FinalProject_LAW

July 5, 2024

1 AI Wordle Solver with Reinforcement Learning

1.1 Group members

- Luke Skerrett
- Alex Vo
- WonJae Lee

2 Abstract

The goal is to implement and evaluate reinforcement learning algorithms on a gymnasium environment for playing the Wordle game with reinforcement learning agents. The algorithms being tested are Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO). The evaluation involves training these models on the Wordle gymnasium environment and comparing their performance based on mean reward over a series of training steps. The results highlight the effectiveness and efficiency of each algorithm, providing valuable insights into their applicability for solving the Wordle game within the typical 6 guesses and having an average success rate of 3 guesses.

3 Background

Wordle is a game that roughly 2 million [1] people play everyday, and it is a web service hosted by the New York times. We all remember in December 2021, when the game rose to fame seemingly out of nowhere! We would like to develop an agent using reinforcement learning that can successfully solve Wordle efficiently and effectively and that would solve the Wordle under the average number of guesses of 4 guesses. Upon doing some research on these agents, we discovered a few articles that help show which methods are best to employ. In an agent designed by Andrew Ho, he utilized the A2C algorithm, which acheived 99% effectiveness [2]. It is a policy gradient method that uses both value functions and policy optimization to achieve efficient learning. It is an improvement over the basic actor-critic methods by using the advantage function to reduce the variance in policy updates. Another agent that would prove to be effective in our project development would be Proximal Policy Optimization (PPO) [3]. It is an advanced policy gradient method designed to maintain a balance between exploration and exploitation, and ensure stable and reliable policy updates. It improves upon the basic policy gradient methods by using a clipped surrogate objective to limit the size of policy updates. The game state is represented as a one-hot encoding (whether the given letter goes in that position or not).

4 Problem Statement

Can certain AI algorithms consistently outperform a human when playing the game Wordle? This can be measured by comparing the average number of guesses between the two different AI algorithms we choose to solve the Wordle game and compare their success rates and average number of guesses to that of humans, and will be replicable by running the study multiple times from a list of different Wordle word sets.

5 Data

We adapted the Gym Wordle environment from the following GitHub: https://github.com/zach-lawless/gym-wordle. This Gym implementation of Wordle has a simple environment of displaying the Wordle game in the terminal. Correct letter guesses are marked green, wrong positions letter guesses are marked yellow, and incorrect guesses are marked black/grey. The dataset we used is 5_words.txt, a text file with a list of 5 letter words from this implementation. Since our data comprises of just a list of 5 letter words, data cleaning was not necessary, and we could directly use it in our project.

6 Proposed Solution

The goal of this project is to apply reinforcement learning algorithms to solve the Wordle game. We aim to train RL agents to effectively guess the hidden five-letter word using the feedback provided by the game. The RL algorithms we use are Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) which are suitable for handling the complex state and action spaces of Wordle. First, we would create a custom Gymnasium environment for Wordle, which will provide the necessary interface for RL agents to interact with the game. Then we train the agents and evaluate their performance using metrics such as Success Ratio and Guesses Per Success. The training procedure would be to first set up the Wordle environment then select the algorithm to use to solve the Wordle, then set training steps and evaluation frequency. Next is the testing steps where we would reset the Wordle environment and load the saved A2C and PPO models and then run the evaluations to play multiple episodes or games of Wordle. We would then collect and record the rewards, number of steps to guess the word, and success rate. And finally, analyze and compare the collect metrics between the two RL algorithms to see which is the more efficient and effective algorithm for solving the Wordle game and compare that to the success ratio and guesses per success of humans. This solution leverages a custom environment, advanced RL algorithms, and a thorough training and evaluation process. The implementation and testing plan ensures reproducibility and provides a clear path for performance comparison, offering insights into the applicability of these RL algorithms to complex word-guessing tasks such as Wordle.

In order to implement A2C and PPO algorithms to solve the Wordle game, we modified the original wordle environment to give correct feedback for our AI to use in the future guesses. We kept track of correct position letters, wrong position letters, and wrong letters. Correct position letters are the correct guesses on both the letter and position, which are indicated as green letters in Wordle. Wrong position letters are the letters that are in the word but guessed on wrong position, which are indicated as yellow letters in Wordle. Wrong letters are the letters that are not in the word, and they are indicated as black/grey in Wordle. With this implementation, we can keep the correct letters in the position for the future guesses, and try different guesses for wrong position guesses

while avoiding the wrong letters. After these changes are applied, we can directly use A2C and PPO as they both take MultiDiscrete action spaces.

