DISTRIBUTED CONSENSUS CONTROL OF OPINION DYNAMICS

Author
Shayan Malekpour

Optimization Problem Summary

Objective Function:

The goal of the optimization is to find optimal opinions for each node in a social network, considering both global and local factors. The objective function is a combination of global and local costs:

$$\text{Objective Function} = \sum_{i=1}^{N} \text{cost\_local}[i] + \text{cost\_global}$$

Variables:

1. **Opinions:** $x[j][i]$ - Opinion of node j with respect to agent i. Dynamic and influenced by the iterative sharing process.

2. **Shared Opinion:** $M[i]$ - Shared opinion value set by the algorithm for agent i in iteration k. Independent of the network structure.

3. **Connection Weights:** $w[i][j]$ - Weight of the connection between agent i and its neighbor j. Constant throughout the optimization.

4. **Constants:** α, β - Parameters that control the trade-off between global and local costs.

Dynamics:

1. **Objective Function Components:**

   Local Cost for Agent j:

   $$\text{cost\_local}[i] = \sum_{j=1}^{N} w[i][j] \cdot |x[j][i] - M[i]|_2^2$$

   Global Cost:

   $$\text{cost\_global} = \alpha \cdot |\bar{X} - \text{fav}| * \beta \cdot \sqrt{k}$$

   Network Characteristics:

   $$X = f_{(T,U)}$$

2. **Optimization Process:**

   - Initialize opinions X, shared opinion values $M[i][k]$, and constants.
   - Adjust opinions based on optimization results and update shared opinion values for the next iteration.

3. **Connection Weights:** Constant throughout the optimization, representing the strength of connections between agents and their neighbors.

4. **Mean Opinion (mean_total):** Calculated based on the updated opinions in each iteration.

Network

The utilized network is a scale-free network in which nodes with the highest number of connections are named *hubs*. These hubs, three in total, represent the influencers of the social network. Connections of the network were generated randomly with a certain weight and direction, varying from 0 to 1.

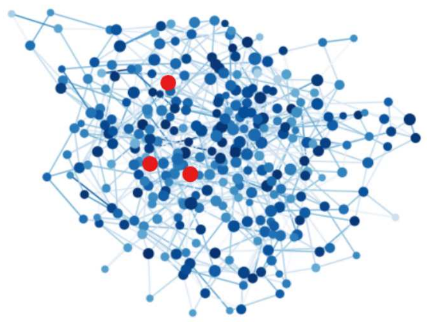


Figure 1: The graph containing hubs (in red) and the nodes (in blue).

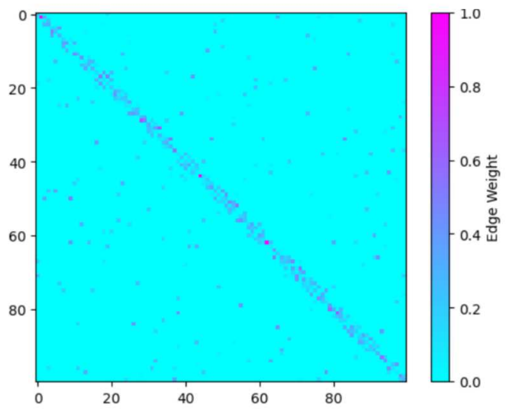


Figure 2: The heatmap of the weights (smaller).

$$\dot{w}_i = \frac{1}{N_F}\sum_{j=1}^{N_F} P\left(w_i, w_j\right)\left(w_j - w_i\right) + \frac{1}{N_L}\sum_{h=1}^{N_L} S\left(w_i, \tilde{w}_h\right)\left(\tilde{w}_h - w_i\right),$$

$$\dot{\tilde{w}}_k = \frac{1}{N_L}\sum_{h=1}^{N_L} R\left(\tilde{w}_k, \tilde{w}_h\right)\left(\tilde{w}_h - \tilde{w}_k\right) + u,$$

Currently, without any interventions or manipulations, the opinions of each agent do not reach the favorable state of consensus.

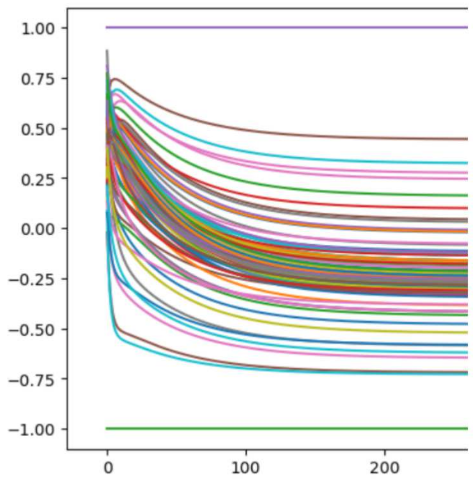


Figure 3: The graph containing hubs (in red) and the nodes (in blue).

