Shared-Experience in Multi-agent Relational Deep Reinforcement Learning

Puttatida Mahapattanakul
Queen Mary University of London
School of Electronic Engineering and Computer Science
puttatida.m@gmail.com

***Abstract*—There is a growing evidence in the Reinforcement Learning research community that a relational representation of the state space has many benefits over a propositional one, such as introducing interpretibility and transparency. Yet, many relational structure has not been exploited for multi-agent reinforcement learning tasks and has only been studied in a single agent context so far. To mend such gap, we proposed a multi-agent extension of the Relational Deep Reinforcement Learning environment, Box-World, which uses the novel Shared-Experience Actor-Critic methods as the learning algorithms, allowing all the agents to learn relational information and co-evolve in a shared environment at the same pace. Our objectives are to explores how a relational structure of the state space can be exploited in an MADRL setting and suggest what can be done in the future to further improve the current work, which still poses several limitations.**

I. Introduction

Deep reinforcement learning (DRL) have contributed to many significant breakthroughs in recent years, surpassing human performance in domains ranging from Atari, Go and no-limit poker. This is majorly due to their their flexibility in how to exploit the statistical structure underlying observations and reward signals. However, they often face the issues of low sample efficiency and poor transfer beyond the specifics of the training environment [2]. And Like many Machine Learning model, RL models suffer from a lack of explainability. This defect can be highly crippling as many promising RL applications (defense, finance, medicine, etc.) need a model that can explain its decisions and actions to human users as a condition to their full acceptation by society [8]. Seeking a favourable tradeoffs between flexibility and efficiency as well as explainability, several studies have explored the use of relational inductive biases in deep learning, which falls under the Relational RL umbrella and exploit the use of relational state (and action) space and policy representations, combining relational learning (or inductive logic programming) with deep reinforcement learning together [20].

Zambaldi et al. [20] from Google's DeepMind explored the use of Inductive Logic Programming and self-attention to represent states, actions and policies through the use of first order logic, using a mechanism similar to graph neural networks and message passing computations, allowing entity-entity relations to be explicitly computed when considering the messages passed between connected nodes of the graph. In their box world environment, Self-attention method was used to compute interactions between entities, which, in this case, are the pixels in the RGB image, in a non-local pairwise manner. Such technique enable visualisation of the agent's attention weights associated to its available actions, where they can be interpreted to improve the understanding of its strategy.

While most reinforcement learning paradigms focus on single agents acting in a static environment, real-world agents often compete or cooperate with other agents in a dynamically shifting environment. In order to learn effectively in multi-agent environments, agents must not only learn the dynamics of their environment, but also those of the other learning agents present [4]. In this paper, we further extend this relational deep reinforcement learning framework by introducing another agent into the box world environment. Multi-Agent Deep Reinforcement Learning (MADRL) is concerned with developing Deep Reinforcement Learning algorithms that enable a set of autonomous agents to make successful decisions in a shared environment. Learning in multi-agent settings is, however, fundamentally more challenging than the single agent case as the agents need to interact with their environment and potentially with each other at the same time. Another issue with introducing another agent into this box world RDRL environment is that it could lead to single agent doing all of the work, while others are doing nothing. Thus, we make use of the "Shared Experience Actor-Critic" algorithm developed by Christianos et al [3], an extension of the off-policy advantage actor-critic(A2C) algorithm which updates the actor and critic parameters of an agent by combining gradients computed on the agent's experience with weighted gradients computed on other agents' experience, allowing all the agents to learn and co-evolve in a shared environment at the same pace; hence getting rid of the lazy agents and learning becomes a collective process.

Because relational deep reinforcement learning remains rarely explored within the multi-agent domain, our main objectives are to introduce explainability and relational representation learning to MADRL and explore the compatibility of the said framework and the SEAC learning algorithm. Although several limitation such as time and computational resources were encountered, the result has shown the the learning algorithm has been correctly implemented into the multi-agent setting, where attention weight could be visualised to gain an insight to the planning and reasoning behaviour of the agents under the context of relational learning.

II. Background and Related Works

This section reviews Relational Representation Learning and Self-Attention mechanism alongside the context of explainable RL, as well as the learning algorithms methods used in MADRL.

A. Explainable RL: Relational Representation and Attention

The understandability of an RL model depends on its transparency (its capacity to be understandable by itself) but also on human understanding. The ability to learn how to map input data to a multi-object disentangled representation is thought to be another next big leap towards a deep learning algorithms that is able to manipulate symbolic presentations. But in the absence of further structure, such black-box approach fails to explain the learned policy ("Why the agent took action x in state s?") in a human understandable way [8]. Futhermore, RL models often faces the issue low sample efficiency and poor transfer beyond the specifics of the training environment. Thus, a number of studies have attempted to overcome such limitations through the use of entity-based and symbolic representations [20]. According to Heuillet et al [8], our framework falls the category of representation-learning-based transparent algorithms, which focuses on learning abstract features that characterize data, in order to make it easier to extract useful information when building predictors. These learned features have the advantage of having low dimensionality, which generally improves training speed and generalization of deep learning models.

