Project Hangman: A Journey Through 12 Guessing Strategies

This document outlines the evolution of my strategy for the Trexquant Hangman challenge. My approach involved an iterative process, starting with a foundational deep learning model and progressively refining it with various heuristics, hybrid systems, and advanced algorithms.

I have used **Bidirectional Long Short-Term Memory (Bi-LSTM)** neural network, trained on the provided dictionary. This type of network is excellent at understanding sequential patterns, as it can read the word's structure (e.g., `_ p p _ e`) both forwards and backwards to predict the missing letters.

VERSIONS

V1: The First Neural Attempt

- **Core Idea:** My first custom agent. It uses a simple Bi-LSTM model to predict the probability of each character for every blank space. For the very first guess, it falls back to a simple letter frequency count based on word length.
- **How it Works:** For a word like `g _ _ s s`, the model analyzes the pattern and outputs probabilities for 'a'-'z' at the blank positions. The agent then picks the unguessed letter with the highest total probability score across all blanks.
- **Performance:** < 50%
- **Takeaway:** A huge leap over the baseline. The model clearly learned underlying English word structures, but its raw predictions could be improved.

V2: The Regularized Model

- **Core Idea:** An architectural improvement on V1. This version uses a more powerful Bi-LSTM (larger layers) and introduces **dropout**, a regularization technique to prevent the model from "memorizing" the training data. This helps it generalize better to the unseen words in the test set.
- **How it Works:** The guessing logic is identical to V1, but the underlying model is more robust and less prone to making strange predictions on unfamiliar patterns.
- **Performance:** < 50%
- **Takeaway:** A more stable model is essential, but a better guessing strategy is needed to push the performance higher.

V3: The First Hybrid Agent (The Chosen One)

- **Core Idea:** This was a breakthrough. Instead of relying solely on the LSTM, this agent combines the strengths of the neural network and the original dictionary-filtering method.
- **How it Works:** It uses the LSTM for the early and mid-game when there are many unknown letters and thousands of possible words. However, once the dictionary has been filtered down to a small, manageable number of candidates (fewer than 750), it **switches** to the simple frequency-counting method on that small list. This is much more precise in the endgame.
- **Performance: ~50%**
- **Takeaway:** This hybrid approach proved highly effective. The LSTM acts as a powerful generalist to navigate the early game's uncertainty, while the dictionary filter acts as a precise specialist to secure the win in the endgame. This version was selected for the final submission due to its high performance and robustness.

V4: The Confident Hybrid

- **Core Idea:** A refinement of the V3 hybrid switch. Instead of switching based on a fixed number of candidates, it switches only if it's *confident* in the dictionary's suggestion.
- **How it Works:** It will only switch to the dictionary-based guess if the most common letter in the candidate words appears in a high percentage (e.g., >25%) of them. It also featured a smarter opening guess sequence.
- **Performance:** ~49-50%
- **Takeaway:** A very strong performer, but the extra complexity of the "confidence" check didn't provide a significant boost over V3's simpler and effective threshold.

V5: The Information Theorist

- **Core Idea:** An experimental endgame strategy. Instead of guessing the most frequent letter, it tries to guess the letter that provides the most *information* by best splitting the remaining candidate words into smaller groups. This is a classic "minimax" approach.
- **Performance:** ~43%
- **Takeaway:** While theoretically powerful, this strategy was brittle. It is highly dependent on the candidate word list being accurate. If the secret word wasn't in the training dictionary, this method could make bizarre guesses, leading to more losses.

V6: The Length-Adaptive Agent

- **Core Idea:** This version focuses on improving the crucial first guess. It pre-calculates the best opening letters not just in general, but specifically for each possible word length.
- **How it Works:** For a 5-letter word, it might start with 's', but for a 10-letter word, it might start with 'e'. The rest of the logic is similar to the V4 hybrid.
- **Performance:** ~48%
- **Takeaway:** Optimizing the first guess is a good idea, but its impact is limited. The core mid-game and endgame logic remains the most important factor.

V7: The Three-Phase Agent

- **Core Idea:** An attempt to combine the best of all worlds: the length-adaptive opening (V6), the LSTM model (V2), and the information-theoretic endgame (V5).
- **Performance:** ~31%
- **Takeaway:** This was a step backward. The brittle nature of the V5 information-gain endgame proved to be a major flaw, dragging down the entire strategy. It was a good lesson that simply combining complex parts doesn't guarantee a better system.

V8: The Corrected Agent

- **Core Idea:** A bug-fixed version of V7. I found and corrected several logical flaws in the information-gain algorithm.
- **Performance:** ~45%
- **Takeaway:** The fixes brought performance back to a respectable level, but it still didn't beat the simpler and more robust hybrid models like V3 and V4. This confirmed that the information-gain approach was a dead end for this problem.

V9: The Robust Hybrid

- **Core Idea:** A return to the successful hybrid model. This version combines the best proven components: the length-adaptive opening from V6 and the confident hybrid logic from V4.
- **Performance:** ~49%
- **Takeaway:** A very polished and high-performing agent, but it ultimately performed on par with V3. This suggested I was reaching the peak performance achievable with this class of hybrid strategy.

V10: The Fusion Agent

- **Core Idea:** Instead of switching between the LSTM and the dictionary, this agent **fuses** their predictions on every turn.

- **How it Works:** It gets a score for each letter from the LSTM and a score from the dictionary frequency. It then combines them using a dynamic weight. Early in the game (many candidates), it trusts the LSTM more. Late in the game (few candidates), it trusts the dictionary more.
- **Performance:** ~47%
- **Takeaway:** An elegant concept, but it was difficult to tune the weighting function perfectly, and it didn't outperform the simpler V3 switch-based model.

V11: The Ensemble Agent

- **Core Idea:** The most complex strategy attempted. It combines three "experts"—the LSTM, the dictionary frequency, and the information-gain solver—and fuses their outputs using a set of hand-tuned weights that change based on the game state.
- **Performance:** ~47%
- **Takeaway:** This was a case of over-engineering. The complexity added more points of failure and did not lead to better performance, confirming that simpler, robust strategies were superior.

V12: The Contextual Rule-Based Agent

- **Core Idea:** An attempt at a completely different method. It pre-computes common n-gram patterns (e.g., in the pattern `a_p`, the blank is most likely to be `p`, as in 'apple'). It uses these contextual rules as a high-priority guess.
- **Performance:** ~49%
- **Takeaway:** This was an interesting side-experiment. While n-grams can capture local context, this approach is less flexible than a full Bi-LSTM and struggles with longer-range dependencies and the start/end of words. It was ultimately shelved in favor of the more successful hybrid LSTM models.