CCNet: Cluster-Coordinated Net for Learning Multi-agent Communication Protocols with Reinforcement Learning

Xin Wen wenxin12@mail.ustc.edu.cn
Zheng-Jun Zha [*] zhazj@ustc.edu.cn
Zilei Wang zlwang@ustc.edu.cn
Liansheng Zhuang lszhuang@ustc.edu.cn
Houqiang Li lihq@ustc.edu.cn
University of Science and Technology of China, Hefei, China

Editors: Jun Zhu and Ichiro Takeuchi

Abstract

Multi-agent system is crucial for many practical applications. Recent years have witnessed numerous research on multi-agent task with reinforcement learning (RL) algorithms. Traditional reinforcement learning algorithms often fail to learn the cooperation between different agents, which is vital for multi-agent problems. A promising solution is to establish a communication protocol among agents. However, existing approaches often suffer from generalization challenges especially in tasks with partial observation and dynamic variation of agent amount. In this paper, we develop a Cluster-Coordinated Network (CCNet) to address the "Learning-to-communicate" problem in multi-agent system by utilizing the combination of a trainable Vector of Locally Aggregated Descriptor (VLAD) algorithm and reinforcement learning. Embedding with a VLAD based end-to-end trainable communication information processing module (called VLAD Processing Core), CCNet can learn efficient communication protocols even from scratch under partially observable environments and possesses robustness to the dynamic changes of agent number as well. Moreover, with the help of communication, CCNet is with less non-stationarity when training the network by common RL algorithms. We evaluated the proposed CCNet on two multi-agent partially observable tasks, *i.e.*, Traffic Junction and Combat Task. The experimental results have demonstrated that CCNet is effective and improves the performance by a large margin over the state-of-the-art methods.

Keywords: multi-agent reinforcement learning, communication protocol, clustering, neural network

1. Introduction

Artificial Intelligence (AI) has witnessed great progresses in the past few decades, especially after the development of deep learning. However, there are still many road stones standing in the way of reaching the great goal of Artificial General Intelligence (AGI) (Goertzel and Pennachin, 2007), and how to learn an optimal policy in multi-agent environments is one of those stones. Many social works involve interactions between multiple agents, such as rubbish removal (Makar et al., 2001), urban traffic control (Kuyer et al., 2008; Van der Pol and Oliehoek, 2016), public resource appropriation (Perolat et al., 2017) and network

[*] Corresponding author

packet delivery (Ye et al., 2015). Moreover, multi-agent self-play has been proved to be crucial when combined with other methods like tree search (Silver et al., 2017b,a).

On the other hand, Reinforcement learning (RL) has recently shown effectiveness when coping with many challenging tasks, such as game playing (Mnih et al., 2015; Silver et al., 2017b), robots controlling (Levine et al., 2016) and data center cooling (DeepMind, 2016). RL has also been applied to multi-agent tasks, known as multi-agent reinforcement learning (MARL) (Busoniu et al., 2008).

One straightforward way of scaling traditional RL algorithms into multi-agent environments is Independent Q-learning (IQL) (Tan, 1993), in which each agent learns its own policy independently, regarding other agents as part of the environment. In MARL, the reward can be seen as a function of the global state and the joint action of all agents: $r(S, A_1, ..., A_N)$. IQL chooses actions according to its own observations only, regarding the reward function as: $r_{IQL}(S, A_i)$. The approximate errors between these two reward functions are often too large and result in the non-stationary problem. From the perspective of any individual agent, the received rewards may be different even though its policy stays the same. Furthermore, non-stationarity leads to stability challenges of training when we combine IQL with the use of past experience replay (Mnih et al., 2015): the non-stationarity brought by IQL indicates that the environment is also changing as the training goes on. Data generated by different versions of the environment is mixed together in the experience replay and then sampled uniformly, this further causes the stability problem of training. Although there are some RL algorithms that do not rely on experience replay and these methods can sometimes learn well in multi-agent environments so long as each agent can get access to the other agents' policies (Foerster et al., 2017b), how to track and represent the policies of other agents remains a challenging problem.

To reduce the non-stationarity, a much better approximation of the reward function can be written as: $r_{hide}(S, h_1, ..., h_N)$, in which agents share their hide states to help estimating other agents' actions. Following this intuition, many researchers tried to solve the MARL problem by pre-defining a communication protocol between agents (Busoniu et al., 2008; Sutton and Barto, 1998), so that the hide states of agents can be shared in a relatively practical way. The more effective the communication method is, the more helpful it will be to reduce non-stationarity. However, these methods cannot be directly generalized to different environments or tasks with large collection of agents, which hinders their applications in practical tasks. Recently, a few of preliminary works have been proposed to enable agents to learn communication protocols all by themselves (Sukhbaatar et al., 2016; Foerster et al., 2016; Mao et al., 2017), demonstrating that although with some limits (*e.g.*, the protocol is still too simple, the number of agents must be settled), this way of solutions is promising.

