Distributed Data Management CS750Shouvik Sarkar

**Pong Using Advantage Actor Critic**  192CS022 M.Tech CSE Semester II

7749996142

[shouviksarkar95@gmail.com](mailto:shouviksarkar95@gmail.com)

**Abstract**

I present a comparison of computational efficiency between training an agent for playing Pong in the Atari environment in a sequential manner and in a parallelized manner. The agent is trained using a Deep Reinforcement Learning class of algorithm Advantage Actor Critic (A2C). The parallel version of the algorithm is analyzed using multiple CPU cores on a single machine instead of special hardware like GPU(Graphical Processing Units) and TPU(Tensor Processing Units). This work presents the training gain that can be achieved by parallelizing the training. The comparison is done on a 24 hour time interval. A note to be taken here is that the results obtained do not say anything about the algorithm itself. More extensive and rigorous experiments have been carried out in [1,2,3]. The findings here should be treated as those obtained in a simplistic setting.

**Keywords**

Advantage Actor Critic, Asynchronous, Synchronous, Deep Reinforcement Learning, Q-Learning, Policy Gradients

**Introduction**

Reinforcement Learning algorithms are a class of algorithms that have proven to be quite promising in the field of decision making. That makes games a suitable area for their application. Combine them with Deep Learning which is known to produce rich representations of input data (visual data in this case) and we get the ability to play games at a human or even better-than-human level [4]. In the past few years the significance of Deep Reinforcement Learning has grown significantly in the fields of games and robotic planning in particular and is also believed to be an important aspect in building agents that are able to transfer learned knowledge across domains.

This work’s focus is on games. The game chosen here is Pong which is an atari game [6] and is provided as an environment in OpenAI gym [5]. The implementation is available at <https://github.com/shouvikcirca/PongUsingA2C>. The attempt here has been to write a sequential and a parallel version of a variant of Advantage Actor Critic algorithms. Owing to limited hardware and time, both the versions have been trained for 24 hours. The sequential version involves only one agent playing and learning in a single environment. But the parallel version involves multiple agents rolling out in their independent environments and their learnings are ultimately combined and put into a final model using asynchronous or synchronous methods.

**Literature Review**

In [4] they use high dimensional raw sensory (visual) input to train an agent to play 7 Atari games using Convolutional Neural Networks (CNNs) and a variant of Q-Learning which is a class of Deep Reinforcement Learning algorithms. The algorithm used here makes use of Q-Learning in part to provide a learned baseline [3]. The problem with Q - Learning is that the policy, more about which has been described later, has to be decided by us and so the choosing of actions is dependent on the policy we describe. This can be overcome by having a neural network learn the policy by itself [3]. That is what describes another class of algorithms called Policy Gradient methods. The algorithm used here makes use of this method in part as well.

Reinforcement Learning is much harder than Supervised Learning because RL agents only have arbitrary signals to learn from instead of a target which is a component of Supervised Learning. As a result they require way more time to converge. [2] and [3] describe strategies of distributing

the learning amongst multiple processor cores [2] and GPUs [3]. [1] describes and highlights the problem with an asynchronous way of updating parameters. However such a strategy suffers from the problem of *stale gradients* where the independent nodes might be working on outdated parameters. So they come up with a synchronous way of updation where there is a master node receiving gradients from the slave nodes and only the master node does the updation.

**Reinforcement Learning**

The setup consists of an agent A that interacts with the environment E. The agent interprets the environment by moving from one state s t at time step t to another state st+1 at time step t + 1. In each state it receives an observation in the form of an n-dimensional tensor which might contain partial or all information about the environment in that state. It chooses an action from an action space A = (a1 , a2 , ....am ) based on some computation and as a result lands in another state and receives a signal called the reward Rt quantifying how good or bad the action was.The agent keeps accumulating these rewards till it reaches the goal or the episode terminates. An episode runs for a predefined t max number of time steps. The action at each time step is taken by the agent in accordance with a policy π that is either learned as in Policy Gradient methods or predefined as in Value Function Based methods. A policy is a function that given a state s, gives a probability distribution over A. The value of a state s under a policy π is the expected return when starting in s and following π thereafter. Similarly the value of taking action a in state s under policy π is the expected return starting from s, taking the action a and thereafter following policy π.

**Advantage Actor Critic**

The agent here consists of 2 neural networks. One of them, the Actor, spits out a probability distribution over a set of actions A at each time step and learns the optimal policy, based on a loss function, more about which is described later. The other neural network, the Critic calculates the action-value at each time step telling how good or bad the action taken is and learns to give better estimates of the value, that is, it learns to be a better critic.

Advantage is a function that tells how good or bad the actor network performed relative to the critic’s expectations.

