Asynchronous Deep Reinforcement Learning for Atari Games

Nikhil Prakash
Computer Science Department
nikhilpr@buffalo.edu

Rahul Varma
Computer Science Department
rvarma@buffalo.edu

***Abstract*—Teaching computers to play Atari games is a challenging problem that has received increased attention since DeepMind's pioneering research in 2013 [1]. We have implemented an asynchronous version of one-step Deep-Q learning, an optimization to the original reinforcement learning algorithm that was proposed by DeepMind in 2016 [3]. The performance of the original algorithm is compared with the asynchronous implementation in this project.**

Keywords—Q-learning, Convolutional Neural Network, Deep Q learning

I. INTRODUCTION

Video games are often simplified version of a real world problem. As a result, algorithms which can take this visual data and make intelligent decisions are highly desirable as they can potentially be applied to solve more complex problems. This project considers Atari Breakout, a popular Atari 2600 game. Deep Reinforcement Learning algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks [3]: it uses more memory and computation per real interaction; and it requires off-policy learning algorithms that can update from data generated by an older policy.

One of the major challenges of Reinforcement Learning is the delay between actions and the corresponding rewards as opposed to the direct association between inputs and targets found in supervised learning. However, a convolutional neural network can be used to overcome this challenge and learn successful control policies from raw video in complex learning environments.

In Section II, we provide related work in this field. In Section III, we explain the formulation of our model and outline the algorithm that was used to parallelize the one step Deep Q Learning. In Section IV, we detail the architectural and experimental setup. In Section V, we provide the results in terms of Q-value and average reward.

II. RELATED WORK

The paper "Playing Atari with Deep Reinforcement Learning" by DeepMind" in 2013 [1], combines convolutional neural networks and deep reinforcement learning to develop an AI agent that plays Atari games. The Deep-Q learning algorithm used achieved better than human performance on 24 of the 47 games it was tested on. The same network was used and hyper parameters were kept fixed while training in every game. However, the training time is a few days even for simple games such as Pong and Breakout and the resource requirements are high. In 2016 DeepMind released a paper "Asynchronous Methods for Deep Reinforcement Learning" [3] that uses an asynchronous variant of Deep-Q learning to overcome these challenges. In this project we implement this asynchronous variant of Deep-Q learning.

III. METHODOLOGY

A. Markov Decision Process

We consider tasks in which the agent interacts with the environment through a sequence of observations, actions and rewards (MDP Process). For a given state, the agent tries to select an action from the control policy that maximizes future rewards. We use a convolutional neural network to approximate the optimal action-value function.

$$Q^*(s,a) = \max_{\pi} E[\, r_t + r_{t+1} + \gamma^2 r_{t+2} + \ldots + | s_t = s, a_t = a, \pi]$$

where r_t – maximum sum of rewards is discounted by γ at each time step t, according to a behavioral policy π = P(s|a), after making an observation s and taking an action a.

We use a model-free reinforcement learning method where the action value function is approximated using a convolutional neural network. Q-Learning tries to approximate the optimal action-value function $Q^*(s,a)$ as $Q(s,a;\theta)$. In one step Q-learning, the network tries to learn the parameters θ by minimizing the loss function using stochastic gradient descent. The loss function for i^{th} iteration is defined as

$$\mathrm{L_i}(\theta) = \mathrm{E}(\mathrm{r} + \gamma \max_{\mathrm{a}'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i))^2$$

where s' is the state after s.

B. Asynchronous Deep-Q Learning

Within the asynchronous variant of one step Q-learning, we aim to reduce the computational time and memory used by experience replay memory in one step Deep Q-Learning. This is achieved using two strategies.

Firstly, we use asynchronous actor-learner threads on a single machine. Each actor-learner thread performs one step of gradient descent. The network parameters and the target network parameters are shared between the threads.

Secondly, we use different exploration policy ϵ in individual threads. Learners running in parallel would be exploring different parts of the environment. By having different exploration policies, we reduce the correlation between successive updates. This replaces the functionality performed by replay memory. This also improves the overall robustness of the system and improves performance through better exploration.

Algorithm 1 Asynchronous one-step Q-Learning – pseudo code for each actor learner thread

// Assume global shared θ,θ^- and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- = \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
Receive new state s' and reward r

$$y = \begin{cases} r & for\ terminal\ s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & for\ nonterminal\ s' \end{cases}$$

Accumulate gradients wrt: θ: $d\theta \leftarrow d\theta + \frac{d(y - Q(s,a,\theta))^2}{d\theta}$
$s = s'$
$T \leftarrow T + 1$ and $t = t + 1$
if T mod $I_{target} == 0$ **then**
Update the target network $\theta^- \leftarrow \theta$
end if
if t mod $I_{AsyncUpdate} == 0$ or s is terminal **then**
Perform asynchronous update of θ using $d\theta$
Clear gradients $d\theta \leftarrow \theta$
end if
until $T > T_{mx}$

