QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement learning

Kyunghwan Son [1] **Daewoo Kim** [1] **Wan Ju Kang** [1] **David Hostallero** [1] **Yung Yi** [1]

Abstract

We explore value-based solutions for multi-agent reinforcement learning (MARL) tasks in the centralized training with decentralized execution (CTDE) regime popularized recently. However, VDN and QMIX are representative examples that use the idea of factorization of the joint action-value function into individual ones for decentralized execution. VDN and QMIX address only a fraction of factorizable MARL tasks due to their structural constraint in factorization such as additivity and monotonicity. In this paper, we propose a new factorization method for MARL, QTRAN, which is free from such structural constraints and takes on a new approach to transforming the original joint action-value function into an easily factorizable one, with the same optimal actions. QTRAN guarantees more general factorization than VDN or QMIX, thus covering a much wider class of MARL tasks than does previous methods. Our experiments for the tasks of multi-domain Gaussian-squeeze and modified predator-prey demonstrate QTRAN's superior performance with especially larger margins in games whose payoffs penalize non-cooperative behavior more aggressively.

1. Introduction

Reinforcement learning aims to instill in agents a good policy that maximizes the cumulative reward in a given environment. Recent progress has witnessed success in various tasks, such as Atari games (Mnih et al., 2015), Go (Silver et al., 2016; 2017), and robot control (Lillicrap et al., 2015), just to name a few, with the development of deep learning techniques. Such advances largely consist of deep neural networks, which can represent action-value functions and policy functions in reinforcement learning problems as a high-capacity function approximator. However, more complex tasks such as robot swarm control and autonomous driving, often modeled as cooperative multi-agent learning problems, still remain unconquered due to their high scales and operational constraints such as distributed execution.

The use of deep learning techniques carries through to cooperative multi-agent reinforcement learning (MARL). MADDPG (Lowe et al., 2017) learns distributed policy in continuous action spaces, and COMA (Foerster et al., 2018) utilizes a counterfactual baseline to address the credit assignment problem. Among value-based methods, value function factorization (Koller & Parr, 1999; Guestrin et al., 2002a; Sunehag et al., 2018; Rashid et al., 2018) methods have been proposed to efficiently handle a joint action-value function whose complexity grows exponentially with the number of agents.

Two representative examples of value function factorization include VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018). VDN factorizes the joint action-value function into a sum of individual action-value functions. QMIX extends this additive value factorization to represent the joint action-value function as a monotonic function — rather than just as a sum — of individual action-value functions, thereby covering a richer class of multi-agent reinforcement learning problems than does VDN. However, these value factorization techniques still suffer structural constraints, namely, additive decomposability in VDN and monotonicity in QMIX, often failing to factorize a factorizable task. A task is factorizable if the optimal actions of the joint action-value function are the same as the optimal ones of the individual action-value functions, where additive decomposability and monotonicity are only sufficient — somewhat excessively restrictive — for factorizability.

Contribution In this paper, we aim at successfully factorizing *any* factorizable task, free from additivity/monotonicity concerns. We transform the original joint action-value function into a new, easily factorizable one with the same optimal actions in both functions. This is done by learning a state-value function, which corrects for the severity of the partial observability issue in the agents.

[1]School of Electrical Enginerring, KAIST, Daejeon, South Korea. Correspondence to: Yung Yi <yiyung@kaist.edu>, Kyunghwan Son <kevinson9473@kaist.ac.kr>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

We incorporate the said idea in a novel architecture, called QTRAN, consisting of the following inter-connected deep neural networks: (i) joint action-value network, (ii) individual action-value networks, and (iii) state-value network. To train this architecture, we define loss functions appropriate for each neural network. We develop two variants of QTRAN: QTRAN-base and QTRAN-alt, whose distinction is twofold: how to construct the transformed action-value functions for non-optimal actions; and the degree of stability and convergence speed. We assess the performance of QTRAN by comparing it against VDN and QMIX in three environments. First, we consider a simple, single-state matrix game that does not satisfy additivity or monotonicity, where QTRAN successfully finds the joint optimal action, whereas neither VDN nor QMIX does. We then observe a similarly desirable cooperation-inducing tendency of QTRAN in more complex environments: modified predator-prey games and multi-domain Gaussian squeeze tasks. In particular, we show that the performance gap between QTRAN and VDN/QMIX increases with environments having more pronounced non-monotonic characteristics.

Related work Extent of centralization varies across the spectrum of cooperative MARL research. While more decentralized methods benefit from scalability, they often suffer non-stationarity problems arising from a trivialized superposition of individually learned behavior. Conversely, more centralized methods alleviate the non-stationarity issue at the cost of complexity that grows exponentially with the number of agents.

Prior work tending more towards the decentralized end of the spectrum include Tan (1993), whose independent Q-learning algorithm exhibits the greatest degree of decentralization. Tampuu et al. (2017) combines this algorithm with deep learning techniques presented in DQN (Mnih et al., 2015). These studies, while relatively simpler to implement, are subject to the threats of training instability, as multiple agents attempt to improve their policy in the midst of other agents, whose policies also change over time during training. This simultaneous alteration of policies essentially makes the environment non-stationary.

The other end of the spectrum involves some centralized entity to resolve the non-stationarity problem. Guestrin et al. (2002b) and Kok & Vlassis (2006) are some of the earlier representative works. Guestrin et al. (2002b) proposes a graphical model approach in presenting an alternative characterization of a global reward function as a sum of conditionally independent agent-local terms. Kok & Vlassis (2006) exploits the sparsity of the states requiring coordination compared to the whole state space and then tabularize those states to carry out tabular Q-learning methods as in Watkins (1989).

The line of research positioned mid-spectrum aims to put together the best of both worlds. More recent studies, such as COMA (Foerster et al., 2018), take advantage of CTDE (Oliehoek et al., 2008); actors are trained by a joint critic to estimate a counterfactual baseline designed to gauge each agent's contribution to the shared task. Gupta et al. (2017) implements per-agent critics to opt for better scalability at the cost of diluted benefits of centralization. MADDPG (Lowe et al., 2017) extends DDPG (Lillicrap et al., 2015) to the multi-agent setting by similar means of having a joint critic train the actors. Wei et al. (2018) proposes Multi-Agent Soft Q-learning in continuous action spaces to tackle the relative overgeneralization problem (Wei & Luke, 2016) and achieves better coordination. Other related work includes CommNet (Sukhbaatar et al., 2016), DIAL (Foerster et al., 2016), ATOC (Jiang & Lu, 2018), and SCHEDNET (Kim et al., 2019), which exploit inter-agent communication in execution.