Our multi-agent RDRL model, which is based on the original DeepMind's paper [21] [20] draws inspiration from previous work that reasons about graphs and relations using neural networks [16] [15] to explicitly represent entities (nodes) and their relations (edges) with sets and graphs, as well as achieving relational reasoning using learned message-passing[20] [7][1] on how objects in complex systems interact, and attention mechanism [9] [17] [19][18], which performs non-local pair-wise computations with a shared function based on the salient (attended) aspects of an input. More previous work with relation nets and multi-head attention nets has shown the extracted non-local information can be used solve tasks that require relational reasoning. (Santoro et al.,2017; Palm et al., 2018; Santoro et al., 2018; Zambaldi et al.,

The paper [20] formalise this approach by defining 'Relational reasoning' as "manipulating structured representations of entities and relations using rules" and inductive bias as "the set of assumptions that the learning algorithm uses to predict outputs of given inputs that it has not encountered". The authors strongly supported that computation over graphs can achieve a strong relational inductive bias much superior than the other deep learning framework such as CNN and RNN and further expand such framework into the Deep RL domain. Their box world and StarCraft II experiments have shown that the attention weights could capture a link between entities (i.e. keys and locks of the same pixel value) through the utilisation of shared computation across entities. In comparison to the prior work of Wang et al.[19] which relies on *a priori* knowledge, RDRL [20] [21] is more dynamic to the particular salient relations.

However, relation presentation learning has rarely been explored in the DMRL, which inspires the current paper to bridge the gap between the two domains.

B. RL Algorithms

RL is generally a very broad field, so a distinction must first be made between model-free and model-based algorithms. In contrast to Model-based algorithms, Unlike Model-based algorithms, knowledge about the model is not a necessity in model-free algorithms, and their main objective is to directly optimise the policy or other value parameters. This type of objective-oriented approach can perform in a wide range of environments and is dynamic to their changes [2]. In this studies, we focus exclusively on model-free algorithms, which can be further divided in two fields: value-based approaches and policy-based approaches.

The goal of value-based approach is to obtain good estimates of the state and/or state–action pair value functions *V(s)* and Q(s, a). To convert the value functions to the actions, a fixed rule is applied to select optimal policy; for example, ϵ-greedy policy, the policy selects the action with the higher Q-value with probability 1 ϵ and a random action with probability ϵ. One example of this approach is the Q-learning algorithm [14]. On the other hand, the value functions in the policy-based methods are not required to be estimated as a parameterised policy (in the form of an action vector) could be used to represent a probability distribution of actions over states. Optimisation of this policy can be done via gradient ascent to reach an optima and defining an objective function. While it is not possible to differentiate the expected value of the reward, it is possible to do it with the policy, which makes gradient ascent possible. [2].

Actor-critic approach is last category of RL algorithms that derived from the intergration between policy-based and value-based approaches. The critic uses temporal difference (TD) methods to perform value function estimation while the actor, in accordance with the direction suggested by the critic, update its policy via policy gradient. [11]. This will be explained in theoretical detail in the Method section.

C. MARL Algorithms

In order to optimise policies for multiple agents with a cooperative or competitive relationship, Multi-Agent Reinforcement Learning (MARL) was introduced. It is very crucial for MARL algorithms to be stable and adaptable in a dynamic environment. This means that convergences are needed for the policies, and that each agent is able to adapt the behaviours of other agents [22]. The most basic approach would be to distinctively train each agent to maximise its individual reward while assuming other agents as a part of the environment. However, in this case, the basic assumption underlying reinforcement learning that the environment should be Markovian and stationary is violated. Thus, failure of standard algorithms designed for stationary Markov decision processes is often

presented in such scenarios [10]. =Also, one of the most common issues found in MARL is that agents learning at different rates often impedes exploration, leading to a less optimal policy selection [3]. Additionally, large amounts of resources such as training time and memory allocation are often the core requirement for training RL agents. To aid such limitations, in 2016, Mnih et al. [13] proposed an actor-critic architecture termed asynchronous advantage actor-critic (A3C), which is a lightweight approach to training multiple agent-environment instances concurrently. This approach effectively reduces the usage of memory allocation and training time, and has become a cutting-edge method in many AI games domain. Nonetheless, OpenAI later noticed that such asynchrony was not needed, and therefore, A2C, a deterministic and synchronous version of A3C was introduced. In A3C, because the global parameters are updated by each agent independently, there are possibilities that thread-specific agents would somehow perform with different policies which can hinder the update optimisation. To prevent such inconsistency, A2C's coordinator would impede the global parameters from updating until all parallel actors have finished their works, so that the all parallel actors can start from the same policy in the next iteration. The synchronised gradients update was also shown to improve the rate of convergence. As a result, we (as well DeepMind's) use A2C algorithm as the baseline algorithms for our studies.

