Solving Wordle Using Deep Reinforcement Learning

Anu Kumar
kumaram@umich.edu

Hardik Patil
hardikyp@umich.edu

Rohan Sequeira
rohseque@umich.edu

Shreyas Bhat
shreyasb@umich.edu

Sonali Mohanty
msonali@umich.edu

Abstract

Wordle is a popular online game where the player has six tries to guess a 5-letter word. Deep reinforcement learning has recently been used to play various games such as Atari Pong and AlphaGo effectively but is yet to be applied to Wordle. This paper uses the Advantage Actor-Critic (A2C) deep reinforcement learning algorithm to create a model that solves Wordle for different lengths of words. Through our work, we highlight the merits and shortcomings of the A2C implementation on character-level and word-level prediction models for Wordle.[1]

1 Introduction

Wordle is an online word game where a player gets six tries to guess a 5-letter word. During each turn, the player is notified if the letters in the guessed word are present in the target word when the tile turns yellow or green. Yellow tiles indicate that the letter is in the target word but not in the correct position. Green tiles indicate that the guessed letter is present in the correct position. In contrast, a gray tile indicates that the letter is not present in the target word. If they can guess the word within the six attempts, the player wins and loses otherwise.

Since the release of this game in October 2021, Wordle has gained tremendous popularity among puzzle enthusiasts and mathematicians alike. Various approaches have been tried to develop an optimal winning strategy to solve Wordle in as few tries as possible, including Information Theory, which has emerged as the best way to solve it so far. Furthermore, a very detailed analysis of human performance has also been documented over a period of time (Lesser [2022]). However, these models have only been employed to solve the game when the correct answer belongs to one of the 2315 5-letter words present in the official Wordle dictionary. The difficulty increases exponentially when the agent is allowed access to the $\sim$13000 5-letter words in the English dictionary.

We implemented the Advantage Actor-Critic (A2C) Deep Reinforcement Learning method to solve Wordle to get close to the performance of the Information theory approach in this project. We experimented with Wordle rules to see if the A2C model can play Wordle in scenarios where the possible answer is not restricted to a word present in the 2315 5-letter dictionary of the official Wordle but $\sim$13,000 possible 5-letter words in the English dictionary. Furthermore, we implemented Wordle for different word lengths (4 and 6 letter words) to observe if the model could maintain the same winning rate and average number of turns to win for the mentioned scenarios.

2 Related Work

The Wordle game consists of 2,315 words as possible answers, and there are 12,972 possible guess words that the game will allow a player to make. Given the game's popularity, there has been a rising interest in solving Wordle using various mathematical methods. However, there are not many published research papers and state-of-the-art methods used for solving this game yet. There has also been a detailed analysis of how humans have performed since the game was first released (Lesser [2022]). In the United States, the average number of guesses to win is 3.92, while the worldwide average is 3.919.

[1] Code: https://github.com/rohsequ/Deep-RL-Wordle- Different models on different branches

A very well-known approach to solving Wordle is 3Blue1Brown's Information Theory model (Sanderson [2022]). The algorithm aims to add the most amount of information (reduce uncertainty measured in bits) about the word with each guess. Every word in the possible guess list can be assigned an entropy or expected information gain as certain letters and words are more likely than others in the English language. After each guess, the list of possible words can be updated based on the feedback from the game (green, yellow, and gray). The next guess will be the word on the reduced guess list with the highest entropy, and this process will continue until the guess is correct or the number of tries runs out. This model was able to guess the word in an average of 3.60 tries when the possible guess list consisted of the 13,000 most common English words and 3.43 tries when the possible guess list consisted of only the *possible Wordle words*.

Another blog post titled Automatic Wordle Solver by Gafni [2022] discusses the use of greedy min-max and average minimize approaches to solving Wordle. This method employs information theory as well to solve the problem. The greedy min-max approach chooses a guess that minimizes the largest possible remaining word list, while the average minimize approach chooses the guess that minimizes the average length of the remaining words list. The first approach can give a worst-case performance of 5 tries and the second one gives a worst-case performance of 6 tries. This model, however, needs the agent to know the exact 2315 words used by the Wordle game and chooses its guess from this restricted list of 5-letter words. It was also found that the worst-case performance for hard mode is 8 tries.

