FEDERATED MODEL AVERAGING FOR DEEP Q NETWORKS(F-DQN)

A PREPRINT

Nishant Kumar
18095048, B-Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
`nishantkr.ece18@itbhu.ac.in`

Sirusala Niranth Sai
18095074, B-Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
`sirusalansai.ece18@itbhu.ac.in`

Piyush Sharan
18095087, B-Tech
Department of Electronics Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
`piyushsharan.ece18@itbhu.ac.in`

Rohan Agarwal
18135080, B-Tech
Department of Mechanical Engineering
Indian Institute of Technology (BHU)
Varanasi-221005
`rohanagarwal.mec18@itbhu.ac.in`

December 8, 2020

ABSTRACT

The advent of reinforcement learning greatly pushed the boundaries of computation, ranging from superhuman performance in video games to previously unseen feet in robotics. One of the key limitations that prevent the deployment of Deep RL based solutions in real-world applications is the need for a reliable data generating environment. The convergence of the algorithms rely on rigorous data collection and also prove to be sample inefficient. However, it is impossible to collect such magnitudes of data from a real system and the usage of simulation leads to domain variations and inaccurate modelling. Federated Learning provides an alternative solution, to deploy distributed learning in a real-world system, ensuring the privacy of the data and also sharing the computation load across multiple workers. This solves the problem of using a single system to collect impractical amounts of data and enables us to obtain samples directly from the deployment domain. In this paper, we propose a novel formulation titled Federated Deep Q Networks (F-DQN) to perform distributed learning for Deep RL algorithms.

Keywords Deep Reinforcement Learning · Federated Learning

[1]

1 Introduction

In most real-world applications, datasets from clients are often privacy sensitive and it is often difficult for such a data centre to guarantee building models of high quality. To deal with the issue, a new learning setting, namely federated learning originated. Here, the model training is firstly done, locally, on independent agents, subsequently, the parameters

[1]The authors thank Lokesh Krishna for reviewing our work and providing t

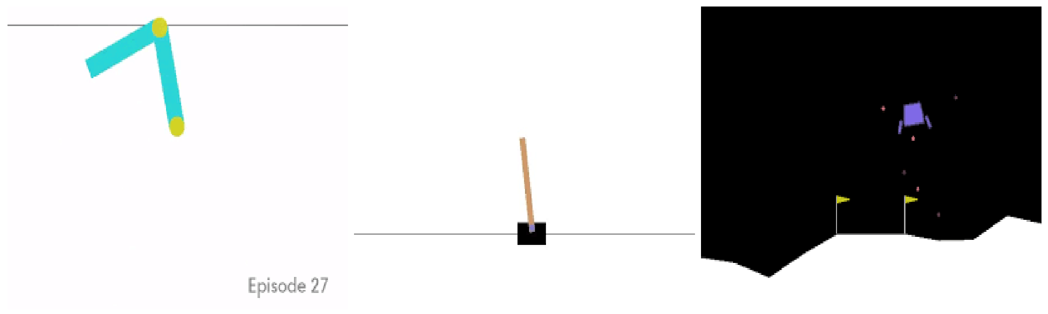

Figure 1: Acrobot, CartPole, and LunarLander environments(left to right)

are sent to a global model where they are then averaged.[1] [7], being a type of distributed learning has also proved to allocate adequate computation load, given the capabilities of the local worker. This enables the accumulation of a diverse set of data, parallelly trained in a varied range of local computation nodes.

The Agent-Environment modelling in RL problems and the MDP formulation, greatly rely on accurate domain specific data samples. The ability to generate these data samples play a crucial role in the performance of such algorithms. This is one of the major reasons for RL algorithms surpassing superhuman feats in video games but hardly translating to any real world applications. The dependency on a controlled environment for data collection, heavily hinders the scalability of such RL based solutions. The transition of RL from tabular settings to function approximation setting a.k.a Deep RL, solved the canonical problem of "curse of dimensionality", thus scaling up RL algorithms to operate in continuous state and action spaces.In deep reinforcement learning, building policies of high quality is challenging when the feature space of states is small and the training data is limited. For Instance, [4], requires samples of the order of million steps. Further more, policy gradients [6], [5], and actor critic methods [3] which have superior performance metrics greatly suffer from data inefficiency.

