AWAC: Accelerating Online Reinforcement Learning with Offline Datasets

Sushanth Doddabasappa*, Steeve Daniel*, Mahum Pervez*, Siddharth Jain*

Arizona State University, Tempe

***Abstract*—Existing Reinforcement Learning faces issues in real-world robotic applications due to the collection of data online from scratch every time a new policy is learned. To improve its performance, previously collected data from off-policy updates, prior runs, or expert demonstrations can be utilized to accelerate online learning. The proposed technique can effectively solve challenging real-world robotic problems. Our work aims to duplicate the results of the Advantage Weighted Actor Critic algorithm proposed by [1] to integrate prior data experience with online learning efficiently in performing robotic control. The algorithm is applied to three dexterous robot manipulation tasks of object reorientation, realignment, and repositioning. The results obtained were compared with the baseline paper in terms of average returns and success rates. A comparison analysis of our results with the baseline showed satisfactory performance.**

***Keywords*—Reinforcement Learning, Advantage Weighted Actor Critic, Dexterous Manipulation, offline learning, online fine tuning, real-world robotic applications.**

I. INTRODUCTION

ADVANCEMENTS in the field of Reinforcement Learning (RL) have encouraged its usage in complex robotic real-world applications [1]. Existing RL algorithms, however, are based on data collected online from scratch each time a new policy is learned. This attribute makes it impractical in robotic applications where large data collection is significantly helpful in accelerating learning. A more practical approach for real-world robotics is to include prior data sets in the learning process along with data collected online for further behavior improvement [1].

Prior data sets in reinforcement learning constitute state-action trajectories with corresponding rewards. The data can be collected from expert demonstrations [2], off-policy data [3], or prior runs of RL. However, this data mostly suffers from the problem of sub-optimality resulting in learning undesired bias by the agent [4].

To address this issue, we have presented an RL algorithm that utilizes the benefits of offline learning from off-policy data sets but keeps on continuously improving with data collection while online learning. The proposed method, Advantage Weighted Actor Critic (AWAC) is derived from the soft actor critic algorithm presented in [5]. It utilizes dynamic programming to train a critic network and a supervised learning style to train an implicitly constrained actor to solve complex robotic tasks. Dynamic programming enables sample-efficient learning while supervised actor update implicitly imposed with constraints limits the effects of distribution shift while training from offline data along with avoiding overly conservative updates [6,7]. The approach of accelerated learning by AWAC is illustrated in Fig. 1.

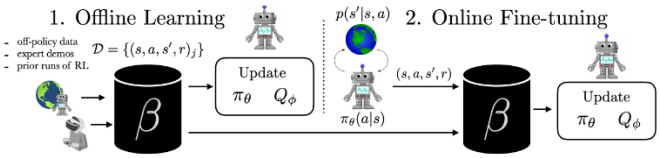

Fig. 1. The problem of accelerating online RL with offline datasets. In (a), the robot learns a policy entirely from an offline dataset. In (b), the robot gets to interact with the world and collect on-policy samples to improve the policy beyond what it could learn offline.

The algorithm is evaluated on a set of three dexterous manipulation robotic tasks: Pen reorientation, opening a door, and relocating an object. Our implementation has been able to achieve similar trends in learning the policies efficiently as obtained in the baseline article [1]. The performance evaluation is made based on a comparative analysis of average returns and success rates for each of these tasks.

The paper is articulated as follows: Section II. discusses the design of the AWAC algorithm whose implementation and simulation are given in Section III. Section IV discusses the results obtained for our implementation. The challenges faced, and solutions adopted, are tabulated in Section V. Section VI concludes with a discussion and Section VII lists the references.

II. ADVANTAGE WEIGHTED ACTOR CRITIC ALGORITHM

AWAC consists of two main steps: policy evaluation performed at the critic network and policy improvement on the actor network.

