Using Importance Weighted Actor-Learner Architecture with V-Trace as a rollout policy for Monte Carlo Tree Search

Ambarish A Gurjar
Intelligent Systems Engineering
Indiana University
Bloomington, IN 47408
agurjar@iu.edu

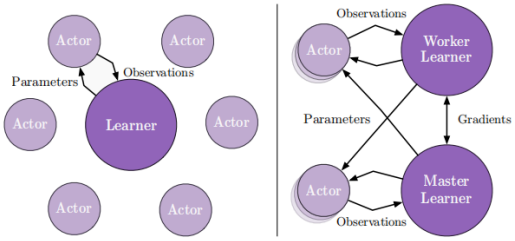
Palash Chatterjee
Data Science
Indiana University
Bloomington, IN 47408
palchatt@iu.edu

Abstract

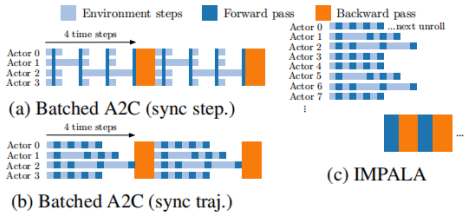
A combination of MCTS and A3C has been shown to perform better than the MCTS And A3C separately. IMPALA has been shown to provide better results over A3C. In this course project, we study the performance of single learner architecture for IMPALA as a rollout heuristic for MCTS algorithm.

1 Background

1.1 IMPALA and the V-trace algorithm



Left: Single Learner. Each *actor* generates trajectories and sends them via a queue to the *learner*. Before starting the next trajectory, *actor* retrieves the latest policy parameters from *learner*. **Right: Multiple Synchronous Learners.** Policy parameters are distributed across multiple *learners* that work synchronously.



Timeline for one unroll with 4 steps using different architectures. Strategies shown in **(a)** and **(b)** can lead to low GPU utilisation due to rendering time variance within a batch. In **(a)**, the actors are synchronised after every step. In **(b)** after every n steps. IMPALA **(c)** decouples acting from learning.

1.1.1 Overview of IMPALA

In A3C-based agents[6], workers communicate gradients with respect to the parameters of the policy to a central parameter server. In IMPALA[4], the actors communicate trajectories of experience to a centralised learner. Since this learner has access to complete trajectories of experience, it performs updates using mini-batches of trajectories. However, because the policy used to generate a trajectory may lag behind the policy on the learner at the time of gradient calculation, an off policy actor critic algorithm called V-trace to correct for this discrepancy.

The objective of the IMPALA algorithm is to learn a global policy π and a baseline value function V_π. The individual actors have their own policy μ which is used to generate n step trajectories of state, action and rewards $x_1, a_1, r_1, ..., x_n, a_n, r_n$ which can be defined as $(x_t, a_t, r_t)_{t=s}^{t=s+n}$ that is sent to

a learner queue. The paper also defines two importance sampled truncated weights namely c_i which is used to control the speed of convergence and ρ_t which is the value function of the policy μ. The paper also defines several equations that make use of the v-trace operator to update value functions and the policies involved.

1.1.2 V-trace operator

The V-trace operator is defined as $\mathcal{R}$:

$$\mathcal{R}V(x) = V(x) + \mathbb{E}_\mu \Big[\sum_{t \geq 0} \gamma^t \big(c_0 \dots c_{t-1}\big) \rho_t \big(r_t + \gamma V(x_{t+1}) - V(x_t)\big) \big| x_0 = x, \mu \Big]$$

where the expectation $\mathbb{E}_\mu$ is with respect to the policy μ which has generated the trajectory $(x_t)_{t \geq 0}$, i.e., $x_0 = x$, $x_{t+1} \sim p(\cdot|x_t, a_t)$, $a_t \sim \mu(\cdot|x_t)$. Here we consider the infinite-horizon operator but very similar results hold for the n-step truncated operator[4].

1.2 Monte Carlo Tree Search

Monte Carlo Tree Search is a class of algorithms[2] that combines tree search with Monte-Carlo evaluation. It performs confidence bounded reinforcement learning by performing rollouts that are used to construct a tree for optimum decision taking. From a rootnode, child nodes are recursively selected until the algorithm reaches a node that requires expansion. The expanded node is simulated within a computational budget and the results are backpropogated through the tree[3].

