A ORGANIZATION OF THE APPENDIX

In appendix **B**, we give the details on the intention propagation network and parameterization of the GNN. We explain intention propgation from the view of the MARL. At last, we extend the intention propagation to other approximations which converges to other solutions of the variational inference. Notice such extension on the algorithm can also be easily parameterized by neural networks.

In Appendix **C**, we give the details of the algorithm deferred from the main paper. Appendix **D** summarizes the configuration of the experiment and MARL environment. Appendix **E** gives more details on baselines and the hyperparameters of GNN used in our model. Appendix **F** conducts the ablation study deferred from the main paper. Appendix **G** and **H** give more experimental results and hyperparameters used in the algorithms. At appendix I, we derive the algorithm and prove the proposition 1.

B INTENTION PROPAGATION NETWORK

B.1 DETAILS ON THE INTENTION PROPAGATION NETWORK

In this section, we give the details on the intention propagation network deferred from the main paper. We first illustrate the message passing of the intention propagation derived in section 4.1. Then we give a details on how to construct graph neural network.

Message passing and explanation from the view of MARL: $\tilde{\mu}_i$ is the embedding of *policy* of agent i, which represents the intention of the agent i. At 0 iteration, every agent makes independent decision. The policy of agent i is mapped into its embedding $\tilde{\mu}_i^0$. We call it the intention of agent i at iteration 0. Then agent i sends its plan to its neighbors . In Figure 5, $\tilde{\mu}_i^m$ is the d dimensional ($d = 3$ in this figure) embedding of q_i at $m-$th iteration of intention propagation. We draw the update of $\tilde{\mu}_1^{(m)}$ as example. Agent 1 receives the embedding (intention) $\tilde{\mu}_2^{m-1}, \tilde{\mu}_5^{m-1}, \tilde{\mu}_6^{m-1}$ from its neighbors, and then updates the its own embedding with operator $\tilde{\mathcal{T}}$. After M iterations, we obtain $\tilde{\mu}_1^M$ and output the policy distribution q_1 using equation 4. Similar procedure holds for other agents. At each RL step t, we do this procedure (with M iterations) once to generate joint policy. M in general is small, e.g., $M = 2$ or 3. Thus it is efficient.

Parameterization on GNN: We then illustrate the parameterization of graph neural network in Figure 6. If the action space is discrete, the output $q_i(a_i|s)$ is a softmax function. When it is continuous, we can output a Gaussian distribution (mean and variance) with the reparametrization trick (Kingma & Welling, 2019). Here, we draw 2-hop (layer) GNN to parameterize it in discrete action intention propagation. In Figure 6 (b), each agent observe its own state s_i. After a MLP and softmax layer (we do not sample here, and just use the output probabilities of the actions), we get a embedding $\tilde{\mu}_i^0$, which is the initial distribution of the policy. In the following, we use agent 1 as an example. To ease the exposition, we assume Agent 1 just has one neighbor, agent 2. Agent 1 receives the embedding $\tilde{\mu}_2^0$ of its neighbor. After a GNN layer to combine the information, e.g, $Relu[W_1(s_1 + s_2) + W_2(\tilde{\mu}_1^0 + \tilde{\mu}_2^0)]$, we obtain new embedding $\tilde{\mu}_1^1$ of agent 1. Notice we also do

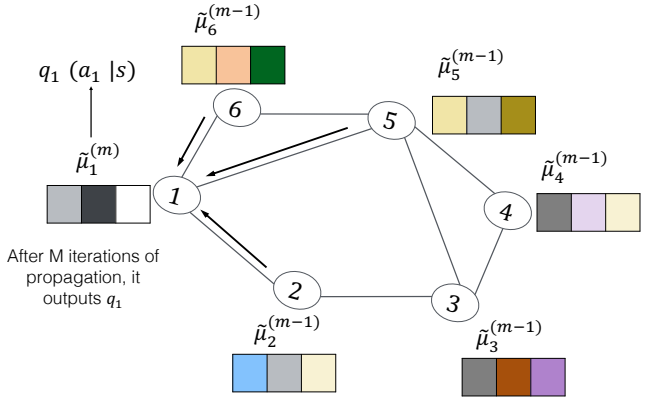


Figure 5: illustrate the message passing in intention propagation network $\Lambda_\theta(\mathbf{a}|s)$.

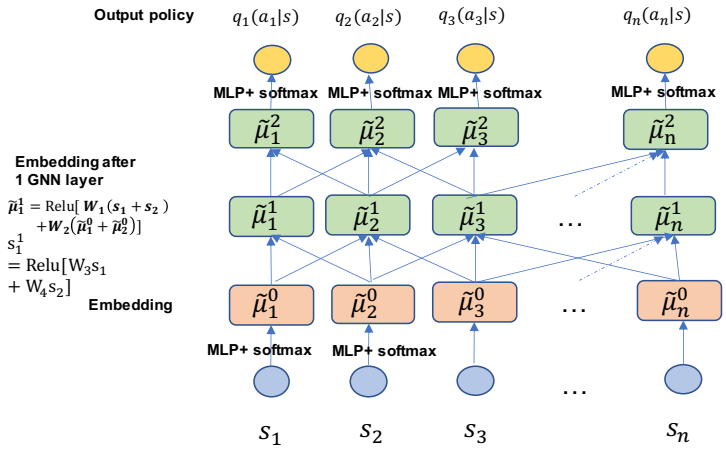


Figure 6: Details of the graph neural network

message passing on state, since in practice the global state is not available. In the second layer, we do similar things. Agent 1 receives the embedding information of $\tilde{\mu}_2^1$ from its neighbors and get a new embedding $\tilde{\mu}_1^2$. Then this embedding passes a MLP+softmax layer and output probability of action, i.e. $q_1(a_1|s)$.

B.2 Extension to other variational inference methods and Neural networks

In this section, we show how to approximate the joint policy with the Loopy Belief Propagation in the variational inference (Yedidia et al., 2001). This will lead to a new form of neural networks beyond vanilla GNN that we illustrate above.

