Autoencoders to enhance Multi-Agent Coordination in a Tightly-Coupled Domain

Enna Sachdeva
Oregon State University
Corvallis, Oregon - 97331
sachdeve@oregonstate.edu

Aashish Adhikari
Oregon State University
Corvallis, Oregon - 97331
adhikara@oregonstate.edu

Ashwin Vinoo
Oregon State University
Corvallis, Oregon - 97331
vinooa@oregonstate.edu

Abstract

Deep Reinforcement learning has been shown to address several multi-agent coordination tasks. However, with the increase in the number of agents as well as in tightly-coupled settings, learning optimal joint policies becomes challenging. In this paper, we address these 2 problems in a partially-observable rover domain environment using Deep Deterministic Policy Gradient (DDPG) and autoencoders. We obtain decentralized optimal policies with each agent trained using independent DDPG learning algorithm. Further, we compress the state space using autoencoders to test the efficacy of the latent representation of the state in finding the optimal policies in a tightly-coupled multiagent environment. We further compare results with 2 different latent state dimensions of 5 and 10, with that of the complete state representation of dimension of 40, and our results show that latent representation of 10 performs equally well as that with complete state representation.

1. Introduction

The advent of deep learning has had a significant impact on reinforcement learning over time since it provides flexible models to learn latent features, value functions and policies[1], [2]. This combination of deep learning and reinforcement learning is significant to address problems with high dimensional state-space due its ability to learn different levels of abstraction from high-dimensional input features [17]. This has led it to solve several high dimensional single-agent problems like the well-known Atari games. Several works have been done to extend reinforcement learning algorithms for multi-agent systems. Learning in a multi-agent setting is not a trivial problem, however [9].

Learning in a multi-agent setting presents key core challenges- 1) Non-Stationarity: Since multiple agents are learning simultaneously, the environment observed by an agent is no longer stationary, which makes the states non-markovian [9]. 2) Partial Observability: Since sensor data is not accurate, the environment perceived by an agent is partially-observable. 3) Scalability: The increase in the number of agents leads to an exponential increase in the size of the state of the system. [10]. Further, in real world, the data from the sensors used are high-dimensional due to high resolutions used and thus, successfully scaling reinforcement learning is crucial to building artificially intelligent systems that can tackle real world tasks. The complexity further exacerbates in a tightly-coupled domain, which requires mutual dependence of robots on each others' actions [3], [6].

In this paper, we intend to address these key challenges in a coordinated multi-agent environment using deep learning. First, to address multi-agent coordination in a partially-observable tightly-coupled domain, we use deep deterministic policy gradient(DDPG) algorithm [11]. In DDPG, each agent has actor and critic neural networks to learn optimal policies to maximize rewards in a co-operative multi-agent environment. Further, to address the scalability issue, we train autoencoders to compress the state space and then use DDPG on top of it to observe the impact of compressing the state space representation. Our hypothesis is that the compressed state representation can encode the relevant information from the state required to obtain optimal policies for each agent while removing the noisy information from the state. We show that our results validate this hypothesis to a good extent and provide a research opportunity to enable us to continue to work further in this direction.

The rest of this paper is organized as follows. In section 2, we provide an overview of the related work and the background for multi-agent systems, Deep Deterministic Policy Gradient (DDPG),and autoencoders. In section 3, we discuss the rover domain environment used for the experiments conducted in the subsequent sections. In section 4, we highlight the methodology we used to verify our hypothesis - how to train agents using DDPG with both- the complete state space and the compressed latent state space. Finally, in section 5, we highlight the experiments performed, the

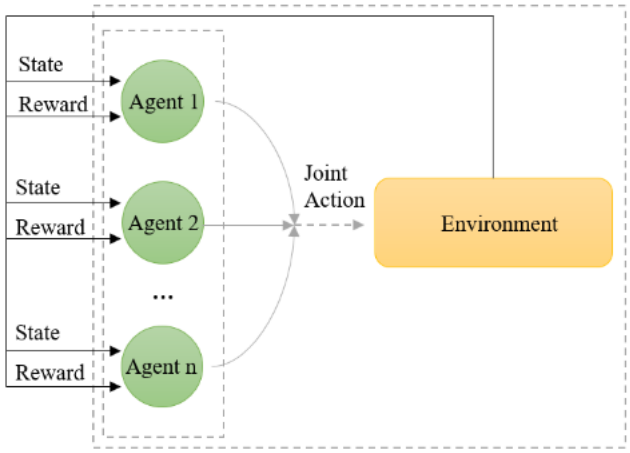


Figure 1: Deep Reinforcement Learning in a Multi-Agent Setting

results attained, and end the paper with our conclusions and future work.

