Dueling DQN with Learnable Value Estimator

Fatemeh Tavakoli and Mohammad Mehdi Ebadzadeh

Department of Computer Engineering, Amirkabir University of Technology, No. 350,

Hafez Ave, Valiasr Square, Tehran, Iran

{Fateme.tavakoli, Ebadzadeh}@aut.ac.ir

Abstract

Deep reinforcement learning has made significant strides in solving complex tasks, with Dueling Deep Q-Networks (DQN) being a prominent architecture. In this study, we propose an enhancement to the Dueling DQN framework by introducing learnable value functions. By allowing the value function to be learned, we aim to improve the accuracy and adaptability of the model. Our approach involves decomposing the value function into the sum of the action-value function and a learnable advantage function. This enables us to separately estimate the value and advantage components, facilitating better generalization and transfer learning. Through experimental evaluations, we demonstrate the efficacy of our proposed method, showcasing improved performance in diverse environments. The incorporation of learnable value functions represents a promising avenue for advancing deep reinforcement learning algorithms, empowering agents to achieve superior results in complex tasks.

1 Introduction

In recent years, Deep Learning has made significant contributions to the fields of Artificial Intelligence and Machine Learning(Lecun et al., 2015a). Reinforcement Learning has witnessed remarkable advancements with algorithms like Deep Q-Learning(Mnih et al., 2015a), Deep Visuomotor Policies (Levine et al., 2015),massively parallel frameworks (Nair et al., 2015),and AlphaGo (Silver et al., 2016)surpassing human performance in various games. These algorithms typically rely on standard neural networks such as MLPs and Convolutional Networks to achieve their impressive results. However, researchers have been actively working to enhance these networks and address the challenges posed by RL methods.

One innovative approach that has emerged is the Dueling Network (Wang et al., 2015) proposed a novel architecture modification that surpasses the state-of-the-art methods. In this paper, we present a new method based on Dueling Networks, aiming to further enhance the aforementioned DRL architecture. The paper is structured as follows: Section 1 provides a brief literature review, Section 2 outlines the preliminary concepts for this research, Section 3 presents the theoretical aspects of our method, Section 4 showcases the results, and finally, Section 5 concludes the paper.

1.1 Related Work

The primary goal of researchers in the field of Artificial Intelligence is to design fully autonomous agents that can interact with the environment and learn optimal behaviors, improving ever so slightly just like humans. Designing such an agent is a daunting task, though researchers in the past have had success creating such agents (Silver et al., 2016), these agents are limited to fairly low-dimensional domains. What plagues the traditional RL approaches is memory, time and sample complexity (Strehl et al., n.d.). In recent years however, rise of Deep Learning made it possible to employ powerful function approximators to overcome these drawbacks. Deep Learning dramatically changed many areas in Machine Learning such as object detection, speech recognition and computer vision (Lecun et al., 2015b). The most important aspect of Deep Learning that allows us to perform tasks that were previously impossible, is the ability to automatically represent high-dimensional data (e.g., images, text and audio) in low-dimensional domain.

This property is achieved using deep neural networks and architecture that almost eliminates the curse of dimensionality (Bengio et al., 2012). RL is no exception, with the Deep Learning as a powerful tool, a new term has been coined by researchers which is "Deep Reinforcement Learning". Deep Learning enables RL to solve real-world problems that otherwise would be intractable. Amongst the recent innovations in DRL, the first that revolutionized and changed the course of RL related researches was development of an algorithm that can learn to play wide range of Atari 2600 games on superhuman level, just by extracting features from images (Mnih et al., 2015a). This research was the first to demonstrate the potency of RL algorithms combined with Deep Learning, which can train agents only with raw and relatively high-dimensional images. AlphaGo is another agent which managed to beat human world champion in Go, it is a hybrid system that combines advanced search trees and deep neural network (Silver et al., 2016).Neural networks in AlphaGo were trained using supervised and reinforcement learning, in tandem with a traditional heuristic search algorithm. DRL algorithms have already achieved great success in wide range of problems such as robotics, where control policy for robots is learned directly from input images of the real world, which outperform controllers that were previously designed using Control Theory. These algorithms are so capable that researchers have been able to design agents with meta-learning ability (learn to learn)[(Fang et al., 2022). Domain of video games is indeed intriguing, but it's not the end goal for DRL algorithms. Main focus behind DRL is to design agents that can learn and adapt in real world situations (Mnih et al., 2015b), having said that, RL is also an interesting way of approaching optimization problems using trial and error (Li & Malik, 2016).DRL methods have also been applied to traditional RL methods, scaling up prior work to high-dimensional problems, as we previously mentioned, DRL can address curse of dimensionality spectacularly (Bengio et al., 2012). the well-known function approximation ability of deep architectures made them a great choice for regressing various RL functions (that we discuss in the next section), one of the earliest implementation is TD-Gammon, neural network that reached champion-level performance in Backgammon decades ago (Gerald Tesauro & Keith, 1995). Current methods, address the highly complex domain of inputs such as images and videos (Fang et al., 2022; Schulman et al., 2015). One of the key components of RL is Q-function, van Hasselt showed that the single estimator in Q-learning is suffering from over-estimation, hence a new algorithm with double estimator was proposed that improved the learning process immensely (Van Hasselt, n.d.). Though Double Q learning requires one additional function to be learned, but researchers proposed a way to use available target network from DQN to reduce the complexity and achieve better results (Wu et al., 2021).

