Weighted QMIX: Expanding Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning

Tabish Rashid, Gregory Farquhar*, Bei Peng, Shimon Whiteson
Department of Computer Science
University of Oxford
{tabish.rashid, gregory.farquhar, bei.peng, shimon.whiteson}@cs.ox.ac.uk

Abstract

QMIX is a popular $Q$-learning algorithm for cooperative MARL in the centralised training and decentralised execution paradigm. In order to enable easy decentralisation, QMIX restricts the joint action $Q$-values it can represent to be a monotonic mixing of each agent’s utilities. However, this restriction prevents it from representing value functions in which an agent’s ordering over its actions can depend on other agents’ actions. To analyse this representational limitation, we first formalise the objective QMIX optimises, which allows us to view QMIX as an operator that first computes the $Q$-learning targets and then projects them into the space representable by QMIX. This projection returns a representable $Q$-value that minimises the unweighted squared error across all joint actions. We show in particular that this projection can fail to recover the optimal policy even with access to $Q^*$, which primarily stems from the equal weighting placed on each joint action. We rectify this by introducing a weighting into the projection, in order to place more importance on the better joint actions. We propose two weighting schemes and prove that they recover the correct maximal action for any joint action $Q$-values, and therefore for $Q^*$ as well. Based on our analysis and results in the tabular setting, we introduce two scalable versions of our algorithm, Centrally-Weighted (CW) QMIX and Optimistically-Weighted (OW) QMIX and demonstrate improved performance on both predator-prey and challenging multi-agent StarCraft benchmark tasks [26].

1 Introduction

Many critical tasks involve multiple agents acting in the same environment. To learn good behaviours in such problems from agents’ experiences, we may turn to multi-agent reinforcement learning (MARL). Fully decentralised policies are often used in MARL, due to practical communication constraints or as a way to deal with an intractably large joint action space. However, when training in simulation or under controlled conditions we may have access to additional information, and agents can freely share their observations and internal states. Exploiting these possibilities can greatly improve the efficiency of learning [7, 9].

In this paradigm of centralised training for decentralised execution, QMIX [25] is a popular $Q$-learning algorithm with state-of-the-art performance on the StarCraft Multi-Agent Challenge [26]. QMIX represents the optimal joint action value function using a monotonic mixing function of per-agent utilities. This restricted function class $Q^{mix}$ allows for efficient maximisation during training, and easy decentralisation of the learned policy. However, QMIX is unable to represent joint action value functions that are characterised as nonmonotonic [16], i.e., an agent’s ordering over its

*Now at Google DeepMind.
own actions depends on other agents’ actions. Consequently, QMIX cannot solve tasks that require significant coordination within a given timestep [22]. In this work, we analyse an idealised, tabular version of QMIX to study this representational limitation, and then develop algorithms to resolve these limitations in theory and in practice.

We formalise the objective that QMIX optimises, which allows us to view QMIX as an operator that first computes the $Q$-learning targets and then projects them into $Q^{mix}$ by minimising the unweighted squared error across all joint actions. We show that, since in general $Q^* \notin Q^{mix}$, the projection of $Q^*$, which we refer to as $Q_{tot}$, can have incorrect estimates for the optimal joint action, yielding suboptimal policies. These are fundamental limitations of the QMIX algorithm independent from exploration and compute constraints, and occur even with access to the true $Q^*$.

These limitations primarily arise because QMIX’s projection of $Q^*$ yields a $Q_{tot}$ that places equal importance on approximating the $Q$-values for all joint actions. Our key insight is that if we ultimately care only about the greedy optimal policy, it is more important to accurately represent the value of the optimal joint action than the suboptimal ones. Therefore, we can improve the policy recovered from $Q_{tot}$ by appropriately weighting each joint action when projecting $Q^*$ into $Q^{mix}$.

Based on this intuition, we introduce a weighting function into our projection. In the idealised tabular setting we propose two weighting functions and prove that the projected $Q_{tot}$ recovers the correct maximal action for any $Q$, and therefore for $Q^*$ as well. Since this projection always recovers the correct maximal joint action, we benefit from access to $Q^*$ (or a learned approximation of it). To this end, we introduce a learned approximation of $Q^*$, from an unrestricted function class, which we call $\hat{Q}^*$. By using $Q_{tot}$, now a weighted projection of $\hat{Q}^*$, to perform maximisation, we show that $\hat{Q}^*$ converges to $Q^*$ and that $Q_{tot}$ thus recovers the optimal policy.

Based on our analysis and results in the tabular setting, we present two scalable versions of our algorithm, Centrally-Weighted (CW) QMIX and Optimistically-Weighted (OW) QMIX. We demonstrate their improved ability to cope with environments with nonmonotonic value functions, by showing superior performance in a predator-prey task in which the trade-offs made by QMIX prevent it from solving the task. Additionally, we demonstrate improved robustness over QMIX to the amount of exploration performed, by showing better empirical performance on a range of SMAC maps. Our ablations and additional analysis experiments demonstrate the importance of both a weighting and an unrestricted $Q^*$ in our algorithm.

2 Background

A fully cooperative multi-agent sequential decision-making task can be described as a decentralised partially observable Markov decision process (Dec-POMDP) [21] consisting of a tuple $G = (S, U, P, r, Z, O, n, \gamma)$. $s \in S$ describes the true state of the environment. At each time step, each agent $a \in A \equiv \{1, \ldots, n\}$ chooses an action $u_a \in U$, forming a joint action $u \in U \equiv U^n$. This causes a transition on the environment according to the state transition function $P(s'|s, u) : S \times U \times S \rightarrow [0, 1]$. All agents share the same reward function $r(s, u) : S \times U \rightarrow \mathbb{R}$ and $\gamma \in [0, 1)$ is a discount factor.

Due to the partial observability, each agent’s individual observations $z \in Z$ are produced by the observation function $O(s, a) : S \times A \rightarrow Z$. Each agent has an action-observation history $\tau_a \in T \equiv (Z \times U)^*$, on which it conditions a (potentially stochastic) policy $\pi^a(u_a|\tau_a) : T \times U \rightarrow [0, 1]$, $\pi$ denotes the action-observation histories of all agents (up to that timestep). The joint policy $\pi$ has a joint action-value function: $Q^\pi(s_t, u_t) = \mathbb{E}_{s_{t+1} \sim \cdot, u_{t+1} \sim \cdot, \tau_t \sim \cdot} [R_t|s_t, u_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return. For our idealised tabular setting, we consider a fully observable setting in which each agent’s observations are the full state. This is equivalent to a multi-agent MDP (MMDP) [21] which is itself equivalent to a standard MDP with $U^n$ as the action space.

We adopt the centralised training and decentralised execution paradigm [20] [14]. During training our learning algorithm has access to the true state $s$ and every agent’s action-observation history, as well as the freedom to share all information between agents. However, during testing (execution), each agent has access only to its own action-observation history.
2.1 QMIX

The Bellman optimality operator is defined by:

\[ T^* Q(s, u) := \mathbb{E}[r + \gamma \max_{u'} Q(s', u')] \tag{1} \]

where the expectation is over the next state \( s' \sim P(\cdot|s, u) \) and reward \( r \sim r(\cdot|s, u) \). Algorithms based on Q-learning\(^\text{[34]}\) use samples from the environment to estimate the expectation in (1), in order to update their estimates of \( Q^* \).

