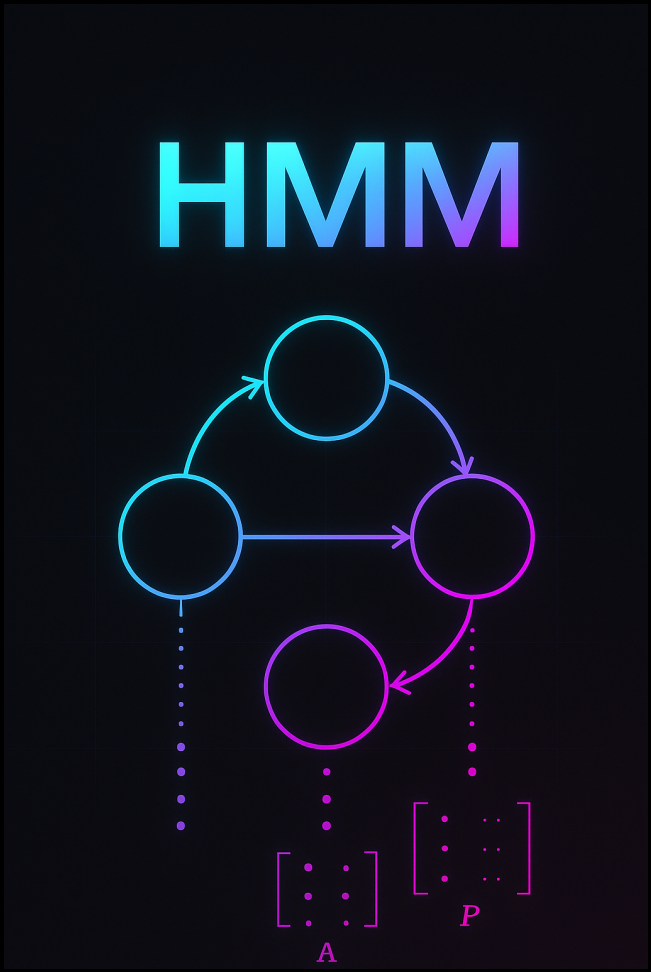


# Assignment Hidden Markov Models

Ahmet Enis Isik, Francesco Baldassarre
aeisik@kth.se, frabal@kth.se

October 22, 2025

# Contents

# 1 Introduction

This project implements the four core HMM building blocks in sequence: next-step emission prediction (HMM0), sequence likelihood via the forward algorithm (HMM1), most-likely state path via Viterbi (HMM2), and parameter learning with Baum-Welch/EM (HMM3). Across tasks, the design follows the standard discrete HMM formulation with hidden states $X_t$ and emissions $O_t$, and the classic trio of HMM problems: evaluation, decoding, and learning.

# 2 Exercise 1: HMM 0 - Next Observation Distribution

## 2.1 Key Design Decisions

- **Row-vector multiplies, no generic matmul.** We compute $\pi A$ and then $(\pi A)B$ with explicit $1 \times N$ by $N \times M$ routines. This mirrors the textbook prediction step and avoids transposes/intermediate buffers.
- **Plain Python lists.** In line with the specification meaning no Numpy was used, this way indexing was kept transparent and input/output formatting was kept simple.
- **Stable, compact printing.** Trailing zeros are trimmed to keep output readable while preserving sufficient precision.

## 2.2 Code Explanation

### 2.2.1 Main Components

- `read_matrix_line()`: parses one "matrix line" into a list-of-rows.
- `row_times_matrix(row, mat, out_cols)`: computes $1 \times N$ times $N \times M$ to produce a $1 \times M$ row.
- `fmt(x)`: float cleanup for formatting.

### 2.2.2 Logic Flow

- Parse $A, B, \pi$.
- Compute state prediction $\pi' = \pi A$.
- Compute emission prediction $e = (\pi A)B$.
- Print a single $1 \times M$ line (rows, cols, values).