1.3 Previous Work

Subramanian and Crowley[9] use MCTS with A3C for predicting spatially spread forest fires and report results that beats A3C, MCTS, DQN[7] and DQN with prioritized experience replay[8] in terms of accuracy. They performed 9 experiments out of which 8 demonstrated an increase boost in accuracy with this approach. However, the reward structure of that MDP was structured for a predictive outcome. In the next section we will describe our proposed method to maximize rewards in a simulation environment.

2 Algorithm

Every iteration of Monte Carlo Tree Search goes through 4 phases - selection, expansion, simulation and back-propagation. We propose to use UCT for the process of selecting nodes within the tree [5], and the single learner IMPALA architecture along with V-trace as the rollout policy. The pseudocode for the proposed algorithm is provided in Algorithm 1.

Algorithm MCTS-IMPALA

```
1:  procedure MCTS(Rootnode, time)
2:      while true do
3:          currentnode ← rootnode
4:          while currentnode ≠ lastnode do
5:              lastnode ← currentnode
6:              currentnode ← UCT-Selection(currentnode)
7:          lastnode ← EXPANSION(lastnode)
8:          SIMULATE using IMPALA
9:          while currentnode do
10:             BACKPROPAGATE(currentnode)
11:             currentnode ← parent(currentnode)
```

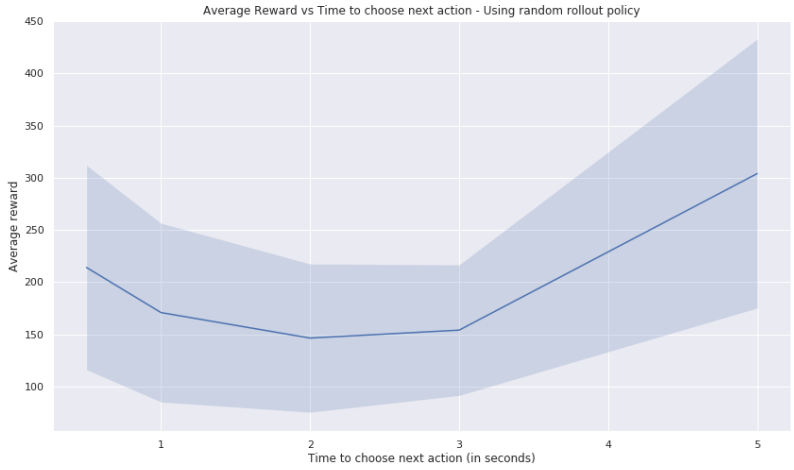
3 Experiments

3.1 Environments

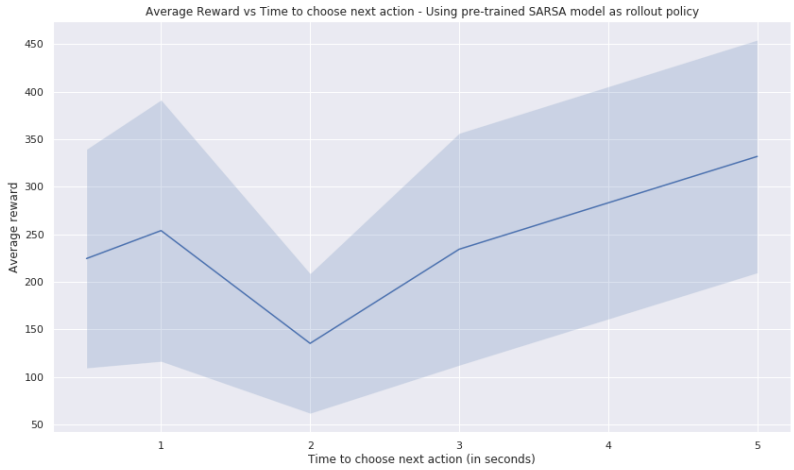
For the purpose of this project, we utilize the Cart Pole environment provided by the OpenAI Gym [1]. The code is available on IU github *

3.2 MCTS

Monte Carlo Tree Search constructs a tree with states as the nodes and actions as the edges. The standard implementation of MCTS uses UCT for selecting the actions until it encounters the leaf node. When it encounters the leaf node, it performs multiple simulations by choosing actions from the action space randomly and playing till the end of the episode. The value of the leaf node is then determined by the average reward as a result of the simulations.



