IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received February 16, 2021, accepted March 2, 2021, date of publication March 12, 2021, date of current version March 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3065710

A Hybrid Multi-Task Learning Approach for Optimizing Deep Reinforcement Learning Agents

NELSON VITHAYATHIL VARGHESE AND QUSAY H. MAHMOUD, (Senior Member, IEEE)
Department of Electrical, Computer and Software Engineering, Ontario Tech University, Oshawa, ON L1G 0C5, Canada

Corresponding author: Nelson Vithayathil Varghese (nelson.vithayathilvarghese@ontariotechu.net)

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

ABSTRACT Driven by recent technological advancements within the field of artificial intelligence (AI), deep learning (DL) has been emerged as a promising representation learning technique across different machine learning (ML) classes, especially within the reinforcement learning (RL) arena. This new direction has given rise to the evolution of a new technological domain named deep reinforcement learning (DRL) that combines the high representational learning capabilities of DL with existing RL methods. Performance optimization achieved by RL-based intelligent agents designed with model-free-based approaches was majorly limited to systems with RL algorithms focused on learning a single task. The aforementioned approach was found to be quite data inefficient, whenever DRL agents needed to interact with more complex, data-rich environments. This is primarily due to the limited applicability of DRL algorithms to many scenarios across related tasks from the same distribution. One of the possible approaches to mitigate this issue is by adopting the method of multi-task learning. The objective of this research paper is to present a hybrid multi-task learning-oriented approach for the optimization of DRL agents operating within different but semantically similar environments with related tasks. The proposed framework will be built with multiple, individual actor-critic models functioning within independent environments and transferring knowledge among themselves through a global network to optimize performance. The empirical results obtained by the hybrid multi-task learning model on OpenAI Gym based Atari 2600 video gaming environment demonstrates that the proposed model enhances the performance of the DRL agent relatively in the range of 15% to 20% margin.

INDEX TERMS Machine learning, deep reinforcement learning, neural networks, transfer learning, actor-critic, multi-task worker.

I. INTRODUCTION

Over the last few decades, the reinforcement learning domain has been well established its position as a vital topic within technological areas such as robotics and intelligent agents [1]. The core objective of RL is to address the problem of how the intelligent agents should explore their operating environment optimally, and thereby learn to take optimal actions to achieve the highest possible reward while in a given state [2]. Supported by recent advancements within the field of ML, the RL has been cemented its position as one of the major machine learning paradigms that deal with agent's behavior patterns while in an environment. In comparison to the performance of ML systems based out of contexts namely supervised learning, and unsupervised learning, the relative performance level of current RL agents was not optimal. This was majorly due to the limitations related to deriving the optimum policy out of the large state-action space linked with the environment of RL problems. At the same time, the inception of DL with its very high level of representational learning capability has given a new dimension to the field of reinforcement learning and led to the evolution of deep reinforcement learning. As a result of these advancements, DRL agents have been applied to various areas such as continuous action control,3D first-person environments, and gaming. Especially in the field of gaming, DRL agents are proven to be extremely successful and could surpass the human-level performance on classic video-games like Atari as well as board games such as chess and Go [3].

Despite the impressive results achieved with a single-task-based methodology, the RL agent is observed to be less efficient within operating environments that are more complex

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

and richer in data such as 3-dimensional environments. One of the possible directions to enhance the efficiency of the RL agent in such an environment is by the application of multi-task-based learning. With multi-task learning, a set of closely related tasks from the operating environment will be learned simultaneously by individual agents with the help of a DRL algorithm such as A3C (Asynchronous Advantage Actor-Critic) [4]. By the application of this approach, at regular intervals, neural network parameters of each of the individual agents will be shared with a global network. By combining the learning parameters of all the individual agents, the global network derives a new set of parameters, which will be shared back with all of the individual agents. The major aim of this methodology is to optimize the overall performance of the RL agent by transferring the learning, shared knowledge, among multiple related tasks running within the same environment. One of the most widely known and accepted multi-task learning methodologies within RL is parallel-based multi-task learning, in which a single RL agent masters a group of diverse tasks [5]. The core idea behind this approach mainly relies on the architecture used by the deep reinforcement learning model based on a single learner, often known as a critic, combined with different actors. Each of the individual actors generates their learning trajectories, which are a set of parameters, and further on sends them to the learner module, also called a critic module, either synchronously or asynchronously. Subsequently, each of the actors retrieves the latest set of policy parameters from the learner before initiating the next learning trajectory. With this approach, learnings from each of the individual tasks will be exchanged with every other agent within the environment, which in turn improves the overall learning momentum of the RL intelligent agent.

A. MOTIVATIONS AND CONTRIBUTIONS

The major motivation behind the proposed hybrid multi-task approach is to address some of the major challenges associated with the existing multi-task deep reinforcement learning (MTDRL) paradigm. Especially, attempting to address key challenges such as partial observability, effective exploration, and lastly the amount of training data and training time required to achieve an acceptable level of performance.

To this end, contributions of this paper are:

1) Design and development of a hybrid multi-task learning model to optimize the performance of DRL agents.
2) Evaluation of DRL agent's performance with hybrid multi-task learning model within the context of the aforementioned challenges.
3) The empirical analysis of the fluctuations in the DRL agent's performance when the degree of semantic similarity between the tasks trained together from multiple game environments (Atari2600).

The rest of this paper is organized as follows. Section II presents a brief background of reinforcement learning concepts, and Section III explains the various existing approaches

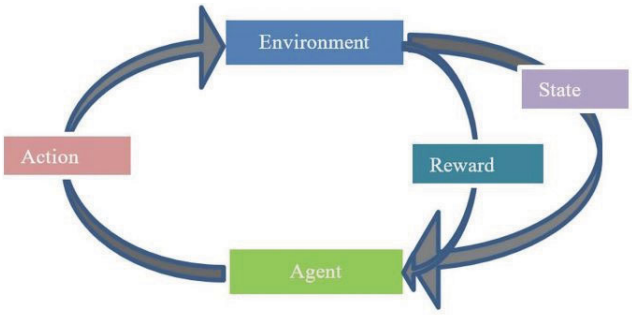

FIGURE 1. The ecosystem of reinforcement learning.

that are attempted on the multi-tasking front of DRL. Section IV discusses the various related work done within the same arena, with special focus given to three state-of-the-art solutions, namely Distral, IMPALA, and PopArt. Section V details the proposed hybrid multi-task model architecture while implementation details are covered in Section VI. Experiments conducted and results obtained with the hybrid multi-task model are presented in Section VII. Analysis of test results are discussed in Section VIII. Finally, Section IX concludes the paper and offers ideas for future work.

II. BACKGROUND

This section provides a brief background of the main aspects related to reinforcement learning. Table 1 indicates notations used within this paper to explain the concepts and equations.

Reinforcement learning is one of the ML paradigms related to sequential decision-making which deals with mapping situations to actions in a way that maximizes the associated reward. Within RL ecosystems, the learner, which is also known as an agent, is not explicitly instructed on which actions to take at each time step t, but instead, the RL agent must follow a trial-and-error method to identify which actions generate the most reward. A standard reinforcement learning setup consists of an agent situated within an environment E, where an agent will be interacting with the environment in discrete timesteps. At each of these timesteps t, the agent will be in a state S_t ($S_t \in S$) and will be performing a chosen action A_t ($A_t \in A$) within the environment E. Further on, the environment responds by updating the current state S_t to a follow-up state S_{t+1} with a new timestep $_{t+1}$ and also gives a reward $r(S_t, A_t) \in \mathcal{R}$ to the agent, indicating the reward value of performing an action in the preceding state S_t [1]. The below Fig. 1 represents the standard ecosystem for a reinforcement learning environment at any given timestep t. By performing multiple actions in a sequential learning manner in a sequence of associated states s, with related actions a, respective follow-up states s' and rewards r, several episodes of tuples of $<s, a, s', r>$ are generated. At any given state S_t, the goal of the agent is to determine a policy π that can create a state-to-action mapping that maximizes the accumulated reward over the lifetime of the agent for that particular state [6].

TABLE 1. List of notations.

Symbol	Name	Details
S	State-space	Set of all the states that an RL agent can go through
A	Action space	Set of all the actions that an RL agent can take
r	The reward at any timestep	Reward that an RL agent can receive (r ∈ $\mathscr{R}$)
S_t	State s at time t	State of an RL agent at any timestep t (s ∈ S)
S_{t+1}	State s at time t+1	State of an RL agent at any timestep t+1 (s ∈ S)
a	Action at any timestep	An individual action taken by an RL agent at any timestep (a ∈ A)
π	Policy	Mapping from state to action
π^*	Optimal policy	Mapping from state to action with maximum expected future reward
$\pi(a\|s)$	Stochastic policy	Policy parameterized by Θ
A_t	Action at time t	Action taken by an RL agent at timestep t
$\mathscr{R}$	Reward space	Set of all the rewards that an RL agent can receive
Rt	The reward at time t	The reward received by an RL agent at timestep t
E	Environment	The environment in which an RL agent operates
γ	Discount factor	Discount factor to calculate expected future reward
$V(s)$	State-value function	The expected return of a state s. $Vw(.)$ is a value function parameterized by w.
$Q(s, a)$	Action-value function	The expected return of a pair of state and action (s, a)
$A(s, a)$	Advantage function	$A(s, a) = Q(s, a) - V(s)$
α	Learning rate	Learning rate used for policy and value function parameter updates
$P(s', r\|s, a)$	Transition probability	Probability of getting into the next state s' with reward r by taking an action a
s,a,s',r	State,action,next state, reward tuplet	Represents the transition from state s to s' by taking an action a with a reward r
$V^{\pi}(s)$	Value of a state with policy π	Value of a state when policy π is followed
$Q^{\pi}(s, a)$	Value of (state, action) pair with policy π	Value of (state, action) pair when policy π is followed

At any point in the time t, the goal of the RL agent is to select the actions in such a way that it maximizes its *expected return*. The reward returned at any given time step t is the quantity that can be represented as

$$R_t = \sum_{\tau=0}^{\infty} \gamma^{\tau} r(S_{t+\tau}, A_{t+\tau})$$

where γ ∈(0,1) is the discount factor that multiplies the future expected reward and varies on the range of [0,1]. At any moment, the goal of the DRL agent is to maximize the expected return from each state S_t. The action value indicated by $Q^{\pi}(s, a) = \mathbb{E}[R_t|S_t = s, a]$ is the expected return for taking an action a in state s by following a policy π. Similarly, the optimal value function indicated by $Q*(s, a) = max_{\pi} Q^{\pi}(s, a)$ is the maximum action value for action a and state s that is achievable by any policy. Similarly, the value of any state s under policy π is defined by $V^{\pi}(s) = \mathbb{E}[R_t|S_t = s]$ which is simply the expected return for following the policy π from state s. The $Q(s, a)$ is often used as a measure of the value of the agent being in that particular state s and taking an action a to reach that state. The famous Bellman's equation mentioned below is used as a reference to calculate the $Q(s, a)$ for every action in every state that helps an agent to make decisions about its future moves.

$$Q'(s, a) = Q(s, a) + \alpha[R(s, a) + \gamma maxQ'(s', a') - Q(s, a)] \quad (1)$$

where $Q(s, a)$, α, $R(s, a)$, γ *and* $maxQ'(s', a')$ represents the current Q value, learning rate, reward for taking that action a in state s, and maximum expected future reward given the new state s' with all the possible action from that state.

In the case of value-based model-free reinforcement learning methods, the action-value function $Q(s, a)$ is often represented by using a function approximation method, such as a neural network. In such a case an approximate action-value function that parameterized with θ represented as $Q(s, a; \theta)$. The updates for the parameters are decided with the help of a suitable RL algorithm. In contrast to the aforementioned value-based methods, policy-based model-free RL methods directly parameterize the policy $\pi(a|s;\theta)$ and update the parameter by performing, typically approximate, gradient ascent on $\mathbb{E}[R_t]$.

III. MULTI-TASK DEEP REINFORCEMENT LEARNING

With the enormous growth that was happened within the AI and DL domains, DRL has been positioned itself as the state-of-the-art and de-facto choice for solving many of the benchmark tasks and real-world issues. As a direct result of this, methods for the optimization of DRL have caught a great level of interest and attention. The subsequent sections cover the information presented in related research efforts and related to methodologies and approaches designed to attain multi-task DRL.

A. TRANSFER LEARNING ORIENTED APPROACH

Before the inception of DL into the domain of RL, transfer learning was used as a major means for guiding the research work towards developing the multi-task learning algorithms with the RL domain. The major aspect of transfer learning (TL) is based on the concept of knowledge transfer happening between different source and target tasks that are closely related to each other. Subsequently, this knowledge transfer is intended to enhance the performance of the ML system's algorithm that would be employed to learn the target task. Within the context of RL, the notion of transfer is majorly concerned about coming up with different approaches and techniques to enable knowledge transfer originating from a group source tasks to a target task. This methodology is proven to be delivering impressive results whenever there is a high amount of similarity between the

source tasks as well as target tasks [7]. Especially when the level of similarity between two ends meets high, the process of knowledge transfer is observed to be happening quite smoothly such that this knowledge transferred in manner plays an important role in assisting the target task's algorithm to solve the tasks with a high level of efficiency. The major reason behind the success of the above scenario is due to the advantage provided by the positive knowledge transfer. This in turn enables the target task's algorithm to optimize its performance level in an easier way than spending more time for it to gain the same amount of knowledge by using more amount of the target task's data sample. Different TL mechanisms that are oriented on the aforementioned approach have been deployed within various RL algorithms based on single-agent [8].

Similar kinds of approaches also experimented with multi-agent systems, and agents operating within the same environment interact among them as a means for exchanging the knowledge gained from their respective actions [9]. Typically, the design of multi-agent systems uses a joint policy that is learned by agents from the source task, and further on same knowledge will be used to generate the initial policy for the agents to use within the context of the target task [10]. Knowledge transfer is made possible among the source and target tasks by adopting different transfer techniques like instance transfer, representation transfer, or parameter transfer. On all these approaches, algorithms that handle the knowledge transfer rely majorly on the prior knowledge gathered while attempting to solve the same kind of source tasks, and then leverage on that as a reference to bias the approach or learning to be followed on a new task [7].