Thus, the contribution of this work is twofold,
a) A new framework for combining the methods of federated model averaging with Deep Q Learning thus leveraging the practicality of federated learning methods in RL based problems.
b) A novel formulation for reward-weighted parameter aggregation, to update a central policy from the contributions of synchronous worker nodes, and provide comparable results to that of a single agent setting.

2 Approach

We use three DQNs as a private network and one DQN as a public network, to conduct tests on the environment. All the agents used were initialized with the same initial parameters with a constant seed. To involve federated learning, we create an aggregated model every 20 steps, by averaging the network's parameter values. There can be two methods to average values: Unweighted avg takes the avg without any weightage. So if an agent performs better than the other, its contribution is not affected in the avg parameter.

$$\mathrm{W}_{avg} = (W_1 + W_2 + W_3)/3$$

Weighted avg: Here, the models are weighted based on their performance in their test runs. If an agent performs better than its counterpart, it is given more importance in the aggregate model.

$$\mathrm{W}_{avg} = (R_1 * W_1 + R_2 * W_2 + R_3 * W_3)/(R_1 + R_2 + R_3)$$

Where R1, R2, R3, are the average rewards obtained by agent 1, agent 2, agent 3 respectively. After averaging, the aggregated private network parameters are set to the global model and its performance is tested. Finally, the global model parameters are sent back to the private networks and they are updated. The private networks are trained again with the new parameters received from the global model.This cycle of training and updating parameters continues till convergence.

3 Implementation

The codebase was made using PyTorch library and the training and testing part was done with the help of Google Colab. Environments from OpenAI gym are uses for training and testing[2]. For our experiments, we used 3 independent

