Safe Deep Reinforcement Learning for Multi-Agent Systems with Continuous Action Spaces

Dimitrios Gkouletsos
dgkouletsos@student.ethz.ch
19-944-909

Athina Nisioti
anisioti@student.ethz.ch
19-945-294

Ziyad Sheebaelhamd
zsheebael@student.ethz.ch
19-950-245

Konstantinos Zisis
kzisis@student.ethz.ch
19-942-465

Abstract

In this project,we present a novel, safe, multi-agent Deep Reinforcement Learning algorithm based on Deep Deterministic Policy Gradient. Preserving safety is a major challenge in multi-agent systems which we tackle by adding a safety layer that solves an optimization problem projecting unsafe actions to a safe domain. In order to take into account infeasible problems that arise, a constraint relaxation that softens the original optimization problem is also introduced. In experiments presented, it was found that the soft-problem introduced achieves a dramatic decrease in constraint violations, making safety available even during the learning procedure.[1]

I. Introduction

In recent years, Deep Reinforcement Learning (Deep RL) has received great attention in the context of a variety of applications such as health-care domain for treatment policies [1], financial market decision processes [2] and multi-robot autonomous exploration [3], only to name a few. The well-known Deep Mind work in Atari games [4] established successfully Deep RL as a control policy learning architecture combined with convolutional neural networks. Motivated by this breakthrough, the authors in [5] propose a framework for self-driving vehicles utilizing Deep RL.

Recent advances in computational resources and algorithmic sophistication have enabled real-time Deep RL applications [5], [6]. However, real-world cases consist of embedded constraints that highlight the importance of safe operation. In [7] the authors suggest a safety filter during the training phase in order to project actions into a safe domain by solving a quadratic program. A novel Deep RL architecture, called OptLayer, is described in [8] which receives as inputs possibly unsafe actions derived by a neural network and produces actions that satisfy constraints. An alternative approach in [9] employs a human-based teaching method in which an agent is assisted during learning by an automatic instructor that prevents agent against constraints violation.

In this work, we propose a multi-agent extension of the idea proposed in [7] that solves an optimization problem in order to preserve safety for all agents during the whole training procedure. It is notice-worthy to mention that our algorithm operates in continuous action spaces unlike the discrete versions portrayed in [10]. In contrast to [11] and [12] that uses an already existing backup policy found by previous domain- specific knowledge, we employ a safety filter that changes the learnt policy in the least intervening way to guarantee safety. This is done by estimating the sensitivity of the constraints to the applied actions via a neural network before deploying the agent training. The data collected during the pertaining phase can generated by following predefined behavioural policy by a human expert or by following a stochastic policy. In our work, we dropped the conservative assumptions proposed in [7] where it was assumed that the optimization problem is always feasible and that only one constraint is active. In real world problems, the optimization formulation proposed has no guarantees to be recursively feasible and in multi-agent coordination problems where agents impose constraints on one another, having one active constraint is a naive assumption. We have therefore attempted to drop those assumptions and generalize the formulation to capture more complex scenarios to reveal the real potential of such work. We therefore propose a specific soft constrained formulation of the problem

[1] The relevant code for implementing our results can be found in our github repository

that enhances safety significantly and is general enough to capture complicated scenarios in centralized multi-agent problems. However, it is to be noted that our approach does not guarantee zero constraint violation, but by tightening the constraints by a tolerance one could achieve almost safe behavior during training.