A. POLICY EVALUATION (CRITIC NETWORK)

The target of our RL algorithm is to maximize the discounted return $J(\pi) = E_{p_\pi(\tau)}[Ro]$ where $p_\pi(\tau)$ is the optimal policy distribution. Using gradient descent, the optimal policy can be learned. However, the learned policy can be ineffective due to estimation errors. Thus, in this study, the advantage, given as $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ (where V^π is the value function and Q^π is the action-value function as per policy π) is used for policy improvement. The critic estimates the target $Q^\pi(s,a)$ for a current policy π. After applying the bellman operator β^π, the action-value function $Q(s,a)$ for state s and action a is given:

sdoddab2@asu.edu, sdanie20@asu.edu, mpervez@asu.edu, sjain238@asu.edu
*Equal Contribution

as:

$$\beta^{\pi}Q(s,a) = r(s,a) + \gamma E_{p(s'|s,a)}[E_{\pi(a'|s')}[Q^{\pi}(s',a')]]$$

Iterating through the above equation with the update given as $Q^{k+1} = \beta^{\pi}Q^{k}$, the result approaches the targeted distribution $Q^{\pi}(s,a)$. The critic network is learned by a deep neural network with critic network parameter φ given as:

$$\varphi_k = \arg\min_{\varphi} E_D[Q_{\varphi}(s,a) - y)^2] \quad (1)$$

$$y = r(s,a) + \gamma E_{s',a'}[Q_{\varphi_{k-1}}(s',a')] \quad (2)$$

Where, $Q_{\varphi}(s,a)$ is the estimated Q value at state *s* and action a at the current time step and $Q_{\varphi_{k-1}}(s',a')$ is the estimated Q value for the next state s' and action a'at the past step *k-1*. The expectation is taken w.r.t data $D = \{(s,a,s',r)_j\}$, $r(s,a)$ is the stage reward at state s with action taken a and γ is the discount factor.

B. POLICY IMPROVEMENT (ACTOR NETWORK)

At the policy improvement stage, the actor network with parameters θ_k predicts the distribution $\pi_k$based on the current estimate $Q^{\pi}$from the critic.

To avoid bootstrapping errors, and to ensure stable data distribution modeling, constraints are added implicitly on the policy improvement update such that:

$$\pi_{k+1} = \arg\max_{\pi} E_{a\sim\pi(.|s)}[A^{\pi_k}(s,a)] \quad (3)$$

$$s.t.\, D_{KL}(\pi(.|s)||\pi_{\beta}(.|s)) \leq \epsilon \quad (4)$$

$\pi_{\beta}(.|s)$ is the distribution of past policies from the prior RL runs obtained from data collection through the replay buffer β. ϵ is the probability of taking exploratory moves. Computing the optimal solution for Eq. (3) and projecting the policy onto it, results in the following actor update:

$$\theta_{k+1} = \arg\max_{\theta} E_{s,a\sim\beta}[\log \pi_{\theta}(a|s) \exp\left(\frac{1}{\lambda}A^{\pi_k}(s,a)\right)] \quad (5)$$

where lambda is the Lagrange multiplier taken as $\lambda = 0.3$. The actor update is based on weighted maximum likelihood where the state action pairs are weighted in each successive iteration based on the advantage received from the trained critic. The actor does not learn explicitly from any parametric behavioral model. It rather simply samples states and actions from the replay buffers.

Fig. 2. Shows a block diagram for AWAC based algorithm for the real-time control in dexterous manipulation with a robotic hand. The states from the environment are used to generate value iteration which is utilized by the critic network for the estimation of discounted rewards using off-policy updates. The estimates are utilized by the actor to predict the policy distribution using an on-policy algorithm where actions are applied based on the system's current state.

C. ALGORITHM SUMMARY

The AWAC algorithm's pseudocode is summarized in Fig. 3. The critic and actor networks are implemented using deep neural networks with Stochastic Gradient Descent (SGD) updates applied to Eq. 1 And Eq. 5 respectively. For data efficiency, AWAC uses an off-policy based critic via Temporal Difference (TD) bootstrapping, while implementing a constrained actor to mitigate bootstrapping errors. The conservative actor updates are avoided by implicit constraints without any explicit behavior modelling of the policy.