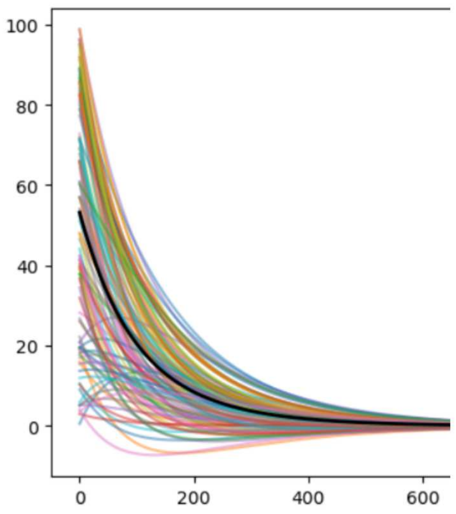


Figure 4: The heatmap of the weights (smaller).

0.1 Function Definition

The network has been defined as a function with inputs being $M[i][k]$ for all manipulating agents. The values of M represent the opinion each hub is expressing to its neighbors.

- $W[i][j]$: This value must stay constant throughout all iterations. It must be defined in the first iteration.
- The function is defined as a single iteration function. In each iteration of the network, the function will be called, and new values for the opinion and the inputs of the network will be assigned with respect to the previous values of the network.

Methods

1 Q-Learning

1.1 Q-Value Update

The Q-value for a state-action pair (s, a) is updated using the Q-learning equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

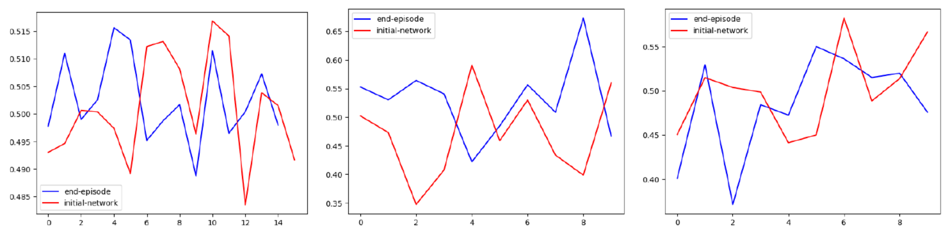
where $Q(s,a)$ is the Q-value for state s and action a, α is the learning rate, γ is the discount factor, $R(s,a)$ is the reward for taking action a in state s, and s' is the next state.

1.2 Exploration vs. Exploitation

To balance exploration and exploitation, we use an ϵ-greedy policy. With probability ϵ, we choose a random action (explore), and with probability $1 - \epsilon$, we choose the action with the highest Q-value (exploit).

1.3 Experimental Results

We implemented the Q-learning algorithm on our social network and conducted experiments to optimize opinion dynamics. We varied the discount factor γ and observed its effect on the convergence of opinions. Figures 5a, 5b, and 5c show the results for discount factors 0.9, 0.5, and 0.1 respectively.



(a) Discount Factor $\gamma = 0.9$ (b) Discount Factor $\gamma = 0.5$ (c) Discount Factor $\gamma = 0.1$

Figure 5: Comparison of Opinion Dynamics with Different Discount Factors

1.4 Discussion

We observed different behaviors in the convergence of opinions for each discount factor:

- For $\gamma = 0.9$, the convergence was slow but stable, resulting in a balanced distribution of opinions across the network.
- For $\gamma = 0.5$, the convergence was faster compared to $\gamma = 0.9$, but the resulting opinions showed some oscillations before stabilizing.
- For $\gamma = 0.1$, the convergence was rapid but resulted in erratic behavior with significant fluctuations in opinions.

The choice of the discount factor γ influences the trade-off between short-term rewards and long-term planning in the Q-learning algorithm.

Actor-Critic

2 Multi-Actor-Actor-Critic (MAAC) Algorithm

The Multi-Actor-Actor-Critic (MAAC) algorithm is an extension of the Actor-Critic algorithm designed for multi-agent environments. It addresses the challenges of learning in environments where multiple agents interact and influence each other's behavior.

2.1 Centralized Training, Decentralized Execution

MAAC follows a centralized training and decentralized execution paradigm. During training, a centralized critic is used to estimate the value function for each agent, taking into account the joint actions of all agents. However, during execution, each agent acts independently based on its local observations without access to the critic or actions of other agents.