2. Background and Related Literature

2.1. Multi-Agent System

A multi-agent system is a system of autonomous agents interacting with the environment with a goal of finding optimal policies to collaborate or compete in the environment as illustrated in Fig. 1. The agents observe the environment, take actions, get rewards, and optimize plans to maximize the global reward. The agents do not have any information about the overall global reward that they are trying to maximize, and also, they cannot communicate with each other. They find optimal policies independently to maximize the global reward together to solve the coordination problem [12].

2.2. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is a popular model-free reinforcement learning algorithm for learning in continuous high-dimensional actions spaces [11],[7]. DDPG uses an actor-critic architecture maintaining a deterministic policy (actor) $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and an action-value function approximation (critic) $\mathcal{Q} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The critic approximates the actor's action-value function $\mathcal{Q}^{\pi}$. Both the actor and the critic are parameterized by (deep) neural networks with θ^{π} and $\theta^{\mathcal{Q}}$, respectively. Separate copies of the actor π' and the critic $\mathcal{Q}'$ networks are kept as target networks for stability. These target networks' parameters are updated periodically to match the actor π and the critic networks $\mathcal{Q}$, modulated by a weighting factor τ.

A noisy version of the policy: $\pi_b(s) = \pi(s) + \mathcal{N}(0, 1)$ where $\mathcal{N}$ is a temporally- correlated noise generated using the Ornstein-Uhlenbeck process [16] is used for training. After each action, the tuple (s_t, a_t, r_t, s_{t+1}) containing the current state, actor's action, observed reward,and the next state respectively is saved into a **cyclic replay buffer** $\mathcal{R}$. The actor and the critic networks perform a parameter update by randomly sampling mini-batches from $\mathcal{R}$. The critic is trained by minimizing the loss function:

$$L = \frac{1}{T}\sum_i (y_i - \mathcal{Q}(s_i, a_i|\theta^{\mathcal{Q}}))^2 \text{ where } y_i = r_i + \gamma \mathcal{Q}'(s_{i+1}, \pi'(s_{i+1}|\theta^{\pi'})|\theta^{\mathcal{Q}'})$$

The actor is trained using the sampled policy gradient:

$$\nabla_{\theta^{\pi}} J \sim \frac{1}{T}\sum \nabla_a \mathcal{Q}(s, a|\theta^{\mathcal{Q}})|_{s=s_i, a=a_i} \nabla_{\theta^{\pi}} \pi(s|\theta^{\pi})|_{s=s_i}$$

The sampled policy gradient with respect to the actor's parameters θ^{π} is computed by backpropagation through the combined actor and critic network[8].

2.3. Autoencoder

An autoencoder is an unsupervised neural network that takes an input x, tries to learn an identity function $f_{W,b}(x)$ and returns x', an approximation of x[13]. By applying constraints on the network, interesting byproducts of the network can be achieved. Thus, autoencoders have found significant usage in denoising, anomaly detection, and compression among others. In case of compression, given an input sample of N dimensions, a model trained to reconstruct the input after passing it through a hidden layer l_m with M nodes such that M $<$ N learns a compression of the data to a latent space of M dimensions[14][15]. The network that takes the original data as input and gives out the compressed latent representation of l_m dimensions is called the encoder. The network that takes this latent space representation of the data to approximate the original data is called a decoder. The loss function, called the reconstruction loss, penalizes the network for reconstructing an output different from the input to the encoder. As a consequence, the encoder discards the information that is irrelevant for the reconstruction of the data and the decoder learns to reconstruct the original data using only the latent representation.

