Machine Learning Hackathon
Hackman - Team 10
Analysis Report

Section: AIML - B
Team Members:

PES2UG23AM072	PIDAPA INDIRA
PES2UG23AM088	SAI HEMANTH MEDICHERLA
PES2UG23AM089	SAI JASWANTH AKULA
PES2UG23AM111	UDAYA PRAGNA GANGAVARAM

Key Observations

The Hangman problem involved combining **probabilistic modeling (HMM)** with **decision-based learning (RL)**.
The main challenges faced during development were:

1) Managing **variable-length words** in the corpus during HMM training.

2) Designing a **stable DQN** that could learn from sparse rewards and limited context.

3) Handling the exploration vs. exploitation trade-off efficiently.

4) Maintaining consistent convergence and preventing repetitive guessing in the RL stage.

Key insight: combining **language-based priors (HMM)** with **adaptive learning (RL)** significantly improved the model's understanding of word patterns, increasing the success rate compared to simple frequency-based guessing.

Strategies

HMM Design

Implemented a Per-Length HMM structure, where a separate HMM was trained for each word length to better capture position-specific letter dependencies.

For instance, shorter words have distinct letter transition patterns compared to longer words. Each of these models learns its own transition and emission probabilities, allowing the system to adapt to variable-length linguistic contexts.

Used MultinomialHMM from the hmmlearn library, as each state emits discrete letter observations from the alphabet.

The design encoded each word as a sequence of one-hot vectors, so the HMM learned direct letter-to-letter dependencies.

During training, forward–backward algorithms were used internally by hmmlearn to optimize transition and emission probabilities through Expectation-Maximization as part of using the Baum–Welch algorithm, get executed internally by the hmmlearn library when you call .fit().

Each trained HMM was serialized and saved via pickle, allowing fast reloading and evaluation during RL integration.

The transition matrix (transmat_) represents how hidden states (corresponding to abstract letter positions) evolve across the word, while the emission matrix (emissionprob_) models how likely each observable letter is to be generated from each hidden state.
Both matrices are initialized uniformly and refined automatically during training to maximize the overall likelihood of the observed corpus.
This enables the HMM to effectively capture common letter sequences and structural regularities of the English language.

This multi-model HMM training approach allowed the system to generalize well — instead of one monolithic model trying to handle all word lengths, it had specialized submodels fine-tuned to their length category.

Impact: This improved both accuracy and efficiency of the probability estimation, especially for words whose letter patterns differ by length.

Reinforcement Learning (DQN) Design

After successfully training the **Hidden Markov Models (HMMs)**, the project transitions into the **Reinforcement Learning (RL)** stage — where a **Deep Q-Network (DQN)** is trained to intelligently guess letters in the Hangman game.

1) Environment Setup (HangmanEnv)

The HangmanEnv class defines the game dynamics.
It interacts with the DQN agent by providing the **current state**, **available actions**, and **reward feedback** after each guess.
Sets up state representations, actions and rewards.

2) Integration with HMM

The function get_hmm_letter_agg() combines the probabilities predicted by the HMM (predict_letter_distribution) into a 26-element vector.
This acts as a prior probability distribution over letters, helping the DQN agent make informed decisions rather than random guesses.
This hybrid setup — combining a probabilistic model (HMM) and a learning-based model (DQN) — allows the agent to leverage language structure *and* experience simultaneously.

3) Deep Q-Network (DQN) Architecture

The DQN uses a three-layer fully connected neural network implemented in PyTorch:
This network learns a mapping from the current game state to Q-values, where each Q-value represents the expected cumulative reward of guessing a particular letter.
The RL component was built using PyTorch, implementing a Deep Q-Network (DQN) that learns to predict the most promising next letter at each guessing step.

4) Replay Buffer and Target Network

Experience Replay

The ReplayBuffer stores experiences in the form of:

Sampling random mini-batches from this buffer during training breaks temporal correlations between consecutive guesses.
It stabilizes learning and improves sample efficiency.
Target Network
Two networks are maintained:
Policy network → the model being trained.
Target network → a slowly updated copy of the policy network.
Every 600 steps, the target network is synchronized with:
This prevents rapid Q-value oscillations and helps stabilize training.