The objective function in Loop Belief Propagation is the Beth Free energy (Yedidia et al., 2001). Different from the mean-field approximation, it introduces another variational variable q_{ij}, which brings more flexibility on the approximation. The following is objective function in our case.

$$\begin{aligned}\min_{q_i, q_{ij} \in \mathcal{E}} & -\sum_i (|\mathcal{N}_i| - 1) \int q_i(a_i|s) \log \frac{q_i(a_i|s)}{\psi_i(s, a_i)} da_i \\ & + \sum_{ij} \int q_{ij}(a_i, a_j|s) \log \frac{q_{ij}(a_i, a_j|s)}{\psi_{ij}(s, a_i, a_j)\psi_i(s, a_i)\psi_j(s, a_j)} da_i da_j. \quad (6) \\ & s.t. \int q_{ij}(a_i, a_j|s) da_j = q_i(a_j|s), \int q_{ij}(a_i, a_j|s) da_i = q_j(a_j|s)\end{aligned}$$

Solve above problem, we have the fixed point algorithm

$$\begin{aligned} m_{ij}(a_j|s) &\leftarrow \int \prod_{k \in \mathcal{N}_i \setminus j} m_{ki}(a_i|s)\psi_i(s, a_i)\psi_{ij}(s, a_i, a_j) da_i, \\ q_i(a_i|s) &\leftarrow \psi_i(s, a_i) \prod_{j \in \mathcal{N}_i} m_{ji}(a_i|s). \end{aligned}$$

Similar to the mean-field approximation case, we have

$$\begin{aligned} m_{ij}(a_j|s) &= f(a_j, s, \{m_{ki}\}_{k \in \mathcal{N}_i \setminus j}), \\ q_i(a_i|s) &= g(a_i, s, \{m_{ki}\}_{k \in \mathcal{N}_i}), \end{aligned}$$

It says the message m_{ij} and marginals q_i are functionals of messages from neighbors.

Denote the embedding $\tilde{\nu}_{ij} = \int \psi_j(s, a_j) m_{ij}(a_j|s) da_j$ and $\tilde{\mu}_i = \int \psi_i(s, a_i) q_i(a_i|s) da_i$, we have $\tilde{\nu}_{ij} = \tilde{\mathcal{T}} \circ \left(s, \{\tilde{\nu}_{ki}\}_{k \in \mathcal{N}_i \setminus j}\right)$, $\tilde{\mu}_i = \tilde{T} \circ \left(s, \{\tilde{\nu}_{ki}\}_{k \in \mathcal{N}_i}\right)$.

Again, we can parameterize above equation by (graph) neural network $\tilde{\nu}_{ij} = \sigma\big(W_1 s + W_2 \sum_{k\in\mathcal{N}_i\setminus j} \tilde{\nu}_{ki}\big)$, $\tilde{\mu}_i = \sigma\big(W_3 s + W_4 \sum_{k\in\mathcal{N}_i} \tilde{\nu}_{ki}\big)$.

Following similar way, we can derive different intention propagation algorithms by changing different objective function which corresponds to e.g., double-loop belief propagation(Yuille, 2002), tree-reweighted belief propagation (Wainwright et al., 2003) and many others.

C ALGORITHM

We present the algorithm of Intention Propagation (algorithm 1) deferred from the main paper.

Algorithm 1 Intention Propagation

Inputs: Replay buffer D. V_i, Q_i for each agent i. Intention propagation network $\Lambda_\theta(\mathbf{a}_t|s)$ with outputs $\{q_{i,\theta}\}_{i=1}^N$. Learning rate $l_\eta, l_\kappa, l_\theta$. Moving average parameter τ for the target network
for each iteration **do**
  for each environment step **do**
    sample $\mathbf{a_t} \sim \prod q_{i,\theta}(a_i^t|s^t)$ from the intention propagation network. $s^{t+1} \sim p(s^{t+1}|s^t, \mathbf{a^t})$, $D \leftarrow D \bigcup \big(s_i^t, a_i^t, r_i^t, s_i^{t+1}\big)_{i=1}^N$
  end for
  for each gradient step **do**
    update η_i, κ_i, θ, $\bar{\eta}_i$.

$$\eta_i \leftarrow \eta_i - l_\eta \nabla J(\eta_i), \kappa_i \leftarrow \kappa_i - l_\kappa \nabla J(\kappa_i)$$
$$\theta \leftarrow \theta - l_\theta \nabla J(\theta), \bar{\eta}_i \leftarrow \tau\eta_i + (1-\tau)\bar{\eta}_i$$

  end for
end for

Remark: To calculate the loss function $J(\eta_i)$, each agent need to sample the global state and $(a_i, a_{\mathcal{N}_i})$. Thus we first sample a global state from the replay buffer and then sample all action $\mathbf{a}$ once using the intention propagation network.

D FURTHER DETAILS ABOUT ENVIRONMENTS AND EXPERIMETAL SETTING

Table 1 summarizes the setting of the tasks in our experiment.

Table 1: Tasks. We evaluate MARL algorithms on more than 10 different tasks from three different environments.

Env	Scenarios	#agents (N)
CityFlow	Realworld:Hang Zhou	N=16
	Realworld:Manhattan	N=96
	Synthetic Map	N=49, 100, 225, 1225
MPE	Cooperative Nav.	N=15, 30, 200
	Heterogeneous Nav.	N=100
	Cooperative Push	N=100
	Prey and Predator	N=100
MAgent	Jungle	N=20, F=12

D.1 CITYFLOW

CityFlow (Tang et al., 2019) is an open-source MARL environment for large-scale city traffic signal control [1]. After the traffic road map and flow data being fed into the simulators, each vehicle moves from its origin location to the destination. The traffic data contains bidirectional and dynamic flows with turning traffic. We evaluate different methods on both real-world and synthetic traffic data. For real-world data, we select traffic flow data from Gudang sub-district, Hangzhou, China and

[1] https://github.com/cityflow-project/CityFlow

Manhattan, USA [2]. For synthetic data, we simulate several different road networks: 7×7 grid network ($N = 49$) and large-scale grid networks with $N = 10 \times 10 = 100$, $15 \times 15 = 225$, $35 \times 35 = 1225$. Each traffic light at the intersection is the agent. In the real-world setting (Hang Zhou, Manhattan), the graph is a 2-d grid induced by the roadmap. Particularly, the roads are edges which connect the node (agent) of the graph. For the synthetic data, the map is a $n * n$ 2-d grid (Something like Figure 7), where edges represents road, node is the traffic light. We present the experimental results deferred from the main paper in Figure 10.

D.2 MPE

In MPE (Mordatch & Abbeel, 2017) [3], the observation of each agent contains relative location and velocity of neighboring agents and landmarks. The number of visible neighbors in an agent's observation is equal to or less than 10. In some scenarios, the observation may contain relative location and velocity of neighboring agents and landmarks.