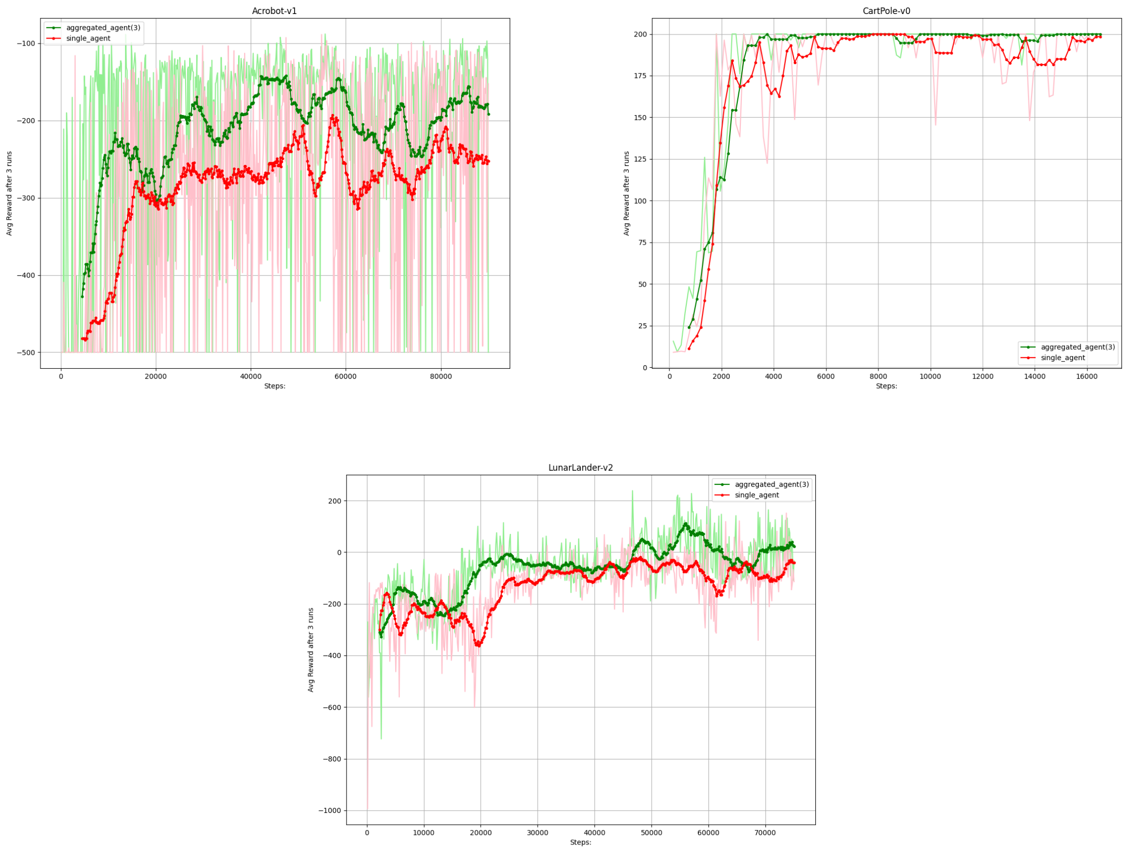

Figure 2: Preformance of single and aggregated agents in Acrobot, CartPole, and LunarLander environments

agents for training, and one global model for creating and testing the aggregated network. For every 20 steps, we update the global model with the average of the three models.

We tried to make the train() function of the agent callable more than one time. So all the initialization parts were moved to the init function of the class. Since we do not have the appropriate resources for proper parallelization of training, we demonstrate our approach by training the networks serially for 20 steps each, before creating a global average. So to train a agent serially, we needed train() to be callable repeatedly.

The current implementation uses independent Replay Buffers for all agents. For testing the agent, we use run the agent for 5 runs and take an average of the rewards, to make sure than an unbiased estimate of the reward can be made.

For testing, we compare the performance of the global agent with a single agent. The single agent is trained independently, and holds no connection to the aggregated agent or the worker nodes.

4 Results and Conclusion

A general trend observed in all the environments was that the aggregated agent is more sample efficient and shows less variance as compared to a single agent. The global agent is able to maintain its reward without much variation, but whereas the single agent has a lot of variation in its performance. A possible reason for this might be the aggregation which is giving out global agent an aggregated experience of all the local agents, so it possibly has the information of some state spaces which are yet unknown to the single agent.

4.1 Acrobot:

An excellent demonstration of the above discussion would be the results we see in the Acrobot-v0 environment (Fig 2) The global aggregated agent is highly sample efficient and stable whereas the single agent is struggling to converge,

4.2 Cart Pole:

In the CartPole-v0 environment we could see that the aggregated agent is performing consistently better than the single-agent, for most time instances(Fig 2). As the aggregated agent holds the combined experience of all the single agents, it is able to perform well even in the state spaces not explored by the single agent. We could also see that the aggregated agent get the maximum reward consistently, whereas the single agent still has a lot of variance.

4.3 LunarLander:

In the case of Lunar Lander, we see that in most cases, the aggregated agent performs consistently better than the single agent(Fig 2). However, in some places, we do see that the single agent's performance comes very close to that of the aggregated agent. We suspect that this happens because of two reasons:

1. Lunar Lander has greater complexity than simpler environments like Cartpole and Acrobot.
2. Due to a higher state dimensionality of the environment, we can safely assume that collecting the data exclusively from only 3 agents is not sufficient for a better exploration of the environment.

We have kept all the files related to this project in the below repository

5 To run the code

• Download the project repo from the link: `https://github.com/nishantkr18/federated-model-averaging-for-DQN`
• Install the requirements mentioned in the `https://github.com/nishantkr18/federated-model-averaging-for-DQN#requirements`
• To run the training script, refer `https://github.com/nishantkr18/federated-model-averaging-for-DQN#running-the-code`
• Plots are stored in the 'plots/' folder of the repository.

References

[1] Igor Adamski et al. *Distributed Deep Reinforcement Learning: Learn how to play Atari games in 21 minutes*. 2018. arXiv: 1801.02852 [cs.AI].

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[3] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 1861–1870. URL: http://proceedings.mlr.press/v80/haarnoja18b.html.

[4] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: (2013). cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013. URL: http://arxiv.org/abs/1312.5602.

[5] John Schulman et al. "Proximal Policy Optimization Algorithms." In: *CoRR* abs/1707.06347 (2017). URL: http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17.

[6] John Schulman et al. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1889–1897. URL: http://proceedings.mlr.press/v37/schulman15.html.

[7] Hankz Hankui Zhuo et al. *Federated Deep Reinforcement Learning*. 2020. arXiv: 1901.08277 [cs.LG].