2 Preliminaries

In this section, we will introduce the field of RL and mathematical formulations. Essentially, RL agent learns through interaction, it interacts with the environment, observes the reward for its actions and can learn to optimize its behavior by the reward it received. This method of learning has its roots in psychology, which is the foundation of RL (Wu et al., 2021). Formally, an agent, following an algorithm, observes a state t from the environment, at timestep t. Then, agent interacts with the environment by performing an action a_t, the environment transitions to the next state s_{t+1} which is dependent on current state s_t and action a_t Agent receives a reward $r_{t+1}$from the environment as feedback. The goal of RL is to learn a policy that maximizes the expected return of the reward function. Given a state, policy returns an action for agent to take, an optimal policy maximizes the expected cumulative reward of the sequence. Every action taken by the agent contains useful information that agent utilizes to learn, which is illustrated in Figure 1

2.1 Markov Decision Processes

RL can be modeled as a Markov Decision Process (MDP), under these assumptions:

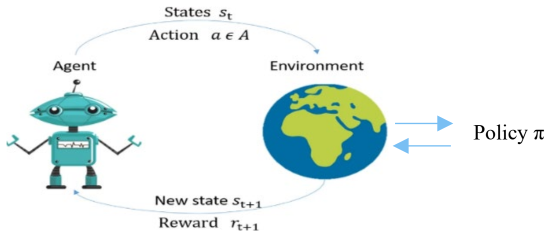

Figure 1: The perception-action-learning-loop, at the timestep t, agent receives the reward $r_t$and observes state s_t, the agent uses its policy to determine which action a_t yields the best rewards, then executes this action and environment transitions to the new state s_{t+1}. Agent uses the state transition tuple in the form of $(s_t, a_t, s_{t+1}, r_{t+1})$ to improve its policy.

- A set of states S and probability of the starting state $p(s_0)$.
- A set of actions A.
- $\mathcal{T}(s_{t+1}|s_t, a_t)$ Transition dynamics, which is the mathematical model of the system that maps current state and action to the next state.
- Reward function r_t.
- discount factor $\gamma \in [0,1]$ in order to put emphasis on immediate rewards and make sum of the rewards tractable.

Generally, the policy maps states to a new distribution over actions: $\pi: s \to p(A = a|s)$. Every sequence of states, actions and rewards is an trajectory of the given policy, which returns a reward function $R = \sum_{t=0}^{T-1} \gamma^t r_{t+1}$ (T is the last time step). RL aims to find an optimal policy that maximizes the reward function:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}[R|\pi] \qquad (1)$$

A key assumption in MDPs is that the current state is only dependent on the previous state and action, under the D-separation property, rather than all previous states. Having said that, this assumption is unrealistic as it requires the states to be fully observable, which does not hold true for every problem.

2.2 Value and Quality Functions

The state-action quality function $Q^\pi(s,a)$ is defined as expected value of the reward function given the state action tuple under a certain policy:

$$Q^\pi(s,a) = \mathbb{E}[R|s,a,\pi] \qquad (2)$$

The state value function V (s) is the expected reward function given state under a certain policy:

$$V^{\pi}(s,a) = \mathbb{E}[R|s,a,\pi] \qquad (3)$$

it can be rewritten as:

$$V^{\pi}(s,a) = \mathbb{E}_{a_t \sim \pi(a_t, s_t)}[Q^\pi(a_t, s_t)] \qquad (4)$$

One may say Value function determines how good or bad a certain state is, by summing over all possible actions, hence, optimal policy's value function is:

$$V^*(s) = \max V^\pi(s) \qquad \forall s \epsilon S \qquad (5)$$

In order to learn the Q function, we employ Markov property and rewrite the function using a Bellman equation [25], which estimates Q function recursively:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi(\pi_{t+1})] \qquad (6)$$

Meaning Q^π can be improved upon by a method called bootstrapping, i.e., one can employ current estimated values of $Q^\pi$to further improve the Q function. This is the bedrock of Q-learning [26] and SARSA algorithm [27].