B. LEARNING SHARED REPRESENTATIONS

Learning the shared representations for value functions is a technique that is quite analogous to transfer learning-oriented methodology. The base of this method lies in the neural network's function approximation capability and how it could be applied to the domain of RL [3]. The key success factor behind the usage of DNN with RL lies in DL algorithms' capability to distill data representation in a quite meaningful manner from the operating environment's high-dimensional input state space [11]. Due to this capability, RL's effective applicability into a wide spectrum of problems with complex real-life problems with high-dimensional input state space was made possible. Before the adoption of DL into RL space, such an attempt always demanded the application of feature engineering at a great level of effort [3]. The major success factors behind the application of DL into RL problems greatly demand two conditions – firstly the capability to derive a high-quality abstraction of the agent's operating environment, and secondly effective agent's role within its operating environment [12]. The major driving factor behind learning sharing representations lies in the idea that the group of different tasks that the agent needs to encounter in the operating environment will highly likely to have a great degree of a shared structure as well as an in-built redundancy [7]. Having a technique in place with the design, and the ability to abstract the aforementioned factors would play a pivotal role in accelerating the whole learning process by an agent. Learning shared representations provides this capability to have this milestone by learning the robust, transferable abstractions of the environment. This is majorly because those elements possess the ability to generalize over a group of tasks that an agent needs to deal with while operating within its environment [13].

The idea of value function plays an important role with the space of RL as it is being extensively used in conjunction with functional approximation methodology to generalize over the large-sized state-action space of the operating environment within which an agent needs to function [14]. The significance of value function lies in its ability to determine the quality of a specific state that an agent needs to be in while in an environment. Value functions often have the ability to demonstrate the compositional structure concerning the state space and goal states [15]. Empirically, prior research efforts have proven that value functions could effectively capture as well as represent knowledge beyond their current goal, and this could be efficiently leveraged or re-used for future learning purposes [14]. It is possible to learn the value of optimal value-functions by efficient usage of space of state-action values that are exchanged among the tasks that an RL agent encounters and deals with during its operation within the operating environment. This could be made possible by having the ability to accommodate the aforementioned common structure into the following value iteration and policy-iteration procedures namely fitted Q-iteration and approximate policy iteration respectively [16].

C. PROGRESSIVE NEURAL NETWORKS

The design of this approach is based on leveraging the function approximation ability of the neural network and having a high degree of similarity with transfer learning methodology. The challenges related to achieving the optimal performance on the multi-tasking in DRL were about the effective application of TL and eliminating the catastrophic forgetting. As an answer to these issues, extensive research efforts were carried out, and as a direct result of that, an approach named progressive neural networks was proposed. The technique possessed the capability to mitigate the impacts of catastrophic forgetting and derived a way to utilize prior accumulate knowledge by using the lateral connections to features that are learned already. The progressive neural network method is proposed by DeepMind as a multi-tasking technique by adopting the idea of lateral features transferring with neural networks [17]. The major aspect of the proposed model by this methodology is having not only the capability to learn new tasks but also to maintain the prior knowledge gathered by using neural networks. The main objective of having a series of progressive neural networks is to conduct the knowledge transfer across a group of tasks quite efficiently. Theoretically, the design of the progressive neural networks is for achieving two key objectives. Firstly, to establish a system having the capability

to efficiently incorporate the previously gathered knowledge while conducting the learning at each layer of the whole feature hierarchy. Secondly, in order to come up with a system having enough immunity to handle the impacts of catastrophic forgetting [7].

The major benefit of adopting this methodology is that progressive networks possess the capability to retain a set of pre-trained models whole throughout the training cycle [17]. Along with this, it can utilize the pre-trained model to learn lateral connections and thereby derive useful features needed for new tasks in the future. Having an approach with such abilities brings the combination of both richer compositionality and an easy room for the integration of previously acquired knowledge at every layer of feature hierarchy. The continual nature of learning associated with this approach facilitates the agents to both learn a group of tasks that are encountered in a sequence as well as gives the capability to perform knowledge transfer from previous tasks to enhance the overall convergence speed [18]. Progressive networks combine both these aspects into model architecture where catastrophic forgetting is eliminated by instantiating a new neural network for each of the tasks that are being solved during an agent's lifetime within its operating environment. In addition to this, transfer of knowledge is made possible through the lateral connections to the list of features from the prior learned neural network columns [17]. At any given time step t, when a new task is learned, the model appends a new column of knowledge into its current framework as a new neural network unit. Subsequently, the newly added unit would be used while learning the successive tasks. Each of the new column or neural network units created would be trained to solve a specific Markov decision process (MDP) [17]. One of the plausible drawbacks related to this approach is that this could lead to a computationally expensive model as its size could be growing as well with the progress in the learning cycle.

D. PathNet

PathNet is another multi-task RL methodology designed for the purpose of achieving artificial general intelligence (AGI) by joining together the aspects of transfer learning, continual learning, and multitask learning [18]. The core design aspect of PatheNet lies with a neural network-oriented algorithm that utilizes multiple agents which are deployed within the neural network. The major role of each agent is to identify which all parts of the network could be re-used while learning new tasks [19]. All agents are being treated as pathways (known by the name genotypes) within the ecosystem of the neural network to decide the subset of parameters that could be utilized during the learning process [20]. All of these parameters used within the forward propagation of the learning cycle undergo updates at the backpropagation phase of the PathNet algorithm. During the learning process, a tournament selection genetic algorithm would be deployed for selecting the pathways through the use of a neural network. During the operation, various actions carried out by the agents inside the neural network build a knowledge database on the efficient re-use of environment parameters for new tasks or actions. All of the agents are designed to function in a parallel manner along with all the other agents present within the system who would be involved in learning other tasks as well as sharing parameters among them for facilitating the positive transfer of knowledge [20].

Generally, PathNet architecture is made up of DNN having L layers, with each of them having M modules. Then each one of these modules itself would another neural network. Consolidated outputs of modules belonging to each layer would then sent into the active modules residing in the subsequent layer [7]. For every individual layer, there would a limit on the maximum number of modules that could be supported for each of the pathways, and often this number lies between 3 to 4 [20]. The final layer within each of the neural networks for each of the tasks that are being learned will be always unique. More importantly, this would not be shared with any of the remaining tasks running within the operating environment. One major advantage of the PathNet approach lies within the re-usability factor by which neural networks could leverage and learn from existing knowledge databases, and thereby save time by avoiding learning from scratch for the new tasks. The impact of this would be more prevalent within the context of RL, as there could more interrelated tasks within the wide action space associated with the operating environment. The PathNet approach has shown impressive results for positive transfer of knowledge for various datasets like binary MNIST (Modified National Institute of Standards and Technology), CIFAR-100 (Canadian Institute For Advanced Research), and SVHN (The Street View House Numbers) supervised learning classification tasks. The same kind of results has been obtained with several Atari2600 gaming and Labyrinth RL tasks.

E. POLICY DISTILLATION

Policy distillation (PD) and actor-mimic (AM) are the two approaches that are based on the concept of distillation intended to achieve multi-task DRL. The core objective of distillation lies in the factor of minimalization in terms of the costs of computations associated with ensemble methods [21]. The idea of an ensemble can be viewed as a group of models wherein the prediction outputs of this group are joined with help of either a technique of weighted average or voting [22]. Studies on ensemble methods were of the prominent research fields within the past decade. The most famous ensemble-oriented methods are namely bagging, boosting, random forests, Bayesian averaging, and stacking [22]. Two major drawbacks related to ensemble-based methods are in terms of their huge memory requirements for operation as well the amount of time needed for execution during runtime. The need for relatively high execution time makes them slow in terms of generating the output. To mitigate these issues, a distillation methodology was suggested, which is designed based on the model compression approach. The main objective of this approach is to compress the learned

function by the complex model, which is an ensemble to a much scaled-down and faster model that has got a relatively comparable level of performance with the original ensemble [22]. Subsequently, this same approach was mapped into the domain of neural networks [23].

By leveraging model compression, PD is being considered as an approach that could be used for the purpose of extracting the policy of an RL agent. Following this, the same policy could be utilized towards training a new network at an optimum level with a relatively smaller size and a higher level of efficiency. Following this, an intelligent agent could utilize the same approach for the consolidation of multiple task-oriented policies into one policy. Earlier research efforts conducted on the PD front were mostly carried out by an RL algorithm namely DQN (deep Q-network). With this, the PD method could be utilized in an effective way to transfer one or more than one active policy from a DQN to another network that is untrained [7]. DQN is a famous, state-of-the-art model-free technique deployed in RL with the help of deep neural networks (DNN). This model functions within an environment having a discrete set of action choices. DQN was proven to exhibiting a performance level that outperforms human-level scores on a collection of multiple Atari 2600 games [3]. In this context, the distillation approach could be applied both at a single task level as single game policy distillation. Similarly, the same could be applied at a multi-task level as a knowledge transfer technique from a teacher model T to a student model S. Within the single task-based policy distillation, the responsibility of data generation would be handled by the teacher network, which is a trained DQN agent, and following this a supervised training would be performed by student network. To achieve multi-task PD, n different DQN-based single-game experts (agents) would be trained independently [22]. Later on, all of these individual agents generate both the inputs and target date, which are stored inside memory buffers. Subsequently, the distillation agent uses all of these n different data stores sequentially for learning.

F. ACTOR-MIMIC

The major design aspect and most desired characteristic of an intelligent agent lies with its ability to act under different operating environments, accumulate information, then subsequently perform knowledge transfer from those past experiences gathered to new situations. The idea behind the actor-mimic is based on the aforementioned methodology with a special focus on aspects such as multi-task learning and transfer learning. Having these two abilities would make an intelligent agent on learning efficiently on how to handle and act concurrently with multiple tasks, and subsequently, generalize that knowledge gathered or accumulated to the new domains [24]. Typically, actor-mimic could be perceived as a technique to train a single deep policy network with the help of a set of source tasks that are related. Empirically, it is shown that impressive levels of performance could be achieved by using models that are trained with this approach on several games. Specifically, with a high amount of similarity level between source as well as target tasks, features that are learned while training source tasks could be quite efficiently used for the generalization of target tasks' training [25].

The actor-mimic methodology utilizes the power of both DRL as well as model compression techniques to train a single policy network. The key intention behind the usage of such a training method is to enable the network to gain knowledge on how to act within a group of distinct tasks under the guidance of multiple expert teachers [7]. Subsequently, representational knowledge gathered by the DRL policy network could be leveraged towards generalizing the new tasks without having any sort of anterior expert guidance. Validation of this technique was majorly carried out within the arcade learning environment (ALE) [26]. Generally, actor-mimic is being considered as part of the larger imitation learning class of methods. These methods are generally rooted in the idea of adopting expert guidance to train an agent on how to act within a particular operating environment. Under the imitation learning methodology, a policy would be directly trained to mimic an expert's behavior while sampling the set of actions from the mimic agent's space [24].

G. ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC

Google DeepMind proposed the idea of a simultaneous, parallel learning-based training approach towards multi-task learning, and formulated an algorithm by the name A3C (asynchronous advantage actor-critic). According to this, multiple intelligent agents, which are also called workers, will be running simultaneously in a parallel fashion within different instances of the same operating environment [4]. All the workers that are running within the environment will be in charge of updating a global value function asynchronously. The key essence of this approach lies in the fact that at the time of training each of these individual agents, at any given time-stamp t, each of these agents would be undergoing through different states within the environment. This property offers them an independent and unique way of learning. The impact of this unique A3C algorithm will be in a position to deliver each of the agents with a very highly efficient learning trajectory within the vast state space of the operating environment [27]. A3C is designed as an enhancement of the original actor-critic methodology, which is having two different, independent neural network units- one for the actor module and the other one for the critic module with its loss functions [7]. At the basic level, an actor module could be treated as a function approximator unit that governs agent at each state, in a quite similar way as being judged by RL methods like Q-learning or in REINFORCE. In these two approaches, a neural network calculates either a function that leads to the calculation of policy or deriving the policy itself directly [28]. When it comes to the critic module, it acts as a more sort of judging unit which effectively evaluates the effectiveness of the policy created by the actor and then

provides feedback on the same which helps further enhancement of the future policy calculations [4].

IV. RELATED WORK

This section provides the details on the related work done on the multi-task DRL front. Before the inception of deep reinforcement learning, most of the multi-task-oriented algorithms relied on transfer learning to realize proper control over different tasks. Besides, some research efforts were carried out to investigate the joint training of multiple value functions or policy functions over a set of tasks [29], [30]. However, the functionalities of all of these algorithms were limited by handcrafted features. Even though a huge amount of work has been done to improve DRL algorithms over single tasks, relatively there is much less amount of work done for multi-task scenarios. Some of those research attempts either focused on the exploration and generative models or explored learning universal abstractions of state-action pairs or feature successors, which are quite similar to transfer learning methodology [31]. DiGrad (Differential Policy Gradient) is an approach developed for simultaneous training of multiple tasks sharing a set of common actions in continuous action spaces. The proposed framework is based on differential policy gradients and can accommodate multi-task learning in a single actor-critic network. This framework was designed predominantly for efficient multi-task learning in complex robotic systems and tested on 8 link planar manipulators and 27 degrees of freedom (DoF) Humanoid for learning multi-goal reachability tasks for 3 and 2 end effectors respectively [32]. Another research work related to the multi-task learning done was based on the model-based approach to deep reinforcement learning which we use to solve different tasks simultaneously. This model was developed with a recurrent neural network inspired by residual networks that decouple memory from computation allowing to model complex environments that do not require lots of memory [5]. Another relevant work at the multi-task front done was mainly attempting to address the partial observability issue of RL with help of the deep decentralized multi-task multi-Agent reinforcement learning method [33]. It was based on a decentralized single-task learning approach that is robust to concurrent interactions of teammates and presented an approach for distilling single-task policies into a unified policy that performs well across multiple related tasks, without explicit provision of task identity [34]. The diffusion-based Distributed Actor-Critic (Diff-DAC) is a deep neural network-oriented distributed actor-critic algorithm designed to single-task and to average multitask reinforcement learning (MRL). In this method, each agent is having access to data from its local task only, and during the learning, process agents share their value-policy parameters with neighbors to converge to a common policy but without having a central node [35]. For the remainder of this section, we will mainly focus on comparing and contrasting the three state-of-the-art approaches namely Distral, IMPALA, and PopArt.