IV. EXPERIMENTS

The simulation for game environment is done using Open AI's gym library. The agent selects an action and acts on the environment. The environment responds with reward, observation and whether it's the end state. We use RMSProp as the optimization method and mini-batch gradient descent with batch size of 32. The experiment uses 8 actor-learner threads running on a single machine. The network uses a convolutional layer with 16 filters of size 8 x 8 with stride 4, followed by a convolutional layer with 32 filters of size 4 x 4 with stride 2, followed by a fully connected layer with 256 hidden units The convolutional neural network is build using TensorFlow. All three hidden layers were followed by a rectifier nonlinearity. The value-based methods had a single linear output unit for each action representing the action-value. We set three exploration rates $\epsilon_1, \epsilon_2, \epsilon_3$ and randomly sampled from these with probabilities 0.4, 0.3 and 0.3. The values were annealed from 1 to 0.1, 0.01 and 0.5 respectively.

V. RESULTS

Measuring the performance of supervised learning algorithms during training is fairly simple as it requires evaluating it on the testing and validation data sets. However for reinforcement learning performance evaluation can be challenging. One of the metrics we have used is the average reward, that is total reward divided by the number of games for an episode. However, the results we obtain using this are very noisy and give the impression that the trained model is not making any progress. Another measure is the action value function Q which provides a measure of the discounted reward that can be obtained by an agent. As seen in the graph the average Q increases steadily over time demonstrating the model is making progress.

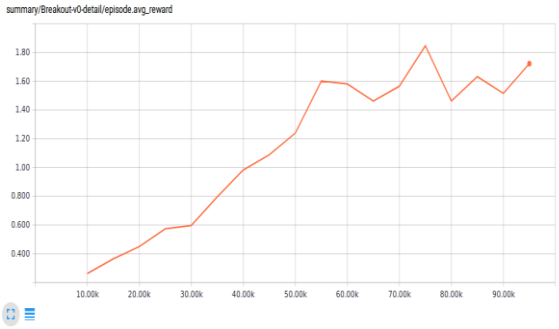

Figure 1: This plot shows the average reward per episode for Breakout during training.

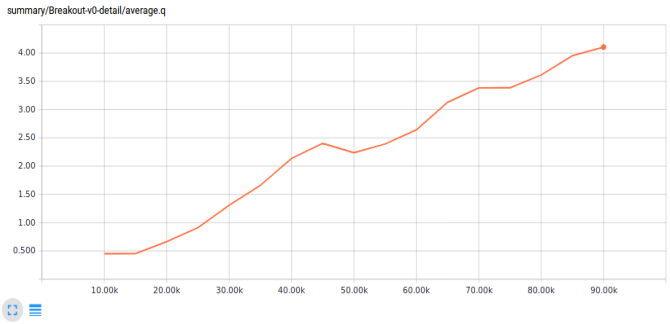

Figure 2: This plot shows the average Q-value over time for Breakout during training.

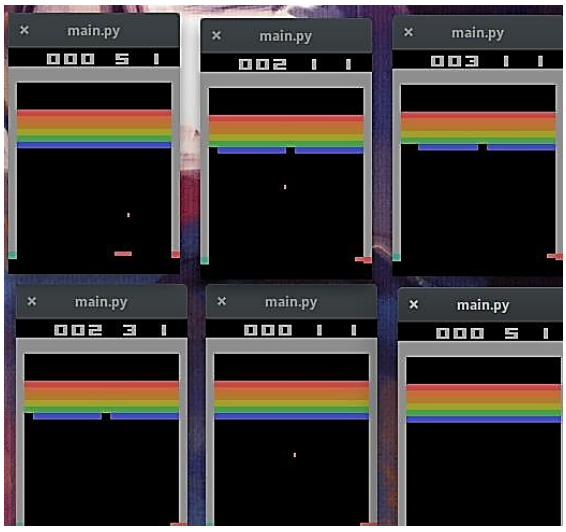

Figure 3: Training Breakout using six actor-learner threads

VI. CONCLUSION

In this project we implemented and evaluated a Deep Reinforcement Learning algorithm for the challenging task of video game playing. We found that by using an asynchronous variant of Deep-Q learning there is a significant performance improvement. This is achieved by removing replay memory and using multiple actor-learner threads which drastically reduces training time.

The algorithm was not able to generalize effectively with more complex games that we tested on. The training time for Atari Breakout was over 30 hours. Other asynchronous methods such as n-step Deep-Q learning and the policy-based advantage actor-critic method have been shown to outperform one-step Deep-Q learning for several Atari games. Future work could explore these alternate approaches.

REFERENCES

[1] DeepMind Technologies, "Playing Atari with Deep Reinforcement Learning".

[2] DeepMind Technologies, "Human-level control through deep reinforcement learning", Nature.

[3] DeepMind Technologies, "Asynchronous Methods for Deep Reinforcement Learning", arXiv:1602.01783v2 [cs.LG] 16 Jun 2016 .

[4] Devsisters Corp, "https://github.com/devsisters/DQN-tensorflow"