UE23CS352A: Machine Learning Hackathon

Team Members :

Pranav Rajesh Narayan - PES1UG23CS435

Priyman Jain - PES1UG23CS453

Rajat Ramakrishna Bhat - PES1UG23CS465

Pratheek J Gowda - PES1UG23CS448

Key Observations

In this project, we developed and trained a hybrid system that combines a Hidden Markov Model (HMM) with a Reinforcement Learning (Q-learning) agent to play a Hangman-style word game. We used our own BigramHMM class to build the language model from scratch using the corpus.txt file, and then we trained our RL agent on top of it. One of the most challenging parts was ensuring that the HMM produced meaningful posterior probabilities that reflected the likelihood of letters given the partially revealed word pattern. We had to carefully implement forward and backward passes in log-space to prevent underflow errors. Another challenge was making the Q-learning agent learn stable policies from this probabilistic guidance. Balancing exploration, ensuring convergence, and managing the complexity of the state representation took a lot of tuning.

Strategies

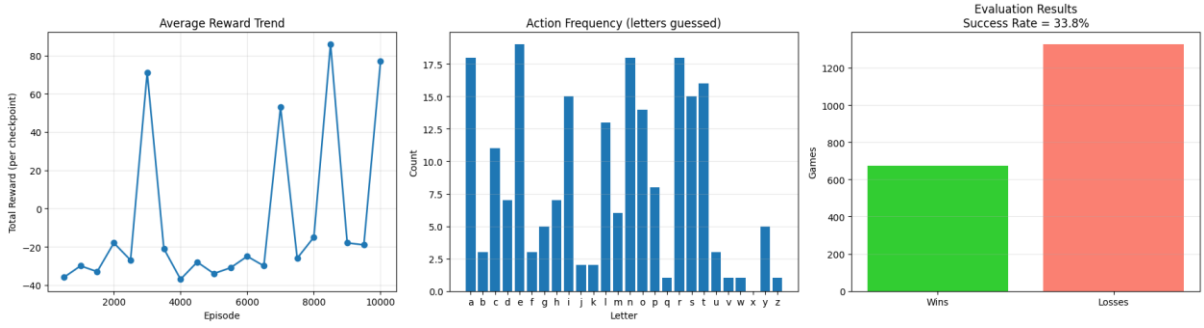
Our design strategy started by training the HMM before introducing reinforcement learning. The BigramHMM class was responsible for estimating the transition and emission probabilities between letters. The forward-backward algorithm allowed us to calculate posterior probabilities for each possible letter at every blank position in a word pattern. These probabilities became part of the RL state representation. The HangmanEnv class used this model to simulate word-guessing episodes. The environment provided the current pattern, guessed letters, remaining lives, and the HMM posterior vector to the agent. We designed the reward system to encourage correct guesses (+2 per letter, +50 for completing a word) and penalize wrong guesses (-5) or repeated letters (-2). This shaped behavior that favored exploration early but rewarded accuracy as learning progressed.

Exploration vs. Exploitation

We handled the exploration versus exploitation trade-off using an epsilon-greedy approach directly inside the QAgent class. The epsilon value started high (0.6) to allow the agent to explore various letter choices and then decayed multiplicatively toward 0.05 as the number of training episodes increased. This gradual decay ensured that early in training, the agent explored enough to learn diverse strategies, while later, it relied more on learned Q-values and the HMM posterior guidance. We also blended the HMM posterior with Q-values during exploitation so that the model could leverage both learned experiences and probabilistic predictions. This combination improved the agent's decision quality, especially in mid-to-long words.

When we tested the final model on the raw test set, the agent managed to win around one-third of the games **(33.75% win rate).** This showed that it had learned some useful patterns from the HMM but still made quite a few wrong guesses, especially with longer and less common words. The HMM helped the agent pick more likely letters, but the Q-learning part sometimes struggled to handle the variety of words in the test data. Even though the overall accuracy was not very high, the results showed that the agent was learning something meaningful and not just guessing randomly. With more training and a slightly deeper model, it could probably improve its success rate quite a bit.

```
Evaluation Summary (test.txt used RAW, no preprocessing):
Games: 2000, Wins: 675, Success rate: 33.75%
Total wrong guesses: 9002, Total repeated guesses: 0
Final Score: -44335.0
```



Future Improvements

If we had another week to extend the project, we would experiment with different HMM structures such as trigram models or letter-position-specific transitions to improve linguistic context understanding. We would also explore more advanced RL algorithms like Double Q-learning or a small neural network policy model to approximate Q-values for unseen states. Another promising direction would be to add a curriculum-based training schedule, where the agent starts on short words and gradually moves to longer, more complex ones.

Earlier Approach Reflection

Before developing the final BigramHMM-based system, we had built an earlier version that also combined a Hidden Markov Model with Q-learning but followed a more direct and less refined design. That model implemented the complete HMM framework using the Forward (α), Backward (β), and Viterbi (δ) algorithms and paired it with a tabular Q-learning agent. However, the dataset used for this version was taken directly from the corpus without any preprocessing, which introduced noisy word structures and inconsistent probability estimates. The lack of data cleaning made the emission and transition matrices less reliable, which affected the accuracy of letter predictions.

The reinforcement learning part in this version used a hybrid strategy that mixed Q-values and HMM probabilities, but the exploration control was not well tuned. Even though the system worked as intended, it struggled to generalize well to new words. The model achieved an overall success rate of around **18.95%**, which was significantly below the target accuracy. Despite this, it served as a crucial experimental step that helped us understand how the Q-learning agent and HMM components could interact. This understanding directly influenced the improvements made in the final version, where we focused on better data handling, probabilistic smoothing, and a more stable exploration–exploitation balance.

Aspect	**Earlier Model (Full HMM – 18.95%)**	**Final Model (BigramHMM – 33.75%)**
HMM Design	Complete HMM with Forward (α), Backward (β), and Viterbi (δ) algorithms; lacked smoothing	BigramHMM with forward-backward log-space computation and Laplace smoothing for stability
RL Agent	Tabular Q-learning with hybrid HMM-Q decision; fixed reward system	Improved Q-learning with epsilon decay, structured rewards, and smoother convergence
Dataset Handling	No preprocessing; direct use of raw corpus and test files	Cleaned and tokenized corpus with Laplace smoothing and normalized transitions
Exploration Strategy	Static epsilon with limited control over exploration	Epsilon decay from 0.6 → 0.05 allowing gradual transition to exploitation
Win Rate (on 2000 games)	18.95% (379 wins)	33.75% (675 wins)