Value-Decomposition Networks For Cooperative Multi-Agent Learning

Peter Sunehag
DeepMind
sunehag@google.com

Guy Lever
DeepMind
guylever@google.com

Audrunas Gruslys
DeepMind
audrunas@google.com

Wojciech Marian Czarnecki
DeepMind
lejlot@google.com

Vinicius Zambaldi
DeepMind
vzambaldi@google.com

Max Jaderberg
DeepMind
jaderberg@google.com

Marc Lanctot
DeepMind
lanctot@google.com

Nicolas Sonnerat
DeepMind
sonnerat@google.com

Joel Z. Leibo
DeepMind
jzl@google.com

Karl Tuyls
DeepMind & University of Liverpool
karltuyls@google.com

Thore Graepel
DeepMind
thore@google.com

Abstract

We study the problem of cooperative multi-agent reinforcement learning with a single joint reward signal. This class of learning problems is difficult because of the often large combined action and observation spaces. In the fully centralized and decentralized approaches, we find the problem of spurious rewards and a phenomenon we call the "lazy agent" problem, which arises due to partial observability. We address these problems by training individual agents with a novel value decomposition network architecture, which learns to decompose the team value function into agent-wise value functions. We perform an experimental evaluation across a range of partially-observable multi-agent domains and show that learning such value-decompositions leads to superior results, in particular when combined with weight sharing, role information and information channels.

1 Introduction

We consider the cooperative multi-agent reinforcement learning (MARL) problem (Panait and Luke, 2005; Busoniu et al., 2008; Tuyls and Weiss, 2012), in which a system of several learning agents must jointly optimize a single reward signal – the *team reward* – accumulated over time. Each agent has access to its own ("local") observations and is responsible for choosing actions from its own action set. Coordinated MARL problems emerge in applications such as coordinating self-driving vehicles and/or traffic signals in a transportation system, or optimizing the productivity of a factory comprised of many interacting components. More generally, with AI agents becoming more pervasive, they will have to learn to coordinate to achieve common goals.

Although in practice some applications may require local autonomy, in principle the cooperative MARL problem could be treated using a *centralized* approach, reducing the problem to single-agent reinforcement learning (RL) over the concatenated observations and combinatorial action space. We show that the centralized approach consistently fails on relatively simple cooperative MARL

problems in practice. We present a simple experiment in which the centralised approach fails by learning inefficient policies with only one agent active and the other being "lazy". This happens when one agent learns a useful policy, but a second agent is discouraged from learning because its exploration would hinder the first agent and lead to worse team reward.[1]

An alternative approach is to train *independent learners* to optimize for the team reward. In general each agent is then faced with a non-stationary learning problem because the dynamics of its environment effectively changes as teammates change their behaviours through learning (Laurent et al., 2011). Furthermore, since from a single agent's perspective the environment is only partially observed, agents may receive spurious reward signals that originate from their teammates' (unobserved) behaviour. Because of this inability to explain its own observed rewards naive independent RL is often unsuccessful: for example Claus and Boutilier (1998) show that independent Q-learners cannot distinguish teammates' exploration from stochasticity in the environment, and fail to solve even an apparently trivial, 2-agent, stateless, 3×3-action problem and the general Dec-POMDP problem is known to be intractable (Bernstein et al., 2000; Oliehoek and Amato, 2016). Though we here focus on 2 player coordination, we note that the problems with individual learners and centralized approaches just gets worse with more agents since then, most rewards do not relate to the individual agent and the action space grows exponentially for the fully centralized approach.

One approach to improving the performance of independent learners is to design individual reward functions, more directly related to individual agent observations. However, even in the single-agent case, reward shaping is difficult and only a small class of shaped reward functions are guaranteed to preserve optimality w.r.t. the true objective (Ng et al., 1999; Devlin et al., 2014; Eck et al., 2016). In this paper we aim for more general autonomous solutions, in which the decomposition of the team value function is learned.

We introduce a novel **learned additive value-decomposition** approach over individual agents. Implicitly, the value decomposition network aims to learn an optimal linear value decomposition from the team reward signal, by back-propagating the total Q gradient through deep neural networks representing the individual component value functions. This additive value decomposition is specifically motivated by avoiding the spurious reward signals that emerge in purely independent learners.The implicit value function learned by each agent depends only on local observations, and so is more easily learned. Our solution also ameliorates the coordination problem of independent learning highlighted in Claus and Boutilier (1998) because it effectively learns in a centralised fashion at training time, while agents can be deployed individually.

Further, in the context of the introduced agent, we evaluate weight sharing, role information and information channels as additional enhancements that have recently been reported to improve sample complexity and memory requirements (Hausknecht, 2016; Foerster et al., 2016; Sukhbaatar et al., 2016). However, our main comparison is between three kinds of architecture; Value-Decomposition across individual agents, Independent Learners and Centralized approaches. We investigate and benchmark combinations of these techniques applied to a range of new interesting two-player coordination domains. We find that Value-Decomposition is a much better performing approach than centralization or fully independent learners, and that when combined with the additional techniques, results in an agent that consistently outperforms centralized and independent learners by a big margin.

1.1 Other Related Work

Schneider et al. (1999) consider the optimization of the sum of individual reward functions, by optimizing local compositions of individual value functions learnt from them. Russell and Zimdars (2003) sums the Q-functions of independent learning agents with individual rewards, before making the global action selection greedily to optimize for total reward. Our approach works with only a team reward, and *learns* the value-decomposition autonomously from experience, and it similarly differs from the approach with coordination graphs (Guestrin et al., 2002) and the max-plus algorithm (Kuyer et al., 2008; van der Pol and Oliehoek, 2016).

Other work addressing team rewards in cooperative settings is based on *difference rewards* (Tumer and Wolpert, 2004), measuring the impact of an agent's action on the full system reward. This reward

[1]For example, imagine training a 2-player soccer team using RL with the number of goals serving as the team reward signal. Suppose one player has become a better scorer than the other. When the worse player takes a shot the outcome is on average much worse, and the weaker player learns to avoid taking shots (Hausknecht, 2016).

has nice properties (e.g. high learnability), but can be impractical as it requires knowledge about the system state (Colby et al., 2016; Agogino and Tumer, 2008; Proper and Tumer, 2012). Other approaches can be found in Devlin et al. (2014); HolmesParker et al. (2016); Babes et al. (2008).