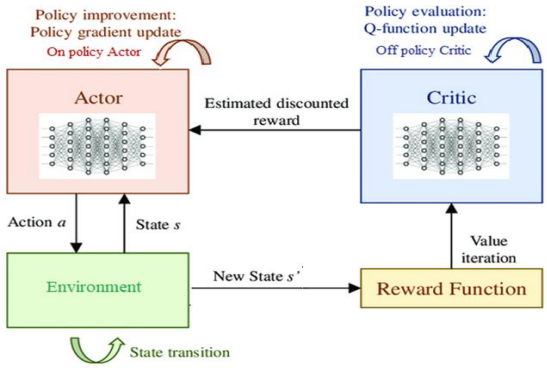

Fig. 2. Block diagram for Advantage Weighted Actor Critic (AWAC) algorithm implementation [1]

III. IMPLEMENTATION & SIMULATION

We evaluated the AWAC algorithm in the Dexterous Manipulation environment. Detailed descriptions of the environment and each task is as follows:

A. Dexterous Manipulation with a Robotic Hand.

This is a set of three different tasks for controlling a 5-fingered robotic hand as illustrated in Fig. 4. All three tasks [4] require the robotic hand to carefully manipulate an object accurately.

(a) **Pen Reorientation**: This task requires the robotic hand to reorient a pen into the desired orientation. The state space dimension is 45 and the action space dimension is 24. The robotic hand rotates the pen from an initial angle, x_o to a final angle d_o. The reward function is given by the equation $r = 1_{|x_o.d_o|\leq 0.95} - 1$. And the cumulative discounted reward is given by $R_t = \sum_{i=t}^{T} \gamma^i r(s_i, a_i)$.

(b) **Door Opening by twisting a latch**: In this task, the robotic hand is required to reach and twist a latch to open the door. The state space dimension is 39 and the action space dimension is 28. The reward function is given by the equation $r = 1_{d>1.4} - 1$

(c) **Object Relocation:** This task relocates an object to the desired position. The state space dimension is 39 and the action space dimension is 30. The initial position of the object is given by $x_p$and the desired position by d_pThe reward function is given by: $r = 1_{|x_p - d_p|\leq 0.1} - 1$

Algorithm 1 Advantage Weighted Actor Critic (AWAC)

Dataset $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)_j\}$
Initialize buffer $\beta = \mathcal{D}$
Initialize π_θ, Q_ϕ
for iteration $i = 1, 2, \ldots$ **do**
  Sample batch $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r) \sim \beta$
  $y = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}', \mathbf{a}'}[Q_{\phi_{k-1}}(\mathbf{s}', \mathbf{a}')]$
  $\phi \leftarrow \arg\min_\phi \ \mathbb{E}_{\mathcal{D}}[(Q_\phi(\mathbf{s}, \mathbf{a}) - y)^2]$
  $\theta \leftarrow \arg\max_\theta \ \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\beta}\left[\log \pi_\theta(\mathbf{a}|\mathbf{s}) \exp\left(\frac{1}{\lambda}A^{\pi_k}(\mathbf{s}, \mathbf{a})\right)\right]$
  if $i >$ num_offline_steps **then**
    $\tau_1, \ldots, \tau_K \sim p_{\pi_\theta}(\tau)$
    $\beta \leftarrow \beta \cup \{\tau_1, \ldots, \tau_K\}$
  end if
end for

Fig. 3. AWAC algorithm pseudocode [1]