We define another important quantity, the advantage function, relating the value and Q functions:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \qquad (7)$$

which means $\mathbb{E}_{a \sim \pi(s)}[A^\pi(s,a)] = 0$.

2.3 Deep Q-networks

To approximate the value functions discussed in the previous section, a deep Q-network (DQN) with parameters θ, denoted as $Q(s,a;\theta)$, can be utilized. The network is trained by optimizing a sequence of loss functions at each iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[\left(y_i^{DQN} - Q(s,a;\theta_i)\right)^2\right] \tag{8}$$

With

$$y_i^{DQN} = r + \gamma \max_{a} Q(s',a';\theta^-) \tag{9}$$

where θ^- represents the parameters of a fixed and separate target network. We could attempt to use standard Q-learning to learn the parameters of the network $Q(s,a;\theta)$ online. However, this estimator performs poorly in practice. A key innovation in (Ba et al., 2014) was to freeze the parameters of the target network $Q(s',a';\theta^-)$ for a fixed number of iterations while updating the online network $Q(s,a;\theta_i)$ by gradient descent. (This greatly improves the stability of the algorithm.) The specific gradient update is:

$$\nabla_{\theta_i} l_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[y_i^{DQN} - Q(s,a;\theta_i)\nabla_{\theta_i} Q(s,a;\theta_i)\right] \tag{10}$$

This approach is model-free as it relies on environment-generated states and rewards. It is off-policy since the states and rewards are obtained using a behavior policy (such as epsilon greedy in DQN) that differs from the policy being learned.

Experience replay is another crucial component of DQN. During the learning process, the agent collects a dataset of experiences, denoted as $D_t = \{e_1, e_2, \ldots, e_t\}$ where each experience e_t consists of the current state s_t, action a_t, reward r_t, and next state s_{t+1} obtained from various episodes. Instead of solely using the current experience, the Q-network is trained by randomly sampling mini-batches of experiences from D, selected uniformly at random.

2.4 Dueling Network Architecture

The dueling network architecture is motivated by the observation that in many states, the value of each action is not equally important.

To address this, a Q-network architecture was developed (shown in Figure 2).

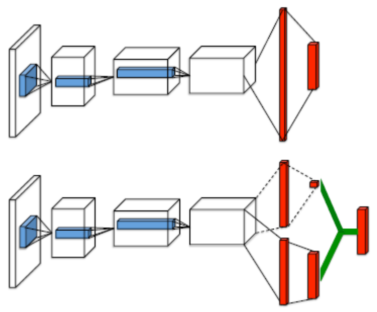
Figure 2: Traditional one stream Q-network (top) and dueling network (bottom).

It consists of convolutional layers, similar to the original DQNs, followed by two sequences of fully connected layers. These sequences are designed to approximate the value and advantage functions separately. Eventually, the two streams are combined to estimate the Q function, producing a set of values. This allows the network to be trained using well-known algorithms such as SARSA.

Blending the two streams together requires careful consideration, and the dueling network architecture was designed with this in mind.

In the architecture, one sequence of fully connected layers outputs a one-dimensional function $V(s;\ \theta,\beta)$, representing the value of the state. The other sequence outputs a $|A|$-dimensional array $A(s,a;\ \theta,\beta)$, where θ represents the parameters of the shared architecture (convolutional layers), while β represents the parameters of the exclusive network for each action (fully connected layers).

Considering the definition of advantage $\mathbb{E}_{a\sim\pi(s)}[A^\pi(s,a)] = 0$, and the fact that a deterministic policy chooses the action a^* with the highest Q-value, we have $Q(s,a^*) = V(s)$, implying that $A(s,a^*) = 0$.

utilize the definition of advantage, which is the difference between the Q-value of an action and the value of the state, and draw a conclusion :

$$Q(s,a;\ \theta,\alpha,\beta) = V(s;\ \theta,\beta) + A(s,a;\theta,\alpha) \tag{11}$$

However, applying this equation to all state-action pairs would require re-estimating $V(s;\ \theta,\beta)$, $|A|$-times. Furthermore, Equation 11 lacks identifiability, meaning that given Q, we cannot uniquely approximate V and A. Using this equation directly often leads to poor performance.