II. Models and Methods

A. Problem formulation

We consider a discrete-time, finite dimensional multi-agent system with N agents, continuous state space $\chi = (\chi_1, \ldots, \chi_N) \subseteq \mathcal{X} \subseteq \mathbb{R}^n$ and continuous action space $\alpha = (\alpha_1, \ldots, \alpha_N) \subseteq \mathcal{A} \subseteq \mathbb{R}^m$. It is implicitly assumed that each $\chi \subseteq \mathcal{X}$ and $\alpha \subseteq \mathcal{A}$ are vectors encoding the concatenation of individual agent states and actions respectively. We also define a reward function $\mathcal{R}$, a discount factor $\gamma \in (0, 1)$ and the immediate constraint (value for each agent for the i-th constraint,) as $c_i(\chi) \quad \forall i \in 1, ...K$. Finally, we define a policy π to be a function mapping states to actions. In this context, we examine the problem of safe exploration in a Constrained Markov Decsion Process (CMDP) and therefore we aim to solve the optimization problem described in (1)

$$\begin{aligned} &\max_{\theta} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\left(\chi_t, \pi_\theta\left(\chi_t\right)\right)\right] \\ &\text{s.t.} \quad c_i\left(\chi_t\right) \leq C_i \quad \forall i = 1, \ldots, K \end{aligned} \tag{1}$$

It is worth stating that our goal is not only to guarantee safety in the solution of the RL algorithm, but also to be completely safe during the training procedure. This is justified by many applications requiring safety in their whole operating period since the complexity of the underlying MDP makes accurate off-line simulation intractable.

B. Safety Signal Model

In order to learn the constraints' sensitivity to the applied actions, a linearization of the form

$$c_i\left(\chi'\right) = \hat{c}_i(\chi, \alpha) \approx c_i(\chi) + g\left(\chi; w_i\right)^\top a \tag{2}$$

was used where χ' is the state that followed χ after applying action α and function g represents a NN with input χ, output of the same dimension as the action α and weights w_i. In order to proceed with $g\left(\chi; w_i\right)$ network training, it is required to generate a data-set $\mathcal{D} = \left\{\left(\chi_j, a_j, \chi'_j\right)\right\}$. In $\mathcal{D}$, each episode is initialized with a random state and actions are chosen according to a sufficiently exploratory policy. With the generated data, the sensitivity network can be trained by specifying the loss function for each constraint as

$$\arg\min_{w_i} \sum_{(\chi, a, \chi') \in D} \left(c_i\left(\chi'\right) - \left(c_i(\chi) + g\left(\chi; w_i\right)^\top a\right)\right)^2 . \tag{3}$$

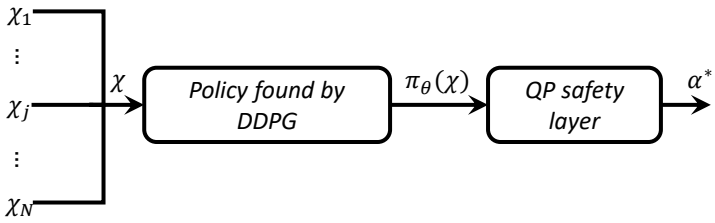

Figure 1: **An illustration of the safety layer used in combination with the DDPG network in order to apply the safe projection of the optimal action**. The individual state spaces of each agent are concatenated to a state space vector that is fed to the DDPG network to produce the optimal unconstrained policy. Then on each forward propagation an optimization problem is solved in order to produce the safe optimal action α^*

where each constraint's sensitivity will be trained separately.

C. Safety Layer Optimization

The policy network is extended by introducing an additional safety layer where it is attempted to solve an optimization problem of the form

$$\begin{aligned} &\arg\min_{a} \|a - \pi_\theta(\chi)\|_2^2 \\ &\text{s.t. } \hat{c}_i(\chi, \alpha) \leq C_i \quad \forall i = 1, \ldots, K \ . \end{aligned} \tag{4}$$

which describes a projection of the action proposed by the policy network $\pi_\theta(\chi)$ onto the safety set. Figure 1 illustrates the whole pipeline for computing an action from a given state.

