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

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. 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 Neural Networks 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

Q-Learning (Watkins, 1989) has been a foundational algorithm in reinforcement learning, especially after q-learning 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 it's 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. A 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, Kavukcuoglu, Silver, Graves, Antonoglou, Wierstra, & Riedmiller, 2013), have also explored asynchronous and distributed architectures for learning.

The General Reinforcement Learning Architecture (Gorila) by (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 update, 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 developed by (Minh 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. This aims to balance computational efficiency with data efficiency.

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, dubbed DQL-All, 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

DQL will be evaluated by running experiments on 3 different environments provided by the OpenAI Gym (Brockman, Cheung, Pettersson, Schneider, Schulman, Tang, & Zaremba, 2016). The 3 environments used will be Taxi-v2, LunarLander-v2, and BipedalWalker-v2. The Taxi-v2 environment is a discrete environment, while the LunarLander-v2, and BipedalWalker-v2 are continuous. For the continuous state space environments, the values for the states will be discretized on a per environment basis to ensure a single learner can learn in the specific environment. Each environment supplies a different level of problem complexity, and as such aims to demonstrate that any performance gains that could be observed using DQL are not subject to the complexity of the environment.

Experiments in each environment will be done using DQL with 2, 4, and 8 agents and done with each variant DQL and DQL-All . Each experiment will be run with 10 trials. The final results being averaged over all trials. Each agent will use an ϵ-greedy behavior policy with ϵ annealed from 1 to .1 over the learning process. The annealing may vary per environment.

The learning rate for Q_c will be annealed from 1 to .1 over the number of updates from agents. The learning rate per agent is annealed from 1 to .1 over the learning process, updating the learning rate after each epoch. The update frequency for each agent, τ, is anticipated to be set to 1 for all experiments. If additional time is available, τ will be varied to use the values 1, 5, and 10 to inspect the effect that update frequency has on learning.

3.2.1 Evaluation

The performance of DQL and DQL-All will be evaluated based on their learning rate to achieve better or equal performance to a single agent system using Q-learning. The learning rate will be measured in the number of epochs required to attain the performance.

The performance measures to evaluate an algorithm in their environment is determined by the provided OpenAI environment. Each environment has a score mechanic that is calculated differently for each one. A learners performance will be judged on the maximization of the environments scoring metric. For example if the goal is to travel a long distance and the score is based on how far the agent got, the performance for that agent will be judge on maximizing the score, i.e. distance.

In addition to traditional performance metrics, the number of newly explored states will be tracked for each agent. As the agent discovers a new state, the state will be tagged with which epoch the state was discovered in. This will then be used to compare the exploration patterns of the agents throughout the training process. Comparing which states are discovered by which agents at what time will give insight into if or when the agents started to converge to the same policies. In addition, it will provide data to evaluate if parallel exploration of the state space provides value to learning.

References

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

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 –.

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.

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.