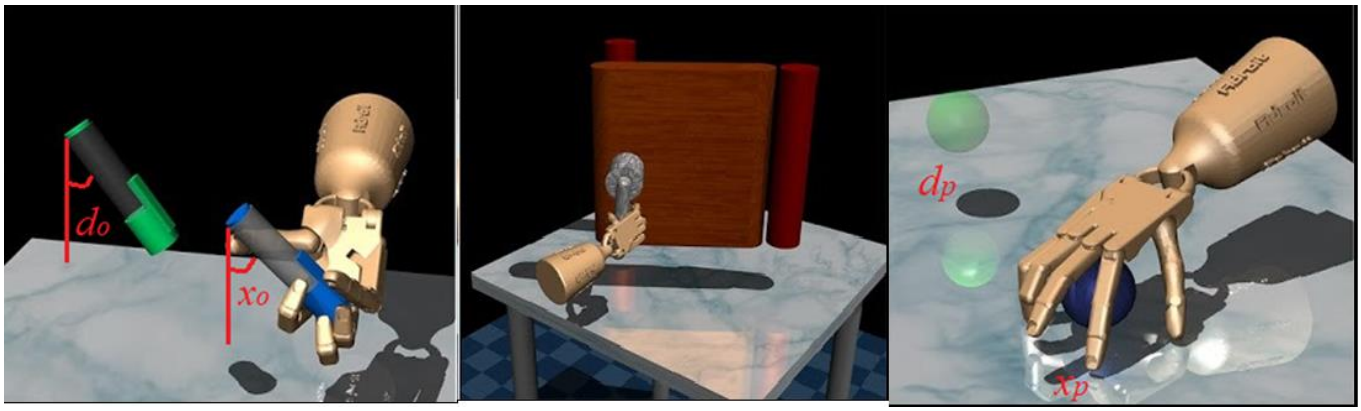

Fig. 4. Dexterous Manipulation Tasks with a Robotic Hand [1] (a) shows controlling a 5-fingered robotic hand to pick up a pen and spin it into a given orientation (Task 1 GIF) (b) shows task of opening a door, which requires first twisting a latch (Task 2 GIF) whereas (c) shows controlling a 5-fingered robotic hand to reposition an object from initial to targeted spot (Task 3 GIF)

B. ACTOR, CRITIC DEEP NEURAL NETWORK ARCHITECTURE & HYPERPARAMETERS

AWAC is implemented using the soft actor critic algorithm given in [5]. The critic network is based on a double Q-learning strategy given in [8] estimating Q values (trainer network) and V values (target network) from states and actions provided at the input layer. The actor network predicts policy distributions π and π_β by taking states at the input layer. All the networks contain two hidden layers with 256 neurons in each hidden layer. The architecture of critic and actor networks for all the dexterous manipulation tasks is depicted in Fig. 5 & Fig.6 respectively.

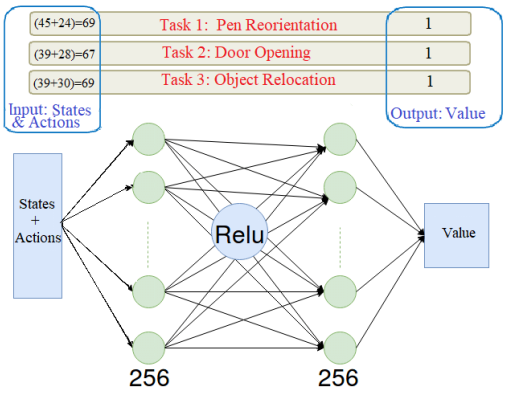

Fig. 5. Critic Network Architecture to estimate Q values/ V values [5,8]

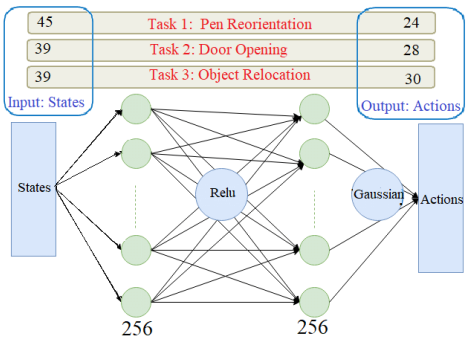

Fig. 6. Actor Network Architecture incorporating offline and online learning to estimate policy functions [5]

Hyperparameters in Deep RL are critical for training successful agents. The base hyperparameters are given in Table 1. The optimization is performed using Adam [9] and the hidden layer activation function is ReLU.
The policy learning rate is set as 3×10^{-4} and discount factor as 0.99.