VDN\(^\text{[29]}\) and QMIX are Q-learning algorithms for the cooperative MARL setting, which estimate the optimal joint action value function \( Q^* \) as \( Q_{\text{tot}} \), with specific forms. VDN factorises \( Q_{\text{tot}} \) into a sum of the per-agent utilities: \( Q_{\text{tot}}(s, u) = \sum_{a=1}^n Q_a(s, u_a) \), whereas QMIX combines the per-agent utilities via a continuous monotonic function that is state-dependent: \( f_x(Q_1(s, u_1), ..., Q_n(s, u_n)) = Q_{\text{tot}}(s, u) \), where \( \frac{\partial f_x}{\partial Q_a} \geq 0, \forall a \in A \equiv \{1, ..., n\} \).

In the deep RL setting neural networks with parameters \( \theta \) are used as function approximators, and QMIX is trained much like a DQN\(^\text{[19]}\). Considering only the fully-observable setting for ease of presentation, a replay buffer stores transition tuples \( (s, u, r, s', d) \), in which the agents take joint action \( u \) in state \( s \), receive reward \( r \) and transition to \( s' \) and \( d \) is a boolean indicating if \( s' \) is a terminal state. QMIX is trained to minimise the squared TD error on a minibatch of \( b \) samples from the replay buffer: \( \sum_{i=1}^{b} (Q_{\text{tot}}(s, u; \theta) - y_i)^2 \), where \( y_i = r + \gamma \max_{u'} Q_{\text{tot}}(s', u'; \theta^-) \) are the targets, and \( \theta^- \) are the parameters of a target network that are periodically copied from \( \theta \). The monotonic mixing function \( f_x \) is parametrised as a feedforward network, whose non-negative weights are generated by hypernetworks\(^\text{[10]}\) that take the state as input.

3 QMIX Operator

In this section we examine an operator that represents an idealised version of QMIX in a tabular setting. The purpose of our analysis is primarily to understand the fundamental limitations of QMIX that stem from its training objective and the restricted function class it uses. We write this function class as \( Q_{\text{mix}}^* \):

\[ Q_{\text{mix}}^* := \{Q_{\text{tot}}|Q_{\text{tot}}(s, u) = f_x(Q_1(s, u_1), ..., Q_n(s, u_n)), \frac{\partial f_x}{\partial Q_a} \geq 0, Q_a(s, u) \in \mathbb{R}\} \]

This is the space of all \( Q_{\text{tot}} \) that can be represented by monotonic functions of tabular \( Q_a(s, u) \). At each iteration of our idealised algorithm, we constrain \( Q_{\text{tot}} \) to lie in \( Q_{\text{mix}}^* \) by solving the following optimisation problem:

\[ \arg\min_{q \in Q_{\text{mix}}^*} \sum_{s,u \in U} (T^* Q_{\text{tot}}(s, u) - q(s, u))^2. \tag{2} \]

To avoid any confounding factors regarding exploration, we assume this optimisation is performed for all states and joint actions at each iteration, as in planning algorithms like value iteration\(^\text{[24]}\). We also assume the optimisation is performed exactly. However, since it is not guaranteed to have a unique solution, a random \( q \) is returned from the set of objective-minimising candidates.

By contrast, the original QMIX algorithm in the deep RL setting interleaves exploration and approximate optimisation of this objective, using samples from the environment to approximate \( T^* \), and a finite number of gradient descent steps to estimate the argmin. Additionally, sampling uniformly from a replay buffer of the most recent experiences does not strictly lead to a uniform weighting across joint actions. Instead the weighting is proportional to the frequency at which a joint action was taken. Incorporating this into our analysis would introduce significant complexity and detract from our main focus: to analyse the limitations of restricting the representable function class to \( Q_{\text{mix}}^* \).

The optimisation in (2) can be separated into two distinct parts: the first computes targets using \( T^* \), and the second projects those targets into \( Q_{\text{mix}}^* \). We define the corresponding projection operator \( \Pi_{\text{QMIX}} \) as follows:

\[ \Pi_{\text{QMIX}} Q := \arg\min_{q \in Q_{\text{mix}}^*} \sum_{s,u \in U} (Q(s, u) - q(s, u))^2 \]

We can then define \( T_{\text{QMIX}}^* := \Pi_{\text{QMIX}} T^* \) as the QMIX operator, which exactly corresponds to the objective in (2).
There is a significant literature studying projections of value functions into linear function spaces for RL (see [30] for a detailed introduction and overview). However, despite some superficial similarities, our focus is considerably different in several ways. First, we consider a nonlinear projection, rendering many methods of analysis inapplicable. Second, as the $Q(s, a)$ are tabular, there is no tradeoff in the quality of representation across different states, and consequently no need to weight the different states in the projection. By contrast, unlike linear $Q$-functions, our restricted space does induce tradeoffs in the quality of representation across different joint actions, and weighting them differently in the projection is central to our method. Third, the targets in our optimisation are fixed to $T^*Q_{tot}(s, a)$ at each iteration, rather than depending on $q$ as they would in a minimisation of Mean Squared Bellman Error (MSBE) or Mean Squared Projected Bellman Error (MSPBE) in linear RL. This makes our setting closer to that of fitted $Q$-iteration [6] in which a regression problem is solved at each iteration to fit a function approximator from a restricted class to $Q$-learning-style targets. Our focus of study is the unique properties of the particular function space $Q_{mix}$, and the tradeoffs in representation quality induced by projection into it.

3.1 Properties of $T_{Q_{mix}}^*$

To highlight the pitfalls of the projection $\Pi_{Q_{mix}}$ into $Q_{mix}$, we consider the effect of applying $\Pi_{Q_{mix}}$ to the true $Q^*$, which is readily available in deterministic normal form games where $Q^*$ is just the immediate reward.

$T_{Q_{mix}}^*$ is not a contraction. The payoff matrix in Table 1 (Left) is a simple example of a value function that cannot be perfectly represented in $Q_{mix}$. Table 1 (Middle) and (Right) show two distinct $Q_{tot}$, both of which are global minima of the optimisation solved by $\Pi_{Q_{mix}}Q^*$. Hence, $T_{Q_{mix}}^*$ is not a contraction, which would have a unique fixed point.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1/3</th>
<th>1/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1/3</td>
<td>1/3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1/3</td>
<td>1</td>
<td>1/3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/3</td>
<td>1</td>
<td>1/3</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/3</td>
<td>1/3</td>
<td>1</td>
<td>1/3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Non-monotonic payoff matrix (Left) and the two possible $Q_{tot}$ returned by $\Pi_{Q_{mix}}$ (Middle and Right).

Qmix’s argmax is not always correct. There exist $Q$-functions such that $\text{argmax } \Pi_{Q_{mix}} Q \neq \text{argmax } Q$. For example, the payoff matrix in Table 2 (Left) (from Son et al. [27]) produces a value function for which QMIX’s approximation (Right) does not result in the correct argmax.

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>-12</th>
<th>-12</th>
<th>-12</th>
<th>-12</th>
</tr>
</thead>
<tbody>
<tr>
<td>-12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Payoff matrix (Left) in which $Q_{tot}$ returned from $\Pi_{Q_{mix}}$ has an incorrect argmax (Right).

QMIX can underestimate the value of the optimal joint action. Furthermore, if it has an incorrect argmax, the value of the true optimal joint action can be underestimated, e.g., $-12$ instead of $8$ in Table 2. If QMIX gets the correct argmax then it represents the maximum $Q$-value perfectly (proved formally in Appendix B). However, if QMIX’s argmax joint action is not the true optimal joint action then QMIX can underestimate the value of that action.

These failure modes are problematic because they show fundamental limitations of QMIX, that are independent from: 1) compute constraints, since we exactly minimise the objectives posed; 2) exploration, since we update every state-action pair; and 3) parametrisation of the mixing function and agent utilities, since we are assume that $Q_{tot}$ can be any member of $Q_{mix}$, whereas in practice we can only represent a subset of $Q_{mix}$.