**Loss Function**

Let us have a look at the components one by one and finally bring them together

1. We use the estimated value by the critic as a baseline to tell how good or bad the actor network is performing. For this we use the advantage function

(E1)

where Vπ denotes the total value estimated at the end of the episode and G is the total discounted reward at the end of the episode.

(E2)

(E3)

γ is the discount factor used to place decaying importance, in this case, on later time steps so that earlier steps are reinforced. In Pong, if the opponent misses, it was probably because the shooter maneuvered the ball in a good way on receiving. Similarly earlier actions had a greater role to play in helping the agent become a better player.

2. The log probability of the sampled action is given by

= log max P ( | ) (E4)

where ai A. One of the reasons why we use log probabilities is that probabilities are confined in [0,1] while log-probabilities are bounded in (−∞,0) and hence a finite number of bits can represent a wide range of values without overflowing or underflowing the computer’s numerical precision.

3. The Actor loss is given by

− ( G − Vπ) (E5)

Where (G− Vπ) is the advantage function and G is the total return.

4. The Critic loss is given by

(G − Vπ)2 (E6)

5. Finally the agent’s loss is given by

actorLoss + c.CriticLoss (E7)

where c is a parameter used to control the relative learning of the actor and the critic.

The implementation is done in a way such that the actor and the critic have some parameters in common. The critic does not update these shared parameters because it can lead to instability.

**N Step Learning**

Instead of waiting for the completion of an entire episode for updating the parameters,it is done every n steps where n is a hyperparameter. If we had gone for 1-step learning where update is done after every time step that will introduce a lot of bias whilst updating after an entire episode makes the model gullible to high variance. The n-step approach is a tradeoff between bias and variance.

**Backpropagation using Gradient Descent**

Gradient Descent is the most widely used optimization method used in Deep Learning. It involves calculating the gradients of parameters with respect to a loss function. It requires that there be points of local minima in the loss function where the model is likely to give a better performance. Updation of parameters in the direction of those minima by changing them against the direction of their gradients is done by *Backpropagation* (9).

**Parallelized Setup**

The setup consists of multiple cores being used, each of which belongs to the same system. Each process gets its own copy of the environment and the agents and each agent learns independently and updates the parameters of the model asynchronously (3) or synchronously (1). In the asynchronous approach, the different processes share the model’s parameters and hence update the parameters independently, however these can lead to stale gradients[reference] as a process might be working on outdated parameters. In the synchronous setting, there is a central process which waits for all the processes to send their updated weights, averages them and then updates the actual model which is then shared to the other processes and the iterations continue.

**Atari 2600 Pong**

Pong is one of the Atari games (6). Each state is available as a 3 dimensional tensor of dimensions 3 X 210 X 160 where 3 represents the RGB channels. Each element is an integer in [0,255]. There are six actions with three of them being redundant. FIRE equals NOOP, LEFT equals LEFTFIRE and RIGHT equals RIGHTFIRE. LEFT and RIGHT move the agent in the corresponding directions while NOOP keeps the agent in its previous position. The agent is displayed on the right and the enemy on the left (7). If the agent misses the ball a reward of -1 is received, if the enemy misses the ball a reward of +1 is received. In all the other states a reward of 0 is received.The first to score 21 goals wins and that marks the end of the episode.

A reward is what the agent implicitly makes use of to learn while the points displayed on the screen are the scores of each of the players. The winner is the first player to score 21.

**Algorithm for Sequential Training**

For Number of episodes

1.While number of timesteps < N and episode is not complete

1. Take raw pixel input of state frame
2. Crop the image
3. Convert to grayscale
4. Flatten the 2D pixel input to a 1D-vector
5. Feed into the model that spits out action to take and a scalar Value .
6. Feed the action to the environment. This will take the game to the next state.
7. Store and and

2. Calculate Loss in accordance with (E7)

3. Backpropagate Loss and update parameters

*A1*

The algorithm for the sequential version is given in A1. In the parallel version there are multiple instances of the mentioned algorithm. The asynchronous version works by letting each of the independent instances update to one central shared model. In the synchronous version, each instance, after a predefined number of episodes, sends the parameters to a central process which averages the corresponding parameters and tests on one central model. This is done iteratively till desired performance is achieved.

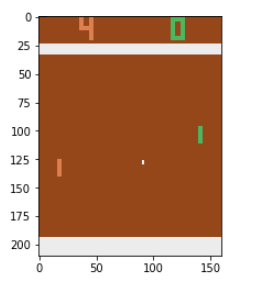
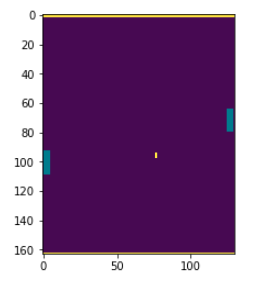
**Implementation**

The model is a 2-headed neural network. One head is the Actor and the other, the Critic.The first few layers of the network are shared. Only fully connected layers have been used. The input is flattened giving an input size of 100800. The entire frame is not needed for decision making as is shown in the figure 1. So the input is cropped and converted to grayscale before feeding into the network. Modifications are illustrated in FIgure 1. That brings down the input size to 21190 since each frame is now 1 X 210 X 160.