We consider four scenarios in MPE. (1) *cooperative navigation*: N agents work together and move to cover L landmarks. If these agents get closer to landmarks, they will obtain a larger reward. In this scenario, the agent observes its location and velocity, and the relative location of the nearest 5 landmarks and N agents. The observation dimension is 26. (2) *prey and predator*: N slower cooperating agents must chase the faster adversaries around a randomly generated environment with L large landmarks. Note that, the landmarks impede the way of all agents and adversaries. This property makes the scenario much more challenging. In this scenario, the agent observes its location and velocity, and the relative location of the nearest 5 landmarks and 5 preys. The observation dimension is 34. (3) *cooperative push*: N cooperating agents are rewarded to push a large ball to a landmark. In this scenario, each agent can observe 10 nearest agents and 5 nearest landmarks. The observation dimension is 28. (4) *heterogeneous navigation*: this scenario is similar with cooperative navigation except dividing N agents into $\frac{N}{2}$ big and slow agents and $\frac{N}{2}$ small and fast agents. If small agents collide with big agents, they will obtain a large negative reward. In this scenario, each agent can observe 10 nearest agents and 5 nearest landmarks. The observation dimension is 26.

Further details about this environment can be found at `https://github.com/IouJenLiu/PIC`.

D.3 MAGENT

MAgent (Zheng et al., 2018) is a grid-world platform and serves another popular environment platform for evaluating MARL algorithms. Jiang et al. (2020) tested their method on two scenarios: *jungle* and *battle*. In *jungle*, there are N agents and F foods. The agents are rewarded by positive reward if they eat food, but gets higher reward if they attack other agents. This is an interesting scenario, which is called by *moral dilemma*. In *battle*, N agents learn to fight against several enemies, which is very similar with the *prey and predator* scenario in MPE. In our experiment, we evaluate our methods on *jungle*.

In our experiment, the size for the grid-world environment is 30×30. Each agent refers to one grid and can observe 11×11 grids centered at the agent and its own coordinates. The actions includes moving and attacking along the coordinates. Further details about this environment can be found at `https://github.com/geek-ai/MAgent` and `https://github.com/PKU-AI-Edge/DGN`.

E FURTHER DETAILS ON SETTINGS

E.1 DESCRIPTION OF OUR BASELINES

We compare our method with multi-agent deep deterministic policy gradient (MADDPG) (Lowe et al., 2017), a strong actor-critic algorithm based on the framework of centralized training with

[2]We download the maps from `https://github.com/traffic-signal-control/sample-code`.

[3]To make the environment more computation-efficient, Liu et al. (2019) provided an improved version of MPE. The code are released in `https://github.com/IouJenLiu/PIC`.

decentralized execution; QMIX (Rashid et al., 2018), a q-learning based monotonic value function factorisation algorithm; permutation invariant critic (PIC) (Liu et al., 2019), a leading algorithm on MPE yielding identical output irrespective of the agent permutation; graph convolutional reinforcement learning (DGN) (Jiang et al., 2020), a deep q-learning algorithm based on deep convolutional graph neural network with multi-head attention, which is a leading algorithm on MAgent; independent Q-learning (IQL) (Tan, 1993), decomposing a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment, which usually serves as a surprisingly strong benchmark in the mixed and competitive games (Tampuu et al., 2017). In homogeneous settings, the input to the centralized critic in MADDPG is the concatenation of all agent's observations and actions along the specified agent order, which doesn't hold the property of *permutation invariance*. We follow the similar setting in (Liu et al., 2019) and shuffle the agents' observations and actions in training batch [4]. In COMA (Foerster et al., 2018), it directly assume the poilcy is factorized. It calculates the counterfactual baseline to address the credit assignment problem in MARL. In our experiment, since we can observe each reward function, each agent can directly approximate the Q function without counterfactual baseline. MFQ derives the algorithm from the view of mean-field game(Yang et al., 2018). Notice the aim of mean-field game is to find the Nash equilibrium rather than maxmization of the total reward of the group. Further more, it needs the assumption that agents are identical.

E.2 Neural Networks Architecture

To learn feature from structural graph build by the space distance for different agents, we design our graph neural network based on the idea of a strong graph embedding tool *structure2vec* (Dai et al., 2016), which is an effective and scalable approach for structured data representation through embedding latent variable models into feature spaces. Structure2vec extracts features by performing a sequence of function mappings in a way similar to graphical model inference procedures, such as mean field and belief propagation. After using M graph neural network layers, each node can receive the information from M-hops neighbors by message passing. Recently, attention mechanism empirically leads to more powerful representation on graph data (Veličković et al., 2017; Jiang et al., 2020). We employ this idea into our graph neural network. In some settings, such as *heterogeneous navigation* scenario from MPE, the observations of different group of agents are heterogeneous. To handle this issue, we use different nonlinear functions to extract the features from heterogeneous observations and map the observations into a latent layer, then use the same graph neural networks to learn the policy for all types of agents. In our experiment, our graph neural network has $M = 2$ layers and 1 fully-connected layer at the top. Each layer contains 128 hidden units.

F Ablation Studies

F.1 Independent policy vs intention propgation.

We first give a toy example where the independent policy (without communication) fails. To implement such algorithm, we just replace the intention propagation network by a independent policy network and remain other parts the same. Think about a 3×3 2d-grid in Figure 7 where the global state (can be observed by all agents) is a constant scalar (thus no information). Each agent chooses an action a_i = 0 or 1. The aim is to maximize a reward $-(a_1 - a_2)^2 - (a_1 - a_4)^2 - (a_2 - a_3)^2 - ... - (a_8 - a_9)^2$, (i.e., summation of the reward function on edges). Obviously the optimal value is 0. The optimal policy for agents is $a_1 = a_2 =, ..., a_9 = 0$ or $a_1 = a_2 =, ..., a_9 = 1$. However independent policy fails, since each agents does not know how its allies pick the action. Thus the learned policy is random. We show the result of this toy example in Figure 7, where intention propagation learns optimal policy.

F.2 Graph types, number of neighbors, and hop size

We conduct a set of ablation studies related to graph types, the number of neighbors, and hop size. Figure 8(a) and Figure 8(b) demonstrate the performance of our method on traffic graph and fully-connected graph on the scenarios (N=49 and N=100) of CityFlow. Figure 8(c) and Figure 8(d)

[4] This operation doesn't change the state of the actions.