(a)



(b)

Figure 1: (a) Using a random policy as rollout policy (b) Using pre-trained SARSA as rollout policy

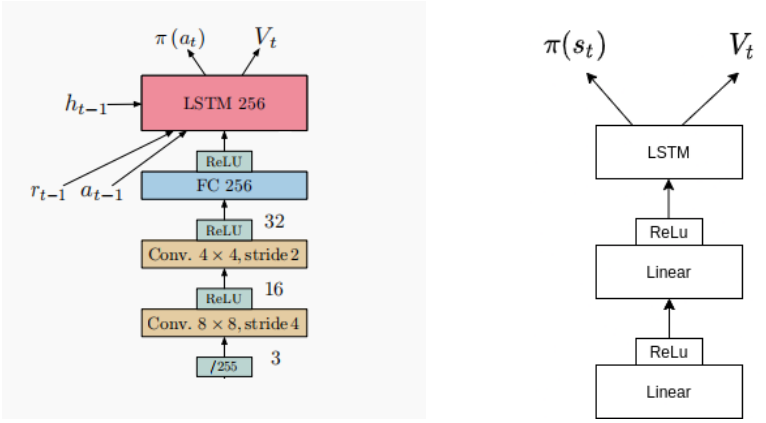
*https://github.iu.edu/palchatt/mcts_impala

We tried two variants of the MCTS algorithm. In the first one, the algorithm took random actions for performing simulations. For the second variant, we took a pre-trained SARSA model, and used that as the rollout policy.

Figure(1) shows the variation of average reward with increasing amount of time taken for performing action selection. As can be observed from the graph, when the algorithm is allowed sufficient time to choose an action, both the variants perform well. When less time is provided to the algorithm to choose an action, the pre-trained SARSA model performs better.

3.3 IMPALA

3.3.1 Architecture



(a) (b)

Figure 2: (a) The original shallow network architecture proposed in the paper. (b) The architecture used for the purpose of this project.

The agent in the original paper was targeted to solve Atari. It used a 2-layer CNN, connected to a dense layer followed by an LSTM Cell with two outputs. Since we were working with Cart Pole environment, we decided to simplify the architecture. We use a 2-layer FCN followed by an LSTM Cell with two outputs.

3.3.2 Hyperparameters

The IMPALA architecture has quite a number of parameters that can be tuned which makes it difficult to tune.

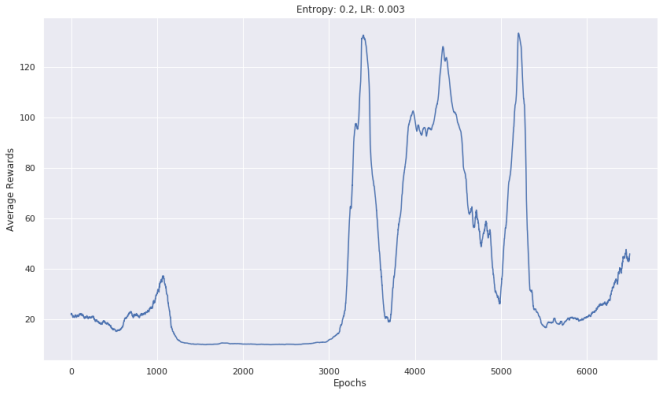
We experimented with tuning entropy and the learning rate of the agent. The other values such as the baseline cost and gamma were kept as per the original paper. We measured the running average of rewards in the last 100 iterations. This was chosen as the metric since OpenAI uses this metric to decide if an environment is solved or not.

In Figure(3) we show one such result where the entropy was kept fixed at 0.2 while the learning rate was varied. It was observed at the best results were obtained for a learning rate of 0.004. For learning rates of 0.003 and 0.005, the average rewards never went above 195 mark which OpenAI considers as the benchmark for solving Cart Pole.