On a different note, two representative examples of value-based methods have recently been shown to be somewhat effective in analyzing a class of games. Namely, VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) represent the body of literature most closely related to this paper. While both are value-based methods and follow the CTDE approach, the additivity and monotonicity assumptions naturally limit the class of games that VDN or QMIX can solve.

2. Background

2.1. Model and CTDE

DEC-POMDP We take DEC-POMDP (Oliehoek et al., 2016) as the *de facto* standard for modelling cooperative multi-agent tasks, as do many previous works: as a tuple $\mathcal{G} =< \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, N, \gamma >$, where $\boldsymbol{s} \in \mathcal{S}$ denotes the true state of the environment. Each agent $i \in \mathcal{N} := \{1, ..., N\}$ chooses an action $u_i \in \mathcal{U}$ at each time step, giving rise to a joint action vector, $\boldsymbol{u} := [u_i]_{i=1}^{N} \in \mathcal{U}^N$. Function $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \times \mathcal{S} \mapsto [0, 1]$ governs all state transition dynamics. Every agent shares the same joint reward function $r(\boldsymbol{s}, \boldsymbol{u}) : \mathcal{S} \times \mathcal{U}^N \mapsto \mathbb{R}$, and $\gamma \in [0, 1)$ is the discount factor. Each agent has individual, partial observation $z \in \mathcal{Z}$, according to some observation function $O(\boldsymbol{s}, i) : \mathcal{S} \times \mathcal{N} \mapsto \mathcal{Z}$. Each agent also has an action-observation history $\tau_i \in \mathcal{T} := (\mathcal{Z} \times \mathcal{U})^*$, on which it conditions its stochastic policy $\pi_i(u_i|\tau_i) : \mathcal{T} \times \mathcal{U} \mapsto [0, 1]$.

Training and execution: CTDE Arguably the most naïve training method for MARL tasks is to learn the individual agents' action-value functions independently, *i.e.*, independent Q-learning. This method would be simple and scalable, but it cannot guarantee convergence even in the limit of infinite greedy exploration. As an alternative solution, recent works including VDN (Sunehag et al., 2018)

and QMIX (Rashid et al., 2018) employ centralized training with decentralized execution (CTDE) (Oliehoek et al., 2008) to train multiple agents. CTDE allows agents to learn and construct individual action-value functions, such that optimization at the individual level leads to optimization of the joint action-value function. This in turn, enables agents at execution time to select an optimal action simply by looking up the individual action-value functions, without having to refer to the joint one. Even with only partial observability and restricted inter-agent communication, information can be made accessible to all agents at training time.

2.2. IGM Condition and Factorizable Task

Consider a class of sequential decision-making tasks that are amenable to factorization in the centralized training phase. We first define **IGM** (Individual-Global-Max):

Definition 1 (IGM)**.** *For a joint action-value function $Q_{\text{jt}} : \mathcal{T}^N \times \mathcal{U}^N \mapsto \mathbb{R}$, where $\boldsymbol{\tau} \in \mathcal{T}^N$ is a joint action-observation histories, if there exist individual action-value functions $[Q_i : \mathcal{T} \times \mathcal{U} \mapsto \mathbb{R}]_{i=1}^N$, such that the following holds*

$$\arg\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \begin{pmatrix} \arg\max_{u_1} Q_1(\tau_1, u_1) \\ \vdots \\ \arg\max_{u_N} Q_N(\tau_n, u_N) \end{pmatrix}, \quad (1)$$

then, we say that $[Q_i]$ satisfy **IGM** *for Q_{jt} under $\boldsymbol{\tau}$. In this case, we also say that $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, or that $[Q_i]$ are factors of Q_{jt}.*

Simply put, the optimal joint actions across agents are equivalent to the collection of individual optimal actions of each agent. If $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ in a given task is factorizable under all $\boldsymbol{\tau} \in \mathcal{T}^N$, we say that the task itself is factorizable.

2.3. VDN and QMIX

Given Q_{jt}, one can consider the following two sufficient conditions for **IGM**:

$$\textbf{(Additivity)} \quad Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \sum_{i=1}^{N} Q_i(\tau_i, u_i), \quad (2)$$

$$\textbf{(Monotonicity)} \quad \frac{\partial Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})}{\partial Q_i(\tau_i, u_i)} \geq 0, \quad \forall i \in \mathcal{N}. \quad (3)$$

VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) are methods that attempt to factorize Q_{jt} assuming additivity and monotonicity, respectively. Thus, joint action-value functions satisfying those conditions would be well-factorized by VDN and QMIX. However, there exist tasks whose joint action-value functions do not meet the said conditions. We illustrate this limitation of VDN and QMIX using a simple matrix game in the next section.

3. QTRAN: Learning to Factorize with Transformation

In this section, we propose a new method called QTRAN, aiming at factorizing any factorizable task. The key idea is to transform the original joint action-value function Q_{jt} into a new one Q'_{jt} that shares the optimal joint action with Q_{jt}.

3.1. Conditions for the Factor Functions $[Q_i]$

For a given joint observation $\boldsymbol{\tau}$, consider an arbitrary factorizable $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$. Then, by Definition 1 of **IGM**, we can find individual action-value functions $[Q_i(\tau_i, u_i)]$ that factorize $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$. Theorem 1 states the sufficient condition for $[Q_i]$ that satisfy **IGM**. Let $\bar{u}_i$ denote the optimal local action $\arg\max_{u_i} Q_i(\tau_i, u_i)$ and $\bar{\boldsymbol{u}} = [\bar{u}_i]_{i=1}^N$,. Also, let $\boldsymbol{Q} = [Q_i] \in \mathbb{R}^N$, *i.e.*, a column vector of $Q_i, i = 1, \ldots, N$.

