Mathematical Framework for YAM (Yet Another Model)

Contents

1 Model Framework

1.1 Directed Graph Representation of Traits

Learning structures, consisting of traits and their prerequisite relationships are represented as a directed graph $G = (V, E)$:

- $V = \{0, 1, \ldots, n-1\}$ is the set of n traits.
- $E \subseteq V \times V$ is the set of directed edges, where an edge (i, j) indicates that trait i is a prerequisite for trait j.

The first trait (trait 0) represents the naive trait, or starting point, of a learning structure. Its edges to other traits represent what a fully naive learner can learn.

The adjacency matrix $A \in \{0,1\}^{n \times n}$ encodes the graph structure:

$$A_{ij} = \begin{cases} 1 & \text{if there is a directed edge from trait } i \text{ to trait } j, \\ 0 & \text{otherwise.} \end{cases}$$

The set of parent traits (prerequisites) for trait j is:

$$P_j = \{i \in V : A_{ij} = 1\}.$$

1.2 Learner State

An learner's state is represented by a vector $r \in \{0,1\}^n$, where:

$$r_j = \begin{cases} 1 & \text{if trait } j \text{ is known (learned) by the agent,} \\ 0 & \text{otherwise.} \end{cases}$$

This state represents the repertoire of a learner We consider a learner that starts in the naive state r_0, in which only trait 0 is known.

1.3 Trait Frequencies

Each trait j has an associated frequency $f_j \geq 0$ in the background population. The computation of frequencies will be discussed later in this document. The frequency of the naive trait (trait 0) is set to 1 and is not updated.

1.4 Learnability of Traits

A trait j is *learnable* at state r if all its prerequisites are learned:

$$L_j(r, A) = \prod_{i \in P_j} r_i.$$

Here, $L_j(r, A) = 1$ if all parent traits of j are known, and $L_j(r, A) = 0$ otherwise. The graph interpretation of this is that a trait is learnable when all traits with edges to that trait are known.

1.5 Possible States

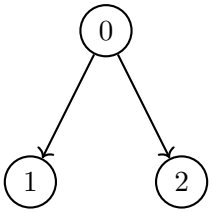
Given a learning structure represented by an adjacency matrix, the possible states of a learner can be determined by examining which traits are learnable from already known traits. We achieve this by following the directed edges in the graph representation to identify sequential trait acquisition paths.

We consider two different structures to illustrate this.

Structure A and its Possible States

For Structure A, the adjacency matrix A is given by:

$$\begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$



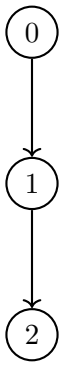
In this configuration, trait 0 is the starting point. From the adjacency matrix, trait 0 has directed edges to traits 1 and 2, indicating that both traits 1 and 2 can be learned directly from trait 0. Hence, the possible states of a learner in structure A, starting from the naive state (having only trait 0), are:

- Initial state: $r = (1, 0, 0)$
- After learning trait 1: $r = (1, 1, 0)$
- After learning trait 2: $r = (1, 0, 1)$
- After learning both traits 1 and 2: $r = (1, 1, 1)$

Structure B and its Possible States

For Structure B, the adjacency matrix B is given by:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$



In this structure, trait 0 enables the learning of trait 1, and subsequently, trait 1 enables the learning of trait 2. Therefore, the possible states, starting from the naive state, are:

- Initial state: $r = (1, 0, 0)$
- After learning trait 1: $r = (1, 1, 0)$
- After learning trait 2 (enabled by learning trait 1 first): $r = (1, 1, 1)$

Through this examination of adjacency matrices and their corresponding directed graphs, we can determine the sequential paths of trait acquisition, revealing all possible learner states.

1.6 Learning Strategies

Learning strategies determine the weights assigned to each trait when deciding which trait to attempt to learn next.

For all learning strategies, the base weight assigned to trait j at state r is zero if the trait is already known ($r_j = 1$), i.e., learners do not attempt to learn traits that are in their repertoire.

1.6.1 Random Learning Strategy

In the Random Learning Strategy, the learner selects traits to attempt to learn based solely on their frequencies, without considering payoffs or other factors.

Base Weights The base weight for trait j is:

$$w_j^* = \begin{cases} f_j, & \text{if } r_j = 0, \\ 0, & \text{otherwise.} \end{cases}$$

1.6.2 Payoff-Based Learning Strategy

In the Payoff-Based Learning Strategy, the agent prefers traits with higher payoffs.

Base Weights The base weight for trait j is:

$$w_j^* = \begin{cases} f_j \cdot p_j, & \text{if } r_j = 0, \\ 0, & \text{otherwise.} \end{cases}$$

1.6.3 Proximal Learning Strategy

The Proximal Learning Strategy models a learner who prefers to learn from demonstrators who are slightly ahead in trait acquisition.