A. DISTRAL

Distral (DIStil and TRAnsfer Learning) is one of the well-known approaches developed by google DeepMind for the purpose of multi-task training. It is a prototype made for concurrent RL with more than one task [36]. The key design objective was to build a generic model to distill the centroid policy first and following this transfer the commonality details and behavior patterns of multiple workers operating within multi-task RL context. Rather than following a parameter sharing-based policy among the multiple worker agents operating in the environment, Distral's design methodology mainly focuses on distributing a distilled policy to individual workers, and this policy should conceive the commonality in behavior across multiple related tasks. Once the distilled policy is derived, then the same can be used to govern the task-specific policies by adopting regularization with the help of Kullback-Leibler (KL) divergence [24]. By this method, initially, knowledge gathered from one task would be distilled in the form of a shared policy, subsequently, the same knowledge could be transferred to other related tasks operating in the environment. By adopting this methodology, each of the individual workers would be trained independently to solve their task, in such a way that each of the workers could be staying more in line with shared policy. Training for this policy will be conducted with the help of the distillation process that serves as centroid for all the individual task policies [36]. This approach is found to present impressive results in terms of the transfer of knowledge within complex 3-dimensional operating environments for RL problems.

Empirically it has been observed that the Distral approach often outperforms the traditional methods, by a significant margin, that are oriented on parameter sharing policy of neural networks towards achieving multitasking or transfer learning. The two key reasons behind this are mentioned below. Firstly, its due to the level of impact distillation has got on the process of optimization. It is more prevalent while adopting KL divergences as a prime method to regularize task models' output in deriving the distilled model extracted from each of the policies of individual tasks. Secondly, the application of distilled model itself as a means to regularize to train the individual task models within the environment. More importantly, the application of the distilled model as a method to regularize comes with the notion of regularizing the collection of individual workers in a much impactful manner by stressing on task policies by more margin than at the level of parameter [20].

B. IMPALA

Google DeepMind came up with another well-known multi-task learning approach by the name IMPALA (Importance Weighted Actor-Learner Architecture). It is based on the idea of having a distributed agent architecture that is designed by adopting the model of a single RL agent with only one set of parameters. The core design characteristic of the IMPALA model is about operating environment flexibility. This model

is designed with the ability to not only in the efficient utilization of resources within a single-machine-oriented training environment but also it can be scaled to operate with multiple machines without the need to sacrifice both data efficiency and utilization of the resource. By following a novel off-policy correction method by the name V-trace, IMPALA is capable of gaining quite a stable learning trajectory with a very high throughput level by combining both decoupled acting as well as learning [37]. In general, the DRL model's architecture follows the notion of a single learner (also known as a critic) clubbed with many numbers actors. Under this ecosystem, initially, each of the individual actors creates its learning parameters called trajectories and subsequently shares that knowledge with the learner (critic) by following a queue mechanism. The learner subsequently accumulates the same kind of knowledge trajectories from all of the multiple actors operating within the environment, which eventually acts as a source of information to prepare the central policy. Before starting the next learning cycle (trajectory), all of the individual actors operating within the environment gather the updated policy parameters details from the learner(critic module) [37]. This approach is quite analogous to the popular RL algorithm named A3C. The architecture of the IMPALA was inspired hugely by the same algorithm. RL algorithm model used within the IMPALA follows a topology of a system having a group of actors and learners who build knowledge through collaboration.

The design of the IMPALA leverages an actor-critic-based model to derive a policy π and, a baseline value function named $V\pi$. Major units of the IMPALA system consist of a group of actors that generates g trajectories of experience in a continuous manner. In addition to this, there could be at least one or more than one learner that leverage the generated trajectories shared from the individual actors to learn the policy π, which is an off-policy. At the beginning of every individual trajectory, an actor initially updates its local policy μ to the latest learner policy π. Subsequently, each actor would adopt and run that policy for n number of steps in its operating environment [37]. Upon completion of these n steps, each of the individual actors sends another set of information consisting of - the trajectory of states, actions, and rewards together with related policy distributions to the learner. In this manner, the learner will have the opportunity to continuously update its policy π each time whenever the actors share their trajectory information from the environment. In this fashion, IMPALA architecture gathers experiences from different individual learners within the environment, which are further passed to a central learner module. Following this, the central learner calculates the gradients and then generates a model having a framework of independent actors as well as learners. One of the major characteristics of the IMPALA architecture is its operational flexibility which allows the actors to be present either on the same machine or it can be evenly distributed across numerous machines.

C. PopArt

A third approach by the name PopArt proposed by Google DeepMind came out as a solution to mitigate the issues associated with the existing IMPALA model. PopArt was aimed to address the reasons behind the suboptimal performance factors and thereby enhance the RL in multi-task-oriented environments. The core design objective of PopArt is to reduce the impacts of the distraction dilemma problem associated with the IMPALA model and, thereby stabilize the learning process in a better way to facilitate the adoption of multi-task RL techniques [38]. The term distraction dilemma refers to the probability of learning algorithms getting distracted only by a fraction of few tasks from the large pool of multiple tasks to be solved. This scenario in turn leads to the challenges related to resource contention. It is about establishing the right balance between the necessities of multiple tasks operating within the same environment competing for a limited number of resources offered from a single learning system. The design methodology of the PopArt model is based on the original IMPALA architecture model by adding multiple CNN layers combined with other techniques like word embeddings with the help of a recurrent neural network of type long-short term memory (LSTM) [38].

PopArt model functions by gathering the trajectories from each of the individual tasks to the RL agent's updates. In this manner, the PopArt model makes sure that every agent within the environment will have its role, subsequently proportional impact during dynamics of overall learning. The key design aspect of the PopArt model relies on the fact that modifying the weights of the neural network, will be based on the output of all tasks operating within the environment. During the first stage of operation, PopArt estimates both mean as well as the spread of the ultimate targets such as the score of a game across all tasks under consideration. Following this, PopArt capitalizes on these estimate values to normalize the targets before making an update on the network's weights. This approach in turn makes the whole learning process more stable and robust. With the set of various experiments conducted with popular Atari games' environment, PopArt has demonstrated its capabilities and improvements over other multi-task RL architectures [38].

V. HYBRID MULTI-TASK LEARNING MODEL

The major motivation behind the proposed hybrid multi-task approach is to address and mitigate some of the key challenges associated with DRL multi-tasking, which are not fully covered by the state of the art. In this paper, we extend our approach in [39] to address the DRL agent's performance optimization bottlenecks by adopting the hybrid multi-task learning-based approach in complex operating environments having a higher number of distinct DRL agents. Besides, this work also examines the impact of semantic dissimilarity of DRL agents' tasks on the overall momentum of performance optimization. The challenges such as partial observability, amount of the training time as well as training data samples required, and effective exploration often

act as the bottlenecks in the performance optimization of a DRL agent.

The proposed approach named the hybrid A3C model is an attempt to address most of these aspects, by extending the basic actor-critic model to two different environments with a high level of semantic similarity. The key aspect the of A3C algorithm is its ability to learn multiple instantiations of a single target task simultaneously, and also its ability to improve the model's performance by transferring the knowledge between multiple instantiations [4]. The proposed hybrid A3C approach will be leveraging this key aspect and will attempt to achieve this objective across, two different by semantically similar environments with related tasks. The hybrid approach will be heavily relying on the applicability of the multi-threaded capability of the A3C algorithm across semantically related tasks running in two different environments. The proposed approach could be treated as a model running two threads of the A3C algorithm, wherein each thread will be managing the multiple instantiations of the tasks running in each environment. Each of these individual threads would consider itself as a subtask such as A and B, with each of them sharing its learning with the learner in an asynchronous manner. Further on, the learner (global network) will be converging the knowledge from both of these threads and deducing a new policy, that will be applied back on the threads. The key aspect of the hybrid approach is only to enhance the performance of the RL agent through a joint-learning through multi-task learning approach by using deep reinforcement learning. Fig. 2 shows a high-level architecture model of the proposed hybrid multi-task approach.

The hybrid A3C model deploys multi-threaded asynchronous variants of the advantage actor-critic algorithm. The major objective behind designing this model is to find a methodology that can train deep neural network policies reliably and without large resource requirements. During the construction of the hybrid A3C model, initially, we conducted its validation on a desktop-based environment which is having a dual-core CPU on a single machine. Under this environment, we have conducted basic level testing with a pair of actor-learner worker threads. With this, one (actor-learner) worker thread was assigned to run the task from each game's environment. Throughout the execution, this model asynchronously attempts to derive and optimize the global policy based on the observations that multiple actors-learners running in parallel are likely to be exploring different parts of the environment. At an individual actor-learner module level, it is possible to have different exploration policies in each module to maximize this diversity. In this way, having different exploration policies in different threads of the actor-learner module, the overall changes being made to the global network parameters by these different actor-learners applying asynchronous updates in parallel are likely to be less correlated. This model is designed to run on a single machine with a standard multi-core CPU and applied to a variety of Atari 2600 domain games for testing.

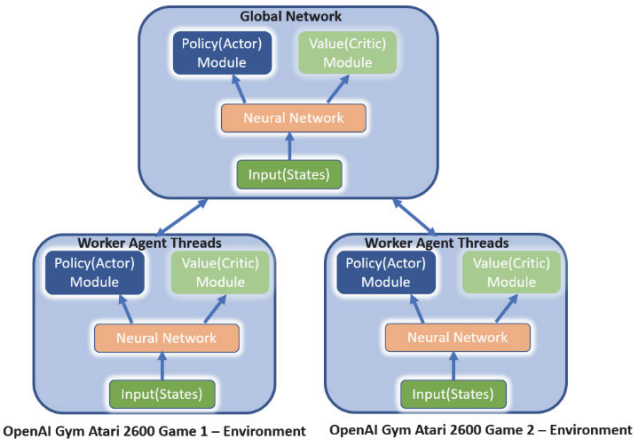

FIGURE 2. The architecture of the hybrid parallel multi-task model.

The semantic similarity aspect of the related tasks running two different gaming environments is the most vital factor to achieve the above-mentioned objectives, which otherwise give challenges in terms of negative knowledge transfer. Negative Transfer is considered to be one of the key challenges while dealing with the multi-tasking aspect within the reinforcement learning domain. The main idea of knowledge transfer learning in a multi-task context is that transferring knowledge accumulated from learning from a set of source samples under one agent may improve the performance of another task agent while learning on the target task [24]. However, this knowledge transfer could impact the overall learning progress and performance of the agent in either way, positively or negatively. If there is a considerable difference between the source tasks and target tasks, then the transferred knowledge could create a negative impact.

Having multiple environments with a high level of semantic similarity would in-directly improve the partial observability by exchanging the learning across the agent's operating environment [40]. Similarly, having multiple actor-critic models operating simultaneously across two semantically similar environments would mitigate RL agent's issues associated with effective exploration, sufficient amount of training samples, and the training time required to reach an optimized performance level.

A. ACTOR-CRITIC METHODOLOGY

Unlike some simpler techniques which are based on either value-iteration (Q-learning) methods or policy-gradient (PG) methods, the actor-critic(AC) methodology combines the best parts of both the methods, which are the algorithms that predict both the value function $V(s)$ as well as the optimal policy function $\pi(s)$. In other words, actor-critic methods consist of two models, namely an actor module and a critic module. Thereby AC attempt to combine the aspects of both policy gradient and value gradient into a single model. Fig. 3 shows the diagram of actor-critic methodology.

The actor acts as a policy network, that decides for a given state s which action a to be taken at each given time step t.

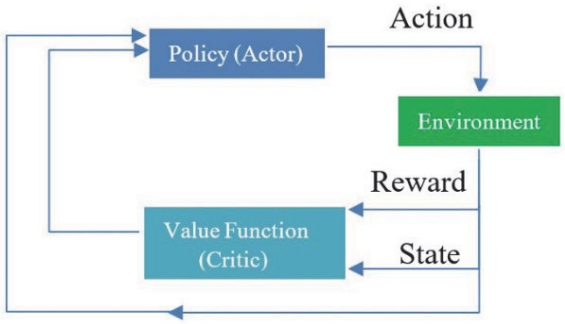

FIGURE 3. Actor-Critic model.

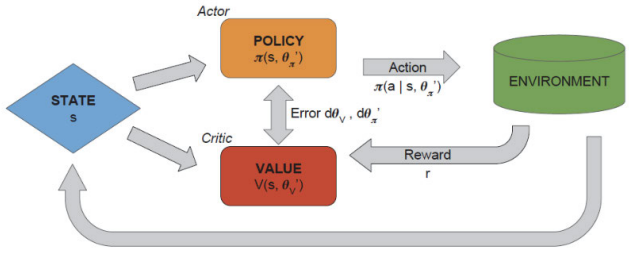

FIGURE 4. A single thread of actor-critic worker execution.

The critic consists of a value network $V\pi(s, a)$ that tells how promising action is under the current state s. Having said that, in its role critic outputs an evaluation value $V(s, a)$ for the actor, which indirectly helps the actor to adjust its policy for better results. At the same time, both actor and critic networks update themselves according to the knowledge gathered by their respective neural networks from the environment. This internally helps the agent to converge its policy to the optimal policy π_θ^*. In summary, the critic module updates the value function parameters w, and depending on the algorithm it could be either action-value $Q_w(a|s)$ or state-value $V_w(s)$ whereas the actor module updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

Fig. 4 shows the single actor-critic worker agent flowchart [41]. The learning agent uses the value from the value function calculated by the critic module to update the optimal policy function of the actor module. Note that here the policy function means the probabilistic distribution of the action space. To be exact, the learning agent determines the conditional probability $P(a|s;\theta)$ which otherwise means parametrized probability that the agent chooses the action a when in state s. The policy is often modeled as a function $\pi_\theta(a|s)$ that is parameterized to θ. The value of the DRL agent's reward function depends on this policy, and the algorithms are used to optimize θ. The reward function is defined as below, wherein $d^\pi(s)$ notation refers to the stationary distribution of Markov chain for π_θ (for the on-policy state distribution under π).

$$J(\theta) = \sum_{s\varepsilon S} d^\pi(s) V^\pi(s) \tag{2}$$

$$= \sum_{s\varepsilon S} d^\pi(s) \sum_{a\varepsilon A} \pi_\theta(a \mid s)\, Q^\pi(s, a) \tag{3}$$

Within the AC model, the critic is in charge of updating the value function parameters w, and based on the DRL algorithm it could be either an action-value function $Q_w(a|s)$ or state value function $V_w(s)$. Based on the details of the value function shared by the critic, the actor updates the policy parameter θ for the $\pi_\theta(a|s)$. The execution of an actor-critic algorithm can be explained by the below steps [42].