In order to reveal the underlying structure in (4), we expand $\hat{c}_i(\chi, \alpha)$ in a linear fashion as denoted in (2) which results in the following approximate quadratic program

$$\begin{aligned} a^* = &\arg\min_{a} \|a - \pi_\theta(\chi)\|_2^2 \\ &\text{s.t. } g\left(\chi; w_i\right)^\top a \leq C_i - c_i(\chi) \quad \forall i = 1, \ldots, K \ . \end{aligned} \tag{5}$$

Due to the strong convexity of the resulting optimization problem, there exists a global unique minimizer to the problem given that it is feasible. In contrast to [7], where recursive feasibility was assumed and therefore a closed form solution using the Lagrangian multipliers was derived, we used a numerical QP-solver to defer from making strong assumptions on the existence of the solution which is not guaranteed for dynamical systems.

Due to the generality of the formulation, it is possible that there exists no recoverable action that can guarantee the agents to be taken to a safe state although the previous iteration of the optimization was indeed feasible. To

take this into account without running into infeasibility problems where the agents would require a backup policy to take an action, a soft constrained formulation is proposed where its solution is equivalent to the original formulation when (5) is feasible, otherwise, the optimizer is allowed to loosen the constraints by a penalized margin as proposed in [13] Our proposed formulation is given as follows

$$\begin{aligned}(a^*, \epsilon^*) = \arg\min_{a,\epsilon} \|a - \pi_\theta(\chi)\|_2^2 + \rho \|\epsilon\|_1 \\ \text{s.t. } g(\chi; w_i)^\top a \leq C_i - c_i(\chi) + \epsilon_i \\ \epsilon_i \geq 0 \,. \quad \forall i = 1, \ldots, K\end{aligned} \tag{6}$$

where $\epsilon = (\epsilon_1, \ldots, \epsilon_K)$ are the slack variables and ρ is the constraint violation penalty weight. We pick $\rho > \|\lambda^*\|_\infty$ where λ^* is the optimal Lagrange multiplier for the original problem in (5), in order to guarantee that the soft-constrained problem derives equivalent feasible optimal solutions compared to the original problem (5) when the original problem is feasible as stated in [13]. In practice, Lagrange multiplier search is time-consuming or even intractable and, hence, we assign to ρ large values by inspection. It is important to mention that (5) still holds a quadratic program structure by extending the optimization vector into (a, ϵ) and using an epigraph formulation. Note that this formulation may cause constraint violations, however, by highly penalizing the slack variables, the violation remains very minimal.

D. Deep Deterministic Policy Gradient

Training a Deep RL agent in continuous action spaces requires the use of policy gradient algorithms that do not require explicit maximization over actions since it would be intractable. Therefore an explicit parameterization of a policy is needed such that gradients can be computed for training the policy. The Deep Deterministic Policy Gradient (DDPG) algorithm firstly introduced in [14] is a model free, off-policy algorithm extending Deep Q-Networks to continuous actions spaces, where it uses actor and critic networks to learn the optimal policy while following an exploratory policy. A thorough description of the DDPG algorithm can be found in 1 .To this context, the trained actor networks trained within the DDPG agent result in the desired optimal policy such that $\pi_\theta(\chi)$=$\mu(\chi; w_\mu)$.

E. Implementation Details

In order to assess how the proposed algorithm works we conducted experiments using the multi-agent particle environment also used in [15] and [16]. In this environment a fixed number of agents are moving collaboratively in a bounded 2-D grid trying to reach a specific target

Algorithm 1 DDPG Algorithm for multiple agents

Initialize randomly weights w_Q and w_μ of critic network $Q(\chi, \alpha; w_Q)$ and actor network $\mu(\chi; w_\mu)$
Initialize weights $\hat{w}_Q \leftarrow w_Q$ and $\hat{w}_\mu \leftarrow w_\mu$ of target networks $\hat{Q}$ and $\hat{\mu}$
Initialize a replay buffer R
for episode $= 1$ **to** m **do**
  Use a random process $\mathcal{N}$ to explore actions
  Obtain initial state χ_1
  for t $= 1$ **to** T **do**
    Pick action $a_t = \mu(\chi_t; w_\mu) + \mathcal{N}_t$ based on actor network value with exploration noise
    Apply action a_t and gain reward r_t and new state χ'_t
    Store transition $\{(\chi_t, a_t, \chi'_t)\}$ in R
    Sample a random mini-batch of λ transitions $\{(\chi_i, a_i, \chi'_i)\}$ from R
    Set $z_i = r_i + \gamma \hat{Q}(\chi'_i, \hat{\mu}(\chi'_i; \hat{w}_\mu); \hat{w}_Q)$
    Update critic network by minimizing $L = \frac{1}{\lambda} \sum_i (z_i - Q(\chi_i, a_i; w_Q))^2$
    Update actor policy based on the sampled policy gradien:

$$\nabla_{w_\mu} J \approx \frac{\sum_i \nabla_a Q(\chi, a; w_Q)|_{a=\mu(\chi_i)} \nabla_{w_\mu} \mu(\chi; w_\mu)|_{\chi_i}}{\lambda}$$

    Update target networks as:

$$\hat{w}_Q \leftarrow \tau w_Q + (1-\tau)\hat{w}_Q \text{ and } \hat{w}_\mu \leftarrow \tau w_\mu + (1-\tau)\hat{w}_\mu$$

  end for
end for

assigned to each agent. In our experiments we used 3 agents and as a constraint the avoidance of collisions between them. The reward assigned to each agent is proportional to the negative proximity of the agent from its corresponding target, penalizing collisions and only providing positive reward when an agent reaches his target. The rewards are shared between all agents where the total reward returned is the sum over all agents' rewards. For the safety layer training, for each of the 6 existing constraints (2 possible collisions for each agent) a simple neural network $g(\chi, w_i)$ with 10 hidden units and ReLU activation function was trained on the randomly produced data set $\mathcal{D}$. (Note that due to symmetry of the constraints used in the experiment, we could train less NN's, but for generality we decided to consider them as independent constraints). The choice of such a simple model to approximate the constraints is justified by the low validation error that was observed. When training Adam optimizer with a batch size of 256 was used. According to [7] it is not the low validation error reported that plays an important role for safety but the exploratory nature

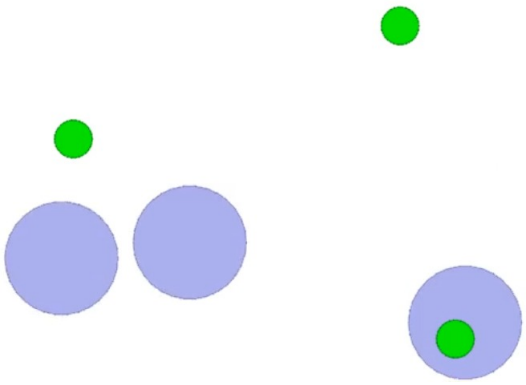

Figure 2: **A snapshot of the mutli- agent particle environment used in the simulations described**. Blue color is used to capture the agents, whereas green color refers to the target positions. In our experiments we have 3 agents trying to move to a specific target without colliding with each other.

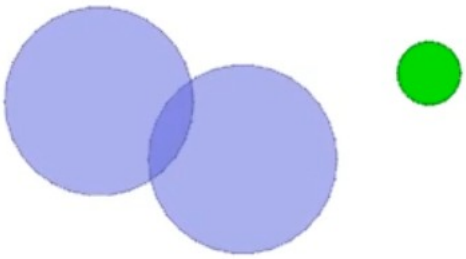

Figure 3: **An illustration of a collision happening during training**. Blue colour depicts the colliding agents, whereas green color refers to a target position

of the randomly produced data set $\mathcal{D}$.For solving the QP-Problem the python qpsolvers package, based on [17] was used. For the DDPG algorithm implementation, the actor networks were composed of 2 hidden layers with 100 units each, while the critic networks were composed of 2 hidden layers with size 500 each. The choice for all activation functions is ReLU except for the output layer of the actor networks were tanh was used to compress the actions in the range [-1,1].

