CS274B Project Report

Jay Phil Yoo
Department of Computer Science
University of California, Irvine
Irvine, CA 92697
jpyoo@uci.edu

Jimin Heo
Department of Computer Science
University of California, Irvine
Irvine, CA 92697
heoj4@uci.edu

Badrish Ananth
Department of Computer Science
University of California, Irvine
Irvine, CA 92697
bananth@uci.edu

Zhaobin Li
Department of Cognitive Science
University of California, Irvine
Irvine, CA 92697
zhaobil1@uci.edu

Arash Dehghani
Department of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, CA 92697
arashd1@uci.edu

1 Introduction

We replicate the paper by Ashwood et. al. [1] In perceptual decision-making, agents such as rodents and humans process sensory information to guide their choices. Traditional computational models, like signal detection theory [2] and evidence accumulation frameworks [3], have typically assumed that individuals apply a consistent decision-making strategy throughout an entire experiment. Recently, reinforcement learning (RL) models [4, 5] have challenged this notion by proposing that agents dynamically adapt their strategies to optimize performance over time.

Despite these advances, current models still struggle to account for unexpected errors that agents commit even in seemingly straightforward trials with strong perceptual cues. These errors, known as "lapses," are generally viewed as sporadic mistakes due to temporary lapses in attention or memory. Standard modeling approaches, such as epsilon-greedy RL, typically assume lapses occur randomly and independently, assigning them a fixed probability (epsilon) where agents ignore available information and make arbitrary choices.

However, lapses may not simply be random events; rather, they could reflect abrupt shifts in underlying decision-making strategies. For instance, rodents might alternate between an "engaged" state, characterized by consistently accurate performance, and a "disengaged" state, marked by increased lapses. To better capture these latent strategic shifts, we propose employing Hidden Markov Models (HMMs). Our hypothesis is that modeling decision-making using HMMs will more accurately reflect experimental behavior compared to traditional models.

This methodological innovation is significant because it provides a more nuanced understanding of the behavioral policies governing perceptual decision-making. By explicitly modeling latent states, HMMs can reveal the hidden structure underlying perceptual errors, showing that lapses reflect deeper strategic changes rather than isolated occurrences.

Identifying and segmenting these latent strategies will open promising avenues for future research, including:

Preprint. Under review.

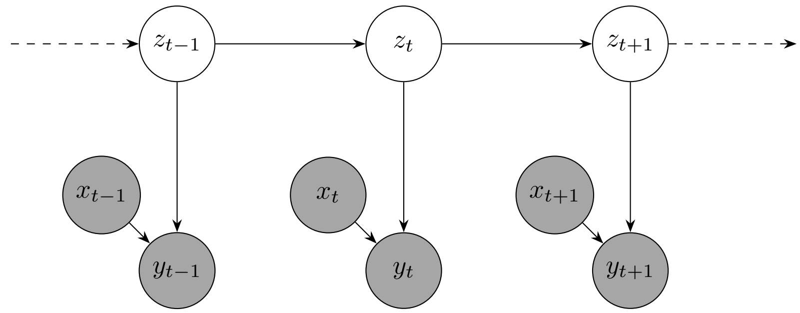

Figure 1: Proposed HMM model: gray nodes are observed and other nodes are hidden

1. Investigating the origins and adaptive functions of various decision-making strategies.
2. Exploring the neural mechanisms underlying distinct behavioral states.
3. Assessing whether these strategies represent bounded rationality or fluctuations in motivation.

2 Dataset

For this project, we will replicate the experiment conducted by the International Brain Laboratory (IBL) [6]. In this study, 140 mice participated in a visual decision-making task requiring them to turn a wheel to indicate whether a contrast grating (Gabor patch) appeared on the right or left side of the screen. The complete dataset from this experiment is publicly accessible, allowing us to rigorously test our proposed HMM approach against traditional models.

The dataset comprises behavioral recordings across multiple sessions for each mouse, which included the stimulus size and contrast level, the wheel turn direction (a decision made by the mouse), and the trial outcome. The stimuli varies in the amount of contrast, corresponding to the difficulty of the trial. The data also includes metadata on the structure of each session.

3 Methodology

This study models the decision-making process of mice, examining how lapses are influenced by latent cognitive states (engaged, biased left, biased right) that influence decisions. Given the temporal structure of perceptual decision-making and the role of hidden states, a hidden Markov model (HMM) is proposed as follows in Fig.1.

where z_t's are latent variables, x_t's are vectors of covariates and y_t's are variables showing actions. More specifically, the x_t in the HMM has 4 values:

1. The stimulus direction
2. A bias term
3. The mouse's choice in the previous trial
4. Whether the mice adhered to a win-stay lose-shift strategy in its previous trials [7]

The latent state distribution is modeled as a multinomial distribution over a arbitrary number of possible states K at time t, denoted as:

$$p(z_t = k) = \pi_k, \quad \sum_{k=1}^{3} \pi_k = 1, \tag{1}$$

where $K = 3$, and $k \in \{1, 2, 3\}$ represents the engaged, biased left, and biased right states. The state transition matrix A has a size of $K \times K$, specifying the hidden state transition from $k_{t-1} = j$ to $k_t = k$ with the Markov assumption.

$$p\left(z_t = k \mid z_{t-1} = j\right) = A_{jk} \tag{2}$$

We describe the distribution of y_t given the parent nodes as:

$$p(y_t = 1|x_t, z_t = k) = \frac{1}{1 + e^{-w_k \cdot x_t}}, \tag{3}$$

where $w_k \in \mathbb{R}^M$ denotes the GLM weights for latent state $k \in \{1, \ldots, K\}$. The transition matrix A is the set of weights representing the transitional probabilities from state z_t to z_{t+1}. The full set of parameters for a HMM would be $\Theta = \{\pi, \{w_k\}_1^K, A\}$.

The inference task optimizes the complete data log-likelihood via the maximum a posteriori (MAP) objective:

$$\hat{\Theta} = \arg\max_{\Theta} \log p(y_t, z_t \mid X, \Theta) + \log p(\Theta), \tag{4}$$

where X is the set of stimuli up to time t, y_t is the decision at time t, z_t is the latent state indicating the lapse, and Θ represents the parameters, including the state transition matrix A, weights w_k influencing choices in each state, and initial state distribution π.

