Hangman AI Agent: Technical Report

Hybrid HMM + Q-Learning Approach

Contributors:

Arjun Anand	Hruthvik Surya	Harshita Chhaparia	Archita Jain
PES1UG23AM065	PES1UG23AM119	PES1UG23AM115	PES1UG23AM063

Executive Summary

This report presents a hybrid artificial intelligence system for playing Hangman that combines Hidden Markov Models (HMMs) with Reinforcement Learning (Q-Learning). The agent achieves intelligent letter guessing by leveraging statistical language patterns and adaptive learning from gameplay experience.

Key Results:

- Successfully integrates probabilistic modeling with reinforcement learning
- Trains on custom word corpus to learn language-specific patterns
- Evaluates performance using a configurable scoring system
- Provides comprehensive training visualization and metrics

---

1. System Architecture

1.1 Overview

The system consists of three main components:

1. **HMM Oracle**: Provides letter probability predictions based on linguistic patterns
2. **Q-Learning Agent**: Learns optimal action selection through experience
3. **Hangman Environment**: Simulates the game with standard rules

1.2 Component Interaction

Corpus Text → HMM Training → Letter Probabilities

↓

Game State → Q-Learning Agent → Action Selection → Environment

↑ ↓

← Experience Replay ← Reward Signal ←

---

2. Hidden Markov Model (HMM) Implementation

2.1 Model Structure

The HMM component treats each word as a sequence of hidden states (letters) with observable emissions (revealed/hidden positions).

Key Parameters:

- **States**: 26 lowercase English letters
- **Observations**: Position-specific letter visibility
- **Start Probabilities**: Distribution of first letters
- **Transition Matrices**: Position-specific letter-to-letter transitions
- **Emission Probabilities**: Likelihood of observing revealed vs. hidden letters

2.2 Forward-Backward Algorithm

The implementation uses the classic forward-backward algorithm for inference:

Forward Pass (α):

- Computes probability of partial observations up to position t
- Implements scaling to prevent numerical underflow
- Handles both revealed and hidden positions dynamically

Backward Pass (β):

- Computes probability of future observations from position t
- Uses similar scaling techniques for stability

Posterior Computation (γ):

- Combines α and β to compute posterior probabilities

- Aggregates position-specific probabilities for unguessed letters
- Normalizes to produce valid probability distribution

2.3 Training Process

The HMM trains on a word corpus by:

1. Computing letter frequency for start probabilities
2. Building position-specific transition matrices from bigram statistics
3. Setting emission probabilities (95% for correct match, 5% for mismatch)
4. Applying smoothing (1e-6) to handle unseen transitions

2.4 Advantages

- Captures positional dependencies between letters
- Models word structure at different lengths
- Provides interpretable probability distributions
- Handles partial information naturally

---

3. Q-Learning Agent

3.1 State Representation

The agent uses a composite state consisting of:

- **Masked Word**: Current pattern of revealed/hidden letters
- **Guessed Letters**: Set of already attempted letters
- **Lives Remaining**: Number of incorrect guesses allowed

This state is converted to a hashable tuple for Q-table indexing.

3.2 Action Selection

The agent employs a **hybrid scoring mechanism**:

Score(letter) = (HMM_probability × 0.7) + (Q_value × 0.3)

This combination allows:

- HMM to provide strong linguistic priors
- Q-learning to adapt to specific game patterns
- Balanced exploration-exploitation through ε-greedy strategy

3.3 Learning Algorithm

Standard Q-Learning Update:

$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \cdot \max Q(s',a') - Q(s,a)]$

Parameters:

- Learning rate (α): 0.001
- Discount factor (γ): 0.95
- Initial epsilon (ε): 1.0
- Epsilon decay: 0.995
- Minimum epsilon: 0.1

3.4 Experience Replay

The agent maintains a replay buffer (capacity: 10,000) and performs batch updates:

- Stores transitions: (state, action, reward, next_state, done)
- Samples random batches of 32 experiences
- Breaks correlation between consecutive updates
- Improves sample efficiency

---

4. Reward Structure

The environment provides shaped rewards to guide learning:

Event	Reward	Purpose
Win game	+100	Strong positive reinforcement
Correct guess (per letter)	+10	Encourage revealing letters
Incorrect guess	-5	Mild penalty for mistakes
Repeated guess	-2	Discourage inefficiency
Loss game	-100	Strong negative reinforcement