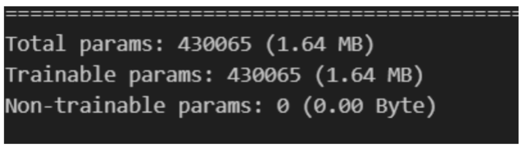
2.2 Algorithm

The MAAC algorithm updates the actor and critic networks based on the centralized critic's estimates of the value function for each agent. It alternates between selecting actions, receiving rewards, and updating the networks to optimize the policy and value function.

2.3 Actor Architecture

The actor network for each agent processes the local observation and outputs the probability distribution over the agent's action space. Here is a detailed breakdown of the actor's architecture:

1. **Input Layer:** The input layer takes the local observation of the agent, which has a shape of (300,).
2. **Layer Normalization 1:** Layer normalization is applied to the output of the first dense layer.
3. **Dense Layer 2:** The second dense layer has 75 units with the Mish activation function.
4. **Dropout 1:** Dropout with a rate of 0.1 is applied to the output of the second dense layer.
5. **Layer Normalization 2:** Layer normalization is applied again.
6. **Dense Layer 3:** The third dense layer has 64 units with the Mish activation function.
7. **Dropout 2:** Dropout with a rate of 0.1 is applied.
8. **Layer Normalization 3:** Layer normalization is applied.
9. **Dense Layer 4:** The fourth dense layer has 32 units with the Mish activation function.
10. **Dropout 3:** Dropout with a rate of 0.1 is applied.
11. **Dense Layer 5:** The final dense layer has 32 units with the Mish activation function.
12. **Output Layer:** The output layer produces a single unit with the softsign activation function, representing the actor's action.



```
==========================================
Total params: 430065 (1.64 MB)
Trainable params: 430065 (1.64 MB)
Non-trainable params: 0 (0.00 Byte)
```

2.4 Critic Architecture

The critic network for each agent estimates the value function by taking the local observation and the corresponding action as input. Here is a detailed breakdown of the critic's architecture:

1. **Input Layer:** The input layer concatenates the local observation and the actor's output from the previous step.
2. **Dense Layer 1:** The first dense layer consists of 64 units with the Mish activation function and L2 regularization.
3. **Layer Normalization 1:** Layer normalization is applied.
4. **Dense Layer 2:** The second dense layer has 32 units with the Mish activation function.
5. **Output Layer:** The output layer produces a single unit with a linear activation function, representing the estimated value for the critic.

2.5 Combined Actor-Critic Model

The combined model concatenates the outputs of all actor networks and critic networks. The architecture allows the model to simultaneously output the action probabilities for each agent and the estimated values for each agent.

2.6 Model Compilation

The entire actor-critic model is compiled using the Adam optimizer, and the loss function is defined as mean squared error (MSE) for both the actor and critic networks. The model is ready for training on the multi-agent environment.

2.7 Training Results

We conducted experiments to train the MAAC algorithm using the provided model architecture. Figure 6 shows the training results, including the learning curves and convergence behavior.

3 Conclusion

Overall, the MAAC algorithm shows promising results in optimizing opinion dynamics in a social network using the provided model architecture. However, the performance may be limited due to certain constraints of the critic network. Further experimentation and refinement of the model architecture may lead to improved results.

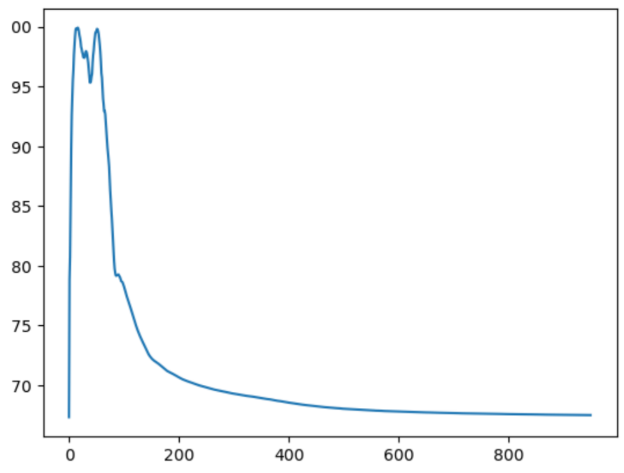


Figure 6: Training Results of the MAAC Algorithm

4 MADDPG

MADDPG is a powerful extension of the Deep Deterministic Policy Gradients (DDPG) algorithm tailored for cooperative multi-agent reinforcement learning scenarios. It addresses challenges such as non-stationarity and partial observability in a decentralized setting.

