Distributed Q-Memory: A Democratically Assembled Memory (Proposal)

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

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. The learner has no prior knowledge of the environment and does not know which actions to take when. The act of maximizing the reward is the learning process.

Since Reinforcement learning requires a $state \rightarrow action \rightarrow state'$ loop, the learning process for that agent is inherently sequential. As a result in large state spaces or complex games, training can be slow. A single agent must often experience many iterations of maximizing a reward in order to learn even a more 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. developed an algorithm to play Atari 2600 video games called DQN (Mnih et al. 2015). To learn to play each game at a human level or higher 50 million frames were required to train an agent for each game and total including all 49 games in the study about 38 days of game experience.

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.

To add to these works of research I propose a parallelized and distributed version a the Q-learning algorithm using a traditional Q-Value memory database, referred to from here on in as the Q-memory(Watkins, 1989). In this distributed form of Q-learning, there is a centralized Q-Memory that is updated by agents that are running in parallel. Each separate agents runs with their own copy of the environment and a Q-memory. Each agent then learns as usual according to the Q-learning algorithm (Watkins, 1989). However, every so often an agent will send updates to the centralized Q-Memory. The centralized Q-memory then is calculates updates to its values, using a linear combination of q-values and hyper parameters explained in Section 3.

I hypothesize that in using this distributed algorithm, there will be a close to linear increase in the learning rate, with respect to the number of parallel agents, of a learner using the centralized Q-memory when compared to a single Q-learning agent. This increased learning rate will reduce the total number of epochs per single agent to achieve similar or better performance, and thus reducing total wall training time.

2. Previous Work

3. Approach

References

Grounds, M., & Kudenko, D. (2008). Parallel reinforcement learning with linear function approximation. In Tuyls, K., Nowe, A., Guessoum, Z., & Kudenko, D. (Eds.), *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pp. 60–74, Berlin, Heidelberg. Springer Berlin Heidelberg.

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 –.

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

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.