Base Weights The base weight for trait j is calculated as follows:

1. **State Weights**: For each possible state s in the set of all possible states $\mathcal{S}$, compute the state weight:
$$w_s = f_s \cdot \phi(\Delta(r, s)),$$
where:
   - f_s is the frequency of state s in the population,
   - $\Delta(r, s) = \sum_{k=0}^{n-1}(s_k - r_k)$,
2. $\phi(\Delta) = 2^{1-\Delta}$ assigns higher weights to states with Δ close to 1.

Trait Contributions: For each trait j, compute:

$$w_j^* = \begin{cases} \sum\limits_{s \in \mathcal{S}} \dfrac{w_s}{N_s} \cdot \delta_{s_j,1}, & \text{if } r_j = 0, \\ 0, & \text{otherwise,} \end{cases}$$

where:

- $N_s = \sum_{k=0}^{n-1}(1 - r_k)s_k$ is the number of traits known in s but not in r,
- $\delta_{s_j,1}$ is the Kronecker delta function, which is 1 if $s_j = 1$ and 0 if $s_j = 0$.

In summary, the proximal learner gives a weight to each observed state, which is maximal for states with exactly one more trait. The state weight is the evenly divided among the traits in the state that are not known by the learner. The base weight for each trait is then the sum of weights for that trait across states.

1.6.4 Prestige-Based Learning Strategy

The Prestige-Based Learner functions similarly to the Proximal Learner but favors more advanced states by assigning higher weights exponentially based on the difference in traits.

Base Weights

1. **State Weights**: For each possible state s in the set of all possible states $\mathcal{S}$, compute the state weight:
$$w_s = f_s \cdot 2^{(\Delta(r,s)-1)},$$
where:
   - f_s is the frequency of state s in the population,
   - $\Delta(r, s) = \sum_{k=0}^{n-1}(s_k - r_k)$.
2. **Trait Contributions**: For each trait j, compute:
$$w_j^* = \begin{cases} \sum_{s \in \mathcal{S}} \frac{w_s}{N_s} \cdot \delta_{s_j,1}, & \text{if } r_j = 0, \\ 0, & \text{otherwise,} \end{cases}$$
where:
   - $N_s = \sum_{k=0}^{n-1}(1 - r_k)s_k$,
   - $\delta_{s_j,1}$ is the Kronecker delta function, which is 1 if $s_j = 1$ and 0 if $s_j = 0$.

1.6.5 Conformity-Based Learning Strategy

The Conformity-Based Learner focuses on the prevalence of states, learning traits that are more common in the population.

Base Weights

1. **State Weights**: The state weight is directly the frequency of the state:
$$w_s = f_s.$$
2. **Trait Contributions**: For each trait j, compute:
$$w_j^* = \begin{cases} \sum_{s \in \mathcal{S}} \frac{w_s}{N_s} \cdot \delta_{s_j,1}, & \text{if } r_j = 0, \\ 0, & \text{otherwise,} \end{cases}$$
where:
   - $N_s = \sum_{k=0}^{n-1}(1 - r_k)s_k$,
   - $\delta_{s_j,1}$ is the Kronecker delta function, which is 1 if $s_j = 1$ and 0 if $s_j = 0$.

1.7 Payoffs

The payoff for each trait is a function of its distance from the root trait (trait 0) and a parameter α:
$$p_j = u_j + \alpha d_j,$$
where:

- $u_j \sim U(0, 1)$ is a uniform random variable,
- d_j is the distance of trait j from the root trait,
- $\alpha \geq 0$ is a parameter that controls the influence of distance on payoff.

2 Constructing the Transition Matrix

2.1 Adjusted Weights

The adjusted weights are obtained by normalizing the base weights for the traits at state r:

$$w_j(r) = \frac{w_j^*}{W(r)},$$

where $W(r) = \sum_{k=0}^{n-1} w_k^*$ is the total weight of all traits (excluding known traits) at state r.