Christianos et al [3] later introduced SEAC, which closely follow A2C using n-step returns and parallel sampled environments, which aims to tackle the problem of how agent learning at different rate often impede exploration, leading to inefficient policy selection. The algorithm works by sharing experiences between agents, while still allowing individual agents to have distinct policies, which leads to an enhancement in coordination and exploration. The study was done on 10 sparse-rewards learning tasks across four environment, in which it has shown to outperform several learning methods such as independent learning, shared policy training, and state-of-the-art MARL algorithms. The performance evaluation was done based on the efficiency and rewards returns. However, SEAC's limitation lies in it's behaviour to stack experience, which quickly lead to a large increase in batch size and slow computation time.

A2C, A3C, and SEAC all fall under the category of Distributed Reinforcement Learning in MARL context as these algorithms aim to effectively utilise large-scale computing resources for RL [4]. Similar to the previously mentioned asynchronous methods, IMPALA [5] and SEED RL [6] are off-policy actor-critic algorithms designed for distributing data collection across many actors while the optimisation is performed on a single learner. However, all the aforementioned approaches only share experience of multiple actors in order to accelerate learning of a single RL agent instead of addressing synchronisation in the multi-agent setting.

Another interesting approach is Centralised Training with Decentralised Execution (CTDE), which presume during the training phase that the data from all the agents is available for access by the learning algorithm to learn decentralised agent policies. [3]. CTDE algorithms such as MADDPG [12] allows learning of powerful critic networks through conditioning on joint observations and actions of every agents. What makes SEAC shines lies in the fact that it uses shared experience to reinforce positive action while MADDPG only takes agent's own tried actions into consideration. It does so without having to learn about the joint-action critics. Christianos et al.'s work [3] also shows that MADDPG is less optimal when comes to learning effective policies in sparse-reward environments compared to SEAC.

III. Theory and Method

This section provides detailed explanation of the multi-agent RDRL theoretical framework and the architecture. The environment is build with OpenAI Gym, which is an extension of Zambaldi et al. from Google DeepMind's mind [21] whose original source code was never released. The main framework of this project is built using PyTorch, which was chosen due to time limitation as the SEAC [3] official repository (https://github.com/semitable/seac) was also built with PyTorch. The model is trained on Google Virtual Machine with num_cpu = 6.

The project's code is available on the author's Github. [1]

A. Environment and Task Description

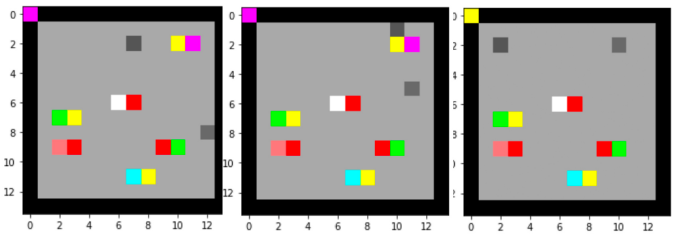

Fig. 1. Multi-Agent Box World environment

The Box-world environment consists of a 12 × 12 pixel room with keys and boxes randomly scattered. Inside the room are two agents, each represented by a single dark grey pixel, which can move in four directions: up, down, left, right. Each of the keys is represented by a single coloured pixel. The agent can pick up a loose key (i.e., the one that is not bounded to any other coloured box) by walking over it. Boxes are represented by two coloured pixels placed alongside to one another. The right pixel is the box's lock and its colour indicates which coloured key can be used to open that lock; the left pixel depict the content inside the box which cannot be retrieved until the box has been unlocked by the correct key. When an agent successfully retrieved to the key, the key is also shared to another agent.

Most boxes contain keys that can be used to open other boxes. One of the boxes contains a gem, represented by a

[1] https://github.com/Glassatlas/SEAC-BoxWorld

single white pixel (reward = +10). Obtaining the gem is the agent's goal. Each of the retrieved keys is displayed as a pixel in the top-left corner of the screen. The sequence of boxes that need to be opened to reach the gem in each level is unique. Opening a distractor box (reward = -1) leads to a dead-end as the key inside it can't be used to open any other box. The episode terminates when the agent open the wrong box or when the gem is retrieved. The user can control 3-parameters: 1) the number of boxes in the path to the goal, ; 2) the number of distractor branches; 3) the length of the distractor branches. In this study, distractor branching value is set to 1 for the sake of simplicity and time limitation.