2 Background

2.1 Reinforcement Learning

We recall some key concepts of the RL setting (Sutton and Barto, 1998), an agent-environment framework (Russell and Norvig, 2010) in which an agent sequentially interacts with the environment over a sequence of timesteps, $t = 1, 2, 3, \ldots$, by executing actions and receiving observations and rewards, and aims to maximize cumulative reward. This is typically modelled as a Markov decision process (MDP) (e.g. Puterman, 1994) defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}_1, \mathcal{T}, R \rangle$ comprising the state space $\mathcal{S}$, action space $\mathcal{A}$, a (possibly stochastic) reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ start state distribution $\mathcal{T}_1 \in \mathcal{P}(\mathcal{S})$ and transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$, where $\mathcal{P}(\mathcal{X})$ denotes the set of probability distributions over the set $\mathcal{X}$. We use $\bar{R}$ to denote the expected value of R. The agent's interactions give rise to a trajectory $(S_1, A_1, R_1, S_2, ...)$ where $S_1 \sim \mathcal{T}_1$, $S_{t+1} \sim \mathcal{T}(\cdot|S_t, A_t)$ and $R_t = R(S_t, A_t, S_{t+1})$, and we denote random variables in upper-case, and their realizations in lower-case. At time t the agent observes $o_t \in \mathcal{O}$ which is typically some function of the state s_t, and when the state is not fully observed the system is called a partially observed Markov decision process (POMDP).

The agent's goal is to maximize expected cumulative discounted reward with a discount factor γ, $\mathcal{R}_t := \sum_{t=1}^{\infty} \gamma^{t-1} R_t$. The agent chooses actions according to a *policy*: a (stationary) policy is a function $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ from states to probability distributions over $\mathcal{A}$. An optimal policy is one which maximizes expected cumulative reward. In fully observed environments, stationary optimal policies exist. In partially observed environments, the policy usually incorporates past agent observations from the *history* $h_t = a_1 o_1 r_1, ..., a_{t-1} o_{t-1} r_{t-1}$ (replacing s_t). A practical approach utilized here, is to parameterize policies using recurrent neural networks.

$V^\pi(s) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} R(S_t, A_t, S_{t+1}) | S_1 = s; A_t \sim \pi(\cdot|S_t)]$ is the value function and the action-value function is $Q^\pi(s, a) := \mathbb{E}_{S' \sim \mathcal{T}(\cdot|s,a)}[R(S, a, S') + \gamma V(S')]$ (generally, we denote the successor state of s by s'). The optimal value function is defined by $V^*(s) = \sup_\pi V^\pi(s)$ and similarly $Q^*(s, a) = \sup_\pi Q^\pi(s, a)$. For a given action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ we define the (deterministic) greedy policy w.r.t. Q by $\pi(s) := \arg\max_{a \in \mathcal{A}} Q(s, a)$ (ties broken arbitrarily). The greedy policy w.r.t. Q^* is optimal (e.g. Szepesvári, 2010).

2.2 Deep Q-Learning

One method for obtaining Q^* is Q-learning which is based on the update $Q_{i+1}(s_t, a_t) = (1 - \eta_t) Q_i(s_t, a_t) + \eta_t (r_t + \gamma \max_a Q_i(s_{t+1}, a))$, where $\eta_t \in (0, 1)$ is the learning rate. We employ the ε-greedy approach to action selection based on a value function, which means that with $1 - \varepsilon$ probability we pick $\arg\max_a Q_i(s, a)$ and with probability ε a random action. Our study focuses on deep architectures for the value function similar to those used by Mnih et al. (2015), and our approach incorporates the key techniques of target networks and experience replay employed there, making the update into a stochastic gradient step. Since we consider partially observed environments our Q-functions are defined over agent observation histories, $Q(h_t, a_t)$, and we incorporate a recurrent network similarly to Hausknecht and Stone (2015). To speed up learning we add the dueling architecture of Wang et al. (2016) that represent Q using a value and an advantage function, including multi-step updates with a forward view eligibility trace (e.g. Harb and Precup, 2016) over a certain number of steps. When training agents the recurrent network is updated with truncated back-propagation through time (BPTT) for this amount of steps. Although we concentrate on DQN-based agent architectures, our techniques are also applicable to policy gradient methods such as A3C (Mnih et al., 2016).

2.3 Multi-Agent Reinforcement Learning

We consider problems where observations and actions are distributed across d agents, and are represented as d-dimensional tuples of primitive observations in $\mathcal{O}$ and actions in $\mathcal{A}$. As is standard

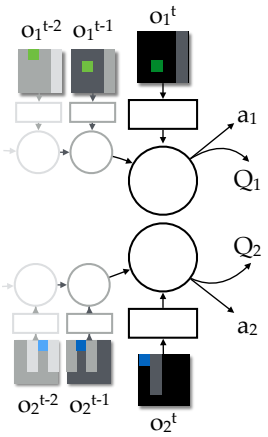

Figure 1: Independent agents architecture showing how local observations enter the networks of two agents over time (three steps shown), pass through the low-level linear layer to the recurrent layer, and then a dueling layer produces individual Q-values.

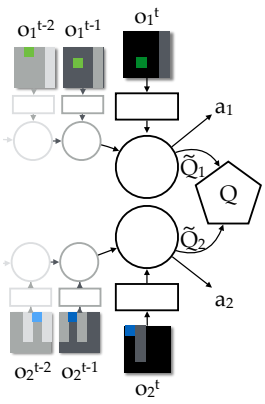

Figure 2: Value-decomposition individual architecture showing how local observations enter the networks of two agents over time (three steps shown), pass through the low-level linear layer to the recurrent layer, and then a dueling layer produces individual "values" that are summed to a joint Q-function for training, while actions are produced independently from the individual outputs.

in MARL, the underlying environment is modeled as a Markov game where actions are chosen and executed simultaneously, and new observations are perceived simultaneously as a result of a transition to a new state (Littman, 1994, 2001; Hu and Wellman, 2003; Busoniu et al., 2008).