1) Initialize s,θ,w at random, and sample $a \sim \pi_\theta(a|s)$
2) For $t = 1 \ldots T$:
   a) Sample reward $Rt{\sim}R(s,a)$ and next state $s'{\sim}P(s'|s,a)$;
   b) Then sample next action $a' \sim \pi_\theta(a'|s')$
   c) Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta ln\pi_\theta(a \mid s)$;
   d) Compute the correction (TD error) at time t for action-value:
      i) $\delta_t = r_t + \gamma Q_w(s' \mid a') - Q_w(s, a)$
      ii) Use it to update the parameters of the action-value function as given $w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$
   e) Update $a \leftarrow a'$ and $s \leftarrow s'$

Both the learning rates α_θ and α_w, are predefined for policy and value function parameter updates respectively.

B. ACTOR

An actor is a module that controls how a policy-based DRL agent behaves within an environment. The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy π^*. The policy-based algorithms such as Policy Gradients (PG) and REINFORCE try to find the optimal policy directly without the Q -value as the intermediate step. Often an actor could be a function approximator such as a neural network with its objective as to identify the best action while a DRL agent is in a state S_t at time step t. The neural network could be either fully connected or a CNN.

C. CRITIC

The critic, on the other hand, evaluates the action by computing the value function (value-based). The role of the critic is to evaluate how good an action is taken by the agent with the help of a value-based approach. As in the case of the actor, the critic also could be a function approximator such as a neural network. The result is that the overall architecture will learn to play the game more efficiently than the two methods separately.

D. ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC(A3C)

A3C is a state-of-the-art DRL algorithm developed based on the AC methodology. This algorithm is designed to function both in discrete and continuous action space environments and can be treated as the multi-thread version of the original AC algorithm. A3C makes the AC algorithm converge faster by running multiple agent threads [43]. Each of these threads consists of an independent actor-critic pair that interacts with

FIGURE 5. The ecosystem of single A3C worker thread with Atari 2600.

the environment simultaneously. The agents, which are also known as workers, are trained in parallel and update periodically a global network, which holds shared parameters. The updates are not happening simultaneously and that's where the asynchronous comes from. The unique exploration experience offered by each of the global actor-critic networks. With such multiple threads sharing the experience with a global network in an asynchronous fashion, A3C eliminates the bias of continuous experience trajectory by feeding only a small batch of experience tuple *(s, a,r,s')* at any time. After each update, the agents reset their parameters to those of the global network and continue their independent exploration and training for n steps until they update themselves again. With this approach, the information flows not only from the agents to the global network but also between agents as each agent resets its weights by the global network, which has the information of all the other agents. Fig. 5 shows the ecosystem of a single actor-critic worker.

A3C uses a deep neural network to model both a policy network $\pi(a_t|s_t;\theta)$ and a value network $V(s_t;\theta)$. For a given state S_t, the policy network (which is the ''actor'') predicts the optimal action to take at S_t while the value network (which is the ''critic'') approximates the future reward from taking the optimal action at S_t. By theory, these two networks are separate, but in practice, we use the same convolutional layers for both the policy and value networks with separate output layers at the end. The Asynchronous nature of A3C means that multiple actor-critic threads are running at the same time, each with its environment. Each thread steps through its environment with its own local CNN, periodically updating a globally shared CNN wherein all networks have an identical architecture. For each thread, at every t_{max} local steps or when a terminal state is reached, that thread syncs its local parameters with the global parameters, computes gradients, and applies them upstream to the global network [41].

A3C follows online learning by adopting a policy gradient method, directly from the states as they are processed by each worker agent thread. The policy is developed naturally as each thread runs within its stochastic Atari 2600 based gaming environment and updates to the global parameters. Fig. 6 indicates the worker agent architecture with CNN.

This methodology suggests that A3C does not overfit to any particular state trajectory of a specific worker thread. The notion of Advantage A is used to measure the difference between the expected reward and estimated reward. By using the value of advantage instead, the agent also learns how much better the rewards were than its expectation. This gives

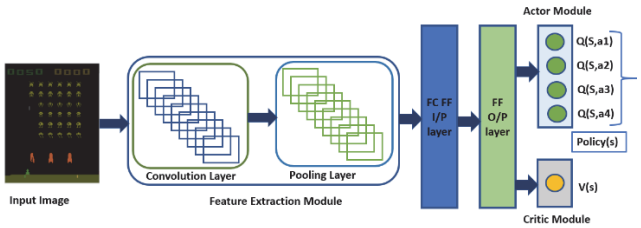

FIGURE 6. The architecture of the worker agent thread in A3C.

a new-found insight to the agent into the environment and thus the learning process is better. The advantage metric is given by the following expression

$$\text{Advantage} : A = Q(s,a) - V(s) \tag{4}$$

where Q refers to the Q value calculated by the critic module based on the actual reward and TD error following an actor's policy-based chosen action. The Advantage function named $A(S_t a_t;\theta,\theta_v)$ is calculated that needs to be discounted future rewards accumulated to t_{max} or at the terminal state.

$$A(S_t a_t;\theta,\theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v) \tag{5}$$

Gradients associated with both policy and value networks are denoted by the following equations (6) and (7) respectively, which are calculated by summing over all the states in the past t_{max} local iterations of each worker agent thread's execution [41].

$$\nabla_{\theta'} log\pi(a_t|s_t;\theta)A(s_t,a_t;\theta,\theta_v) \tag{6}$$

$$d\theta = d\theta + \partial(R - V\left(s_t;\theta_V^{'}\right))^2/\partial\theta^{'} \tag{7}$$

The pseudocode of the A3C algorithm for each worker agent thread within the hybrid multi-task model is given by the algorithm mentioned below [4].

VI. IMPLEMENTATION OF HYBRID MULTI-TASK SYSTEM PROTOTYPE

This section details the methodology adopted towards the prototype implementation of the proposed hybrid multi-task model which is based on the A3C algorithm. Throughout the implementation, the prototype was tested with various games under the Atari 2600 environment provided within the OpenAI Gym [44]. The Gym library is a toolkit made by OpenAI for developing and comparing RL algorithms. The first stage of the hybrid multi-task model was constructed by adopting the A3C algorithm for the gaming environment Breakout-v0. The high-level architecture of the model is based on the actor-critic methodology. In our context, the actor is a neural network that parameterizes the policy π (a | s) and critic is another neural network that parameterizes the value function $V(s)$. The policy network outputs the policy (π), based on which the actor chooses an action within the environment, and the value network outputs the value function $V(s)$. Each

Algorithm A3C Algorithm – Pseudocode for Each Actor-Leaner Thread

// Assume global shared parameter vectors θ *and* θ_v*and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ^1 and $\theta_v^{'}$
Initialize thread step counter $t \leftarrow 1$
repeat
Reset gradients $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
Synchronize thread-specific parameters $\theta^{'} = \theta$ and $\theta_v^{'} = \theta_v$
$t_{start} = t$
Get state s_t
repeat
perform $a_t$according to the policy $\pi(a_t|s_t; \theta^{'})$
Receive reward r_t and new state s_t
$t \leftarrow t + 1$
$T \leftarrow T + 1$
until terminal s_t **or** $t - t_{start} == t_{max}$
$R = \{0$ terminal s_t
$R = \{V(s_t, \theta_v^{'})$ for terminal s_t //Bootstrap from last state
for $i \in \{t - 1 \ldots t_{start}$ **do**
$R \leftarrow r_i + \gamma R$
Accumulate gradients wrt $\theta^{'} : d\theta \leftarrow d\theta + \nabla_{\theta^{'}} log\pi(a_i|s_i; \theta^{'})(R - V(s_i; \theta_v^{'}))$
Accumulate gradients wrt $\theta_v^{'} : d\theta_v + \partial(R - V(s_i; \theta_v^{'}))^2/\partial\theta_v$
end for
Perform asynchronous update of θ using $d\theta$ and of θ using $d\theta_v$
until $T > T_{max}$

of these networks has its respective weights which are often represented by notations such as θp and θv.

$$\pi(a|s, \theta p) = Neural\ Network\ (input : s,\ weights : \theta p) \tag{8}$$

$$V(s, \theta v) = Neural\ Network\ (inputs,\ weights : \theta v) \tag{9}$$

A more graphic intense Atari 2600 game environment named- Breakout-v0 is being relatively treated as a complex environment as we will be having an infinite number of state-action spaces to deal with. To accommodate and handle this environment, the neural network-based model was used for the validation. At the root level, this environment will employ a pair of CNN models to implement both actor and critic modules for a single worker. There will be multiple instances of the CNN class objects to implement the multiple worker threads used within the multi-task model. Similarly, the global network was also deployed as a pair of CNN to support the implementation of actor-critic modules at the global network level.

Fig. 7 shows the high-level architecture view of the multi-task model having N worker threads of execution coordinated and managed by a global network. Each of these

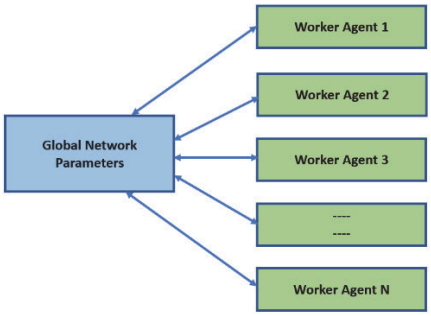

FIGURE 7. A3C multi-task worker agent model.

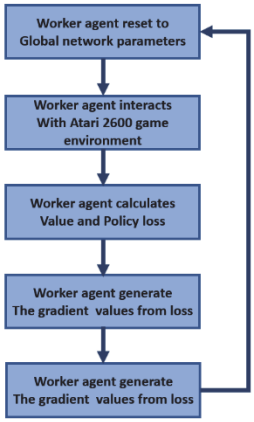

FIGURE 8. Training workflow of worker agent thread.

individual blocks is made up of a pair of CNN networks, each for the actor(policy) and critic (value function) modules. In other words, A3C utilizes N worker agents attacking the same game environment while being initialized differently. This indirectly points out that each of these agents starts at a different point in their environment so they will go through the same environment in different ways to solve the same problem.

Fig. 8 shows the training workflow of each worker agent.

Within the A3C-based multi-task worker agent environment, each of the individual worker agents is managed by the global network directly. Under this scheme, initially, each of the workers is reset with parameter values shared by the global network, later on, the worker interacts with its copy of the environment. Even though each of the worker agents is operating within the same game environment, they are being initialized differently. This allows each of these agents to start at a different point in their environment. During its operation, each worker agent plays a fixed number of game episodes and calculates the value and respective policy loss. As these modules, both actor and critic are implemented using the neural network, gradient values are calculated from the losses incurred during its operation. These gradient values will be shared with the global network after the work agent finishes a fixed number of game episodes. The algorithm behind the operation of the A3C multi-task worker agents' model is mentioned below.

Algorithm Algorithm of A3C-Based Multi-Task Model Worker Agent

while not done:

$a = sample\ an\ action\ a \sim \pi_\theta(a|s)$

$s`, r, done = Perform\ action\ a -env.step(a)$

$G = r + \gamma V(s`)$

$L_p = -(G - V(s))log(\pi\ (a|s,\theta_p))$

$L_v = (G - V(s))\wedge^2$

$\theta_p = \theta_p - \alpha * d_{Lp}/d_{\theta p}$

$\theta_v = \theta_v - \alpha * d_{Lv}/d_{\theta_v}$

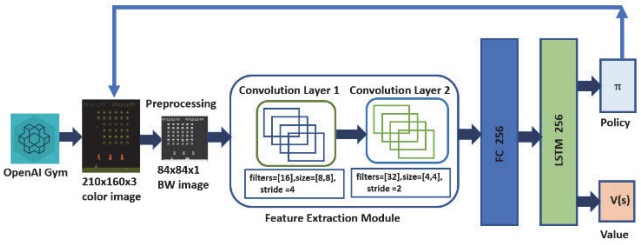

FIGURE 9. The CNN-based architecture of a single A3C worker agent.

During the operation, each of the worker agents loops through each step of the game, and samples the action, and updates the weights of both the neural networks- actor and critic. The algorithm runs until a preset number of episodes of the game are played, wherein initially action is sampled from the actor (policy network). Further on, upon completion of that action respective reward *(r)* and a new state (s') are calculated. Based on the new state reached, the total discounted future return (G) is calculated by applying the discount factor (gamma). Based on this each of the individual neural networks calculates its policy loss (Lp), and value loss (Vp) [45]. Further on, the neural network uses gradient descent to update the respective network weights (θp – policy network weight and θv – value network weight) to minimize the loss.

At the root level, this environment will employ a pair of convolutional neural network (CNN) models to implement both actor and critic modules for a single worker. There will be multiple instances of the CNN class objects to implement the multiple-worker threads used within the multi-task model. Similarly, the global network is also deployed as a pair of CNN-based actor-critic modules at the global network level. These neural network models act as a function approximator by processing each screenshot of the game as its input. We have used RMSpromp optimizer with this implementation. During the first stage of experiments, the evaluation of the multi-task learning model was performed on a machine having two cores (dual-core). Under this initial test-setup, each worker agent will be running on each core, and hence both worker agents are executed in parallel. Fig. 9 is a diagrammatic representation of the CNN-based model used to implement each of the individual worker agent threads.

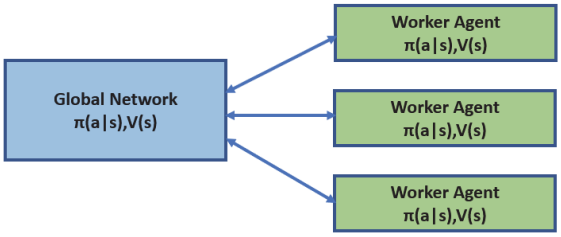

FIGURE 10. Gradient update by worker agents with the global network.

Now every so often, this global network is going to send its weights to a set of worker agents each with their copy of policy and value network. Further on each of these individual worker agents will be playing a few episodes of the game under its environment using its network weights from its own experience. From its own experience, each worker agent can calculate its policy gradient updates and value updates. Knowledge of these updates will be limited to only these individual worker agents. Eventually, worker agents send their gradient values to the global network so that the global network can update their weights accordingly. Every so often the global network gives its new updated parameters back to its working agents so worker agents are always working with a relatively recent copy of the global network. In this working model, worker threads play episodes of games under their respective environments, find the errors, and calculate the update gradients which will be shared with the global network regularly. Fig. 10 shows the sharing of gradient updates by individual worker agents with the global network.