**Answers to Exercise Questions:**

**Q2.** $\pi A$ is the predicted state distribution at $t + 1$.

**Q3.** $(\pi A)B$ is the next-step emission distribution.

# 3 Exercise 2: HMM 1 - Probability of the Observation Sequence

## 3.1 Key Design Decisions

- **Forward ($\alpha$-pass) as in theory.** We implement $\alpha_t(i) = b_i(o_t) \sum_j \alpha_{t-1}(j) a_{j,i}$ and return $P(O_{1:T}) = \sum_i \alpha_T(i)$. Only the current $\alpha$ row is kept (the scalar likelihood is sufficient).
- **Robust parsing.** Observation tokens like `8` or `8.0` are accepted to match variants in sample files.

## 3.2 Code Explanation

### 3.2.1 Main Components

- `alpha` (size $N$): current forward probabilities.
- `fmt()`: scalar likelihood printing with trimmed zeros.

### 3.2.2 Logic Flow

- Read $A, B, \pi$ and the observation IDs.
- Initialize $\alpha_1(i) = \pi_i \, b_i(o_1)$.
- For each $t$: update via the inner sum over predecessors and multiply by $b_i(o_t)$.
- Sum the final $\alpha_T$ and print.

**Answer to Exercise Question:**

**Q4.** Because of the HMM's conditional independence, $P(O_t \mid X_t, O_{1:t-1}, X_{1:t-1}) = P(O_t \mid X_t)$. Hence, conditioning on $X_t$ lets us replace $O_{1:t}$ by $O_t$ inside the emission factor.

# 4 Exercise 3: HMM 2 – Estimate Sequence of States

## 4.1 Key Design Decisions

- **Viterbi with explicit backpointers.** We compute $\delta_t(i) = \max_j \delta_{t-1}(j) a_{j,i} \, b_i(o_t)$ and record $\psi_t(i) = \arg\max_j$ to reconstruct the path. This is the canonical decoding dynamic program.
- **Memory-lean $\delta$.** Only the current $\delta$ row is maintained; the full backpointer grid is stored for traceback.

## 4.2 Code Explanation

### 4.2.1 Main Components

- `delta` (size $N$): best path scores at time $t$.
- `backptr` (T×N): argmax predecessor indices.
- Final selection uses $\arg\max_i \delta_T(i)$, then reverse traversal via `backptr`.

### 4.2.2 Logic Flow

- Initialize $\delta_1(i) = \pi_i\, b_i(o_1)$.
- For each $t$: for every state $i$, take the best predecessor $j$ and store it.
- Pick the best terminal state; backtrack to obtain the full path.
- Print zero-based indices.

**Answer to Exercise Question:**

**Q5.** If stored fully, $\delta$ has $N \times T$ values; the backpointer matrix has $N \times (T-1)$ entries (often implemented as $N \times T$ with the first column unused).

# 5 Exercise 4: HMM 3 – Estimate Model Parameters

## 5.1 Key Design Decisions

- **Scaled forward-backward.** We use per-time scaling $c_t$ in $\alpha/\beta$ to avoid underflow and compute $\log P(O \mid \lambda) = -\sum_t \log c_t$ as the convergence signal. Baum–Welch (EM) ensures non-decreasing likelihood and convergence to a stationary point.
- **Posterior normalization.** We form $\xi_t(i,j) \propto \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)$, normalize $\sum_{i,j} \xi_t(i,j) = 1$, then set $\gamma_t(i) = \sum_j \xi_t(i,j)$. Re-estimation uses the standard expected-count ratios for $A, B$ and $\pi$.
- **Convergence & safeguards.** Stop when $\Delta \log L < \varepsilon$ or a max-iter cap is reached; if a row denominator is zero during updates, fall back to a uniform row to keep rows stochastic. As EM is local-optimum seeking, multiple restarts are advisable in general.

