Distributed Q-Learning: A Democratic Learning Process

Max Robinson max.robinson@jhu.edu
Johns Hopkins University,
Baltimore, MD 21218 USA

Abstract

1. Introduction

Reinforcement learning can be described simply as the learning process of an agent in an environment trying to reach a goal. The agent learns by attempting to maximize a reward. In some situations the learner has very little prior knowledge of the environment. The act of maximizing the reward is the learning process.

Reinforcement learning typically requires an agent to follow a $state \rightarrow action \rightarrow state'$ loop. As a result, the learning process is also typically sequential. For large state spaces or complex environments, this process can be slow. A single agent must often experience many iterations of maximizing a reward in order to learn in even a simple environment.

A famous example of Reinforcement Learning is Tesauro's TD-Gammon agent. The best performing agent required 1,500,000 training games to beat one of the best Backgammon players at the time (Tesauro, 1995). As a more modern example, Mnih et al. (2013) developed an algorithm to play Atari 2600 video games called DQN. To learn to play each game at a human level or higher, 50 million frames were required to train an agent for each game. Total training on all 49 games in the study took about 38 days of game experience (Mnih et al. 2015).

The constraint of a lone agent acting sequentially can create situations where training an agent to learn a task can take an exorbitant amount of time. To combat this, researchers have focused on ways to adapt these reinforcement learning algorithm to run in parallel to decrease the amount of time it takes a single agent to learn.

Recent research has studied speeding up Deep Neural Networks for reinforcement learning such as DQN (Mnih et al., 2013) and others. Quite a few papers have suggested ways of parallelizing both the computation for these methods as well as distributing actors and learners to run in parallel, which send back gradients to be used to update a centralized parameter store.

This proposal suggests a step back to explore a slightly more simplistic model of distributed reinforcement learning using Q-learning. It suggests a parallelized and distributed version of the Q-learning algorithm using a traditional Q-memory, hereon in referred to as Distributed Q-learning (DistQL). DistQL, similar to other distributed reinforcement learning approaches, uses multiple separate agents and environments with local copies of parameters and memory to learn, and has a centralized main Q-memory. Different from other algorithms, however, DistQL sends learned Q-values to the centralize memory which are then combined with the existing values. The centralized Q-memory then updates values, using a linear combination of Q-values and hyperparameters explained in Section **??**.

An advantage to this approach is that the traditional Q-learning algorithm is being leveraged. This means that the optimality convergence guarantees of Q-learning, even in an asynchronous settings, are likely to still hold true.

2. Previous Work

Q-Learning (Watkins, 1989) has been a foundational algorithm in reinforcement learning, especially after it was shown to have optimal convergence in the limit (Watkins & Dayan, 1992). Very soon after this, asynchronous methods of Q-learning were being explored.

Tsitsiklis (1994) studied if Q-learning would still keep its optimal convergence property in an asynchronous setting. The results showed that the Q-learning algorithm does continue to keep this convergence guarantee in an asynchronous setting, given that old information is eventually discarded through the update process, along with a few other conditions.

More recently there has been a flurry of work done in parallelizing reinforcement learning algorithms. Mannion, Duggan, & Howley (2015) used a distributed Q-learning algorithm with multiple agents and environments to learn in, applied to traffic signal control. In their algorithm, at every time step the agents update a global Q Matrix in accordance with the Q-learning update equation.

In addition to using a tabular implementation of Q-learning, function approximation versions using deep neural networks, such as DQN (Mnih et al., 2013), have also explored asynchronous and distributed architectures for learning.

The General Reinforcement Learning Architecture (Gorila) (Nair et al. 2015) uses a general, massively distributed architecture of agent, and environments to learn. Each agent in Gorila has its own environment, a copy of the model which is periodically updated, a separate replay memory and a learner that aims to learn using a DQN (Mnih et al., 2015). The learner samples data from the memory replay and computes the gradients for the parameters, which are then sent to a centralized parameter server. The parameter server then updates a central copy of the model based on gradients.

A3C (Mnih et al. 2016) collapses Gorila onto a single machine, using multiple processors to parallelize the agents actions in an environment. A3C similarly computes gradients for each agent's model at each time step, locally, but accumulates these gradients over a series of time steps before updating the central model, every so often. This aims to balance computational efficiency with data efficiency.

3. Q-Learning

4. Distributed Q-learning Algorithm

5. Experiments

To demonstrate the effects of DistQL, the Taxi world (Dietterich, 2000) and Cart Pole (Barto, Sutton, & Anderson, 1983) environments were chosen for agents to learn in. The environment implementations used are part of the OpenAI Gym and are publicly available. The goal in each environment is for the agent to achieve the highest reward possible in the environment.

The taxi world is a 5 by 5 grid world where the agent is a taxi driver. The goal of the agent is to navigate to the passenger, pick him or her up, navigate to the goal, and drop the passenger off at another destination. For rewards, the agent receives +20 points for successfully dropping off the passenger, -1 point for every time step, and -10 penalty for any illegal pick-up or drop-off actions made. The actions available to the agent are to move in the four cardinal and perform a pick-up or drop-off action. An episode finishes after the passenger has been successfully dropped off at their destination.

The Cart Pole environment consists of trying to balance a pole upright while attached by a joint to a cart on a frictionless track. The pole starts upright and the goal is to keep the pole balanced upright for as long as possible. The cart can move left or right along the track by applying a force of +1 or -1. The episode finishes after the pole leans 15 degrees left or right or the cart has moved 2.4 units from the center. A reward of +1 is given to the agent for every time step that the pole stays upright. For this experiment, the OpenAI implementation was modified so that when an episode is terminated a reward of -200 is incurred.

For Cart Pole, the implementation uses a continuous state space with 4 features. In order to apply Q-Learning, the continuous state space must be discretized. The first three features are discretized into 8 bins of equal size, with values between -2.4 to 2.4, -2 to 2 and -1 to 1 for features 0, 1, and 2 respectively. The fourth feature is discretized into 12 bins of equal size between the values of -3.5 to 3.5. This specific discretization is chosen because of the anecdotal and demonstrated success described by Victor Vilches (2017).

5.1 Evaluation

5.2 Results

6. Discussion

7. Conclusion

References

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*(5), 834–846.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym..

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, *13*(1), 227–303.

Mannion, P., Duggan, J., & Howley, E. (2015). Parallel reinforcement learning for traffic signal control. *Procedia Computer Science*, *52*, 956 – 961. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In

Balcan, M. F., & Weinberger, K. Q. (Eds.), *Proceedings of The 33rd International Conference on Machine Learning*, Vol. 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, New York, New York, USA. PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529 EP –.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., Maria, A. D., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., & Silver, D. (2015). Massively parallel methods for deep reinforcement learning. *CoRR*, *abs/1507.04296*.

Tesauro, G. (1995). Temporal difference learning and td-gammon. *Commun. ACM*, *38*(3), 58–68.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and q-learning. *Machine Learning*, *16*(3), 185–202.

Vilches, V. M. (2017). Basic reinforcement learning tutorial 4: Q-learning in openai gym..

Watkins, C. (1989). *Learning From Delayed Rewards*. Ph.D. thesis, Cambridge University, U.K.

Watkins, C., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*, 279–292.