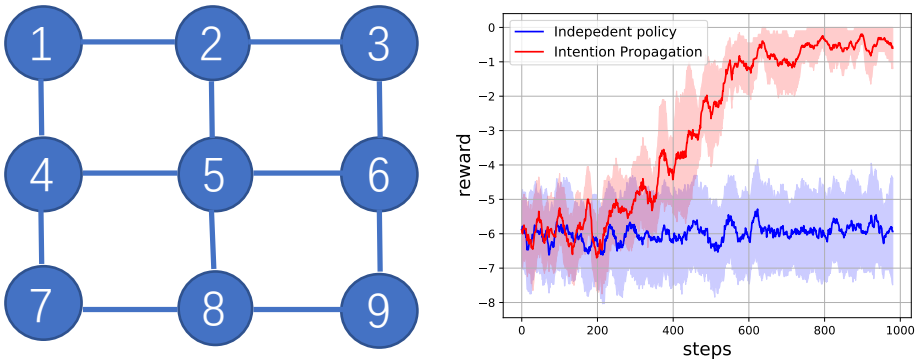


Figure 7: (a) a toy task on 2d-grid. (b) The performance of independent policy and intention propagation.

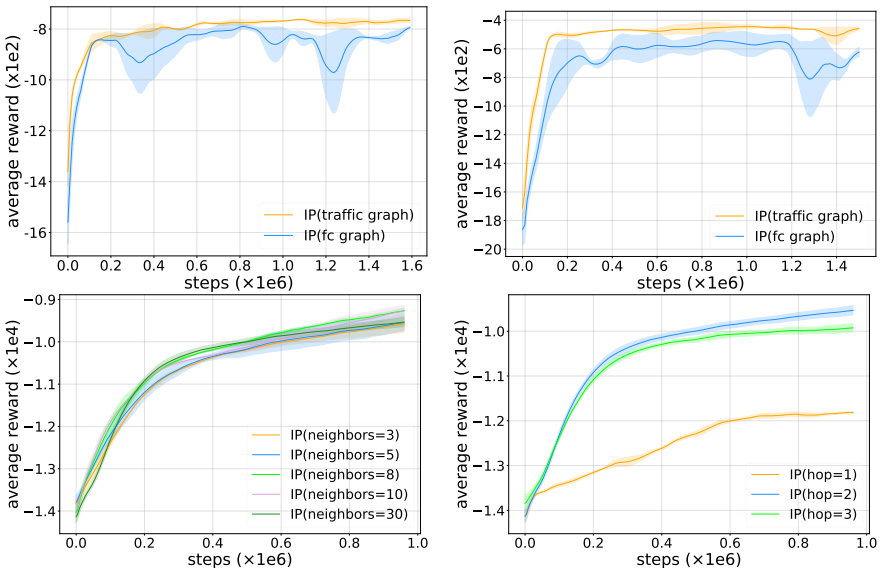
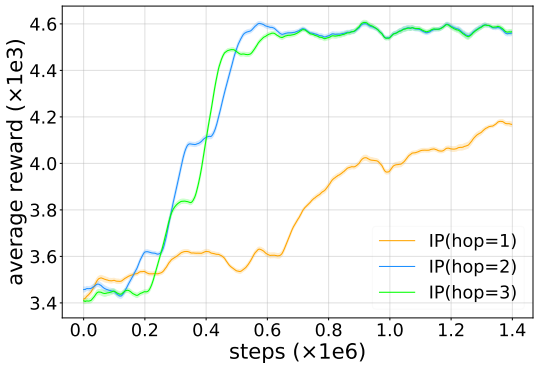


Figure 8: Performance of the proposed method based on different ablation settings. (a) Traffic graph and fully connected (fc) graph on CityFlow (N=49). (b) Traffic graph and fully connected (fc) graph on CityFlow (N=100). (c) Different number of neighbors. (d) Different hop size graph neural networks.

demonstrate the performance under different number of neighbors and hop size on *cooperative navigation* (N=30) respectively.

F.3 Assumption Violation

The aforementioned experimental evaluations are based on the mild assumption: the actions of agents that are far away would not affect the learner because of their physical distance. It would be interesting to see the performance where the assumption is violated. As such, we modify the reward in the experiment of cooperative navigation. In particular, the reward is defined by $r = r1 + r2$, where $r1$ encourages the agents to cover (get close to) landmarks and $r2$ is the log function of the distances between agents (farther agents have larger impact). To make a violation, we let $r2$ dominate the reward. We conduct the experiments with $hop = 1, 2, 3$. Figure 9 shows that the rewards obtained by our methods are 4115 ± 21, 4564 ± 22, and 4586 ± 25 respectively. It's expected in this scenario, since we should use large hop to collect information from the far-away agents.



(a) Coop. Nav. Violation (N=30)

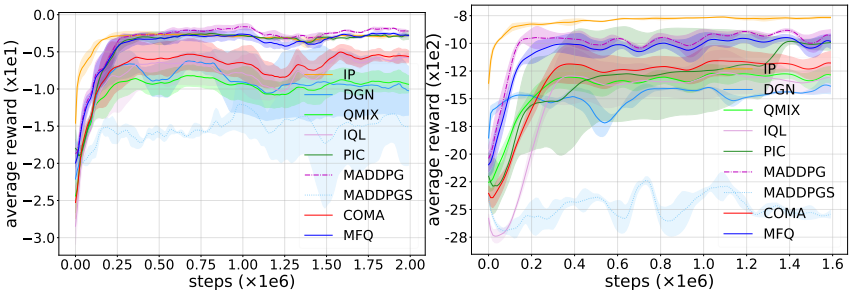
Figure 9: Further experimental results. Cooperative navigation (N=30) with assumption violation.

G FURTHER EXPERIMENTAL RESULTS

For most of the experiments, we run them long enough with 1 million to 1.5 million steps and stop (even in some cases our algorithm does not converge to the asymptotic result), since every experment in MARL may cost several days. We present the results on Cityflow in Figure 10. Figure 11 provides the experimental results on the cooperative navigation instances with $N = 15$, $N = 30$ and $N = 200$ agents. Note that, the instance with $N = 200$ is a large-scale and challenging multi-agents reinforcement learning setting (Chen et al., 2018; Liu et al., 2019), which typically needs several days to run millions of steps. It's clear that IQL, MADDPG, MADDPG perform well in the small setting (N=15), however, they failed in large-scale instances ($N = 30$ and $N = 200$). In the instance with $N = 30$, MADDPGS performs better than MADDPG. The potential reason is that with the help of shuffling, MADDPGS is more robust to handle the manually specified order of agents. Although QMIX performs well in the instance of $N = 15$ and $N = 30$, it has large variances in both settings. DGN using graph convolutional network can hold the property of permutation invariance, it obtains much better performance than QMIX on these two settings. However, it also fails to solve the large-scale settings with $N = 200$ agents. Empirically, after 1.5×10^6 steps, PIC obtains a large reward (-425085 ± 31259) on this large-scale setting. Despite all these, the proposed intention propagation (IP) approaches -329229 ± 14730 and is much better than PIC. Furthermore, Figure 11 shows the results of different methods on (d) jungle (N=20, F=12) and (e) *prey and predator* (N=100). The experimental results shows our method can beats all baselines on these two tasks. On the scenario of *cooperative push* (N=100) as shown in Figure 11(f), it's clear that DGN, QMIX, IQL, MADDPG and MADDPGS all fail to converge to good rewards after 1.5×10^6 environmental steps. In contrast, PIC and the proposed IP method obtain much better rewards than these baselines. Limited by the computational resources, we only show the long-term performance of the best two methods. Figure 11(f) shows that IP is slightly better than PIC in this setting.

