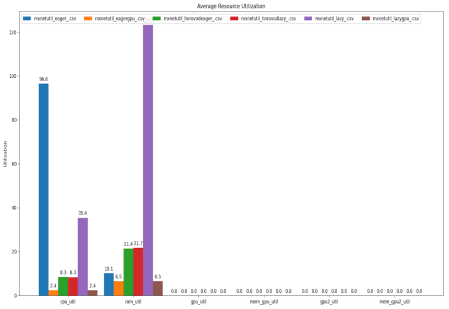

Figure 8: Average resource utilization by MXNet

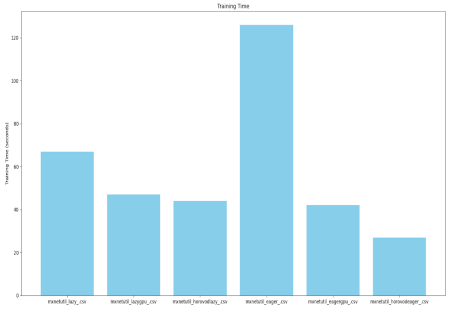

Figure 9: Training times by MXNet

For MXNet, Lazy execution consistently achieves the highest performance, with all tasks reaching a final score of 1000.

Also, the CPU utilization performance is highest for eager execution with CPU, while both lazy and eager execution with GPU shows the lowest performance in terms of CPU utilization. Similarly, the RAM utilization performance is highest for lazy execution on the CPU, while both lazy and eager execution with GPU shows the lowest RAM utilization.

Lastly, the eager execution on CPU had the highest training time, while eager execution with Horovod had the lowest as we can see in Figure 9.

In sum, the result reveals that achieving optimal performance requires tasks that fully utilize GPU resources, indicating a need for more compute-intensive tasks. Across all frameworks, lazy execution consistently outperforms eager execution in terms of training time. Moreover, when utilizing GPU resources, there is a notable improvement in performance regarding RAM and memory utilization, regardless of the chosen execution method for the task. This suggests that leveraging GPU resources enhances efficiency and resource management, irrespective of the execution paradigm adopted.

Also, here are the results to compare for all tasks and frameworks below.