7 Evaluation Metrics

7.1 Summary

The evaulation metrics that we utilized were fitting for our project, and they has been used on other wordle AI related projects before. We ended up using **Success Ratio** along with **Guesses Per Success** over 100000 games. We think that these are valid metrics to use since when we think about an AI playing a wordle, the first question you ask is "did it get the word?" The obvious next question is "in how many guesses did take?". We believe that we represent these questions perfectly with our metrics and will show how they are great quanitifiers for our agents. The **Success Ratio** is determined by the wins vs losses that an AI attains along 100000 games, and the guesses per success are the amount of guesses that each successful run took. We additionally used some common metrics to analyze the models: **mean**, **standard deviation**, and **average number of guesses** to measure the effectiveness, consistency, and efficiency of each model respectively. ## Mathematical Notation Success Ratio = $\frac{\sum_{i=1}^{100000} w_i}{100000}$ where w_i is one hot encoded, $w_i = 0$ is a loss and $w_0 = 1$ is a win

Guesses per Success = $\frac{\sum_{i=1}^{G} g_i}{G}$, where G is the amount of games won and g_i is the amount of guesses.

8 Results

8.0.1 First Attempt at Wordle AI

We first were lost when it came to the implementation and took a while to achieve a solid a plan on how to implement wordle… should we make it a single player game? should we have a VS AI mode? How will we interface with the game and how will we implement the AI portion? We first downloaded a complicated GUI, and tried to poke at it… to no avail, we decided against the complicated GUI, and researched Github until we stumbled upon the gym_wordle repository, which allowed for a seamless API integration with the project.

8.0.2 Algorithm Selection

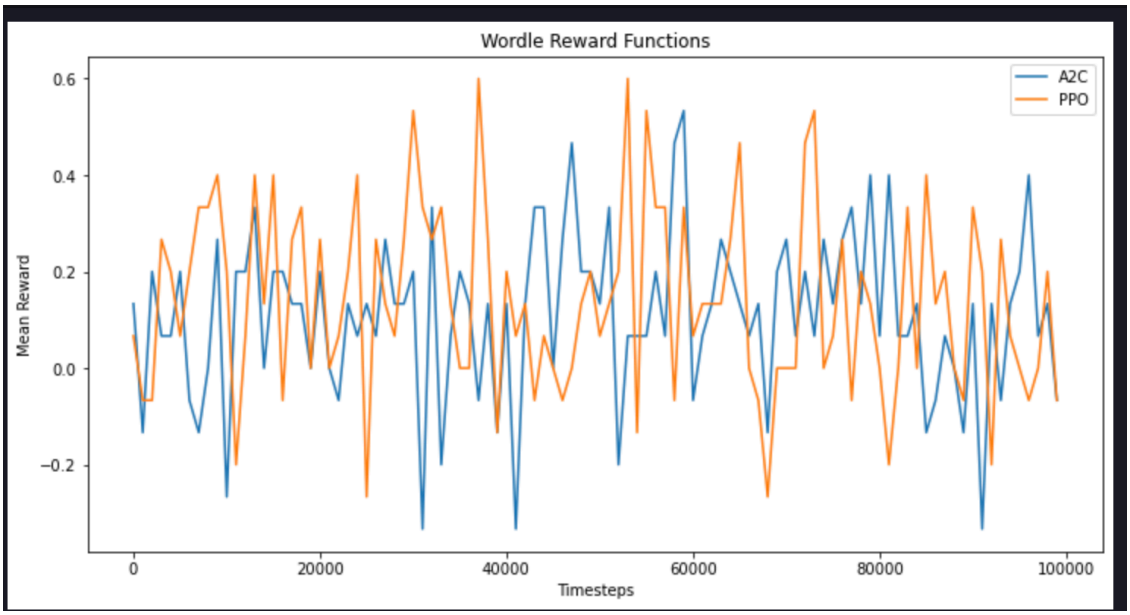
We were initially uncertain about which algorithms to and spent some time brainstorming, researching, and meeting to discuss which to use. Should we go for Q-Learning or try something more complex like DDPG? It was a question of not only how well would these models handle our environment, but how would efficiently would they be? We researched and tried Q-Learning which we went over a lot in class, but we realized it struggled with the complexity of our action spaces. DDPG seemed like another option but was a little hot and cold as it required a lot of tuning and could become unstable while training. After doing a lot of trial and error we came across both A2C and PPO. A2C offered an efficient way to train across multiple agents simultaneously which we knew would give an even approach between bias and variance. PPO impressed us with its ability to manage policy updates smoothly without excessive tweaking. In the end, we chose A2C and PPO for their practical benefits in balancing efficiency and stability, which perfectly aligned with our project's needs.