Though Wordle appears to be a simple game, a paper by Lokshtanov and Subercaseaux [2022] recently proved that the natural formulation of the Wordle game is NP-hard. Even in cases with a small word length, namely 5 letters, it is NP-hard to approximate the minimum number of guesses required to guarantee that the correct word is guessed.

Our approach differs from the existing work in a way where we modify the game for both 4 and 6-letter words and apply the A2C algorithm to Wordle using Character-level and Word-level prediction models. We tried allowing the correct answer in the entire 5-letter dictionary rather than a small selected list. However, the model does not perform well when the action space is vast, and hence, we decided to stick to the original rules of Wordle.

3 Methodology

We solved this problem by creating our model based on the PyTorch Lighting implementation of deep reinforcement learning models (Wang) and applied them to a Wordle environment. The model uses an online learning approach and generates data as it plays the game.

The Advantage Actor-Critic (A2C) algorithm combines Policy-based and Value-based reinforcement learning algorithms. The architecture consists of an actor and a critic network. Given the current state, the actor-network generates a probability distribution over the action space. The action to be performed is then sampled according to this distribution during training. During the evaluation, the action with the highest probability is chosen. The critic network tries to learn the value function for the model. It provides a value for each state which is then used to train the actor. In A2C, the loss function is a weighted sum of the actor loss, the critic loss, and the entropy loss.

$$L = L_{actor} + \beta_{critic} L_{critic} + \beta_{entropy} L_{entropy}$$

An advantage function is used to compute the actor loss. It is the difference between the discounted cumulative rewards we got across a batch of state transitions and the value predicted by the critic network.

$$G_t = r_t + \lambda G_{t+1},\ \forall t \in [0, n]$$

$$L_{actor} = -\frac{1}{n} \sum_t (G_t - V_t) \log(\pi(a_t|s_t))$$

The critic loss is computed by the mean squared error between the cumulative discounted reward across a batch and the predicted value by the critic network.

$$L_{critic} = \frac{1}{n} \sum_t (V_t - G_t)^2$$

Finally, the entropy bonus is computed to encourage exploration by the actor.

$$L_{entropy} = -\frac{1}{n} \sum_t \pi(a_t|s_t) \log(\pi(a_t|s_t))$$

Here, G_t is the empirical return over the batch of transitions stored in the buffer, starting from state s_t in the batch. r_t is the reward at time step t in the batch, a_t is the action at time t in the batch, n is the batch size, V_t is the output of the critic network which is the "value" of state s_t, $log(\pi(a_t|s_t))$ is the log probability of choosing action a_t at state s_t according to the actor network.

3.1 A2C Pipeline

Deep reinforcement learning has been widely used to play computer games. Traditional supervised machine learning algorithms are trained with a specific target in mind. However, the optimal next action in online games is unknown, making it difficult to use a traditional model. Reinforcement learning consists of an environment (game) and an agent (the player). For every action, the agent receives feedback and can be trained to make better, and eventually optimal, decisions for the next action, even when the agent has never seen this particular action before. Deep reinforcement learning uses a deep neural network that can maximize the game score with unstructured data input. This reinforcement learning approach has worked to solve a variety of games such as Pong, Go, and Sonic the Hedgehog (Simonini [2018]).

One common method for deep reinforcement learning in playing games is Deep Q Learning (DQN). However, this model is slightly outdated and only value-based. *Policy Gradients* is a policy-based method that has been used to solve a variety of games as well but has a major flaw as it is challenging to find a good way to compute how good a policy is.

A2C is particularly advantageous in the situation of Wordle because it combines value-based methods and policy-based methods and is better for discrete state spaces. The actor generates a probability distribution over the action space given the current state, and the critic tries to learn the value function for the model. A2C's loss function is a weighted sum of the actor loss, critic loss, and entropy loss.