The prior distribution is expressed as:

$$p(\Theta) = p(\{w_k\})p(A)p(\pi), \tag{5}$$

assuming independence among the weights, initial state distribution, and state transition matrix. The weights follow a normal distribution with mean 0 and variance σ^2, while A and π follow Dirichlet distributions with parameter α and α_π:

$$p(\Theta) = \prod_{k=1}^{K} \mathcal{N}(w_k; 0, \sigma^2 I) \cdot \mathrm{Dir}(\pi; \alpha_\pi) \cdot \mathrm{Dir}(A; \alpha). \tag{6}$$

Note that we will assume $\alpha_\pi = 1$ for prior distribution of π as it is assumed by the original paper[1].

EM Algorithm for Inference

An expectation-maximization (EM) algorithm calculates log-likelyhood of $p(z_t, y_t|\Theta)$ in the E-step and optimizes Θ in the M-step. The E-step employs the sum-product (forward-backward) algorithm to obtain the log-likelihood of current distribution, computing the posterior probability.

The M-step optimizes continuous parameters using the Broyden-Fletcher-Goldfarb-Shanno (BFGS). The learning behaviors of these optimizers are compared to enhance model performance. [1]

To estimate the parameters of the HMM, the Expectation-Maximization (EM) algorithm is used to estimate the model parameters by iteratively refining the latent state assignments z_t in the E-step and the model parameter Θ in the M-step. This involves finding the state transition matrix A and the weight vectors $\{w_k\}$ that influence choices in each latent state, and the initial latent state distribution π.

Expected Complete Log Likelihood

Let $y_{1:T}$ denote the observed data, $z_{1:T}$ the latent variables (e.g., HMM states), and Θ the model parameters, which include the transition matrix A, initial distribution π, and state-specific GLM weights $\{w_k\}_{k=1}^K$.

The EM algorithm seeks to maximize the (regularized) log-likelihood:

$$p(y_{1:T} \mid x_{1:T}, \Theta)p(\Theta) \tag{7}$$

Now taking the log:

$$\log p(y_{1:T} \mid x_{1:T}, \Theta) + \log p(\Theta) = \log \sum_{z_{1:T}} p(y_{1:T}, z_{1:T} \mid x_{1:T}, \Theta) + \log p(\Theta) \tag{8}$$

Introduce an auxiliary distribution $q(z_{1:T})$, often taken as the posterior under current parameters:

$$q(z_{1:T}) = p(z_{1:T} \mid y_{1:T}, x_{1:T}, \Theta^{\text{old}})$$

By Jensen's inequality and knowing that $\log$ is a concave :

$$\log \sum_{z_{1:T}} q(z_{1:T}) \cdot \frac{p(y_{1:T}, z_{1:T} \mid x_{1:T}, \Theta)}{q(z_{1:T})} \geq \sum_{z_{1:T}} q(z_{1:T}) \log \frac{p(y_{1:T}, z_{1:T} \mid x_{1:T}, \Theta)}{q(z_{1:T})}$$

This gives the Evidence Lower Bound (ELBO):

$$\mathcal{L}(\Theta, q) = \mathbb{E}_{q(z_{1:T})} \left[\log p(y_{1:T}, z_{1:T} \mid x_{1:T}, \Theta)\right] + H(q) + \log p(\Theta)$$

Here $H(q) = -\sum_{z_{1:T}} q(z_{1:T}) \log q(z_{1:T})$ is the entropy of the variational distribution,

By the Markov and emission assumptions, the joint log-likelihood of the observed and latent variables of one sequence factorizes as:

$$p(y_{1:T}, z_{1:T} \mid x_{1:T}, \Theta) = p(\Theta)p(z_1) \prod_{t=1}^{T-1} p(z_{t+1} \mid z_t) \prod_{t=1}^{T} p(y_t \mid x_t, z_t) \tag{9}$$

If we have i.i.d samples of data (sequences) with prior on Θ:

$$p(\Theta) \prod_{s=1}^{S} p(y_{s,1:T_s}, z_{s,1:T_s} \mid x_{s,1:T_s}, \Theta) = p(\Theta) \prod_{s=1}^{S} p(z_{s,1}) \prod_{t=1}^{T_s-1} p(z_{s,t+1} \mid z_{s,t}) \prod_{t=1}^{T_s} p(y_{s,t} \mid x_{s,t}, z_{s,t}) \tag{10}$$

Now, taking the logarithm:

$$\begin{aligned}\sum_{s=1}^{S} \log p(y_{s,1:T_s}, z_{s,1:T_s} \mid x_{s,1:T_s}, \Theta) = &\sum_{s=1}^{S} \log p(z_{s,1}) + \sum_{s=1}^{S} \sum_{t=1}^{T_s-1} \log p(z_{s,t+1} \mid z_{s,t}) \\ &+ \sum_{s=1}^{S} \sum_{t=1}^{T_s} \log p(y_{s,t} \mid x_{s,t}, z_{s,t})\end{aligned}$$

We now take the expectation of this quantity with respect to the posterior over the latent states, given the observations and the current parameter estimate Θ^{old}:

$$\mathbb{E}_{p(z_{1:T} \mid y_{1:T}, x_{1:T}, \Theta^{\text{old}})} \left[\log p(y_{1:T}, z_{1:T} \mid x_{1:T}, \Theta)\right]$$

This expected value is what forms the core of the E-step in the EM algorithm, and is often called the *Expected Complete Log-Likelihood (ECLL)*. Hence, after adding $\log$ priors, we have:

$$\begin{aligned}\text{ECLL}(\Theta) = &\mathbb{E}\left[\sum_{s=1}^{S} \log p(z_{s,1})\right] &(11)\\ &+ \mathbb{E}\left[\sum_{s=1}^{S} \sum_{t=1}^{T_s-1} \log p(z_{s,t+1} \mid z_{s,t})\right] &(12)\\ &+ \mathbb{E}\left[\sum_{s=1}^{S} \sum_{t=1}^{T_s} \log p(y_{s,t} \mid x_{s,t}, z_{s,t})\right] &(13)\\ &+ \log p(\Theta) &(14)\\ & &(15)\end{aligned}$$

All expectations are on posterior distribution as we mentioned previously. **We will rewrite the terms based on the probabilities we compute and terms we define in E-step.**

The goal is to maximize the log-posterior of the parameters given the observed choices and input data. Since computing the full posterior is infeasible due to the exponential number of latent state sequences, the EM algorithm approximates it through an iterative procedure of E-steps and M-step.