3. Rover Domain

We consider a fully-cooperative multi-agent "Rover Domain" environment [4], [3],[5] in which a team of autonomous rovers explore an unknown environment, searching for Points of Interest (POIs) with no active communication abilities. Each rover has 3 sensors, one to detect Points of Interests (POIs), second to detect other rovers, and third to detect boundary walls of the environment. Each rover sees the world using these sensors and learns to coordinate using only its local observation of the environment in order to guarantee robust and distributed operations. The

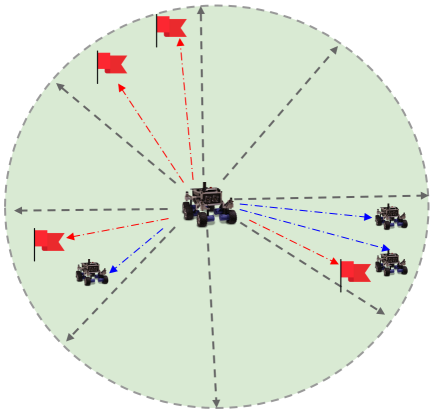


Figure 2: Sensor resolution for each rover sensor. The space of 360° around the sensor is divided into several sectors. It can observe POIs and rovers within this sector, within its observation radius, shown in green. Also, rover has a wall sensor, with which it can detect the distance of the 4 walls around it.

resolutions of these sensors can vary, which divide the entire 360° into several sectors. The environment is partially-observable as a rover just estimates the relative distance to other rovers and POIs but not their absolute positions and not the direction they are heading to as a consequence of the observation range limit of the sensors and the fact that the sensors provide us with densities of POIs and other rovers in each sector, but not their absolute positions. The state representation for each rover encodes the relative distance between it from other rovers and POIs within some observation radius in each of its sectors as illustrated in Fig. 2. In addition, it also contains four values to indicate its distance from four boundary walls. The detection of the rovers, the POIs, and the boundary walls in each of the sectors q of a rover j is used to generate the state input vector S_j as follows:

$$S_{rover_{j,q}} = \sum_{j' \epsilon J_q} \frac{1}{L_{j'}}$$

$$S_{POI_{j,q}} = \sum_{i \epsilon I_q} \frac{V_i}{L_i}$$

$$S_{walls} =< d_0, d_1, d_2, d_3 >$$

$$S_j =< S_{rover_{j,q}}, S_{POI_{j,q}}, S_{walls} >$$

where,
J_q is the set of observed robots in the sectors q,
$L_{j'}$ is the relative distance from robot j to robot j'
I_q is the set of observed POIs in quadrant q,
L_i is the relative distance between robot j and POI i
S_j is a hot vector of 2*n values (2 for each of the n sectors).
d_0, d_1, d_2, d_3 are the distances of rover from the 4 walls.

In tightly-coupled settings, agents will only receive a reward when a POI is observed simultaneously by at least the minimum number of agents required to observe the POI. Consider a coupling of 2 where an agent (agent A) infiltrates the activation radius of a certain POI. It does not get a reward unless there is another agent (agent B) within the same POI's activation radius simultaneously. There is no partial reward for agent A in case another agent is not within the activation radius as mentioned above. This makes this domain quite interesting as well as challenging to solve. Once both the rovers are within the activation radius simultaneously, that POI is considered observed and both agents will receive the same reward, given by the following equation.

$$R = \frac{V_i}{\sum_{j \epsilon J_q} \frac{1}{l_i j}}$$

where,
V_i is the activation value of POI i,
$l_i j$ is the distance of rover j from POI i, within the activation region of rover j,

The action space is continuous and the action for each rover is given by the change in distance along x and y axes, $< d_x, d_y >$

4. Methodology

To address the issue of multi-agent coordination in a tightly-coupled domain, this work utilizes a deep reinforcement learning algorithm called DDPG for the rover domain with continuous action space, as shown in Fig, 3. Further, each state is compressed to a latent state using autoencoders so as to scale the algorithm for highly complex environment with many agents, each with very high resolution sensors, as shown in Fig. 4. The implementations of both of these methods are discussed in the subsequent subsections.