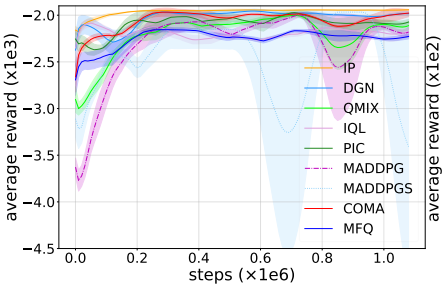
G.1 POLICY INTERPRETATION

Explicitly analyzing the policy learned by deep multi-agent reinforcement learning algorithm is a challenging task, especially for the large-scale problem. We follow the similar ideas from (Zheng et al., 2019) and analyze the learned policy on CityFlow in the following way: We select the same period of environmental steps within [210000, 1600000] and group these steps into 69 intervals (each interval contains about 20000 steps). We compute the ratio of vehicle volume on each movement and the sampled action volume from the learned policy (each movement can be assigned to one action according to the internal function in CityFlow). We define the ratio of vehicle volume over all movements as the *vehicle volume distribution* and define the ratio of the sampled action volume from the learned policy over all movements as the *sampled action distribution*. It's expected that a good MARL algorithm will hold the property: these two distributions will very similar over a period of time. Figure 12 reports their KL divergence by intervals. It's clear that the proposed intention propagation method (IP) obtains the lowest KL divergence (much better than the state-of-the-art baselines). Because KL divergence is not symmetrical metric, we also calculate their Euclidean distances. Specifically, the distance of our method is 0.0271 while DGN is 0.0938 and PIC is 0.0933.

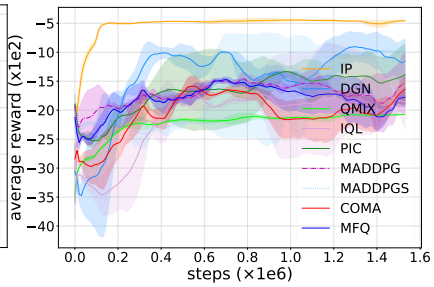


(a) CityFlow:Hang Zhou

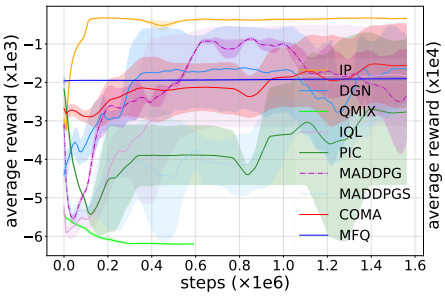
(b) CityFlow:7*7



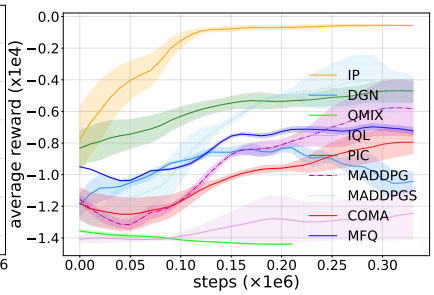
(c) CityFlow: Manhattan, USA.



(d) CityFlow:10*10



(e) CityFlow:15*15



(f) CityFlow:35*35

Figure 10: Performance of different methods on traffic lights control scenarios in CityFlow environment: (a) N=16 (4×4 grid), Gudang sub-district, Hangzhou, China. (b) N=49 (7×7 grid), (c) N=96 (irregular grid map), Manhattan, USA. (d) N=100 (10×10 grid), (e) N=225 (15×15 grid), (f) N=1225 (35×35 grid). The horizontal axis is time steps (interaction with the environment). The vertical axis is average episode reward, which refers to negative average travel time. Higher rewards are better. The proposed intention propagation (IP) obtains much better performance than all the baselines on large-scale tasks .