E-step:

The E-step estimate the posterior distribution over latent states using the forward–backward algorithm with fixed Θ. The objective is to yield $\gamma_{s,t,k}$: the probability of being in latent state k at time t in session s and $\xi_{s,t,j,k}$, the probability of transitioning from state i to state j between time t and $t+1$. Formally, for each trial t within session s and each state k, the forward pass message a is defined as

$$a_{s,t,k} \equiv p\left(\mathbf{y}_{s,[1:t]}, z_{s,t} \mid \{\mathbf{x}_{t'}\}_{t'=1}^{t}\right) \tag{16}$$

and the backward pass message b is defined as

$$b_{s,t,k} \equiv p\left(\mathbf{y}_{s,[t+1:T_s]} \mid z_{s,t} = k, \{\mathbf{x}_{t'}\}_{t'=t+1}^{T_s}\right) \tag{17}$$

. Therefore, the probability of being in latent state can be calculated by solving the combination of the forward pass message and the backward pass message as follows:

$$\begin{aligned}
\gamma_{s,t,k} &\equiv p\left(z_{s,t} = k \mid \mathbf{y}_s, \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}}\right) \\
&= \frac{p\left(\mathbf{y}_{s,[0:t]}, z_{s,t} = k \mid \{\mathbf{x}_{s,t}\}_{t'=1}^{t}, \Theta^{\text{old}}\right) p\left(\mathbf{y}_{s,[t+1:T_s]} \mid z_{s,t} = k, \{\mathbf{x}_{s,t'}\}_{t'=t+1}^{T_s}, \Theta^{\text{old}}\right)}{p\left(\mathbf{y}_s \mid \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}}\right)} \\
&= \frac{a_{s,t,k} b_{s,t,k}}{\sum_{k=1}^{k} a_{s,T_{s,k}}}.
\end{aligned} \tag{18}$$

Similarly, the probability of transitioning from state j to state k between time t and $t+1$, $\xi_{s,t,i,j}$, can be expressed incoorporating the message terms.

$$\xi_{s,t,j,k} \equiv p(z_{s,t} = j, z_{s,t+1} = k \mid \mathbf{y}_s, \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}})$$

Using Bayes' theorem, this can be expressed as the ratio of the joint probability to the marginal probability of the observations and given the HMM structure, the joint probability can be factored due to conditional independence of observations given the states and inputs.

$$\begin{aligned}
\xi_{s,t,j,k} &\equiv p(z_{s,t} = j, z_{s,t+1} = k \mid \mathbf{y}_s, \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}}) \\
&= \frac{p(z_{s,t} = j, z_{s,t+1} = k, \mathbf{y}_s \mid \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}})}{p(\mathbf{y}_s \mid \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}})} \\
&= \frac{\begin{aligned} &p(\mathbf{y}_{s,[1:t]}, z_{s,t} = j \mid \{\mathbf{x}_{s,t'}\}_{t'=1}^{t}, \Theta^{\text{old}}) \\ &\quad \cdot p(z_{s,t+1} = k \mid z_{s,t} = j, \Theta^{\text{old}}) \\ &\quad \cdot p(\mathbf{y}_{s,t+1} \mid z_{s,t+1} = k, \mathbf{x}_{s,t+1}, \Theta^{\text{old}}) \\ &\quad \cdot p(\mathbf{y}_{s,[t+2:T_s]} \mid z_{s,t+1} = k, \{\mathbf{x}_{s,t'}\}_{t'=t+1}^{T_s}, \Theta^{\text{old}}) \end{aligned}}{p(\mathbf{y}_s \mid \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}})} \\
&= \frac{a_{s,t,j} A_{jk} b_{s,t+1,k} p(\mathbf{y}_{s,t+1} \mid z_{s,t+1} = k, \mathbf{x}_{s,t+1}, \mathbf{w}_k)}{\sum_{k=1}^{K} a_{s,T_s,k}}
\end{aligned} \tag{19}$$

The E-step uses these probabilities to compute the expected log-likelihood of the current distribution, which is maximized in the subsequent M-step to update Θ^{old}.

M-step:

In M-step, we compute parameters of the model by maximizing ECLL.

First, we need to rewrite the terms of ECLL. For the first term 11, the expectation would be simply on posterior of z_1:

$$\begin{aligned}
\mathbb{E}\left[\sum_{s=1}^{S} \log p(z_{s,1})\right] &= \sum_{s=1}^{S} \sum_{k=1}^{K} p\left(z_{s,1} = k \mid \mathbf{y}_s, \{\mathbf{x}_{s,t}\}_{t=1}^{T_s}, \Theta^{\text{old}}\right) \log p(z_{s,1} = k) \\
&= \sum_{s=1}^{S} \sum_{k=1}^{K} \gamma_{s,1,k} \log \pi_k
\end{aligned}$$

In the second line, we simply substitute using the definition of parameters.

For the second term 12, the expectation for each pair would be on the joint posterior distribution:

$$\begin{aligned}\mathbb{E}\left[\sum_{s=1}^{S}\sum_{t=1}^{T_s-1}\log p(z_{s,t+1}\mid z_{s,t})\right] &= \sum_{s=1}^{S}\sum_{t=1}^{T_s-1}\sum_{k=1}^{K}\sum_{j=1}^{K} p(z_{s,t}=j, z_{s,t+1}=k\mid \mathbf{y}_s,\{\mathbf{x}_{s,t}\}_{t=1}^{T_s},\Theta^{\text{old}})\times\\ &\log p(z_{s,t+1}=j\mid z_{s,t}=k)\\ &= \sum_{s=1}^{S}\sum_{t=1}^{T_s-1}\sum_{k=1}^{K}\sum_{j=1}^{K}\xi_{s,t,j,k}\log A_{jk}\end{aligned}$$

In the second line, we simply substitute using the definition of parameters.