B. Multi-agent RDRL Framework and Architecture

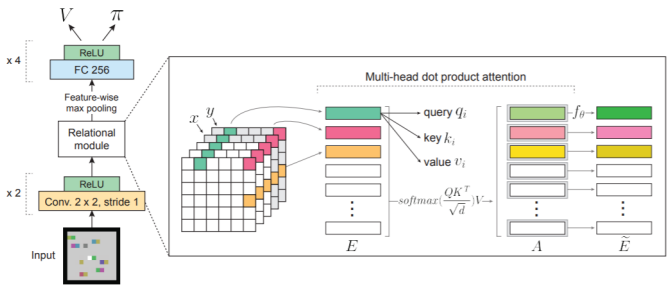

Fig. 2. The original RDRL architecture by Zambaldi et al.

The framework is composed of three mains parts: the Input module, the Relational module, and the Output module. The convolutional neural network is used because images (of the scene) are used as our input state. Our architecture follows the original, with the input module containing two convolutional layers with 12 and 24 kernels of size 2x2 and a stride of 1, followed by a rectified linear unit (ReLU) activation function. The output of this module is relayed to the relational module, which composes of multiple relational blocks with shared shared parameters. We uses 2 attention heads to produce the queries, keys and values, in which the embedding size is 64. The output updated entities are then aggregated via feature-wise max pooling function before being passed to 4 fully connected layers, which also uses ReLU downstream. The model then output the Policy logits, π of size 4 and baseline function, B of size 1 through linear projection.

1) Shared Experience Actor-Critic Algorithms: The original work [20][21] used A2C [13] and IMPALA [5] as their learning algorithms, where the agents creates an embedded state representation, S, from its input observation, which is then used to compute as output a policy, π (the "actor"), and a baseline value, B (the "critic").

In our multi-agent environment, we implemented the SEAC algorithm, which is based on the A2C algorithm that uses n-step returns and n parallel sampled environments. Below, we briefly explained 5 core equations that form the backbone of the SEAC framework. A more detailed explanation can be found in the original paper [3].

$$\mathcal{L}(\phi_i) = -log\pi(a_t^i|o_t^i;\phi_i)(r_t^i+\gamma V(o_{t+1}^i;\theta_i)-(V(o_t^i;\theta_i)) \quad (1)$$

$$\mathcal{L}(\theta_i) = ||V(o_t^i;\theta_i) - y_i||^2 with y_i = r_t^i + \gamma V(o_{t+1}^i;\theta_i) \quad (2)$$

Equation (1) and (2) defines the policy (π) loss and baseline value (V) loss for agent i in the most basic AC algorithms for multi-agent settings. Here, ϕ and θ are the parameters. In a normal setting, when on-policy training is preferred, RL algorithms takes into consideration only the individual agent's sampled trajectory to update the agent's network w.r.t. equation (1). And in order to use the trajectories of other agents as well as assuming that they are off-policy data, the authors had to define the loss for such off-policy policy gradient optimisation from a behavioural policy (Equation (3))

$$\nabla_\phi \mathcal{L}(\phi) = -\frac{\pi(a_t|o_t;\phi)}{(a_t|o_t}) \nabla_\phi log\pi(a_t|o_t;\phi)(r_t+\gamma V(o_{t+1};\theta)-(V(o_t;\theta))) \quad (3)$$

Lastly, equation (4) and (5) are simply extentions to equation (1) and (2). With equation (4) defining the extended policy loss which utilise the agent's own trajectories (i) along with the tracjectory of other agents (k). The Value loss is then updated based on this loss function (Equation (5)), where each agent is trained on on-policy data while also taking into account the off-policy data retrieved by other agents at every training step.

$$\mathcal{L}(\phi_i) = -\lambda \sum_{k \neq i} \frac{\pi(a_t^k|o_t^k;\phi_i)}{\pi(a_t^k|o_t^k;\phi_k)} log\pi(a_t^k|o_t^k;\phi)(r_t^k+\gamma V(o_{t+1}^k;\theta)-(V(o_t^k;\theta) \quad (4)$$

$$\mathcal{L}(\theta_i) = ||V(o_t^i;\theta_i)-y_i||^2+\lambda \sum_{k \neq i} \frac{\pi(a_t^k|o_t^k;\phi_i)}{\pi(a_t^k|o_t^k;\phi_k)} ||V(o_t^k;\theta_i)-y_k^i||^2 \quad (5)$$

The SEAC algorithm can be view simply as the extension of equation (1) and (2), which are the policy loss formulations for agent i with a value function minimising

Instead of using the experience of each agent's own sampled trajectory to update the agent's networks with respect to Equation (1), SEAC uses the trajectories of other agents while considering that it is off-policy data, i.e. the trajectories are generated by agents executing different policies than the one optimised. Correcting for off-policy samples requires importance sampling. The loss for such off-policy policy gradient optimisation from a behavioural policy can be written as