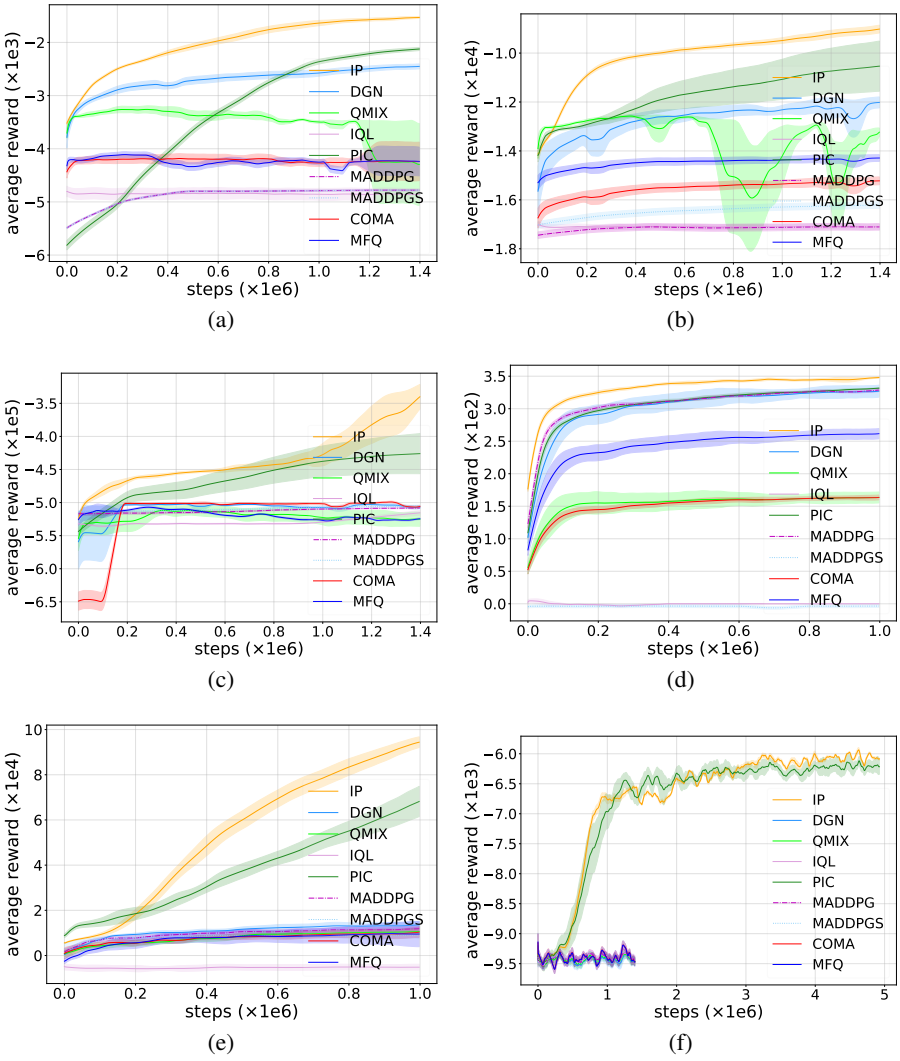


Figure 11: Comparison on cooperative navigation instances with different number of agents: (a)N=15, (b) N=30 and (c) N=200 respectively. (d) *jungle* (N=20, F=12), (e) *prey and predator* (N=100), and (f) *cooperative push* (N=100). The horizontal axis is environmental steps (number of interactions with the enviroment). The vertical axis is average episode reward. The larger average reward indicates better result. The proposed intention propagation (IP) beats all the baselines on different scale of instances.

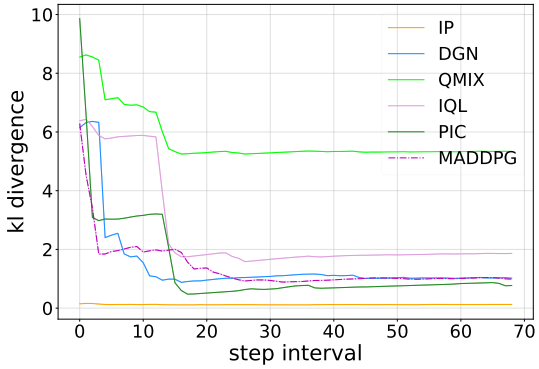


Figure 12: Policy interpreation on CityFlow task (N=49)

H HYPERPARAMETERS

The parameter on the environment. For the max episode length, we follow the similar settings like that in the baselines (Lowe et al., 2017) . Particularly, we set 25 for MPE and set 100 for CityFlow. For MAgent, we find that setting the max episode length by 25 is better than 100. All the methods share the same setting.

We list the range of hyperparameter that we tune in all baselines and intention propagation. γ : $\{0.95, 0.98, 0.99, 0.999\}$, learning rate : $\{1, 5, 10, 100\}\times$1e-4. activation function: $\{relu, gelu, tanh\}$, batch size:$\{128, 256, 512, 1024\}$, gradient steps: $\{1, 2, 4, 8\}$. Number of hidden units in MLP: $\{32, 64, 128, 256, 512\}$, number of layers in MLP:$\{1, 2, 3\}$ in all experiment. In Qmix, GRU hidden unites are $\{64, 128\}$. A fully connected layer is before and after GRU. Hypernetwork and mixing network are both single layer network(64 hidden units with Relu activation from the Qmix paper). The parameter of intention propagation is reported in Table.2.

Table 2: Hyperparameters

Parameter	Value
optimizer	Adam
learning rate of all networks	0.01
discount of reward	0.95
replay buffer size	10^6
max episode length in MPE, MAgent	25
max episode length in CityFlow	100
number of hidden units per layer	128
number of samples per minibatch	1024
nonlinearity	ReLU
target smoothing coefficient (τ)	0.01
target update interval	1
gradient steps	8
regularizer factor(α)	0.2

I DERIVATION

I.1 PROOF OF PROPOSITION 1

We prove the result by induction using the backward view.

To see that, plug $r(s^t, \mathbf{a}^t) = \sum_{i=1}^{N} r_i(s^t, a_i^t, a_{\mathcal{N}_i}^t)$ into the distribution of the optimal policy defined in section 3.

$$p(\tau) = [p(s^0)\prod_{t=0}^{T} p(s^{t+1}|s^t, \mathbf{a^t})] \exp \sum_{t=0}^{T}\sum_{i=1}^{N} r_i(s^t, a_i^t, a_{\mathcal{N}_i}^t)$$

Recall the goal is to find the best approximation of $\pi(\mathbf{a}^t|s^t)$ such that the trajectory distribution $\hat{p}(\tau)$ induced by this policy can match the optimal trajectory probability $p(\tau)$. Thus we minimize the KL divergence between them $\min_\pi D_{KL}(\hat{p}(\tau)||p(\tau))$, where $\hat{p}(\tau) = p(s^0)\prod_{t=0}^{T} p(s^{t+1}|s^t, \mathbf{a^t})\pi(\mathbf{a^t}|s^t)$. We can do optimization w.r.t. $\pi(a^t|s^t)$ as that in (Levine, 2018) and obtain a backward algorithm on the policy $\pi^*(\mathbf{a}^t|s^t)$ (See equation 13 in I.2.)

$$\pi^*(\mathbf{a^t}|s^t) = \frac{1}{Z}\exp\big(\mathbb{E}_{p(s^{t+1:T}, \mathbf{a}^{t+1:T}|s^t, \mathbf{a}^t)}[\sum_{t'=t}^{T}\sum_{i=1}^{N} r_i(s^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^{t'}) - \sum_{t'=t+1}^{T}\log\pi(\mathbf{a^{t'}}|s^{t'})]\big). \quad (7)$$

Using the result equation 7, when $t = T$, the optimal policy is

$$\pi^*(\mathbf{a}^T|s^T) = \frac{1}{Z}\exp(\sum_{i=1}^{N} r_i(s^T, a_i^T, a_{\mathcal{N}_i}^T)).$$

Obviously, it satisfies the form $\pi^*(\mathbf{a}^T|s^T) = \frac{1}{Z}\exp(\sum_{i=1}^N \psi_i(s^T, a_i^T, a_{\mathcal{N}_i}^T))$.

Now suppose from step $t+1$ to T, we have

$$\pi^*(\mathbf{a}^{t'}|s^{t'}) = \frac{1}{Z}\exp(\sum_{i=1}^N \psi_i(s^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^{t'})) \tag{8}$$

for $t' = t+1, ..., T$.

