Social Learning via Multi Agent Deep Reinforcement Learning

Ege Can Doğaroğlu

Social Learning in Networks

We adapt the model in Brandl (2024). Specifically, we let $N = \{1, \dots, n\}$ be the set of agents, and $T = \{1, 2, \dots, t\}$ be the finite set of periods. In each period, each agent chooses an action a from the same finite set of actions A. The finite set of states of the world is Ω, where the true state $\omega \in \Omega$ is a random variable with full support distribution over Ω. Agents share the same utility function $u : A \times \Omega \to \mathbb{R}$ that depends on their own action and the state of the world, where a_θ denotes the unique optimal (correct) action in that state. It's also assumed that $a_\theta \neq a_{\theta'}$ for any two distinct states $\theta, \theta' \in \Omega$. In each period $t \in T$, each agent i privately observes a signal s_t^i from a set of signals S. Conditional on each state θ, s_t^i has distribution $\mu_\theta \in \Delta(S)$, and signals are independent across agents and periods. Observing the signal realization $s \in S$ changes the log-likelihood ratio of the observing agent between the states θ and θ' by,

$$\ell_{\theta,\theta'}(s) = \log \frac{d\mu_\theta(s)}{d\mu_{\theta'}(s)}.$$

Each agent i also observes the actions of her neighbors $N^i \subset N$, thus the information available to Agent i in period t before choosing an action is the collection of information sets $\mathcal{I}^i_{\leq t} = S^t \times A^{|N^i| \times (t-1)}$. It's said that Agent i makes a mistake in period t if $a_t^i \neq a_\omega$, and that i learns at rate r if,

$$r = \liminf_{t \to \infty} -\frac{1}{t} \log P(a_t^i \neq a_\omega).$$

Theorem 1 states that for any number of agents and any strategy profile, there exists an agent i such that,

$$r^i \leq r_{bdd} = \min_{\theta \neq \theta'} \{E_\theta[\ell_{\theta,\theta'}] + E_{\theta'}[\ell_{\theta',\theta}]\}.$$

Partially Observable Markov Games

Following the MARL literature, we model the social learning framework as a partially observable Markov game defined by the tuple,

$$(N, \Psi, A, \{O_i\}_{i \in N}, \Gamma, R, \gamma)$$

where,

- N is the set of agents,
- Ψ is the Markov state space, with individual element ψ,
- A is the shared action space for all agents, with individual element a,
- O_i is the observation space for agent i, with individual element o_i,
- $\Gamma : \Psi \times A^n \times \Psi \to [0, 1]$ is the transition function,
- $R : A \times \Psi \to \mathbb{R}$ is the shared reward function for all agents,
- $\gamma \in [0, 1)$ is the discount factor.

Markov States, Signals & Actions

For simplicity, we let $\Omega = \{\theta_1, \ldots, \theta_k\}$ be the set of possible states of the world; $S = \{s_k : \theta_k \in \Omega\}$ be the signal space, with s_k being the signal for state θ_k and $A = \{a_k : \theta_k \in \Omega\}$ be the action space, with a_k being the optimal action for state θ_k. We can then define the Markov state space as

$$\Psi = \{\omega\} \times S \times A^n$$

which accounts for the true state of the world, current period signal and current period actions. We leave the tracking of signal and action histories to the Recurrent Neural Networks (RNNs), thereby avoiding exponentially increasing Markov state space formulations.

Observations

The observation for each agent i at time t is,

$$o_t^i = (s_t^i, \boldsymbol{a_t}) \in O_i,$$

where $O_i = S \times A^n$ and $\boldsymbol{a_t} = \{a_t^1, \ldots, a_t^n\}$ is the action profile at period t.