4.1. DDPG for Coordinated Multi-Agent System

For the multi-agent cooperative environment, we consider each agent to be a DDPG agent with its own actor and critic networks. The actor for each agent has the state information corresponding to that agent, and it gives the output action (d_x, d_y) for that agent. In this distributed setting, each agent takes its own state information from the environment and outputs the required action. The actor and critic

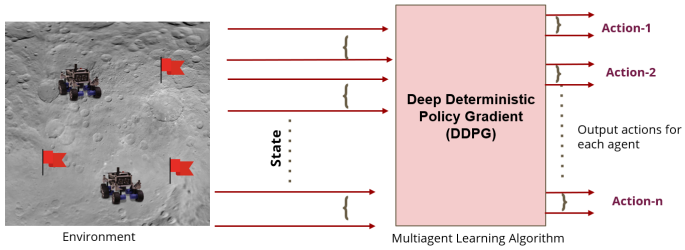


Figure 3: Architecture for a Multi-Agent Setting where the state of each agent goes to the individual DDPG network, and the network gives the output action $< d_x, d_y >$

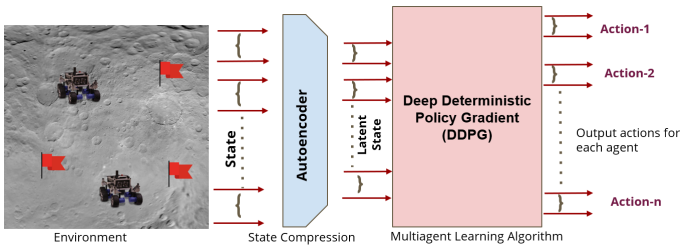


Figure 4: Architecture for a Multi-Agent Setting with encoded states where the state of each agent is encoded before feeding to the individual DDPG network, and the network gives the output action $< d_x, d_y >$

are in fact two neural networks. All agents have a common replay buffer, storing experiences of actions for each agent. Sharing the replay buffer increases the diversity of experiences for all the agents. This way, the learned behavior of one agent does not depend only on its experiences it, but also on the experiences of all the agents which are getting trained. The architecture for the multi-agent DDPG is shown in Fig. 5.

4.2. DDPG with Autoencoders for Scalability

To address the problem of an increase in the state space dimensions with the increase in the number of agents in a cluttered environment, we use the latent state representation of the state space for the DDPG network, instead of using the original state and eventually learn the optimal policies.

5. Experiments and Results

5.1. Environment Setup

To test the performance of each of the approach discussed above, we use a rover domain environment with 4 rovers and 10 POIs. The environment is a continuous space of 30x30 (30 units width and 30 units length). While training, the Points of interests are randomly initialized and the rovers are initialized at the center of the world at each episode. The rovers gradually search for the POIs. The rovers have a common speed constraint of 1 unit per time step and can only observe a POI when they are within a observation distance of 20 units from the POI. In this experiment, we consider a coupling factor of 2 i.e., each POI needs 2 rovers to be considered observed. This means that each rover will get a reward by observing a POI only when there is another rover which is observing the same rover simultaneously within the activation distance of the POI. The activation distance of a POI is the radius around it within which a rover must exist to be considered as observing this particular POI. Here, we take an activation distance of 4 units.

The 3 sensors of each rover, corresponding to observing POIs, rovers, and boundary walls are assumed to have a resolution of 20°, i.e, the 360 degree space around a rover is divided into 18 equal sectors of 20° each. The rover sensors are configured to detect the density of points of interest (POIs) and other rovers. In addition, the wall sensors can detect the distance to the four walls of the world. Hence, we have a total state space of 40 dimensions, (18 POI density values + 18 rover density values + 4 distances to the walls). The maximum reward that can be attained in this environment setup is 10, i.e to observe 10 POIs.

5.2. Implementation Details: DDPG with Complete State Space

For the first set of experiment, we used DDPG with input state of 40 dimensions, which is the actual dimension of the input state for each agent. The Actor model is a neural network with 2 hidden layers with size of 512 nodes each, and Tanh non- linearity is used for each layer. The output layer consists of 2 nodes to give 2 dimensional actions d_x, d_y for each rover at each time step t. A mini-batch of size 156 is taken from the replay buffer while training these networks.

The Critic model is a neural network with 2 hidden layers with 512 and 1024 nodes respectively, and Tanh nonlinearity is used in each of these layers. There are 2 inputs to the critic: one is the state of the agent, second is the action vector output obtained from actor network. Both these inputs to critic are independently performed with "linear + Tanh" operation giving an output of dimension 512 for each. These 2 outputs are concatenated to form a hidden layer of size 1024, which then gives a target action vector of 2 dimensions $< d_x, d_y >$. This is shown in Fig. 5.