In this paper, we propose a Cluster-Coordinated Net (CCNet) framework, which coordinates agents to learn how to communicate with each other in a more effective way. Each agent is modeled by a feed-forward neural network, and all agents' models are connected together by a communication channel in which the communication messages as well as gradients flows. Inspired by Arandjelovic et al. (2016) and the Vector of Locally Aggregated Descriptors (VLAD) representation (Jégou et al., 2010), we propose a VLAD Processing Core which is amenable to training via backpropagation to process communication messages (each agent generates one message), and thus enables agents to learn their own communication protocols with common RL algorithms or supervised learning. Our framework has three advantages: ***(a)*** in VLAD Processing Core, the dimension of the clustered results is independent of the number of input communication messages. In other words, VLAD Processing Core can be seen as a pooling method which turns the input communication messages with dynamic dimension into a vector with deterministic dimension. This enables the framework to deal with tasks in which the number of agents changes frequently. ***(b)*** the clustered vectors are expected to learn an effective representation of all agents' status according to communication messages. Different cluster centers represent different typical status agents may be in, and the intensity of each clustered vector represents the number of agents which are in this status. ***(c)*** the clustered communication information is an effective representation of all agents' hide states, so that our framework suffers less non-stationarity when combined with classical RL algorithms. These advantages allow the proposed framework to be more suitable to tackle a wide range of more complicated tasks involving partial visibilities of the environment and dynamic changes in the number of agents.

We consider the setting in which the agents are fully cooperative, which means that agents receive the same reward r from the environment independent of their contribution. All agents are cooperating to maximize the reward r. With parameter sharing, agents of the same type can be seen and trained as a same deep feed-forward network with different inputs. From the perspective of a single agent, the network maps its observation of the environment as well as the clustered communication information to its action.

We explore the proposed CCNet on two tasks which have different settings, most of them are under partially observable environments. Since the policy of each agent is learned totally from scratch, RL algorithms must be used to provide a training signal for the model to update its parameters, but note that our framework can also be directly trained via supervised learning. Our framework outperforms the state-of-the-art results by a large margin in nearly every setting, which indicates the CCNet to be effective to solve multi-agent tasks.

2. Related Work

Researchers have been studying the collaborations and interactions in multi-agent environments for decades and reinforcement learning has been used to learn optimal collaboration policies from the very beginning (Littman, 1994; Schmidhuber, 1996; Tan, 1993). As a result of that, there is a rich literature on the MARL problem, especially in the domains like robotics (Fox et al., 2000). Building a communication protocol among agents has always been a key component in solving the MARL problem, but most of the traditional approaches use a pre-defined protocol (Tan, 1993; Maravall et al., 2013). Recently, many solutions which enable agents to learn communication protocols via backpropagation have been proposed, such as differentiable inter-agent learning (DIAL) (Foerster et al., 2016), CommNet (Sukhbaatar et al., 2016), Bidirectionally- Coordinated Network (BiCNet) (Peng et al., 2017) and VAIN (Hoshen, 2017).

DIAL (Foerster et al., 2016) is one of the pioneers in this domain, in which the communication information is able to be broadcast between agents in order to solve partially observable tasks like riddles. In DIAL, each agent is modeled by a recurrent neural network which outputs the Q-value of an individual agent together with a communication message at each timestep. The generated message is then broadcast to another agent and used as part of the inputs of that agent at the next timestep. DIAL builds a channel in which messages as well as gradients can be propagated between different agents. From the perspective of each agent, the global information is represented by its current observation, last action, and the received message. However, DIAL broadcasts the message in a discrete manner with the lag of one timestep and only two agents are involved, while our approach allows continuous message to be shared among arbitrary number of agents at the current timestep.

CommNet (Sukhbaatar et al., 2016) is designed for agents to learn their joint action policies and is the closest approach to ours. Unlike DIAL, CommNet can be viewed as a large model which consists of many small controllers. Each controller represents an individual agent and all agents are connected together by a communication channel. CommNet employed a mean-pooling method to make sure the model can be able to deal with the environments in which the number of agents changes dynamically. However, the mean-pooling method is sometimes too simple to process and carry enough communication information and thus has limits when handling sophisticated tasks, especially when involving large amount of agents or lacking observations. This is the reason why our CCNet outperforms CommNet in most settings.

BiCNet (Peng et al., 2017) chooses bi-directional recurrent neural network to deliver communication information. In contrast to DIAL and CommNet, BiCNet is based on Actor-Critic framework and the output actions are represented in continuous space. The RNN architecture enables a single agent to share its information with other teammates one by one with the expanding of the network. However, the communication information of a certain agent will vanish gradually due to the RNN's sequential connected structure, because the information generated earlier may be covered when passing through the subsequent agents. In our method, the order of agents does not matter since all the communication messages are processed together and there is no such vanishing problem.

Recently, the idea of centralized learning and decentralized execution has been proposed. Lowe et al. (2017) introduced MADDPG framework and evaluated it in mixed cooperative-competitive environments. The COMA model tries to solve the problem of credit assignment in multi-agent environments through counterfactual rewards (Foerster et al., 2017a). ACCNet (Mao et al., 2017) followed these ideas and proposed a decentralized learning and centralized execution framework. But these methods may not work well in the environments where the number of agents changes dynamically and cannot scale to environments with huge amount of agents, since the number of input units of the network is settled. Foerster et al. (2017b); Usunier et al. (2016) and Shao et al. (2018) attempt to tackle the multi-agent tasks from the perspective of IQL by improving the training procedure, reply buffer and the method of reward assignment, but they do not focus on the partially observable environments.