2.2 Transition Probabilities

From state r, the agent attempts to learn a trait j based on the adjusted weights $w_j(r)$. The transition probabilities are defined as:

1. **Attempting to Learn Trait j**:

$$P_{\text{attempt}}(j \,|\, r) = w_j(r).$$

2. **Outcome of Attempt**:

   - **If Trait j is Learnable at r ($L_j(r, A) = 1$)**:

   $$P(r \to r') = P_{\text{attempt}}(j \,|\, r),$$

   where $r' = r + e_j$.
   - **If Trait j is Not Learnable at r ($L_j(r, A) = 0$)**:

   $$P(r \to r) = P_{\text{attempt}}(j \,|\, r).$$

Therefore, the total probability of moving to a new state $r' = r + e_j$ is:

$$P(r \to r') = w_j(r) \cdot L_j(r, A).$$

The total probability of staying in the same state r is:

$$P(r \to r) = \sum_{j=0}^{n-1} w_j(r) \cdot (1 - L_j(r, A)) \,.$$

2.3 Summary of Transition Probabilities

$$P(r \to r') = \begin{cases} w_j(r) \cdot L_j(r, A), & \text{if } r' = r + e_j, \\ w_j(r) \cdot (1 - L_j(r, A)) \,, & \text{if } r' = r, \\ 0, & \text{otherwise.} \end{cases}$$

2.4 Constructing the Transition Matrix P

The transition matrix P represents the probabilities of transitioning from one learner state r to another state r'. It is constructed using the possible states and their corresponding transition probabilities, capturing the dynamics of the learning process.

To construct P, we proceed as follows:

1. **List All Possible States:**

- Identify all possible learner states r in the model. Each state corresponds to a unique combination of learned traits in $\{0,1\}^n$.

2. **Compute Transition Probabilities for Each State r:**

   (a) **Calculate Base Weights w_j^*:**
   - For each trait j not yet known ($r_j = 0$), compute the base weight w_j^* according to the chosen learning strategy (as defined in previous sections).

   (b) **Calculate Adjusted Weights $w_j(r)$:**
   - Normalize the base weights to obtain adjusted weights:

$$w_j(r) = \frac{w_j^*}{\sum_{k:r_k=0} w_k^*} \quad \text{for } r_j = 0.$$

   (c) **Determine Learnable Traits:**
   - For each trait j, determine if it is learnable from state r by evaluating $L_j(r, A)$:

$$L_j(r, A) = \prod_{i \in P_j} r_i.$$

   (d) **Compute Transition Probabilities $P_{r,r'}$:**
   - For each trait j:
     - *If trait j is learnable ($L_j(r, A) = 1$)*:

$$P_{r,r'} = w_j(r),$$

       where $r' = r + e_j$ is the state after learning trait j.
     - *If trait j is not learnable ($L_j(r, A) = 0$)*:

$$P_{r,r} = P_{r,r} + w_j(r),$$

       accumulating the probability of remaining in state r.

By systematically applying these steps to each state r, we fill the entries of the transition matrix P. Each row of P corresponds to a current state r, and each column corresponds to a potential next state r', including the possibility of remaining in the same state. This matrix fully captures the probabilistic dynamics of state transitions within the learning structure, given a strategy.

2.4.1 Absorbing States

An absorbing state in this context is a learner state r where the probability of remaining in the same state is 1, and there are no new traits to learn. This occurs when the learner has acquired all available traits, i.e., $r_j = 1$ for all traits j. In such a state, the adjusted weights $w_j(r)$ are zero for all traits (since all traits are already known), and thus, there are no possible transitions to a new state. The transition probability satisfies $P_{r,r} = 1$, making r an absorbing state in the Markov process.

2.5 Computing Trait Frequencies

Trait frequencies f_j are essential for constructing the transition matrix P, as they influence learning strategies and, consequently, the dynamics of the model. However, trait frequencies themselves depend on the state occupancies of the population, leading to a circular dependency: the trait frequencies are needed to compute P, but computing state occupancies from P requires knowing the trait frequencies.

To resolve this circularity, we employ a two-step approach:

1. **Preliminary Transition Matrix Construction:**
   - We begin by assuming equal frequencies for all traits, setting $f_j = 1$ for every trait j.
   - With these preliminary trait frequencies, we construct an initial transition matrix $P^{(0)}$ using the methods outlined in previous sections, incorporating the chosen learning strategies.
2. **Calculating Trait Frequencies from State Occupancies:**
   - **Computing the Stationary Distribution:**
     - The Markov process defined by $P^{(0)}$ may include absorbing states (states from which the process cannot leave). To focus on the dynamics before absorption, we consider the submatrix $Q^{(0)}$, representing the transient submatrix of $P^{(0)}$ and corresponding to transitions among the transient (non-absorbing) states.
     - We compute the stationary distribution π over the transient states by solving the system:
       $$\pi^\top = \pi^\top Q^{(0)},$$
       subject to the normalization condition:
       $$\sum_{r \in \mathcal{T}} \pi_r = 1,$$
       where $\mathcal{T}$ denotes the set of transient states, and π_r is the stationary probability of state r.
     - This linear system can be expressed as:
       $$\left(I - Q^{(0)\top}\right) \pi = 0,$$
       with I being the identity matrix.
     - To solve for π, we can remove one equation (since the system is singular due to the probability constraint) and replace it with the normalization condition. The resulting system of linear equations is then solved using LU decomposition with partial pivoting.
   - **Updating Trait Frequencies:**
     - With the stationary distribution π computed, we calculate the frequency of each trait j as the expected presence of that trait across all transient states:
       $$f_j = \sum_{r \in \mathcal{T}} \pi_r \cdot r_j,$$
       where r_j indicates whether trait j is known in state r ($r_j = 1$) or not ($r_j = 0$).
3. **Final Transition Matrix Construction:**