Theorem 1. *A factorizable joint action-value function $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is factorized by $[Q_i(\tau_i, u_i)]$, if*

$$\sum_{i=1}^{N} Q_i(\tau_i, u_i) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau}) = \begin{cases} 0 & \boldsymbol{u} = \bar{\boldsymbol{u}}, \quad (4a) \\ \geq 0 & \boldsymbol{u} \neq \bar{\boldsymbol{u}}, \quad (4b) \end{cases}$$

where

$$V_{\text{jt}}(\boldsymbol{\tau}) = \max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \sum_{i=1}^{N} Q_i(\tau_i, \bar{u}_i).$$

The proof is provided in the Supplementary. We note that conditions in (4) are also *necessary* under an affine transformation. That is, there exists an affine transformation $\phi(\boldsymbol{Q}) = A \cdot \boldsymbol{Q} + B$, where $A = [a_{ii}] \in \mathbb{R}_+^{N \times N}$ is a symmetric diagonal matrix with $a_{ii} > 0, \forall i$ and $B = [b_i] \in \mathbb{R}^N$, such that if Q_{jt} is factorized by $[Q_i]$, then (4) holds by replacing Q_i with $a_{ii}Q_i + b_i$. This is because for all i, b_i cancels out, and a_{ii} just plays the role of re-scaling the value of $\sum_{i=1}^N Q_i$ in multiplicative (with a positive scaling constant) and additive manners, since **IGM** is invariant to ϕ of $[Q_i]$.

Factorization via transformation We first define a new function Q'_{jt} as the linear sum of individual factor functions $[Q_i]$:

$$Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) := \sum_{i=1}^{N} Q_i(\tau_i, u_i). \quad (5)$$

We call $Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ the *transformed joint-action value function* throughout this paper. Our idea of factorization is as follows: from the additive construction of Q'_{jt} based on $[Q_i]$, $[Q_i]$ satisfy **IGM** for the new joint action-value function Q'_{jt}, implying that $[Q_i]$ are also the factorized individual action-value functions of Q'_{jt}. From the fact that $\arg\max_{\boldsymbol{u}} Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) = \arg\max_{\boldsymbol{u}} Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$, finding $[Q_i]$ satisfying (4) is precisely the factorization of $Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$.

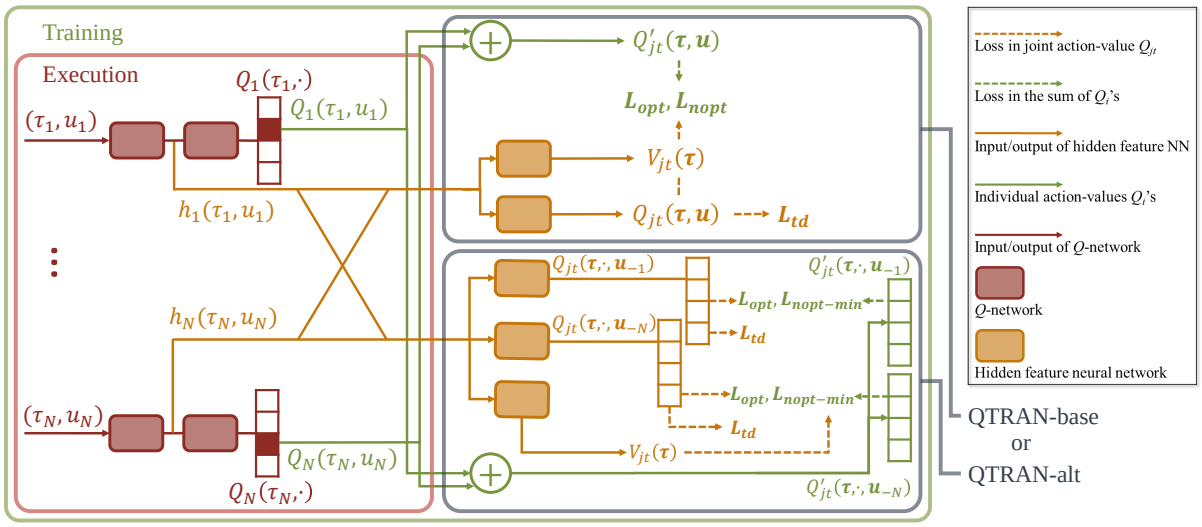


Figure 1. QTRAN-base and QTRAN-alt Architecture

One interpretation of this process is that rather than directly factorizing Q_{jt}, we consider an alternative joint action-value function (*i.e.*, Q'_{jt}) that is factorized by additive decomposition. The function $V_{\text{jt}}(\boldsymbol{\tau})$ corrects for the discrepancy between the centralized joint action-value function Q_{jt} and the sum of individual joint action-value functions $[Q_i]$. This discrepancy arises from the partial observability of agents. If bestowed with full observability, V_{jt} can be re-defined as zero, and the definitions would still stand. Refer to the Supplementary for more detail.

3.2. Method

Overall framework In this section, we propose a new deep RL framework with value function factorization, called QTRAN, whose architectural sketch is given in Figure 1. QTRAN consists of three separate estimators: (i) each agent i's individual action-value network for Q_i, (ii) a joint action-value network for Q_{jt} to be factorized into individual action-value functions Q_i, and (iii) a state-value network for V_{jt}, *i.e.*,

$$\begin{aligned}\textbf{(Individual action-value network)} \quad & f_{\text{q}} : (\tau_i, u_i) \mapsto Q_i,\\ \textbf{(Joint action-value network)} \quad & f_{\text{r}} : (\boldsymbol{\tau}, \boldsymbol{u}) \mapsto Q_{\text{jt}},\\ \textbf{(State-value network)} \quad & f_{\text{v}} : \boldsymbol{\tau} \mapsto V_{\text{jt}}.\end{aligned}$$

Three neural networks are trained in a centralized manner, and each agent uses its own factorized individual action-value function Q_i to take action during decentralized execution. Each network is elaborated next.

Individual action-value networks For each agent, an action-value network takes its own action-observation history τ_i as input, and produces action-values $Q_i(\tau_i,\cdot)$ as output. This action-value network is used for each agent to determine its own action by calculating the action-value for a given τ_i. As defined in (5), Q'_{jt} is just the summation of the outputs of all agents.