The authors extended the policy loss of the AC algorithm to utilise the agent's i own trajectories along with the experience of k other agents (equation 4)

The Algorithm for SEAC from the original study is presented below

2) Input Module: The framework begins with an image of a scene as input that get transformed into embedded state representation (spatial feature map) S which will later be converted into a set of entity vectors, E to be used as input to the relational module. Before feature-to-entity process happens, it was assumed that all the entities (i.e. pixels) must

Algorithm 1 Shared Experience Actor-Critic Framework

for timestep $t = 1 \ldots$ **do**
  Observe $o_t^1 \ldots o_t^N$
  Sample actions $a_t^1, \ldots, a_t^N$ from $P(o_t^1; \phi_1), \ldots, P(o_t^N; \phi_N)$
  Execute actions and observe $r_t^1, \ldots, r_t^N$ and $o_{t+1}^1, \ldots, o_{t+1}^N$
  for agent $i = 1 \ldots N$ **do**
    Perform gradient step on ϕ_i by minimising Eq. (4)
    Perform gradient step on θ_i by minimising Eq. (5)
  end for
end for

Fig. 3. SEAC Algorithms

have their position in space and the co-ordinate *x* ,*y* is assigned to each pixel value in the feature maps. This is done by tagging two additional channels dedicated specifically to the co-ordinates with the values are evenly spaced between -1 and 1. This is to preserve the the spatial coordinate structure of the feature map since remove of such spatial information could lead to issues in the downstream computation.

Next, the $m \times n \times f$ feature maps are converted into an $N \times$f set of *entity vectors*, E where $N = m \cdot n$. Each row of E correspond to a feature vector $S_{x,y}$ at particular coordinates. The feature vector can be thought of as slicing the feature map depth-wise. This is parallel to how the Non-Local neural network [19] works, in which the authors emphasised on the importance this feature-to-entity transformation as it allows for non-local computation between entities and learning to find relational clues independent of the distance in space and time.

3) Relational Module and Self-Attention: Analogous to the graph neural network [16], the model computes pairwise interaction $p_{i,j}$ between each entities before updating them based on the information about all of its interaction using multiple paralleled attention heads. The updated individual entity can be represented as a function $e = g_\theta(a_i^{h=1:H})$, where a_i denotes the accumulated interactions. This one-step relational computation closely follows Vasnani et al. [17] self-attention mechanism which uses Multi-head dot product attention (MHDPA). Here, the entities vector, E, gets represented as matrices consist of three core elements (vectors); Key, Value, and Query, denoted as *K, V, Q*. The process follows the equation below:

$$A = softmax(\frac{QK^T}{\sqrt{d}})V \tag{6}$$

where d is the dimensionality of Q and K we compute the dot-product between vector Q and K, which will represent the spatial proximity in the embedding space between the two vectors. The dot-product is larger if both vectors are closer together and vice-versa. The dot-product then gets normalised into attention weight through a softmax function, which is then passed for the pairwise interaction computation. The output of each MHDPA step, A, which consist of concatenated a_i, is then passed to the multilayer perceptron (MLP), producing updated entities, $\mathcal{E}$

4) Output Module: The final output of the network π (policy) and B (baseline value) are computed from the update entities received from the relational modules. Here, the dimensionality of $\mathcal{E}$ is reduced to f-dimensional vector via max-pooling over the entity dimension, where it then gets passed to the fully connected layer for the final computation. Eventually, the model is expected to converge, where relevant relationship between different entities are computed (**Figure 4**)

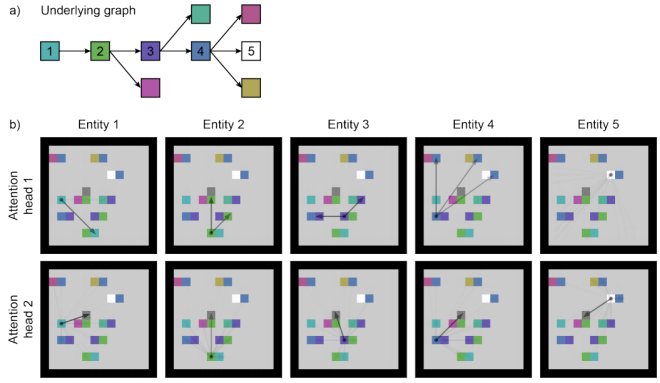

Fig. 4. Weight Visualization in single-agent environment by Zambaldi et al. [20] and the underlying graph of the optimal path.

IV. Results and Evaluation

In this section, we described the evaluation methods and present results of our trained model as well as the replicated baseline result from both Pytorch [2] and Tensorflow. [3] implementations. We compared our model to the baseline results which uses A2C based on the mean reward, episode length, and policy and value loss. We also implemented the *make_animation* function (in evaluate.py) which allows visualisation of the trained agents' action. Evaluation are done on both Non-Random and Random Environment configuration (detailed explanation on how the environment is generated available in the original paper [21]) with *level* = easy.