- Using the updated trait frequencies f_j derived from the stationary distribution, we construct the final transition matrix P.
- This matrix now accurately captures the learning dynamics of the population, as the trait frequencies reflect the learners' state occupancies before absorption.

3 Outcome Variables

In this section, we describe how to compute the two key outcome variables of the learning model: the expected success rate of a learning attempt (expected transitions per step) and the expected payoff per learning event. Note that each known trait contributes its payoff at each step.

3.1 Expected Success Rate of a Learning Attempt

The expected success rate of a learning attempt quantifies the average number of successful learning transitions an agent accomplishes in each step within the learning process. To compute this measure, we proceed as follows:

1. **Fundamental Matrix**:

   Let Q be the transient submatrix of the transition matrix P, containing the probabilities of transitioning among transient (non-absorbing) states. The fundamental matrix N is defined as:

   $$N = (I - Q)^{-1}, \tag{1}$$

   where I is the identity matrix of appropriate size.

2. **Expected Steps Before Absorption**:

   The expected number of steps before reaching an absorbing state, starting from the initial transient state s_0, is given by:

   $$\text{Expected Steps} = t_{s_0}, \tag{2}$$

   where $\mathbf{t}$ is the vector of expected times to absorption, computed as:

   $$\mathbf{t} = N\mathbf{1}, \tag{3}$$

   and $\mathbf{1}$ is a column vector of ones.

3. **Occupancy Probabilities**:

   The expected number of times the process is in state s_i, starting from s_0, is given by the entry N_{s_0,s_i} of the fundamental matrix N. The occupancy probability π_i is then:

   $$\pi_i = \frac{N_{s_0,s_i}}{\text{Expected Steps}}. \tag{4}$$

4. **Self-Transition Probabilities**:

   The self-transition probability q_{ii} is the diagonal entry of the matrix Q corresponding to state s_i:

   $$q_{ii} = Q_{s_i,s_i}. \tag{5}$$

5. **Expected Self-Transition Probability**:

   The expected self-transition probability over all transient states is:

   $$\text{Expected Self-Transition Probability} = \sum_i \pi_i \cdot q_{ii}. \tag{6}$$

6. **Expected Transitions per Step**:

   The expected number of transitions per step (i.e., the expected success rate) is then:

$$\text{Expected Transitions per Step} = 1 - (\text{Expected Self-Transition Probability})\,. \tag{7}$$

This calculation reflects the average fraction of time steps in which the learner successfully transitions to a new state (learns a new trait), as opposed to remaining in the same state.

3.2 Expected Payoff per Learning Event

The expected payoff per learning event is defined as the expected cumulative payoff divided by the expected number of steps before reaching an absorbing state. To compute this metric, we follow these steps:

1. **Payoff Vector**:

   Let $\mathbf{p}$ be the vector of payoffs for the transient states, where the payoff for state s_i is the total payoff from all known traits in that state:

$$p_i = \sum_{j=0}^{n-1} r_{i,j} \cdot p_j, \tag{8}$$

   where $r_{i,j}$ indicates whether trait j is known in state s_i ($r_{i,j} = 1$) or not ($r_{i,j} = 0$), and p_j is the payoff of trait j.

2. **Expected Cumulative Payoff**:

   The expected cumulative payoff vector $\mathbf{u}$ starting from each transient state is obtained by solving the system:

$$(I - Q)\,\mathbf{u} = \mathbf{p}. \tag{9}$$

3. **Expected Cumulative Payoff from Initial State**:

   The expected cumulative payoff starting from the initial state s_0 is:

$$\text{Expected Cumulative Payoff} = u_{s_0}. \tag{10}$$

4. **Expected Payoff per Learning Event**:

   The expected payoff per learning event is calculated as:

$$\text{Expected Payoff per Event} = \frac{\text{Expected Cumulative Payoff}}{\text{Expected Steps}}. \tag{11}$$

This ratio represents the average payoff accumulated per time step during the learning process before absorption, adjusted for the expected duration of the learning process.

By using these matrix operations, we accurately compute the outcome variables based on the dynamics of the Markov process defined by the transition matrix P.