Joint action-value network The joint action-value network approximates Q_{jt}. It takes as input the selected action and produces the Q-value of the chosen action as output. For scalability and sample efficiency, we design this network as follows. First, we use the action vector sampled by all individual action-value networks to update the joint action-value network. Since the joint action space is $\mathcal{U}^N$, finding an optimal action requires high complexity as the number of agents N grows, whereas obtaining an optimal action in each individual network is done by decentralized policies with linear-time individual $\arg\max$ operations. Second, the joint action-value network shares the parameters at the lower layers of individual networks, where the joint action-value network combines hidden features with summation $\sum_i h_{Q,i}(\tau_i,u_i)$ of $h_i(\tau_i,u_i) = [h_{Q,i}(\tau_i,u_i), h_{V,i}(\tau_i)]$ from individual networks. This parameter sharing is used to enable scalable training with good sample efficiency at the expense of expressive power.

State-value network The state-value network is responsible for computing a scalar state-value, similar to $V(s)$ in the dueling network (Wang et al., 2016). V_{jt} is required to provide the flexibility to match Q_{jt} and $Q'_{\text{jt}} + V_{\text{jt}}$ at $\arg\max$. Without state-value network, partial observability would limit the representational complexity of Q'_{jt}. The state-value is independent of the selected action for a given $\boldsymbol{\tau}$. Thus, this value network does not contribute to choosing an action, and is instead used to calculate the loss of (4). Like the joint action-value network, we use the combined hidden features $\sum_i h_{V,i}(\tau_i)$ from the individual networks as input to the value network for scalability.

3.3. Loss Functions

There are two major goals in centralized training. One is that we should train the joint action-value function Q_{jt} to estimate the true action-value; the other is that the transformed action-value function Q'_{jt} should "track" the joint action-value function in the sense that their optimal actions are equivalent. We use the algorithm introduced in DQN (Mnih et al., 2015) to update networks, where we maintain a target network and a replay buffer. To this end, we devise the following global loss function in QTRAN, combining three loss functions in a weighted manner:

$$L(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) = L_{\text{td}} + \lambda_{\text{opt}} L_{\text{opt}} + \lambda_{\text{nopt}} L_{\text{nopt}}, \tag{6}$$

where r is the reward for action $\boldsymbol{u}$ at observation histories $\boldsymbol{\tau}$ with transition to $\boldsymbol{\tau}'$. L_{td} is the loss function for estimating the true action-value, by minimizing the TD-error as Q_{jt} is learned. L_{opt} and L_{nopt} are losses for factorizing Q_{jt} by $[Q_i]$ satisfying condition (4). The role of L_{nopt} is to check at each step if the action selected in the samples satisfied (4b), and L_{opt} confirms that the optimal local action obtained satisfies (4a). One could implement (4) by defining a loss depending on how well the networks satisfy (4a) or (4b) with actions taken in the samples. However, in this way, verifying whether (4a) is indeed satisfied would take too many samples since optimal actions are seldom taken at training. Since we aim to learn Q'_{jt} and V_{jt} to factorize for a given Q_{jt}, we stabilize the learning by fixing Q_{jt} when learning with L_{opt} and L_{nopt}. We let $\hat{Q}_{\text{jt}}$ denote this fixed Q_{jt}. λ_{opt} and λ_{nopt} are the weight constants for two losses. The detailed forms of L_{td}, L_{opt}, and L_{nopt} are given as follows, where we omit their common function arguments $(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}')$ in loss functions for presentational simplicity:

$$\begin{aligned}
L_{\text{td}}(;\boldsymbol{\theta}) &= \left(Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - y^{\text{dqn}}(r, \boldsymbol{\tau}'; \boldsymbol{\theta}^-)\right)^2, \\
L_{\text{opt}}(;\boldsymbol{\theta}) &= \left(Q'_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}}) + V_{\text{jt}}(\boldsymbol{\tau})\right)^2, \\
L_{\text{nopt}}(;\boldsymbol{\theta}) &= \left(\min\left[Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau}), 0\right]\right)^2,
\end{aligned}$$

where $y^{\text{dqn}}(r, \boldsymbol{\tau}'; \boldsymbol{\theta}^-) = r + \gamma Q_{\text{jt}}(\boldsymbol{\tau}', \bar{\boldsymbol{u}}'; \boldsymbol{\theta}^-)$, $\bar{\boldsymbol{u}}' = [\arg\max_{u_i} Q_i(\tau'_i, u_i; \boldsymbol{\theta}^-)]_{i=1}^N$, and $\boldsymbol{\theta}^-$ are the periodically copied parameters from $\boldsymbol{\theta}$, as in DQN (Mnih et al., 2015).

3.4. Tracking the Joint Action-value Function Differently

We name the method previously discussed **QTRAN-base**, to reflect the basic nature of how it keeps track of the joint action-value function. Here on, we consider a variant of QTRAN, which utilizes a counterfactual measure. As mentioned earlier, Theorem 1 is used to enforce **IGM** by (4a) and determine how the individual action-value functions $[Q_i]$ and the State-value function V_{jt} jointly "track" Q_{jt} by (4b), which governs the stability of constructing the correct factorizing Q_i's. We found that condition (4b) is often too loose, leading the neural networks to fail their mission of constructing the correct factors of Q_{jt}. That is, condition (4b) imposes undesirable influence on the non-optimal actions, which in turn compromises the stability and/or convergence speed of the training process. This motivates us to study conditions stronger than (4b) that would still be sufficient for factorizability, but at the same time would also be necessary under the aforementioned affine transformation ϕ, as in Theorem 1.