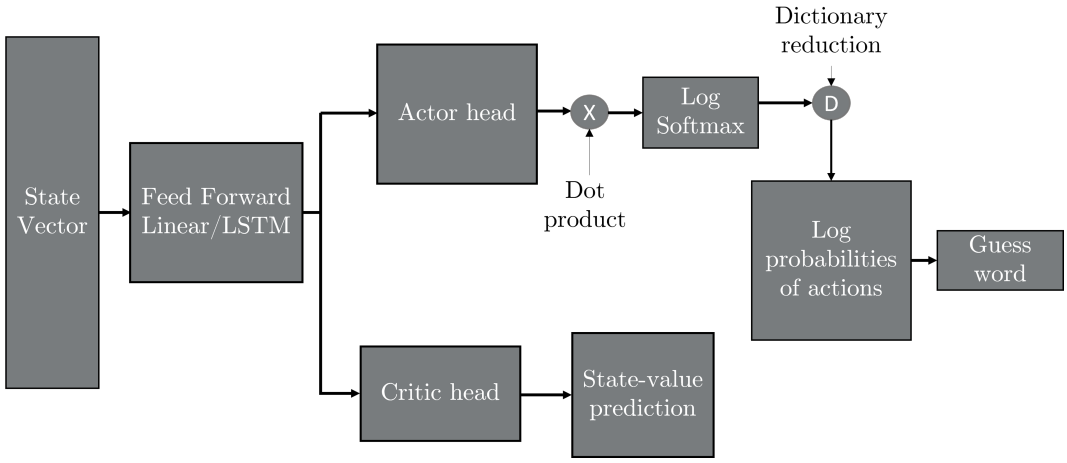


Figure 1: A2C network architecture

Figure 1 demonstrates the pipeline for the A2C model. The system starts out with a state vector of dimension `[1+26+26*3*word length]` for character-level, or `[1+26+26*word length]` for word-level. The state vector keeps track of number of turns remaining in the game, characters used and unused up to the current turn, and information about the letters being present, maybe present and absent at a certain position in the target word. For the word-level model, this state vector is passed into the feed-forward block which then goes into the actor head which gives the log probabilities of all the words for a given word length. The output from the log-softmax goes through dictionary reduction. The final log gives us log-probabilities of all possible actions (words for the next turn, in our case) and results in the guessed word. For the character–level model, we take the log probabilities over individual letters in the output of the actor head to get the guess word. The output from the feed-forward layer is passed through the critic head which tries to predict values for the given state.

3.2 State Representation

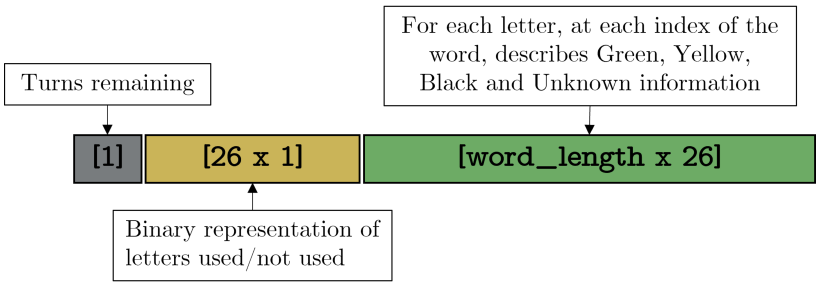
We use two different state representations for the character-level and word-level models (as shown in Figure 2). The main difference between them is the length of the vector used to represent the state and how the information about the possible letters in the goal word is stored.

3.2.1 Word-level Wordle model

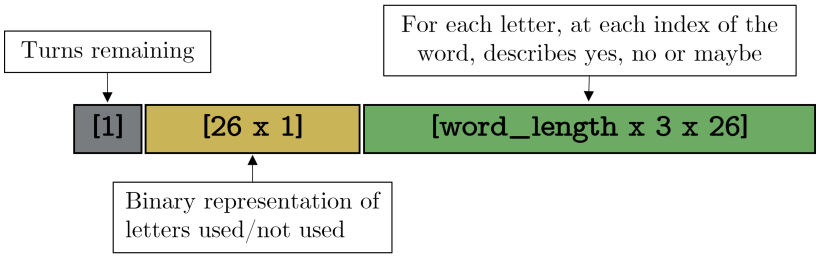
In this representation, the first value denotes the number of turns remaining in the game. The next 26 values represent whether a letter has been tried yet or not and the final block of length `[word_length*26]` represents whether a letter at a specific position is *green* (1), *yellow* (0.75), *gray* (0.5), or *unknown* (0.25).