VII. EXPERIMENTS AND RESULTS

A3C provides a multi-threaded and asynchronous approach to deep reinforcement learning [43]. This algorithm gives the capability to have a model to be trained with multiple, different explorations of a single target task, providing data sparsity, and avoiding the use of memory replay. Given the multi-threading characteristics, the proposed hybrid model attempts to leverage A3C's ability to perform multi-task learning without modifications when applied to different, but semantically related tasks. To do so, we simultaneously train multiple tasks using a single A3C model, allowing the network to asynchronously share knowledge obtained from and to all tasks. The hybrid A3C model attempts to learn two different tasks and then combine the learning to accelerate the performance. Evaluation of the proposed hybrid multi-task model will be conducted on a prototype based on the A3C model and trained with the Atari 2600 environment provided in the OpenAI Gym. The Gym library is a toolkit for developing and comparing reinforcement learning algorithms [44]. It makes no assumptions about the structure of your agent and is compatible with numerical computation libraries, such as TensorFlow or Theano. A3C algorithm used for the experiments will be based on Google DeepMind's paper titled-asynchronous methods for deep reinforcement learning.

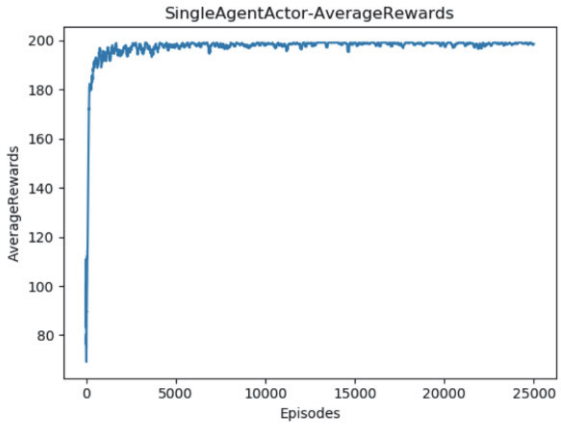

FIGURE 11. Single-agent actor – average rewards.

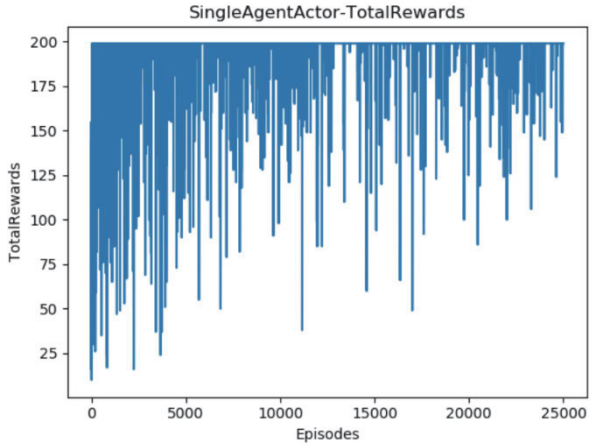

FIGURE 12. Single-agent actor- total rewards.

As a preliminary step towards the development of the proposed system model, the initial set of experiments are conducted with the game of CartPole-v0 which is having a finite set of action and state space. The methodology followed was to individually develop the single-agent actor which is based on policy gradient, and similarly a single agent critic which is a value-based network to measure the performance. Both these networks were developed as the standard feedforward neural networks and experiments are conducted for the finite number of episodes. As an outcome of the experiment performance of both single-agent actor and critic are measured.

Fig. 11 to Fig. 13 represent the statistics generated for the single-agent actor feedforward neural networks.

Fig. 14 to Fig. 16 represent the statistics generated for the single-agent critic feedforward neural networks.

It is evident from the statistics that policy gradient-based actors can increase the rewards over the episodes gradually. At the same time, the value-based critic module can show the increment in performance in the early episodes, with a small dip in the mid episodes with a fluctuating result for the forthcoming episodes.

Following OpenAI Atari 2600 gaming environments will be used for the evaluation of the proposed model,

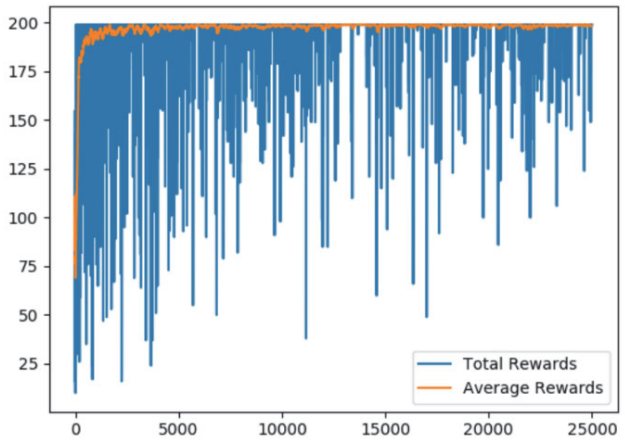

FIGURE 13. Single-agent actor- merged results.

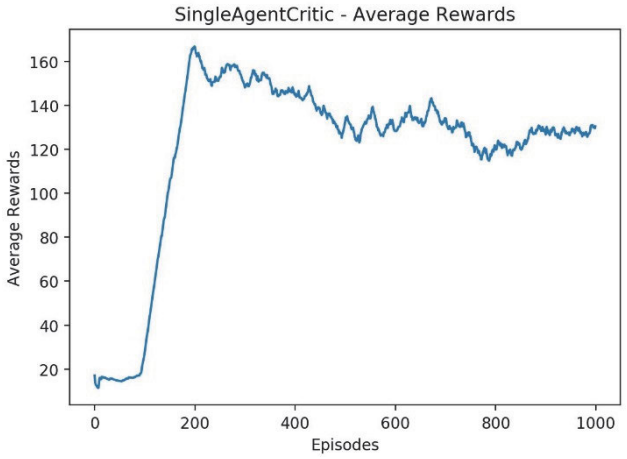

FIGURE 14. Single-agent critic-average rewards.

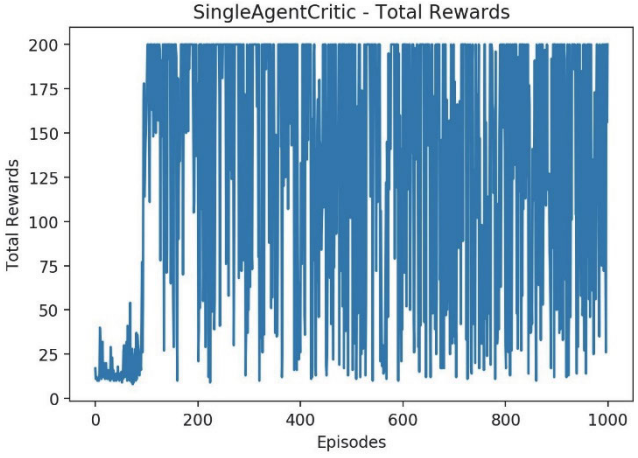

FIGURE 15. Single-agent critic-total rewards.

Pong-v0, Breakout-v0, SpaceInvaders-v0, DemonAttack-v0, and Pheonix-v0. During the first stage of evaluation, the performance of the reinforcement learning agent will be measured individually on each of these gaming environments to generate the initial test statistics. The test results

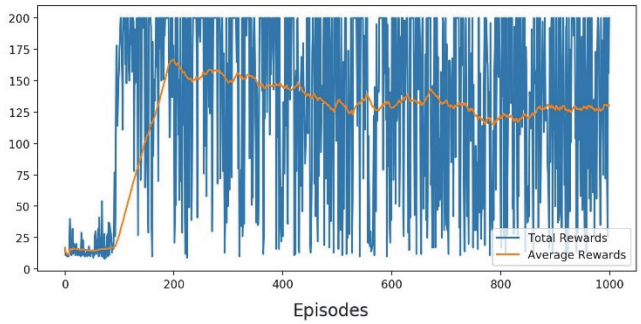

FIGURE 16. Single-agent critic-merged results.

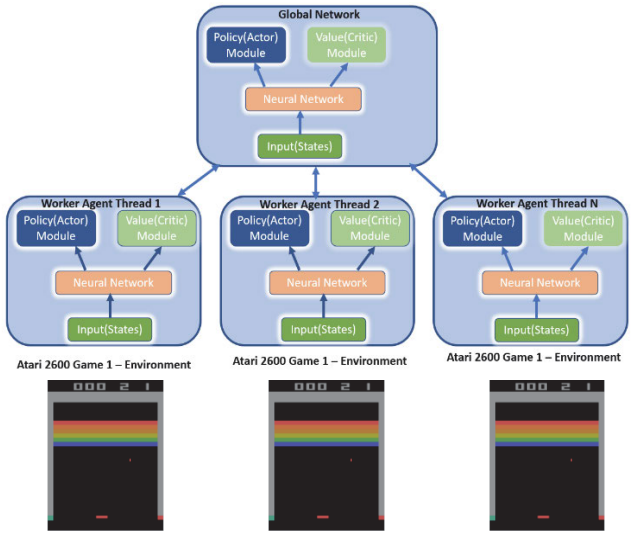

FIGURE 17. Breakout-v0 multi-task workers environment.

generated by the A3C model were trained within the OpenAI Atari 2600 environments provided in the OpenAI Gym [10]. In the next step towards the evaluation of a proposed hybrid multi-task model, the A3C algorithm based on a multi-worker agent-based environment is created for a more graphic intense Atari 2600 game environment Breakout-v0. From the perspective of a DRL agent, this environment is being treated as a complex one as we will be having an infinite number of state-action spaces to deal with. To accommodate and handle this environment, a convolutional neural network (CNN) based model was used for the validation. This configuration was tested under a desktop-based environment by using a multi-task environment having four worker threads that combinedly executed 500,000 steps of the game. Each of the individual threads is having its copy of the environment but different from one another in terms of the view of the gaming environment. Fig. 17 shows the multi-task worker based environment for Breakout-v0

Fig. 18 to Fig. 20 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Breakout-v0. This testing was carried out by using 4-worker agents or worker threads based A3C model to generate the initial set of results of a desktop-based test environment.

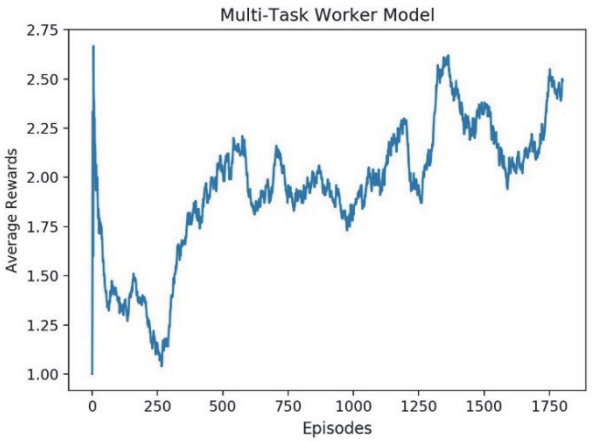

FIGURE 18. Breakout-v0 multi-task workers model-average rewards.

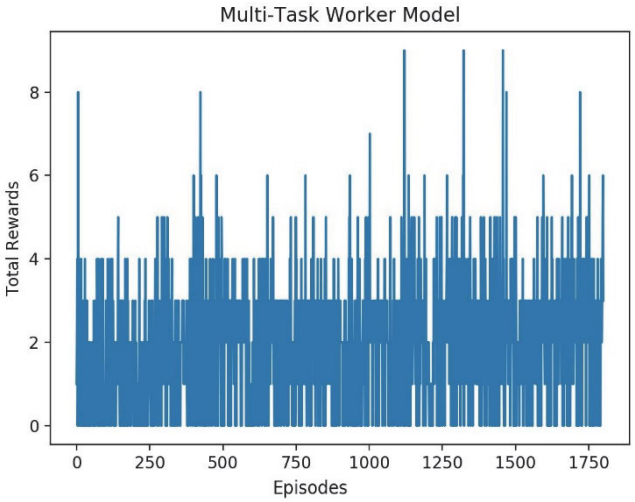

FIGURE 19. Breakout-v0 multi-task workers model-total rewards.

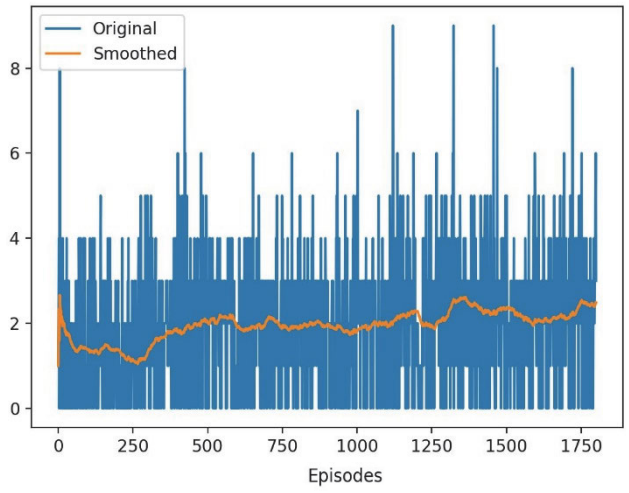

FIGURE 20. Breakout-v0 multi-task workers model –merged results.

As a further attempt towards the evaluation of the proposed hybrid multi-task model, the A3C algorithm-based multi-worker agent environment is also created for one more

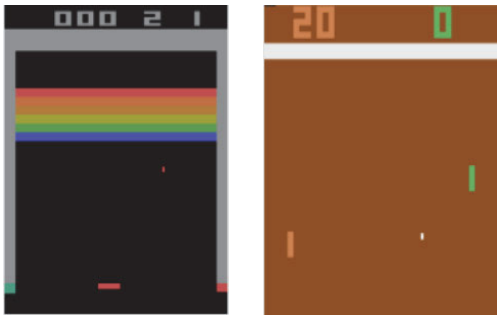

FIGURE 21. Snapshot of Breakout-V0 and Pong-v0 environment.

graphic intense Atari 2600 game environment named- Pong-v0. The decision to choose Pong-V0 was after the careful examination of the high level of similarity level among these two games, Breakout-V0 and Pong-V0. Having a reasonable level of similarity could act as an accelerator during the validation of the proposed hybrid multi-task learning model execution. Similar to the way how Breakout-v0 was tested earlier under a multi-task worker environment, the Pong-v0 game was also tested under a desktop-based environment by using a multi-task environment having four worker threads that combinedly executed 5 million steps of the game.

Each of the individual threads is having its copy of the environment but different from one another in terms of the view of the gaming environment. This environment will be having an infinite number of state-action spaces to deal with during the optimization of the DRL agent. To accommodate and handle the Pong-v0 gaming environment, a similar CNN-based model was used during the validation of the multi-task learning model. In both Pong and Breakout, a player must control a paddle to hit a ball. For Pong, the player must attempt to make an opponent miss the ball, while for Breakout the goal is to break as many bricks as possible. Fig. 21 shows the graphical representation for the Atari-2600 Breakout-v0 and Pong-v0 gaming environments.