Although agents have individual observations and are responsible for individual actions, each agent only receives the joint reward, and we seek to optimize $\mathcal{R}_t$ as defined above. This is consistent with the Dec-POMDP framework (Oliehoek et al., 2008; Oliehoek and Amato, 2016).

If we denote $\bar{h} := (h^1, h^2, ..., h^d)$ a tuple of agent histories, a joint policy is in general a map $\pi : \mathcal{H}^d \to \mathcal{P}(\mathcal{A}^d)$; we in particular consider policies where for any history $\bar{h}$, the distribution $\pi(\bar{h})$ has independent components in $\mathcal{P}(\mathcal{A})$. Hence, we write $\pi : \mathcal{H}^d \to \mathcal{P}(\mathcal{A})^d$. The exception is when we use the most naive centralized agent with a combinatorial action space, aka joint action learners.

3 A Deep-RL Architecture for Coop-MARL

Building on purely independent DQN-style agents (see Figure 1), we add enhancements to overcome the identified issues with the MARL problem. Our main contribution of value-decomposition is illustrated by the network in Figure 2.

The main assumption we make and exploit is that the joint action-value function for the system can be additively decomposed into value functions across agents,

$$Q((h^1, h^2, ..., h^d), (a^1, a^2, ..., a^d)) \approx \sum_{i=1}^{d} \tilde{Q}_i(h^i, a^i)$$

where the $\tilde{Q}_i$ depends only on each agent's local observations. We learn $\tilde{Q}_i$ by backpropagating gradients from the Q-learning rule using the joint reward through the summation, i.e. $\tilde{Q}_i$ is learned implicitly rather than from any reward specific to agent i, and we do not impose constraints that the $\tilde{Q}_i$ are action-value functions for any specific reward. The value decomposition layer can be seen in the top-layer of Figure 2. One property of this approach is that, although learning requires some centralization, the learned agents can be deployed independently, since each agent acting greedily with respect to its local value $\tilde{Q}_i$ is equivalent to a central arbiter choosing joint actions by maximizing the sum $\sum_{i=1}^{d} \tilde{Q}_i$.

For illustration of the idea consider the case with 2 agents (for simplicity of exposition) and where rewards decompose additively across agent observations[2], $r(\mathbf{s}, \mathbf{a}) = r_1(o^1, a^1) + r_2(o^2, a^2)$, where (o^1, a^1) and (o^2, a^2) are (observations, actions) of agents 1 and 2 respectively. This could be the case in team games for instance, when agents observe their own goals, but not necessarily those of teammates. In this case we have that

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r(\mathbf{s}_t, \mathbf{a}_t) | \mathbf{s}_1 = \mathbf{s}, \mathbf{a}_1 = \mathbf{a}; \pi]$$

$$= \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_1(o_t^1, a_t^1) | \mathbf{s}_1 = \mathbf{s}, \mathbf{a}_1 = \mathbf{a}; \pi] + \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_2(o_t^2, a_t^2) | \mathbf{s}_1 = \mathbf{s}, \mathbf{a}_1 = \mathbf{a}; \pi]$$

$$=: \bar{Q}_1^\pi(\mathbf{s}, \mathbf{a}) + \bar{Q}_2^\pi(\mathbf{s}, \mathbf{a})$$

where $\bar{Q}_i^\pi(\mathbf{s}, \mathbf{a}) := \mathbb{E}[\sum_{t=1}^{\infty} \gamma^{t-1} r_1(o_t^i, a_t^i) | \mathbf{s}_1 = \mathbf{s}, \mathbf{a}_1 = \mathbf{a}; \pi], i = 1, 2$. The action-value function $\bar{Q}_1^\pi(\mathbf{s}, \mathbf{a})$ – agent 1's expected future return – could be expected to depend more strongly on observations and actions (o^1, a^1) due to agent 1 than those due to agent 2. If (o^1, a^1) is not sufficient to fully model $\bar{Q}_1^\pi(\mathbf{s}, \mathbf{a})$ then agent 1 may store additional information from historical observations in its LSTM, or receive information from agent 2 in a communication channel, in which case we could expect the following approximation to be valid

$$Q^\pi(\mathbf{s}, \mathbf{a}) =: \bar{Q}_1^\pi(\mathbf{s}, \mathbf{a}) + \bar{Q}_2^\pi(\mathbf{s}, \mathbf{a}) \approx \tilde{Q}_1^\pi(h^1, a^1) + \tilde{Q}_2^\pi(h^2, a^2)$$

Our architecture therefore encourages this decomposition into simpler functions, if possible. We see that natural decompositions of this type arise in practice (see Section 4.4).

One approach to reducing the number of learnable parameters, is to share certain network weights between agents. Weight sharing also gives rise to the concept of agent invariance, which is useful for avoiding the lazy agent problem.

Definition 1 (Agent Invariance)**.** *If for any permutation (bijection)* $p : \{1, ..., d\} \to \{1, ..., d\}$,

$$\pi(p(\bar{h})) = p(\pi(\bar{h}))$$

we say that π *is* agent invariant*.*

It is not always desirable to have agent invariance, when for example specialized roles are required to optimize a particular system. In such cases we provide each agent with *role information*, or an identifier. The role information is provided to the agent as a 1-hot encoding of their identity concatenated with every observation at the first layer. When agents share all network weights they are then only *conditionally agent invariant*, i.e. have identical policies only when conditioned on the same role. We also consider information channels between agent networks, i.e. differentiable connections between agent network modules. These architectures, with shared weights, satisfy agent invariance.

4 Experiments

We introduce a range of two-player domains, and experimentally evaluate the introduced value-decomposition agents with different levels of enhancements, evaluating each addition in a logical sequence. We use two centralized agents as baselines, one of which is introduced here again relying on learned value-decomposition, as well as an individual agent learning directly from the joint reward signal. We perform this set of experiments on the same form of two dimensional maze environments used by Leibo et al. (2017), but with different tasks featuring more challenging coordination needs. Agents have a small $3 \times 5 \times 5$ observation window, the first dimension being an RGB channel, the second and third are the maze dimensions, and each agent sees a box 2 squares either side and 4 squares forwards, see Figures 1 and 2. The simple graphics of our domains helps with running speed while, especially due to their multi-agent nature and severe partial observability and aliasing (very small observation window combined with map symmetries), they still pose a serious challenge and is comparable to the state-of-the-art in multi-agent reinforcement learning (Leibo et al., 2017), which exceeds what is common in this area (Tuyls and Weiss, 2012).