3.2.2 Character-level Wordle model

The first 27 values in this state representation are identical to the ones used for the word-level model. The final block of length `[word_length*3*26]` keeps track of whether a letter is possible (`[0, 1, 0]`), definitely present (`[1, 0, 0]`), or definitely not present (`[0, 0, 1]`) at a particular index.



(a) Word-level



(b) Character-level

Figure 2: State representations used in our models

3.3 Action Representation

At each stage of the game, the actor can choose any of the available words in its vocabulary. For example, in the 5-letter Wordle game case, the actor's vocabulary consists of all 13,000 5-letter words in the English language. Instead of outputting vocab size values from the actor-network, we only output a `[26*word_length]` vector. For the character-level mode, we directly take the softmax over each 26-wide block of the output to get the probabilities of each letter at each position in the word. For the word-level model, we then encoded all the words in the vocabulary into `26*word_length` vectors. Each of the 26-wide groups would be a one-hot vector representing a particular letter at a particular position. To give a simple example, consider the two-letter word 'to.' It will be encoded into a 52-dimensional vector where the 20th and the (26+15th) positions would be 1. We get the log probabilities of each action by taking the dot product of the output of the actor-network with each of these encoded words.

3.4 Reward Function

Figure 3 shows the types of rewards and their associated reward values we set. Reward is handed out only for unique greens, yellows and grays. There is a step reward applied from the 3rd turn onward, which helps to encourage exploitation. There is no reward for guessing the word correctly on the 1st turn. If the word is guessed correctly on the first turn, the model does not learn, therefore, this discourages the model from learning incorrect moves.

3.5 Model Variations

We created a character-level model based on A2C for 4, 5, and 6-letter versions of Wordle. We also created word-level model based on A2C for 4 and 5-letter versions of Wordle and trained them on linear and Bi-LSTM neural networks. The action space for the 4, 5, and 6 letter Wordle is as follows: 4-letter: 2348 words, 5-letter: 13000 words (2315 target words), 6-letter: 7000 words (2000 target words). The dictionary reduction functionality was introduced to reduce number of possible actions at each step to help train the model better.

Reward Description	Reward
Win Reward	+ 30
Step Reward	- 1
Loss Reward	- 50
Green Reward	+ 2
Yellow Reward	+ 1
Grey Reward	+ 0.1
Repeated Grey Letters Reward (character level)	- 3

Figure 3: Reward Function

Hyperparameter	Word-level (Linear)	Word-level (LSTM)	Character-level
Batch size	64	128	128
Learning rate	10^{-4}	10^{-4}	5×10^{-5}
Hidden layer size	512	1024	512
Number of hidden layers	5	2	2
`prob_cheat`	n/a	n/a	0.05
β_{critic}	0.5	0.5	0.5
$\beta_{entropy}$	0.01	0.01	0.01
γ	0.99	0.99	0.99
Size of replay buffer	1000	1000	1000

Table 1: Hyperparameters used for training

3.6 Environment

We created a Wordle environment to generate games for training the RL agent. The environment acts as a state updater. It takes in the current state and action and returns the next state, reward earned, and a boolean indicating whether the current game is finished or not. The environment selects a goal word from the list of allowable goal words that the environment is initialized with to start a game. Whenever it receives a state and an action, it compares the action with the goal word letter by letter and updates the state. A game is finished if either the goal word is guessed or no more turns remain for the player. We created separate Wordle environments with 4, 5, and 6-letter word dictionaries. We set the maximum number of tries that the player gets as 6, 6, and 7 for the 4, 5, and 6-letter cases, respectively.

3.7 Dataset

For our 5-letter models, we used the official Wordle list of possible target words and possible guesses since this is publicly available. We extracted data from the Google Books English n-gram public data set for the 4 and 6-letter words. We extracted a list of the 60000 most frequently used words in the English language and then extracted separate lists for 4-letter and 6-letter words. There were 2348 4-letter words in the 60000 most frequently used words (allowing all to be the target words). Similarly, there were 6997 6-letter words, and we chose the first 2000 to be target words. We chose the most frequent words to be target words as the official Wordle words are common while the possible guess words can be obscure.