Transitions

The stochastic components of the Markov state transitions are the signal realizations and the actions taken by the agents. Formally, for state $\psi_t = (\omega, (s_t^i)_{i \in N}, \boldsymbol{a_t})$ and next state $\psi_{t+1} = (\omega, (s_{t+1}^i)_{i \in N}, \boldsymbol{a_{t+1}})$, the transition function can be written as,

$$\Gamma(\psi_{t+1} | \psi_t) = \prod_{i \in N} \mu_\omega(s_{t+1}^i) \prod_{i \in N} \sigma_t^i(\mathcal{I}_{\leq t}^i)(a_{t+1}^i)$$

where σ_t^i is the *policy* or *strategy* of agent i at period t. There are two important points to consider here. First, in this formulation, we condition the policies on action and signal histories that are not part of the Markov state space. The fact that we can still use this information to compute the policies stems from the technology of RNNs that allows for a memory buffer from past observations, without explicitly storing the observations themselves. Second, diverging from the original paper, we allow for policies to be time-dependent, which corresponds to the learning process of the RL agents.

Rewards

The reward functions are identical across agents with $R_i(a, \psi) = R(a, \psi)$, $\forall i \in N$. The true reward is the flow payoff $u(a_t^i, \omega)$, where $u : A \times \Omega \to \mathbb{R}$ is known but its realizations are not observed. Lacking this observation, we follow Huang et al. (2024) and construct an observed payoff function $v(a, s)$ with the property that,

$$\mathbb{E}_{s \sim \mu^\theta}[v(a, s)] = u(a, \theta), \quad \forall \theta \in \Omega.$$

Setting $u(a, \omega) = \mathbf{1}_{\{a = a^\omega\}}$ the problem of constructing such a function amounts to solving a system of $|\Omega|$ linear equations for each action. More precisely, let $\boldsymbol{\mu}$ be an $k \times k$ matrix representing the signal distributions, where the entry $\mu^\theta(s)$ denotes the probability of observing signal s in state θ. We assume $\mu^\theta(s_\theta) > \mu^\theta(s)$, $\forall s \in S$, $\theta \in \Omega$, which implies that signal distributions are linearly independent across states and $\boldsymbol{\mu}$ is invertible. Similarly, for each action $a \in A$, define the *utility vector* $\boldsymbol{u}_a \in \mathbb{R}^k$ as,

$$\boldsymbol{u}_a = \begin{bmatrix} u(a, \theta_1) \\ u(a, \theta_2) \\ \vdots \\ u(a, \theta_k) \end{bmatrix}.$$

The vector of observable reward functions $\boldsymbol{v}_a \in \mathbb{R}^n$ for action a is the unique solution to the linear system as in,

$$\begin{aligned} \boldsymbol{\mu} \boldsymbol{v}_a &= \boldsymbol{u}_a, \\ \boldsymbol{v}_a &= \boldsymbol{\mu}^{-1} \boldsymbol{u}_a. \end{aligned}$$

Hence, the reward $v(a, s_j)$ for choosing action a when observing signal s_j is,

$$v(a, s_j) = \sum_{l=1}^{k} \left(\boldsymbol{\mu}^{-1}\right)_{jl} u(a, \theta_l),$$

where $\left(\boldsymbol{\mu}^{-1}\right)_{jl}$ denotes the (j, l)-th element of $\boldsymbol{\mu}^{-1}$.

Binary Case with $|\Omega| = 2$. For the case with two states of the world and symmetric signals with accuracy $q > 0.5$ we have,

$$\boldsymbol{\mu} = \begin{bmatrix} q & 1-q \\ 1-q & q \end{bmatrix}, \quad \boldsymbol{\mu}^{-1} = \frac{1}{2q-1} \begin{bmatrix} q & -(1-q) \\ -(1-q) & q \end{bmatrix}$$

Hence, for action a and flow utility $u(a, \omega) = \mathbf{1}_{\{a=\omega\}}$, the observed reward function becomes,

$$R(a, \psi) = v(a, s) = \frac{q \cdot \mathbf{1}_{\{a=s\}} - (1-q) \cdot \mathbf{1}_{\{a \neq s\}}}{2q-1}.$$