## 5.2 Code Explanation

### 5.2.1 Main Components

- **E-step:** `alpha_pass()` (scaled), `beta_pass()` (scaled with the same $c_t$), and `compute_gammas()` to obtain $\gamma, \xi$.
- **M-step:** `reestimate()` applies $\pi_i = \gamma_0(i)$, $a_{ij} = \frac{\sum_{t<T} \xi_t(i,j)}{\sum_{t<T} \gamma_t(i)}$, $b_{ik} = \frac{\sum_t \mathbf{1}[o_t=k]\gamma_t(i)}{\sum_t \gamma_t(i)}$.
- `log_likelihood(c)` computes $\log P(O \mid \lambda)$ from the scales.

### 5.2.2 Logic Flow

- Initialize with given $A, B, \pi$.
- Iterate: forward $\to$ backward $\to \gamma, \xi \to$ re-estimate $A, B, \pi \to$ check $\log L$.
- Stop on small $\Delta \log L$ or max iterations and output learned $A, B$ (per Kattis specification).

**Answer to Exercise Question:**

**Q6.** Dividing $\xi_t(i,j)$ by $\sum_k \alpha_T(k) = P(O_{1:T} \mid \lambda)$ converts the joint $P(X_t = i, X_{t+1} = j, O)$ into the posterior $P(X_t = i, X_{t+1} = j \mid O)$, ensuring $\sum_{i,j} \xi_t(i,j) = 1$ so EM updates use valid expected counts.

# 6 Exercise 5: Fishing Derby – Player Agent

## 6.1 Key Design Decisions

- **One HMM per species (2 hidden states):** We maintain a discrete HMM for each species $\lambda^{(s)} = \{A^{(s)}, B^{(s)}, \pi^{(s)}\}$ with two hidden states. Two states are sufficient to capture "slow/stable" vs. "active" motion regimes without using too many parameters on short sequences.
- **Scoring vs. training:** For classification we use the *unscaled* forward sum $\sum_i \alpha_T(i)$ (as in HMM1) to compare models. For training, we use the *scaled* forward/backward with per-time factors $c_t$ and monitor convergence via $\log P(O \mid \lambda) = -\sum_t \log c_t$ (as in HMM3).
- **Lightweight EM:** After a wrong reveal, we retrain only the *true* species model for a small number of EM iterations (about 10). This focuses computation where it matters and avoids timeouts.
- **Delayed guessing:** We delay guessing until there are exactly enough steps left to make one guess per remaining fish.

## 6.2 Code Explanation

### 6.2.1 Main Components

- `hmm.py`: `alphaPass` (scaled forward), `betaPass` (scaled backward), `computeGamma`/`reestimateModel` (Baum–Welch/EM), and `sequnceProbabilities` (unscaled forward sum used for scoring).
- `player.py`: per-fish observation buffers, one HMM per species, a scoring function to choose the (fish, species) pair, local EM update of the true model after a wrong reveal.

### 6.2.2 Logic Flow

- **Data accumulation.** At each step we append the observed emission to each fish's sequence. While step $\leq N_{\text{steps}} - N_{\text{fish}}$ we collect data only.
- **Guess selection.** For every unguessed fish $f$ and species model $\lambda^{(s)}$, compute

$$\text{score}(f, s) \;=\; P\Big(O^{(f)}_{1:T} \mid \lambda^{(s)}\Big) \;\approx\; \sum_{i=1}^{N} \alpha^{(s)}_T(i),$$

  using the unscaled forward (HMM1 view). The guess is the fish–species pair with the highest score.
- **Update on reveal.** After feedback, if the guess is wrong we add the fish's sequence to the dataset of the true species and run a few Baum–Welch iterations:

$$\pi_i \leftarrow \gamma_0(i), \quad a_{ij} \leftarrow \frac{\sum_{t<T} \xi_t(i,j)}{\sum_{t<T} \gamma_t(i)}, \quad b_i(k) \leftarrow \frac{\sum_{t:O_t=k} \gamma_t(i)}{\sum_t \gamma_t(i)}.$$

  Scaling factors $c_t$ ensure numerical stability and allow tracking the log-likelihood increase.