Note that our multi-agent model could only be trained on 10 million timesteps due to time and computational resource limitation, while the the baseline work [20] was trained at 1.4 billion timesteps, where it bagan to show an indication of learning at 200million timesteps. We attempted to replicate the baseline result using 6 CPU cores the distractor box branching factor set to 1 which was run at 65 million timesteps. The result shows an indication of learning at 30 million time steps (See Appendix A).

Because SEAC was implemented in PyTorch, we had to switch to a PyTorch implementation of the RDRL for compatibility. The baseline model was trained up to 18 million timesteps for 3 days. The model did not show indication of learning yet at this point, which is highly likely due to the low number of training timesteps.

For both random and non-random environment with SEAC, the policy and value loss plot (**Figure 6** and **Figure 7**) shows that both agents learn at similar rate, confirming that the

[2]https://github.com/mavischer/RDRL
[3]https://github.com/gyh75520/Relational_DRL

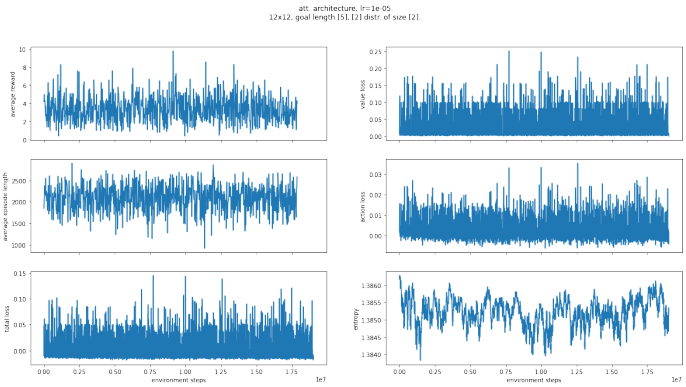

Fig. 5. Single Agent Box-world Random (PyTorch Implementation)

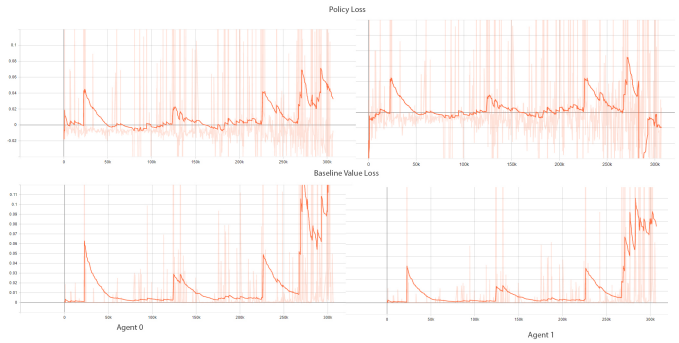

Fig. 6. Comparison between the two agents in Non-random environment trained at 6 millions timesteps

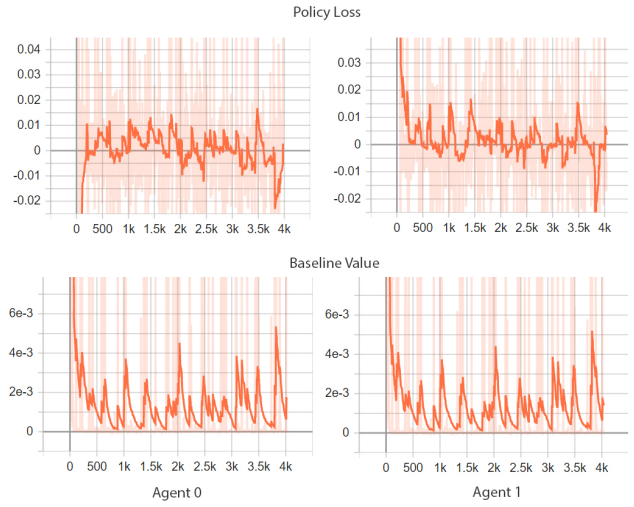

Fig. 7. Comparison between the two agents in Non-random environment trained at 3 millions timesteps

implementation take effects and the learned experience are shared among both agents. Note that 300k steps here does not refer to the actual environmental timesteps, which was at 6 millions when the training terminated.