Recurrent Neural Networks

Before proceeding with the RNN architecture and learning framework, we define the key activation functions and operations:

The hyperbolic tangent function $\tanh : \mathbb{R} \to (-1, 1)$ is defined as,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The sigmoid function $\text{sigmoid} : \mathbb{R} \to (0, 1)$ is defined as,

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

The softmax function $\text{softmax} : \mathbb{R}^k \to \Delta^{k-1}$ maps a vector to a probability distribution,

$$\text{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}.$$

The Hadamard (element-wise) product $\odot : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ is defined as,

$$(x \odot y)_i = x_i y_i.$$

Following our partial observability setting, each agent i observes both their private signal s_t^i and the action profile $\mathbf{a}_t = (a_t^1, \ldots, a_t^n)$ at time t. The collection of these observations up to time t forms the information set $\mathcal{I}_{\leq t}^i = S^t \times A^{|N_i|(t-1)}$. Rather than including this growing information set in our Markov state space, we employ Recurrent Neural Networks to maintain a compact representation while preserving the essential historical information.

RNN Architecture for Information Processing

For each agent i, we implement an RNN that processes the sequence of observations $o_t^i = (s_t^i, \mathbf{a}_t) \in \mathcal{O}_i$ to produce a hidden state $h_t^i \in \mathbb{R}^d$ that encodes the information set $\mathcal{I}_{\leq t}^i$. The RNN updates its hidden state through a recursive function,

$$h_t^i = f_{\eta_i}(h_{t-1}^i, o_t^i),$$

where f_{η_i} is parameterized by η_i and implemented using a Gated Recurrent Unit (GRU) architecture, following Cho et al. (2014). The GRU utilizes two primary gates - an update gate and a reset gate - to selectively incorporate new information while preserving relevant historical context:

$$\begin{aligned}
z_t^i &= \text{sigmoid}(W_z^i[h_{t-1}^i, s_t^i, \mathbf{a}_t] + b_z^i) && \text{(update gate)} \\
\rho_t^i &= \text{sigmoid}(W_\rho^i[h_{t-1}^i, s_t^i, \mathbf{a}_t] + b_\rho^i) && \text{(reset gate)} \\
\tilde{h}_t^i &= \tanh(W_h^i[\rho_t^i \odot h_{t-1}^i, s_t^i, \mathbf{a}_t] + b_h^i) && \text{(candidate state)} \\
h_t^i &= (1 - z_t^i) \odot h_{t-1}^i + z_t^i \odot \tilde{h}_t^i && \text{(final update)}
\end{aligned}$$

Here, W_z^i, W_ρ^i, W_h^i are weight matrices and b_z^i, b_ρ^i, b_h^i are bias vectors specific to agent i.

The update gate z_t^i determines how much of the previous hidden state should be retained versus updated with new information. When z_t^i is close to 1, the network prioritizes the new candidate state $\tilde{h}_t^i$; when close to 0, it maintains more of the previous state h_{t-1}^i. This mechanism is particularly important in our social learning context, as it allows agents to appropriately weight new signals against accumulated historical evidence.

The reset gate ρ_t^i controls how much of the previous hidden state should influence the computation of the new candidate state $\tilde{h}_t^i$. When ρ_t^i is close to 0, it effectively "forgets" the previous state, allowing the network to drop irrelevant historical information. This is crucial when the agent needs to adapt to significant changes in the environment or when previous beliefs need to be revised based on new, strong evidence.