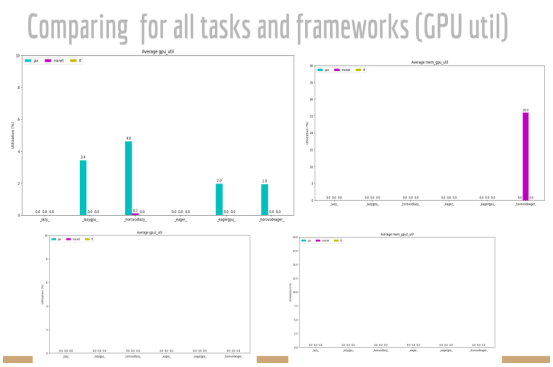

Figure 10: GPU utilization for all tasks and frameworks

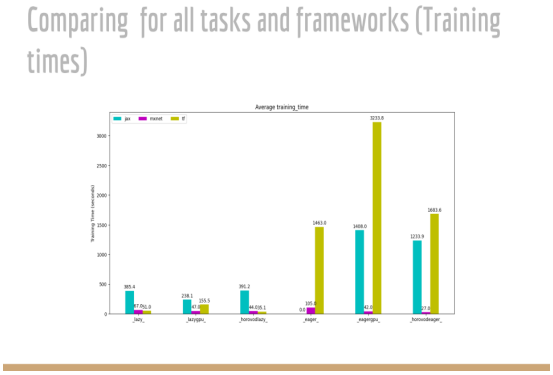

Figure 11: Training times for all tasks and frameworks

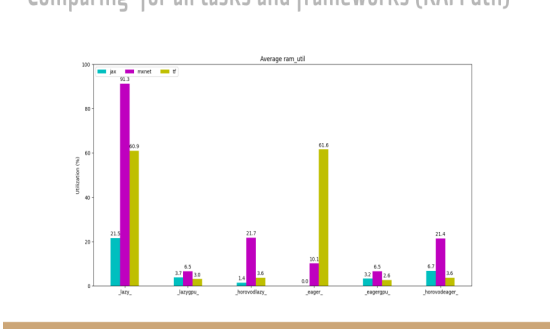

Figure 12: RAM utilization for all tasks and frameworks

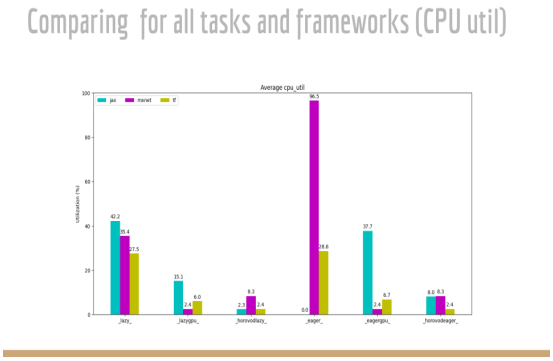

Figure 13: CPU utilization for all tasks and frameworks

In conclusion, when comparing TensorFlow, JAX, and MXNet for deep learning tasks, each framework presents its own trade-offs in terms of resource utilization and training efficiency.

TensorFlow, while known for its widespread adoption and user-friendly interface, tends to utilize resources less efficiently compared to other frameworks. This inefficiency can result in longer training times. However, TensorFlow boasts extensive documentation and strong developer support, making it an attractive option for those prioritizing ease of use and accessibility.

On the other hand, JAX demonstrates lower RAM utilization, which is beneficial in scenarios where memory is limited. However, this advantage comes at the cost of longer training times compared to TensorFlow. Therefore, JAX may be recommended in situations where memory constraints are a primary concern and where users are willing to accept slightly longer training times for the sake of memory efficiency.

MXNet stands out for its efficient utilization of resources (CPU), resulting in shorter training times compared to TensorFlow and JAX. This makes MXNet a compelling choice for users who prioritize training speed and computational efficiency. While MXNet may not offer the same level of user-friendliness as TensorFlow, its high efficiency makes it particularly appealing for large-scale deep learning tasks where computational resources are a significant factor.

In summary, TensorFlow excels in ease of use and developer support but may require more resources and time for training. JAX is recommended for memory-constrained environments despite longer training times. Meanwhile, MXNet shines in terms of resource efficiency and faster training, making it ideal for computationally intensive tasks. Ultimately, the choice between these frameworks depends on the specific requirements and priorities of the user or project at hand.

CONCLUSION

In conclusion, the study presents a detailed comparison of the performance of three major computational frameworks in executing DQN on the CartPole task. The findings suggest that MXNet generally offers the best performance in terms of training time and resource utilization, making it the most efficient framework among those tested. TensorFlow, while not as efficient, is noted for its ease of use and extensive documentation which might be beneficial for developers new to reinforcement learning. JAX, although it showed higher training times, could be preferable in scenarios with limited memory availability.

FUTURE WORKS

For future work, the team suggests exploring two promising directions: Firstly, expanding the evaluation of framework performance through the implementation of more complex tasks, such as Atari games. This extension would allow for a comparative analysis of the learning speed and sample efficiency of DQN models across different computational frameworks under more challenging conditions, thus providing a clearer understanding of each framework's robustness and efficiency. Secondly, investigating ensemble learning techniques to boost DQN performance offers another

valuable avenue. Training multiple DQN models with variations in their architecture or training datasets and then combining their outputs could potentially enhance model robustness and generalization capabilities, thereby fostering more reliable and effective reinforcement learning applications.

REFERENCES

[1] Jia Zou, 2024. CSE 598: *Data Intensive System for Machine Learning*. Lecture Slides

[2] Caldas, Sebastian, et al, 2018. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210.* DOI: https://doi.org/10.48550/arXiv.1812.07210

[3] Hafner, Danijar, James Davidson, and Vincent Vanhoucke, 2017. Tensorflow agents: Efficient batched reinforcement learning in tensorflow. *arXiv preprint arXiv:1709.02878*. DOI: https://doi.org/10.48550/arXiv.1709.02878

[4] Google JAX. 2024. Google Jax Documentation. Retrieved April 27, 2024 from:https://jax.readthedocs.io/

[5] TensorFlow. 2024. TensorFlow Documentation. Retrieved April 27, 2024 from https://www.tensorflow.org/

[6] Apache MXNet. 2024. Apache MXNet Documentation. Retrieved April 27, 2024 from https://mxnet.apache.org/

[7] OpenAI Gym. 2024. OpenAI Gym Documentation. Retrieved April 27, 2024 from https://www.gymlibrary.dev/index.html