[2]Or, more generally, across agent histories.

Agent	V.	S.	Id	L.	H.	C.
1						
2	✓					
3	✓	✓				
4	✓	✓	✓			
5	✓	✓	✓	✓		
6	✓	✓	✓		✓	
7	✓	✓	✓	✓	✓	
8	✓					✓
9						✓

Table 1: Agent architectures. V is value decomposition, S means shared weights and an invariant network, Id means role info was provided, L stands for lower-level communication, H for higher-level communication and C for centralization. These architectures were selected to show the advantages of the independent agent with value-decomposition and to study the benefits of additional enhancements added in a logical sequence.

4.1 Agents

Our agent's learning algorithm is based on DQN (Mnih et al., 2015) and includes its signature techniques of experience replay and target networks, enhanced with an LSTM value-network as in Hausknecht and Stone (2015) (to alleviate severe partial observability), learning with truncated back-propagation through time, multi-step updates with forward view eligibility traces (Harb and Precup, 2016) (which helps propagating learning back through longer sequences) and the dueling architecture (Wang et al., 2016) (which speeds up learning by generalizing across the action space). Since observations are from a local perspective, we do not benefit from convolutional networks, but use a fully connected linear layer to process the observations.

Our network architectures first process the input using a fully connected linear layer with 32 hidden units followed by a ReLU layer, and then an LSTM, with 32 hidden units followed by a ReLU layer, and finally a linear dueling layer, with 32 units. This produces a value function $V(s)$ and *advantage function* $A(s, a)$, which are combined to compute a Q-function $Q(s, a) = V(s) + A(s, a)$ as described in Wang et al. (2016). Layers of 32 units are sufficiently expressive for these tasks with limited observation windows.

The architectures (see Appendix B for detailed diagrams) differ between approaches by what is input into each layer. For architectures without centralization or information channels, one observation of size $3 \times 5 \times 5$ is fed to the first linear layer of 32 units, followed by the ReLU layer and the LSTM (see Figure 1). For the other (information channels and centralized) agents, d such observations are fed separately to identical such linear layers and then concatenated into 64 dimensional vectors before passing though ReLUs to an LSTM.

For architectures with information channels we concatenate the outputs of certain layers with those of other agents. To preserve agent invariance, the agent's own previous output is always included first. For low-level communication, the signal's concatenation is after the first fully connected layer, while for high-level communication the concatenation takes place on the output of the LSTM layer. Note, that this has the implication that what starts as one agent's gradients are back-propagated through much of the other agents network, optimizing them to serve the purposes of all agents. Hence, representing in that sense, a higher degree of centralization than the lower-level sharing.

We have found a trajectory length of 8, determining both the length of the forward view and the length of the back propagation through time is sufficient for these domains. We use an eligibility trace parameter $\lambda = 0.9$. In particular, the individual agents learning directly from the joint reward without decomposition or information channels, has worse performance with lower λ. The Adam (Kingma and Ba, 2014) learning rate scheme initialized with 0.0001 is uniformly used, and further fine-tuning this per agent (not domain) does not dramatically change the total performance. The agents that we evaluate are listed in the table above.

4.2 Environments

We use 2D grid worlds with the same basic functioning as Leibo et al. (2017), but with different tasks we call Switch, Fetch and Checkers. We have observations of byte values of size $3 \times 5 \times 5$ (RGB), which represent a window depending on the player's position and orientation by extending 4 squares ahead and 2 squares on each side. Hence, agents are very short-sighted. The actions are: step forward, step backward, step left, step right, rotate left, rotate right, use beam and stand still. The

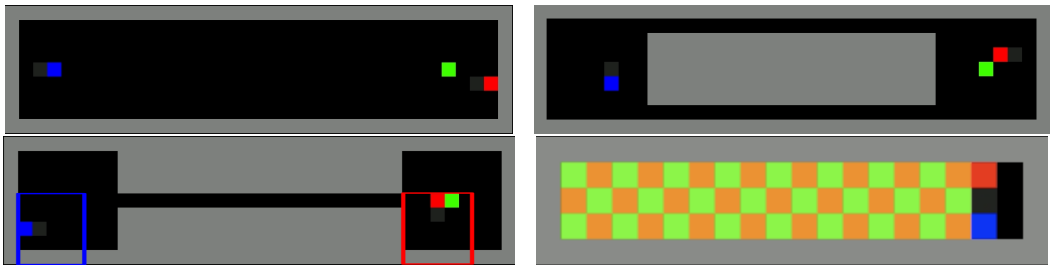

Figure 3: Maps for Fetch and Switch: open map (top left), map with 1 corridor (bottom left) and 2 corridors (top right). The green square is the goal for the agent to the left (blue). A similar goal is seen for the other agent (red) to the left but not displayed. The agents' observation windows are shown in the bottom left. Bottom right is the map for Checkers. Lemons are orange, apples green and agents red/blue.

beam has no effect in our games, except for lighting up a row or column of squares straight ahead with yellow. Each player appears as a blue square in its own observation, and the other player, when in the observation window, is shown in red for Switch and Escape, and lighter blue for Fetch. We use three different maps shown in Figure 3 for both Fetch and Switch and a different one for Checkers, also shown in Figure 3 (bottom right). The tasks repeat as the agents succeed (either by full reset of the environment in Switch and Checkers or just by pickup being available again in Fetch), in training for 5,000 steps and 2,000 in testing.

Switch: The task tests if two agents can effectively coordinate their use of available routes on two maps with narrow corridors. The task is challenging because of strong observation aliasing. The two agents appear on different ends of a map, and must reach a goal at the other end. If agents collide in a corridor, then one agent needs to leave the corridor to allow the other to pass. When both players have reached their goal the environment is reset. A point is scored whenever a player reaches a goal.

Fetch: The task tests if two agents can synchronize their behaviour, when picking up objects and returning them to a drop point. In the Fetch task both players start on the same side of the map and have pickup points on the opposite side. A player scores 3 points for the team for pick-up, and another 5 points for dropping off the item at the drop point near the starting position. Then the pickup is available to either player again. It is optimal for the agents to cycle such that when one player reaches the pickup point the other returns to base, to be ready to pick up again.