The candidate state $\tilde{h}_t^i$ is computed using the reset gate-modulated previous state along with current observations. The hyperbolic tangent activation ensures the candidate state values remain bounded between -1 and 1, promoting stable learning dynamics. The strategy or policy σ_t^i of agent i at time t is then derived from this hidden state through a policy network,

$$\sigma_t^i(\mathcal{I}_{\leq t}^i) = \pi_{\phi_i}(h_t^i),$$

where $\pi_{\phi_i}^i$ is a neural network with parameters ϕ_i that maps the hidden state to a distribution over actions,

$$\pi_{\phi_i}^i(h_t^i) = \text{softmax}(W_\pi^i h_t^i + b_\pi^i).$$

This architecture effectively augments our original Markov state space $\Psi = \{\omega\} \times S \times A^n$ with the RNN hidden states,

$$\tilde{\Psi} = \{\omega\} \times S \times A^n \times \mathcal{H}^n,$$

where $\mathcal{H} \subset \mathbb{R}^d$ is the space of hidden states. The transition function Γ is accordingly modified to include the hidden state updates,

$$\tilde{\Gamma}(\tilde{\psi}_{t+1}|\tilde{\psi}_t) = \prod_{i\in N} \mu_\omega(s_{t+1}^i) \prod_{i\in N} \pi_{\phi_i}^i(h_t^i),$$

where $\tilde{\psi}_t = (\omega, (s_t^i)_{i\in N}, \mathbf{a}_t, (h_t^i)_{i\in N})$. This gated architecture allows each agent to maintain a compact yet informative representation of their observation history, enabling them to learn sophisticated social learning strategies while keeping the state space tractable.

Centralized Training with Decentralized Execution

We adopt a CTDE paradigm where each agent maintains a decentralized actor network $\pi_{\phi_i}^i$ for action execution, while training utilizes a centralized critic network Q_ξ that has access to the global state. This approach preserves the decentralized nature of action execution while leveraging additional information during training, which mimics the role of a *social planner.*

Decentralized Actor Network

The decentralized actor network for agent i maps the hidden state to a policy distribution as before,

$$\pi_{\phi_i}^i(a_t^i|h_t^i) = \text{softmax}(W_\pi^i h_t^i + b_\pi^i).$$

Then, we use the policy gradient update to optimize this network given as,

$$\nabla_{\phi_i} J(\phi_i) = \mathbb{E}\left[\nabla_{\phi_i} \log \pi_{\phi_i}^i(a_t^i|h_t^i) A^i(\tilde{\psi}_t)\right]$$

where $A^i(\tilde{\psi}_t)$ is the advantage function computed using the centralized critic.

Centralized Critic Network

The centralized critic takes as input the global state ψ_t and all agents' hidden states:

$$Q_\xi(\psi_t, \mathbf{h}_t) = W_Q[\psi_t, \mathbf{h}_t, \mathbf{a}_t] + b_Q$$

where $\mathbf{h}_t = (h_t^1, \ldots, h_t^n)$. The critic network is updated to minimize the *temporal difference* error with the loss function,

$$\mathcal{L}(\xi) = \mathbb{E}\left[(r_t + \gamma Q_\xi(\tilde{\psi}_{t+1}) - Q_\xi(\tilde{\psi}_t))^2\right].$$

The advantage function for each agent is computed as,

$$A^i(\tilde{\psi}_t) = Q_\xi(\tilde{\psi}_t) - V_\xi(\tilde{\psi}_t),$$

where $V_\xi(\tilde{\psi}_t) = \mathbb{E}_{\mathbf{a}_t \sim \boldsymbol{\pi}}[Q_\xi(\tilde{\psi}_t]$ is the value function.

Learning Algorithm

The complete learning process alternates between the actor network update,

$$\phi_i \leftarrow \phi_i + \alpha \nabla_{\phi_i} J(\phi_i),$$

and the critic network update,

$$\xi \leftarrow \xi - \beta \nabla_\xi \mathcal{L}(\xi).$$

Single Agent Implementation

To validate our approach, we first implement a single-agent version of the learning framework. This simplified setting allows us to verify the learning dynamics and compare them against the theoretical bounds established in Brandl (2024). The implementation follows the binary state case ($|\Omega| = 2$) with symmetric signals of accuracy $q > 0.5$.