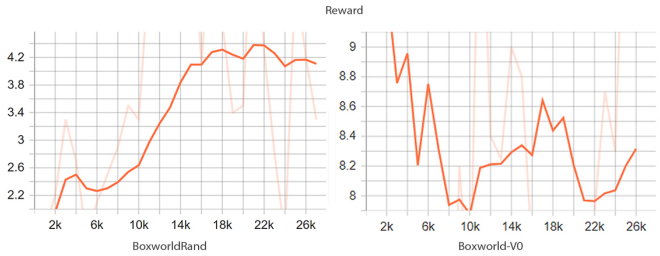

Fig. 8. Average reward trained at 3million timesteps

The results presented are inadequate in terms of performance, as expected, due to the lack of training resources and time. **Figure 8** shows poor learning capability of the agents at 3 million timesteps, although the agents in the non-random environment is shown to achieve a slightly better performance with the mean rewards between 8 and 9 while the other only achieved between the range of 2 and 4. This is to be expected due to the simpler nature of the static environment compared to the more complex randomly generated environment which presents the boxes at distinct spatial location (that the agents have never encountered before) each time.

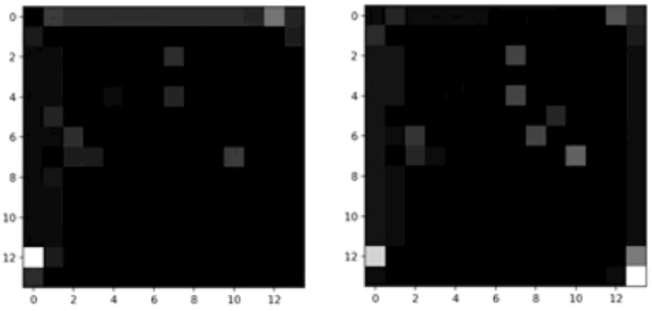

Fig. 9. Attentional weight visualisation

The visualised attention weights (**Figure 9**) shows how both agents attend to different entities within the environment while still resembling the similar pattern that stemmed from sharing of experience. The lighter colours denotes higher attention weight. Do recall each specific rows of the matrix A produced by equation (5) or **Figure 2** perform mapping onto to related objects in the observation space. Ideally, what we wish achieve is that for each single attention heads, each key attends mostly to the locks that can be unlocked with that key.

V. Discussion

There are several approaches for applying Actor-Critic algorithm to the multi-agent RL with the simplest one being Independent Actor-Critic (IAC) methods, where both agents

are entirely independent from one another in terms of policy network and experience. Going back to equation (1) and (2) in Section 2, the model directly optimised both equations while assuming another agent is simply another part of the environment. This scenario could lead to one agent doing all of the work while the other one does nothing at all, which creates a lazy agent. This very nature of the problem and the egocentric nature of each agent in our multi-agent RDRL framework are what make SEAC very attractive, since it allows experience sharing between agents. The update of the parameters are done by integrating gradients computed on one agent's experience with weighted gradient computed from others', which allows both agents to learn relational information between entities in the sparse reward environment at the same pace, forcing both agents to contribute to the task instead of diverging the learning progress. This is shown in the policy and baselines value loss plot between both agent in **Figure 6** and **Figure 7**, which entails akin learning behaviour. SEAC allows the agents to learn concurrently akin policies but not restricted to identical policies, which, in an ideal situation where training time is sufficient to train the model optimally, could lead to enhanced exploration and co-ordination behaviour of the two agents (i.e. whichever agent is closer can go ahead and unlock the attended box).

The visualised attention weight (**Figure 9**)also provide an insight to how the agent may attend to different entities in the observation space. In our case, the attention weights may seem slightly random and attend to some blank location instead of boxes. This is because our model was not trained optimally. However, ideally, it is expected that the agents could learn an abstract representation of what it means to "unlock" similar to *unlock(key,lock)*, in which such function can be used to generalise to key-lock combinations that they have never encountered before. Because this is an multi-environment extension with a shared-experience approach, we would expect that the collaborative effort between the two active agents could lead to faster convergence, higher final returns, and shorter episode length.

A. Limitation and Future Implication

SEAC seemed to faced a slower computation time due to its nature to stack experience. A paper by Christianos et al., [4] suggests the use of Selective Parameters Sharing (SePS), a newer approach to automatically identify agents whom sharing parameters could be beneficial to via segregating them based on abilities and goals. This method may impede the exploration benefits but could allow much higher number of agents. Thus, the implementation of SePS and increasing the number of agents should be attempted in the future work.

The learning algorithm in which SEAC was nased on should not be limited to A2C but should be extended to other AC methods as well (i.e. PPO, IMPALA). This was attempted but incomplete. Due to limited time, we also did not have a chance to test different hyperparameters which could lead to a more optimal learning behaviour.Thus, one of the scopes of the future work could be to improve the choice of the hyperparameters as well as increasing the training timesteps to at least 200 million.

A small implementation error was later found at max-pooling layer at the final phase of training in the PyTorch version of RDRL, which reduces the output kernel size by 1. This could negatively affect the performance of the model. Max pooling is an important process which aids in the reduction of the dimensionality of the updated entities matrices into an f-dimension vector which is used to computer the baseline value and policy. A correction should lead to a significant improvement in the model performance.