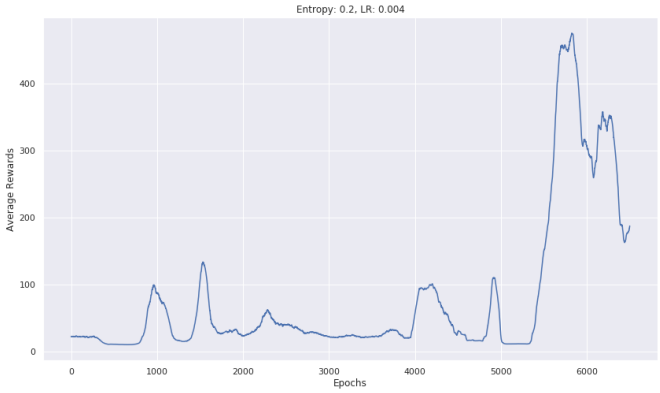
For a learning rate of 0.003, it is observed that the rewards keep fluctuating. Since the learning rate was lower, the agent was allowed to run for a greater number of epochs. However, the agent failed to perform poorly. The complete graph is shown in Figure(4). For learning rate of 0.005, it is observed that the average rewards spike up initially to around 120, but it never recovers and performs extremely poorly.

Despite trying out various combinations, we were not able to make the model converge.

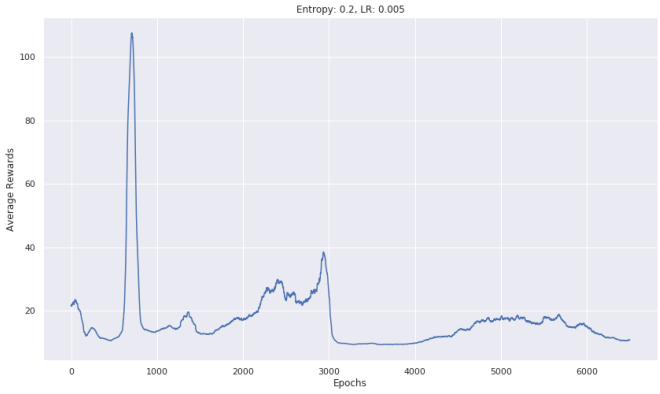
We also tried tweaking the batch size however the results didn't show any significant difference.



(a)



(b)



(c)

Figure 3: The average rewards calculated are running averages of the last 100 epochs. The entropy is fixed at 0.2 and the learning rate has been varied for these three cases. The learning rates are (a) 0.003 (b) 0.004 , and (c) 0.005. The graphs have been truncated to 6500 epochs to make them consistent with each other.

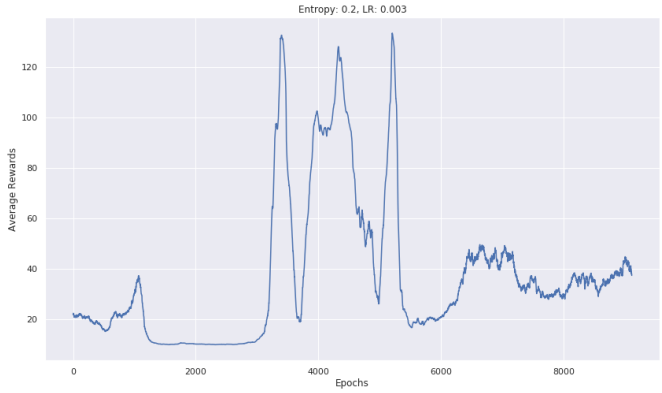


Figure 4: Variation of average rewards with number of epochs. Entropy: 0.2 and LR: 0.003

3.4 MCTS + IMPALA

3.4.1 Pre-trained IMPALA as Rollout policy

The current implementation uses a pre-trained IMPALA model to perform simulations in MCTS. The IMPALA model was trained with entropy = 0.2 and LR = 0.004. The model was saved when the running average was at the peak.