To overcome this challenge, one approach is to constrain the advantage function approximator to have zero advantage for the selected action. In other words, the final equation in the architecture is designed to handle the forward mapping and ensure that the advantage at the chosen action is zero.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a'\epsilon|A|} A(s,a';\theta,\alpha)) \tag{12}$$

Now, for $a^* = \underset{a'\epsilon A}{\operatorname{argmax}}\, Q(s,a';\theta,\alpha,\beta) = \underset{a'\epsilon A}{\operatorname{argmax}}\, A(s,a';\theta,\alpha)$, we obtain $Q(s,\ a^*;\ \theta,\alpha,\beta) = V\ (s;\ \theta,\beta)$. Thus, the sequence $V\ (s;\ \theta,\beta)$ is an estimation for the value function, while the other is for advantage function. Alternatively, in order to have more stable training, max operator can be replaced with an average:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)) \tag{13}$$

With the introduction of the constant bias in the estimation of V and A, the original semantics of V and A may be altered. However, this modification simplifies the optimization process, as the advantages now change at the same rate as the mean, rather than compensating for changes in the advantage of the optimal action, as described in Equation 13.

3 Dueling architecture with learnable Value Estimators

In this study, we propose an enhancement to the Dueling Deep Q-Network (DDQN) algorithm by introducing learnable Value Estimators. Our approach focuses on improving the estimation of the value function, V(s), by decomposing it into the action-value function, Q(s,a), and the advantage function, A(s,a):

$$V(s) = Q(s,a) - A(s,a) \tag{14}$$

To estimate the action-value function, Q(s,a), we adopt the Monte Carlo approach :

$$Q(s,a) = \sum_{t'=t}^{T} \gamma^{(t\prime-t)} \times r(s_{t\prime}, a_{t\prime}) \tag{15}$$

For the estimation of the value function, V(s), we define the target value, y_i, as:

$$y_i = V_\beta(s_t) = \sum_{t'=t}^{T} \gamma^{(t\prime-t)} \times r(s_{t'}, a_{t'}) - A_\alpha(s_t, a_t) \tag{16}$$

To train the neural network we formulate the objective as minimizing the mean squared loss between the estimated value, $V(s_i)$, and the target value, y_i. The loss function is defined as :

$$loss(\alpha_i) = \sum (V(s_i) - y_i)^2 \tag{17}$$

By incorporating learnable Value Estimators into the DDQN algorithm and employing this proposed method for value function estimation, we aim to enhance the algorithm's performance in complex reinforcement learning tasks. This method holds the potential to improve the learning capabilities of the DDQN algorithm and enable more efficient training in challenging environments.

4 experiments

We now show the practical performance of the Dueling network with learnable Value Estimator.

4.1 Lunar Lander

In this experiment, we aim to evaluate the performance of the proposed method on the LunarLander-v2 environment from the Box2D gymnasium, as illustrated in Figure 3. The objective is to train an agent to successfully land a lunar lander spacecraft using reinforcement learning techniques.

The observation space of the LunarLander-v2 environment is an 8-dimensional vector representing various aspects of the lander's state. These dimensions include the lander's coordinates (x and y), linear velocities (x and y), angle, angular velocity, and indicators for leg contact with the ground (left and right).

The action space is discrete and consists of four possible actions{do nothing, fire the left orientation engine, fire the main engine, and fire the right orientation engine}. The agent chooses actions based on the observed state to control the lander's movement. At each step of the environment, the agent receives rewards based on its actions and the resulting state. The rewards are designed to guide the agent towards a successful landing. They are determined by the following factors: {1. Proximity to the landing pad: The reward increases as the lander gets closer to the landing pad and decreases as it moves further away. 2. Velocity control: The reward increases for slower lander velocities and decreases for faster velocities. 3. Angle control: The reward decreases as the lander tilts more from the horizontal position. 4. Leg contact: The agent receives a reward of 10 points for each leg in contact with the ground, encouraging a safe landing. 5. Engine usage penalties: Firing the side engines incurs a penalty of 0.03 points per frame, while firing the main engine incurs a penalty of 0.3 points per frame. This encourages fuel efficiency. 6. Episode termination reward: The agent receives a reward of -100 for crashing and +100 for a safe landing}.

An episode in the LunarLander-v2 environment ends under the following conditions:{1. Lander crash: If the lander body comes into contact with the lunar surface, the episode terminates. 2. Out of viewport: If the x-coordinate of the lander exceeds 1, indicating that it has moved outside the viewport, the episode terminates. 3. Lander not awake: If the lander is not moving or colliding with any other body in the environment, it enters a sleep state and the episode terminates}.