For the third term, we have the logistic regression involved. For logistic regression we have

$$p(y_{s,t}|x_{s,t},z_{s,t}=k) = (\frac{1}{1+e^{-w_k\cdot x_{s,t}}})^{y_{s,t}}\cdot(\frac{e^{-w_k\cdot x_{s,t}}}{1+e^{-w_k\cdot x_{s,t}}})^{(1-y_{s,t})}$$

Hence:

$$\begin{aligned}\log p(y_{s,t}\mid x_{s,t},z_{s,t}) &= -y_{s,t}\log(1+e^{-w_k\cdot x_{s,t}}) + (1-y_{s,t})\log(e^{-w_k\cdot x_{s,t}}) - (1-y_{s,t})\log(1+e^{-w_k\cdot x_{s,t}})\\ &= y_{s,t}w_k^T x_{s,t} + \log(\frac{e^{-w_k\cdot x_{s,t}}}{1+e^{-w_k\cdot x_{s,t}}})\\ &= y_{s,t}w_k^T x_{s,t} - \log(1+e^{-w_k^T x_{s,t}})\end{aligned}$$

So, we can rewrite the third term 13 as follows:

$$\begin{aligned}\mathbb{E}\left[\sum_{s=1}^{S}\sum_{t=1}^{T_s}\log p(y_{s,t}\mid x_{s,t},z_{s,t})\right] &= \sum_{s=1}^{S}\sum_{t=1}^{T_s}\sum_{k=1}^{K} p\left(z_{s,t}=k\mid \mathbf{y}_s,\{\mathbf{x}_{s,t}\}_{t=1}^{T_s},\Theta^{\text{old}}\right)\log p(y_{s,t}\mid x_{s,t},z_{s,t})\\ &= \sum_{s=1}^{S}\sum_{t=1}^{T_s}\sum_{k=1}^{K}\gamma_{s,t,k}\left[y_{s,t}(w_k^\top x_{s,t} - \log\left(1+e^{w_k^\top x_{s,t}}\right)\right]\end{aligned}$$

For the regularization term. Since $\mathrm{Dir}(\mathbf{p}\mid\alpha)\propto\prod_{i=1}^{K}p_i^{\alpha-1}$ and $\log(\mathcal{N}(w_k;0,\sigma^2 I))\propto e^{-\frac{1}{\sigma^2}w_k^T w_k}$:

$$\begin{aligned}\log p(\Theta) &= \log(\prod_{k=1}^{K}\mathcal{N}(w_k;0,\sigma^2 I)\cdot\cdot\mathrm{Dir}(A;\alpha))\\ &= \sum_{k=1}^{K}\log(\mathcal{N}(w_k;0,\sigma^2 I)) + \log(\mathrm{Dir}(A;\alpha)))\\ &= -\sum_{k=1}^{K}\frac{1}{\sigma^2}w_k^T w_k + (\alpha-1)\sum_{j=1}^{K}\sum_{k=1}^{K}\log A_{jk}\end{aligned}$$

Note that as we mentioned the paper considered $\alpha_\pi = 1$, so we consider no prior for π.

Now, we bring all term together for ECLL(as well as the regularization term):

$$\begin{aligned}\mathrm{ECLL}(\Theta) = &\sum_{s=1}^{S}\sum_{k=1}^{K}\gamma_{s,1,k}\log\pi_k &(20)\\ &+\sum_{s=1}^{S}\sum_{t=1}^{T_s-1}\sum_{k=1}^{K}\sum_{j=1}^{K}\xi_{s,t,j,k}\log(A_{jk}) + (\alpha-1)\sum_{j=1}^{K}\sum_{k=1}^{K}\log(A_{jk}) &(21)\\ &+\sum_{s=1}^{S}\sum_{t=1}^{T_s}\sum_{k=1}^{K}\gamma_{s,t,k}\left[y_{s,t}(w_k^\top x_{s,t} - \log\left(1+e^{w_k^\top x_{s,t}}\right)\right] - \sum_{k=1}^{K}\frac{1}{\sigma^2}w_k^T w_k &(22)\end{aligned}$$

We should find a Θ that maximizes ECLL.

- π: Only 20 depends on π:

$$\max_{\pi} \sum_{s=1}^{S} \sum_{k=1}^{K} \gamma_{s,1,k} \log \pi_k \quad \text{subject to} \quad \sum_{k=1}^{K} \pi_k = 1, \quad \pi_k \geq 0$$

Using Lagrangian multipliers the solution would be :

$$\pi_k = \frac{\sum_{s=1}^{S} \gamma_{s,1,k}}{\sum_{s=1}^{S} \sum_{k=1}^{K} \gamma_{s,1,k}} \tag{23}$$

- A: Only 21 depends on A:

$$\max_{A} \sum_{k=1}^{K} \sum_{j=1}^{K} (\sum_{s=1}^{S} \sum_{t=1}^{T_s-1} \xi_{s,t,j,k} + \alpha - 1) \log A_{jk} \quad \text{subject to} \quad \sum_{k=1}^{K} A_{jk} = 1, \quad A_{jk} \geq 0 \quad \text{for each } j$$

Using Lagrangian multipliers, similar to π, we have:

$$A_{jk} = \frac{\alpha - 1 + \sum_{s=1}^{S} \sum_{t=1}^{T_s-1} \xi_{s,t,j,k}}{K(\alpha - 1) + \sum_{k=1}^{K} \sum_{s=1}^{S} \sum_{t=1}^{T_s-1} \xi_{s,t,j,k}} \tag{24}$$

- W: Only 22 depends on W. we need to maximize the following function with no constraints:

$$\sum_{s=1}^{S} \sum_{t=1}^{T_s} \sum_{k=1}^{K} \gamma_{s,t,k} \left[y_{s,t} (w_k^\top x_{s,t} - \log \left(1 + e^{w_k^\top x_{s,t}} \right) \right] - \sum_{k=1}^{K} \frac{1}{\sigma^2} w_k^T w_k \tag{25}$$

Each EM iteration increases the log-posterior. Hyperparameters such as σ^2 (Gaussian prior variance for weights) and α (Dirichlet prior for transition probabilities) are selected via grid search based on validation set performance.

Summary and Implementation

The E-step computes probabilities γ and ξ using the Θ of the last M-step. Then, M-step calculates π and A using the formulas which we derived and γ and ξ. After that, by maximizing the objective function 25, we calculate W. For optimization we used `scipy.optimize.minimize()` and 'BFGS' algorithm as it was preferred in the paper.

