ML HACKATHON

Hackman Project Analysis Report

Nithya Sri P. B.	PES2UG23CS399
Pooja Reddy V	PES2UG23CS413
Preksha S Hiremath	PES2UG23CS444
Raghhav Dharmenddra Malani	PES2UG23CS460

1. Key Observations and Challenges

The most challenging part of the project was implementing HMM and Reinforcement Learning using DQN into a single pipeline.Careful tuning was essential for designing meaningful state representations and enabling the RL agent to learn stable policies, preventing it from collapsing into repetitive guessing behaviors.

Another key challenge involved balancing accuracy with computational efficiency. The DQN, while effective, demanded extensive training episodes to achieve stability. Conversely, the HMM offered more linguistically coherent predictions but incurred significant computational costs due to its probability calculations.

2. Strategies: HMM and RL Design

Design Strategy for HMM:
Our method made use of a Hidden Markov Model (HMM) that was organized as a bigram model. Because each hidden state in this design is associated with a letter, the transitions can capture the likelihood that one letter will come right after another.

Essential design requirements:

Transition Matrix: The transition matrix was calculated using the training data's letter-pair counts. For unseen bigrams, we used Laplace smoothing to avoid zero probabilities.

Emission Model: The letters in the current Hangman pattern were dynamically reflected by the model. Then, using these known letters, posteriors were computed, hiding any possibilities that were already guessed or otherwise unfeasible.

Normalization: To maintain valid conditional probabilities, each row in the transition matrix was normalized to ensure its values summed to 1.

RL State and Reward Design
To provide comprehensive context for the agent, the state representation combined multiple data sources:

- The current word pattern (e.g., `_ a _ _ _`).
- A binary mask indicating which letters had already been guessed.
- The number of remaining lives (allowed wrong guesses).
- The posterior probabilities for each letter, as calculated by the HMM.
-

This design gave the agent probabilistic guidance (the probability of each letter that was not guessed) as well as structural information (the pattern of the word).

The reward structure was carefully shaped to encourage efficient, strategic guessing over short-term gains:

- Correct Letter: `+1`
- Incorrect Letter: `-1`
- Game Win (Word Completed): `+10`
- Game Loss (Lives/Steps Exhausted): `-5`

3. Exploration vs. Exploitation

We used the standard epsilon -Greedy Strategy to manage the trade-off.

- Initialization: Epsilon started at 1.0 (pure exploration) to quickly gather diverse data in the replay buffer.
- Decay: epsilon decayed gradually (e.g., 0.9995 per episode) down to a minimum value of 0.01 . This ensures that as the agent learns better Q-values, it transitions smoothly from random trial-and-error to selecting the highest-valued action (exploitation).
- Action Masking: Crucially, even during epsilon -greedy exploration, the agent was only allowed to select from the pool of un-guessed letters. This prevents the agent from wasting time on repeated guesses during training and focuses exploration on valid choices.

4. Future Improvements

If we had more time to refine the agent, the following two improvements would be critical:

1. Length-Specific HMMs: Currently, one HMM is trained over the entire corpus. A significant improvement would be to train separate HMMs for different word lengths (e.g., HMM-5, HMM-8, HMM-12). This would allow the HMM oracle to give far more accurate probabilities that respect the length constraint, potentially leading to a higher win rate and lower wrong guesses.
2. Advanced Candidate Filtering Integration: The current state vector includes the HMM output, but doesn't explicitly tell the RL agent how many possible words are left in the candidate pool. A strong feature would be including log(Candidates) in the state vector, which is a powerful information measure. The agent could then learn that when “Candidates” is small, it should trust its decision more, and when it is large, it should prioritize letters that maximize the informational gain (i.e., letters that split the remaining candidates most effectively).