Checkers: The map contains apples and lemons. The first player is very sensitive and scores 10 for the team for an apple (green square) and -10 for a lemon (orange square). The second, less sensitive player scores 1 for the team for an apple and -1 for a lemon. There is a wall of lemons between the players and the apples. Apples and lemons disappear when collected, and the environment resets when all apples are eaten. It is important that the sensitive agent eats the apples while the less sensitive agent should leave them to its team mate but clear the way by eating obstructing lemons.

4.3 Results

We compare the eight approaches listed in Table 1, on the seven tasks. Each is run ten times, with different random seeds determining spawn points in the environment, as well as initializations of the neural networks. We calculated curves of the average performance over 50,000 episodes (plots in Appendix A) for each approach on each task and we display the normalized area under the curve in Figure 4. Figure 5 displays the normalized final performance averaged over runs and the last 1,000 episodes. Average performance across tasks is also shown for both ways of evaluation.

The very clear conclusion is that architectures based on value-decomposition perform much better, with any combination of other techniques or none, than the centralized approach and individual learners. The centralized agent with value-decomposition is better than the combinatorially centralized as well as individual learners while worse than the more individual agents with value-decomposition.

We particularly see the benefit of shared weights on the hard task of Fetch with one corridor. Without sharing, the individual value-decomposition agent suffers from the lazy agent problem. The agent with weight sharing and role information also perfectly learns the one corridor Fetch task. It performs

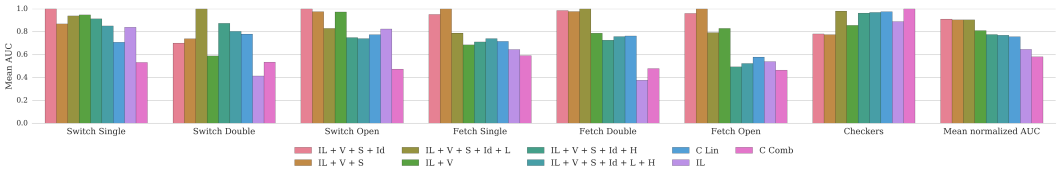

Figure 4: Barplots showing normalized AUC for each agent and domain over 50000 episodes of training and the mean across domains.

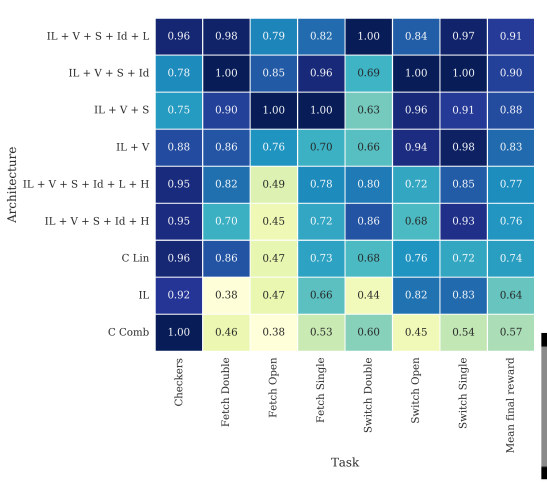

Figure 5: Heatmap showing each agent's final performance, averaged over the last 5,000 episodes of 50,000 and across ten runs, normalized by the best architecture per task. The agents are ordered according to average over the domains, which can be seen in the right most column. Value-Decomposition architecture strongly outperform Individual Learners and Centralization

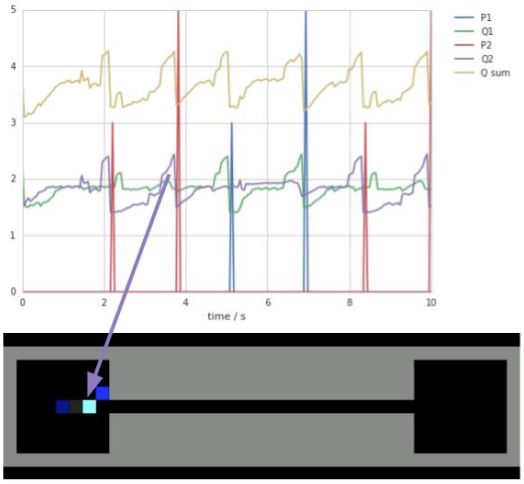

Figure 6: The learned Q-decomposition in Fetch. The plot shows the total Q-function (yellow), the value of agent 1 (green), the value of agent 2 (purple), rewards from agent 1 (blue) events and agent 2 (red). Highlighted is a situation in which agent 2's Q-function spikes (purple line), anticipating reward for an imminent drop-off. The other agent's Q-function (green) remains relatively flat.

better than the agent just sharing weights on Switch, where coordination, in particular with one corridor, is easier with non-identical agents. Further, shared weights are problematic for the Checkers task because the magnitude of rewards (and hence the value function) from one agent is ten times higher than for the other agent.

Adding information channels does increase learning complexity because the input comes from more than one agent. However, the checkers task, designed for the purpose, shows that it can be very useful. Overall, the low-level channels where the agent's LSTM processes the combined observations of both agents turned out to learn faster in our experiments than the more centralized high level communication (after the LSTM).

4.4 The Learned Q-Decomposition

Figure 6 shows the learned Q-decomposition for the value-decomposition network, using shared weights, in the game of Fetch. A video of the corresponding game can be seen at Video (2017). Spikes correspond to pick-up events (short spikes, 3 reward points), and return events (large spikes, 5 reward points). These are separated into events due to agent 1 (blue spikes) and agent 2 (red spikes). This disambiguation is for illustration purposes only: the environment gives a reward to the whole team for all of these events. The total Q-function is seen in yellow, clearly anticipating the team reward events, and dropping shortly afterwards. The component Q-functions $\tilde{Q}_1$ and $\tilde{Q}_2$ for agents 1 and 2 are shown in green and purple. These have generally disambiguated the Q-function into rewarding events separately attributable to either player. The system has learned to autonomously decompose the

joint Q-function into sensible components which, when combined, result in an effective Q-function. This would be difficult for independent learners since many rewards would not be observed by both players, see e.g. the situation at 15-16 seconds in the corresponding video available at Video (2017).