5) Training Loop
The main training loop runs for 10,000 episodes:
Exploration–Exploitation:
Controlled using ε-greedy strategy:
Starts with ε = 1.0 (fully random exploration).
Gradually decays to ε = 0.05 using EPS_DECAY = 0.9993.
Ensures balanced exploration early on and exploitation later.
Optimization:
Mini-batches of size 128 sampled from replay buffer.
Target Q-values computed using the Bellman equation:
Loss computed using Mean Squared Error (MSE) between predicted and target Q-values.
Optimized using Adam optimizer with LR = 1e-4.
Target Updates:
After every 600 training steps, the target network parameters are updated.
Model Checkpoint:
After training, the policy is saved as

6) Policy Effect
The final learned policy (policy) combines:
Statistical priors from the HMM (language-driven guess likelihoods).
Dynamic value-based optimization from the DQN (reward-driven guessing behavior).
This makes the agent:
Prefer linguistically plausible guesses early on.
Adapt guessing strategies dynamically as more letters are revealed.

Impact: The resulting RL policy evolved beyond static letter frequency models — it made intelligent, context-aware guesses that improved the overall success rate and reduced the average number of incorrect attempts.

Exploration vs. Exploitation

The ε-greedy policy enabled controlled exploration.

Early episodes: high ε → more random exploration

Later episodes: low ε → exploit learned Q-values

EPS_DECAY = 0.9993 ensured gradual convergence.

Experience replay stabilized learning and reduced correlation between consecutive samples.

Target network synchronization (TARGET_UPDATE_FREQ = 600) prevented Q-value oscillation.

This careful tuning allowed the agent to improve letter prediction efficiency and reduce redundant exploration.

Future Improvements

If given additional time, several enhancements could further improve both the HMM and RL components:

Enhancing the HMM

Contextual HMM Smoothing:

Incorporate smoothed transition probabilities using a weighted combination of forward and backward likelihoods to handle unseen letter sequences and rare words better.

Improving the RL Policy

Policy Optimization:

Upgrade the DQN into a Double DQN or Dueling DQN to reduce overestimation bias and stabilize Q-value learning.

Alternatively, explore Actor–Critic methods (e.g., A2C) for continuous improvement of policy updates.

Curriculum Learning:

Start training on shorter, simpler words before progressing to longer, more complex words.

This structured learning path helps the model gradually build robust representations.

Adaptive Reward Shaping:

Introduce intermediate rewards for partial progress — such as revealing vowels, reducing the number of blanks, or guessing consecutive correct letters.

Hybrid Integration:

Feed HMM forward-backward probabilities directly as additional features into the RL state vector, allowing the agent to reason with probabilistic letter likelihoods.

Visualization & Monitoring Tools:

Build a small dashboard to visualize the HMM transition matrices, forward–backward probabilities, and real-time evolution of the DQN's Q-values during training.

Results:

```
🔍 Evaluating on 2000 words...
Showing first 15 words:

--------------------------------------------------
🔤 Target Word: marmar
Start: ______
Guess: e | Masked: ______ | Wrong: 1
Guess: s | Masked: ______ | Wrong: 2
Guess: a | Masked: _a__a_ | Wrong: 2
Guess: r | Masked: _ar_ar | Wrong: 2
Guess: m | Masked: marmar | Wrong: 2
--------------------------------------------------
🔤 Target Word: janet
Start: _____
Guess: e | Masked: ___e_ | Wrong: 0
Guess: r | Masked: ___e_ | Wrong: 1
Guess: s | Masked: ___e_ | Wrong: 2
Guess: a | Masked: _a_e_ | Wrong: 2
Guess: t | Masked: _a_et | Wrong: 2
Guess: c | Masked: _a_et | Wrong: 3
Guess: w | Masked: _a_et | Wrong: 4
Guess: n | Masked: _anet | Wrong: 4
Guess: k | Masked: _anet | Wrong: 5
Guess: z | Masked: _anet | Wrong: 6
...
Win Rate: 71.05%
Avg Wrong Guesses: 3.89
Final Score (sum of rewards): 27955.00
==============================
```