3. Background

3.1. Problem Formulation

In this paper, CCNet needs to control a group of agents to achieve their goals together. We formulate the environment as a Markov game (Littman, 1994), which is a partially observable Markov decision process (POMDP) with an extension to multi-agent system. In a Markov game with N agents, a set of S is used to describe the properties of the environment as well as the status of all agents, and a set of observations $O_1, \ldots, O_N$ together with a set of actions $A_1, \ldots, A_N$ for agents. Each agent i chooses its own action according to a stochastic or learned policy π_{θ_i}: $O_i \times A_i \rightarrow [0,1]$. Then the successive state S' is generated based on the state transition function $\tau : S \times A_1 \times \ldots \times A_N \rightarrow S'$. Each agent i receives a private observation of the state $o_i : S \rightarrow O_i$ and a reward r_i as a function of the environment state and the joint action of all agents $r_i : S \times A_1 \times \ldots \times A_N \rightarrow \mathbb{R}$. Since our work considers the setting where the agents are fully cooperative, rewards of all agents are identical and equal to r, which means $r_1 = \ldots = r_N = r$. The goal of the learned policies is to maximize the total expected reward $R = \sum_{t=1}^{T} \gamma^{t-1} r^t$, where r^t is the reward received at timestep t, $\gamma \in [0,1]$ denotes a discount factor and T is the length of a game episode. Figure.1 shows the diagram of a Markov game and demonstrates how agents interact with the environment in multi-agent system.

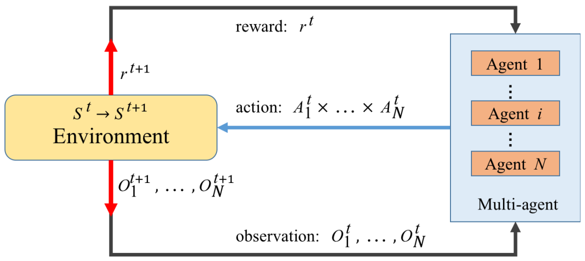
reward: r^t
r^{t+1}
$s^t \rightarrow s^{t+1}$
Environment
action: $A_1^t \times \ldots \times A_N^t$
Agent 1
Agent i
Agent N
Multi-agent
$o_1^{t+1}, \ldots, o_N^{t+1}$
observation: $o_1^t, \ldots, o_N^t$

Figure 1: Demonstration of how RL agents interact with the environment in multi-agent system.

3.2. Reinforcement Learning

Deep Q-Network(DQN). Q-learning (Watkins, 1989) has always been one of the most popular methods in reinforcement learning. In Q-learning, an action-value function called Q-function is defined to measure the effect of actions. The Q-function of a policy π is $Q^{\pi}(s,a) = \mathbb{E}[R|s^t = s, a^t = a]$. The optimal Q-function $Q^*(s,a) = \max_a Q^{\pi}(s,a)$ follows the Bellman optimality equation and can be recursively written as $Q^*(s,a) = \mathbb{E}_{s'}[r(s,a) + \gamma \max_{a'} Q^*(s',a')|s,a]$. DQN trains a neural network to approximate the optimal Q-function with parameters θ, written as $Q(s,a;\theta)$ (Mnih et al., 2015). DQN learns Q^* by minimizing the loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,r,s'}[(y_i^{DQN} - Q(s,a;\theta))^2] \tag{1}$$

where $y_i^{DQN} = r + \max_{\mathrm{a'}} \mathrm{Q}(\mathrm{s'}, \mathrm{a'}; \theta^-)$, and θ^- denotes the parameters of a target network copied periodically from θ and is not updated by gradients. During training, the action is chosen from $Q(s,a;\theta)$ based on an $\varepsilon -$ greedy policy which chooses the action randomly with a probability of ε and selects the action according to $Q^*(s,a;\theta)$ with a probability of $1-\varepsilon$. Another important component of DQN is the use of an experience replay buffer which stores the $< s, a, r, s' >$ tuple to stabilize the training procedure, but experience replay has many limits when applied to multi-agent tasks.

Policy Gradient (PG) Algorithms. Methods based on Policy gradient (Sutton et al., 2000) are another popular choice when solving RL tasks. In contrast to value-based methods, policy gradient methods directly model the agents' policies $\pi_\theta(a|s)$ with parameters θ. The parameters are updated by applying gradient ascent on the objective $J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[R]$, where p^π is the state distribution and actions are selected according to the modeled policy. Basic PG algorithm updates θ in the direction:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)Q^\pi(s,a)] \tag{2}$$

Estimating Q^π in different ways gives rise to a variety of practical algorithms. For instance, one could learn a model of Q^π by Q-learning, which leads to the actor-critic algorithms (Sutton and Barto, 1998). Standard REINFORCE algorithm (Williams, 1992) simply uses the total expect reward during a period of time $R^t = \sum_{i=t}^{T} \gamma^{i-t} r^i$ to estimate Q^π, which is an unbiased estimate of the objective. However, policy gradient methods often suffer the high variance gradient estimates, and we can reduce this variance and keep it unbiased at the same time by subtracting a baseline $b(s^t)$ (Williams, 1992), which is a learned estimation of the state basing on the reward. So the gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)(R^t - b(s^t))] \tag{3}$$

4. Methods

In this section, we demonstrate the detailed structure of our Cluster-Coordinated Net and how the communication information is clustered in VLAD Processing Core. In general, our framework outputs the distribution over actions $p(a_i^t|s_i^t;\theta)$ of all agents at any timestep t, and t will be omitted for brevity.