Table. 1. Hyperparameters used for the RL experiment [1]

Hyper-parameter	Value
Training Batches Per Timestep	1
Exploration Noise	None (stochastic policy)
RL Batch Size	1024
Discount Factor	0.99
Reward Scaling	1
Replay Buffer Size	1000000
Number of pretraining steps	25000
Policy Hidden Sizes	[256, 256, 256, 256]
Policy Hidden Activation	ReLU
Policy Weight Decay	10^{-4}
Policy Learning Rate	3×10^{-4}
Q Hidden Sizes	[256, 256, 256, 256]
Q Hidden Activation	ReLU
Q Weight Decay	0
Q Learning Rate	3×10^{-4}
Target Network τ	5×10^{-3}
Optimizer	Adam

C. SYSTEM SETUP, DATA SET & TRAINING PROCESS

To ensure flexibility while experimenting & training, and to accelerate development, a local computing machine was utilized. Anaconda environment was set up on it to run simulations and perform training jobs. The computing machine's hardware specifications are as follows:

- CUDA enabled GTX1660 Ti NVIDIA GPU with 6GB of super-fast GDDR6 VRAM
- Intel i7 11th gen 8 core CPU
- 16 GB RAM

The dexterous manipulation demonstration dataset for the three robotic hand tasks, provided by the authors, was utilized for the replication of AWAC results.

Three separate training jobs were performed, one for each of the dexterous manipulation tasks. In the above-mentioned machine, an average training time of 6 hours was observed for each training job. For each training task, the number of seeds was set to 1 to improve the training time. Each task's training job was executed for 500 episodes, with 1000 timesteps per episode.

IV. RESULTS & ANALYSIS

This section describes the training details and results of each manipulation robotic task at hand.

The training dataset for all three tasks consisted of five hundred trajectories constructed from a behavioral cloned policy [1]. During training for each task, the hyperparameters are listed in Table. 1 and neural network configurations showcased in Fig. 5 and Fig.6 have been utilized**.

A. Pen Reorientation Task Training & Results:

For the reorientation task two graphs were generated during training: average return vs timesteps, Fig. 7(a), and success rate vs timesteps, Fig. 7(b). We compared the success rate graph with the baseline paper (Fig. 7(c)) to observe a high correlation. From Fig. 7(a), the saturation of average return at 200,000 timesteps indicates the training's convergence to an optimal policy.

Table.2 summarizes the performance parameters associated with the reorientation training job. Fig. 8 is a screenshot from the simulation video that showcases the reorientation task being solved. We have been able to successfully achieve the desired success rate for this task.

B. Door Opening Task Training & Results

For the door opening task the two graphs of average return vs timesteps, Fig. 9(a) and success rate vs timesteps, Fig. 9(b) were generated.

For the success rate graph (Fig. 9(b)), we see noticeable correlations with the baseline graph, Fig. 9(c), such as the dip at around 100k timesteps. However, the success rate in our implementation saturates in the vicinity of 0.6.

Table. 3 summarizes the performance parameters associated with the door opening task's training job. Fig. 10 is a screenshot from the simulation video that showcases the door opening task being solved. The success rate we obtained for the task is satisfactory.

C. Object Relocation Task Training & Results

For the object relocation task success rate vs timesteps, Figure 11(a) was generated.

The object relocation task is the most difficult one among the three. This is apparent from Figure 11(b) which showcases the author's training plot [1] for over 4 million timesteps. This is 8 times more than our 500,000 timesteps implementation. Notably, comparing within the window of the first 500,000 timesteps, we observe a high correlation between our implementation and the authors'. We were able to successfully achieve the desired success rate for this task in this window.