III. Results

In this section, we illustrate the results of our simulations meant to validate the proposed algorithms. To assess performance, three metrics, namely reward, number of collisions and infeasible occurrences are evaluated. A collision is defined as the incident in which two agents intersect with each other as shown in Figure 3. An infeasible occurrence appears in case the hard-constrained QP in (5) fails to determine a solution that satisfies the imposed constraints.

As seen previously, we deploy three different Deep RL strategies 1) an unconstrained DDPG 2) a hard-constrained DDPG and 3) a soft-constrained DDPG. The first one prioritizes exploration and learning against safety. The hard-constrained DDPG takes into account safe operation and highlights constraint violations. The soft-constrained DDPG reveals a trade-off between safety and performance and is considered as an intermediate approach among unconstrained and hard-constrained DDPG.

Moreover,in order to gain better understanding of our results and depict the actual usage of the safety layer, test-videos of test-simulations ran, were also generated. These videos are available in the following Video repository.

In Figure 4, the reward gained during the training phase against number of episodes is illustrated. It is apparently seen that for a low number of episodes up to 1000, the rewards depict a sharp increase. Afterwards, the reward curve flattens slowly and terminates for slightly positive values. We observe that all three algorithms present a similar trend and, thus, they fulfill the main goal of reaching a target.

Figure 5 presents the cumulative number of collisions over the episode range. The soft-constrained DDPG exhibits collisions in 2.4875% of the total training episodes and as by definition expands the feasible region so as to allow constraint violation. On the other hand, the unconstrained DDPG leads to the highest number of collisions as it does not consist of a safety layer. As far as the hard-constrained DDPG is concerned, the QP safety layer prevents a significant number of collisions compared to the unconstrained DDPG. Note that the cumulative collisions present a sub-linear trend over a growing number of episodes in both unconstrained and hard-constrained DDPG cases.

To evaluate the impact of the hard-constrained DDPG, it is essential to investigate the infeasible occurrences. Over half of the episodes and in particular 59.22% of them are directly related to infeasible conditions. This highlights the necessity for a constrained safety layer that preserves safety. The soft-constrained configuration relaxes the constraints such that the feasibility is always guaranteed and, hence, does not present any infeasible occurrences.

IV. Discussion

In this work, we proposed a mechanism that is able to guarantee safety in a multi agent environment with action spaces in the continuous domain. Our method solves a soft-constrained optimization problem in order to ,not only project the action in the safe domain when the ini-

tial optimization problem is feasible, but also tackle the cases when the optimization problem that needs to be solved is infeasible without posing any extra assumptions that would make the studied problem less general. It goes without saying that ,there are many applications that could benefit of such an approach, especially with the rise of interest in the usage of reinforcement learning in many industry sectors.Future work might attempt the study of the behaviour of more complex optimization problems instead of the QP-program solved in this approach. Finally, a big challenge and possible inspiration for future work would be to extend these methods to decentralized multi agent environments.