4.1. Cluster-Coordinated Net Framework

In CCNet, each agent is modeled by a multilayer feed-forward neural network called *agent-model* (shown in Figure.2) and all the *agent-model*s are connected together by a communication channel. The *agent-model* of agent i can be written as a function f_i which takes its observation of the environment o_i and the clustered communication information it receives c_i as inputs, then map them to a distribution over actions. From the perspective of network structure, o_i and c_i are fed into different layers of the *agent-model*.

Each *agent-model* consists of an encoder and a decoder. **Firstly,** for agent i, there is an encoder function $e(\cdot)$ which maps its observation o_i into a hidden state h_i, written as $h_i = e(o_i)$. We choose a two-layer perceptron as the encoder, and the form of encoder can be replaced by other models like Look-Up-Table depending on the specific task. **Secondly,** h_i is delivered to VLAD Processing Core through communication channel. In VLAD Processing Core, hidden states of all agents are stacked together and become a dataset $\mathcal{D} = \{h_1, \ldots, h_N\}$, then a clustering procedure is processed and outputs K vectors, where K is the number of cluster centers, so VLAD Processing Core can be written as a function:

$$c = f_{VLAD}(h_1, \ldots, h_N) \tag{4}$$

where c is the concatenated result of these K vectors. c can be seen as a concise representation of all agents' status and is then broadcast to all agents, which means $c_1 = \ldots = c_N = c$.

Lastly, for agent i, a decoder $d(\cdot)$ which takes h_i and c_i as inputs is used to generate the distribution over actions. The decoder is also a two-layer perceptron and followed by a softmax function, so the results are computed by:

$$p(a_i|s) = p(a_i|o_i) = d(h_i, c_i) \tag{5}$$
$$d(h_i, c_i) = f_{softmax}(\sigma(Hh_i + Cc_i)) \tag{6}$$

in which σ is a non-linear activation function, H and C are parameters. Agent selects its action by sampling from the output of decoder: $a_i \sim d(h_i, c_i)$. As shown in Figure.2, the CCNet framework can also be treated as a larger model Φ, which maps a concatenation of all agents' observations $\mathcal{O} = \{o_1, \ldots, o_N\}$ to a concatenation of discrete actions $\mathcal{A} = \{a_1, \ldots, a_N\}$. Note that the entire CCNet framework includes VLAD Processing Core as well as the *agent-model* of each agent. Parameters of the CCNet can be viewed as a communication protocol between agents, which includes encoding, information processing and decoding procedures. The number and order of agents do not matter in our framework because of the connection method between *agent-models* and VLAD Processing Core. The operating mechanism of VLAD Processing Core is illustrated in the following section.

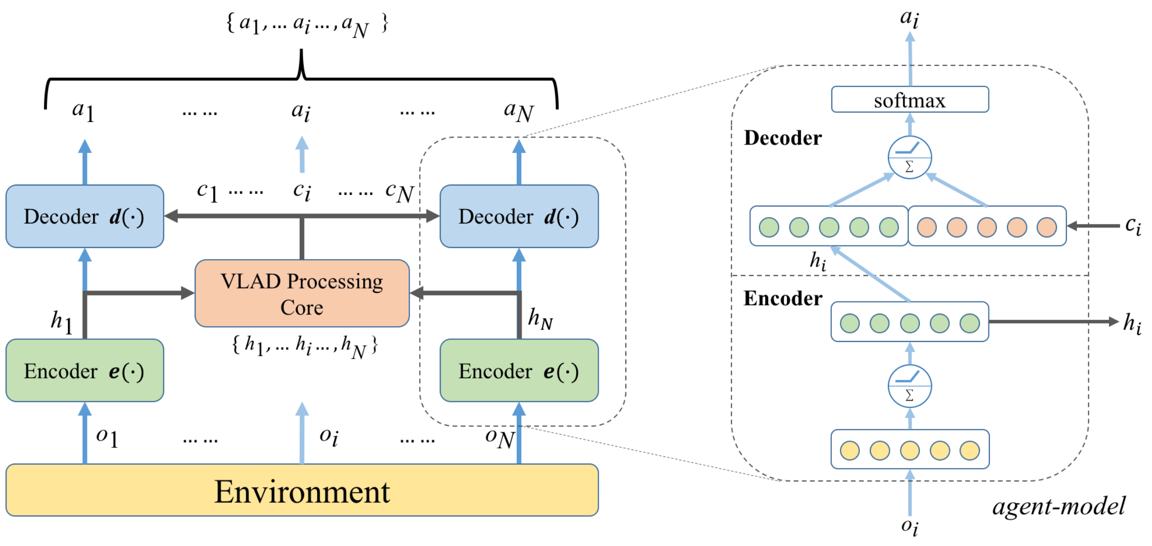

Figure 2: An overview of our proposed CCNet framework. **Left**: the whole framework of CCNet Φ, showing how the information flows inside the framework. Note that the inputs of VLAD Processing Core are the hidden states of all agents $\{h_1, \ldots, h_N\}$, and its output c is shared with all agents. **Right**: inner structure of an *agent-model*, each *agent-model* corresponds to a single agent i. Among the agents of same type, parameters of the *agent-models* are shared in order to speed up the training procedure and reduce the non-stationary at the same time.

