Learning Communication, Coalgebraically

1 The question

How can two agents learn to communicate?

Imagine a sequential interaction in which each of two players can output messages, but these messages may initially be written in completely different *languages* (or even modalities). How, in principle, can they bridge this gap and eventually communicate?

One response would be to create some external goal that the two players A and B must mutually achieve and to treat language like any other action to be learned via reinforcement learning. Specifically, a particularly elegant and instrumentally convergent task is that of predicting future observations.

At this point, there are several modeling choices. For instance, we could give A access to B's next observation and have A learn to give short messages to B such that B can predict its observations conditional on A's messages (the messages would have to be short or of a different type than the observations to be interesting, because otherwise the message could be a copy of the observations).

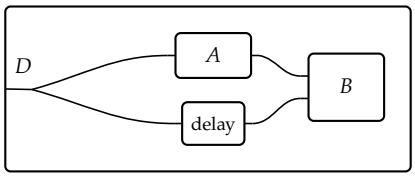

Figure 1: Asymmetric sender–receiver setup (counterexample of what we do not want to do). A and B both receive the shared stream D, but A's message directly conditions B. The upper branch shows A's message feeding into B; the lower branch contains one one-step delay block before reaching B, ensuring B sees D with a lag so A can comment on the previous observation.

Exposition. The outer frame denotes a synchronous system. At each step, both agents receive D. Agent A also emits a signal that conditions agent B directly at its top input, while a second copy of D traverses one one-step delay block before entering B's lower input. This lag means that by the time B observes D on the lower branch, A can send a message about the previous observation. Subsequent figures make the messages and learning signals explicit.

2 Toward a symmetric setup

But this setup eschews some of the more cooperative/bidirectional/mutually recursive aspects of the genesis of effective communication. Instead, consider a situation where neither player has privileged information to incoming data; rather, they both have access to a shared input data stream. They each can send messages to the other player, and each player's goal is to maximize the sum of the log probabilities that each player ascribes to their observations, making it a kind of cooperative game.

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \left(\log P_A(\text{observation}_t \mid \text{ctxt}) + \log P_B(\text{observation}_t \mid \text{ctxt}) \right)$$

Simple Symmetric Communication

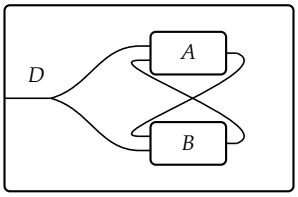

Figure 2: Simple Symmetric Communication. Timing convention (Moore): at each discrete step t, each agent first emits a message based only on its current internal state. After emission, both agents read the shared observation D_t and the partner's message to update their internal states for the next step. Messages at step t therefore do not depend on D_t.

3 Players as coalgebras

We can model each player then as an F-coalgebra $(S_X,\ \alpha_X \colon S_X \longrightarrow FS_X)$, where F will be specialized to encode the notion that a player can *receive inputs* and *produce outputs* – e.g. $F(S) = \mathsf{Out} \times S^{\mathsf{In}}$ for some types Out and In.

Now we can add in requirements on the structure of S, In, and Out to match the problem setup. First, to model the emergence of communication, we should model the ability of the players to learn. We can do this by expressing S as $\Theta \times \_$, where Θ is the space of parameterizations of the player's behavior. Next Out must include A's "raw" message type $M_{A\to B}$. It is "raw" because we will need to include some metadata in order to give the learning algorithm the necessary reward signals.

Next we have In_A and In_B which each consist of the other player's output type and the data type D to be predicted: $\mathsf{In}_A = D \times \mathsf{Out}_B$ and $\mathsf{In}_B = D \times \mathsf{Out}_A$.

Then we need a way of talking about the "probability that a player ascribes to an observation", and for the learning algorithm to work, we will also need to compute the "probability of taking a particular action", where both of these functions are parameterized by $\theta \in \Theta$.

$$\Theta \xrightarrow{\pi, P} \Delta M \times (\Delta D)^M$$

where π is called the policy, and we require implementations of π and P for both players. So the policy produces a distribution over output messages (actions) given a state, and the predictive model P produces a function from messages to distributions over input data.

The state update part of α is going to be a gradient descent step with respect to $\theta \in \Theta$, but the loss function needs to include not only player A's own score $\ln P^A_\theta(d \in D \mid m_{B\to A})$, but also B's score from the previous round given A's previous message. We will explicitly tie parameters within each agent: the same $\theta_X \in \Theta_X$ controls both the action policy π_X and the prediction model P_X.

So A's state must include not just their weights $\theta_A \in \Theta_A$, but also the score that B received in the previous time step as well as the message that caused that score for the other player.