4 Weighted QMIX Operator

In this section we introduce an operator for an idealised version of our algorithm, Weighted QMIX (WQmix), in order to compare it to the operator we introduced for QMIX.

The negative results in Section 3.1 concern the scenario in which we optimise QMIX’s loss function across all joint actions for every state. We argue that this equal weighting over joint actions when performing the optimisation in [3] is responsible for the possibly incorrect argmax of the objective-minimising solution. Consider the example in Table 2. A monotonic $Q_{tot} \in Q_{mix}$ cannot increase its estimate of the value of the single optimal joint action above $-12$ without either increasing the estimates of the value of the bad joint actions above their true value of $-12$, or decreasing the estimates of the zero-value joint actions below $-12$. The error for misestimating several of the suboptimal joint
actions would outweigh the improvement from better estimating the single optimal joint action. As a result the optimal action value is underestimated and the resulting policy is suboptimal.

By contrast, consider the extreme case in which we only optimise the loss for the single optimal joint action \( u^* \). For a single action, the representational limitation of QMIX has no effect so we can optimise the objective perfectly, recovering the value of the optimal joint action.

However, we still need to learn that the other action values are lower than \( Q_{tot}(u^*) \) in order to recover the optimal policy. To prioritise estimating \( Q_{tot}(u^*) \) well, while still anchoring down the value estimates for other joint actions, we can add a suitable weighting function \( w \) into the projection operator of QMIX:

\[
\Pi_w Q := \arg\min_{q \in Q^{mix}} \sum_{u \in U} w(s, u)(Q(s, u) - q(s, u))^2. \tag{3}
\]

The weighting function \( w : S \times U \to (0, 1] \) weights the importance of each joint action in QMIX’s loss function. It can depend on more than just the state and joint action, but we omit this from the notation for simplicity. Setting \( w(s, u) = 1 \) recovers the projection operator \( \Pi_{Q^{mix}} \).

4.1 Weightings

The choice of weighting is crucial to ensure that WQMIX can overcome the limitations of QMIX. As shown in Section 3, even if we have access to \( Q^* \), if we use a uniform weighting then we can still end up with the wrong argmax after projection into the monotonic function space. We now consider two different weightings and show in Theorems 1 and 2 that these choices of \( w \) ensure that the \( Q_{tot} \) returned from the projection has the correct argmax. The proofs of these theorems can be found in Appendix B. For both weightings, let \( \alpha \in (0, 1] \) and consider the weighted projection of an arbitrary joint action value function \( Q \).

**Idealised Central Weighting**  The first weighting, which we call Idealised Central Weighting, is quite simple:

\[
w(s, u) = \begin{cases} 
1 & u = u^* = \arg\max_u Q(s, u) \\
\alpha & \text{otherwise}
\end{cases}. \tag{4}
\]

To ensure that the weighted projection returns a \( Q_{tot} \) with the correct argmax, we simply down-weight every suboptimal action. However, this weighting requires computing the maximum across the joint action space, which is often infeasible. In Section 5 we discuss an approximation to this weighting in the deep RL setting.

**Theorem 1.** Let \( w \) be the Idealised Central Weighting from (4). Then \( \exists \alpha > 0 \) such that \( \arg\max \Pi_w Q = \arg\max Q \) for any \( Q \).

Theorem 1 provides a sanity check that this choice of weighting guarantees we recover a \( Q_{tot} \) with the correct argmax in this idealised setting, with a nonzero weighting for suboptimal actions.

**Optimistic Weighting**  The second weighting, which we call Optimistic Weighting, affords a practical implementation:

\[
w(s, u) = \begin{cases} 
1 & Q_{tot}(s, u) < Q(s, u) \\
\alpha & \text{otherwise}
\end{cases}. \tag{5}
\]

This weighting assigns a higher weighting to those joint actions that are underestimated relative to \( Q \), and hence could be the true optimal actions (in an optimistic outlook).

**Theorem 2.** Let \( w \) be the Optimistic Weighting from (5). Then \( \exists \alpha > 0 \) such that, \( \arg\max \Pi_w Q = \arg\max Q \) for any \( Q \).

Theorem 2 shows that Optimistic Weighting also recovers a \( Q_{tot} \) with the correct argmax.

4.2 Weighted QMIX Operators

We have shown that these two weightings are guaranteed to recover the correct maximum joint action for any \( Q \), and therefore for \( Q^* \) as well. This is in contrast to the uniform weighting (\( w = 1 \)) of
QMIX, which can fail to recover the correct optimal joint action even for simple matrix games. The weighted projection now allows us to fully take advantage of $Q^*$. Since we do not have access to the true optimal value function in general, we learn an approximation to it: $\hat{Q}^*$, which does not need to lie in the restricted monotonic function space $Q_{mix}$. Performing an exact maximisation of $\hat{Q}^*$ requires a search over the entire joint action space, which is typically intractable and does not admit decentralisation. We instead use our QMIX approximation $Q_{tot}$ to suggest the maximum joint action(s), which can then be evaluated by $\hat{Q}^*$.

Learning $\hat{Q}^*$ instead of using $Q_{tot}$ in its place brings some advantages. First, it allows us a richer representational class to approximate $Q^*$ with, since we place no restrictions on the form of $\hat{Q}^*$. In the idealised tabular setting, $Q^*$ is exactly representable by $\hat{Q}^*$. Second, since we are weighting each joint action in $\Pi_w$, $Q_{tot}$ (unlike $\hat{Q}^*$) likely has less accurate estimates for those joint actions with a low weighting. Due to these factors, we may bootstrap using more accurate estimates by using $\hat{Q}^*$ instead of $Q_{tot}$. These properties are necessary to ensure that WQMIX converges to the optimal policy. The operator used to update $\hat{Q}^*$ is:

$$T_w^* \hat{Q}^*(s, u) := \mathbb{E}[r + \gamma \hat{Q}^*(s', \text{argmax}_u Q_{tot}(s', u'))]. \quad (6)$$

Since $Q_{tot}$ is monotonic, the argmax in (6) is tractable. Similarly $Q_{tot}$ is updated in tandem using:

$$T_w^{WQMIX} Q_{tot} := \Pi_w T_w^* \hat{Q}^* \quad (7)$$

$T_w^*$ is similar to the Bellman Optimality Operator in (1) but does not directly maximise over $\hat{Q}^*$. Instead it uses $Q_{tot} \in Q_{mix}$ to suggest a maximum joint action. Setting $w$ to be uniform ($w = 1$) here does not recover QMIX since we are additionally learning $\hat{Q}^*$.

Finally, using our previous results, we show that $\hat{Q}^*$ converges to $Q^*$ and that $Q_{tot}$ recovers an optimal policy. This provides a firm theoretical foundation for Weighted QMIX: in an idealised setting it converges to the optimal policy, whereas QMIX does not.

**Corollary 1.** Letting $w$ be the Idealised Central or Optimistic Weighting, then $\exists \alpha > 0$ such that the unique fixed point of $T_w^*$ is $Q^*$. Furthermore, $\Pi_w Q^* \subseteq Q_{mix}$ converges to an optimal policy, and

$$\max \Pi_w Q^*(s, \cdot) = \max Q^*(s, \cdot).$$

In this section we have shown that an idealised version of Weighted QMIX can converge to $Q^*$ and recover an optimal policy. Restricting $Q_{tot}$ to lie in $Q_{mix}$ does not prevent us from representing an optimal policy, since there is always an optimal deterministic policy [24] and all deterministic policies can be derived from the argmax of a $Q$ that lies in $Q_{mix}$. Thus, we do not expand the function class that we consider for $Q_{tot}$. Instead, we change the solution of the projection by introducing a weighting.