4.2. VLAD Processing Core

Vector of Locally Aggregated Descriptors (VLAD). VLAD (Jégou et al., 2010) is a well-known descriptor pooling method which is often used in the areas like image classification (Gong et al., 2014) and instance level retrieval. VLAD is represented by the sum of the difference vectors between the descriptors and their assigned cluster centers.

Assuming each descriptor to be a D-dimensional vector (in our case, descriptors correspond to the hidden states of agents, so we have N descriptors), and there are K cluster centers (the coordinates of k-th cluster center in feature space is represented by u_k), then the output VLAD representation V is a $K \times D$ dimensional matrix. V is viewed as an effective representation of images (in our case, a representation of the hidden states of all agents). The (k, j) element of V is computed as follows:

$$V(k,j) = \sum_{i=1}^{N} q_k(h_i)(h_i(j) - u_k(j)) \tag{7}$$

where $h_i(j)$ and $u_k(j)$ denote the j-th dimension of the i-th descriptor and the j-th dimension of the k-th cluster center. $q_k(h_i)$ is an assignment function, and $q_k(h_i) = 1$ when the closest cluster center to h_i is k and $q_k(h_i) = 0$ otherwise. The matrix V is then converted to a vector v and subsequently L_2-normalized by $v := v/\|v\|$.

Trainable VLAD. In traditional VLAD algorithms, the cluster centers are learned in advance by other clustering algorithms like k-means. These methods have great limits when directly applied to multi-agent reinforcement learning tasks, because the communication messages of agents may change dramatically throughout the training procedure and the cluster centers needs to be modified constantly to adapt to the new situations. In order to benefit from the years of achievements in VLAD and conquer its limits, we introduce an improved VLAD algorithm which is trainable via backpropagation. "Trainable" means that operations inside VLAD need to be differentiable with respect to all their parameters as well as inputs.

Traditional VLAD is non-differentiable mainly because the hard assignment function $q_k(h_i)$ outputs either 0 or 1. Inspired by Arandjelovic et al. (2016), we replace $q_k(h_i)$ with a soft assignment function which assigns descriptors to all clusters according to distance:

$$\bar{q}_k(h_i) = \frac{e^{-\alpha\|h_i - u_k\|^2}}{\sum_{k'} e^{-\alpha\|h_i - u_{k'}\|^2}} \tag{8}$$

This equation assigns descriptor h_i to cluster center k with a weight which is proportional to the square of distance between them. With normalization, $\bar{q}_k(h_i)$ ranges from 0 to 1, and the closest cluster center has the highest weight. α is a positive constant which controls the weight distribution between the distances with different magnitudes. Note that $\bar{q}_k(h_i)$ is equivalent to the original hard assignment function when $\alpha \to +\infty$, since $\bar{q}_k(h_i) = 1$ for the closest cluster center and 0 otherwise.

Expanding the squares in (8) can lead to a simpler equation:

$$\bar{q}_k(h_i) = \frac{e^{-\alpha(\|h_i\|^2 - 2u_k h_i + \|u_k\|^2)}}{\sum_{k'} e^{-\alpha(\|h_i\|^2 - 2u_{k'} h_i + \|u_{k'}\|^2)}} \tag{9}$$

$$= \frac{e^{w_k^T h_i + b_k}}{\sum_{k'} e^{w_{k'}^T h_i + b_{k'}}} \tag{10}$$

where $2\alpha u_k$ and $-\alpha \|u_k\|^2$ is renamed as w_k and b_k. After combining equation (10) with (7), the final form of our trainable VLAD is:

$$V(k,j) = \sum_{i=1}^{N} \frac{e^{w_k^T h_i + b_k}}{\sum_{k'} e^{w_{k'}^T h_i + b_{k'}}} (h_i(j) - u_k(j)) \tag{11}$$

Note that $\{w_k\}$, $\{b_k\}$ and $\{u_k\}$ are all trainable parameters correspond to cluster center k. Different from the original VLAD which has only one set of parameters $\{u_k\}$, this trainable VLAD method has three independent sets of parameters and can be updated by gradients in an end-to-end manner, which enables a more effective and flexible representation of the environment.

Implement of VLAD Processing Core. Basing on the derivations mentioned above, we designed VLAD Processing Core. The key is how to implement equation (11) with basic calculation modules like linear network or CNN kernel. The detailed structure of VLAD Processing Core is illustrated in the Figure.3.