4 Prediction and policy components

Written out, we have $S_A = \Theta_A \times \mathbb{R} \times M_{B\to A}$ and $S_B = \Theta_B \times \mathbb{R} \times M_{A\to B}$. Each player X consists of an initial state s_0^X and functions $U_X : \mathsf{In}_X \times S_X \to S_X$ and $\Pi_X : S_X \to \mathsf{Out}_X$, where $\mathsf{Out}_A = M_{A\to B} \times \mathbb{R} \times M_{B\to A}$, $\mathsf{In}_A = D \times \mathsf{Out}_B$, and $\mathsf{In}_B = D \times \mathsf{Out}_A$.

Recall we have access to functions $\pi_A : \Theta_A \to \Delta M_{A\to B}$, $\pi_B : \Theta_B \to \Delta M_{B\to A}$, $P_A : \Theta_A \to (M_{B\to A} \to \Delta D)$, and $P_B : \Theta_B \to (M_{A\to B} \to \Delta D)$ to implement U_A, U_B, Π_A, and Π_B. Within each agent, θ_X is shared between π_X and P_X. We use Moore timing for emission and update; explicit online update equations appear in the Coalgebraic Maximum-Likelihood section.

Detailed Symmetric Communication

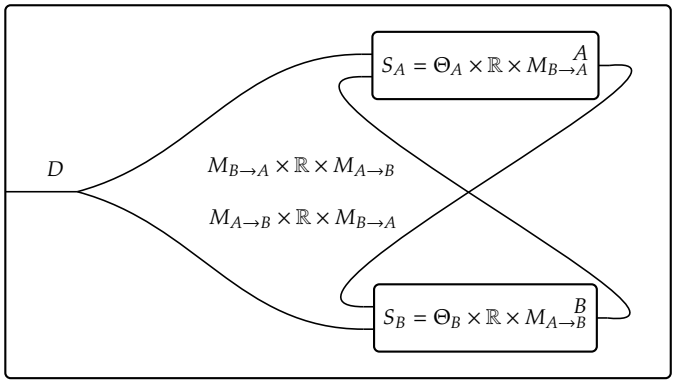

Figure 3: Detailed Symmetric Communication (Moore timing): at each discrete step t, each agent X first emits its message $\Pi_X(s_t^X)$ (equivalently, sample $m_t^X \sim \pi_{\theta_X}(\cdot)$ based on s_t^X). After emission, X consumes the current inputs (D_t, partner's message at t) and applies U_X to obtain s_{t+1}^X. Thus, actions at step t are functions of s_t^X only, while observations D_t influence the next state.

Explicit policy (output) maps For $X \in \{A, B\}$ with state $S_X = \Theta_X \times \mathbb{R} \times M_{Y\to X}$ and output $\mathsf{Out}_X = M_{X\to Y} \times \mathbb{R} \times M_{Y\to X}$,

$$\Pi_A\big((\theta_A, r_A, m_{B\to A})\big) = \text{let } m_{A\to B} \sim \pi_{\theta_A}(\cdot) \text{ in } (m_{A\to B},\ r_A,\ m_{B\to A}),$$
$$\Pi_B\big((\theta_B, r_B, m_{A\to B})\big) = \text{let } m_{B\to A} \sim \pi_{\theta_B}(\cdot) \text{ in } (m_{B\to A},\ r_B,\ m_{A\to B}).$$

Explicit update maps (single-step Monte Carlo) With Moore timing, at step t each agent first emits via Π_X, then consumes inputs (D_t, partner out) and updates its state. A simple instantiation of the online ascent update (using the immediate cooperative score as the learning signal for the previous action) is:

$$U_A\big((d,(m_{B\to A}, r_B, m_{A\to B}^{\text{old}})),\ (\theta_A,\ r_A,\ m_{B\to A}^{\text{old}})\big) :=$$
$$\Big(\theta_A + \underbrace{\nabla_{\theta_A} \ln P_{\theta_A}(d \mid m_{B\to A})}_{\text{predictor ascent}} + \underbrace{r_B \nabla_{\theta_A} \ln \pi_{\theta_A}(m_{A\to B}^{\text{old}})}_{\text{policy ascent}},\ \underbrace{\ln P_{\theta_A}(d \mid m_{B\to A})}_{\text{store new score}},\ m_{B\to A}\Big),$$
$$U_B\big((d,(m_{A\to B}, r_A, m_{B\to A}^{\text{old}})),\ (\theta_B,\ r_B,\ m_{A\to B}^{\text{old}})\big) :=$$
$$\Big(\theta_B + \underbrace{\nabla_{\theta_B} \ln P_{\theta_B}(d \mid m_{A\to B})}_{\text{predictor ascent}} + \underbrace{r_A \nabla_{\theta_B} \ln \pi_{\theta_B}(m_{B\to A}^{\text{old}})}_{\text{policy ascent}},\ \underbrace{\ln P_{\theta_B}(d \mid m_{A\to B})}_{\text{store new score}},\ m_{A\to B}\Big).$$

Here the second component of each state stores the last predictive log-likelihood (used as the scalar learning signal in the partner's next update), and the third stores the last partner message. More sophisticated multi-step or discounted updates (with baselines, eligibility traces, or buffers) can be substituted per Section 6 and the Coalgebraic MLE discussion.