4 Results

4.1 Loss ratio vs Number of iterations of training

In Figure 4, we plot the loss ratio as a function of the number of iterations of training for the different model variations when they have access to the full actions space of 4, 5, and 6-letter words. We define loss ratio as the ratio of the number of losses to the total number of games played within the last 200 games played. The action space size is 2348, 13000 (with 2315 possible goal words), and 7000 (with 2000 possible goal words) for the 4, 5, and 6-letter cases, respectively. As can be seen, the word-level models struggle to get a reasonable win rate when the action space is unrestricted and converge to a pretty high loss ratio. On the other hand, the character-level model seems to perform much better, although the number of training iterations is also higher. However, the training is much faster than that for the word-level models. We believe that this is the case because the character-level model can prune the action space

much better since it does not have to choose valid words at each turn. It is, therefore, able to acquire more information about the goal word compared to the word-level models. The word-level models use dictionary reduction, but when the action space is this huge, at every turn, the list of possible actions may be hundreds or even thousands of words long.

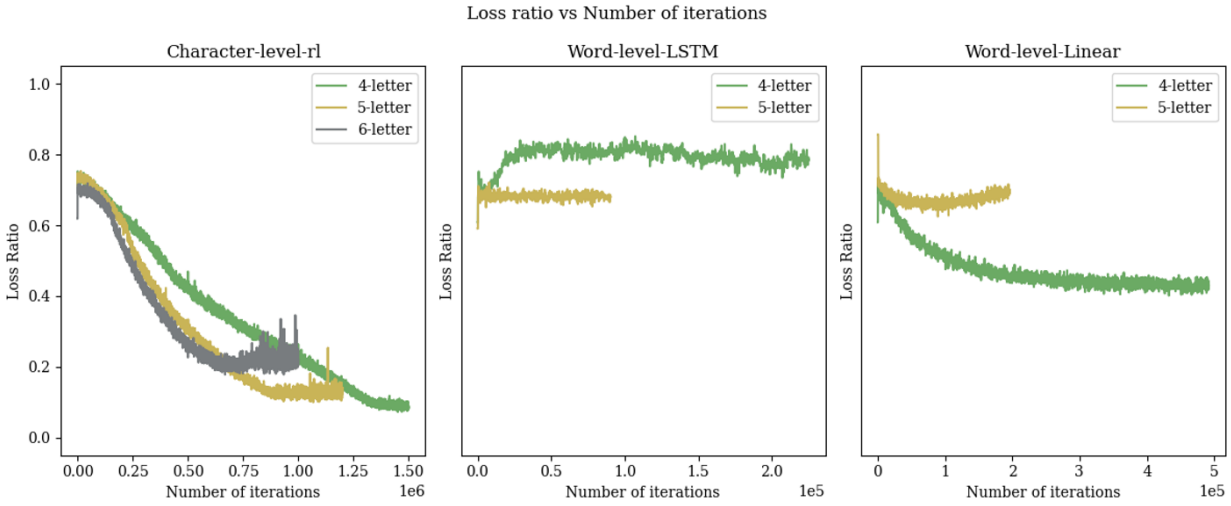


Figure 4: Loss ratio vs Number of iterations of training for the various model variations on the full action set

4.2 Win percentage on the full action space

Figure 5 shows the percentage of games won by the trained model on the full action space for 4, 5, and 6-letter Wordle games. The goal word ultimately defines a game. Hence, the number of games possible is the same as the number of goal words possible. We use a greedy version of the actor-critic network for playing the games once the network has been trained. It always selects the actions with the highest probability output from the actor-network. As shown in Figure 5, only the character-level model consistently wins around 80% of the games for all word lengths. However, it also takes a very high number of turns on average to win. The word-level models perform poorly but require fewer turns to win the games that it ends up winning.