This studies was carried out during the pandemic, in which several limitations were faced. These include the lack of computational resources and the delayed administrative responses, which lead to a slower than usual progression of the project. The author attempted the training on a single CPU Alienware R5 Laptop which was presented with a memory leakage issue mid-training. Thus, Google Virtual Machine was later used to train the model as the last resort. Training time was greatly limited and the baseline result could not be replicated as planned. The same reason applies to the training of the Multi-agent RDRL, which could only reach a few million steps.

This paper was written with minimal supervision due to the author's last minute decision to make major changes to the research scope, so an expert opinion maybe lacking and the discussion of results is based only on the author's own understanding of the topic.

VI. Conclusion

Understanding relationship between entities is often considered as one of most important element in reasoning AI. In this studies, we attempted to implement the SEAC learning algorithm into the multi-agent extension of the DeepMind's Box World. The objective of the study is to study the relational learning/reasoning behaviour of the two agents in a multi-agent collaborative settings. This study was not aimed to achieve a state-of-the-art results, but to shine a light for future research on the possibility of combining relational learning with MADRL. This is the first known attempt to extend the box-world environment to a multi-agent settings, as well as the first to implement SEAC in a relational learning task. Our implementation is proven to be successful, although many limitations have been issued, which lead to poor results. Thus, future direction has been suggested to either improve the current performance of the model through hyperparameters optimisation or attempt a wider exploration of SEAC implementation on other MADRL algorithms.

References

[1] Peter W Battaglia et al. "Interaction networks for learning about objects, relations and physics". In: *arXiv preprint arXiv:1612.00222* (2016).

[2] Lorenzo Canese et al. "Multi-Agent Reinforcement Learning: A Review of Challenges and Applications". In: *Applied Sciences* 11.11 (2021), p. 4948.

[3] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. "Shared experience actor-critic for multi-agent reinforcement learning". In: *arXiv preprint arXiv:2006.07169* (2020).

[4] Filippos Christianos et al. "Scaling Multi-Agent Reinforcement Learning with Selective Parameter Sharing". In: *arXiv preprint arXiv:2102.07475* (2021).

[5] Lasse Espeholt et al. "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1407–1416.

[6] Lasse Espeholt et al. "Seed rl: Scalable and efficient deep-rl with accelerated central inference". In: *arXiv preprint arXiv:1910.06591* (2019).

[7] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[8] Alexandre Heuillet, Fabien Couthouis, and Natalia Dıaz-Rodrıguez. "Explainability in deep reinforcement learning". In: *Knowledge-Based Systems* 214 (2021), p. 106685.

[9] Yedid Hoshen. "Vain: Attentional multi-agent predictive modeling". In: *arXiv preprint arXiv:1706.06122* (2017).

[10] Shariq Iqbal and Fei Sha. "Actor-attention-critic for multi-agent reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2961–2970.

[11] Vijay R Konda and John N Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[12] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *arXiv preprint arXiv:1706.02275* (2017).

[13] Volodymyr Mnih et al. "Asynchronous methods for deep reinforcement learning". In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.

[14] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[15] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs". In: *International conference on machine learning*. PMLR. 2016, pp. 2014–2023.

[16] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[17] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[18] Petar Veličković et al. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[19] Xiaolong Wang et al. "Non-local neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7794–7803.

[20] Vinicius Zambaldi et al. "Deep reinforcement learning with relational inductive biases". In: *International Conference on Learning Representations*. 2018.

[21] Vinicius Zambaldi et al. "Relational deep reinforcement learning". In: *arXiv preprint arXiv:1806.01830* (2018).

[22] Lixiang Zhang et al. "Multi-agent reinforcement learning by the actor-critic model with an attention interface". In: *Neurocomputing* (2021).

APPENDIX

Appendix A

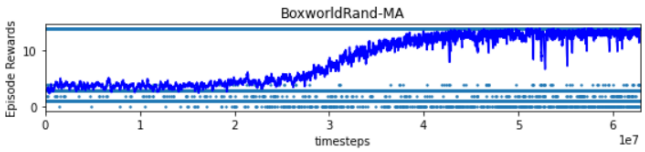

Fig. 10. Single Agent Box-world Random-easy (Tensorflow Implementation)

Appendix B

```python
@dataclass
class AlgoConfig:
    lr: float = 3e-4
    adam_eps: float = 0.001
    gamma: float = 0.99
    use_gae: bool = False
    gae_lambda: float = 0.95
    entropy_coef: float = 0.01
    value_loss_coef: float = 0.5
    max_grad_norm: float = 0.5

    use_proper_time_limits: bool = True
    use_recurrent_policy: bool = False
    use_linear_lr_decay: bool = False

    seac_coef: float = 1.0

    num_processes: int = 4
    num_steps: int = 5

    device: str = "cpu"
    base: Optional[NNBase] = DRRLBase
```

Fig. 11. Algorithm configuration used