The learning rate for both the actor and the critic is $1e-4$ with soft update of target set to $1e-3$. Gamma discount is 0.99 with weight decay set to 0. The replay buffer has a size of 1e8. Batch size is 512 with maximum time steps of 50 in each episode. A large replay buffer is crucial for successful learning. The number of time steps plays an important role as the agent needs to learn for enough time steps to have a good balance between exploitation and exploration. The OrnsteinUhlenbeck noise factor of 0.5 takes care of main-

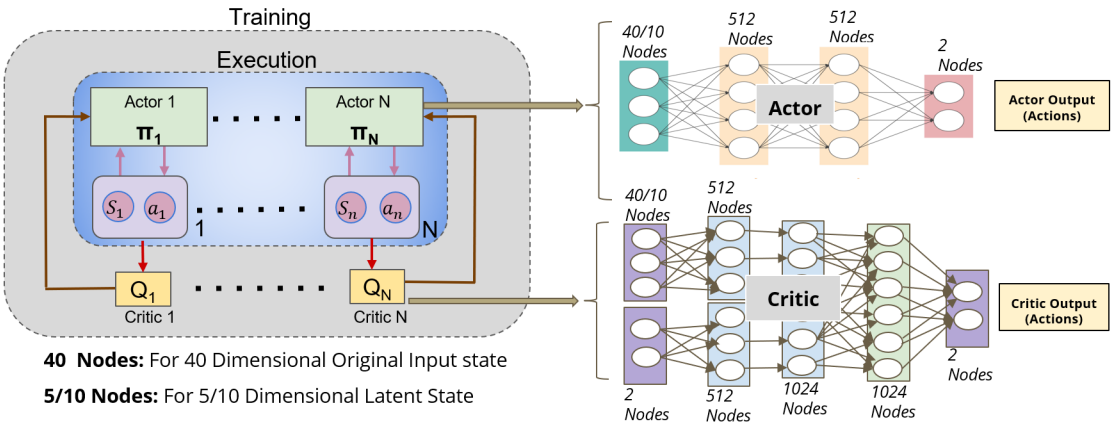


Figure 5: DDPG Architecture for a Multi-Agent Setting, where each agent has its own actor and critic networks. The neural network architecture for the actor and critic networks are shown in the right.

taining the trade-off between exploration and exploitation. With these settings, the result for 4 rovers, 10 POIs with a coupling of 2, is shown in Fig, 8. The blue plot in the figure shows the global reward across episodes, and the decentralized learning performs well in rover domain and attains a reward close to the oracle value of 10. Therefore, we could successfully evaluate the performance of the DDPG in rover domain. Further, we tested the same decentralized planning approach with compressed states, as discussed in subsequent sections.

5.3. Implementation Details: DDPG with Compressed State Space

5.3.1 Autoencoders Architecture

The Autoencoder architecture that we used for the experiments is shown in Fig. 7. Table 2 summarizes the network parameters of the autoencoder used for training in the experiments. The hyper-parameters shown were tuned accordingly.

Table 1: Reconstruction Loss

Latent Dimension	Training Loss	Testing Loss
5	2.5334	2.9522
10	2.5588	3.0911
20	1.5401	1.7943

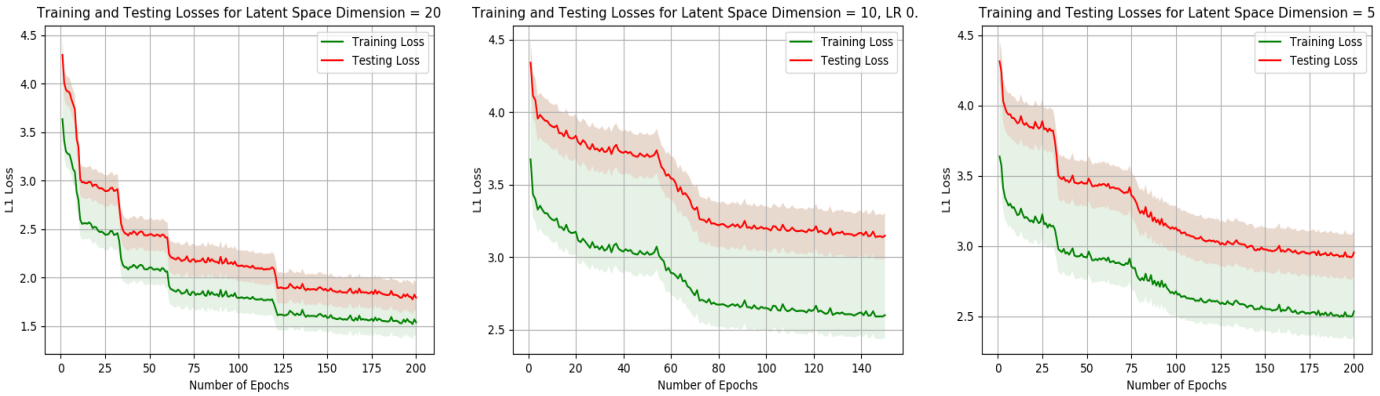
5.3.2 Data Preprocessing for training Autoencoder