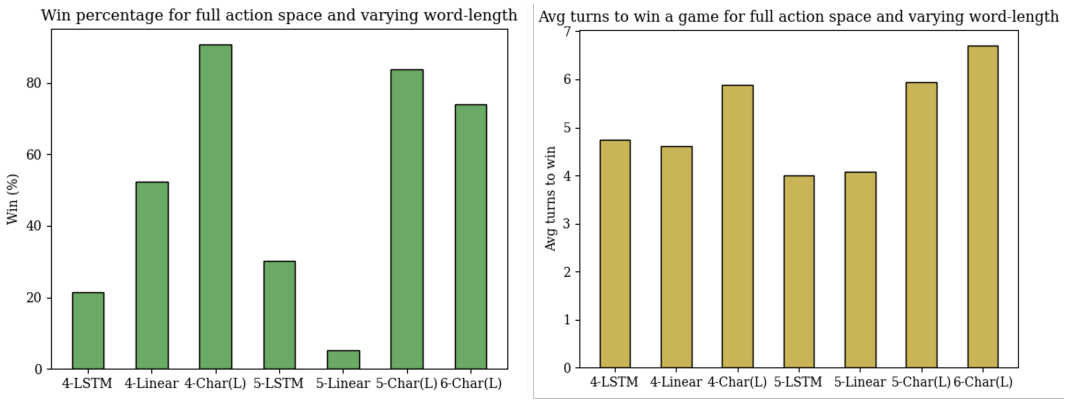


Figure 5: Percentage of games won and average turns to win after training is complete on the full action space for the various model variations

4.3 Example Games

Figure 6 shows some examples of wins and losses for the word-level model for 4 and 5-letter Wordle games. We observed that the model sometimes gets stuck at local minima, chooses the same guess word multiple times, and does not explore the action space to get to the goal word. Furthermore, in cases where the target word has repeated letters or rhymes with multiple words in the dictionary, the model has a difficult time guessing the correct word.

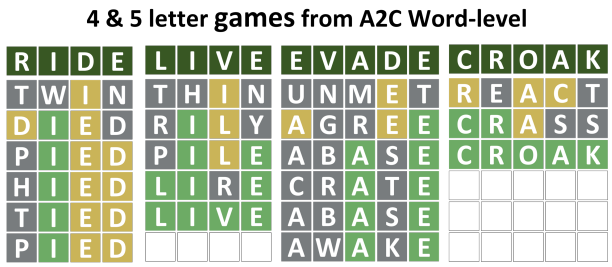


Figure 6: Examples of wins and losses for the word-level model on 4 and 5-letter Wordle games

Figure 7 shows some examples of wins and losses for the character-level model for 4, 5, and 6-letter Wordle games. In contrast to the word-level model, this model seems to retain information about each letter and explores the space for more information until it guesses the goal word. This, however, may not be the most optimal strategy for playing Wordle. Once we have some green letters, the optimal action may be to choose a word with letters other than the green ones to maximize information gain. However, our model seems greedy and retains the green letters since it thinks it can win in the next turn if it retains those letters. Furthermore, it is interesting that the model tends to learn and try actual English words, especially in the 4-letter Wordle case.

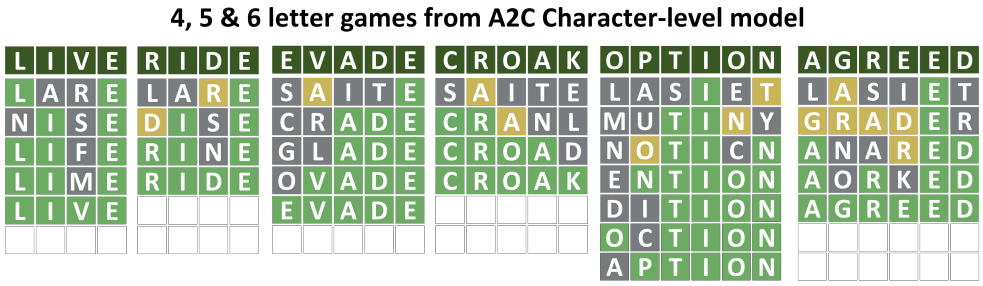


Figure 7: Examples of wins and losses for the character-level model on 4, 5, and 6-letter Wordle games

4.4 Effect of size of action space

As a next step, we tried to investigate the effect of restricting the size of the action space on the models. We noticed that this did not affect the character-level model since it does not depend on a dictionary of words. So, it still requires many iterations to start performing well. For the word-level models, we restricted the action space to only contain 10, 100, and 1000-words from the dictionary and trained the models for 10000 iterations. We noted that these models perform well when the size of the action space is limited, giving a reasonable win rate and requiring fewer turns on average to win a game. Figure 8 shows the win rate and the average turns to win a game for these cases.