Recall that we have the result

$$\pi^*(\mathbf{a}^{\mathbf{t}}|s^t) = \frac{1}{Z}\exp\big(\mathbb{E}_{p(s^{t+1:T}, \mathbf{a}^{t+1:T}|s^t, \mathbf{a}^t)}[\sum_{t'=t}^T \sum_{i=1}^N r_i(s^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^t) - \sum_{t'=t+1}^T \log \pi^*(\mathbf{a}^{\mathbf{t}'}|s^{t'})]\big). \tag{9}$$

Now plug equation 8 into equation 9, we have

$$\pi^*(\mathbf{a}^{\mathbf{t}}|s^t) = \frac{1}{Z}\exp\big(\mathbb{E}_{p(s^{t+1:T}, \mathbf{a}^{t+1:T}|s^t, \mathbf{a}^t)}[\sum_{t'=t}^T \sum_{i=1}^N r_i(s^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^{t'}) - \sum_{t'=t+1}^T \sum_{i=1}^N \psi_i(s_i^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^{t'}) + C]\big), \tag{10}$$

where C is some constant related to the normalization term. Thus, we redefine a new term

$$\tilde{\psi}_i(s^t, a^t, a_{\mathcal{N}_i}^t) = \mathbb{E}_{p(s^{t+1:T}, a^{t+1:T}|s^t, a^t)}\big[\sum_{t=t'}^T \big(r_i(s^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^{t'}) - \sum_{t'=t+1}^T \psi_i(s^{t'}, a^{t'}, a_{\mathcal{N}_i}^{t'})\big)\big]. \tag{11}$$

Then obviously $\pi^*(\mathbf{a}^t|s^t)$ satisfies the form what we need by absorbing the constant C into the normalization term . Thus we have the result.

I.2 Derivation of the algorithm

We start the derivation with minimization of the KL divergence $KL(\hat{p}(\tau)||p(\tau))$, where $p(\tau) = [p(s^0)\prod_{t=0}^T p(s^{t+1}|s^t, \mathbf{a}^{\mathbf{t}})]\exp\big(\sum_{t=0}^T \sum_{i=1}^N r_i(s^t, a_i^t, a_{\mathcal{N}_i}^t)\big)$, $\hat{p}(\tau) = p(s^0)\prod_{t=0}^T p(s^{t+1}|s^t, \mathbf{a}^{\mathbf{t}})\pi(\mathbf{a}^{\mathbf{t}}|s^t)$.

$$\begin{aligned} KL(\hat{p}(\tau)||p(\tau)) =& \mathbb{E}_{\tau\sim\hat{p}(\tau)} \sum_{t=0}^T \big(\sum_{i=1}^N r_i(s^t, a_i^t, a_{\mathcal{N}_i}^i) - \log \pi(\mathbf{a}^t|s^t)\big) \\ =& \sum_{\tau}[p(s^0)\prod_{t=0}^T p(s^{t+1}|s^t, \mathbf{a}^{\mathbf{t}})\pi(\mathbf{a}^{\mathbf{t}}|s^t)] \sum_{t=0}^T \big(\sum_{i=1}^N r_i(s^t, a_i^t, a_{\mathcal{N}_i}^t) - \log \pi(\mathbf{a}^t|s^t)\big). \end{aligned} \tag{12}$$

Now we optimize KL divergence w.r.t $\pi(\cdot|s^t)$. Considering the constraint $\sum_j \pi(j|s^t) = 1$, we introduce a Lagrangian multiplier $\lambda(\sum_{j=1}^{|\mathcal{A}|} \pi(j|s^t) - 1)$ (Rigorously speaking, we need to consider another constraint that each element of π is larger than 0, but later we will see the optimal value satisfies this constraint automatically). Now we take gradient of $KL(\hat{p}(\tau)||p(\tau)) + \lambda(\sum_{j=1}^{|\mathcal{A}|} \pi(j|s^t) - 1)$ w.r.t $\pi(\cdot|s)$, set it to zero, and obtain

$$\log \pi^*(\mathbf{a}^t|s^t) = \mathbb{E}_{p(s^{t+1:T}, \mathbf{a}^{t+1:T}|s^t, \mathbf{a}^t)}[\sum_{t'=t}^T \sum_{i=1}^N r_i(s^{t'}, a_i^{t'}, a_{\mathcal{N}_i}^{t'}) - \sum_{t'=t+1}^T \log \pi(\mathbf{a}^{\mathbf{t}'}|s^{t'})] - 1 + \lambda.$$

Therefore

$$\pi^*(\mathbf{a}^t|s^t) \propto \exp\big(\mathbb{E}_{p(s^{t+1:T},\mathbf{a}^{t+1:T}|s^t,\mathbf{a}^t)}[\sum_{t'=t}^{T}\sum_{i=1}^{N} r_i(s^{t'},a_i^{t'},a_{\mathcal{N}_i}^{t'}) - \sum_{t'=t+1}^{T}\log\pi(\mathbf{a}^{\mathbf{t}'}|s^{t'})]\big).$$

Since we know $\sum_j \pi(j|s^t) = 1$, thus we have

$$\pi^*(\mathbf{a}^t|s^t) = \frac{1}{Z}\exp\big(\mathbb{E}_{p(s^{t+1:T},\mathbf{a}^{t+1:T}|s^t,\mathbf{a}^t)}[\sum_{t'=t}^{T}\sum_{i=1}^{N} r_i(s^{t'},a_i^{t'},a_{\mathcal{N}_i}^{t'}) - \sum_{t'=t+1}^{T}\log\pi(\mathbf{a}^{\mathbf{t}'}|s^{t'})]\big). \quad (13)$$

For convenience, we define the soft V function and Q function as that in (Levine, 2018), and will show how to decompose them into V_i and Q_i later.

$$\begin{aligned} V(s^{t+1}) &:= \mathbb{E}\big[\sum_{t'=t+1}^{T}\sum_{i=1}^{N} r_i(s^{t'},a_i^{t'},a_{\mathcal{N}_i}^{t'}) - \log\pi(\mathbf{a}^{\mathbf{t}'}|s^{t'})|s^{t+1}\big], \\ Q(s^t,\mathbf{a}^t) &:= \sum_{i=1}^{N} r_i(s^t,a_i^t,a_{\mathcal{N}_i}^t) + \mathbb{E}_{p(s^{t+1}|s^t,\mathbf{a}^t)}[V(s^{t+1})] \end{aligned} \quad (14)$$

Thus $V(s^t) = E_\pi[Q(s^t,a^t) - \log\pi(\mathbf{a}^{\mathbf{t}}|s^t)]$. The optimal policy $\pi^*(\mathbf{a}^t|s^t) = \frac{\exp(Q(s^t,\mathbf{a}^t)}{\int\exp Q(s^t,\mathbf{a}^t)d\mathbf{a}^t}$ by plugging the definition of Q into equation 13.