The data was generated with random initialization of POIs and rovers in the 30x30 world, and then rovers are made to move in the world with random set of stochastic policies π. The generated data is quite sparse as most of the sectors of the sensors contain no rovers or POIS. This makes training a bit difficult. Further, there is an inherent redundancy in the data, as the next state of the rover (after it has taken an action) is not very different from the current state, due to the constraint on the actions (limited to a maximum of 1 unit along x and y axes). Also, the data from testing set is quite similar to that in the training set, since the trajectory taken by the rovers while generating the data during training, can be similar to that while testing. This makes the network easier to overfit to the data and the data in training and testing set become very similar. To eliminate such issues, following measures were taken.

- Data redundancy was eliminated and it was assured that only unique samples exist in the dataset, for both training and testing. This leads to a better performance of the autoencoder in terms of L1- loss.
- Training-testing ratio is 8:2.
- The samples were shuffled in random during each epoch of training to reduce the chances of being overfitting to data samples which are generated from subsequent states, and have very high chance similar to each other corresponding to adjacent positions in the rover domain for a rover after one single time step, though unique.

Three different latent representations of size 5, 10 and 20 were used and Table 2 summarizes the reconstruction loss for these compressed-state representations after training. The corresponding results are shown in Fig. 6.

Further, to test the the performance of the encoded state representation in the rover domain, we used the state representation of size 5 and 10. The performance of DDPG using encoded information is shown in Fig. 8. The red and the green plots show the performance for latent space of 5 dimension and 10 dimensions, respectively.



((a)) ((b)) ((c))

Figure 6: Autoencoder training and testing losses decrease across training episodes. The plot shows the training and testing losses for a (a) latent dimension of 20, (b) latent dimension of 10, (c) latent dimension of 5

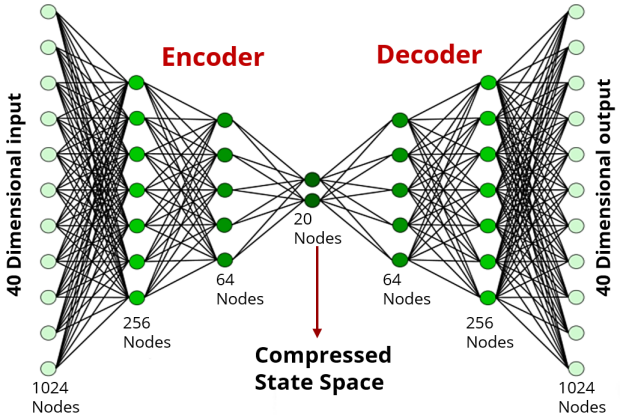


Figure 7: Architecture of the Autoencoder used to compress the state space to latent dimensions of 5, 10 and 20

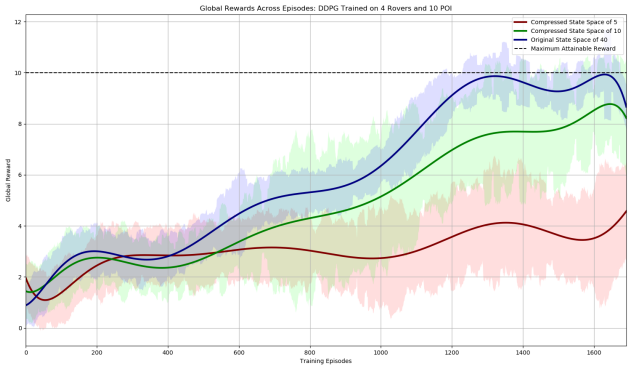


Figure 8: Global rewards across training episodes with different latent representation dimensions. Red represents the state dimension with latent space of 5, reaching a global reward of 5; green represents 10, reaching a global reward of 8; blue represents the original state dimension of size 40, reaching a global reward of oracle value 10