## 5 Deep RL Algorithm

So far, we have only considered an idealised setting in order to analyse the fundamental properties of QMIX and Weighted QMIX. However, the ultimate goal of our analysis is to inform the development of new scalable RL algorithms, in combination with, e.g., neural network function approximators. We now describe the realisation of Weighted QMIX for deep RL, in a Dec-POMDP setting in which each agent does not observe the full state, as described in Section 2.

There are three components to Weighted QMIX: 1) $Q_{tot}$, i.e., the per-agent utilities $Q_a$ (from which the decentralised policies are derived) and the mixing network, 2) an unrestricted joint action $\hat{Q}^*$, and 3) a weighting function $w$, as in $\Pi_w$.

**$Q_{tot}$** The $Q_{tot}$ component is largely the same as that of Rashid et al. [25], using the architecture from Samvelyan et al. [26]. $Q_{tot}$ is trained to minimise the following loss:

$$\sum_{i=1}^{b} w(s, u)(Q_{tot}(\tau, u, s) - y_i)^2, \quad (8)$$

where $y_i := r + \gamma \hat{Q}^*(s', \tau', \text{argmax}_u \hat{Q}_{tot}(s', \tau', u'))$ is treated as a fixed target. This differs from the idealised setting considered in Sections 3 and 4 because we are now only optimising the $Q$-values...
for the state-action pairs present in the minibatch sampled from the replay buffer, as opposed to every state-action pair.

**Centralised \( \hat{Q}^* \)** We use a similar architecture to \( Q_{tot} \) to represent \( \hat{Q}^* \): agent networks whose chosen action’s utility is fed into a mixing network. \( \hat{Q}^* \) is thus conditioned on the state \( s \) and the agents’ action-observation histories \( \tau \). For the agent networks we use the same architecture as QMIX. They do not share parameters with the agents used in representing \( Q_{tot} \). The mixing network for \( \hat{Q}^* \) is a feed-forward network that takes the state and the appropriate actions’ utilities as input. This mixing network is not constrained to be monotonic by using non-negative weights. Consequently, we can simplify the architecture by having the state and agent utilities be inputs to \( \hat{Q}^* \)’s mixing network, as opposed to having hypernetworks take the state as input and generate the weights. The architecture of \( \hat{Q}^* \) is shown in Figure 1 (Left). \( \hat{Q}^* \) is trained to minimise the following loss, using \( y_i \) from (8):

\[
\sum_{i=1}^{b} (\hat{Q}^*(s, \tau, \hat{u}) - y_i)^2.
\]

\[
(9)
\]

**Weighting Function** Idealised Central Weighting requires knowing the maximal joint action over \( \hat{Q}^* \), which is computationally infeasible. In order to derive a practical algorithm, we must make approximations. For each state-action pair in the sampled minibatch, the weighting we use is:

\[
w(s, u) = \begin{cases} 1 & y_i > \hat{Q}^*(s, \tau, \hat{u}^*) \text{ or } u = \hat{u}^* \\ \alpha & \text{otherwise}, \end{cases}
\]

where \( \hat{u}^* = \arg\max_u Q_{tot}(\tau, u, s) \). Since we do not know the maximal joint action for each state, we make a local approximation: if \( y_i > \hat{Q}^*(s, \tau, \hat{u}^*) \), then \( u \) might be the best joint action. We use \( \hat{Q}^*(s, \tau, \hat{u}^*) \) instead of \( T_w \hat{Q}^*(s, \tau, \hat{u}^*) \) since we do not have direct access to it. We refer to this weighting function as Centrally-Weighted QMIX (CW).

The Optimistic Weighting presented in (5) does not require any approximations. The exact weighting we use is:

\[
w(s, u) = \begin{cases} 1 & Q_{tot}(\tau, u, s) < y_i \\ \alpha & \text{otherwise}. \end{cases}
\]

We refer to it as Optimistically-Weighted QMIX (OW).

In a deep RL setting, QMIX implicitly weights the joint-actions proportional to their execution by the behaviour policies used to fill the replay buffer. This forces QMIX to make trade-offs in its \( Q \)-value approximation that are directly tied to the exploration strategy chosen. However, as we have shown earlier, this can lead to poor estimates for the optimal joint action and thus yield suboptimal policies. Instead, Weighted QMIX separates the weighting of the joint actions from the behaviour policy. This allows us to focus our monotonic approximation of \( Q^* \) on the important joint actions, thus encouraging better policies to be recovered irrespective of the exploration performed.
6 Results

In this section we present our experimental results on the Predator Prey task considered by Bohmer et al. [2] and on a variety of SMAC scenarios. More details about the implementation of each are included in Appendix C, as well as additional ablation experiments (Appendix E). For every graph we plot the median and shade the 25%-75% quartile. Code is available at https://github.com/oxwhirl/wqmix.

6.1 Predator Prey

We first consider a partially-observable Predator Prey task involving 8 agents [2], which was designed to test coordination between agents by providing a punishment of -2 reward when only a single agent (as opposed to two agents) attempt to capture a prey. Algorithms which suffer from relative overgeneralisation [23, 35], or which make poor trade-offs in their representation (as VDN and QMIX do) can fail to solve this task.

As shown in [2], QMIX fails to learn a policy that achieves positive test reward, and our results additionally show the same negative results for MADDPG and MASAC. Interestingly, QPLEX also fails to solve the task despite not having any restrictions on the joint-action $Q$-values it can represent, suggesting difficulties in learning certain value functions. Figure 2 shows that both CW-QMIX and OW-QMIX solve the task faster than QTRAN.

6.2 SMAC

6.2.1 Robustness to increased exploration

QMIX is particularly brittle when there is significant exploration being done, since it then tries (and fails) to represent the value of many suboptimal joint actions. We would like our algorithms to efficiently learn from exploratory behaviour. Hence, we use two of the easier SMAC maps, 3s5z and 5m_vs_6m, to test the robustness of our algorithms to exploration. We use an $\epsilon$-greedy policy in which $\epsilon$ is annealed from 1 to 0.05 over 1 million timesteps, increased from the 50k used in [26]. Figure 3 shows the results of these experiments, in which both Weighted QMIX variants significantly outperform all baselines. Figure 4 (Left) additionally compares the performance of QMIX and Weighted QMIX on bane_vs_bane, a task with 24 agents, across two $\epsilon$-schedules. We can see that both variants of Weighted QMIX are able to solve the task irrespective of the level of exploration, whereas QMIX fails to do so.

6.2.2 Necessity of increased exploration

Next, we compare our method on the challenging 6h_vs_8z, which is classified as a super hard SMAC map due to current method’s poor performance [26]. Figure 4 (Right) compares QMIX and Weighted

---

2We utilise SC2.4.6.2.69232 (the same version as [26]) instead of the newer SC2.4.10. Performance is not comparable across versions.
QMIX with two differing exploration schedules (annealing $\epsilon$ over 50k or 1 million timesteps, denoted -50k $\epsilon$ and -1mil $\epsilon$ respectively). We can see that a larger rate of exploration is required, and only Weighted QMIX can successfully recover a good policy, demonstrating the benefits of our method for improving performance in a challenging coordination problem.

6.2.3 Limitations

Finally, we present results on 2 more super-hard SMAC maps in order to show the limitations of our method in Figure 5. In 3s5z_vs_3s6z we observe that the extra exploration is not helpful for any method. Since QMIX is almost able to solve the task, this indicates that both exploration and the restricted function class of QMIX are not limiting factors in this scenario. On corridor we see that only QMIX with an extended exploration schedule manages non-zero performance, showing the importance of sufficient exploration on this map. The poor performance of Weighted QMIX shows that the extra complexity of our method (notably learning $\hat{Q}^*$) can sometimes harm performance, indicating that closer attention must be paid to the architecture and weighting functions.

