International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 04 | Apr 2020 www.irjet.net p-ISSN: 2395-0072

# Asynchronous Advantage Actor Critic with Random Exploration Exploitation (A3C-REE)

**[1]Eashan Kaushik**

*[1]School of Computer Science and Engineering, Vellore Institute of Technology(Student), Chennai, India.*

---

**Abstract -** *Asynchronous Advantage Actor Critic algorithm was developed in the field of Deep Reinforcement Learning. It was first introduced in 2016 by Google's DeepMind [1] artificial division. A3C algorithm is a state-of the art algorithm and it outshines any of the current existing artificial intelligence algorithms. In this paper we would like to discuss in great detail the architecture of A3C in which we will talk about the 3 A's of A3C and how these 3 A's come together in the algorithm. We will propose "Random Exploration Exploitation" (A3C-REE) approach to this algorithm which will enable the algorithm to balance between exploration and exploitation. Finally, we will implement A3C algorithm to play Breakout game and see how might random exploration and exploitation be advantageous.*

***Key Words***: **Asynchronous, Advantage, Actor, Critic, Exploration, Exploitation.**

## 1. INTRODUCTION

We have come a long way from using Supervised Learning Models in which the training dataset comes with an answer key and we use this answer key to train our model. However, in Reinforcement Learning thee agent learns from the environment it is introduced in. There is no answer key, agent is given a positive or a negative reward based on its actions. Our goal is to maximize this reward in order for the agent to learn through trial-and-error. One of the most effective way to implement RL is Deep Q-Network [5]. Experience Replay [2] plays a very important role in training of Deep Q-Network.

In ER, we store the agents experience at each time step ($e_0$, $e_1$, .... $e_n$) in the data set called the replay memory. At a particular time, t, the agents experience $e_t$= (Current State-$S_t$, Action taken-$A_t$, Reward Received-$R_{t+1}$ ,Next State-$S_{t+1}$) is stored in the memory, the data is then batched or randomly sampled for the agent to learn from. The reason ER exists is because if we use the online RL agent to sample transitions there would be a strong chronological relationship between them. Stochastic gradient decent works best with identically and independently distributed samples. Deep Q-Networks based on ER have achieved lots of success in last few years. However, ER has several drawbacks, (1) ER uses more memory and computation per real interaction and also (2) it requires off-policy learning algorithms that can update from data generated by an older policy.

A3C algorithm doesn't use experience replay, instead it asynchronously executes multiple agents on multiple instance of the environment in parallel. This parallelism makes the process more stationary as at any given time-step the parallel agents will be experiencing a variety of different states. A3C algorithm takes full advantage of exploration and exploitation, the two core fundamentals on which RL is dependent. Our ability to use N agents gives us the opportunity to explore our environment properly and swiftly. Also once the agents find the best action, it also exploits this action to give better results.
Previous approaches RL heavily relied on specialized hardware such as Graphical Processing Unit(GPU) [3] or massively distributed architectures [4], A3C can run on a single machine with a standard multi-core CPU. This gives A3C practical advantages as well.

## 2. ARCHITECTURE

### 2.1 Actor-Critic

The architecture of A3C is not so different from a convolutional neural network the only difference between the two is that the A3C outputs two values: The Actor and The Critic. The actor gives a set of Q (S, $a_n$) values, which gives the probability to take a particular action $a_n$ (Probability distribution over actor space), this defines the policy based value of the algorithm. The critic outputs the Value Function V(s) which is used to measure how good the action taken is. The actor critic is like the brain of the A3C model. The agent uses the value of Value Function V(s) to update the policy function.