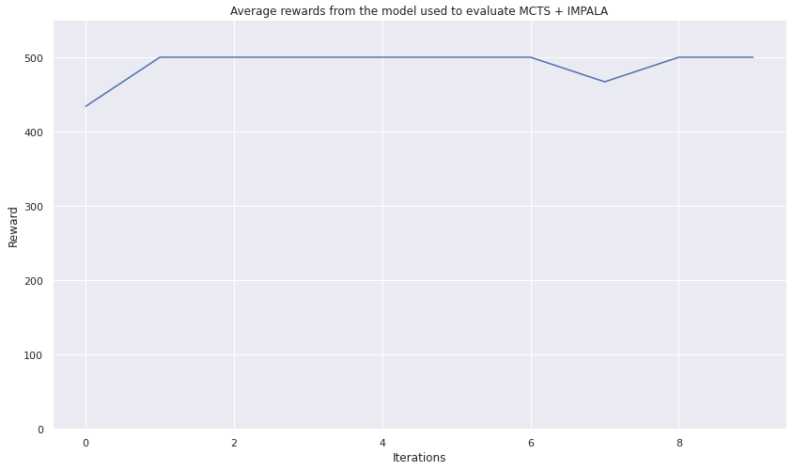


Figure 5: Rewards obtained from the saved model

Figure (5) show the returns obtained using this model. It is observed that the performance of the model created using IMPALA is extremely good.

It is observed that this agent doesn't perform as well as the agent using random policy.

3.4.2 Simultaneous training of IMPALA

The proposed approach to simultaneously train the learner from trajectories of the MCTS rollout reaches an early termination. Our assumption is that the MCTS makes moves with respect to a learner weights that have not matured enough to start giving stable rewards. As seen from the performance graphs, the IMPALA learner takes a substantial amount of time (5000 iterations) before it can start producing meaningful heuristic for the MCTS. We have proposed an approach in future work that should take care of this.

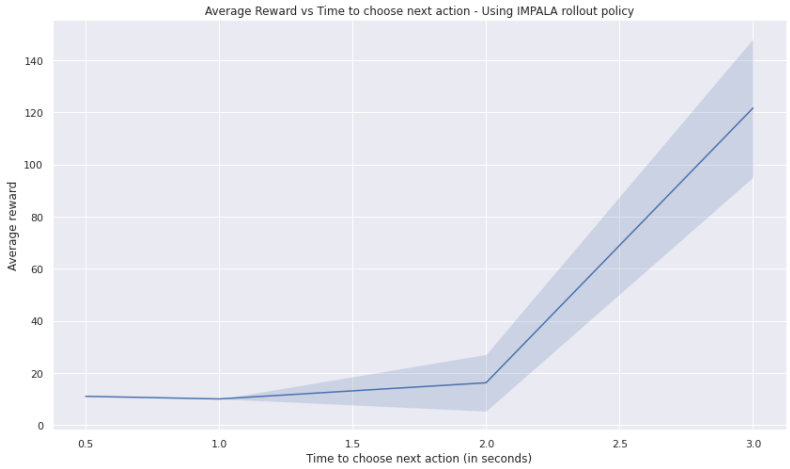
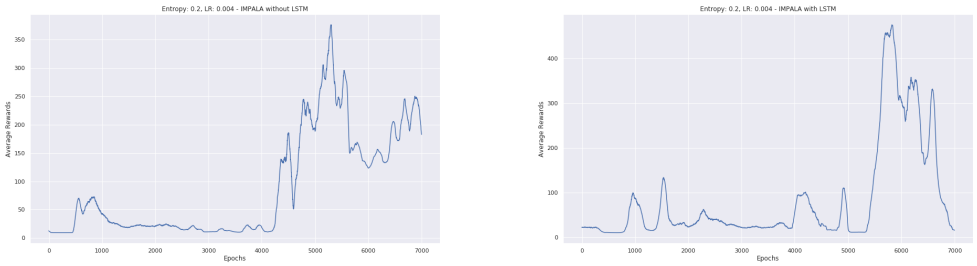


Figure 6: MCTS with IMPALA as the rollout policy

Another problem we faced were the unstable models. For simultaneously training IMPALA with MCTS, the learner should be stable in order to provide good heuristics consistently. We hope that this can be achieved if the learner is fine tuned further and is allowed to train for a longer duration of time.