5 Key Features of MADDPG

1. **Decentralized Actor, Centralized Critic:** Each agent maintains a decentralized actor network for individual decisions, while a centralized critic network observes joint actions for global evaluation.
2. **Experience Sharing:** Agents share experiences with each other, allowing collaborative learning across the team to address non-stationarity.
3. **Soft Updates:** Target networks for both the actor and critic are updated softly, reducing variance during training.
4. **Exploration:** MADDPG employs noise in action selection for exploration, similar to DDPG, facilitating the discovery of better policies.

MADDPG Model Architecture

5.1 Actor Networks

The actor network for each agent i takes the state observation s_i as input and outputs a probability distribution over actions $\pi_i(a_i|s_i)$. The parameters of the actor network are updated using the policy gradient method, where the gradient is computed as:

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{a_i \sim \pi_i} \left[\nabla_{\theta_i} \log \pi_i(a_i|s_i) A_i \right]$$

Here, A_i represents the advantage function, which is computed as the difference between the estimated action-value function $Q(s, a_1, ..., a_N)$ and the state-value function $V(s)$.

$$\text{Actor Input: } s_i \quad \text{(Local observation for agent } i\text{)} \tag{2}$$

Actor Architecture:

Dense Layers with Mish Activation, Layer Normalization, and Dropout

Actor Output:

$$\text{Action Output: } a_i = \text{softsign}(\text{Actor}(s_i)) \tag{3}$$

5.2 Centralized Critic Network

The critic network estimates the action-value function $Q(s, a_1, ..., a_N)$, which represents the expected return when all agents take actions $a_1, ..., a_N$ in state s. The parameters of the critic network are updated by minimizing the mean squared error between the predicted value and the target value:

$$\text{Loss}_{\text{critic}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q(s, a_1, ..., a_N))^2$$

where y_i is the target value for agent i, computed using the Bellman equation:

$$y_i = r_i + \gamma Q(s', \pi_1(s'), ..., \pi_N(s'))$$

Here, r_i is the reward received by agent i from the environment, γ is the discount factor, and s' is the next state.

$$\text{Critic Input: } [s_1, s_2, \ldots, s_N, a_1, a_2, \ldots, a_N] \quad \text{(Joint state and actions)} \tag{4}$$

Critic Architecture:

Concatenation, Dense Layers with Mish Activation, and Layer Normalization

Critic Output:

$$\text{Value Output: } Q = \text{Critic}([s_1, s_2, \ldots, s_N, a_1, a_2, \ldots, a_N]) \tag{5}$$

5.3 Combined Actor-Critic Model

Combined Model Output:

$$[a_1, a_2, \ldots, a_N, Q] = \text{ActorCritic}([s_1, s_2, \ldots, s_N], [a_1, a_2, \ldots, a_N]) \tag{6}$$

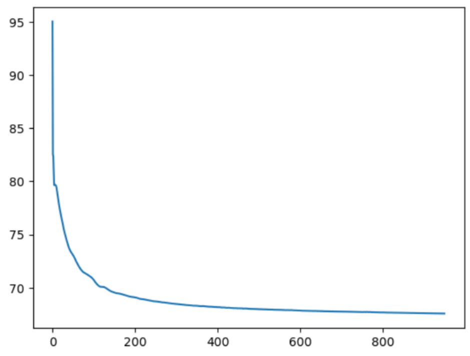
6 Model Compilation

The combined actor-critic model is compiled using the Adam optimizer, and the loss function is defined as the mean squared error (MSE) for both the actor and critic networks. The model is ready for training on a cooperative multi-agent environment.

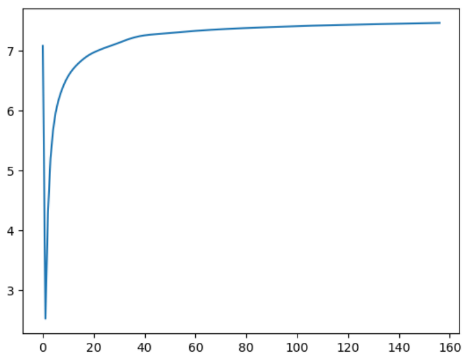
6.1 Training Loop

The training loop consists of the following steps:

1. Sample a batch of experiences $(s, a_1, ..., a_N, r, s')$ from the environment.
2. Compute the target value y_i for each agent using the Bellman equation.
3. Update the parameters of the critic network by minimizing the mean squared error.
4. Compute the advantage A_i for each agent.
5. Update the parameters of the actor network for each agent using the policy gradient method.