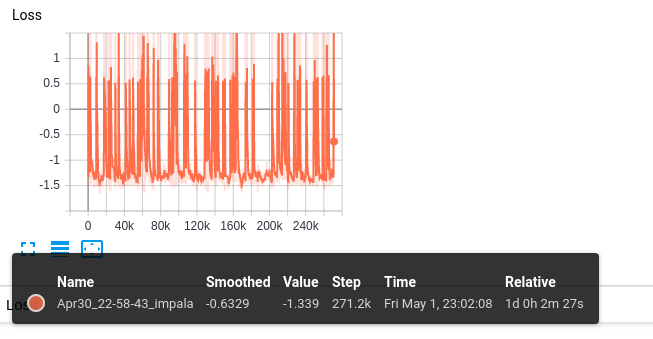
The output of the Actor head consists of three nodes giving a probability distribution over the action space. The output of the Critic head has one node because it gives a scalar. The learning rate chosen is 10−3 and the optimizer used is Adam (8) . These hyperparameters were chosen in accordance with [1]. The value of N was chosen to be 100. The Critic network does not update the shared parameters.

The training loop was designed in a way that the epoch which gave the lowest loss had the parameters saved after updation. The graph is shown in Figure 2. It includes information like the **Relative** is duration of training that is 1 day 0 hours 2 minutes 27 seconds, **Step** that is the number of epochs, **Time** is the timestamp of the measurement, **Value** is the loss at that timestamp, **Smoothed** is the focus on the graph and **Name** gives information about when the training was started. Tensorboard has been used to plot the graph.

The reward system was reconfigured to +5 on scoring a point and -6 if the opponent gets a point during the training process.



*Figure 1a on the left shows a state frame as provided by the environment. Figure 1b on the right shows the same frame after cropping and converting to grayscale.*

*****Figure 2 shows the loss function curve for the sequential training and some other information.*

*Epochs are on the horizontal axis and the loss is on the vertical axis.*

**Results**

The model was trained sequentially for about 24 hours and 271205 epochs. The loss graph is given below. The average score per game over 1000 games was 0.327. The distribution of the scores over the games is given in Table 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Score | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| Frequency | 789 | 132 | 57 | 14 | 6 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Table 1*

*Table 1 shows the frequencies of the scores of the trained agent over 1000 games.*

The score distribution for the untrained agent is given in Table 2. Though the average score is greater for the untrained agent (0.506), we can see that the trained agent scored 9 points in one of the games. That could perhaps be an indicator that the agent is learning and greater training time would yield better results.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Score | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| Frequency | 644 | 241 | 87 | 22 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

*Table 2*

*Table 2 shows the frequencies of the scores of the untrained agent over 1000 games.*

**Future Work**

The network used here consists only of fully connected layers. Since the input is an image, the obvious next step would be to try variants of Convolutional Neural Networks

in the model for which there is an enormous amount of literature out there.

**References**

[1] Adamski, Igor, Adamski, Robert, Grel, Tomasz, Jedrych, Adam, Kaczmarek, Kamil,

Michalewski, Henrik

*Distributed Deep Reinforcement Learning: Learn how to play Atari games in 21*

*Minutes*

arXiv:1801.02852 [cs.AI]

[2] Adamski, R., Grel, T., Klimek, M., Michalewski, H.

*Atari games and intel processors*

CoRR abs/1705.06936 (2017)

[3] Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., Kautz, J

*Reinforcement learning through asynchronous advantage actor-critic on a GPU*

CoRR abs/1611.06256 (2016)

[4] DeepMind

*Playing Atari with deep reinforcement learning.*

NIPS Deep Learning Workshop (2013)

[5] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba,

W.

*OpenAI Gym. CoRR abs/1606.01540 (2016)*

[6] Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.

*The arcade learning environment: An evaluation platform for general agents.*

CoRR abs/1207.4708 (2012)

[7] Atzmueller, Martin, Duivesteijn, Wouter

*Artificial Intelligence*

[8] Kingma, P. Diederik, Ba, Jimmy

*Adam: A Method For Stochastic Optimization*

CoRR abs/1412.6980 (2014)

[9] Rumelhart, D. E., Hinton, G. E., and Williams, R. J

*Learning representations by back-propagating errors*.

Neurocomputing: foundations of research. MIT Press, Cambridge, MA, USA, 696-699