4 Ablation Study

We wanted to test if for simple environments, like Cart Pole, LSTM was required. We evaluated the models with and without LSTM with the same values of hyper-parameters. We found that the results were at par with the agent with LSTM. They did seem to be more stable than the agent with LSTM. However it is important to note that the intermediate rewards are slightly higher than with LSTM.



(a) Performance of IMPALA with LSTM

(b) Performance of IMPALA without LSTM

Figure 7: Comparison of architectures as an ablation study

5 Challenges

During the course of the project, we faced numerous challenges. The foremost was trying to tune IMPALA. With multiple hyperparameters available, it was difficult to tune it for Cart Pole. We started with the parameters made available as a part of the paper and decided to tune entropy cost and the learning rate.

We also tried to get the learner on CUDA while keeping the actors on CPU, so that the learning can be done in an efficient manner. However, we were not able to do so. The current implementation keeps both the learner and actors on CPU.

6 Future Scope and Improvements

As is evident from the results, the IMPALA models currently don't converge. The models need to be tuned further to ensure they are stable.

In future we plan to get the algorithm running on the GPU. We also plan to test it against complicated environments like the Atari Arcade Learning Environment and Mujoco (With a student license).

At present, we use a pre-trained IMPALA model with MCTS to perform simulations. We had to take this route since training the learner on CPU is extremely time-consuming. If the learner runs on CUDA, then a possible scope of work can be to have a persistant learner which is trained alongside MCTS.

We also propose that we set a very high threshold for actor processes to timeout in the start. This way the learner will be much more mature per every MCTS step. The exploration (or in this case how many actions the actors take) will gradually reduce as the learner estimates a better model.

In the current version we have implemented only a single learner. The architecture defined in the original paper can also leverage the presence of multiple learners to achieve a higher throughput. We plan to see how the multiple learner architecture behaves with respect to our proposed algorithm.

Acknowledgments

The authors would like to thank Sriram Ganapathi Subhramaniam from the University of Waterloo for providing insights and feedback.

References

[1] *Brockman Greg, Cheung Vicki, Pettersson Ludwig, Schneider Jonas, Schulman John, Tang Jie, Zaremba Wojciech*. Openai gym // arXiv preprint arXiv:1606.01540. 2016.

[2] *Browne Cameron B, Powley Edward, Whitehouse Daniel, Lucas Simon M, Cowling Peter I, Rohlfshagen Philipp, Tavener Stephen, Perez Diego, Samothrakis Spyridon, Colton Simon*. A survey of monte carlo tree search methods // IEEE Transactions on Computational Intelligence and AI in games. 2012. 4, 1. 1–43.

[3] *Coulom Rémi*. Efficient selectivity and backup operators in Monte-Carlo tree search // International conference on computers and games. 2006. 72–83.

[4] *Espeholt Lasse, Soyer Hubert, Munos Rémi, Simonyan Karen, Mnih Volodymyr, Ward Tom, Doron Yotam, Firoiu Vlad, Harley Tim, Dunning Iain, others* . IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures // ICML. 2018.

[5] *Kocsis Levente, Szepesvári Csaba*. Bandit based monte-carlo planning // European conference on machine learning. 2006. 282–293.

[6] *Mnih Volodymyr, Badia Adria Puigdomenech, Mirza Mehdi, Graves Alex, Lillicrap Timothy, Harley Tim, Silver David, Kavukcuoglu Koray*. Asynchronous methods for deep reinforcement learning // International conference on machine learning. 2016. 1928–1937.

[7] *Mnih Volodymyr, Kavukcuoglu Koray, Silver David, Rusu Andrei A, Veness Joel, Bellemare Marc G, Graves Alex, Riedmiller Martin, Fidjeland Andreas K, Ostrovski Georg, others* . Human-level control through deep reinforcement learning // Nature. 2015. 518, 7540. 529–533.

[8] *Schaul Tom, Quan John, Antonoglou Ioannis, Silver David*. Prioritized experience replay // arXiv preprint arXiv:1511.05952. 2015.

[9] *Subramanian Sriram Ganapathi, Crowley Mark*. Combining MCTS and A3C for prediction of spatially spreading processes in forest wildfire settings // Canadian Conference on Artificial Intelligence. 2018. 285–291.