Metrics Tracking

We track two key metrics throughout the learning process:

- *Mistake Rate*: The empirical probability of the agent choosing an incorrect action, computed as the running average of mistakes:

$$P(a_t \neq a_\omega) = \frac{1}{t}\sum_{s=1}^{t} \mathbf{1}_{\{a_s \neq a_\omega\}}$$

- *Learning Rate*: Following the paper's definition, we compute:

$$r = \liminf_{t\to\infty} -\frac{1}{t}\log P(a_t \neq a_\omega)$$

  To approximate this limit, we compute the learning rate over exponentially increasing time windows and take the minimum rate, ensuring we capture the asymptotic behavior.

Environment

The environment implements the binary state case with the following components:

- A fixed true state $\omega \in \{0, 1\}$ drawn uniformly at random at initialization
- Signal generation with accuracy q:

$$P(s_t = \omega) = q, \quad P(s_t \neq \omega) = 1 - q$$

- The reward function derived in the binary case section:

$$v(a, s) = \frac{q \cdot \mathbf{1}_{\{a=s\}} - (1-q)\cdot \mathbf{1}_{\{a\neq s\}}}{2q - 1}$$

Neural Network Architecture

The agent employs an actor-critic architecture with both networks using GRU cells to process the history of signals. The GRU implementation follows the equations presented in the RNN Architecture section, with the following specifications:

Actor Network The actor network maps the signal history to a policy over actions:

$$\begin{aligned} h_t &= \mathrm{GRU}(s_t, h_{t-1}) \\ \pi(a_t|h_t) &= \mathrm{softmax}(W_\pi h_t + b_\pi) \end{aligned}$$

Critic Network The critic network estimates state values using the same GRU architecture:

$$\begin{aligned} h_t &= \mathrm{GRU}(s_t, h_{t-1}) \\ V(h_t) &= W_v h_t + b_v \end{aligned}$$

Training Process

The agent is trained using a variant of the Advantage Actor-Critic (A2C) algorithm. The complete training procedure is detailed in Algorithm 1.

Algorithm 1 Single Agent Social Learning Training

```
Require: Number of steps $T$, signal accuracy $q$, discount factor $\gamma$
Require: Actor learning rate $\alpha$, critic learning rate $\beta$
 1: Initialize actor network $\pi_\phi$ with parameters $\phi$
 2: Initialize critic network $Q_\xi$ with parameters $\xi$
 3: Initialize environment with signal accuracy $q$
 4: Sample true state $\omega \sim \mathrm{Uniform}\{0, 1\}$
 5: Initialize metrics tracker $\mathcal{M}$
 6: Get initial signal $s_0$
 7: Initialize hidden states $h_0 = \mathbf{0}$
 8: for $t = 1$ to $T$ do
 9:     // Actor forward pass with GRU
10:     $z_t = \mathrm{sigmoid}(W_z[h_{t-1}, s_{t-1}] + b_z)$
11:     $\rho_t = \mathrm{sigmoid}(W_\rho[h_{t-1}, s_{t-1}] + b_\rho)$
12:     $\tilde{h}_t = \tanh(W_h[\rho_t \odot h_{t-1}, s_{t-1}] + b_h)$
13:     $h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$
14:     $\pi_\phi(h_t) = \mathrm{softmax}(W_\pi h_t + b_\pi)$
15:     Sample action $a_t \sim \pi_\phi(\cdot|h_t)$
16:     $s_t, r_t, \mathrm{is\_mistake} = \mathrm{step}(a_t)$          ▷ Environment step
17:     // Critic forward pass
18:     $Q_\xi(s_{t-1}, h_{t-1}) = W_Q[s_{t-1}, h_{t-1}] + b_Q$
19:     $Q_\xi(s_t, h_t) = W_Q[s_t, h_t] + b_Q$
20:     $A_t = r_t + \gamma Q_\xi(s_t, h_t) - Q_\xi(s_{t-1}, h_{t-1})$
21:     $\mathcal{L}(\phi) = -\log \pi_\phi(a_t|h_t) A_t$          ▷ Policy loss
22:     $\mathcal{L}(\xi) = (r_t + \gamma Q_\xi(s_t, h_t) - Q_\xi(s_{t-1}, h_{t-1}))^2$          ▷ Value loss
23:     // Backward propagation
24:     $\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}(\phi)$          ▷ Update actor
25:     $\xi \leftarrow \xi - \beta \nabla_\xi \mathcal{L}(\xi)$          ▷ Update critic
26:     $\mathcal{M}$.add_mistake(is_mistake)
27:     $\mathcal{M}$.update_metrics()
28:     if $t \bmod 1000 = 0$ then
29:         Print current mistake rate and learning rate
30:     end if
31: end for
32: return trained networks $\pi_\phi$, $Q_\xi$ and metrics $\mathcal{M}$
```