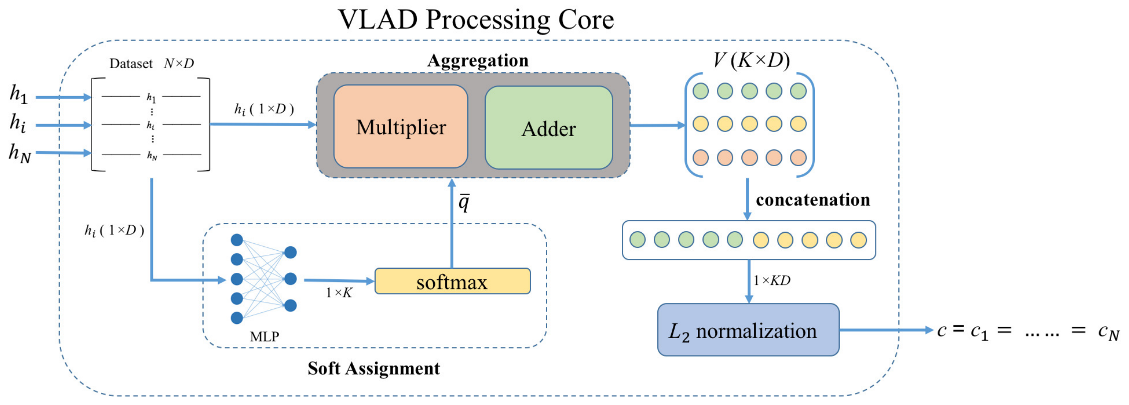

Figure 3: Inner architecutre of VLAD Processing Core. VLAD Processing Core can be easily implemented by basic calculation modules (perceptron, adder, multiplier and L_2-normalization). With hidden states of all agents as inputs, VLAD Processing Core can generate a clustered communication message to be broadcast among the agents.

First, all the input descriptors are stacked together to form a dataset $\mathcal{D} = \{h_1, \ldots, h_N\}$, and pick descriptor one at a time. **Second**, the soft assignment function $\bar{q}_k(h_i)$ can be viewed as linear mappings of the descriptor followed by a softmax function. These linear mapping operations can be achieved by a two-layer perceptron which has D input units (correspond to the dimension of hidden states) and K output units with no non-linear activation function. Alternatively, if we stack all the descriptors together (to form a $N \times D$ matrix) and regard them as an image, then the linear mapping operation can be implemented by K convolution filters whose size is $1 \times D$, the vertical stride of these filters is 1 and 0 for horizontal stride. The outputs are then passed through a softmax function $f_{softmax}$ to get the final result of $\bar{q}_k(h_i)$. **Third**, multiply soft assignment weights with the difference vectors and aggregate them together for every cluster center k. This operation can be realised by simple multipliers and adders. **Finally**, these K aggregated vectors are concatenated together and a normalizing operation is applied. The output vector is then directly used as a clustered communication message and is broadcast among all agents.

Since the input and output sizes of the calculation modules are independent of the number of descriptors N, VLAD Processing Core is more suitable to deal with the tasks in which N changes dynamically. Additionally, VLAD Processing Core can be represented by a directed acyclic graph and is ready to be inserted into other models as an “end-to-end trainable clustering module” for some specific tasks.

5. Experiments

We tested our framework on two multi-agent tasks (named Traffic Junction and Combat Task) which are based on the MazeBase environment (Sukhbaatar et al., 2015). A modified baseline subtracting REINFORCE algorithm (described in the background section) is adapted to train our framework. Additionally, an on-policy episodic training method is utilized instead of experience replay. Assuming the state sequence in an episode is $s^1, \ldots, s^T$, and the corresponding actions are $a^1, \ldots, a^T$, in which T denotes the length of this game episode. Similar to Wang et al. (2015), an extra head is designed to generate a scalar value $b(s^t)$ to represent how “good” this state is and is used as baseline during training. The model parameters θ can be optimized by adding:

$$\Delta\theta = \sum_{t=1}^{T} \left[\frac{\partial \text{log} p(a^t|s^t,\theta)}{\partial\theta} (\sum_{i=t}^{T} r^i - b(s^t,\theta)) - \beta \frac{\partial}{\partial\theta} (\sum_{i=t}^{T} r^i - b(s^t,\theta))^2 \right] \tag{12}$$

where r^i is the reward received from the environment at timestep i, the hyperparameter β is set to 0.03 in all experiments.

As for network structure, the output size of encoder is set to 50 for Traffic Junction and 25 for Combat Task, the number of cluster centers is 4, so the input size of decoder is 250/125 respectively. Framework is trained for 600 epochs on each task, each epoch includes 100 weight updates with RMSProp, the mini-batch of each update is 288 game episodes. Since the variance of the learned policy is also an important part of the evaluations of RL algorithms, we repeated CCNet’s experiments for 3 times with different initializations, and the results are reported together with their mean values and standard deviations.

Like CommNet, we also compared our results with the following baseline models. Note that all the baseline models as well as our model are trained under the identical RL algorithm and with the same hyperparameters (like the number of epochs and learning rate). In other words, the only difference is network architecture.

Independent controller. The communication between agents is disabled and each agent chooses its action only based on its observation. The parameters are still shared among the agents of same type, but agents often act differently since they have different observations.