Figure 6 shows the results on corridor for Weighted QMIX with a slightly modified architecture for $\hat{Q}^*$. The significantly improved performance for Weighted QMIX indicates that the architecture of $\hat{Q}^*$ is partly responsible for the regression in performance over QMIX.

7 Conclusions and Future Work

This paper presented Weighted QMIX, which was inspired by analysing an idealised version of QMIX that first computes the $Q$-learning targets and then projects them into $Q_{mix}$. QMIX uses an unweighted projection that places the same emphasis on every joint action, which can lead to suboptimal policies. Weighted QMIX rectifies this by using a weighted projection that allows more emphasis to be placed on better joint actions. We formally proved that for two specific weightings, the weighted projection is guaranteed to recover the correct maximal joint action for any $Q$. To fully take advantage of this, we additionally learn an unrestricted joint action $\hat{Q}^*$, and prove that it converges to $Q^*$. We extended Weighted QMIX to deep RL and showed its improved ability to coordinate and its robustness to an increased rate of exploration. For future work, more complicated weightings could be considered, as opposed to the simplistic weightings we used where $w$ is either 1 or $\alpha$ in this paper. Additionally, some of our results demonstrate the limitations of our method, partially stemming from the architecture used for $\hat{Q}^*$. 
Broader Impact

Due to the broad applicability of cooperative Multi-Agent Reinforcement Learning, we limit our discussion to the cooperative setting in which agents must act independently without communication. One such potential application is in self-driving cars, in which the agents should be able to make safe and sensible decisions even without access to a communication network. Due to the sample inefficiency of current RL methods, combined with their lack of safe exploration it is also necessary to first train agents in a simulated setting before even beginning to train in the real world. Hence, the class of algorithms we consider in this paper could be used to train agents in these scenarios and would likely be chosen over fully-decentralised options. It is important then to obtain a better understanding of the current approaches, in particular of their limitations. In this paper, we focus primarily on the limitations of QMIX due to its strong performance [26]. We also investigate the links between QTRAN and our algorithm and observe poor empirical performance for Actor-Critic style approaches. Investigating all of these further should improve the performance of all algorithms in this domain, and provide a better understanding of their relative strengths and weaknesses.

One particular limitation of QMIX is that it can fail in environments in which an agent’s best action is dependent on the actions the other agents take, i.e., in environments in which agents must coordinate at the same timestep. However, in a Multi-Agent setting it is often crucial to coordinate with the other agents. Our approach lifts this restriction, and is theoretically able to learn the optimal policy in any environment which greatly increases its applicability. Extending the capabilities of cooperative MARL algorithms should further extend the applicability of these algorithms in a broader range of applications. However, our approach introduces extra complexity and can perform poorly in certain challenging domains. It is important then to consider whether the extra modelling capacity of our method is required to achieve good performance on a selected task.

Acknowledgments and Disclosure of Funding

We thank the members of the Whiteson Research Lab for their helpful feedback. This project has received funding from the European Research Council (ERC), under the European Union’s Horizon 2020 research and innovation programme (grant agreement number 637713). It was also supported by an EPSRC grant (EP/M508111/1, EP/N509711/1) and the UK EPSRC CDT in Autonomous Intelligent Machines and Systems. The experiments were made possible by a generous equipment grant from NVIDIA. Shimon Whiteson is the Head of Research at Waymo, UK.

References


[38] Xinghu Yao, Chao Wen, Yuhui Wang, and Xiaoyang Tan. SMI(X$(\lambda)$): Enhancing centralized value functions for cooperative multi-agent reinforcement learning. 2019.
A Related Work

In this section we briefly describe related work on cooperative MARL in the common paradigm of Centralised Training and Decentralised Execution (CTDE). For a more comprehensive survey of cooperative MARL, see [22].

Tampuu et al. [31] train each agent’s DQN [19] using Independent Q-learning [32], which treats the other agents as part of the environment, which can lead to many pathologies and instability during training [4, 8]. By contrast VDN [29] and QMIX [25] learn the joint-action Q-values, which avoids some of these issues. Qatten [37] change the architecture of QMIX’s mixing network to a 2-layer linear mixing, in which the weights of the first layer are produced through an attention-based mechanism. Bohmer et al. [3] learn a centralised joint action Q-Value that is approximately maximised by coordinate descent and used to generate trajectories that IQL agents train on. Optimistic Weighting uses a smaller weighting when decreasing Q-learning estimates, similarly to Hysteretic Q-learning [17] which uses a smaller learning rate when decreasing value estimates of independent learners.

Relationship to Actor-Critic. Weighted QMIX bears many similarities to an off-policy actor-critic algorithm, if we view $\hat{Q}^*$ as the critic and the policy implied by $Q_{tot}$ as the actor. Define the deterministic QMIX greedy policy (assuming full observability to simplify the presentation) as:

$$\pi_w(s) = \left( \underset{u_1}{\text{argmax}} \ Q_1(s, u_1) \right) \ldots \left( \underset{u_n}{\text{argmax}} \ Q_n(s, u_n) \right).$$

Weighted QMIX trains $\hat{Q}^*$ to approximate $Q^w$, the Q-values of this policy. This is also an approximation to Q-learning since $\pi_w \approx \text{argmax} \ Q^*$. Viewed in this manner, Weighted QMIX is similar to MADDPG [15] with a single critic, except for how the actors are trained. MADDPG trains each agent’s policy $\pi_a$ via the multi-agent deterministic policy gradient theorem, whereas Weighted QMIX trains the policy indirectly by training $Q_{tot}$ via the weighted loss in [3]. Multi-agent Soft Actor Critic (MASAC) [11, 12] is another off-policy actor-critic based approach that instead trains the actors by minimising the KL divergence between each agent’s policies and the joint-action Q-values. These actor-critic based approaches (as well as COMA [9] and LIIR [5]) do not restrict the class of joint action Q-values they can represent, which theoretically allows them to learn an optimal value function and policy. However, in practice they do not perform as well as QMIX, perhaps due to relative overgeneralisation [35] or the presence of bad local minima.

Relationship To QTRAN. QTRAN [27] is another Q-learning based algorithm that learns an unrestricted joint action Q function and aims to solve a constrained optimisation problem in order to decentralise it. However, it is empirically hard to scale to more complex tasks (such as SMAC).

We can view QTRAN as specific choices of the 3 components of Weighted QMIX, which allows us to better understand its trade-offs and empirical performance in relation to WQMIX. However, the motivations for QTRAN are significantly different. $Q_{tot}$ is represented using VDN instead of QMIX, and trained using $Q^*$ as the target (instead of $Q_{tot}$). This can limit QTRAN’s empirical performance because QMIX generally outperforms VDN [26]. $Q^*$ is a network that takes as input an embedding of all agents’ chosen actions and observations (and additionally the state if it is available). The agent components share parameters with the agent networks used to approximate $Q_{tot}$. 
Weighting. The weighting function is as follows:

\[
    w(s, u) = \begin{cases} 
    \lambda_{opt} & u = \hat{u} \\
    \lambda_{opt} & Q_{tot}(s, u) < Q^*(s, u) \\
    0 & \text{otherwise,}
    \end{cases}
\]  

(11)

where \( \hat{u} = \arg\max Q_{tot}(s, \cdot) \). Using too small a weight in the weighting can have a substantial negative effect on performance (as we show in Appendix E). However, using a 0 weight for overestimated \( Q \)-values is a fundamental part of the QTRAN algorithm.