Theorem 2. *The statement presented in* Theorem 1 *and the necessary condition of* Theorem 1 *holds by replacing* (4b) *with the following* (7)*: if* $\boldsymbol{u} \neq \bar{\boldsymbol{u}}$*,*

$$\begin{aligned}
\min_{u_i \in \mathcal{U}} \Big[& Q'_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) - Q_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) \\
& + V_{\text{jt}}(\boldsymbol{\tau}) \Big] = 0, \quad \forall i = 1, \dots, N,
\end{aligned} \tag{7}$$

where $\boldsymbol{u}_{-i} = (u_1, \dots, u_{i-1}, u_{i+1}, \dots, u_N)$, i.e.*, the action vector except for* i*'s action.*

The proof is presented in the Supplementary. The key idea behind (7) lies in what conditions to enforce on *non-optimal* actions. It stipulates that $Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) + V_{\text{jt}}(\boldsymbol{\tau})$ be set to zero for some actions. Now, it is not possible to zero this value for every action u, but it is available for at least one action whilst still abiding by Theorem 1. It is clear that condition (7) is stronger than condition (4b), as desired. For non-optimal actions $\boldsymbol{u} \neq \bar{\boldsymbol{u}}$, the conditions of Theorem 1 are satisfied when $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) - V_{\text{jt}}(\boldsymbol{\tau}) \leq Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u}) \leq Q'_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}})$ for any given $\boldsymbol{\tau}$. Under this condition, however, there can exist a non-optimal action $\boldsymbol{u} \neq \bar{\boldsymbol{u}}$ whose $Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is comparable to $Q'_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}})$ but $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ is much smaller than $Q_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}})$. This may cause instability in the practical learning process. However, the newly devised condition (7) compels $Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ to track $Q_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ even for the problematic non-optimal actions mentioned above. This helps in widening the gap between $Q'_{\text{jt}}(\boldsymbol{\tau}, \boldsymbol{u})$ and $Q'_{\text{jt}}(\boldsymbol{\tau}, \bar{\boldsymbol{u}})$, and this gap makes the algorithm more stable. Henceforth, we call the deep MARL method outlined by Theorem 2 **QTRAN-alt**, to distinguish it from the one due to condition (4b).

Counterfactual joint networks To reflect our idea of (7), we now propose a counterfactual joint network, which replaces the joint action-value network of QTRAN-base, to efficiently calculate $Q_{\text{jt}}(\boldsymbol{\tau}, \cdot, \boldsymbol{u}_{-i})$ and $Q'_{\text{jt}}(\boldsymbol{\tau}, \cdot, \boldsymbol{u}_{-i})$ for all $i \in \mathcal{N}$ with only one forward pass. To this end, in the QTRAN-alt module, each agent has a counterfactual joint network with the output of $Q_{\text{jt}}(\boldsymbol{\tau}, \cdot, \boldsymbol{u}_{-i})$ for each possible action, given other agents' actions. As a joint action-value network, we use $h_{V,i}(\tau_i)$ and the combined hidden features $\sum_{j \neq i} h_{Q,j}(\tau_j, u_j)$ from other agents. Finally, $Q'_{\text{jt}}(\boldsymbol{\tau}, \cdot, \boldsymbol{u}_{-i})$ is calculated as $Q_i(\tau_i, \cdot) + \sum_{j \neq i} Q_j(\tau_j, u_j)$ for all agents. This architectural choice is realized by choosing the loss function to be $L_{\text{nopt-min}}$, replacing L_{nopt} in

u_1 \ u_2	A	B	C
A	**8**	-12	-12
B	-12	0	0
C	-12	0	0

(a) Payoff of matrix game

Q_1 \ Q_2	**4.16**(A)	2.29(B)	2.29(C)
3.84(A)	**8.00**	6.13	6.12
-2.06(B)	2.10	0.23	0.23
-2.25(C)	1.92	0.04	0.04

(b) QTRAN: Q_1, Q_2, Q'_{jt}

u_1 \ u_2	A	B	C
A	**8.00**	-12.02	-12.02
B	-12.00	0.00	0.00
C	-12.00	0.00	-0.01

(c) QTRAN: Q_{jt}

u_1 \ u_2	A	B	C
A	**0.00**	18.14	18.14
B	14.11	0.23	0.23
C	13.93	0.05	0.05

(d) QTRAN: $Q'_{\text{jt}} - Q_{\text{jt}}$

Q_1 \ Q_2	-3.14(A)	**-2.29**(B)	-2.41(C)
-2.29(A)	-5.42	-4.57	-4.70
-1.22(B)	-4.35	-3.51	-3.63
-0.73(C)	-3.87	**-3.02**	-3.14

(e) VDN: Q_1, Q_2, Q_{jt}

Q_1 \ Q_2	-0.92(A)	0.00(B)	**0.01**(C)
-1.02(A)	-8.08	-8.08	-8.08
0.11(B)	-8.08	0.01	**0.03**
0.10(C)	-8.08	0.01	0.02

(f) QMIX: Q_1, Q_2, Q_{jt}

Table 1. Payoff matrix of the one-step game and reconstructed Q_{jt} results on the game. Boldface means optimal/greedy actions from the state-action value

QTRAN-base as follows:

$$L_{\text{nopt-min}}(\boldsymbol{\tau}, \boldsymbol{u}, r, \boldsymbol{\tau}'; \boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\min_{u_i \in \mathcal{U}} D(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}))^2,$$

where

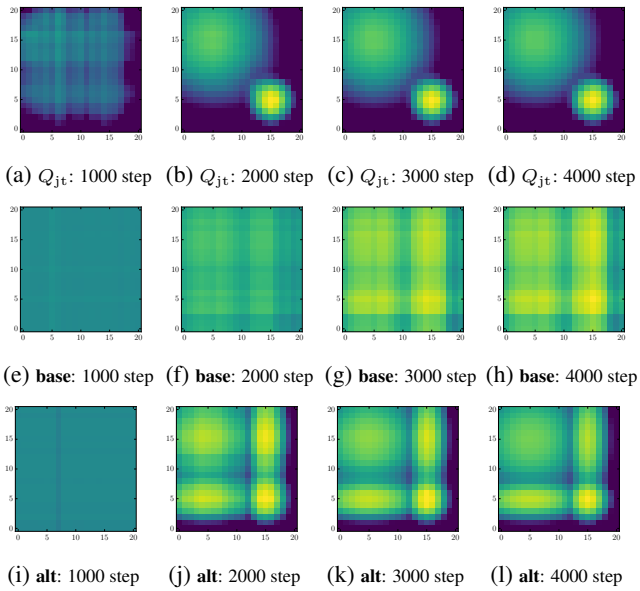
$$D(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) = Q'_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) - \hat{Q}_{\text{jt}}(\boldsymbol{\tau}, u_i, \boldsymbol{u}_{-i}) + V_{\text{jt}}(\boldsymbol{\tau}).$$

In QTRAN-alt, $L_{\text{td}}, L_{\text{opt}}$ are also used, but they are also computed for all agents.