---

5. Training Procedure

5.1 HMM Training

1. Load corpus from corpus.txt
2. Filter valid words (alphabetic, lowercase)
3. Group words by length
4. Train separate HMM for each word length
5. Compute transition and emission probabilities

5.2 Q-Learning Training

1. Initialize Q-table and replay buffer
2. Run 2,000 training episodes
3. For each episode:
    - Reset environment with random word
    - Select actions using ε-greedy policy
    - Store experiences in replay buffer
    - Perform batch Q-table updates
    - Decay exploration rate

5.3 Convergence Monitoring

- Track moving average reward (window: 100)
- Monitor success rate over episodes
- Print progress every 100 episodes

---

6. Evaluation Methodology

6.1 Test Protocol

- Runs 2,000 evaluation games
- Uses greedy policy (no exploration)
- Loads words from test.txt if available
- Falls back to random sampling from corpus

6.2 Metrics Collected

- **Success Rate**: Percentage of won games
- **Total Wrong Guesses**: Sum of incorrect attempts
- **Total Repeated Guesses**: Sum of duplicate attempts

6.3 Scoring Formula

The system uses a composite score designed to encourage positive values:

Win Score = Wins × 100

Wrong Penalty = Total Wrong Guesses × 2

Repeat Penalty = Total Repeated Guesses × 1

Final Score = max(0, Win Score - Wrong Penalty - Repeat Penalty)

This scoring system:

- Heavily rewards winning games
- Applies moderate penalties for inefficiency
- Ensures non-negative scores
- Balances success rate with guess quality

---

7. Visualization

The system generates two training plots:

7.1 Reward Progression

- Shows moving average of episode rewards
- Indicates learning progress
- Helps identify convergence

7.2 Win Rate Progression

- Displays moving average of success rate
- Tracks improvement over training
- Validates learning effectiveness

---

8. Technical Implementation Details

8.1 Numerical Stability

- Implements scaling in forward-backward passes
- Adds small epsilon (1e-10) to prevent division by zero
- Uses smoothing in probability distributions

8.2 Memory Management

- Bounded replay buffer (deque with maxlen=10,000)
- Lazy Q-table initialization with defaultdict
- Efficient state key hashing with frozensets

8.3 Fallback Mechanisms

- Corpus-wide letter frequency for unknown word lengths
- Random selection when no valid actions available
- Graceful handling of missing test file

---

9. Strengths and Limitations

9.1 Strengths

- **Hybrid Approach**: Combines domain knowledge with learned behavior
- **Scalability**: Separate HMMs for different word lengths
- **Robustness**: Multiple fallback strategies
- **Interpretability**: Probabilistic reasoning is explainable

9.2 Limitations

- **Tabular Q-Learning**: Doesn't generalize across similar states
- **Memory Constraints**: Q-table grows with state space
- **Fixed Weights**: HMM/Q-weight ratio (0.7/0.3) is hardcoded
- **Independence Assumption**: Doesn't model letter co-occurrence beyond bigrams

---

10. Potential Improvements

10.1 Architecture Enhancements

- Replace tabular Q-learning with Deep Q-Networks (DQN)
- Implement higher-order n-gram models (trigrams, 4-grams)
- Add prioritized experience replay
- Use double Q-learning to reduce overestimation

10.2 Feature Engineering

- Include letter frequency features
- Add word pattern similarity metrics
- Incorporate common prefix/suffix patterns
- Use pre-trained word embeddings

10.3 Training Optimizations

- Implement curriculum learning (easy to hard words)
- Add adaptive weight adjustment between HMM and Q-values
- Use meta-learning for quick adaptation
- Ensemble multiple agents

---

11. Conclusion

This Hangman AI agent demonstrates effective integration of classical probabilistic models (HMMs) with modern reinforcement learning (Q-Learning). The hybrid approach leverages the strengths of both paradigms: HMMs provide strong linguistic priors while Q-Learning enables adaptive behavior through experience.

The system achieves reasonable performance through careful reward shaping, experience replay, and numerical stability techniques. While tabular Q-learning has limitations in state space generalization, the combination with HMM probabilities provides a solid foundation for intelligent letter selection.

Future work could explore deep reinforcement learning architectures and more sophisticated language models to further improve performance and generalization.