Fig. 22 to Fig. 24 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Pong-v0.

As part of the detailed and exclusive evaluation of the proposed hybrid multi-task model, we decided to pick one more pair of Atari 2600 games namely Space Invaders-v0 and DemonAttack-v0 from the Gym library. The decision to choose these two games as the second test pair was after the examination of the high level of semantic similarity between their pattern play. Both these games are based on the theme of shooting wherein the player should be able to control a moving ship with the capability of shooting and hitting the enemies. In terms of complexity, Space Invaders is relatively less complex as the enemies in this game move more in a regular fashion than in the other game. Whereas in Demon Attack, there are a wide variety of enemies who moves more randomly with the capability to shoot back, which makes the gameplay more complex from the perspective of the RL agent. More importantly, every game used in this experiment has its reward structure that is in-built by the Gym library. In other words, even though there is some level of semantic similarity between the games chosen within each test pair, the scoring and reward structure followed within each game is unique.

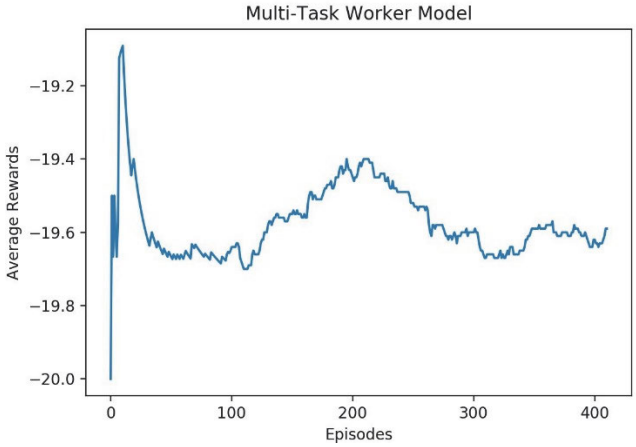

FIGURE 22. Pong-v0 multi-task workers model-average rewards.

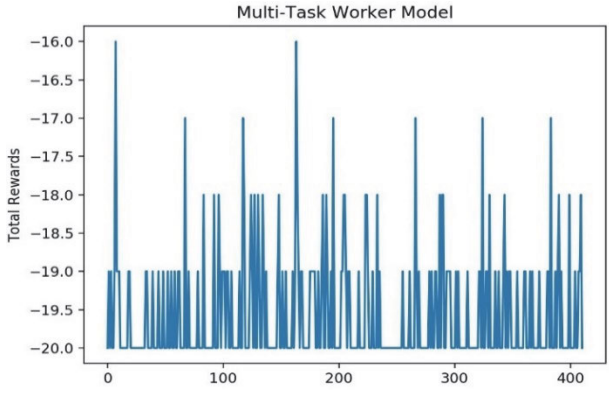

FIGURE 23. Pong-v0 multi-task workers model-total rewards.

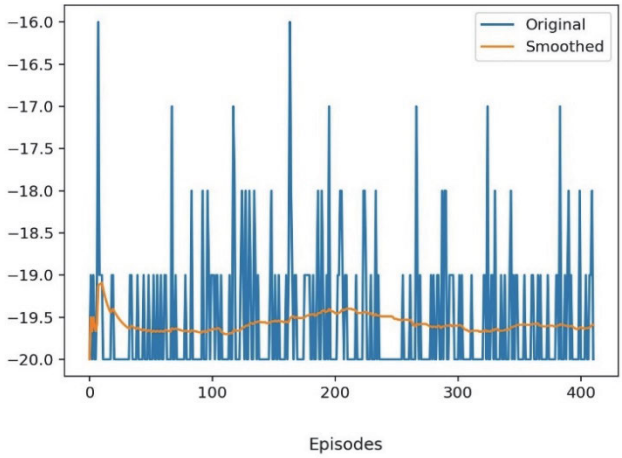

FIGURE 24. Pong-v0 multi-task workers model-merged results.

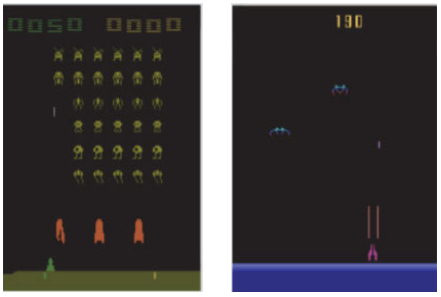

FIGURE 25. SpaceInvaders-v0 and DemonAttack-V0 environment.

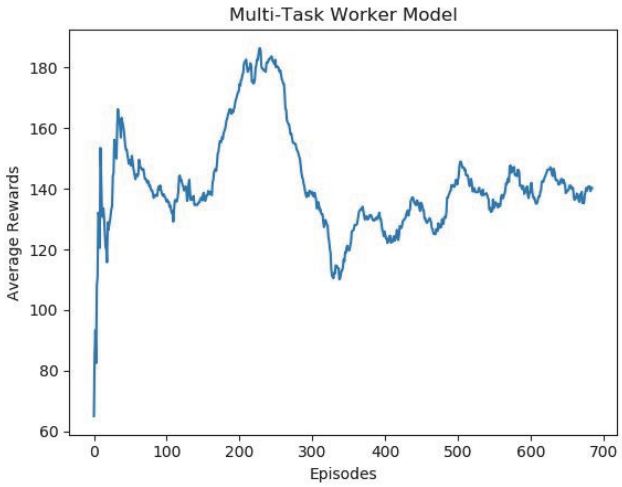

FIGURE 26. SpaceInvaders-v0 multi-task workers model-average rewards.

Fig. 25 shows the graphical representation for Atari-2600 based SpaceInvaders-v0 and DemonAttack-v0 gaming environments.

Similar to the way how previous two games from the first pair were tested, the SpaceInvaders-v0 game was also tested under a desktop-based test environment by using a multi-task worker model having four worker threads that combinedly executed about 500,000 steps of the game. Fig. 26 to Fig. 28 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Space Invaders-v0.

Each of the individual threads is having its copy of the environment but different from one another in terms of the view of the gaming environment. This environment will be having an infinite number of state-action spaces to deal with during the optimization of the DRL agent. To accommodate and handle the Space Invader-v0 gaming environment, a similar CNN based model was used during the validation of the multi-task learning model

This testing was carried out by using 4-worker agents or worker threads-based A3C model to generate the initial set of results of a desktop-based test environment.

Similar to the way how Space Invader-v0 was tested earlier under a multi-task worker environment, the DemonAttack-v0 game was also tested under a desktop-based environment by using a multi-task environment having four worker threads

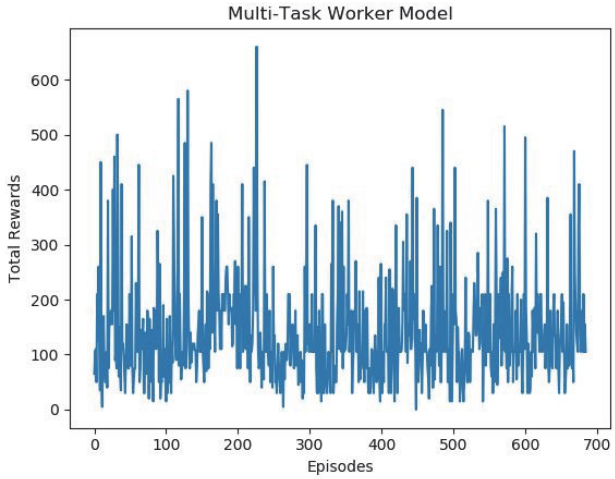

FIGURE 27. SpaceInvaders-v0 multi-task workers model-total rewards.

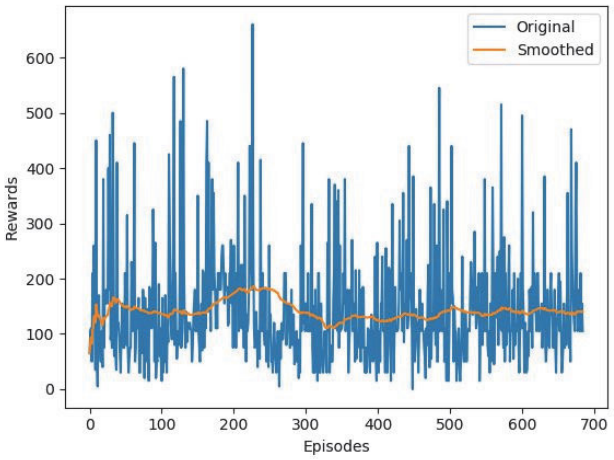

FIGURE 28. SpaceInvaders-v0 multi-task workers model- merged results.

that combinedly executed 500,000 steps of the game. Each of the individual threads is having its copy of the environment but different from one another in terms of the view of the gaming environment. This environment will be having an infinite number of state-action spaces to deal with during the optimization of the DRL agent.

Fig. 29 to Fig. 31 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environment named Demon Attack-v0.

To test and generate better results with a higher number of episodes of gameplay for each game under the proposed hybrid multi-task model, we decided to test the proposed model under a cloud-based test environment. As part of this, we opted to move our testing to machines with GPU with CUDA cores support under the cloud environment hosted by Paperspace. This allowed us to rent a server in the cloud with much higher throughput than that of our local machine.

Paperspace server used has up to 8GB of graphic memory and 32 GB of RAM and equipped with NVIDIA GPU - Quadro P5000 having CUDA support (with

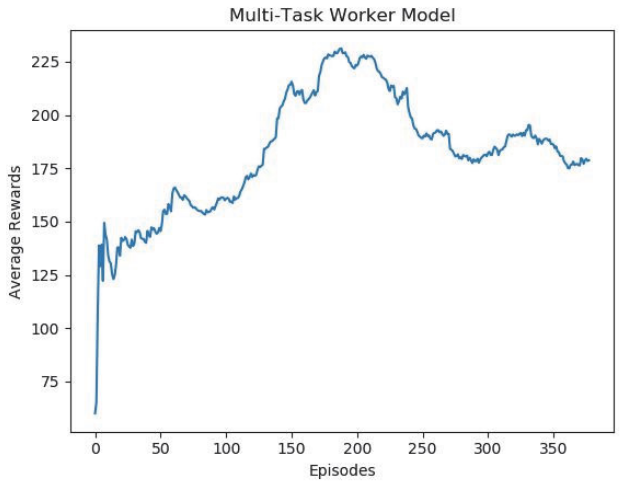

FIGURE 29. Demon Attack-v0 multi-task workers model-average rewards.

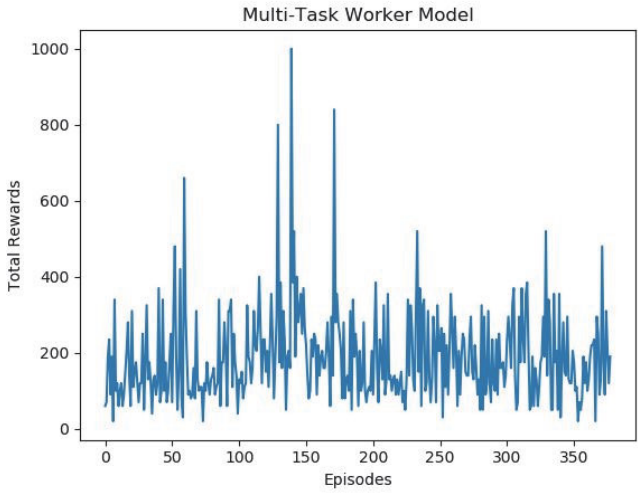

FIGURE 30. DemonAttack-v0 multi-task workers model-total rewards.

2560 CUDA cores) to facilitate the parallel computing for deep learning applications.

During this process, we configured a couple of Windows OS-based virtual test machines namely Gen 2 (P4000) having NVIDIA GPU supported with CUDA cores in the cloud environment. Each of the Atari2600 games was tested with 8 worker agents for a higher number of global steps. To capture the test results, a tensor board visualization tool was employed which uses the event file captured during the test execution to generate the test execution results. Fig. 32 depicts the working environment of the cloud server machine.

Fig. 33 to Fig. 36 show the test results captured for the A3C algorithm based on the multi-task worker model for the Atari 2600 gaming environments under the virtual test machines under the cloud environment. Note that these figures were generated within TensorBoard (TensorFlow's visualization toolkit), the numbers on the x-axis represent the global steps in millions (taken by the agent), and the numbers on the y-axis

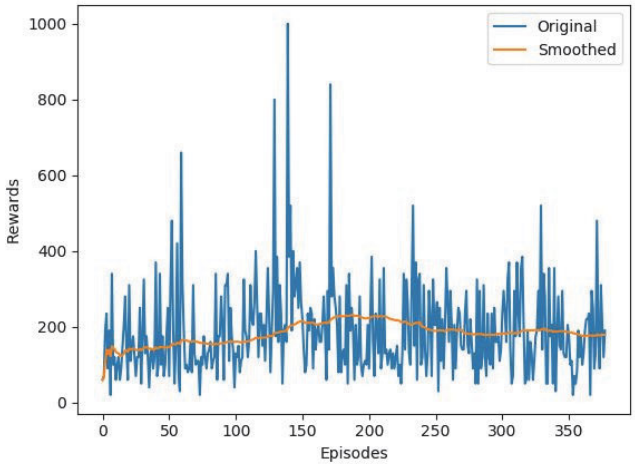

FIGURE 31. DemonAttack-v0 multi-task workers model-merged results.

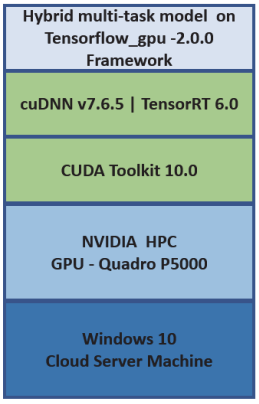

FIGURE 32. Test environment of Paperspace cloud server machine.

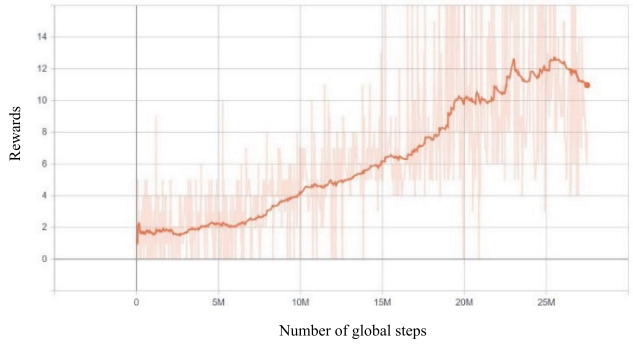

FIGURE 33. Breakout-v0 standalone test result with 8 multi-task workers.

represent the rewards (game score). The same convention applies to Figures 38 to 48.