5 Conclusions

We study cooperative multi-agent reinforcement learning where only a single joint reward is provided to the agents. We found that the two naive approaches, individual agents learning directly from team reward, and fully centralized agents, provide unsatisfactory solutions as previous literature has found in simpler environments, while our value-decomposition networks do not suffer from the same problems and shows much better performance across a range of more complex tasks. Further, the approach can be nicely combined with weight sharing and information channels, leading to agents that consistently optimally solve our new benchmark challenges.

Value-decomposition networks are a step towards automatically decomposing complex learning problems into local, more readily learnable sub-problems. In future work we will investigate the scaling of value-decomposition with growing team sizes, which make individual learners with team reward even more confused (they mostly see rewards from other agents actions), and centralized learners even more impractical. We will also investigate decompositions based on non-linear value aggregation.

References

A. K. Agogino and K. Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic environments. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

M. Babes, E. M. de Cote, and M. L. Littman. Social reward shaping in the prisoner's dilemma. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 3*, pages 1389–1392, 2008.

D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov Decision Processes. In *UAI '00: Proceedings of the 16th Conference in Uncertainty in Artificial Intelligence, Stanford University, Stanford, California, USA, June 30 - July 3, 2000*, pages 32–37, 2000.

L. Busoniu, R. Babuska, and B. D. Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions of Systems, Man, and Cybernetics Part C: Applications and Reviews*, 38(2), 2008.

C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 746–752, 1998.

M. Colby, T. Duchow-Pressley, J. J. Chung, and K. Tumer. Local approximation of difference evaluation functions. In *Proceedings of the Fifteenth International Joint Conference on Autonomous Agents and Multiagent Systems*, Singapore, May 2016.

S. Devlin, L. Yliniemi, D. Kudenko, and K. Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings of the Thirteenth International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2014.

A. Eck, L. Soh, S. Devlin, and D. Kudenko. Potential-based reward shaping for finite horizon online POMDP planning. *Autonomous Agents and Multi-Agent Systems*, 30(3):403–445, 2016.

J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2137–2145, 2016.

C. Guestrin, M. G. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 227–234, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-873-7. URL `http://dl.acm.org/citation.cfm?id=645531.757784`.

J. Harb and D. Precup. Investigating recurrence and eligibility traces in deep Q-networks. In *Deep Reinforcement Learning Workshop, NIPS 2016, Barcelona, Spain*, 2016.

M. J. Hausknecht. *Cooperation and Communication in Multiagent Deep Reinforcement Learning*. PhD thesis, The University of Texas at Austin, 2016.

M. J. Hausknecht and P. Stone. Deep recurrent Q-learning for partially observable MDPs. *CoRR*, abs/1507.06527, 2015.

C. HolmesParker, A. Agogino, and K. Tumer. Combining reward shaping and hierarchies for scaling to large multiagent systems. *Knowledge Engineering Review*, 2016. to appear.

J. Hu and M. P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4:1039–1069, 2003.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

L. Kuyer, S. Whiteson, B. Bakker, and N. A. Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part I*, pages 656–671, 2008.

G. J. Laurent, L. Matignon, and N. L. Fort-Piat. The world of independent learners is not Markovian. *Int. J. Know.-Based Intell. Eng. Syst.*, 15(1):55–64, 2011.

J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017)*, Sao Paulo, Brazil, 2017.

M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 157–163, 1994.

M. L. Littman. Friend-or-foe q-learning in general-sum games. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 322–328, 2001.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1928–1937, 2016.

A. Y. Ng, D. Harada, and S. J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 278–287, 1999.

F. A. Oliehoek and C. Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer, 2016.

F. A. Oliehoek, M. T. J. Spaan, and N. A. Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *J. Artif. Intell. Res. (JAIR)*, 32:289–353, 2008.

L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

S. Proper and K. Tumer. Modeling difference rewards for multiagent learning (extended abstract). In *Proceedings of the Eleventh International Joint Conference on Autonomous Agents and Multiagent Systems*, Valencia, Spain, June 2012.

M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.

S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 3^{nd} edition, 2010.

S. J. Russell and A. Zimdars. Q-decomposition for reinforcement learning agents. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 656–663, 2003.

J. G. Schneider, W. Wong, A. W. Moore, and M. A. Riedmiller. Distributed value functions. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 371–378, 1999.

S. Sukhbaatar, A. Szlam, and R. Fergus. Learning multiagent communication with backpropagation. *CoRR*, abs/1605.07736, 2016. URL `http://arxiv.org/abs/1605.07736`.

R. Sutton and A. Barto. *Reinforcement Learning*. The MIT Press, 1998.

C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

K. Tumer and D. Wolpert. A survey of collectives. In K. Tumer and D. Wolpert, editors, *Collectives and the Design of Complex Systems*, pages 1–42. Springer, 2004.

K. Tuyls and G. Weiss. Multiagent learning: Basics, challenges, and prospects. *AI Magazine*, 33(3): 41–52, 2012.

E. van der Pol and F. A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. *NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems*, 2016.

Video. Video for the q-decomposition plot. 2017. URL `https://youtu.be/aAH1eyUQsRo`.

Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1995–2003, 2016.

Appendix A: Plots

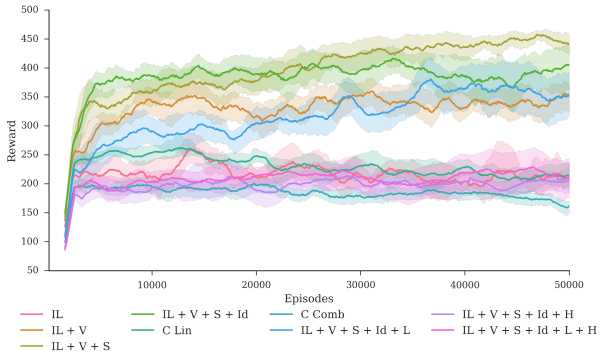

Figure 7: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Fetch domain with the open map