(a) Mean opinion value



(b) Progress

6.2 Conclusion

MADDPG provides a robust framework for multi-agent collaboration, combining decentralized decision-making with a centralized evaluation mechanism. The experience sharing and soft update mechanisms contribute to the stability of learning in complex environments.

7 COMA

The Counterfactual Multi-Agent Policy Gradients (COMA) algorithm is a reinforcement learning algorithm designed for cooperative multi-agent scenarios. It addresses the challenge of credit assignment in such environments by introducing a counterfactual baseline into the policy gradient update. This counterfactual baseline estimates the value of an agent's action considering the actions of other agents, enabling more effective credit assignment and improved learning stability.

7.1 Algorithm Overview

The key idea behind COMA is to decompose the joint action-value function into individual agent contributions using counterfactual baselines. The algorithm consists of the following steps:

1. **Agent Policy Update:** Each agent updates its policy based on the observed state and the counterfactual baseline.

   The policy update equation for agent i is given by:

$$\nabla_{\theta_i} J(\pi_i) = \mathbb{E}_{\pi_i}\left[\sum_{t=0}^{T} \nabla_{\theta_i} \log \pi_i(a_{i,t}|s_t) Q_i^{\pi}(s_t, \mathbf{a})\right] \tag{7}$$

   where θ_i represents the parameters of agent i's policy π_i, $J(\pi_i)$ is the performance metric, $\pi_i(a_{i,t}|s_t)$ is the probability of taking action $a_{i,t}$ in state s_t, $Q_i^{\pi}(s_t, \mathbf{a})$ is the action-value function, and $\mathbf{a}$ is the joint action vector.

2. **Joint Action-Value Estimation:** The joint action-value function is estimated using counterfactual baselines for each agent.

   The estimated joint action-value function is given by:

$$Q^{\pi}(s_t, \mathbf{a}) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} r_t\right] \tag{8}$$

   where r_t is the reward at time step t.

3. **Policy Gradient Calculation:** The policy gradient is computed using the estimated joint action-value function and the observed actions.

   The policy gradient for agent i is given by:

$$\nabla_{\theta_i} J(\pi_i) = \mathbb{E}_{\pi_i}\left[\sum_{t=0}^{T} \nabla_{\theta_i} \log \pi_i(a_{i,t}|s_t)(Q^{\pi}(s_t, \mathbf{a}) - b_i(s_t))\right] \tag{9}$$

   where $b_i(s_t)$ is the counterfactual baseline for agent i in state s_t.

4. **Policy Update:** The policies of all agents are updated simultaneously using the policy gradient.

7.2 Application to Multi-Agent Opinion Dynamics Problem

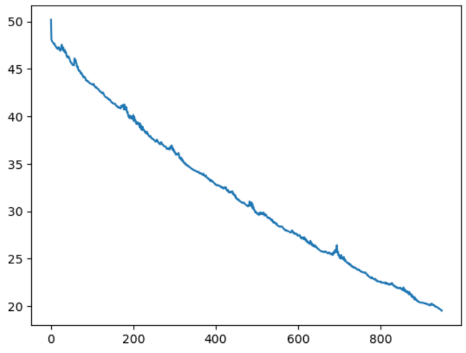
To apply the COMA algorithm to the multi-agent opinion dynamics problem, we can adapt the learning framework to incorporate the COMA-specific components. Specifically, we update the policy update step to include the counterfactual baseline and modify the joint action-value estimation step accordingly. Additionally, we adjust the policy gradient calculation to account for the counterfactual baselines.

7.3 Model Architecture

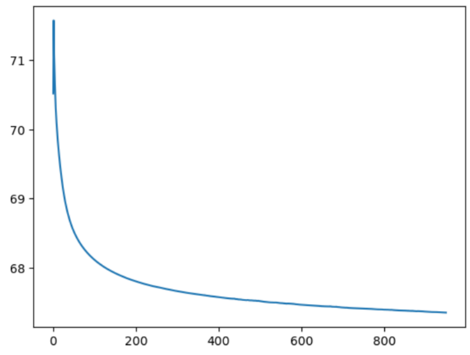
The model architecture used for COMA consists of individual actor networks for each agent and a centralized critic network. Each actor network takes the agent's observation as input and outputs the action. The centralized critic network takes the joint action and observation as input and estimates the joint action-value function.

7.4 Training Procedure

During training, the actor networks are updated based on the observed state and the counterfactual baseline, while the centralized critic network estimates the joint action-value function using counterfactual baselines for each agent. The policy gradient is then calculated using the estimated joint action-value function and the observed actions.



50
45
40
35
30
25
20
0
200
400
600
800



71
70
69
68
0
200
400
600
800