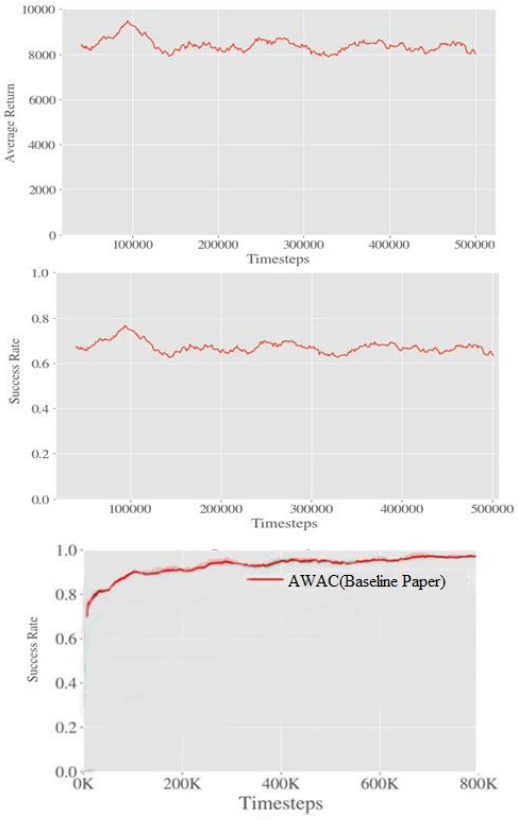

Fig. 7. Simulation Results of Pen Reorientation task (a) Average Return versus iterations for our implementation of AWAC (b) Success Rate versus iterations for our implementation of AWAC (c) Success Rate versus iterations from baseline paper [1]

Table 2. Performance parameters observed for average returns versus time steps for Pen Reorientation task

Performance Parameters	Values
Average of all Returns	8383.55
Deviation Average Return	2880
Number of episodes	500
Time steps per episode	1000

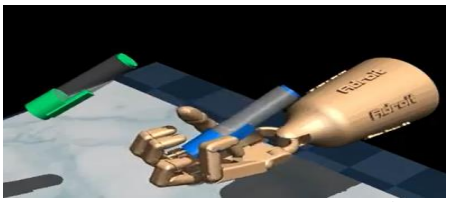

Fig. 8. Pen Reorientation Task - Screenshot from the simulation video

**The results we produced for the three manipulation tasks was with number of seeds = 1, while baseline paper [1] have used number of seeds = 3.