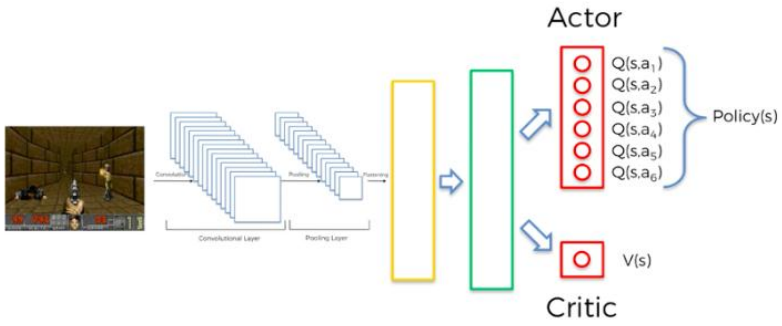


**Figure 1:** A3C Model [7]

## 2.2 Asynchronous

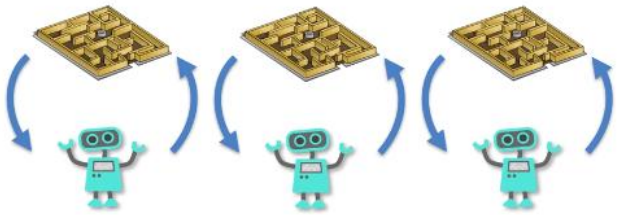

**Figure 2**: Agents and Environments [7]

The main advantage of using A3C is its ability to asynchronously execute multiple agents in multiple instances of the environment. This gives our agent an advantage to explore the environment more meticulously. As several agents attack the environment, they take different paths to reach the end goal as a result it prevents the agent from getting stuck in the local maxima.

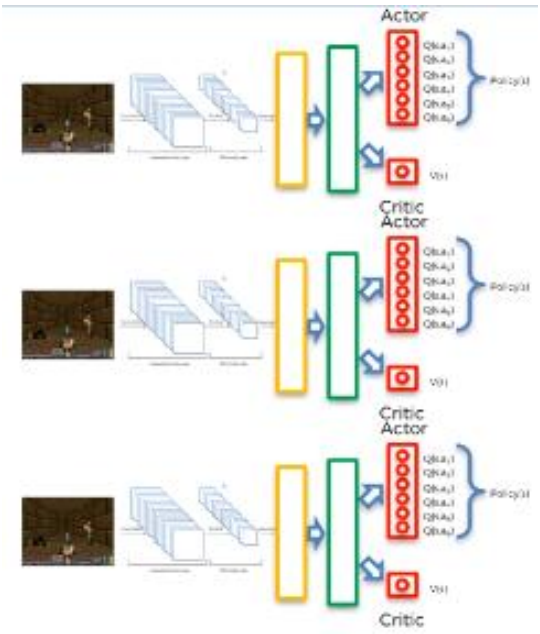


**Figure 3:** Asynchronous [7]

On a basic level we can visualize that we are attacking the same environment using N (number of agents is the above diagram N=3) agents, tripling the amount of experience at a given time. These agents explore the environment independently from the other agents. As these agents have no connection to each other, we need a way these agents share their experience with one another, for this purpose we have the Critic. The value function V(s) is shared among the three agents.

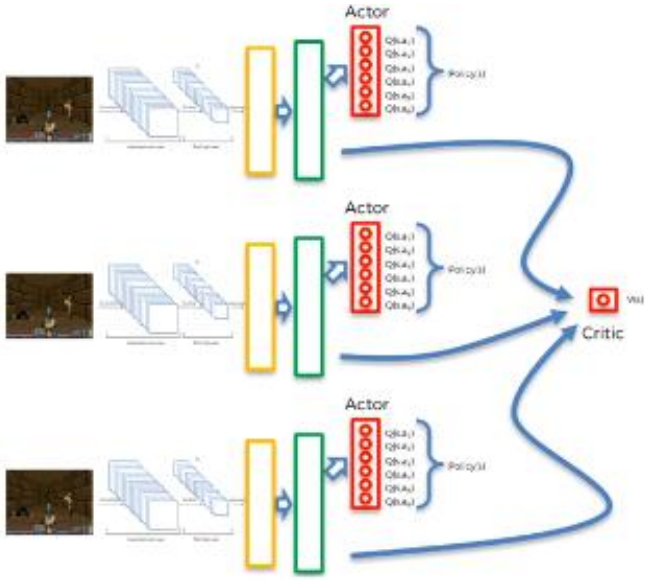


**Figure 4:** Shared Critic [7]

The shared V(s) value is the expected value of the states. This value is compared with the output V(s) value of the three agents and this helps the agents to adjust their neural networks (***Value Loss***).