In Fig. 8, we can see that with the latent space representation, our decentralized planning approach using DDPG learns to optimize the reward in a tightly-coupled domain. This illustrates that the rovers are able to learn to observe POIs and coordinate their tightly-coupled actions even with the latent state space representation of $1/4^{th}$ the size of the complete state space representation which tends to address the scalability issue. This is illustrated in the paths of the rovers shown in Fig. 9 (middle figure), where the robots are able to observe all the POIs and eventually get a reward of 10 even with a latent state space of 10. This eliminates the need to use the latent dimension of more than 10 for this environment. However, the average maximum reward across several episodes drops down to 8 (as shown in Fig. 8)in the presence of OU noise parameter in DDPG algorithm during training. This noise enables the agents to explore the environment, thereby resulting in a reward less than 10 in some of the episodes, which brings the average to less than 10 (closer to 8) with latent state dimension of 10. Nonetheless, the noise is not present during the testing phase(complete exploitation), consequently giving a reward of 10 for the latent dimension of 10. This perfectly aligns with our hypothesis that the encoded information can be useful in learning policies in complex multi-agent settings.

While the encoded state dimension of 10 serves the purpose, the latent space of 5 is not able to perform that well, as shown in Fig. 8. Despite that, we look forward to improve its performance with more number of epochs for learning or with latent state space lesser than size of 10. These results

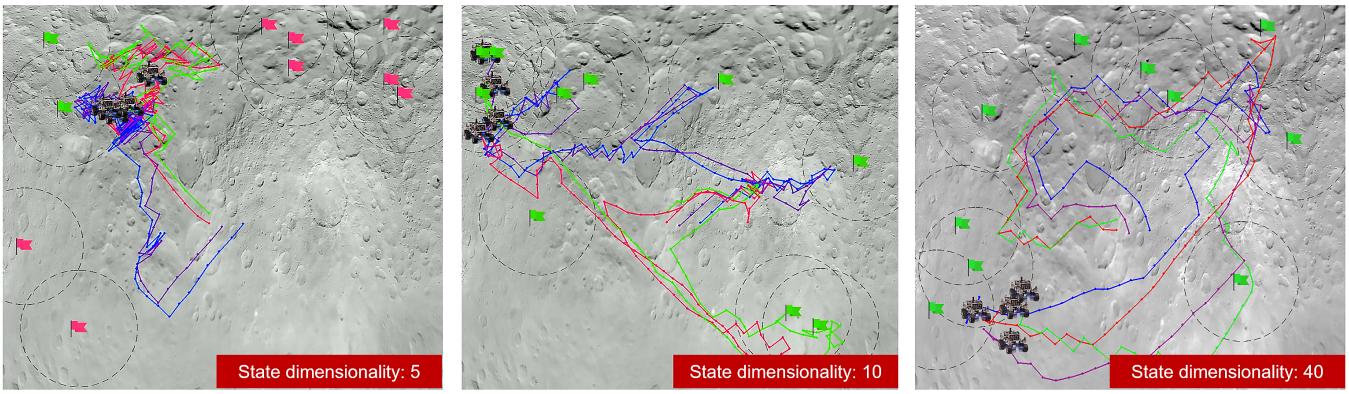


Figure 9: Path traced by a 4-rover multi-agent team using a compressed state of size 5 (left), a compressed state of size 10 (middle) and the original-40 dimensional state space (right)

Table 2: Network Parameters of the Autoencoder

Parameters	**Values**
Encoder Architecture	$1024 \rightarrow 256 \rightarrow 64 \rightarrow 32 \rightarrow$ Latent Dimension (5 or 10 or 20)
Decoder Architecture	Latent Dimension $\rightarrow 32 \rightarrow 64 \rightarrow 256 \rightarrow 1024$
Activation Function	Leaky ReLU
Learning Rate	0.0001
Weight Decay	0.00001
Momentum	0.5
Batch Size	64
Reconstruction Criterion	L1 Loss
Optimizer	ADAM
Number of Epochs	200

motivate us to continue working in this direction with different types of architecture for the autoencoder to achieve better performance for highly complex environments. Further, it helps us to identify the impact of encoded information on system performance and how it can be useful to enhance scalability in highly complex multi-agent domains.