### 6.3 Results & Discussion

With two states per species, unscaled forward scoring and targeted online EM, the controller achieves stable performance and a sufficient Kattis score. Delaying guesses to the final stretch increases sequence length at decision time, improving the prediction. Limiting EM to a few iterations keeps runtime low and prevents timeouts while still allowing models to specialize progressively.

## 7 Personal Contribution

### 7.1 Francesco Baldassarre

I was mostly responsible for the final part of the assignment: implementing the HMM-based agent for the Fishing Derby problem. Building on the earlier components, I designed the player to maintain one discrete HMM per species and to classify each fish by comparing forward scores across species. I integrated a lightweight, targeted Baum-Welch update after wrong reveals, and adopted the delayed guess strategy to wait until there were enough steps left to decide on all remaining fish.

This was an instructive bridge from theory to practice: I saw how HMMs map a concrete sequential decision problem with partial information and time limits. I learned how to implement forward/backward and Baum–Welch. The experience left me curious about applying HMMs beyond this project, for example to simple gesture/activity recognition or anomaly detection.

### 7.2 Ahmet Enis Isik

My main personal contribution was building the end-to-end HMM pipeline across Exercises 1–4. For **Exercise 1 (HMM0)** I implemented the Kattis-style parser and minimal row-vector $\times$ matrix routines to compute $\pi A$ and $(\pi A)B$ without NumPy, together with a compact formatter that preserves precision. For **Exercise 2 (HMM1)** I coded the forward ($\alpha$-pass) exactly as specified, kept only the current $\alpha$ row to stay memory-lean, and hardened input handling so integer/float observation tokens are both accepted. For **Exercise 3 (HMM2)** I wrote Viterbi with explicit backpointers: a tight max-over-predecessors loop for $\delta_t(i)$, an argmax grid $\psi_t(i)$, and a clean traceback that returns zero-based state indices. For **Exercise 4 (HMM3)** I implemented Baum–Welch with per-time scaling $c_t$ to prevent underflow, computed the log-likelihood $-\sum_t \log c_t$ for convergence checks, and added safe row renormalization so $A$ and $B$ remain stochastic; the E-step $(\alpha, \beta, \gamma, \xi)$ and M-step follow the expected-count formulas. Finally, I tied everything together in practice: a lightweight test harness to run `pypy3 MyHMM.py < file.in` against the provided `.ans`, sanity checks that each row of $A$, $B$, and $\pi$ sums to one (within a small $\varepsilon$), and formatting guards so outputs match Kattis requirements. This gave hands-on intuition for how prediction, evaluation, and decoding compose with EM, and how small engineering choices (scaling, normalization, and strict I/O) make a robust, test-passing solution.

## 8 Conclusion

This assignment guided us from the fundamental HMM to a full application in the Fishing Derby. In the first four exercises we implemented the canonical elements of a discrete HMM: next-step emission prediction via row-vector $\times$ matrix products, sequence evaluation with the forward algorithm, decoding with Viterbi and explicit backpointers, and parameter learning through Baum-Welch with per-time scaling factors. Along the way, we emphasized numerically stable computation (scaled $\alpha/\beta$ and log-likelihood as $-\sum_t \log c_t$), while having a stochastic row normalization.

In the final exercise, we translated these components into a functioning agent for the Fishing Derby. We modeled each species with a compact two-state HMM sharing the same alphabet, scored fish sequences by comparing unscaled forward sums across species, and used targeted online EM to refine the correct model after a wrong reveal. This approach achieved a sufficient score on Kattis, meeting the assignment's objective.

Overall, the project made the connection between HMM theory and practice concrete. Implementing the algorithms clarified how modeling choices determine performance.