Remind in section 4.1, we have approximated the optimal joint policy by the mean field approximation $\prod_{i=1}^{N} q_i(a_i|s)$. We now plug this into the definition of equation 14 and consider the discount factor. Notice it is easy to incorporate the discount factor by defining a absorbing state where each transition have $(1-\gamma)$ probability to go to that state. Thus we have

$$\begin{aligned} V(s^{t+1}) &:= \mathbb{E}\big[\sum_{t'=t+1}^{T}(\sum_{i=1}^{N} r_i(s^{t'},a_i^{t'},a_{\mathcal{N}_i}^{t'}) - \sum_{i=1}^{N}\log q_i(a_i^{t'}|s^{t'}))|s^{t+1}\big], \\ Q(s^t,\mathbf{a}^t) &:= \sum_{i=1}^{N} r_i(s^t,a_i^t,a_{\mathcal{N}_i}^t) + \gamma\mathbb{E}_{p(s^{t+1}|s^t,\mathbf{a}^t)}[V(s^{t+1})]. \end{aligned} \quad (15)$$

Thus we can further decompose V and Q into V_i and Q_i. We define V_i and Q_i in the following way.

$$\begin{aligned} V_i(s^{t+1}) &= \mathbb{E}[\sum_{t'=t+1}^{T}\big(r_i(s^{t'},a_i^{t'},a_{\mathcal{N}_i}^{t'}) - \log q_i(a_i^{t'}|s^{t'})\big)|s^{t+1}], \\ Q_i(s^t,a_i^t,a_{\mathcal{N}_i}^t) &= r_i(s^t,a_i^t,a_{\mathcal{N}_i}^t) + \gamma\mathbb{E}_{p(s^{t+1}|s^t,\mathbf{a}^t)}[V_i(s^{t+1})]. \end{aligned}$$

Obviously we have $V = \sum_{i=1}^{N} V_i$ and $Q = \sum_{i=1}^{N} Q_i$.

For V_i, according to our definition, we obtain

$$V_i(s^t) = \mathbb{E}_{\mathbf{a}^t\sim\prod_{i=1}^{N} q_i}[r_i(s^t,a_i^t,a_{\mathcal{N}_i}^t) - \log q_i(a_i^t|s^t) + \mathbb{E}_{p(s^{t+1}|s^t,\mathbf{a}^t)}V_i(s^{t+1})]. \quad (16)$$

Now we relate it to Q_i, and have

$$V_i(s^t) = \mathbb{E}_{\mathbf{a}^t\sim\prod_{i=1}^{N} q_i}[Q_i(s_i^t,a_i^t,a_{\mathcal{N}_i}^t) - \log q_i(a_i^t|s^t)] = \mathbb{E}_{(a_i,a_{\mathcal{N}_i})\sim(q_i,q_{\mathcal{N}_i})}Q_i(s_i^t,a_i^t,a_{\mathcal{N}_i}^t) - \mathbb{E}_{a_i\sim q_i}\log q_i(a_i^t|s^t).$$

Thus it suggests that we should construct the loss function on V_i and Q_i in the following way. In the following, we use parametric family (e.g. neural network) characterized by η_i and κ_i to approximate V_i and Q_i respectively.

$$J(\eta_i) = \mathbb{E}_{s^t \sim D}[\frac{1}{2}\big(V_{\eta_i}(s^t) - \mathbb{E}_{(a_i, a_{\mathcal{N}_i}) \sim (q_i, q_{\mathcal{N}_i})}[Q_{\kappa_i}(s^t, a_i^t, a_{\mathcal{N}_i}^t)] - \log q_i(a_i^t|s^t)\big)^2],$$

$$J(\kappa_i) = \mathbb{E}_{(s^t, a_i^t, a_{\mathcal{N}_i^t}) \sim D}[\frac{1}{2}\big(Q_{\kappa_i}^i(s^t, a_t^i, a_{\mathcal{N}_i}^t) - \hat{Q}(s^t, a_t^i, a_{\mathcal{N}_i}^t)\big)^2]. \tag{17}$$

where $\hat{Q}_i(s^t, a_i^t, a_{\mathcal{N}_i}^t) = r_i + \gamma \mathbb{E}_{s^{t+1} \sim p(s^{t+1}|s^t, a^t)}[V_{\eta_i}(s^{t+1})]$.

Now we are ready to derive the update rule of the policy, i.e., the intention propagation network.

Remind the intention propagation network actually is a mean-field approximation of the joint-policy.

$$\min_{p_1, p_2, \ldots, p_n} KL(\prod_{i=1}^N p_i(a_i|s)||\pi^*(\mathbf{a}|s)).$$

It is the optimization over the *function* p_i rather than certain parameters. We have proved that after M iteration of intention propagation, we have output the nearly optimal solution q_i.

In the following, we will demonstrate how to update the parameter θ of the propagation network $\Lambda_\theta(\mathbf{a}^t|s^t)$, if we use neural network to approximate it. Again we minimize the KL divergence

$$\min_\theta \mathbb{E}_{s^t} KL(\prod_{i=1}^N q_{i,\theta}(a_i^t|s^t)||\pi^*(\mathbf{a}^t|s^t))$$

Plug the $\pi^*(\mathbf{a}^t|s^t) = \frac{\exp(Q(s^t, \mathbf{a}^t))}{\int \exp Q(s^t, \mathbf{a}^t) d\mathbf{a}_t}$ into the KL divergence. It is easy to see, it is equivalent to the following the optimization problem by the definition of the KL divergence.

$$\max_\theta \mathbb{E}_{s^t}\big[\mathbb{E}_{\mathbf{a}^t \sim \prod q_{i,\theta}(a_i^t|s^t)}[\sum_{i=1}^N Q_{\kappa_i}(s^t, a_i^t, a_{\mathcal{N}_i}^t) - \sum_{i=1}^N \log q_{i,\theta}(a_i^t|s^t)]\big].$$

Thus we sample state from the replay buffer and have the loss of the policy as

$$J(\theta) = \mathbb{E}_{s^t \sim D, \mathbf{a}^t \sim \prod_{i=1}^N q_{i,\theta}(a_i^t|s^t)}[\sum_{i=1}^N \log q_{i,\theta}(a_i^t|s^t) - \sum_{i=1}^N Q_{\kappa_i}(s^t, a_i^t, a_{\mathcal{N}_i}^t)].$$