The BFGS algorithm (Broyden–Fletcher–Goldfarb–Shanno) is a popular quasi-Newton optimization method used to find local maxima or minima of differentiable functions, especially when second-order derivatives (Hessians) are expensive to compute or store. In some cases, the algorithm is better than other well-known methods in fast convergence.

And before integrating different parts of the code, we developed a HMM sampler to sample z, y using the distribution in the assumption and parameter Θ to check if M-step outputs the expected Θ parameters.

4 Experiment

We fitted the EM algorithm for one mice across 5040 trials. The number of latent states is set to 2, 3, and 4 and the number of initializations is set to 3. These values are selected to allow the code to run on a laptop. We used 2-fold cross validation to select the best run with the highest log-likelihood on the validation fold.

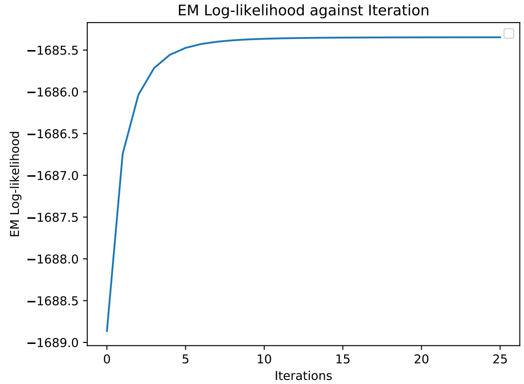

Figure 2: Log-likelihood against iterations.

We stopped the EM algorithm when the change in log-likelihood is less than 10^{-4}. To validate the EM algorithm we implemented, we plotted the log-likelihood across iterations (Figure 2). The log-likelihood increased consistently across iterations. We also checked the GLM weights and transition matrix against those generated by the original code (Figure 9) and both matches within 1%.

5 Results

Our implementation of the EM algorithm achieved results consistent with the original study. Figure 3 illustrates that increasing the number of latent states enhances the test log-likelihood. However, because the improvement in test log-likelihood from 3 to 4 latent states is minimal and accompanied by greater complexity, we focused our subsequent analyses on a 3-state model.

Figure 4 visualizes the transition matrix, demonstrating that mice typically remain within their current latent state but retain a small probability of transitioning to other states.

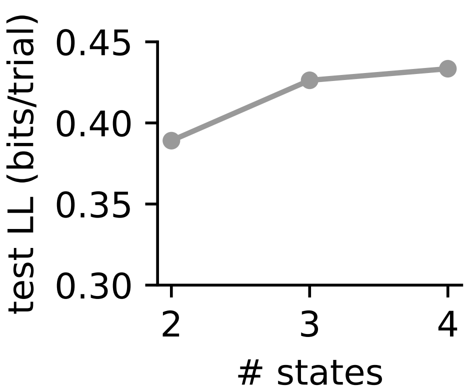

Figure 3: Test log-likelihood for GLM-HMMs with different latent states

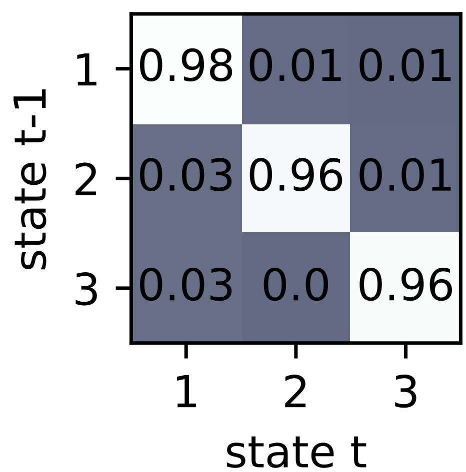

Figure 4: Transition matrix for fitting a 3-state GLM-HMM

Figure 5 presents the GLM weights for each latent state, highlighting the influence of covariates on mice responses in each trial. The orange weights corresponding to state 1 indicate sensitivity to the stimulus, characterizing an engaged state where mice exhibit optimal task performance and highest accuracy, as evidenced in Figure 6. Conversely, states 2 and 3, represented by green and blue weights respectively, correspond to left- and right-biased decision-making states where mice exhibit stimulus insensitivity. Across all three states, the covariates representing previous choices and win-stay lose-shift strategies showed minimal impact on decision-making processes.

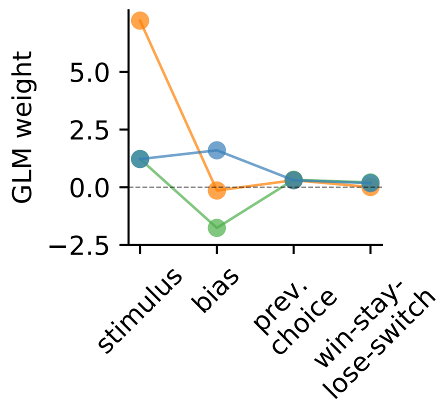

Figure 5: Visualization of the inferred GLM weights for each state

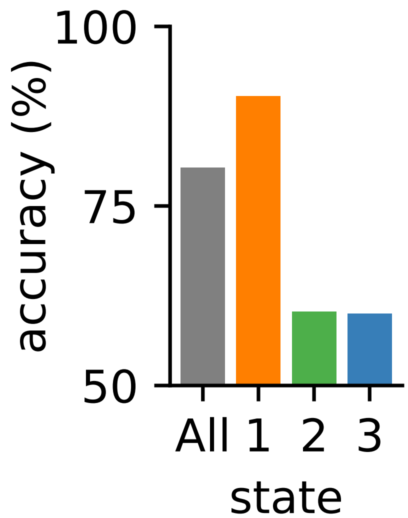

Figure 6: Accuracy for each latent state

Additionally, Figure 7 depicts logistic curves representing the probability of a rightward wheel movement as a function of stimulus strength, further elucidating behavioral tendencies across latent states. The engaged state demonstrates no directional bias in the absence of stimulus, whereas the left- and right-biased states clearly show directional preferences.

Figure 8 visualizes example sessions, plotting the posterior distribution of latent states over time across three representative sessions. These plots confirm that the model assigns high probabilities to latent states, effectively capturing behavioral dynamics. Furthermore, these latent state assignments support the prediction that task accuracy is significantly higher in the engaged state compared to biased states.