The algorithm implements the A2C training loop, processing signals through GRU-based actor and critic networks while tracking learning metrics. The actor network learns a policy that minimizes mistakes in predicting the true state, while the critic network estimates state values to compute advantages for policy updates.

Theoretical Bounds

For the binary case with symmetric signals, the theoretical bound on the learning rate is:

$$r_{bdd} = -\log(1 - q_{bdd}) = -\log(1 - (2q - 1))$$

where $q_{bdd} = 2q - 1$ is the bound on the probability of choosing the correct action. This provides a benchmark against which we can evaluate the empirical learning rate of our implementation.

Empirical Results

The implementation is evaluated over a single continuos training run for a signal accuray of $q = 0.75$ and $T = 10^5$ steps. We track the evolution of the mistake rate and empirical learning rate over time to compare against the theoretical bound. The results are visualized in Figure 1. The learning curves demonstrate that the agent successfully learns to minimize mistakes and that the learning rate stays below the theoretical learning rate bound, validating the implementation.

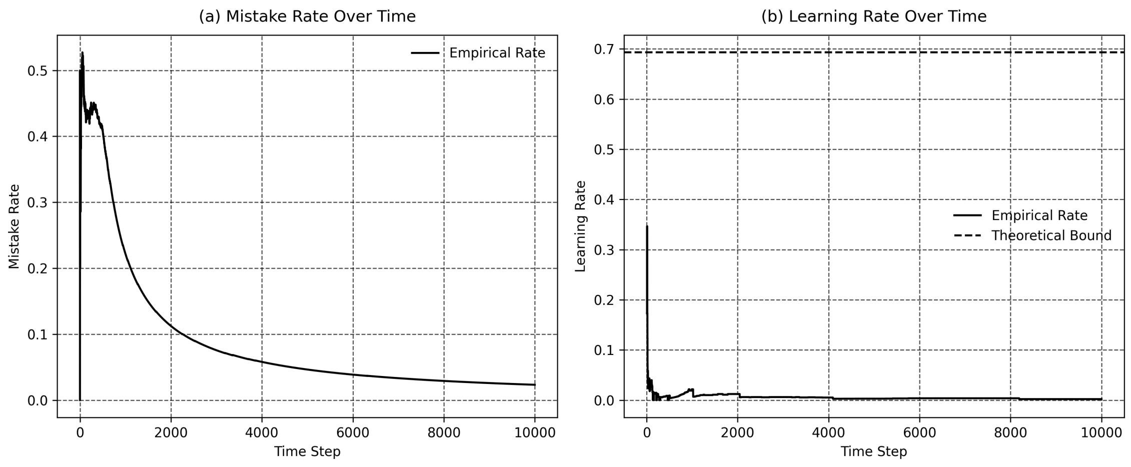

Figure 1: Learning curves for single agent

References

Brandl, F. (2024). The social learning barrier.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.