8.0.3 Model and Training Implementation

```python
env = Monitor(WordleEnv())

models = {"A2C": A2C, "PPO": PPO}

n_steps = 100000
eval_freq = 1000

def train(models, n_steps, eval_freq):
    results = {}
    for alg_name, alg in models.items():
        model = alg("MultiInputPolicy", env, verbose=1)
        start_time = time.time()
        rewards = []
        for step in range(0, n_steps, eval_freq):
            model.learn(total_timesteps=eval_freq, reset_num_timesteps=False)
            mean_reward, std_reward = evaluate_policy(model, env, n_eval_episodes=30)
            rewards.append(mean_reward)
            training_time = time.time() - start_time
            results[alg_name] = np.mean(rewards), np.var(rewards), rewards, training_time
            model.save(f"{alg_name}_wordle")
    return results

A2C_PPO = train(models, n_steps, eval_freq)

def extract_mean_std(results):
    extracted_data = {}
    for key, value in results.items():
        mean_reward, std_reward, _, training_time= value
        extracted_data[key] = (mean_reward, std_reward**2, training_time)
    return extracted_data

A2C_PPO_mean_std = extract_mean_std(A2C_PPO)
print(A2C_PPO_mean_std)
```

We decided to implement both of the models one after the other to get a good idea of the differences between the two. Above is the code that runs each of the models, and extracts the std and mean to graphs.

8.0.4 Mean Reward Graph



Both models exhibit considerable fluctuations in performance, with A2C showing wider variability and occasionally dipping below zero. PPO achieves higher peaks in mean rewards shortly but is more stable overall. A2C's synchronous updates lead to greater exploration but lead to a more variant performance, whereas PPO's "safer" policy updates yield more stable yet oscillating and shifting

results. These differences are a result of the underlying attributes of each algorithm's updating and their strategies for balancing exploration and exploitation in the wordle environment.

8.0.5 Success Ratio and Guesses Per Success

Success ratio over 100000 games using **A2C**: 0.57209 with 2.9488 guesses on average

Success ratio over 100000 games using **PPO**: 0.57323 with 2.94933 guesses on average

9 Discussion

9.0.1 Similar Results Between A2C and PPO

We observed similar results using A2C and PPO, with success ratios of 0.57463 and 0.57208, respectively, and an average of around 2.95 guesses per success. A2C's parallelized actor-critic approach efficiently balances exploration and exploitation, leading to great policy learning. PPO's clipped objective function provides stable policy updates, preventing large deviations and maintaining control. Both algorithms excel at managing the trade-off between exploration and exploitation. These complementary strengths resulted in nearly identical performance in our complex environment, partly due to the shear amount of games we played, they both deviated to a similar success rate.

9.0.2 Performance Difference Between Human and Our AI

Our AI models A2C and PPO, appear to be worse than human players in Wordle, who typically solve the puzzle in an average of 4.71 guesses with only 2.92% failing to solve the game[4]. Our AI achieves a success ratio of around 0.57 with about 5.15 guesses on average per game.

9.0.3 RL: Good for Complex Environments

Reinforcement Learning (RL) excels in complex environments like Wordle due to its ability to learn and adapt from cumulative feedback, refining strategies over time. Algorithms like A2C, PPO, SAC, and DQN can develop nuanced policies by learning from large amounts of games while capturing complex patterns and long-run paradigms that are crucial for solving puzzles efficiently. One of our biggest discoveries was that this adaptive learning capability allows RL models to surpass past approaches by continuously updating their strategies based on advancing given game states.

9.0.4 Limitations

We can say that since these models were trained on a specific set of words that they may struggle with words not in the original set. Also, the success rate of solving the Wordle game over 100,000 runs was around only 57% for both algorithms, which is slightly above half of the time. Considering the random guesses for the same number of iterations resulted in 0.004% success and 5.9998 guesses on average, it is significantly improving the performance. However, this may be improved by training on more data and longer training timesteps.

9.0.5 Ethics & Privacy

We couldn't find any pressing issues for ethics and privacy regarding our Wordles models, but we would like to state how we can address potential issues.

In the future, we will guard informed consent, making sure all parties know of their involvement in AI. We will always value data security, and refuse to work in environments where security may be an issue or may be breached.

9.0.6 Conclusion

A2C and PPO produced nearly identical results in our AI Wordle experiments, with success ratios around 0.57 and averaging approximately 2.95 guesses per game - due to their effective joining of exploration and exploitation. This convergence is attributed to their strong policy and updating mechanisms and extensive game play. Unfortunately, we didn't see our models outperforming humans with our current implementation, but there still is room for improvement.

10 Footnotes

1.[^](#tettamanti): Tettamanti, T. How To Solve Wordle Using Machine Learning. *Rootstrap*. https://www.rootstrap.com/blog/how-to-solve-wordle-using-machine-learning 2.[^](#post): Wordle players are cheating every day, mathematician says. *The Washington Post* https://nationalpost.com/news/wordle-cheating-claims#:~:text=Put%20another%20way%3A%20Of%20the,that%20might%20rise%20to%201%2C320.
3.[^](#ppo): How PPO Algorithm works. https://medium.com/@oleglatypov/a-comprehensive-guide-to-proximal-policy-optimization-ppo-in-ai-82edab5db200#:~:text=PPO%2C%20or%20Proximal%20Policy%
4.[^](#ppo): Average Wordle Scores. https://wordsrated.com/average-wordle-scores/#:~:text=The%20odds%20of%20solving%20a,tries%20is%2033.10%25%20on%20average.

[ ]: