Department of Computer Science Engineering

UE23CS352A: Machine Learning Hackathon

Hackman

Name	SRN	Section
Praagnya Parimi	PES2UG23CS417	G
Preksha	PES2UG23CS443	G
Priyanka Satish	PES2UG23CS450	G
Priyanka Shankar Naik	PES2UG23CS451	G

Key Observations

- **HMM Training Stability**: Ensuring the HMM converges for words of varying lengths, especially with limited data for longer words, was tricky. Initial parameter choices and sequence grouping impacted EM convergence and result quality.

- **State/Reward Design for RL**: Balancing informative states with compactness for Q-learning was a challenge. Representing Hangman game states in a way that allowed learning and generalization (e.g., accounting for guessed letters, remaining lives, word progress, top-K HMM predictions) was non-trivial.

- **Exploration-Exploitation Trade-off**: Tuning epsilon decay and integrating HMM-based priors with RL actions was complex. Too much exploitation led to stagnant policies and missed rewards, while excessive exploration slowed convergence.

- **Performance Tuning**: Achieving stable reward signals and robust evaluation across a large word corpus required systematic reward shaping and evaluation function design.

- **Integration of HMM and RL:** Combining probabilistic inference (HMM) with reinforcement learning was conceptually challenging. The HMM produces a probability distribution over letters, while the RL agent selects discrete actions based on Q-values. Aligning these two outputs—so that the RL policy could meaningfully interpret and act on HMM predictions—required careful normalization and feature scaling. Any mismatch caused unstable training or biased action preferences.

Insights Gained

- Compact state summaries (mask, lives, progress, HMM top-K guesses) worked well for RL, enabling generalization across diverse words and game progressions.

- Hybrid agents leveraging statistical (HMM) probabilities with Q-learning achieved competitive performance but required careful balancing of model weights and Q-table updates.

- Pure HMM agents performed surprisingly well in greedy guessing scenarios, often outperforming hybrid RL in overall score and error minimization for Hangman.

Strategies

- **HMM Design Choices:**
    - Used discrete HMMs with multinomial emissions over word alphabets; trained separate HMMs for each word length with Baum-Welch EM and Viterbi inference.

- For stability, the number of hidden states was set via a heuristic: $\min(\max(2, \text{length}/2), 10)$, which balanced expressive capacity vs. overfitting.

- String masks were handled with per-position predictions, allowing targeted letter probability inference for each blank.

- **RL State and Reward Design:**
    - RL states encoded the masked word, guessed letter set, remaining lives, and HMM top-K probabilistic hints.

    - Rewards favored correct guesses (+5), winning (+50), and penalized incorrect guesses (-5), losses (-50), and repeats (-2). More extreme penalties for repeated mistakes improved stability and discouraged uninformative guessing.

    - The Q-table was indexed by compact state keys, incorporating both game progress and HMM summary signals.

Exploration vs Exploitation

- Approach:
    - Used an epsilon-greedy strategy with decay (init 0.20, min 0.01, decay 0.995); balancing was critical: initial exploration helped discover winning strategies, while late-stage exploitation capitalized on learned Q-values and high-probability HMM guesses.

    - HybridAgent weighted HMM probabilities heavily (weight 8), but allowed Q-table values (weight 2) to gradually influence action selection as training progressed.

- In cases with uncertain or ambiguous HMM feedback, fallback unigram letter frequencies provided a safety net for exploration.

Future Improvements

- **Ensemble Models:** Try combining multiple HMMs or sequence models for more robust letter prediction especially for long/rare words.

- **Deep RL:** Experiment with neural policy/value networks or recurrent RL architectures to better capture sequential dependencies and partially observable states.

- **Reward Engineering:** Refine reward functions for more granular feedback—e.g., reward streaks, partial word reveals, or penalize suboptimal but not strictly incorrect guesses.

- **Richer State Features:** Encode guess history, position uncertainty, or semantic word clustering to aid RL generalization and reduce blind spots.

- **Transfer Learning:** Pre-train on similar word games or use curriculum learning to bootstrap agent performance for difficult words.

- **Interactive Visualization:** Add more explainable AI outputs—visualize guessing trajectories, show confidence scores, highlight Q-value updates per step.