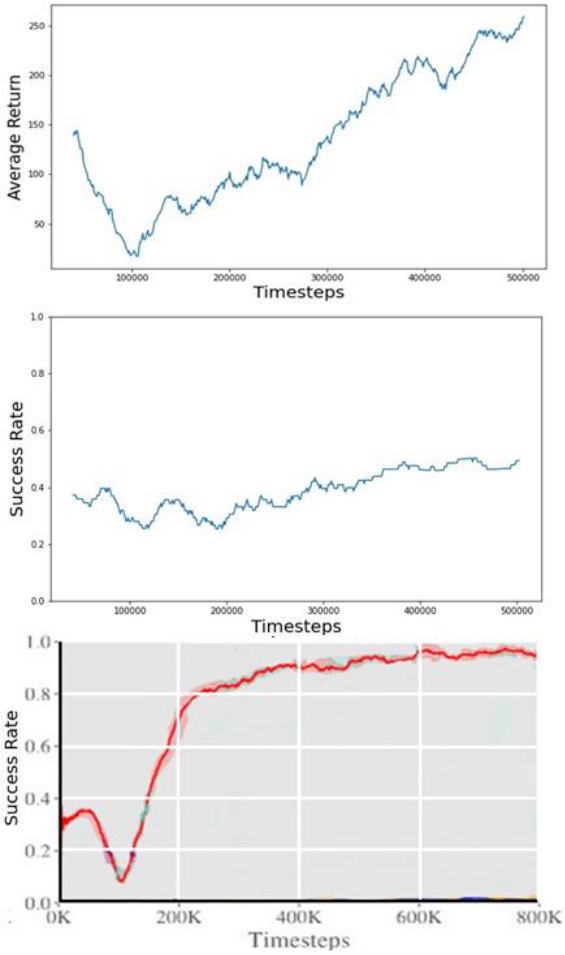

Fig. 9. Simulation Results of Door Opening by twisting a latch task (a) Average Return versus iterations for our implementation of AWAC (b) Success Rate versus iterations for our implementation of AWAC (c) Success Rate versus iterations from baseline paper [1]

Table 3. Performance parameters observed for average returns versus time steps for Door Opening task

Performance Parameters	Values
Average of all Returns	140.4
Deviation Average Return	108.10
Number of episodes	500
Time steps per episode	1000

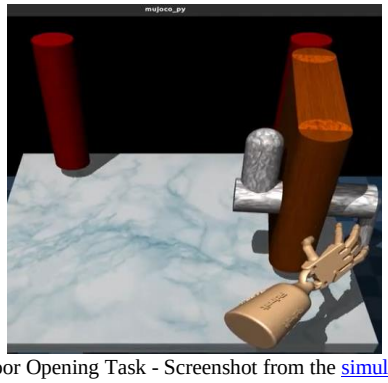

Fig. 10. Door Opening Task - Screenshot from the simulation video

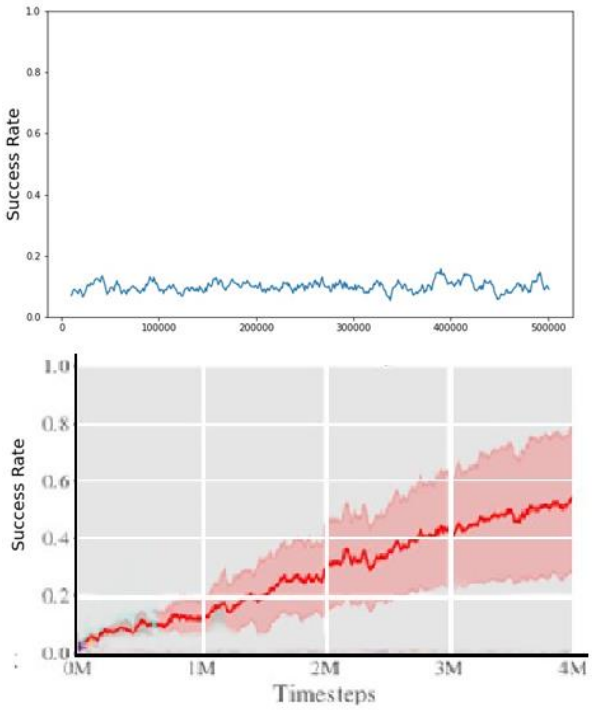

Fig. 11. Simulation Results of Object Reorientation Task (a) Success Rate versus iterations for our implementation of AWAC (b) Success Rate versus iterations from baseline paper [1]

V. CHALLENGES FACED & RESOLUTIONS

This section mentions the challenges we faced during the implementation of the given project tabulated in Table 4.

VI. CONCLUSION & FUTURE WORK

In this study, the limitations of existing RL strategies were addressed by proposing a novel technique of AWAC combining the benefits of offline training with online fine tuning in various practical robotic applications. We simulated our technique on three dexterous manipulation tasks and compared our results with the baseline article for satisfactory performance. The tasks of reorientation and realignment gave satisfactory results in terms of average returns and success rates. However, the reallocation task's duplication faced issues due to limited computational resources.

While training the agent, we require a system with an efficient graphics card and a powerful Central Processing Unit (CPU). Due to our system limitations, we were not able to train the agent for more than 500 episodes.