3.5. Example: Single-state Matrix Game

In this subsection, we present how QTRAN performs compared to existing value factorization ideas such as VDN and QMIX, and how the two variants QTRAN-base and QTRAN-alt behave. The matrix game and learning results are shown in Table 1[1]. This symmetric matrix game has the optimal joint action (A, A), and captures a very simple cooperative multi-agent task, where we have two users with three actions each. Evaluation with more complicated tasks are provided in the next subsection. We show the results of VDN, QMIX, and QTRAN through a full exploration (*i.e.*, $\epsilon = 1$ in ϵ-greedy) conducted over 20,000 steps. Full exploration guarantees to explore all available game states. Therefore, we can compare only the expressive power of the methods. Other details are included in the Supplementary.

Comparison with VDN and QMIX Tables 1b-1f show the learning results of QTRAN, VDN, and QMIX. Table 1b shows that QTRAN enables each agent to jointly take the optimal action only by using its own locally optimal action, meaning successful factorization. Note that Tables 1c and 1d demonstrate the difference between Q_{jt} and Q'_{jt}, stemming from our transformation, where their optimal actions are the nonetheless same. Table 1d shows that QTRAN also satisfies (4), thereby validating our design principle as described in Theorem 1. However, in VDN, agents 1's and 2's individual optimal actions are C and B, respectively. VDN fails to factorize because the structural constraint of additivity $Q_{\text{jt}}(\boldsymbol{u}) = \sum_{i=1,2} Q_i(u_i)$ is enforced, leading to deviations from **IGM**, whose sufficient condition is additivity, *i.e.*, $Q_{\text{jt}}(\boldsymbol{u}) = \sum_{i=1,2} Q_i(u_i)$ for all $\boldsymbol{u} = (u_1, u_2)$. QMIX also fails in factorization in a similar manner due to the structural constraint of monotonicity.



(a) Q_{jt}: 1000 step (b) Q_{jt}: 2000 step (c) Q_{jt}: 3000 step (d) Q_{jt}: 4000 step

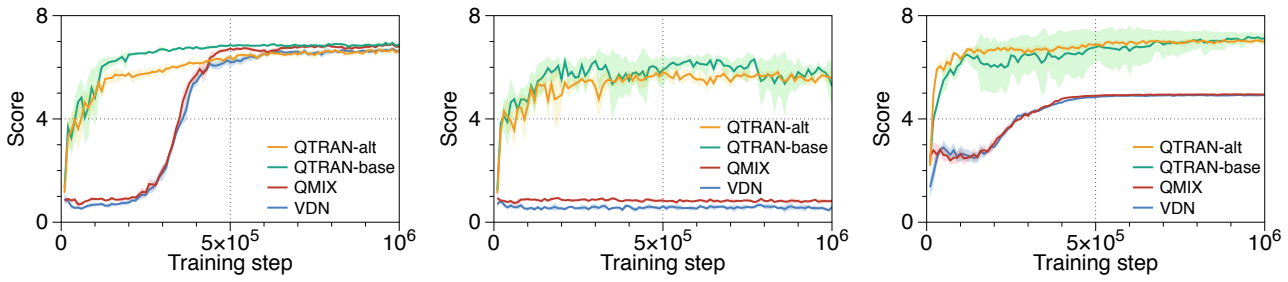
(e) **base**: 1000 step (f) **base**: 2000 step (g) **base**: 3000 step (h) **base**: 4000 step

(i) **alt**: 1000 step (j) **alt**: 2000 step (k) **alt**: 3000 step (l) **alt**: 4000 step

Figure 2. **base**: QTRAN-base, **alt**: QTRAN-alt. x-axis and y-axis: agents 1 and 2's actions, respectively. Colored values represent the values of Q_{jt} ((a)-(d)) and Q'_{jt} ((e)-(l)) for selected actions.

Impact of QTRAN-alt In order to see the impact of QTRAN-alt, we train the agents in a matrix game where two agents each have 21 actions. Figure 2 illustrates the joint action-value function and its transformations of both QTRAN-base and QTRAN-alt. The result shows that both algorithms successfully learn the optimal action by correctly estimating the Q_{jt} for $\boldsymbol{u} = \bar{\boldsymbol{u}}$ for any given state. Q'_{jt} values for non-optimal actions are different from Q_{jt}, but it has a different tendency in each algorithm as follows. As shown in Figures 2e-2h, all Q'_{jt} values in QTRAN-base have only a small difference from the maximum value of Q_{jt}, whereas Figures 2i-2l show that QTRAN-alt has the ability to more accurately distinguish optimal actions from non-optimal actions. Thus, in QTRAN-alt, the agent can smartly explore and have better sample efficiency to train the networks. This feature of QTRAN-alt also prevents learning unsatisfactory policies in complex environments. Full details on the experiment are included in the Supplementary.

[1]We present only Q_{jt} and Q'_{jt}, because in fully observable cases (*i.e.*, observation function is bijective for all i) Theorem 1 holds for $V_{\text{jt}}(\tau) = 0$. We discuss further in Supplementary.



(a) Gaussian Squeeze (b) Gaussian Squeeze without epsilon decay (c) Multi-domain Gaussian Squeeze $K = 2$

Figure 3. Average reward on the GS and GMS tasks with 95% confidence intervals for VDN, QMIX, and QTRAN

4. Experiment

4.1. Environments

To demonstrate the performance of QTRAN, we consider two environments: (i) Multi-domain Gaussian Squeeze and (ii) modified predator-prey. Details on our implementation of QTRAN, the source code of our implementation in TensorFlow, and other experimental scripts are available in the Supplementary and a public repository: `https://github.com/Sonkyunghwan/QTRAN`.