Concurrent Work. Wang et al. [33] propose QPLEX, which also expands the class of joint-action \( Q \)-values that can be represented. They achieve this by decomposing \( Q_{tot} \) as a sum of a value function and a non-negative advantage function. Crucially, this advantage function is 0 for the joint-action in which every agent maximises their own utilities (\( \hat{u}^* \) in our notation). This ensures consistency between the agent’s greedy joint-action (\( \hat{u}^* \)), and the true maximum joint-action of \( Q_{tot} \). In contrast, Weighted QMIX does not maintain this consistency. Our experimental results show that despite not restricting the class of joint-action \( Q \)-values that can be represented, QPLEX can struggle to learn a good policy in some environments like QMIX.

Son et al. [28] propose QOPT, which also learns an unrestricted \( \hat{Q}^* \) and utilises an optimistic-style weighting to train \( Q_{tot} \) that is represented by QMIX. In contrast to Weighted QMIX, whose \( \hat{Q}^* \) does not share any parameters with \( Q_{tot} \), the unrestricted joint-action \( Q \)-values of QOPT are obtained through an unrestricted mixing network (the weights are not constrained to be non-negative) which takes as input the agent utilities. The weighting function used for training \( Q_{tot} \) is similar to the weighting for CW-QMIX, in which a smaller weighting is used for the the joint-actions whose \( Q \)-values are estimated as lower than the \( Q \)-values for the approximate best joint-action (\( \hat{u}^* \)).

B Proof of Theorems

Proposition 1. For any \( w : S \times U \rightarrow (0, 1] \) and \( Q \). Let \( Q_{tot} = \Pi Q \). Then \( \forall s \in S, \hat{u} \in \arg\max Q_{tot}(s, \cdot) \). We have that \( Q_{tot}(s, \hat{u}) \geq Q(s, \hat{u}) \). If \( \hat{u} = u^* := \arg\max_u Q(s, u) \) then \( Q_{tot}(s, \hat{u}) = Q(s, u^*) \).

Proof. Consider a \( s \in S \). Assume for a contradiction that \( Q_{tot}(s, \hat{u}) < Q(s, \hat{u}) \).

Define \( Q_{tot}' \) as follows:

\[
    Q_{tot}'(s, u) = \begin{cases} 
    Q(s, u) & u = \hat{u} \\
    Q_{tot}(s, u) & \text{otherwise,}
    \end{cases}
\]

By construction we have that \( Q_{tot}' \in Q^{mix} \), and

\[
    \sum_{u \in U} w(s, u)(Q(s, u) - Q_{tot}'(s, u))^2 \\
    = \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q_{tot}'(s, u))^2 + w(s, \hat{u})(Q(s, \hat{u}) - Q_{tot}'(s, \hat{u}))^2 \\
    = \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q_{tot}'(s, u))^2 \\
    + (Q_{tot}'(s, \hat{u}) = Q(s, \hat{u})) \\
    + (Q_{tot}'(s, u) = Q_{tot}(s, u) \forall u \neq \hat{u}) \\
    < \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2 + w(s, \hat{u})(Q(s, \hat{u}) - Q_{tot}(s, \hat{u}))^2 \\
    + (Q_{tot}(s, \hat{u}) < Q(s, \hat{u})) \\
    = \sum_{u \in U} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2.
\]
Thus $Q_{tot}$ cannot be the solution of $\Pi_w Q$, a contradiction. And so $Q_{tot}(s, \hat{u}) \geq Q(s, \hat{u})$.

Now consider the scenario in which $\hat{u} = u^*$, and assume for a contradiction that $Q_{tot}(s, \hat{u}) > Q(s, \hat{u})$.

Define $Q'_{tot}$ as follows:

$$Q'_{tot}(s, u) = \begin{cases} Q(s, u) & u = \hat{u} = u^* \\ \min\{Q_{tot}(s, u), Q(s, u^*)\} & \text{otherwise,} \end{cases}$$

Again, by construction $Q'_{tot} \in Q^{mix}$.

$$\sum_{u \in U} w(s, u)(Q(s, u) - Q'_{tot}(s, u))^2$$

$$= \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q'_{tot}(s, u))^2 + w(s, \hat{u})(Q(s, \hat{u}) - Q'_{tot}(s, \hat{u}))^2$$

$$= \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q'_{tot}(s, u))^2$$

$$\leq \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2$$

If $\min\{Q_{tot}(s, u), Q(s, \hat{u})\} = Q_{tot}(s, u)$ then $Q'_{tot}(s, u) = Q_{tot}(s, u)$. Otherwise $Q(s, u^*) < Q_{tot}(s, u) \implies (Q(s, u^*) - Q(s, u))^2 < (Q_{tot}(s, u) - Q(s, u))^2$.

Since $Q(s, u) \leq Q(s, u^*)$.

$$< \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2 + w(s, \hat{u})(Q(s, \hat{u}) - Q_{tot}(s, \hat{u}))^2$$

$$= \sum_{u \neq \hat{u}} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2.$$ 

Thus, $Q_{tot}$ cannot be the solution of $\Pi_w Q$, a contradiction. This proves that $Q_{tot}(s, \hat{u}) = Q(s, \hat{u})$ if $\hat{u} = u^*$.

**Proposition 2.** Let $Q_{tot} = \Pi_w Q$. $\forall s \in S \exists \hat{u} \in \text{argmax } Q_{tot}(s, \cdot)$ such that $Q_{tot}(s, \hat{u}) = Q(s, \hat{u})$.

**Proof.** Assume for a contradiction that $\forall \hat{u} \in \text{argmax } Q_{tot}$ we have that $Q_{tot}(\hat{u}) > Q(\hat{u})$.

Define $\Delta_s := Q_{tot}(s, \hat{u}) - \max\{Q_{tot}(s, u) | u \in U, Q_{tot}(s, u) < Q_{tot}(s, \hat{u})\}$ to be the difference between the maximum Q-Value and the next biggest Q-Value (the action gap $\Pi$). $\Delta_s$ is well defined as long as there exists a sub-optimal action. If there is not a suboptimal action, then trivially any $u \in U$ satisfies the condition.

Let $\epsilon = \min\{\Delta_s/2, (Q_{tot}(s, \hat{u}) - \max\{Q(s, u) | u \in \text{argmax } Q_{tot}(s, \cdot)\})/2\} > 0$.

Define $Q'_{tot}$ as follows:

$$Q'_{tot}(s, u) = \begin{cases} Q_{tot}(s, u) - \epsilon & u \in \text{argmax } Q_{tot} \\ Q_{tot}(s, u) & \text{otherwise.} \end{cases}$$

i.e. we have decreased the Q-Value estimates for the argmax joint actions by a small non-zero amount. Since $\epsilon < \Delta_s$, we do not need to worry about adjusting other action’s estimates.

By construction $Q'_{tot} \in Q^{mix}$.

Then $Q'_{tot}$ has a smaller loss than $Q_{tot}$ since the estimates for the argmax actions are closer to the true values.

This gives our contradiction since $Q_{tot} \in \Pi_w Q$. 

15
Thus \( \exists \hat{u} \in \arg\max Q_{tot} \) such that \( Q_{tot}(\hat{u}) \leq Q(\hat{u}) \). Combined with Proposition 1 gives us the required result. 

**Corollary 2.** If \( Q_{tot} \) has a unique \( \arg\max \hat{u} \), then \( Q_{tot}(\hat{u}) = Q(\hat{u}) \).

**Proof.** Proposition 2 showed the existence of an \( \arg\max \) action whose \( Q_{tot} \)-value matches \( Q \) exactly. If there is a unique \( \arg\max \), then it must match exactly giving our result. 

**Theorem 1.** Let \( w \) be the Idealised Central Weighting from Equation 4. Then \( \exists \alpha > 0 \) such that, \( \arg\max \Pi_w Q = \arg\max Q \) for any \( Q \).

