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.

A lot of research recently has been around speeding up Deep NN for RL such as DQN 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.

I propose the we step back and explore a slightly more simplistic model of distributed RL using Q-learning with a Q-value database to explore the effects of a full combination of states and the multiple affects of distributing learners, such as state exploration rates, effects of learner contributions at different stages of learning, and different models for updating the distributed learners and their effect on exploration and performance.

To add to these works of research I suggest a parallelized and distributed version of the Q-learning algorithm using a traditional Q-Memory, Distributed Q-learning (DQL). 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 an increase in the learning rate of a learner using the centralized Q-memory when compared to a single Q-learning agent, as the number of parallel agents increases. This increased learning rate will reduce the total number of epochs per single agent to achieve similar or better performance. This will also likely reducing total wall time it takes to train.

2. Previous Work

3. Approach

The focus of this proposed experiment will be on testing the distributed Q-learning algorithm described in Section 3.1. The new algorithm and its variations will be tested according to the experimental approach described in 3.2

3.1 Distributed Q-learning Algorithm

The proposed algorithm to study is a novel distributed and parallelized version of Q-learning, referred to as DQL. DQL uses multiple agents environment pairs to learn rather than a single agent and environment. A central Q-memory Q_c server is also kept and is updated as the agents learns and completes epochs. In addition to the central Q-memory, there is also a central learning rate α_c which decays according to the number of updates Q_c has received.

More formally DQL has n agents $A_1, ..., A_n$. Each agent also has a copy of the environment in which it acts, $E_1, ..., E_n$. Each agent keeps track of a local Q-Memory Q_n, in addition to local parameters such as the discount factor, learning rate, and ϵ. Each agent then acts and learns inside side of its environment according to the ϵ-greedy Q-learning algorithm.

As the agent learns it will send updates to Q_c. In order to send updates, A list of $Q_i(s, a)$s that have been modified since the agent last sent updates is kept by the local agent, $Set_{q_i} = Set(Q'_i(s,a), ...)$. The agent then sends updates to Q_c every τ epochs. τ is a hyperparameter that can be adjusted to change how frequently each agent reports updates.

Updates are then performed by the Q_c server each time an update is retrieved. Q_c will be updated according to equation 1, where α_c and α_i are the learning rate for Q_c and A_i respectively. States in the update set

$$Q_c(s, a) = (1 - \frac{\alpha_c^2}{\alpha_i})Q_c(s, a) + \frac{\alpha_c^2}{\alpha_i}Q_i(s, a) \tag{1}$$

After updates to Q_c are complete, there are two variations that will be explored in this experiment for updating the local agent A_i. The first variation will send back a set of updated values from Q_c for just the set of states sent from A_i, Set_{q_i}. The agent will then replace any local values with those from Q_c This has the affect of reconciling just the states that local agent has updated recently with possible updates from other agents. The second variation overwrites Q_i with the entire Q_c memory. This provides information from other agents to a local agent and homogenizes the Q_is of the different agents. In this variation,

states that were explored by one agent would then be shared with other agents even if that agent has not explored the states itself.

Learning halts after Q_c has been updated a maximum number of times, or performance ceases to improve. To measure the performance of Q_c an agent and environment A_t, E_t are created with Q_c as the instantiated Q-memory. The agent then acts in the environment with no learning occurring. The performance metric being used is than captured upon completion and the test agent and environment are destroyed. How often the performance of Q_c is assessed is configurable and depends upon the fidelity with which the trainer wishes to assess the learning.

3.2 Experimental 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.