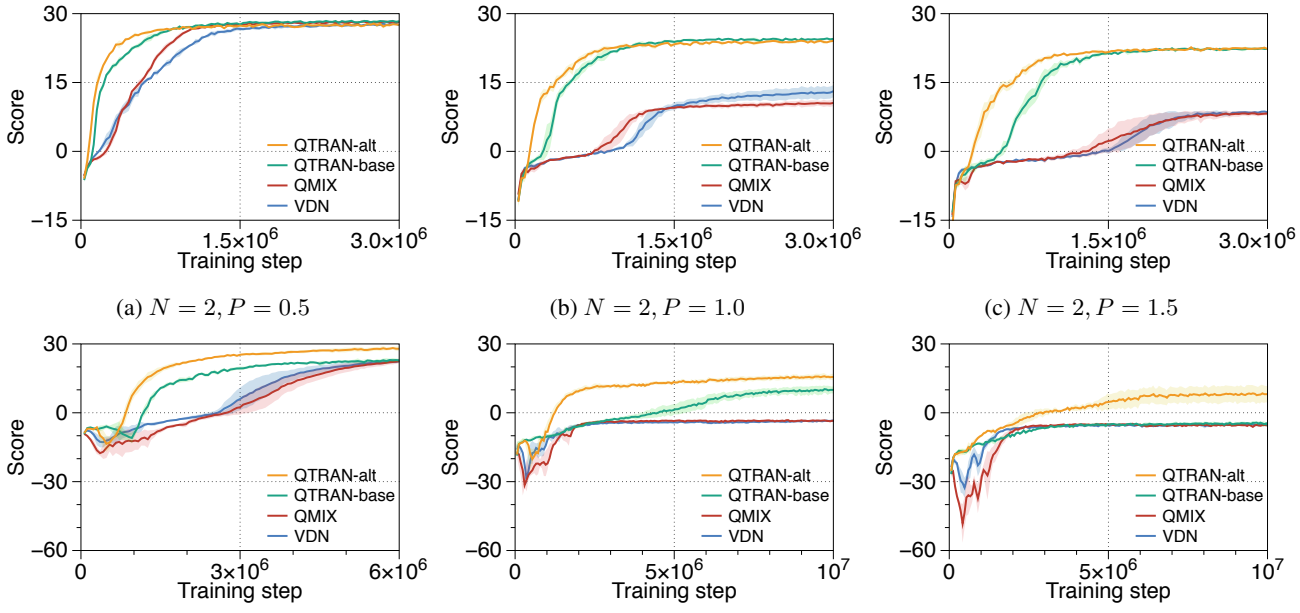
Multi-domain Gaussian Squeeze (MGS) Gaussian Squeeze (GS) (HolmesParker et al., 2014) is a simple non-monotonic multi-agent resource allocation problem, used in other papers, *e.g.*, Yang et al. (2018) and Chen et al. (2018). In GS, multiple homogeneous agents need to work together for efficient resource allocation whilst avoiding congestion. We use GS in conjunction with Multi-domain Gaussian Squeeze (MGS) as follows: we have ten agents; each agent i takes action u_i, which controls the resource usage level, ranging over $\{0, 1, ..., 9\}$. Each agent has its own amount of unit-level resource $s_i \in [0, 0.2]$, given by the environment *a priori*. Then, a joint action $\boldsymbol{u}$ determines the overall resource usage $x(\boldsymbol{u}) = \sum_i s_i \times u_i$. We assume that there exist K domains, where the above resource allocation takes place. Then, the goal is to maximize the joint reward defined as $G(\boldsymbol{u}) = \sum_{k=1}^{K} xe^{-(x-\mu_k)^2/{\sigma_k}^2}$, where μ_k and σ_k are the parameters of each domain. Depending on the number of domains, GS has only one local maximum, whereas MGS has multiple local maxima. In our MGS setting, compared to GS, the optimal policy is similar to that in GS, and through this policy, the reward similar to that in GS can be obtained. Additionally, in MGS, a new sub-optimal "pitfall" policy that is easier to achieve but is only half as rewarding as the optimal policy. The case when $K > 1$ is usefully utilized to test the algorithms that are required to avoid sub-optimal points in the joint space of actions. The full details on the environment setup and hyperparameters are described in the Supplementary.

Modified predator-prey (MPP) We adopt a more complicated environment by modifying the well-known predator-prey (Stone & Veloso, 2000) in the grid world, used in many other MARL research. State and action spaces are constructed similarly to those of the classic predator-prey game. "Catching" a prey is equivalent to having the prey within an agent's observation horizon. We extend it to the scenario that positive reward is given only if multiple predators catch a prey simultaneously, requiring a higher degree of cooperation. The predators get a team reward of 1, if two or more catch a prey at the same time, but they are given negative reward $-P$, when only one predator catches the prey.

Note that the value of penalty P also determines the degree of monotonicity, *i.e.*, the higher P is, the less monotonic the task is. The prey that has been caught regenerated at random positions whenever caught by more than one predator. In our evaluation, we tested up to $N = 4$ predators and up to two prey, and the game proceeds over fixed 100 steps. We experimented with six different settings with varying P values and numbers of agents, where $N = 2, 4$ and $P = 0.5, 1.0, 1.5$. For the $N = 4$ case, we placed two prey; otherwise, just one. The detailed settings are available in the Supplementary.

4.2. Results

Multi-domain Gaussian Squeeze Figure 3a shows the result of GS, where it is not surprising to observe that all algorithms converge to the optimal point. However, QTRAN noticeably differs in its convergence speed; it is capable of handling the non-monotonic nature of the environment more accurately and of finding an optimal policy from well-expressed action-value functions. VDN and QMIX have some structural constraints, which hinder the accurate learning of the action-value functions for non-monotonic structures. These algorithms converge to the locally optimal point — which is the globally optimal point in GS, where $K = 1$ — near the biased samples by a wrong policy with epsilon-decay exploration. To support our claim, we experiment with the full exploration without epsilon-decay for the same environment, as shown in Figure 3b. We observe that QTRAN learns more or less the same policy as in Figure 3a, whereas VDN and QMIX significantly deteriorate. Fig-



(a) $N = 2, P = 0.5$ (b) $N = 2, P = 1.0$ (c) $N = 2, P = 1.5$

(d) $N = 4, P = 0.5$ (e) $N = 4, P = 1.0$ (f) $N = 4, P = 1.5$

Figure 4. Average reward per episode on the MPP tasks with 95% confidence intervals for VDN, QMIX, and QTRAN

ure 3c shows the result for a more challenging scenario of MGS, with each of $[s_i]$ being the same, where agents can always achieve higher performance than GS. VDN and QMIX are shown to learn only a sub-optimal policy in MGS, whose rewards are even smaller than those in GS. QTRAN-base and QTRAN-alt achieve significantly higher rewards, where QTRAN-alt is more stable, as expected. This is because QTRAN-alt's alternative loss increases the gap between Q'_{jt} for non-optimal and optimal actions, and prevents it from being updated to an undesirable policy.