**Theorem 2.** Let \( w \) be the Optimistic Weighting from Equation 5. Then \( \exists \alpha > 0 \) such that, \( \arg\max \Pi_w Q = \arg\max Q \) for any \( Q \).

**Proof.** Since the proof of both theorems contains a significant overlap, we will merge them both into a single proof.

We will start by first considering the Idealised Central Weighting: Let \( Q_{tot} = \Pi_w Q \) (\( Q_{tot} \in \Pi_w Q \) if there are distinct solutions).

Let \( u^* \in \arg\max Q \), be an optimal action.

Consider a state \( s \in S \).

Define \( \Delta_s := Q(s, u^*) - \max\{Q(s, u) | u \in U, Q(s, u) < Q(s, u^*)\} \) to be the difference between the maximum \( Q \)-Value and the next biggest \( Q \)-Value (the action gap \( \Pi \)). \( \Delta_s \) is well defined as long as there exists a sub-optimal action. If there is not a suboptimal action, then trivially \( \arg\max \Pi_w Q = \arg\max Q \) for state \( s \).

Let \( \hat{u} \in \arg\max \Pi_w Q \), and consider the loss when \( \hat{u} = u^* \).

By Proposition 1 we have that \( Q_{tot}(s, \hat{u}) = Q(s, \hat{u}) \).

Then the loss:

\[
\sum_{u \in U} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2 = \alpha \sum_{u \neq u^*} (Q(s, u) - Q_{tot}(s, u))^2 < \alpha(\frac{R_{max}}{1 - \gamma})^2 |U|^n,
\]

where \( R_{max} := \max r - \min r \). The last inequality follows since the maximum difference between \( Q \)-values in the discounted setting is then \( \frac{R_{max}}{1 - \gamma} \), and there are \(|U|^n\) joint-actions total.

Whereas if \( \hat{u} \neq u^* \), then the loss

\[
\sum_{u \in U} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2
= (Q(s, u^*) - Q_{tot}(s, u^*))^2 + \alpha \sum_{u \neq u^*} (Q(s, u) - Q_{tot}(s, u))^2
\geq \Delta_s^2,
\]

since \( Q(s, u^*) - Q_{tot}(s, u^*) \geq \Delta_s \), which is proved below.

By Proposition 2 let \( \hat{u}' \in \arg\max Q_{tot} \) such that \( Q_{tot}(s, \hat{u}') = Q(s, \hat{u}') \). Then \( Q(s, u^*) \geq \Delta_s + Q(s, \hat{u}') = \Delta_s + Q_{tot}(s, \hat{u}') > \Delta_s + Q_{tot}(s, u^*) \Rightarrow Q(s, u^*) - Q_{tot}(s, u^*) > \Delta_s \).

The strict inequality \( Q_{tot}(s, \hat{u}') > Q_{tot}(s, u^*) \) used is due to \( \hat{u} \neq u^* \).

Setting \( 0 < \alpha_s < \frac{\Delta_s^2 (1 - \gamma)}{(R_{max})^2 |U|^n} \) then gives the required result for state \( s \).

Letting \( \alpha = \min_s \alpha_s > 0 \) completes the proof for the Idealised Central Weighting.

For the proof of the Optimistic Weighting we will use many of the same arguments and notation.

We will once again consider a single state \( s \in S \) and the action gap \( \Delta_s \).

Now, let us consider a \( Q_{tot} \) of a specific form: For \( s \in S \), let \( Q_{tot}(s, \hat{u}) = c_s + \epsilon \), where \( \epsilon << \Delta_s \) and \( Q_{tot}(s, u) = c_s \), \( \forall u \neq \hat{u} \). Note that here \( Q_{tot} \) has a unique maximum action.
For this $Q_{tot}$, consider $\hat{u} = u^*$ then the loss:
\[
\sum_{u \in U} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2 = \alpha \sum_{u \neq u^*} (Q(s, u) - Q_{tot}(s, u))^2 \leq \alpha \frac{R_{\max}}{1 - \gamma} |U|^n < \Delta_s^2,
\]
since $Q_{tot}(s, u) = c_s = Q(s, u^*) - \epsilon > Q(s, u), \forall u \neq u^*$ by Proposition 1 which means that $w(s, u \neq u^*) = \alpha$. The final inequality follows due to setting $0 < \alpha_s < \frac{\Delta_s^2 (1-\gamma)^2}{(R_{\max})^2 |U|^n}$ as earlier.

Now consider any $Q'_{tot} \in Q^{mix}$.

If for this $Q'_{tot}$, $\hat{u} \neq u^*$, then $w(s, u^*) = 1$ and thus the loss
\[
\sum_{u \in U} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2
= (Q(s, u^*) - Q_{tot}(s, u^*))^2 + \sum_{u \neq u^*} w(s, u)(Q(s, u) - Q_{tot}(s, u))^2
\geq \Delta_s^2.
\]

By Proposition 2 let $\hat{u}' \in \text{argmax} Q'_{tot}$ such that $Q'_{tot}(s, \hat{u}') = Q(s, \hat{u}')$. Since $Q(s, u^*) \geq \Delta_s + Q(s, \hat{u}') - Q_{tot}(s, \hat{u}') \geq \Delta_s + Q_{tot}(s, u^*) \implies Q(s, u^*) - Q_{tot}(s, u^*) > \Delta_s$.

Thus, we have shown that for any $Q'_{tot}$ with $\hat{u} \neq u^*$ the loss is greater than the $Q_{tot}$ we first considered with $\hat{u} = u^*$.

And so for state $s$, $\text{argmax} \Pi_w Q(s, \cdot) = u^* = \text{argmax} Q(s, \cdot)$.

Letting $\alpha = \min_s \alpha_s > 0$ once again completes the proof. \hfill \qed

**Corollary 3.** Letting $w$ be the Central or Optimistic Weighting, then $\exists \alpha > 0$ such that the unique fixed point of $T^*_w$ is $Q^*$. Furthermore, $\Pi_w Q^* \subseteq Q^{mix}$ recovers an optimal policy, and max $\Pi_w Q^* = \max Q^*$.

**Proof.** Using the results of Theorems 1 and 2 we know that $\exists \alpha > 0$ such that $\text{argmax} \Pi_w Q = \text{argmax} Q$. We also know from their proofs that the same $\alpha$ works for both weightings.

Instead of updating $Q_{tot}$ in tandem with $\hat{Q}^*$, we can instead write $T^*_w$ as:
\[
T^*_w \hat{Q}^*(s, u) := \mathbb{E}[r + \gamma \hat{Q}^*(s', \text{argmax} \Pi_w \hat{Q}^*)(s', u')].
\]

And so:
\[
\hat{Q}^*(s', \text{argmax} \Pi_w \hat{Q}^*)(s', u') = \max_{u'} \hat{Q}^*(s', u'), \forall s' \in S.
\]

Thus, our operator $T^*_w$ is equivalent to the usual Bellman Optimality Operator $T^*$, which is known to have a unique fixed point $Q^*$ \cite{bellman53} \cite{puterman94}.

Once again by the results of Theorems 1 and 2 we know that $Q^*_{tot} \in \Pi_w Q^*$ achieves the correct argmax for every state. Thus it is an optimal policy. Finally, Proposition 1 shows that $\max \Pi_w Q^* = \max Q^*$.

\hfill \qed

**C Experimental Setup**

We adopt the same training setup as \cite{jaderberg16}, using PyMARL to run all experiments. The architecture for QMIX is also the same as in \cite{jaderberg16}.

The architecture of the mixing network for $\hat{Q}^*$ is a feed forward network with 3 hidden layers of 256 dim and ReLU non-linearities. Shown in Figure 1.