Fully-connected. All agents are modeled together by a four-layer fully-connected neural network with the size of hidden layers is set to 50. Model's input is the concatenation of the observations of all agents and directly outputs the distributions over actions of all agents. This model allows agents to communicate, but agents may get drown in massive communication information and the model itself is not flexible.

5.1. Traffic Junction

As shown in Figure. 4, a 4-way crossroad is simulated in a 14 × 14 grid. At each timestep, new cars enter the grid from one of the four directions with a certain probability p_{arrive}, and will exit the grid as soon as they reach one of the edges of the grid. The maximum number of cars running on the roads at any timestep is set to 10 in our experiments. Each car's destination is randomly chosen from three possible edges the moment it enters the grid, and cars should keep driving on the right-side of the roads. Each car has two optional actions: *gassing* to move forward by one cell on its route or *breaking* to stay at its current location.

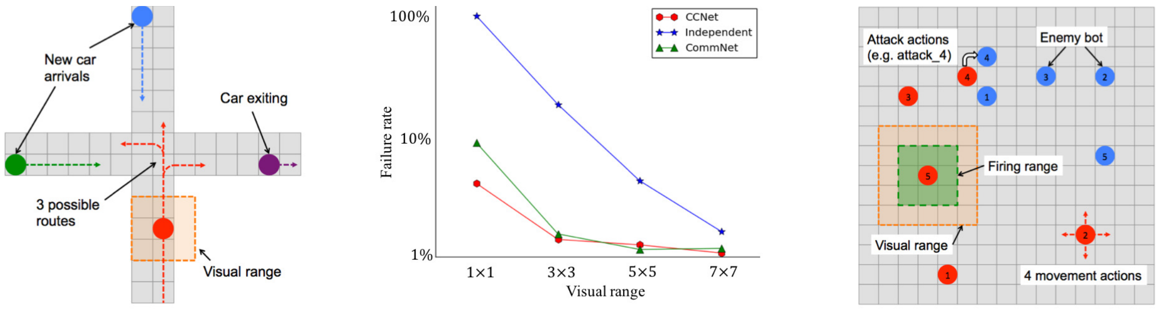

Figure 4: **Left**: Traffic Junction task in which each colored circle corresponds to a running car (agent). Agents have to pass the crossroad safely as soon as possible to get higher rewards. **Middle**: The failure rates of different models with different visual ranges after 300 episodes' training. Results show that CCNet outperforms other models especially when visual range is small. **Right**: Combat task, in which the trained model needs to control a group of bots (red circles) to defeat the enemy (blue circles), bots of two sides are equal in number and strength. Agents in both tasks have a visual range (orange region) which leads to a partially observable environment.

Two cars are considered as a collision if they stand on the same cell and this game episode is classified as a failure, but the simulation will continue until it reaches the termination timestep (40 in our experiments). The reward which agents received at timestep t is defined as:

$$r^t = C^t r_{coll} + \sum_{i=1}^{N^t} (-0.01)\tau_i \tag{13}$$

here C^t denotes the number of collisions occurred at time t, $r_{coll} = -10$ is the reward when a collision happened, N^t is the number of cars currently on roads and τ_i is the total timesteps passed since the car i arrived (this term is used to discourage traffic jam).

We consider each car to be an individual agent which is denoted by its ID number n, current location l, route number r and coded by an one-hot binary vector. Each car has a vision range (a $m \times m$ neighborhood) and can only observe the cars within this range. Then the observation of each car o_i is a $m^2 \times |n| \times |l| \times |r|$ dimensional vector.

We mainly evaluated the performances of different models under zero visibility situations (the visual range is 1×1, agent can see nothing but itself) and utilized the failure rate of one epoch (approximately 30000 episodes) as evaluation standard. The results are shown in Table. 1 (The results of Discrete comm, CommNet and A-CCNet-sha are cited from Sukhbaatar et al. (2016); Mao et al. (2017) directly).

	300 episodes	600 episodes
Independent	100	-
Discrete comm	100	-
Fully-connectd	47.48	-
CommNet	10	-
A-CCNet-sha	$7.88_{\pm ?}$	$4.96_{\pm ?}$
CCNet	$\mathbf{6.79_{\pm 1.25}}$	$\mathbf{5.74_{\pm 1.06}}$

Table 1: Failure rates(%) of different models under zero visibility. **Note that** A-CCNet-sha does not report its results in the form of mean value and standard deviation, which is not so convincing especially in RL domains (RL algorithms often have high variance problem). Our best result of 600 episodes is **4.29%** and is better than the reported A-CCNet-sha result.

Due to the limits of visibility, agents need to learn a communication protocol which can reflect the status of all agents to help avoid collisions. Not surprisingly, CCNet achieves the best results when compared to other proposed methods and keeps the robustness of agent number at the same time, showing the effectiveness of the communication protocols learned by our framework. We also explored the performance with different visual ranges (shown in Figure. 4). Our CCNet still presents comparable results with CommNet when visual range is large and the failure rates of both models are less than 2%. These results also reveal the fact that the larger the visual range is, the smaller the effect of communication will be.