Modified predator-prey Figure 4 shows the performance of the three algorithms for six settings with different N and P values, where all results demonstrate the superiority of QTRAN to VDN and QMIX. In Figures 4a and 4d with low penalties P, all three algorithms learn the policy that catches the prey well and thus obtain high rewards. However, again, the speed in finding the policy in VDN and QMIX is slower than that in QTRAN. As the penalty P grows and exacerbates the non-monotonic characteristic of the environment, we observe a larger performance gap between QTRAN and the two other algorithms. As shown in Figures 4b and 4c, even for a large penalty, QTRAN agents still cooperate well to catch the prey. However, in VDN and QMIX, agents learn to work together with somewhat limited coordination, so that they do not actively try to catch the prey and instead attempt only to minimize the penalty-receiving risk. In Figures 4e and 4f, when the number of agents N grows, VDN and QMIX do not cooperate at all and learn only a sub-optimal policy: running away from the prey to minimize the risks of being penalized.

For the tested values of N and P, we note that the performance gap between QTRAN-base and QTRAN-alt is influenced more strongly by N. When $N = 2$, the gap in the convergence speed between QTRAN-base and QTRAN-alt increases as the penalty grows. Nevertheless, both algorithms ultimately turn out to train the agents to cooperate well for every penalty value tested. On the other hand, with $N = 4$, the gap of convergence speed between QTRAN-base and QTRAN-alt becomes larger. With more agents, as shown in Figures 4d and 4e, QTRAN-alt achieves higher scores than QTRAN-base does, and QTRAN-base requires longer times to reach positive scores. Figure 4f shows that QTRAN-base does not achieve a positive reward until the end, implying that QTRAN-base fails to converge, whereas QTRAN-alt's score increases, with training steps up to 10^7.

5. Conclusion

This paper presented QTRAN, a learning method to factorize the joint action-value functions of a wide variety of MARL tasks. QTRAN takes advantage of centralized training and fully decentralized execution of the learned policies by appropriately transforming and factorizing the joint action-value function into individual action-value functions. Our theoretical analysis demonstrates that QTRAN handles a richer class of tasks than its predecessors, and our simulation results indicate that QTRAN outperforms VDN and QMIX by a substantial margin, especially so when the game exhibits more severe non-monotonic characteristics.

Acknowledgements

This work was supported by Institute for Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00170,Virtual Presence in Moving Objects through 5G (PriMo-5G))

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science and ICT(No. 2016R1A2A2A05921755)

References

Chen, Y., Zhou, M., Wen, Y., Yang, Y., Su, Y., Zhang, W., Zhang, D., Wang, J., and Liu, H. Factorized q-learning for large-scale multi-agent systems. *arXiv preprint arXiv:1809.03738*, 2018.

Foerster, J., Assael, I. A., de Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. In *Proceedings of the Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of AAAI*, 2018.

Guestrin, C., Koller, D., and Parr, R. Multiagent planning with factored mdps. In *Proceedings of the Advances in Neural Information Processing Systems*, pp. 1523–1530, 2002a.

Guestrin, C., Lagoudakis, M., and Parr, R. Coordinated reinforcement learning. In *Proceedings of ICML*, pp. 227–234, 2002b.

Gupta, J. K., Egorov, M., and Kochenderfer, M. Cooperative multi-agent control using deep reinforcement learning. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, 2017.

HolmesParker, C., Taylor, M., Zhan, Y., and Tumer, K. Exploiting structure and agent-centric rewards to promote coordination in large multiagent systems. In *Proceedings of Adaptive and Learning Agents Workshop*, 2014.

Jiang, J. and Lu, Z. Learning attentional communication for multi-agent cooperation. In *Proceedings of NIPS*, pp. 7265–7275, 2018.

Kim, D., Moon, S., Hostallero, D., Kang, W. J., Lee, T., Son, K., and Yi, Y. Learning to schedule communication in multi-agent reinforcement learning. In *Proceedings of ICLR*, 2019.

Kok, J. R. and Vlassis, N. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7(Sep):1789–1828, 2006.

Koller, D. and Parr, R. Computing factored value functions for policies in structured mdps. In *Proceedings of IJCAI*, pp. 1332–1339, 1999.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arxiv preprint arXiv:1509.02971*, 2015.

Lowe, R., WU, Y., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of NIPS*, pp. 6379–6390, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529, 2015.

Oliehoek, F. A., Spaan, M. T., and Vlassis, N. Optimal and approximate q-value functions for decentralized pomdps. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

Oliehoek, F. A., Amato, C., et al. *A concise introduction to decentralized POMDPs*, volume 1. Springer, 2016.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of ICML*, 2018.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Stone, P. and Veloso, M. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8 (3):345–383, 2000.

Sukhbaatar, S., Fergus, R., et al. Learning multiagent communication with backpropagation. In *Proceedings of the Advances in Neural Information Processing Systems*, pp. 2244–2252, 2016.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V. F., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of AAMAS*, pp. 2085–2087, 2018.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., and Vicente, R. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

Tan, M. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of ICML*, pp. 330–337, 1993.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1995–2003, 2016.

Watkins, C. J. C. H. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Wei, E. and Luke, S. Lenient learning in independent-learner stochastic cooperative games. *The Journal of Machine Learning Research*, 17(1):2914–2955, 2016.

Wei, E., Wicke, D., Freelan, D., and Luke, S. Multiagent soft q-learning. *arXiv preprint arXiv:1804.09817*, 2018.

Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., and Wang, J. Mean field multi-agent reinforcement learning. In *Proceedings of ICML*, pp. 5567–5576, 2018.