For the experiment in Figure 6 the architecture for $\hat{Q}^*$ is modified slightly. We replace the first hidden layer with a hypernetwork layer. A hypernetwork with a single hidden layer of dim 64 (with
We consider $\alpha$-QMIX. This ablation allows us to see if a better bootstrap estimate alone can explain the performance of $\hat{Q}^\ast$. In order to better understand our method, we examine 3 additional baselines:

**QMIX + $\hat{Q}^\ast$.** This ablation removes the weighting in our loss ($w = 1$), but still uses $\hat{Q}^\ast$ to bootstrap. This ablation allows us to see if a better bootstrap estimate alone can explain the performance of WQMIX.

**C.1 Predator Prey**

For Weighted QMIX variants (and ablations with just a weighting), we consider $\alpha \in \{0.1, 0.5\}$ and set $\alpha = 0.1$ for all variants.

For QTRAN we set $\lambda_{\text{opt}} = 1$ and consider $\lambda_{\text{nopt}} \in \{0.1, 1, 10\}$ (since only their relative weighting makes a difference), and the dim of the mixing network in $\{64, 256\}$. We set $\lambda_{\text{nopt}} = 10$ and the dim of the mixing network to 64.

For MASAC we consider $\alpha_{\text{ent}} \in \{0, 0.001, 0.01, 0.1\}$ and set it to 0.001.

**C.2 SMAC Robustness to exploration**

For Weighted QMIX we consider $\alpha \in \{0.01, 0.1, 0.5, 0.75\}$ and set $\alpha = 0.75$ for CW and $\alpha = 0.5$ for OW. All lines are available in Appendix [E].

For the Weighted QMIX ablations we considered $\alpha \in \{0.5, 0.75\}$ and set $\alpha = 0.75$.

For QTRAN we set $\lambda_{\text{opt}} = 1$ and consider $\lambda_{\text{nopt}} \in \{1, 10\}$ (since these 2 performed best in preliminary experiments), and the dim of the mixing network in $\{64, 256\}$. We set $\lambda_{\text{nopt}} = 10$ for 5m_vs_6m and $\lambda_{\text{opt}} = 1$ for 3s5z. The dim of the mixing network is set to 64.

For MASAC we consider $\alpha_{\text{ent}} \in \{0, 0.001, 0.01\}$ and set it to 0 for 3s5z and 0.01 for 5m_vs_6m.

**C.3 SMAC Super Hard Maps**

We consider $\alpha \in \{0.01, 0.1, 0.5, 0.75\}$ and set $\alpha = 0.5$ for OW-QMIX and $\alpha = 0.75$ for CW-QMIX. For the experiment in Figure [G] we only considered $\alpha \in \{0.5, 0.75\}$ and set $\alpha = 0.75$ for both methods.

**D Ablations**

In order to better understand our method, we examine 3 additional baselines:

**MADDPG and MASAC’s critic shares the same architecture as $\hat{Q}^\ast$.**

**C.3 SMAC Super Hard Maps**

For MASAC we consider $\alpha$.

For MADDPG and MASAC’s critic shares the same architecture as $\hat{Q}^\ast$.

For Weighted QMIX variants (and ablations with just a weighting), we consider $\alpha$.

MADDPG is trained using the deterministic multi-agent policy-gradient theorem, via the Gumbel-Softmax trick, as in [15, 12]. Specifically for agent $a$ we produce $Q(s, u_a, u^{-a})$ (with target network $\hat{Q}$) for each possible action, where $u^{-a}$ are the actions of the other agents produced by their most recent policies. We multiply these by the agent’s policy (one-hot vector since it is deterministic) and use the Straight Through Gumbel-Softmax estimator [13] to differentiate through this with a temperature of 1.

For Weighted QMIX we consider $\alpha$.

MASAC is trained by minimising the KL divergence between each agent’s policy $\pi_a$ and $\exp(Q(s, \cdot, u^{-a}) - \alpha_{\text{ent}} \log \pi_a)$.. Since the KL divergence is an expectation: $\mathbb{E}_{\pi_a} \left[ \log \left( \frac{\exp(Q(s, u^{-a}) - \alpha_{\text{ent}} \log \pi_a)}{\pi_a} \right) \right]$, we approximate it by sampling an action from $\pi_a$ for each agent. These sampled actions are used for $u^{-a}$. For the actor’s policies we use the same $\epsilon$-greedy floor technique as in [9].

QPLEX uses the same setup for its mixing network as for the SMAC experiments in [33].

ReLU) takes the state as input and generates the weight matrix. Inspired by [37] we then take the column-wise softmax of this weight matrix, which can be viewed as an approximation of multi-head attention.

The architecture and setup for QTRAN is also the same, except we use a 3 layer feedforward network of dim $\{64, 256\}$ to match the depth of $\hat{Q}^\ast$. For MASAC we consider $\alpha$.

For QTRAN we set $\lambda_{\text{opt}} = 1$ and consider $\lambda_{\text{nopt}} \in \{0.1, 1, 10\}$ (since only their relative weighting makes a difference), and the dim of the mixing network in $\{64, 256\}$. We set $\lambda_{\text{nopt}} = 10$ and the dim of the mixing network to 64.

For Weighted QMIX ablations we considered $\alpha$. For OW. All lines are available in Appendix [E].

For QTRAN we set $\lambda_{\text{opt}} = 1$ and consider $\lambda_{\text{nopt}} \in \{1, 10\}$ (since these 2 performed best in preliminary experiments), and the dim of the mixing network in $\{64, 256\}$. We set $\lambda_{\text{nopt}} = 10$ for 5m_vs_6m and $\lambda_{\text{opt}} = 1$ for 3s5z. The dim of the mixing network is set to 64.

For MASAC we consider $\alpha_{\text{ent}} \in \{0, 0.001, 0.01\}$ and set it to 0.001.
Figure 7: Median test return for the predator prey environment, comparing Weighted QMIX and 3 ablations.

Figure 8: Median test win % with an increased rate of exploration, comparing Weighted QMIX and 3 ablations.

**QMiX + CW/OW.** This ablation introduces the CW/OW weightings into QMiX’s loss function for $Q_{tot}$. We do not additionally learn $\hat{Q}^*$.  

**E Results**  

In this section we present the results of additional experiments that did not fit in the main paper.  

**Ablation experiments for Predator Prey.**  

Our ablation experiments in Figure 7 show the necessity for both $\hat{Q}^*$ and a weighting in order to solve this task. As expected QMiX + $Q^*$ is unable to solve this task due to the challenges of relative overgeneralisation. The use of a uniform weighting in the projection prevents the learning of an optimal policy in which two agents coordinate to capture a prey. Thus, even if $\hat{Q}^*$ can theoretically represent the $Q$-values of this optimal policy, the QMiX agents are unable to recover it. Figure 7 also shows that QMiX with just a weighting in its projection (and no $\hat{Q}^*$) is unable to successfully solve the task.  

**Ablation experiments testing robustness to increased exploration.**  

Figure 8 shows the results of further ablation experiments, confirming the need for both $\hat{Q}^*$ and a weighting to ensure consistent performance. Note in particular that combining QMiX with a weighting results in significantly worse performance in $5m_{vs}6m$, and no better performance in $3s5z$.  

**Effect of $\alpha$ on performance.**  

Figure 9 shows the effect of varying $\alpha$ in the weighting function for CW-QMiX and OW-QMiX. We can see that if $\alpha$ is too low, performance degrades considerably.
Figure 9: Median test win % with an increased rate of exploration. **Above:** The effect of varying $\alpha$ for CW-QMIX. **Below:** The effect for OW-QMIX.