Suppose instead of three different neural networks the agents have one common neural network. This implementation is a much better implementation as all the agents will make adjustments to one neural network. Giving the neural network more experience and will increase its accuracy considerably. This network is called Global Network.

### 2.3 Advantage

$$A = Q(S,a) - V(S)$$

This is the advantage equation. Advantage is calculated for each agent and their goal is to maximize this value. Suppose after applying softmax we get Q value Q(S,a), the agent needs to compare how good or bad this value is from the current value of the state. We subtract the two values to give the advantage. This is used to calculate the ***Policy Loss***. If the actor selects a good action i.e. Q(S,a) > V(S) advantage will be a positive value, however if it's a bad action Q(S,a) < V(S) then advantage will be a negative value. Therefore, Advantage is used to improve the agents behaviour by making it choose more of good actions and less of bad actions.

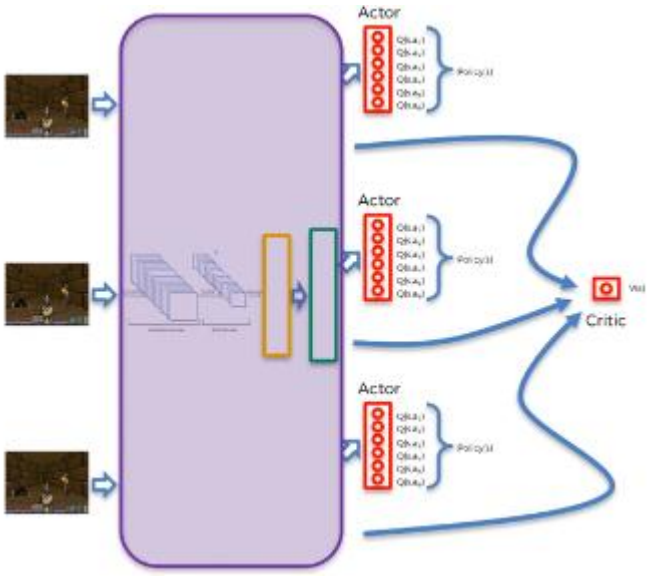


**Figure 5:** A3C Better Implementation [7]

### 2.4 Policy Loss and Value Loss

**Equation 1:** Policy and Value Loss [9]

$$Value\ Loss: L = \Sigma(R - V(s))^2$$

$$Policy\ Loss: L = -log(\pi(s)) * A(s) - \beta * H(\pi)$$

An agent then uses these losses to obtain gradients with respect to the parameter of its network. These gradients are used to update the global network parameters. Global network is constantly being updated by each of the agents. However, each of these gradients should be clipped in order to prevent large parameters update which can destabilize the policy.

## 3. EVOLUTION TO A3C

Bellman equation is one of the most important and fundamental equation in RL. This equation is used in Q-Learning to update Q-values. The equation can be written as [6]:

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)]$$

Update Value

New Q value for that state and that action | Current Q value | Learning Rate | Reward for taking that action at that state | Discount rate | Maximum expected future reward given the new s' and all possible actions at that new state

**Equation 2:** Bellman Equation[6]

This equation is used to implement **Deep Q-Learning** which is basically Q-learning with neural networks. This equation finds the NewQ (s, a) by adding last or current Q (s, a), to alpha (learning rate) time's the temporal distance (update value).