Now, as the next step in the verification of our proposed hybrid multi-task model, we have tested the model by running two semantically similar games simultaneously.

At the end of testing, the individual test score for each game was captured. Since we have chosen two pairs of games with semantic similarity, we created a separate test setup for each pair. Fig. 37 shows the diagrammatic representation for each pair under the hybrid multi-task model. To maintain

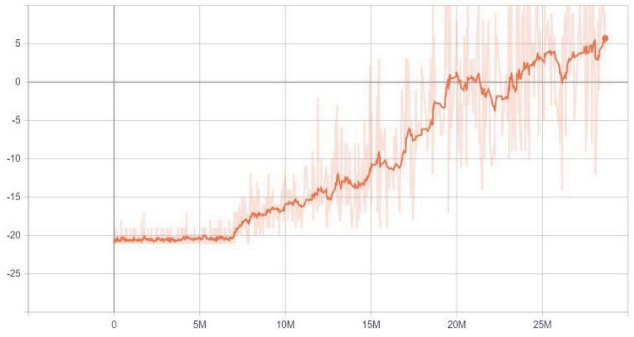

FIGURE 34. Pong-v0 standalone test result with 8-multi-task workers.

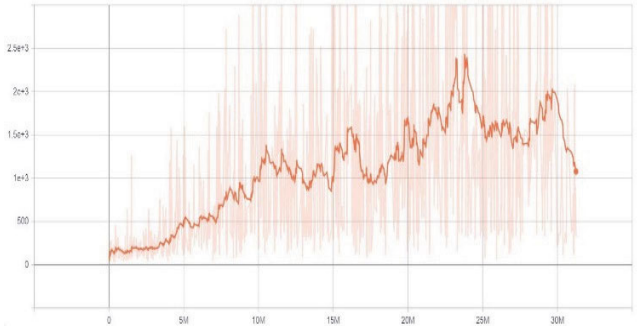

FIGURE 35. DemonAttack-v0 with 8-multi-task workers.

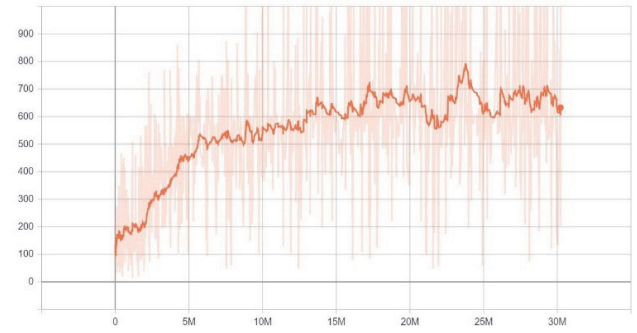

FIGURE 36. SpaceInvaders-v0 with 8-multi-task workers.

the uniformity of testing, each of the individual games was tested with 8 worker agents which totals to 16 worker threads altogether within the test environment. We have the RMSprop optimizer for the testing with related hyperparameters such as learning rate, decay, momentum, epsilon, clip norm parameter. We also have other hyperparameters such as the discount rate factor for rewards, maximum global steps, and worker threads having CNNs with 2 hidden layers with a ReLU activation function. Fig. 38 and Fig. 39 respectively show the test execution results captured for breakout-v0 and Pong-v0 under the joint test environment.

These test results are generated based on the experiments conducted with the Paperspace cloud server machines having the Nvidia GPU supported by CUDA cores. This environment facilitates the large-scale testing for the hybrid multi-task model having a CNN-based feature extraction module.

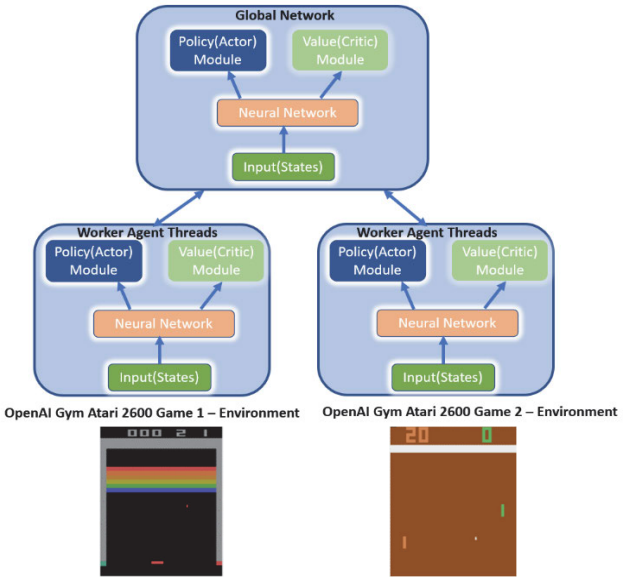

FIGURE 37. Hybrid multi-task model of Breakout-v0 and Pong-v0.

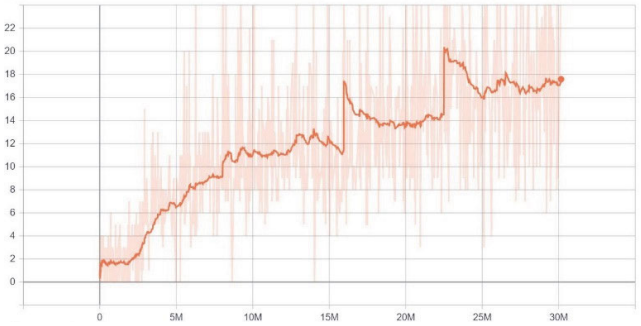

FIGURE 38. Breakout-v0 test results with the hybrid multi-task model.

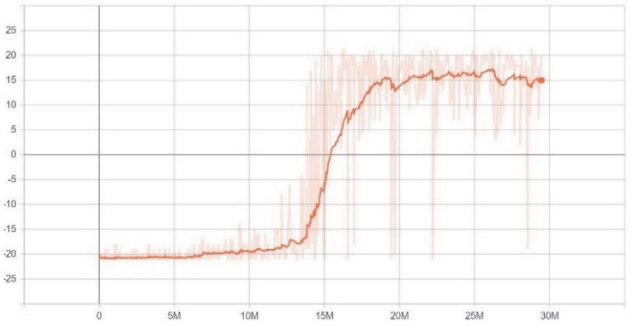

FIGURE 39. Pong-v0 test results with the hybrid multi-task model.

Similarly, we created the joint test environment for the second test pair consisting of Atari2600 gaming environments, SpaceInvader-v0, and DemonAttack-v0. To maintain the uniformity of testing, each of the individual games was tested with 8 worker agents which totals to 16 worker threads altogether within the test environment. Fig. 40 and Fig. 41 show the diagrammatic representation for each pair.

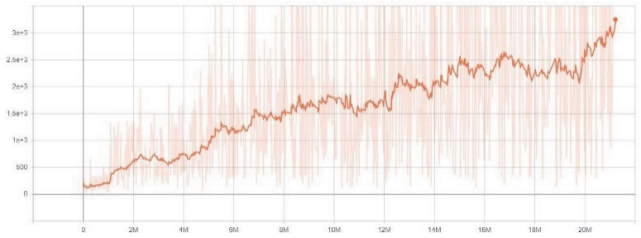

FIGURE 40. DemonAttack-v0 test results with the hybrid multi-task model.

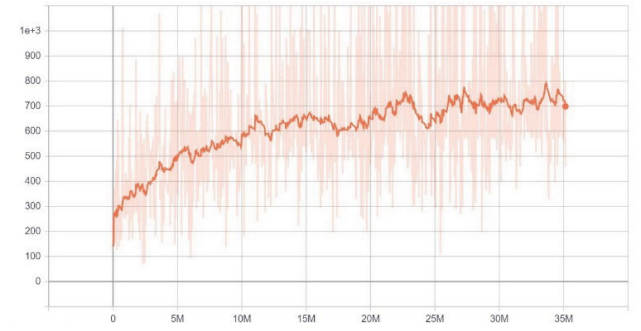

FIGURE 41. SpaceInvaders-v0 test results with the hybrid multi-task model.

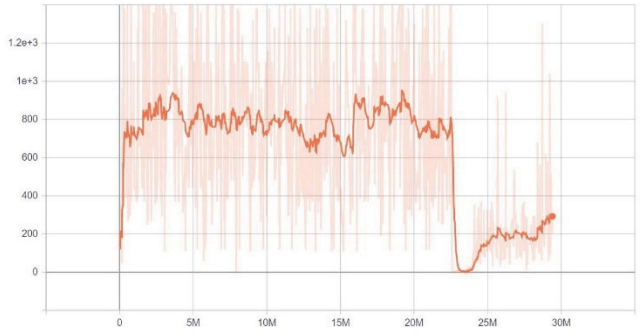

FIGURE 42. DemonAttack-v0 results for the semantic dissimilar test.

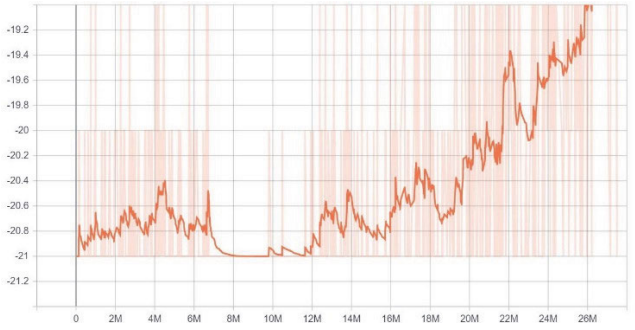

FIGURE 43. Pong-v0 results for the semantic dissimilar test.

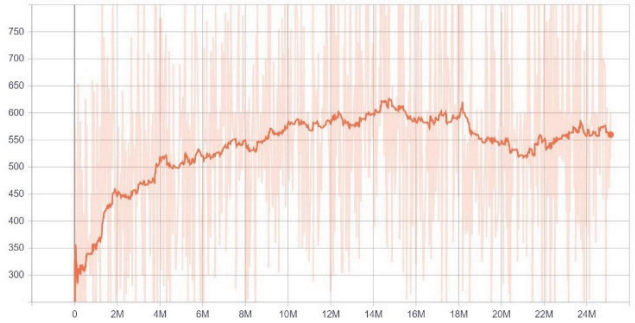

FIGURE 44. SpaceInvaders-v0 results for the semantic dissimilar test.

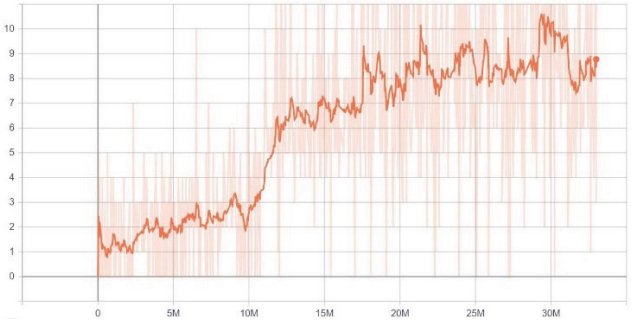

FIGURE 45. Breakout-v0 results for the semantic dissimilar test.

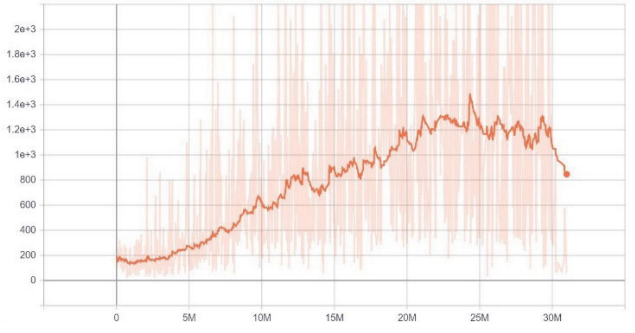

FIGURE 46. DemonAttack-v0 test results with hybrid multi-task model for 3 semantically similar environments.

To measure the impact of the DRL agent's performance with the hybrid multi-task model while testing with environments with high semantic dissimilarity, we have also conducted two pairs of testing. In this testing first pair of testing was done using DemonAttack-v0 and Pong-v0, which are having a high level of semantic dis-similarity level. Under this test environment, the performance of each of the individual games will be measured to see the impact of negative knowledge (gradient transfer). A similar test setup will be made ready for the second pair consisting of Atari2600 gaming environments namely, SpaceInvader-v0 and Breakout-v0. Results shown from Fig. 42 to Fig. 45 show the test results for each test pair.

During our test efforts, we also conducted experiments to measure the impact of individual game scores when the hybrid multi-task model is tested with three semantic similar games namely SpaceInvader-v0, DemonAttack-v0, and Pheonix-v0. Even though each of these games has a semantic similarity factor, at the same time, each of them is having its reward structure Fig. 46 to Fig. 48 show the respective test results captured with the hybrid multi-task model for the three OpenAI Atari 2600 gaming environments with the high level of semantic similarity.

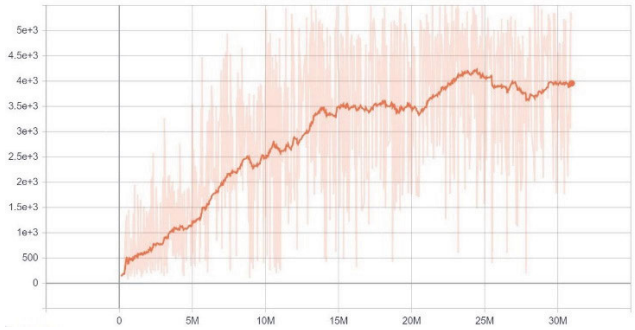

FIGURE 47. Pheonix-v0 test results with hybrid multi-task model for 3 semantically similar environments.

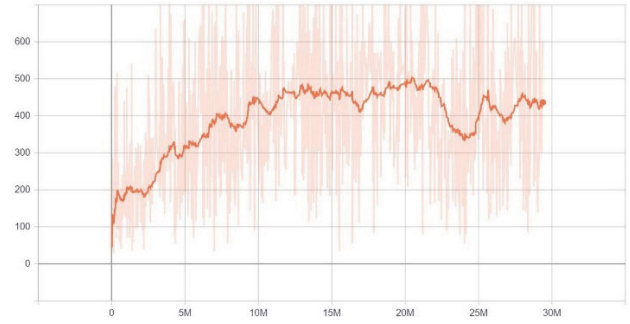

FIGURE 48. SpaceInvaders-v0 test results with hybrid multi-task model for 3 semantically similar environments.