The important point is how the weights get updated in U_A / U_B. The $\nabla_{\theta_B} \ln P_{\theta_B}(d \mid m_{A\to B})$ term allows the player to get better at predicting observations given the other's messages, and $r_A \nabla_{\theta_B} \ln \pi_{\theta_B}(m_{B\to A})$ is a policy-gradient/REINFORCE-style term; the combined objective and full online update are given below.

5 Coalgebraic Maximum-Likelihood Objective

We can phrase the cooperative objective as a maximum-likelihood problem over the *composite* synchronous system formed by wiring the two agents together with the exogenous data source D under Moore timing.

Coalgebraic view. Each agent $X \in \{A, B\}$ is an F-coalgebra

$$\alpha_X : \; S_X \longrightarrow \Delta(M_{X\to Y}) \times S_X^{D\times M_{Y\to X}}, \tag{1}$$

whose components are the stochastic output map $s \mapsto \pi_{\theta_X}(\cdot \mid s)$ and the deterministic/stochastic update map $(s,(d,m)) \mapsto U_X(s,d,m)$. Wiring A and B yields a composite Moore machine over the joint state $S = S_A \times S_B$.

Subjective per-step likelihood. Since D_t is exogenous, the only θ-dependent "likelihood" the composite can assign to the observation stream is via the agents' predictors. At time t define the cooperative score

$$r_t = \log P_{\theta_B}(D_t \mid m^t_{A\to B}) + \log P_{\theta_A}(D_t \mid m^t_{B\to A}), \tag{2}$$

and the multiplicative weight

$$w_t = \exp(r_t) = P_{\theta_B}(D_t \mid m^t_{A\to B})\, P_{\theta_A}(D_t \mid m^t_{B\to A}). \tag{3}$$

Coalgebraic MLE objective. For a discounted horizon $\gamma \in (0,1)$, define the trajectory log-likelihood functional produced by the composite machine as

$$\mathcal{L}_\gamma(\tau;\theta) = \sum_{t\geq 1} \gamma^{t-1} \log w_t \tag{4}$$

$$= \sum_{t\geq 1} \gamma^{t-1} \Big(\log P_{\theta_B}(D_t \mid m^t_{A\to B}) + \log P_{\theta_A}(D_t \mid m^t_{B\to A}) \Big). \tag{5}$$

Our coalgebraic MLE problem is then

$$\max_\theta \; J_\gamma(\theta) := \; \mathbb{E}_{\tau\sim P_\theta}\big[\mathcal{L}_\gamma(\tau;\theta)\big]. \tag{6}$$

that is, choose the coalgebra parameters θ (which determine output and update components) so that the *wired* Moore machine assigns maximal subjective likelihood to the observed stream according to its internal predictors. The exogenous factors $P(D_t)$ cancel as they are θ-independent.

Online ascent. Let $G_t = \sum_{k\geq t} \gamma^{k-t} r_k$ and let $\widehat{G}_t$ be any causal estimator with $\mathbb{E}[\widehat{G}_t \mid \mathcal{F}_t] = G_t$. With Moore timing (emission precedes update), unbiased stochastic-ascent updates that maximize J_γ are

$$\theta_A^{t+1} = \theta_A^t + \eta_t \left[(\widehat{G}_t - b_t)\, \nabla_{\theta_A^t} \ln \pi_{\theta_A^t}(m^t_{A\to B}) + \gamma^{t-1} \nabla_{\theta_A^t} \ln P_{\theta_A^t}(D_t \mid m^t_{B\to A}) \right],$$

$$\theta_B^{t+1} = \theta_B^t + \eta_t \left[(\widehat{G}_t - b_t)\, \nabla_{\theta_B^t} \ln \pi_{\theta_B^t}(m^t_{B\to A}) + \gamma^{t-1} \nabla_{\theta_B^t} \ln P_{\theta_B^t}(D_t \mid m^t_{A\to B}) \right],$$

where b_t is any baseline independent of the emission given $\mathcal{F}_t$. The policy term propagates credit for how messages at time t influence all future discounted predictive scores; the supervised term improves the immediate predictive likelihood. Under standard step-size conditions, these updates ascend J_γ in expectation.

6 Possible improvements

Our online estimator can be high-variance. Two practical mitigations are:

1. maintain truncated K-step memories or eligibility traces to better approximate $\sum_{i=0}^{\infty} \gamma^i r_i$ for past actions,
2. add batching/averaging (e.g., GRPO-style) to reduce estimator variance.