Deep Q-learning helps to train our agent in various types of environments, however the entire concept of neural networks is based on our ability to mimic the human brain. We need an algorithm which acts more like humans and the only difference between Deep Q-Learning and human learning is our ability to see our environment with our eyes, our ability to take out information and make decisions based on what we see. This gave rise to **Deep Convolution Q-Learning** which is similar to Deep Q-Learning but is implemented using convoluted neural networks. Basic architecture of convoluted neural network is given below [7]:

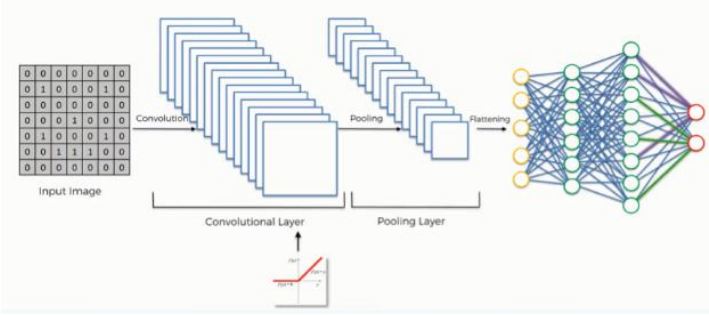


***Figure 6:*** *Convoluted Neural Network [7]*

The input is given as a matrix and various feature detectors are applied to this input matrix (Convolution) to give convolution layer. Each matrix in this convolution layer identifies a certain feature. Then max Pooling is applied to this convolutional layer to give a Pooling Layer. The output of convolution and pooling is flattened into a numeric vector which is inputted into the neural network.

Now that our agent has vision, we can talk about if we can further increase the accuracy of our learnings. The answer to this is A3C.

## 4. IMPLEMENTATION A3C: BREAKOUT

### 4.1 A3C Brain(Model.py)

```
class ActorCritic(torch.nn.Module):

    def __init__(self, num_inputs, action_space):
        super(ActorCritic, self).__init__()
        self.conv1 = nn.Conv2d(num_inputs, 32, 3, stride=2, padd
        self.conv2 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.conv3 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.conv4 = nn.Conv2d(32, 32, 3, stride=2, padding=1)
        self.lstm = nn.LSTMCell(32 * 3 * 3, 256)
        num_outputs = action_space.n
        self.critic_linear = nn.Linear(256, 1)
        self.actor_linear = nn.Linear(256, num_outputs)
        self.apply(weights_init)
        self.actor_linear.weight.data = normalized_columns_initi
        self.actor_linear.bias.data.fill_(0)
        self.critic_linear.weight.data = normalized_columns_init
        self.critic_linear.bias.data.fill_(0)
        self.lstm.bias_ih.data.fill_(0)
        self.lstm.bias_hh.data.fill_(0)
        self.train()

    def forward(self, inputs):
        inputs, (hx, cx) = inputs
        x = F.elu(self.conv1(inputs))
        x = F.elu(self.conv2(x))
        x = F.elu(self.conv3(x))
        x = F.elu(self.conv4(x))
        x = x.view(-1, 32 * 3 * 3)
        hx, cx = self.lstm(x, (hx, cx))
        x = hx
        return self.critic_linear(x), self.actor_linear(x), (hx,
```

### 4.2 Result: A3C vs DQN vs DDQN

| | |
|---|---|
| A3C | 744.8 |
| DQN | 303.9 |
| DDQN | 368.9 |

Raw scores 30 minutes' emulator time. Scores for DQN and DDQN taken from (Nair et al., 2015) [4] & (VanHasseltetal.,2015) [3] respectively.
We can see that scores for A3C far exceeds that of DQN and DDQN algorithms

## 5. IMPROVEMENTS

In this section we will talk about the ways in which we can improve the A3C algorithm even further. We will introduce three ideas which can be used to improve the functioning of A3C before that we need to discuss few terms.

### 5.1 Actions-Selection

Action selection policies are applied after the neural network yields the Q values for each action. These policies help to pick the action which the agent should take based on its Q values.

1. **ε-greedy**

This policy picks the action which has the maximum Q value ε % of times and picks other actions (1- ε) % of times.

2. **ε-Soft (1- ε)**

This policy picks the action which has maximum Q value (1- ε) % of times and other actions ε% of times.