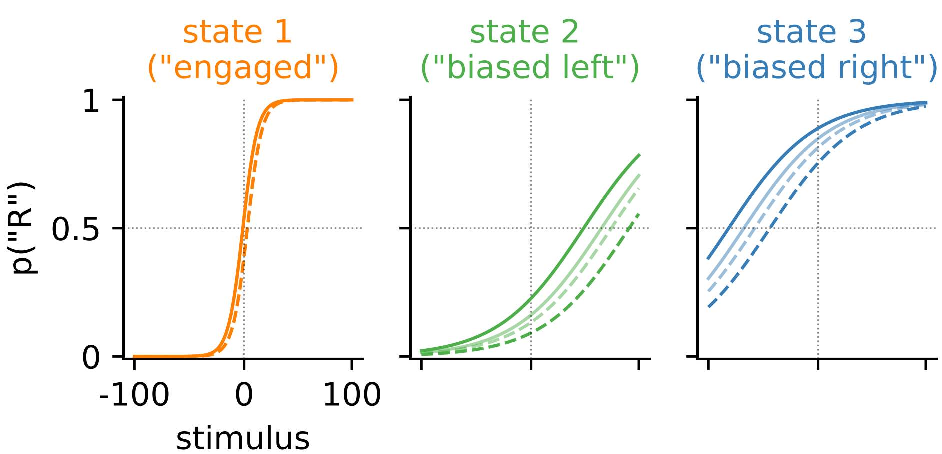

Figure 7: Probability that mice choose right based on latent state and stimulus

6 Conclusion

In this study, we successfully replicated and extended the findings presented by Ashwood et al. [1], confirming that GLM-HMM models effectively capture discrete latent states underlying decision-making behavior in mice. Our implementation of the EM algorithm robustly identified latent states

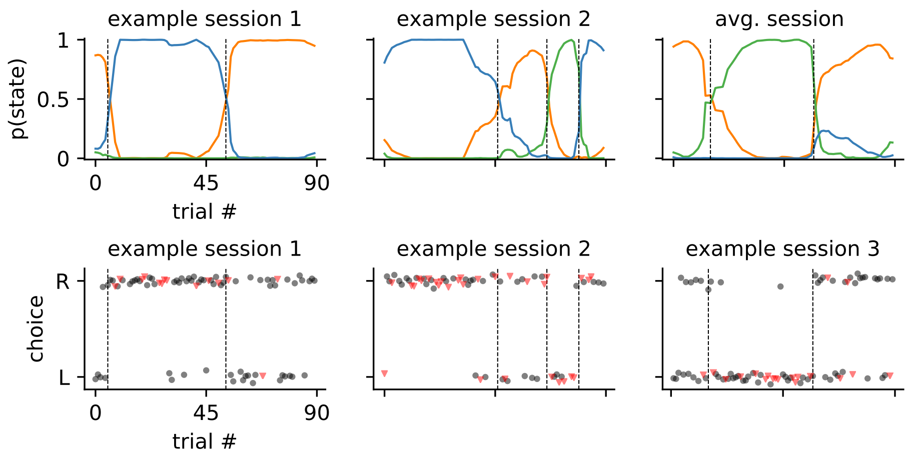

Figure 8: Posterior probabilities of latent states across multiple trials for three example sessions

and accurately reflected their dynamics. Specifically, our analysis demonstrated that perceptual lapses are not merely random events but rather indicative of distinct shifts in internal cognitive strategies. The state-specific GLM weights and transition probabilities provided meaningful insights into how sensory input, inherent biases, and previous decisions collectively shape mice's decision-making behaviors.

Our results underscore the advantage of employing GLM-HMMs over traditional modeling approaches by offering a nuanced understanding of behavioral states and strategic transitions. This model's capability to identify and interpret latent states opens promising avenues for further exploration into the neural mechanisms and adaptive functions underpinning perceptual decision-making. Future research may build on these findings by examining whether such state transitions reflect bounded rationality, motivational fluctuations, or other adaptive cognitive processes.

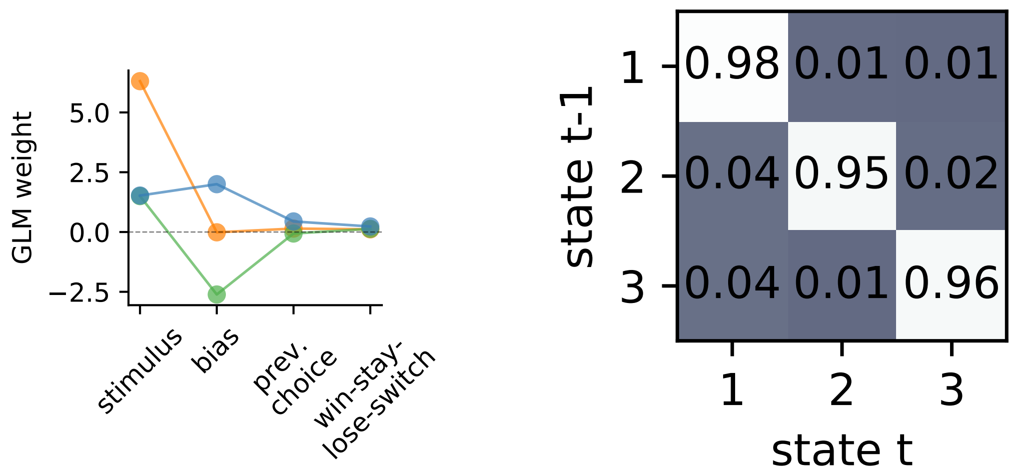

Figure 9: GLM weights and transition matrix based on the original code

7 Individual Contributions

7.1 Badrish

Wrote dataset section, results section, and helped integrate M-step into original code.

7.2 Jay

Wrote E-step section, coded up E-step, debugged and integrated M-step code into original code.

7.3 Zhaobin

Wrote introduction and experiment sections, make M-step compatible with original code, adapted code to ran all experiments and plotted all graphs

7.4 Arash

Wrote M-step section and theory of M-step and ECLL, implemented M-step and a HMM sampler with model to debug M-step.

7.5 Jimin

Helped a bit with everything.