By conducting this experiment, we aim to demonstrate the effectiveness of the proposed method in training an agent to autonomously land the lunar lander spacecraft. The evaluation will provide insights into the agent's learning capabilities, its ability to navigate and control the lander effectively, and its overall performance in the LunarLander-v2 environment.

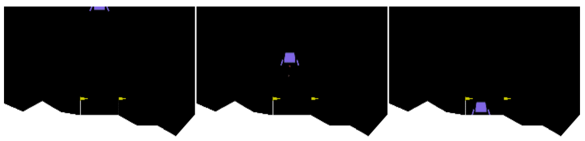

Fig 3: LunarLander_v2 environment.

4.2 Results

In this section, we analyze and discuss the findings presented in Table (1). It is important to note that our proposed architecture was implemented on a system consisting of a single Nvidia GTX 3050 Ti graphics card and an Intel Core i9-12900HK CPU. Due to these hardware differences compared to the system used by (Guo et al., 2014),there may be variations in the achieved scores.

As shown in Table (1), our implementation of the DDQN with learnable Value Estimators on the lunarlander_v2 environment resulted in an increased average score and a reduced number of episodes required to reach completion. The termination condition was met when the model achieved an average score of 200 in the last 100 episodes of training.

LunarLander-v2	***# episode***	***Avg score***
DDQN	450	240.11
DDQN+ learnable Value Estimators	**366**	**245.41**

5 conclusion

In conclusion, we have improved the Dueling Deep Q-Network (DDQN) algorithm by utilizing learnable Value Estimators instead of a basic fully connected layer. This enhancement, implemented on our specific hardware configuration, resulted in higher average scores and faster convergence in the lunar land_v2 environment. Our findings suggest that incorporating learnable Value Estimators enhances the performance of DDQN in complex tasks. Further research can explore this approach in other reinforcement learning scenarios and examine its sensitivity to hardware variations.

References

Ba, J., Mnih, V., & Kavukcuoglu, K. (2014). *Multiple Object Recognition with Visual Attention*. http://arxiv.org/abs/1412.7755

Bengio, Y., Courville, A., & Vincent, P. (2012). *Representation Learning: A Review and New Perspectives*. http://arxiv.org/abs/1206.5538

Fang, Q., Xu, X., Wang, X., & Zeng, Y. (2022). Target-driven visual navigation in indoor scenes using reinforcement learning and imitation learning. *CAAI Transactions on Intelligence Technology*, *7*(2), 167–176. https://doi.org/10.1049/cit2.12043

Gerald Tesauro, B., & Keith, T. (1995). Temporal Difference Learning and TD-Gammon. In *Communications of the ACM* (Vol. 38, Issue 3).

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems* (Vol. 27). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2014/file/8bb88f80d334b1869781beb89f7b73be-Paper.pdf

Lecun, Y., Bengio, Y., & Hinton, G. (2015a). Deep learning. In *Nature* (Vol. 521, Issue 7553, pp. 436–444). Nature Publishing Group. https://doi.org/10.1038/nature14539

Lecun, Y., Bengio, Y., & Hinton, G. (2015b). Deep learning. In *Nature* (Vol. 521, Issue 7553, pp. 436–444). Nature Publishing Group. https://doi.org/10.1038/nature14539

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2015). *End-to-End Training of Deep Visuomotor Policies*. http://arxiv.org/abs/1504.00702

Li, K., & Malik, J. (2016). *Learning to Optimize*. http://arxiv.org/abs/1606.01885

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015a). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015b). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., Panneershelvam, V., Suleyman, M., Beattie, C., Petersen, S., Legg, S., Mnih, V., Kavukcuoglu, K., & Silver, D. (2015). *Massively Parallel Methods for Deep Reinforcement Learning*. http://arxiv.org/abs/1507.04296

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). *Trust Region Policy Optimization*. http://arxiv.org/abs/1502.05477

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature 2016 529:7587*, *529*(7587), 484–489. https://doi.org/10.1038/nature16961

Strehl, A. L., Li, L., Wiewiora, E., Langford, J., & Littman, M. L. (n.d.). *PAC Model-Free Reinforcement Learning*.

Van Hasselt, H. (n.d.). *Double Q-learning*.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2015). *Dueling Network Architectures for Deep Reinforcement Learning*. http://arxiv.org/abs/1511.06581

Wu, J., Sun, X., Zeng, A., Song, S., Rusinkiewicz, S., & Funkhouser, T. (2021). *Spatial Intention Maps for Multi-Agent Mobile Manipulation*. https://doi.org/10.1109/ICRA48506.2021.9561359