UE23CS352A: Machine Learning Hackathon
Hackman

Analysis Report

Team member 1 :
Name : Akshaya R
SRN: PES2UG23CS047

Team member 2 :
Name : Anagha Aravind Jois
SRN: PES2UG23CS060

Team member 3 :
Name : Ananya Prabhu
SRN: PES2UG23CS063

Team member 4 :
Name : Ananya Reddy G
SRN: PES2UG23CS064

Key observations:

Challenges Faced:

- **Sparse and Delayed Rewards:**
  In Hangman, meaningful rewards (like completing a word) occur rarely. This made it difficult for the agent to correlate actions (letter guesses) with eventual success.
- **Large and Complex State Space:**
  Each word length introduces unique letter positions and dependencies. Encoding the pattern, guessed letters, and remaining lives in a compact way that preserves semantic meaning was challenging.
- **Language Dependency:**
  Hangman inherently depends on linguistic probability (e.g., “q” usually followed by “u”). The DQN alone could not infer such rules efficiently without help from a language model.
- **Fusion Tuning (DQN + HMM):**
  Finding the right fusion factor α between the DQN’s Q-values (strategic reasoning) and the HMM’s probabilities (language structure) required multiple experiments.
  We observed that α = 0.7 gave the most stable results — allowing the agent to follow linguistic logic while still prioritizing its learned policy.

Insights Gained

- The combination of reinforcement learning and probabilistic modeling works far better than either approach alone — DQN handles decision patterns while HMM captures language flow.
- Even with modest success (~14% win rate), the letter accuracy (~50%) showed that the agent had learned structured guessing instead of random selection.
- The reward curve revealed high variability depending on word difficulty — shorter or common words yielded high rewards, while rare or long words caused performance dips.
- Designing the right reward function and state encoding had a bigger impact on stability than network size or training duration.

Strategies:

HMM Design Choices:

- We trained five separate HMMs, each specializing in a word-length range:
    - HMM_1 (3–5 letters)
    - HMM_2 (6–7 letters)
    - HMM_3 (8–9 letters)
    - HMM_4 (10–12 letters)
    - HMM_5 (13+ letters)
- Each HMM captures the statistical transitions between letters (e.g., `t → h → e`) in words of that range.
- The best number of hidden states for each model was found using the Bayesian Information Criterion (BIC) to prevent overfitting.
- During evaluation, the HMM corresponding to the word's length predicts letter probability distributions, providing language-based priors to the DQN.

Reinforcement Learning (DQN) Design

- **State Representation (41 features):**
    1. 26-length binary pattern vector (revealed letters).
    2. 26-length guessed-letter vector (history of guesses).

- **Network Architecture:**
  The DQN had **41 → 256 → 128 → 26** neurons, using ReLU activations.
  Output dimension (26) corresponds to all possible letter actions.

- **Reward Function Design:**

Action	Reward
Correct letter	+1
Complete word	+10
Wrong letter	−1
Repeated guess	−0.5

- These values balance short-term accuracy and long-term success, preventing random exploration while encouraging efficient word completion.

Exploration vs. Exploitation

Training Phase

- Used an ε-greedy policy to manage the exploration–exploitation trade-off:
    - Initially, ε = 1.0 i.e pure exploration (random guesses).
    - Gradually decayed to ε = 0.1 , so more exploitation as the model learned.
- This approach ensured:
    - The agent explored enough letter combinations early on.
    - Later, it exploited high-value Q-actions to stabilize performance.

Evaluation Phase

- Set ε = 0 (pure exploitation).
  The agent only used the trained Q-values and fused HMM probabilities for

deterministic, consistent gameplay.

Outcome

- The ε-decay strategy reduced random noise and improved convergence.
- Combined with α-fusion, it maintained a balance between learning from environment (RL) and leveraging linguistic priors (HMM).

Future Improvements:

(a) Improved State Encoding

Replace one-hot letter encoding with **learned embeddings** or **positional encodings** (similar to Transformers) for more expressive state representations.

(b) Curriculum-Based Training

Start with short, simple words and gradually progress to longer, complex words to improve training stability and prevent early overfitting.

(c) Smarter Exploration

Use softmax exploration or entropy regularization instead of ε-greedy to ensure more consistent exploration of near-optimal actions.

(d) Transformer-Based Language Priors

Replace HMMs with a character-level Transformer model to better capture long-term letter dependencies and context.

(e) Reward Shaping

Introduce intermediate rewards for uncovering new letters or partial segments to provide denser learning signals and faster convergence.