8 Code

The code is available on github at `https://github.com/bluija/GMM`. We adapted the code in `https://github.com/zashwood/glm-hmm` and implemented a custom EM algorithm in `https://github.com/bluija/GMM/blob/main/2_fit_models/fit_individual_glmhmm%20em/glm_hmm_utils.py`

To enable the code to run on a laptop, we also tweaked the hyperparameters. The instructions to run the code are in the original README at `https://github.com/zashwood/glm-hmm/blob/main/README.md`. The code is also printed after the bibliography.

References

[1] Zoe C Ashwood et al. "Mice alternate between discrete strategies during perceptual decision-making". In: *Nature Neuroscience* 25.2 (2022), pp. 201–212.

[2] Thomas Steckler. "Using signal detection methods for analysis of operant performance in mice". In: *Behavioural brain research* 125.1-2 (2001), pp. 237–248.

[3] Onyekachi Odoemene et al. "Visual evidence accumulation guides decision-making in unrestrained mice". In: *Journal of Neuroscience* 38.47 (2018), pp. 10143–10155.

[4] Gediminas Luksys, Wulfram Gerstner, and Carmen Sandi. "Stress, genotype and norepinephrine in the prediction of mouse behavior using reinforcement learning". In: *Nature neuroscience* 12.9 (2009), pp. 1180–1186.

[5] Brice Bathellier et al. "A multiplicative reinforcement learning model capturing learning dynamics and interindividual variability in mice". In: *Proceedings of the National Academy of Sciences* 110.49 (2013), pp. 19950–19955.

[6] International Brain Laboratory et al. "Standardized and reproducible measurement of decision-making in mice". In: *Elife* 10 (2021), e63711.

[7] Martin Nowak and Karl Sigmund. "A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game". In: *Nature* 364.6432 (1993), pp. 56–58.

Listing 1: hmm.py

```
def m_step(expectations, datas, inputs, regularizer=1.0):
    """
    M-step for GLM-HMM that calculates parameters from scratch.

    Parameters
    ----------
    expectations : list of tuples (Ez, Ezzp1, normalizer)
        - Ez      : (T, K) expected state responsibilities
        - Ezzp1   : (T, K, K) expected state transitions
        - normalizer : float (unused)

    datas : list of ndarray (T,)
        - Binary output labels (choices), one array per session

    inputs : list of ndarray (T, M)
        - Feature matrix (design matrix) for each session

    regularizer : float
        - L2 regularization coefficient (Gaussian prior on weights)

    Returns
    -------
    dict with keys:
        'pi' : ndarray (K,)
            - Updated initial state distribution
        'A' : ndarray (K, K)
            - Updated transition matrix
        'w' : ndarray (K * M,)
            - Flattened GLM weights for each state
    """
    # Infer K and M from inputs and expectations
    Ez_sample, Ezzp1_sample, _ = expectations[0]
    K = Ez_sample.shape[1]
    M = inputs[0].shape[1]

    # --- Update initial state distribution ---
    pi_numer = np.zeros(K)
    for Ez, _, _ in expectations:
        pi_numer += Ez[0]
    pi_new = pi_numer / pi_numer.sum()

    # --- Update transition matrix ---
    A_num = np.zeros((K, K))
    A_den = np.zeros(K)
    for _, Ezzp1, _ in expectations:
        E = Ezzp1.squeeze()
        A_num += E
        A_den += E.sum(axis=1)
    A_new = A_num / A_den[:, None]

    # --- Update GLM weights ---
    def neg_ECLL(w_flat):
        w = w_flat.reshape((K, M))
        loss = 0
        for (Ez, _, _), x, y in zip(expectations, inputs, datas):
            for k in range(K):
                logits = x @ w[k]
                log_prob = y * logits - np.log1p(np.exp(logits))
                loss += np.sum(Ez[:, k] * log_prob)
        loss -= 0.5 * regularizer * np.sum(w ** 2)
        return -loss

    def grad_neg_ECLL(w_flat):
```

```
        w = w_flat.reshape((K, M))
        grad = np.zeros_like(w)
        for (Ez, _, _), x, y in zip(expectations, inputs, datas):
            for k in range(K):
                logits = x @ w[k]  # (T,)
                probs = 1 / (1 + np.exp(-logits))  # (T,)
                # ensure Ez_k is a 1-D array of length T
                grad[k] += (Ez[:, k][:, None] * (y.squeeze() - probs)[:, None] * x).sum(axis=0)

        grad -= regularizer * w
        return -grad.flatten()

    # Initialize weights
    w0 = np.random.randn(K * M) * 0.1
    res = minimize(neg_ECLL, w0, jac=grad_neg_ECLL, method='BFGS')
    w_new = res.x

    return {'pi': pi_new, 'A': A_new, 'w': w_new}


# Bind custom functions
def fit_glm_hmm(datas, inputs, masks, K, D, M, C, N_em_iters,
                transition_alpha, prior_sigma, global_fit,
                params_for_initialization, save_title):
    '''
    Instantiate and fit GLM-HMM model
    :param datas:
    :param inputs:
    :param masks:
    :param K:
    :param D:
    :param M:
    :param C:
    :param N_em_iters:
    :param global_fit:
    :param glm_vectors:
    :param save_title:
    :return:
    '''
    if global_fit == True:
        # Prior variables
        # Choice of prior
        this_hmm = ssm.HMM(K,
                           D,
                           M,
                           observations="input_driven_obs",
                           observation_kwargs=dict(C=C,
                                                   prior_sigma=prior_sigma),
                           transitions="sticky",
                           transition_kwargs=dict(alpha=transition_alpha,
                                                  kappa=0))
        # Initialize observation weights as GLM weights with some noise:
        glm_vectors_repeated = np.tile(params_for_initialization, (K, 1, 1))
        glm_vectors_with_noise = glm_vectors_repeated + np.random.normal(
            0, 0.2, glm_vectors_repeated.shape)
        this_hmm.observations.params = glm_vectors_with_noise
    else:
        # Choice of prior
        this_hmm = ssm.HMM(K,
```

```python
                           D,
                           M,
                           observations="input_driven_obs",
                           observation_kwargs=dict(C=C,
                                                   prior_sigma=prior_sigma),
                           transitions="sticky",
                           transition_kwargs=dict(alpha=transition_alpha,
                                                  kappa=0))
        # Initialize HMM-GLM with global parameters:
        this_hmm.params = params_for_initialization
        # Get log_prior of transitions:
    print("=== fitting GLM-HMM ========")
    sys.stdout.flush()

    # # Bind custom expected_states and _fit_em to this_hmm
    this_hmm.expected_states = types.MethodType(expected_states, this_hmm)
    this_hmm._fit_em = types.MethodType(_fit_em, this_hmm)

    # Fit this HMM and calculate marginal likelihood
    lls = this_hmm.fit(datas,
                       inputs=inputs,
                       masks=masks,
                       method="em",
                       num_iters=N_em_iters,
                       initialize=False,
                       tolerance=10 ** -4)
    # Save raw parameters of HMM, as well as loglikelihood during training
    np.savez(save_title, this_hmm.params, lls)
    return None


def expected_states(self, data, input=None, mask=None, tag=None):
    """
    Compute expected states using the forward-backward algorithm.

    Parameters:
    - data: (T,) array of binary observations (0 or 1)
    - input: (T, D) array of inputs
    - mask: (T,) array of 1s (valid) or 0s (invalid)
    - tag: optional tag for multiple sequences

    Returns:
    - gamma: (T, K) array of posterior probabilities P(z_t | y)
    - xi: (T-1, K, K) or (1, K, K) array of joint posterior probabilities P(z_t, z_{t+1} | y)
    - loglik: scalar log likelihood
    """
    T = len(data)
    pi0 = self.init_state_distn.initial_state_distn
    Ps = self.transitions.transition_matrices(data, input, mask, tag)
    log_likes = self.observations.log_likelihoods(data, input, mask, tag)

    # Check if transition matrix is stationary
    stationary = Ps.shape[0] == 1

    # Compute log transition matrices
    with np.errstate(divide="ignore"):
        log_Ps = np.log(Ps + 1e-10)

    # Forward pass
```

```python
    alpha = np.zeros((T, self.K))
    log_p_y = np.zeros(T)
    alpha[0] = np.log(pi0 + 1e-10) + log_likes[0]
    log_p_y[0] = logsumexp(alpha[0])

    # Commented forward pass normalizer
    # alpha[0] -= log_p_y[0]

    for t in range(1, T):
        for k in range(self.K):
            alpha[t, k] = log_likes[t, k] + logsumexp(alpha[t - 1] +
                log_Ps[min(t - 1, log_Ps.shape[0] - 1), :, k])
        log_p_y[t] = logsumexp(alpha[t])

        # Commented forward pass normalizer
        # alpha[t] -= log_p_y[t]

    # Backward pass
    beta = np.zeros((T, self.K))
    beta[-1] = 0
    for t in range(T - 2, -1, -1):
        for k in range(self.K):
            beta[t, k] = logsumexp(log_Ps[min(t, log_Ps.shape[0] - 1),
                 k, :] + log_likes[t + 1, :] + beta[t + 1, :])

    # Compute gamma (expected_states)
    gamma = alpha + beta
    gamma -= logsumexp(gamma, axis=1, keepdims=True)
    gamma = np.exp(gamma)

    # Compute xi (expected_joints)
    if stationary:
        xi = np.zeros((1, self.K, self.K))
        log_xi = np.zeros((1, self.K, self.K))
        for t in range(T - 1):
            for i in range(self.K):
                for j in range(self.K):
                    log_xi[0, i, j] += np.exp(
                        alpha[t, i] + log_Ps[0, i, j] + log_likes[t +
                            1, j] + beta[t + 1, j] - log_p_y[-1])
        xi[0] = log_xi[0] / np.sum(log_xi[0])  # Normalize
    else:
        xi = np.zeros((T - 1, self.K, self.K))
        for t in range(T - 1):
            for i in range(self.K):
                for j in range(self.K):
                    xi[t, i, j] = alpha[t, i] + log_Ps[t, i, j] +
                        log_likes[t + 1, j] + beta[t + 1, j]
            xi[t] -= logsumexp(xi[t])  # Normalize
            xi[t] = np.exp(xi[t])  # Convert to probability

    # log-likelihood
    loglik = log_p_y[-1]  # Normalizer from forward pass

    return gamma, xi, loglik


# The original code.
def _fit_em(self, datas, inputs, masks, tags, verbose=2, num_iters
    =100, tolerance=0,
            init_state_mstep_kwargs={}, transitions_mstep_kwargs={},
                observations_mstep_kwargs={}, **kwargs):
    """
    Fit the parameters with expectation maximization.

