ML HACKATHON REPORT

Nilay Srivastava -PES2UG23CS390

Nisha Patil-PES2UG23CS393

Nisschay Khandelwal- PES2UG23CS394

Nitali Rajesh-PES2UG23CS395

Key Observations: What were the most challenging parts? What insights

did you gain?

1. **Balancing HMM and RL**: Finding the optimal blend weight (ended at 98% HMM trust) was critical. Too much RL led to erratic behavior; too little made training pointless.
2. **Sparse State Space**: With a compact state representation,the Q-table struggled to generalize across different word patterns. This is why HMM guidance became so crucial.
3. **Exploration vs. Exploitation Trade-off**: Starting with high exploration hurt performance because random guesses waste lives. The solution was ultra-low initial epsilon (0.15) combined with frequency-based exploration.
4. **Early Game Strategy**: The first few guesses are critical. Pure HMM probabilities weren't enough - needed to boost common English letters (e, t, a, o, i, n, s, h, r, d) by 50% in the first 6 guesses.

Key Insights:

- **HMM is the workhorse**: Context-based letter probabilities from bigram transitions (order=2) provided the foundation. RL acted more as a fine-tuning mechanism.
- **Frequency matters more early**: Common letters should be prioritized before context clues emerge.
- **Simplicity wins**: Complex state representations led to overfitting. The compact 4-tuple state was enough.

Strategies: Discuss your HMM design choices. Detail your RL state and

reward design and why you chose them.

HMM Design Choices:

1. **Order-2 (Bigram) Model**:
    - Captures immediate context (e.g., "qu" → 'e' is common)
    - More than order-2 led to data sparsity and overfitting
    - Less than order-2 missed critical patterns
2. **Triple Probability Combination**:

P(char | context, pos) = 0.70 × P_transition + 0.20 × P_positional + 0.10 × P_frequency

    - **70% transition**: Primary signal from bigram context
    - **20% positional**: Character preferences by position (e.g., vowels in middle)
    - **10% frequency**: Baseline letter distribution
3. **Laplace Smoothing (α=0.01)**: Prevents zero probabilities for unseen bigrams while not diluting strong signals
4. **Context Padding**: Uses <S> tokens to handle word beginnings uniformly

RL State Design:

State Tuple: (word_length, num_blanks, lives_left, last_char)

Rationale:

- **word_length**: Differentiates short vs. long words (strategy differs)
- **num_blanks**: Progress indicator (near completion needs different tactics)
- **lives_left**: Risk management (few lives = conservative choices)
- **last_char**: Recent context hint (helps with patterns like double letters)

This compact representation keeps the Q-table manageable while capturing essential game dynamics.

Reward Design:

Correct guess: +2 × count (count = occurrences of letter in word)

Wrong guess: -1

Repeated guess: -3

Win game: +10

Lose game: -5

Rationale:

- **+2 per letter**: Rewards high-frequency letters (vowels often appear multiple times)

- **-1 for wrong**: Penalty proportional to wasted lives
- **-3 for repeats**: Strong discouragement (wastes a turn completely)
- **+10 win bonus**: Incentivizes completion over conservative play
- **-5 loss penalty**: Ensures agent learns to avoid risky strategies near game-end

---

Exploration: Managing Exploration vs. Exploitation Trade-off

Strategy:

1. **Ultra-Low Initial Epsilon (0.15)**:
   - Traditional RL starts at ε=1.0, but Hangman punishes random exploration
   - Starting at 0.15 means 85% exploitation from the start
   - RL learns from HMM's guidance rather than blind exploration
2. **HMM-Guided Exploration**:
   When exploring (15% of the time), actions are sampled from HMM probabilities rather than uniformly:

```
if random() < epsilon:
    # Sample from HMM distribution (not uniform!)
    action = np.random.choice(n_actions, p=hmm_probs)
```

This ensures even "exploration" is informed.

3. **Aggressive Epsilon Decay (0.998)**:
   - Decays to minimum (0.01) over ~70,000 episodes
   - Final policy is 99% exploitation with only 1% HMM-guided exploration
   - Justification: Hangman has deterministic dynamics; once patterns are learned, exploration hurts
4. **Frequency Boost During Exploitation**:
   - Even when exploiting, common letters get a 50% probability boost in the first 6 guesses
   - This acts as a form of "smart exploitation" - following linguistic priors
5. **Valid Action Masking**:
   - Already-guessed letters are masked out (probability set to 0)

  - Prevents wasting turns on invalid actions
  - Makes exploration space smaller and more efficient

Final Balance:

- **Training**: Started 15% explore → ended 1% explore (extremely greedy)
- **Evaluation**: 0% exploration (pure exploitation of learned Q-values + HMM)
- **Result**: ~99% success rate on test set, showing the policy learned to trust HMM guidance with minor RL adjustments

Future Improvements: If you had another week, what would you do to

improve your agent?

1. Trie-Based Word Filtering (Biggest Impact)

What it does: Instead of guessing blindly, filter the dictionary to only words matching the current pattern.

Example:

- Current mask: "_a_"
- Filter corpus to only 3-letter words with 'a' in position 1: {cat, bat, hat, mat, ...}
- Calculate letter frequencies ONLY from these candidates
- Much smarter guesses!

Why it works:

- Dramatically reduces search space
- Focuses on actually possible words
- Especially powerful for rare/long words

Expected gain: +5-8% success rate

---

2. Information Gain Selection (Smarter Strategy)

What it does: Instead of picking the most probable letter, pick the letter that gives you the **most information**.

Example:

- Mask: "___e_"

- Letter 't' appears in 60% of candidates
- Letter 's' appears in 45% of candidates, BUT splits candidates more evenly (reveals more info)
- Choose 's' because it eliminates more uncertainty

Why it works:

- Maximizes what you learn per guess
- Especially valuable when lives are low
- Classic information theory approach

Expected gain: +3-5% success rate

Combined: These two changes could push your score from ~99% to **99.5%+** with minimal code complexity.