For future work, we can compare our results using AWAC with the other potential on-policy/ off-policy RL algorithms for analysis and performance comparison in more challenging real-world robotic applications. For overcoming the limitations of the system, we aim to utilize the on-campus computational resources for fair comparison analysis.

Table 4. Challenges faced in the implementation of AWAC and the solutions adopted

No.#	Challenges Faced	Solutions adopted
1.	The training script was designed with number of seeds as 3 to run three training parallel experiments on 3 Amazon Web Services ec2 machines.	The training script was restructured to accommodate local training and we were able to reproduce results with just one seed.
2.	The requirements listed for the environment setup were incomplete.	By parsing the logs, the missing packages were installed. We were cautious while installing these packages and ensured that they don't break the existing conda environment by using the command: pip install –no-dependencies <package_name>
3.	Deprecated packages such as path and multiworld raised "module not found" errors.	Path package was replaced with pathlib in all such instances. mutiworld was installed from the source.

VII. REFERENCES

[1] Nair, Ashvin, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. "Awac: Accelerating online reinforcement learning with offline datasets." arXiv preprint arXiv:2006.09359 (2020).

[2] Atkeson, Christopher G., and Stefan Schaal. "Robot learning from demonstration." In ICML, vol. 97, pp. 12-20. 1997.

[3] Gao, Yang, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. "Reinforcement learning from imperfect demonstrations." arXiv preprint arXiv:1802.05313 (2018).

[4] Rajeswaran, Aravind, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations." arXiv preprint arXiv:1709.10087 (2017).

[5] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." In International conference on machine learning, pp. 1861-1870. PMLR, 2018.

[6] Fujimoto, Scott, David Meger, and Doina Precup. "Off-policy deep reinforcement learning without exploration." In International conference on machine learning, pp. 2052-2062. PMLR, 2019.

[7] Kumar, Aviral, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. "Stabilizing off-policy q-learning via bootstrapping error reduction." Advances in Neural Information Processing Systems 32 (2019).

[8] Fujimoto, Scott, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In International conference on machine learning, pp. 1587-1596. PMLR, 2018.

[9] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

PROJECT CONTRIBUTIONS:

Sushanth Doddabasappa: Sushanth worked on the system setup for the simulation of the environment and training. Sushanth refactored the author's training script to support local training and worked on resolving dependency issues related to the training script. Sushanth developed a custom script to investigate and understand the input/output configurations and integrations of the policy neural network. Sushanth has contributed to the writing of the progress check and has assisted with revisions to the subsequent drafts. Sushanth has been instrumental in the developmental work for the processing of the output of the training results and for incorporating it into the report's results section. Sushanth has contributed to the writing of the final report.

Steve Daniel: Steeve worked majorly on running the trainings on his machine and creating records of the training progress. Steeve was also responsible for working with Sushant to fix the issues faced during system setup and making sure the training was run. Steeve was also responsible for recording the training results and simulations to be incorporated into the report. Steeve has contributed to the writing of the progress checks. Steeve has contributed to the writing of Abstract, Introduction and the Implementation sections of the final report.

Mahum Pervez: Mahum has become the subject matter expert for the theory behind each of the algorithms and advises the rest of the team on the theory of the algorithms. She helped the team in understanding and code the deep neural network architecture in the given project. She has been actively involved in writing progress reports, and compilation of the final report. She has written the AWAC architecture section, challenges faced and solutions adopted, and conclusion of the report and proofread the complete document for technical writing check.

Sidharth Jain: Siddharth has provided research, literature review and understanding of the paper and the algorithm. Siddharth has contributed to the writing of the progress check and has assisted with revisions to the subsequent drafts. Siddharth has also helped with the understanding of the kinematics and dynamics part of the robotic hand by defining the different spaces. Siddharth has contributed to the writing of the final report by working on the hyperparameters, implementation and discussion section of the report.