3. **Softmax**

**Equation 3:** Softmax Function[19]

$$\sigma(\mathbf{z})_i = \frac{e^{\beta z_i}}{\sum_{j=1}^{K} e^{\beta z_j}}$$

Softmax function takes as input a vector of k real numbers and normalizes it such that each component is in the interval (0,1) and these components add up to one so that they can be treated as probabilities.

### 5.2 Entropy

Entropy is a measure of disorder in a dataset and it is used to skew the values used by the optimizer in a neural network to encourage heterogeneity in assigning probability to the actions. Entropy helps our agent to explore the environment by giving each action some probability to be chosen. Our goal is to maximize entropy so that the probability distribution best represents the current state of knowledge. Higher the entropy more the probabilities will be spread out.

A3C algorithm uses softmax function as is action selection policy. In a way, the policy is still a greedy approach to selecting actions as action with highest Q value will have the highest probability and therefore be chosen maximum number of times and as a result exploitation dominates over exploration. We propose an approach such that we can balance exploitation with exploration. One possible approach is given(James B. Holliday 2018) [13], in which it is proposed to explore more during the middle of the episode and exploit during the beginning and the end.

Our approach does not put any restrictions on when to choose exploitation or exploration. It introduces a random function into "Follow then Forage Exploration" [13]. This random function outputs the value 1 or 0 and is invoked every time the agent is in a new state. When the function outputs 1 the agent always exploit i.e. it chooses action according to the policy however if function outputs 0 the agent always explores i.e. it chooses actions from a uniform probability distribution of actions.

Pseudo code-1 (A3C algorithm) [1] [13]

```
//Assume global shared parameter vectors θ and θv and global shared counter T = 0
//Assume thread-specific parameters vectors θ0 and θ0 v
//Assume thread-specific variables m,n,o and p
Initialize m,n,o and p←0
Initialize thread step counter t←1
repeat
     Reset gradients: dθ←0 and dθv ←0.
     Synchronize thread-specific parameters θ0 =θ and θ0 v =θv
     tstart =t
     Get state st
     repeat
             Perform REE
             Receive reward rt and new state st+1
             t←t+1
             T ←T +1
     until terminal st ort-tstart ==tmax
     if terminal st then
             Perform FFE Update
     endif
     R ={   0 for terminal st
        {   V(st,θ0 v) for non-terminal st
     for i∈{t-1,...,tstart}do
             R←ri+γR
             Accumulate gradients θ0 and θ0 v
     endfor
     Perform asynchronous update ofθ0 and θ0 v
 untilT >Tma
```

e defines the number of iteration the user wants the A3C algorithm to perform. Ecurrent is the current iteration number. For iteration number between 50% and 75% we would perform REE. This enables the algorithm to initialize the Q values for all the states first and then balance between exploration and exploitation and then in the end refines the Q values of all the states.

Pseudo Code-2 (REE algorithm)

```
1: e= number of iterations(by user)
2: if ecurrent > e(0.5).floor() ecurrent < e(0.75).floor() then
3:      a=random(0,1)
4:      if a==1 then
5:              Choose at according to policyπ(at|st;θ0)
6:       else
7:              Choose at from a uniform distribution in a
8: else
9:      return
```

### 5.3 Proposed Implementation(A3C-REE)

While training we choose the action based on REE random function

```
prob = F.softmax(action_values)
log_prob = F.log_softmax(action_values)
entropy = -(log_prob * prob).sum(1)
entropies.append(entropy)
action = REERandom(prob,iterations,current_iteration)
```

While testing we choose the action with best probability

```
prob = F.softmax(action_value)
action = prob.max(1)[1].data.numpy()
```

## 6. CONCLUSIONS

Asynchronous Advantage Actor Critic is one of the finest artificial intelligence algorithms and it outperforms algorithms like DQN and DDQN. However, A3C depends heavily on entropy and we show that it might not be the best idea to do so. We introduce A3C-REE algorithm which takes entropy out of the equation for a certain number of iterations to give the agent equal probability (50%) to explore and exploit in all the states of the environment. We saw how A3C-REE algorithm can be implemented.