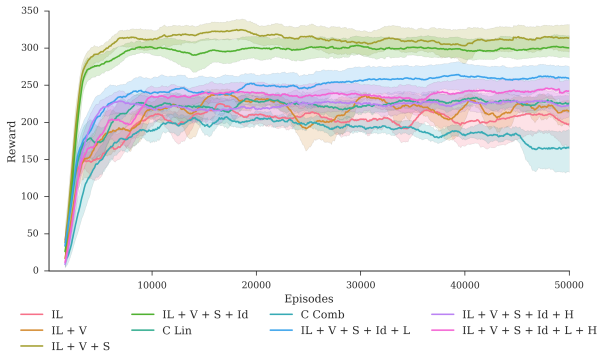

Figure 8: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Fetch domain with one corridor

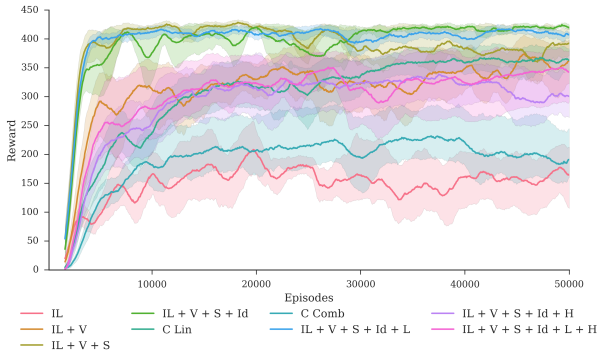

Figure 9: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Fetch domain with two corridors

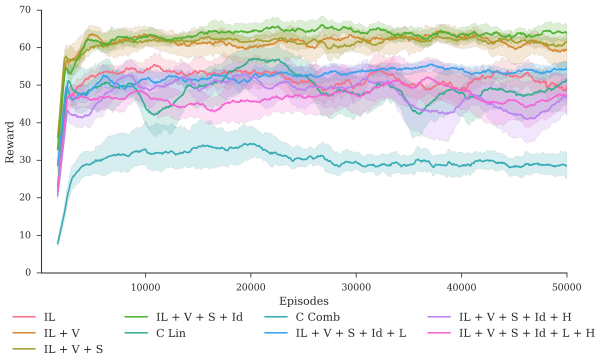

Figure 10: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Switch domain with the open map

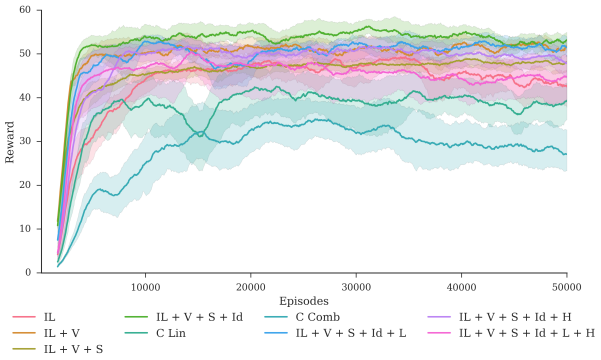

Figure 11: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Switch domain with one corridor

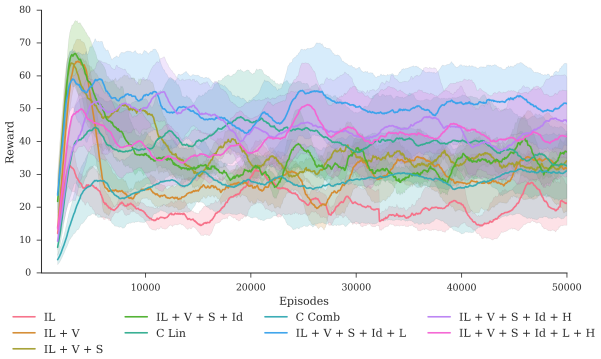

Figure 12: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Switch domain with two corridors

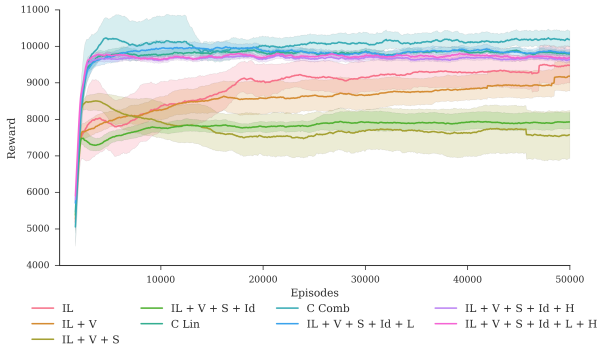

Figure 13: Average reward with 90% confidence intervals for ten runs of the nine architectures on the Checkers domain