5.2. Combat Task

Combat Task simulates a 15×15 grid battle field in which two teams of bots fight against each other (shown in Figure. 4). Each side has 5 bots (agents) and these bots are generated uniformly in a 5×5 region whose center is also picked uniformly in the whole battle field. Each bot has a couple of actions: four *moving* actions each corresponds to one of the four directions; *attacking* an enemy bot by specifying its ID j; or *staying* at the current location (there are $4 + 5 + 1 = 10$ actions in total). Except for visual ranges (5×5 area) like Traffic Junction, each bots has a firing range (3×3 area) and attacking the enemies within firing range is considered as a valid attack. Each agent has 3 health points when game starts and each valid attack will injure the target enemy bot by 1 health point. One timestep is needed for cooling down after attack, during which that agent cannot attack again. Bot dies when its health point reaches 0 and is then removed from the battle field.

Reaching the maximum length of one game episode ($T_{max} = 40$) is classified as a draw, and killing all the enemies is classified as a win. The RL model only controls one side of bots, while the another team is controlled by an environment build-in controller which follows a simple policy: attack the nearest enemy within firing range or get close to the nearest enemy within visual range.

Like Traffic Junction, the agent's observation o_i is one-hot coded and the information of one agent includes ID number n, team ID t, health point h, cool down c and location l. Then the dimension of o_i is $5^2 \times |n| \times |t| \times |h| \times |c| \times |l|$. A reward is designed to encourage agents to attack and defeat enemies: $-0.1 \times h_{enemy}$ for all $t < T$, and -1 if loses or draws at timestep T, where h_{enemy} is the sum of health points of the enemies.

	300 episodes	600 episodes
Independent	$34.2_{\pm 1.3}$	-
Discrete comm	$29.1_{\pm 6.7}$	-
Fully-conneced	$17.7_{\pm 7.1}$	-
CommNet	$44.5_{\pm 13.4}$	-
CCNet	$\mathbf{52.9_{\pm 0.36}}$	$\mathbf{58.9_{\pm 0.9}}$

Table 2: Win rates(%) of different models. CCNet outperforms other methods by a large margin with the highest win rate and the lowest standard deviation.

Table. 2 demonstrates the win rates (%) of different models. Compared to the Traffic Junction task, Combat Task is more complicate because agents need to learn how to exchange the information about the spotted enemies and the communication messages need to carry more information. The huge improvement on Combat Task shows that CCNet is useful in communication protocol learning and information processing, especially in complicate partially observable situations.

6. Conclusion

This work addressed the "Learning-to-communicate" problem among multiple agents with reinforcement learning algorithm. We have proposed a new cluster-coordinated network (CCNet), a VLAD based end-to-end trainable framework which can learn the communication protocols even from scratch under partially observable environments and outperforms the state-of-the-art results on two challenging tasks. By taking advantages of the trainable

VLAD clustering algorithm, the learned communication protocols have strong representation ability and can effectively handle the tasks with dynamic variation in the total number of agents. The CCNet, especially VLAD Processing Core, can also be embedded into other networks (like RNN, LSTM, CNN) to solve different multi-agent tasks (so long as the communication is enabled).

Acknowledgments

This work was supported by the National Key R&D Program of China under Grant 2017YFB1300201.

References

Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5297–5307, 2016.

Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008.

AI DeepMind. Reduces google data centre cooling bill by 40%, 2016.

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017a.

Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017b.

Dieter Fox, Wolfram Burgard, Hannes Kruppa, and Sebastian Thrun. A probabilistic approach to collaborative multi-robot localization. *Autonomous robots*, 8(3):325–344, 2000.

Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

Yunchao Gong, Liwei Wang, Ruiqi Guo, and Svetlana Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *European conference on computer vision*, pages 392–407. Springer, 2014.

Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010.

Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 656–671. Springer, 2008.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pages 246–253. ACM, 2001.

Hangyu Mao, Zhibo Gong, Yan Ni, and Zhen Xiao. Accnet: Actor-coordinator-critic net for" learning-to-communicate" with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1706.03235*, 2017.

Darío Maravall, Javier de Lope, and Raúl Domínguez. Coordination of communication in robot teams by reinforcement learning. *Robotics and Autonomous Systems*, 61(7): 661–666, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Peng Peng, Ying Wen, Yaodong Yang, Quan Yuan, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games. *arXiv preprint arXiv:1703.10069*, 2017.

Julien Perolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In *Advances in Neural Information Processing Systems*, pages 3643–3652, 2017.

Jurgen Schmidhuber. A general method for multi-agent reinforcement learning in unrestricted environments. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pages 84–87, 1996.

Kun Shao, Yuanheng Zhu, and Dongbin Zhao. Starcraft micromanagement with reinforcement learning and curriculum transfer learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017b.

Sainbayar Sukhbaatar, Arthur Szlam, Gabriel Synnaeve, Soumith Chintala, and Rob Fergus. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)*, 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards.* PhD thesis, King's College, Cambridge, 1989.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Dayong Ye, Minjie Zhang, and Yun Yang. A multi-agent framework for packet routing in wireless sensor networks. *sensors*, 15(5):10026–10047, 2015.