```

```python
        Parameters:
        - datas: list of observation arrays
        - inputs: list of input arrays
        - masks: list of mask arrays
        - tags: list of tags
        - verbose: verbosity level
        - num_iters: maximum number of EM iterations
        - tolerance: convergence tolerance
        """
        lls = [self.log_probability(datas, inputs, masks, tags)]
        pbar = ssm_pbar(num_iters, verbose, "LP: {:.1f}", [lls[-1]])

        for itr in pbar:
            # E step: compute expected latent states with current parameters
            expectations = [self.expected_states(data, input, mask, tag)
                            for data, input, mask, tag
                            in zip(datas, inputs, masks, tags)]

            # M step: maximize expected log joint wrt parameters

            mstep_results = m_step(expectations, datas, inputs, regularizer=1.0)

            self.init_state_distn.log_pi0 = np.log(mstep_results['pi'] + 1e-8)

            self.transitions.log_Ps = np.log(mstep_results['A'] + 1e-8)

            self.observations.w = mstep_results['w'].reshape(self.K, -1)

            # self.init_state_distn.m_step(expectations, datas, inputs, masks, tags, **init_state_mstep_kwargs)
            # self.transitions.m_step(expectations, datas, inputs, masks, tags, **transitions_mstep_kwargs)
            # self.observations.m_step(expectations, datas, inputs, masks, tags, **observations_mstep_kwargs)

            # Store progress
            lls.append(self.log_prior() + sum([ll for (_, _, ll) in expectations]))

            if verbose == 2:
                pbar.set_description("LP: {:.1f}".format(lls[-1]))

            # Check for convergence
            if itr > 0 and abs(lls[-1] - lls[-2]) < tolerance:
                if verbose == 2:
                    pbar.set_description("Converged to LP: {:.1f}".format(lls[-1]))
                break

        return lls
```