VIII. DISCUSSION OF TEST RESULTS

This section analyzes the test results obtained with the hybrid multi-task model tested with various Atari 2600 gaming environments. In the first stage of the testing, we conducted a standalone kind of testing with each of the individual gaming environments individually. To conduct this testing, we have created the A3C algorithm-based multi-thread model wherein each of the games is tested by using 8 worker threads. To maintain the uniformity of the testing throughout this experiment, we have kept the count of worker threads as 8 for all the gaming environments. We tested our model by adding the final LSTM layer after the feedforward network to obtain the best performance of the A3C algorithm as a whole. We have extensively used NVIDIA GPU - Quadro P5000 having CUDA support (with 2560 CUDA cores) to facilitate parallel computing as it involves the use of CNN to process game screen images. More importantly, in the first stage of testing, we choose two sets of games, with set 1 consisting of Breakout-v0 and Pong-v0, then set 2 consisting of games SpaceInvaders-v0 and DemonAttack-v0. The decision to choose these games to form two sets was after the clear examination of semantic similarity factor among them. As anticipated, the base A3C-based multi-thread model was able to achieve performance enhancement on all of these games during the testing due to the parallel multi-task learning aspect of A3C. We have conducted the testing for 25 million to 30 million global steps for each of these individual games to have convincing test results for comparison with future state tastings planned.

In the second stage of testing, we experimented with our proposed hybrid multi-task model approach, wherein we trained two games, but with high-level semantic similarity, simultaneously. In contrast to the first stage testing, where gradients shared to the global network by worker agents are all of the same types, in the hybrid environment we have two different types of worker threads. As it is anticipated, the performance of individual games under the hybrid environment was not on par with standalone performance results obtained with the first stage of testing. As and when the progress of the game, we could see the impact of positive knowledge sharing among these two tasks that are trained jointly. Due to the semantic similarity among them, updates shared by the global network could mitigate some of the key challenges associated with partial observability in comparison to a single game-based environment. Based on the test results obtained with each of the sets that we mentioned earlier, we could see that each of the games under each set could boost its performance throughout the training. By this, we can establish that our hybrid multi-task model can learn multiple similar gaming tasks simultaneously without degradation in performance for any one of the individual gaming tasks. In comparison to the state-of-the-art methods discussed which are based on the distillation methodology, the hybrid multi-task model adopts to train and learn the method for a multi-task actor-critic network from the scratch. Along with this, the hybrid multi-task approach also measures the impact amount of positive knowledge transfer done through parameter sharing. As we have adopted a model-free-based approach, it is relatively less computationally intensive compared to a model-based approach.

In the next stage of testing with the hybrid multi-task model, we conducted experiments by testing the hybrid multi-task model with two different pairs of games with a high level of semantic dissimilarity. As we could see from the test results obtained, negative knowledge transfer or the gradients shared by two semantically dissimilar worker training threads had a huge impact on the individual games' score. As the test results indicate, all the individual games 'performance was hugely affected due to negative knowledge transfer. Finally, we also tested our model to see the impact on the positive knowledge transfer by training more than two semantically similar tasks with the same number of workers allocated to each game. The test results obtained indicate that as the number of worker threads increases, updates shared by the global network deteriorates in comparison to a hybrid multi-task model with two semantically similar tasks. This situation possibly requires more tuning on the hyperparameter front as well as catastrophic forgetting of the neural networks of the gaming environments, which will be addressed in the future work planned.

The objective behind the proposed hybrid multi-task learning model is to leverage multi-task learning capabilities offered by the core actor-critic methodology by using the A3C algorithm to optimize the DRL's performance. By having a hybrid multi-task-based learning environment, wherein

agents belonging to different but semantically similar games, we aimed at addressing some of the key challenges associated with the existing multi-task DRL. To showcase, the extent to which our model could address those issues, we would like to have a case study based on the test results obtained. For this purpose, we are using both the standalone and hybrid model test results obtained for the Breakout-v0 game as indicated by Fig. 33 and Fig. 38 respectively. To have a fair comparison and derive a convincing conclusion, we have ensured that the same amount of resources have been allotted in both test scenarios in terms of the number of worker threads, test configurations, and the number of global steps parameter. By having a comparison of these two test results, it is quite evident that in terms of the training time needed, the hybrid model could surpass the performance of the standalone model much ahead of time. After running the Breakout-v0 under a standalone model for 2.5+e5 (25 Million) global steps, the highest score it could achieve was a little over the range of 12, whereas the hybrid model could surpass the same level in almost half of its execution time. In continuation to this, it is reasonable to conclude that hybrid multi-task learning by having a group of different but semantically similar environments with similar tasks could reduce the impact of partial observability which restricts a DRL agent from choosing the optimal action while in a state. Due to the impacts of the positive knowledge transfer facilitated by the gradient transfers from the second environment's agents, the actor module within each worker is having a better policy to choose the optimal action while in a step. Having said this, by possessing better policy parameters actor module is in a better position to explore the environment in a much effective way and choose the optimal action in each state. This in turn is expected to improve throughout the DRL agents' execution as more positive knowledge transfer is anticipated to happen with more global steps of gameplay. The same kind of comparison case study could be applied to other game test pairs from the experiment. Seen in the light of these observations, it is reasonable to conclude that the hybrid multi-task learning model can address the objectives, it was aiming for, to a great extent.

Finally, we also would like to have a comparison of the proposed hybrid multi-task learning model against the three state-of-the-art techniques that were mentioned under the related work. In comparison to the hybrid multi-task model which relies on the idea of sharing the network learning parameters by a global network to individual workers, the Distral model works on the idea of sharing a distilled centroid policy that would regularize the workers running in the environment. When it comes to the comparison with the IMPALA model, its design approach is having similarity to the hybrid multi-task learning model in terms of the actor-critic methodology as it follows the topology of a set of actors with either a single learner or multiple learners. Within the IMPALA model, the learner's role is to create a central policy to be shared with the actors. Along with these learners have the flexibility to communicate among themselves for sharing the gradients. In the hybrid multi-task model, workers' accumulated gradients transfer or knowledge transfer among workers always governed by the global network. Additionally, the current implementation of the hybrid multi-task model mandates that all the workers be present on the same machine, where the IMPALA model supports distributed system-based working environment for the workers. The PopArt model is being considered as an extension of the IMPALA model itself and designed to address key issues such as distraction dilemma and thereby stabilize the process of multi-task learning.

IX. CONCLUSION AND FUTURE WORK

In this research work, we propose a hybrid multi-task model-that follows a parallel, multi-tasking approach for optimizing the performance of deep reinforcement learning agents. We present how to combine the multi-task learnings from two different deep reinforcement learning agents operating within two different by semantically similar environments running with related tasks. Initial stage experiments are conducted by applying the DRL algorithm A3C to Atari 2600 gaming environment to draw the results. During the experiments, we can establish that our hybrid multi-task model can learn multiple similar gaming tasks, at least two, simultaneously without making changes in the algorithm and degradation in performance for any one of the individual gaming tasks. The semantic similarity aspect of the related tasks running two different environments is the most vital factor to reduce the challenges posed in terms of the possible negative knowledge transfer.

For future work, we plan to conduct the experiments of the hybrid multi-task model with more complex gaming environments having a higher number of worker threads under GPU cloud server-based machine environment to draw strong conclusions on parallel multi-task learning. Along with this, we also would like to investigate the steps to mitigate the impacts of negative knowledge transfer and catastrophic forgetting in deep reinforcement multi-task learning.

ACKNOWLEDGMENT

The authors acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proc. Adv. Neural Inf. Process. Syst.*, 1996, pp. 1038–1044.
[2] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[4] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[5] A. Mujika, "Multi-task learning with deep model based reinforcement learning," 2016, *arXiv:1611.01457*. [Online]. Available: http://arxiv.org/abs/1611.01457
[6] R. Glatt and A. H. R. Costa, "Improving deep reinforcement learning with knowledge transfer," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 5036–5037.

[7] N. Vithayathil Varghese and Q. H. Mahmoud, "A survey of multi-task deep reinforcement learning," *Electronics*, vol. 9, no. 9, p. 1363, Aug. 2020.

[8] G. Boutsioukis, I. Partalas, and I. Vlahavas, "Transfer learning in multi-agent reinforcement learning domains," in *Proc. Eur. Workshop Reinforcement Learn.* Berlin, Germany: Springer, 2011, pp. 249–260.

[9] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1999.

[10] N. D. Nguyen, T. T. Nguyen, D. Creighton, and S. Nahavandi, "A visual communication map for multi-agent deep reinforcement learning," 2020, *arXiv:2002.11882*. [Online]. Available: http://arxiv.org/abs/2002.11882

[11] Y. Bengio, *Learning Deep Architectures for AI*. Boston, MA, USA: Now, 2009.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[13] D. Borsa, T. Graepel, and J. Shawe-Taylor, "Learning shared representations in multi-task reinforcement learning," 2016, *arXiv:1603.02041*. [Online]. Available: http://arxiv.org/abs/1603.02041

[14] R. S. Sutton, and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[15] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*. [Online]. Available: http://arxiv.org/abs/1511.05952

[16] T.-L. Vuong, D.-V. Nguyen, T.-L. Nguyen, C.-M. Bui, H.-D. Kieu, V.-C. Ta, Q.-L. Tran, and T.-H. Le, "Sharing experience in multitask reinforcement learning," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 3642–3648.

[17] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016, *arXiv:1606.04671*. [Online]. Available: http://arxiv.org/abs/1606.04671

[18] M. E. Taylor and P. Stone, "An introduction to intertask transfer for reinforcement learning," *AI Mag.*, vol. 32, no. 1, p. 15, Mar. 2011.

[19] R. Caruana, "Machine learning," *Mach. Learn.*, vol. 28, no. 1, pp. 41–75, 1997.

[20] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra, "PathNet: Evolution channels gradient descent in super neural networks," 2017, *arXiv:1701.08734*. [Online]. Available: http://arxiv.org/abs/1701.08734

[21] A. A. Rusu, S. Gomez Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," 2015, *arXiv:1511.06295*. [Online]. Available: http://arxiv.org/abs/1511.06295

[22] C. Bucilu, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 535–541.

[23] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*. [Online]. Available: http://arxiv.org/abs/1503.02531

[24] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," 2015, *arXiv:1511.06342*. [Online]. Available: http://arxiv.org/abs/1511.06342

[25] M. S. Akhtar, D. S. Chauhan, and A. Ekbal, "A deep multi-task contextual attention framework for multi-modal affect analysis," *ACM Trans. Knowl. Discovery Data*, vol. 14, no. 3, pp. 1–27, May 2020.

[26] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, Jun. 2013.

[27] Y. Wang, J. Stokes, and M. Marinescu, "Actor critic deep reinforcement learning for neural malware control," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 1005–1012.

[28] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 228–239, Feb. 2021.

[29] A. Lazaric and M. Ghavamzadeh, "Bayesian multitask reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2010, pp. 599–606.

[30] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *Proc. Eur. Workshop Reinforcement Learn.* Berlin, Germany: Springer, 2011, pp. 273–284.

[31] Z. Yang, K. E. Merrick, H. A. Abbass, and L. Jin, "Multi-task deep reinforcement learning for continuous action control," in *Proc. IJCAI*, 2017, pp. 3301–3307.

[32] P. Dewangan, S. Phaniteja, K. M. Krishna, A. Sarkar, and B. Ravindran, "DiGrad: Multi-task reinforcement learning with shared actions," 2018, *arXiv:1802.10463*. [Online]. Available: http://arxiv.org/abs/1802.10463

[33] T. T. Nguyen, N. D. Nguyen, and S. Nahavandi, "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3826–3839, Sep. 2020.

[34] S. Omidshafiei, J. Pazis, C. Amato, J. P. How, and J. Vian, "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," 2017, *arXiv:1703.06182*. [Online]. Available: http://arxiv.org/abs/1703.06182

[35] S. V. Macua, A. Tukiainen, D. G.-O. Hernández, D. Baldazo, E. M. de Cote, and S. Zazo, "Diff-DAC: Distributed actor-critic for average multitask deep reinforcement learning," 2017, *arXiv:1710.10363*. [Online]. Available: http://arxiv.org/abs/1710.10363

[36] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4496–4506.

[37] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," 2018, *arXiv:1802.01561*. [Online]. Available: http://arxiv.org/abs/1802.01561

[38] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, "Multi-task deep reinforcement learning with popart," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3796–3803.

[39] N. V. Varghese and Q. H. Mahmoud, "Optimization of deep reinforcement learning with hybrid multi-task learning," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Vancouver, BC, Canada, 2021, pp. 1–8.

[40] D. S. Chaplot, L. Lee, R. Salakhutdinov, D. Parikh, and D. Batra, "Embodied multimodal multitask learning," 2019, *arXiv:1902.01385*. [Online]. Available: http://arxiv.org/abs/1902.01385

[41] T. Chesebro and A. Kamko. (Dec. 15, 2016). *Learning Atari: An Exploration of the A3C Reinforcement Learning Method*. Accessed: Oct. 17, 2020. [Online]. Available: https://bcourses.berkeley.edu/files/70573736/download?download_frd=1

[42] L. Weng. (Apr. 18, 2018). *Policy Gradient Algorithms*. Accessed: Dec. 28, 2020. [Online]. Available: https://lilianweng.github.io/ and https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#actor-critic

[43] Z. Gu, Z. Jia, and H. Choset, "Adversary A3C for robust reinforcement learning," 2019, *arXiv:1912.00330*. [Online]. Available: http://arxiv.org/abs/1912.00330

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*. [Online]. Available: http://arxiv.org/abs/1606.01540

[45] Lazy Programmer. (Aug. 24, 2020). *Deep Reinforcement Learning in Python*. Accessed: Oct. 21, 2020. [Online]. Available: https://github.com/ and https://github.com/lazyprogrammer

NELSON VITHAYATHIL VARGHESE received the Bachelor of Technology degree in computer engineering from the Cochin University of Science and Technology, Cochin, India, and the M.A.Sc. degree in electrical and computer engineering from Ontario Tech University, Canada. His research interests include machine learning, neural networks, and data science.

QUSAY H. MAHMOUD (Senior Member, IEEE) was the Founding Chair of the Department of Electrical, Computer and Software Engineering, Ontario Tech University, Canada. He has worked as an Associate Dean of the Faculty of Engineering and Applied Science, Ontario Tech University. He is currently a Professor of Software Engineering. His research interests include intelligent software systems and cybersecurity.