5 Discussion and Conclusion

We created a character-level model for 4, 5, and 6-letter versions of Wordle and a word-level model for 4 and 5-letter versions of Wordle and trained them on linear and LSTM neural networks. We found that the character-level model performs well on the full action space when trained longer, but the average turns to win also increases. It also does not use valid words in each turn and hence does not follow the rules of official Wordle.

The word-level model performs poorly due to huge action space, gets stuck in local minimums, and suffers from vanishing gradients. The performance is better when the action space is small (less than 100 words). We observed that all models perform poorly when the target word rhymes with possible words from the dictionary.

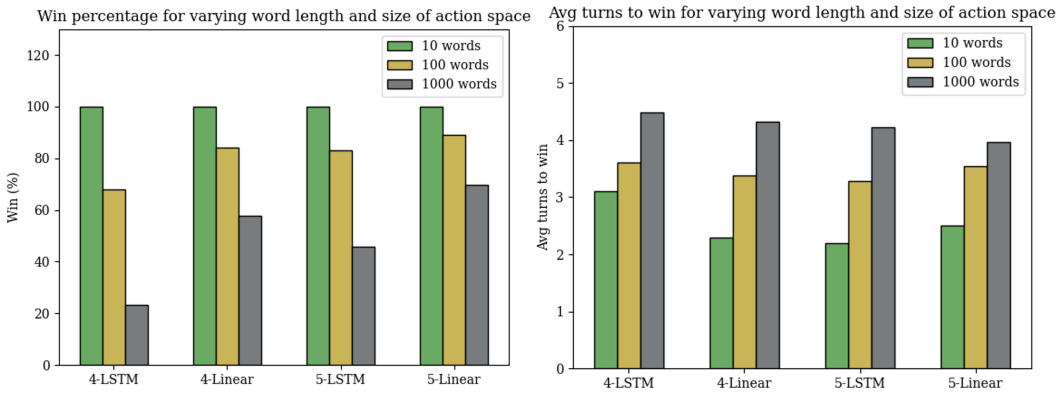


Figure 8: Percentage of games won and average turns to win in restricted action-spaces for the word-level models

6 Future Work

Various steps can be taken to improve the performance of our models. We can perform hyperparameter tuning by trying different discount factors for the rewards, weights for actor and critic loss, the batch size for training, the number of hidden layers, and the size of hidden layers. A more robust reward function can be designed, incorporating a penalty for repeating words in the word-level model and trying invalid words in the character-level model. Furthermore, sequentially training the model on increasing size of action space and retraining on words that the model fails to learn should prove fruitful.

Author Contributions

All authors contributed equally to the project, including literature review, logic design for the Wordle environment, programming, testing and data generation, and drafting reports and poster.

Repository

The code for this project can be found in **Deep-RL-Wordle** repository (Kumar et al.) Different branches contain codes for word-level and character-level models. We used Pytorch Lightning A2C implementation based on Ho [2022] code for a wordle solver.

We completely changed the state representation for our word-level model, implemented a new reward function, added the dictionary reduction feature, modified the original state representation for the character-level model and changed the way action is chosen for the character-level model.

References

Y. Gafni. Automatic wordle solving, Jan 2022. URL https://towardsdatascience.com/automatic-wordle-solving-a305954b746e.

A. Ho. Wordle-solver, 2022. URL https://github.com/andrewkho/wordle-solver.

A. Kumar, H. Patil, R. Sequeira, S. Mohanty, and S. Bhat. Wordle deep rl repository. URL https://github.com/rohsequ/Deep-RL-Wordle-.

R. Lesser. Wordle, 15 million tweets later, Mar 2022. URL https://observablehq.com/@rlesser/wordle-twitter-exploration.

D. Lokshtanov and B. Subercaseaux. Wordle is np-hard, Mar 2022. URL https://arxiv.org/abs/2203.16713.

G. Sanderson. Solving wordle using information theory, Feb 2022. URL https://www.youtube.com/watch?v=v68zYyaEmEA.

T. Simonini. An intro to advantage actor critic methods: let's play sonic the hedgehog!, Jul 2018. URL https://medium.com/free-code-camp/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d.

J. Wang. Pytorch lightning bolts. URL https://lightning-bolts.readthedocs.io/en/latest/deprecated/models/reinforce_learn.html#actor-critic-models.