For future work, we plan to compare performance of A3C with A3C-REE and introduce phenomenon's like Eligibility Tracing into our algorithm. We also plan to explore different ways in which we can use entropy to improve A3C algorithm. We can approach this idea by controlling the entropy of the algorithm. Suppose during an episode (agent goes from start to end) we decrease the value of entropy such the probability of each action is approx. equal and gives our agent a chance to explore more.

## References

[1] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. (2016)." Asynchronous Methods for Deep Reinforcement Learning". arXiv:1602.01783v2 [cs.LG].
[2] Adam, S., Busoniu, L., & Babuska, R. (2012). Experience Replay for Real-Time Reinforcement Learning Control. *IEEE Transactions On Systems, Man, And Cybernetics, Part C (Applications And Reviews)*, *42*(2), 201-212. doi: 10.1109/tsmcc.2011.2106494
[3] Van Hasselt, Hado, Guez, Arthur, and Silver, David. Deep reinforcement learning with double q-learning. arXiv preprint arXiv:1509.06461, 2015.
[4] Nair et al, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, Maria, Alessandro De, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, Legg, Shane, Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David. Massively parallel methods for deep reinforcement learning. In ICML Deep Learning Workshop. 2015.
[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller.(2013)" Playing Atari with Deep Reinforcement Learning". arXiv:1312.5602 [cs.LG]

[6] Maheshwari, S. (2020). Implementing the A3C Algorithm to train an Agent to play Breakout!. Retrieved 11 February 2020, from https://medium.com/@shagunm1210/implementing-the-a3c-algorithm-to-train-an-agent-to-play-breakout-c0b5ce3b3405
[7] SuperDataScience. (2020). Retrieved 11 February 2020, from https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-summary/
[8] Asynchronous Advantage Actor Critic (A3C) algorithm - GeeksforGeeks. (2020). Retrieved 12 February 2020, from https://www.geeksforgeeks.org/asynchronous-advantage-actor-critic-a3c-algorithm/
[9] Simple Reinforcement Learning with Tensorflow Part 8: Asynchronous Actor-Critic Agents (A3C). (2020). Retrieved 12 February 2020, from https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-8-asynchronous-actor-critic-agents-a3c-c88f72a5e9f2
[10] Van Hasselt, Hado, Guez, Arthur, and David. Deep reinforcement learning with double q-learning. arXiv preprint arXiv:1509.06461, 2015
[11] Wang,Z.,deFreitas,N.,andLanctot,M. DuelingNetwork Architectures for Deep Reinforcement Learning. ArXiv e-prints, November 2015.
[12] Majd, M., & Safabakhsh, R. (2019). Correlational Convolutional LSTM for human action recognition. *Neurocomputing*. doi: 10.1016/j.neucom.2018.10.095
[13] James Bradley Holliday (2018). "Improving Asynchronous Advantage Actor Critic with a More Intelligent Exploration Strategy" University of Arkansas, Fayetteville. Retrieved 11 February 2020
[14] Conti, E., Madhavan, V., Petroski Such, F., Lehman, J., O. Stanley, K., & Clune, J. (2020). mproving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. *Papers Nips*.
[15] Chen S., Zhang XF., Wu JJ., Liu D. (2018) Averaged-A3C for Asynchronous Deep Reinforcement Learning. In: Cheng L., Leung A., Ozawa S. (eds) Neural Information Processing. ICONIP 2018. Lecture Notes in Computer Science, vol 11303. Springer, Cham"
[16] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan R. Salakhutdinov (2012)." Improving neural networks by preventing co-adaptation of feature detectors" arXiv:1207.0580
[17] Andrew G. Howard(2013)." Some Improvements on Deep Convolutional Neural Network Based Image Classification". arXiv:1312.5402
[18] Janisch, J. (2020). Let's make an A3C: Implementation. Retrieved 12 February 2020, from https://jaromiru.com/2017/03/26/lets-make-an-a3c-implementation/.
[19] Softmax function. (2020). Retrieved 12 February 2020, from https://en.wikipedia.org/wiki/Softmax_function.