5.3.3 Video Demonstrations

DDPG trained on a compressed state space of size 5:
video 1
DDPG trained on a compressed state space of size 10:
video 2
DDPG trained on the original state space of size 40:
video 3

6. Conclusion and Future Work

In this paper, we addressed 2 key challenges of multi-agent reinforcement learning in a partially observable environment using deep learning- 1) Tight-coupling in a cooperative multi-agent setting, and 2) Scalability issue as a consequence of high dimensionality of state space. To address tight-coupling in a decentralized learning approach, we implemented each agent with its own DDPG network and trained it to learn decentralized policies given the state of the system. Further, we trained an autoencoder to compress the input state to a latent representaion and observed that with the reduction in the dimension of state to $1/4^{th}$ of the original dimension, the encoder was able to encode the information as efficiently as the original dimension to learn optimal policies in an cooperative multiagent environment of 4 rovers and 10 POIs. For future work, we shall make the environment harder by introducing adversarial agents in a competitive environment. We also look forward to generate counterfactuals using encoded state information for reward-shaping in a tightly-coupled environment.

References

[1] Li Deng and Dong Yu (2014), "Deep Learning: Methods and Applications", Foundations and Trends in Signal Processing: Vol. 7: No. 34, pp 197-387. http://dx.doi.org/10.1561/2000000039 1

[2] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, 2018. 1

[3] Analyzing and visualizing multi-agent rewards in dynamic and stochastic domains, Agogino, Adrian K., Tumer, Kagan, Autonomous Agents and Multi-Agent Systems, 2008, 8 October 01 17 2 1573-7454 1, 2

[4] Lowe, Ryan, et al. "Multi-agent actor-critic for mixed cooperative-competitive environments." Advances in Neural Information Processing Systems. 2017. 2

[5] Gupta, Jayesh K., Maxim Egorov, and Mykel Kochenderfer. "Cooperative multi-agent control using deep reinforcement learning." International Conference on Autonomous Agents and Multiagent Systems. Springer, Cham, 2017. 2

[6] Adrian Agogino and Kagan Turner. 2005. Multiagent Reward Analysis for Learning in Noisy Domains. In Proceedings of the fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '05). ACM, New York, NY, USA, 81-88. DOI: https://doi.org/10.1145/1082473.1082486 1

[7] De Asis, Kristopher, et al. "Multi-step reinforcement learning: A unifying algorithm." Thirty-Second AAAI Conference on Artificial Intelligence. 2018. 2

[8] Wang, Ziyu, et al. "Sample efficient actor-critic with experience replay." arXiv preprint arXiv:1611.01224 (2016). 2

[9] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529. 1

[10] Liu, Jyi-Shane, and Katia P. Sycara. "Multiagent coordination in tightly coupled task scheduling." Proceedings of the Second International Conference on Multi-Agent Systems. 1996. 1

[11] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015). 1, 2

[12] Choi, Young-Cheol, and Hyo-Sung Ahn. "A survey on multi-agent reinforcement learning: Coordination problems." Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications. IEEE, 2010. 2

[13] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." science 313.5786 (2006): 504-507. 2

[14] Van Der Maaten, Laurens, Eric Postma, and Jaap Van den Herik. "Dimensionality reduction: a comparative." J Mach Learn Res 10 (2009): 66-71. 2

[15] Le, Quoc V. "A tutorial on deep learning part 2: Autoencoders, convolutional neural networks and recurrent neural networks." Google Brain (2015): 1-20. 2

[16] Plappert, Matthias, et al. "Parameter space noise for exploration." arXiv preprint arXiv:1706.01905 (2017). 2

[17] Franois-Lavet, Vincent, et al. "An Introduction to Deep Reinforcement Learning." Foundations and Trends in Machine Learning 11.3-4 (2018): 219-354. 1