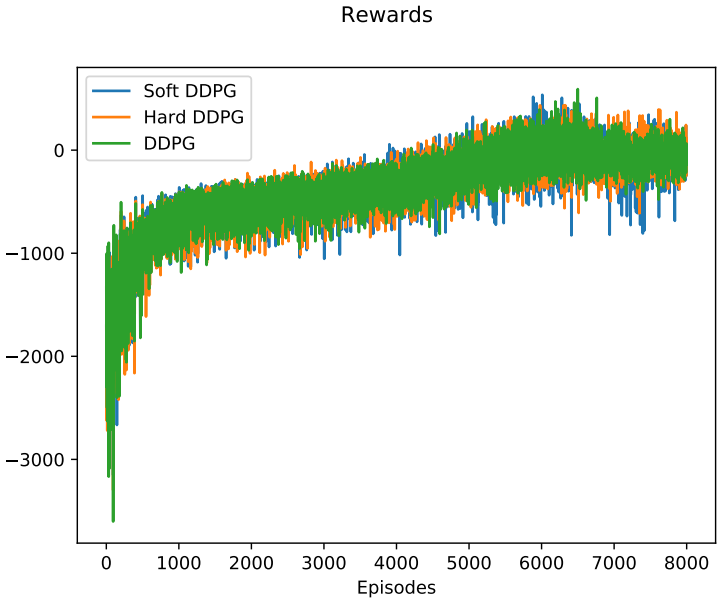

Figure 4: **Curves illustrating the reward gained during the training procedure over episodes.** Observe that the rewards gained in all 3 algorithms are almost the same, since no compromise was made with respect to reward when solving the optimization problem.

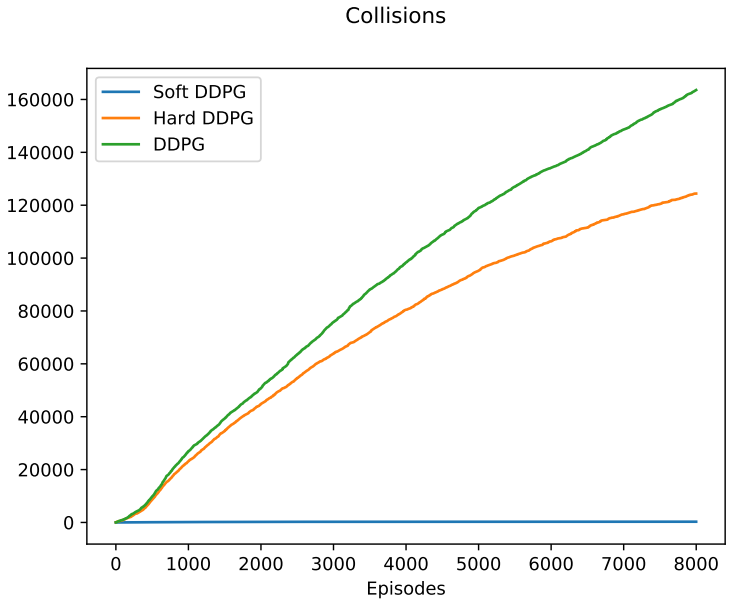

Figure 5: **Curve showing the cumulative number of collisions during training in the 3 different training procedures followed**. Observe that when solving the hard optimization problem there is a decrease on the number of collisions that could not guarantee the desired property of safety, while when solving the soft-constraint optimization problem a dramatic decrease can be observed.

References

[1] Lu Wang, Wei Zhang, Xiaofeng He, and Hongyuan Zha. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2447–2456, 2018.

[2] Zhengyao Jiang and Jinjun Liang. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*, pages 905–913. IEEE, 2017.

[3] Junyan Hu, Hanlin Niu, Joaquin Carrasco, Barry Lennox, and Farshad Arvin. Voronoi-based multi-robot autonomous exploration in unknown environments via deep reinforcement learning. 2015.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[6] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. In *Advances in Neural Information Processing Systems*, pages 3814–3823, 2018.

[7] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

[8] Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243. IEEE, 2018.

[9] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[10] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[11] Wenbo Zhang, Osbert Bastani, and Vijay Kumar. Mamps: Safe multi-agent reinforcement learning via model predictive shielding. *arXiv preprint arXiv:1910.12639*, 2019.

[12] Arbaaz Khan, Chi Zhang, Shuo Li, Jiayue Wu, Brent Schlotfeldt, Sarah Y Tang, Alejandro Ribeiro, Osbert Bastani, and Vijay Kumar. Learning safe unlabeled multi-robot planning with motion constraints. *arXiv preprint arXiv:1907.05300*, 2019.

[13] Eric C Kerrigan and Jan M Maciejowski. Soft constraints and exact penalty functions in model predictive control. 2000.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[15] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[16] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[17] Donald Goldfarb and Ashok Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical programming*, 27(1):1–33, 1983.