Appendix B: Diagrams

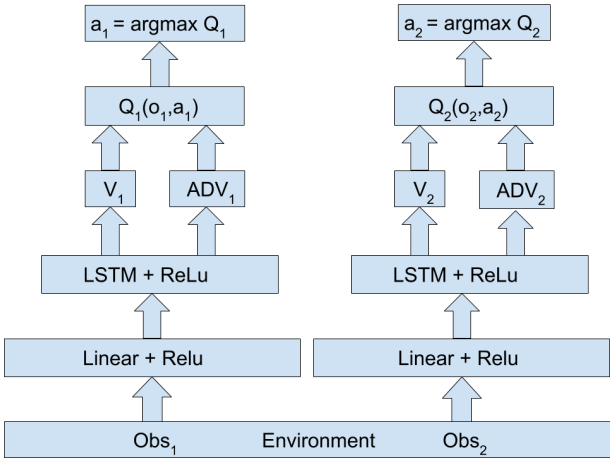

Figure 14: Independent Agents Architecture

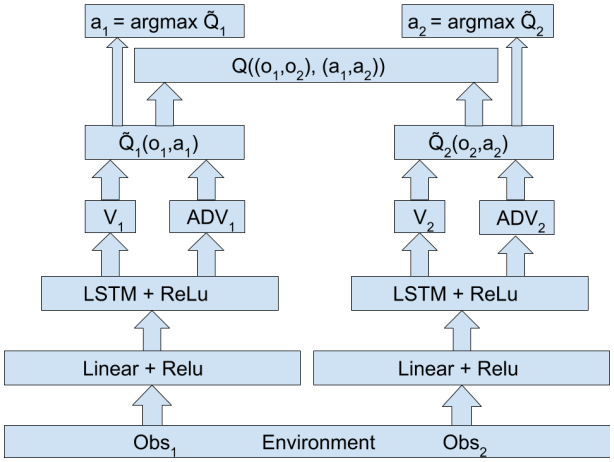

Figure 15: Value-Decomposition Individual Architecture

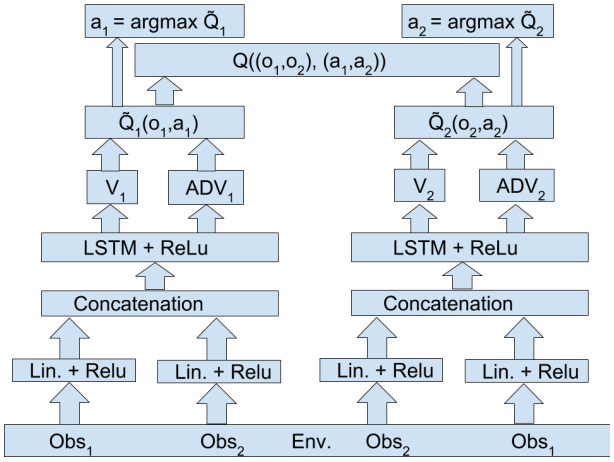

Figure 16: Low-level communication Architecture

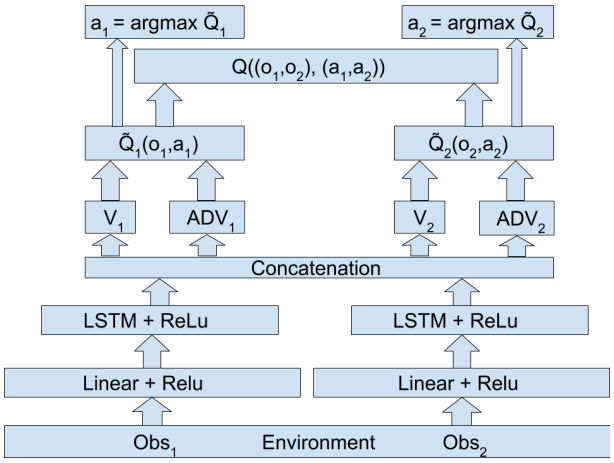

Figure 17: High-level communication Architecture

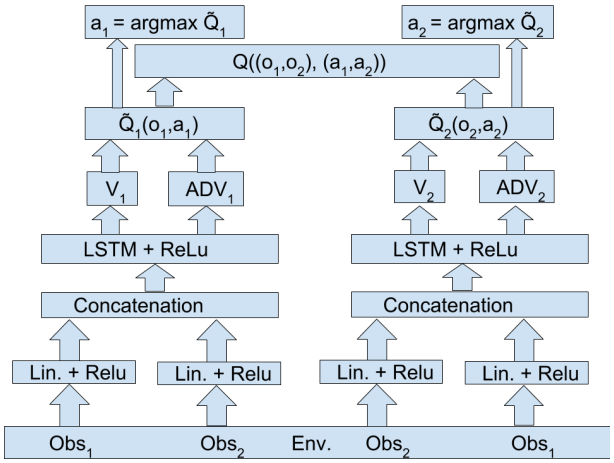

Figure 18: Low-level communication Architecture

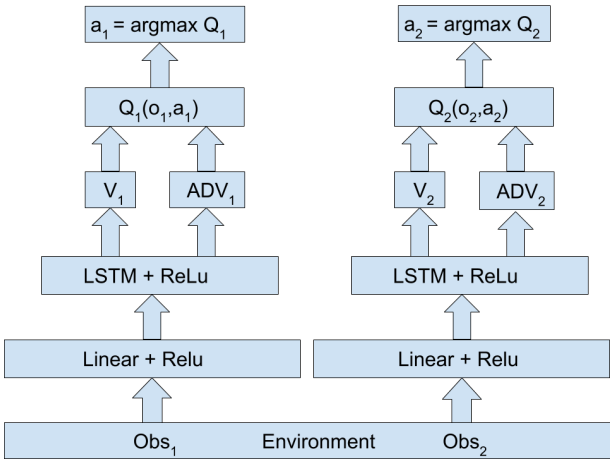

Figure 19: Independent Agents Architecture

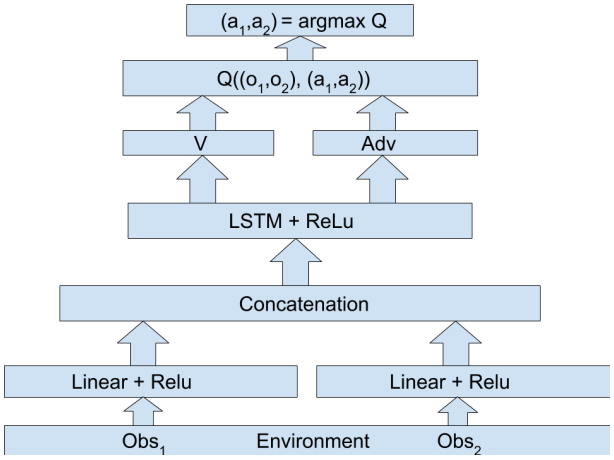

Figure 20: Combinatorially Centralized Architecture

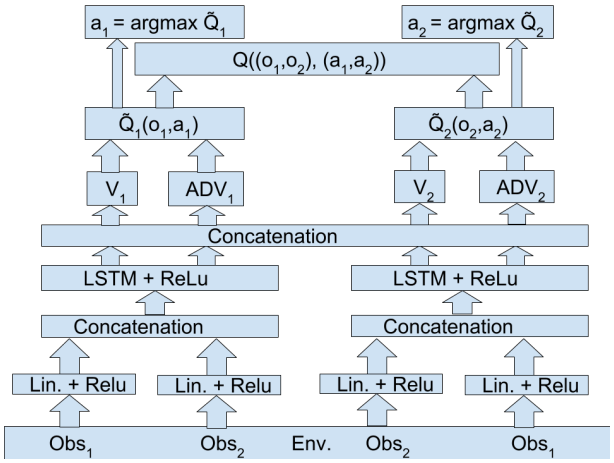

Figure 21: High+Low-level communication Architecture