Word Autobots: Using Transformers for Word Association in the Game Codenames

Catalina M. Jaramillo, Megan Charity, Rodrigo Canaan, Julian Togelius
Game Innovation Lab, New York University
370 Jay St, Brooklyn, NY 11201, USA
{cmj383, mlc761, rodrigo.canaan}@nyu.edu, julian@togelius.com

Abstract

Winning the social game Codenames involves combining cooperative and language understanding capabilities. We developed six cooperative bots designed to play the Codemaster and Guesser roles in the Codenames AI Competition and tested them using the provided framework and a round-robin tournament set. The bots are based on term frequency - inverse document frequency (TF-IDF), Naive-Bayes and GPT-2 Transformer word embedding. Additionally, Transformer-based bots were assessed and compared with the concatenation of word2vec and GloVe baseline bot developed by Codenames AI Competition creators. Results from this Transformer implementation rivals the concatenated bot in terms of win rates and guess precision and outperforms it in terms of minimum and average turns taken to win the game and training data load time. Additionally, in an initial evaluation performed with 10 human players, the Transformer agent performed slightly better than the baseline as Codemaster, but worse as a Guesser.

Introduction

Most AI research using games as testbeds use either classical board games such as Chess or Go, or video games, typically older ones (Yannakakis and Togelius 2018). But different games pose different challenges for humans and agents. Building agents for games which are less explored is interesting for multiple reasons, for example as a way to better understand the design of these games and which cognitive challenges they pose, to enable playtesting, balancing and content generation, or to create agents that are interesting to play with.

This paper explores bots for the social board game Codenames, where one of the main cognitive challenges is predicting how players on your team make word associations. While we are not the very first to build a bot for Codenames, we are the first to present a bot based on modern deep learning methods, we conduct a thorough empirical comparison, and we perform a user study with human players.

Codenames has two characteristics that are individually appealing and which together distinguish it from other game-based AI competitions and benchmarks:

- Natural language based. Many promising AI applications also hinge on language models, such as translation, reading comprehension, and natural language inference (Brown et al. 2020). Different environments, like LIGHT (Learning in Interactive Games with Humans and Text) platform (Urbanek et al. 2019) and AI Dungeon (Walton 2019), focus on AI language exploration and offer opportunities to further develop related applications. Codenames, as played between humans, uses language in a very rich way (homonyms, antonyms, rhymes, popular culture references, etc.) and relies in making inferences and can also be used with this objective.
- Cooperative in nature. Many real-world AI applications involve cooperation, not competition, with humans. Most game-based AI benchmarks are competitive, and even some of those that involve cooperation (e.g. GVGAI multi-agent (Gaina, Pérez-Liébana, and Lucas 2016), Pommerman (Resnick et al. 2018)) have had successful entries that largely ignore the partner agent and simply attempt to optimize one's own behavior. In Codenames, partner behavior is critical, as evidenced by the fact that codemaster-guesser pairs using the same modeling achieve much higher scores and win rates than mixed pairs. This is similar to what is observed in Hanabi (Walton-Rivers, Williams, and Bartle 2019; Bard et al. 2020), where although agents with shared conventions achieve near-perfect scores, cooperation with unknown agents is still an open problem.

Codenames, the Board Game

Figure 1: An example Codenames game board (right) and key (left). Each word is highlighted in the image (but not in the game) with the key's corresponding color.

Codenames (Chvatil 2015) is a board game where the objective is to identify a team's secret agents (represented by words on the board) in the minimum number of turns possible. Also, it is important to avoid the words that represent bystanders (losing the turn), opposite team's agents (losing the turn and adding a guess to the other team's count) and the assassin (losing the game). Each team consists of one Codemaster and one or more Guessers. The role of the Codemaster is to provide a clue word and number of secret agents related with that word, based on the Codemaster's knowledge of the board words' assignment. The role of the Guessers is to pick the own team's words from the board using the provided clue. This is a cooperative game, where the Codemaster of a team, using word association, picks a clue for the Guessers to identify as many of the own agents, from the 25 options in the board, while keeping the Guessers away from the wrong key names.

For the Codenames AI Competition framework, the subset of words corresponding to our team are the red words. The remaining words on the board are identified as bad words and are split in a subset of blue words (corresponding to the opposite team), civilian words (representing bystanders) and one word labeled as assassin.

Codenames, the AI Competition

The Codenames AI Competition Framework (Summerville et al.) tests AI's understanding of human language and communication capacity using the game Codenames as testbed.

It is a simplified version with a single team formed by a Codemaster and a Guesser. Participant bots are paired at random trying to complete the game in the least moves, with a 30 seconds limit per turn. For each turn, the Codemaster gives a clue -a word and a number of guesses (n); then the Guesser takes one or more guesses until (whichever comes first): the guesser passes the turn, n+1 guesses have been made, or a mistake has been made (a non "red" word was guessed). The penalty for guessing a bad word is a lost turn.

Scoring is based on the number of turns (lower score is better), and a lost round gets 25 points; following the approach used by Kim et al. (2019), the average number of turns is calculated using only won games. Competition baseline bots use word2vec (Mikolov et al. 2013), GloVe (Pennington, Socher, and Manning 2014), and WordNet models (Fellbaum 2012).

Related Work

Previous research with the artificial playing of Codenames by Kim et al. (2019) developed and compared performance for both Codemaster and Guesser bots using different word embedding models (WordNet, word2vec and GloVe), and different distance measures in a round-robin tournament. The WordNet based bot was excluded from the original study because of its poor performance. For the remaining bots, results were mixed, showing better performance for Codemaster agents that were paired with similar approach Guesser, and for agents that used a concatenated word2vec and GloVe strategy. Their word2vec strategy looked to create word vectors that predict both words and contexts, while their GloVe strategy looked to find co-occurences between words through weightings.

Outside of the Codenames challenge, the concept of identifying word relatedness and word contextualizing is a largely explored area of natural language processing. The TF-IDF algorithm was used by Wu et al. (2008) for making word query relevancy decisions based on both document-wide and local relevance. They developed a context-based ranking formula to sort probabilities accordingly. Jurafsky and Martin (2014) demonstrate that the use of a multi-label classification Naive-Bayes method helps with word sense disambiguation (WSD) and determine word similarities. OpenAI's GPT-2 Transformer (Openai 2019) is intended for sequential word prediction based on unsupervised learning: it provides a pretrained word embedding model that comprises words' meanings by considering the contextual relationships among words (Radford et al. 2019).

Methods

All of the bots follow the baseline algorithm developed for the Codenames AI framework that is shown in Kim et al. (2019). From there, the following word association algorithms are applied to enhance the bots' performances within the game.

Term Frequency - Inverse Document Frequency algorithm

The TF-IDF algorithm determines the importance of a word based on its frequency in a document compared to an entire corpus of documents. For our experiment, our corpuses are Wikipedia summaries retrieved from the Wikipedia Python library and dictionary definitions retrieved from the PyDict library for the words on the board. The bot preprocesses the embeddings for the words by retrieving these summaries, removing stop words, and tokenizing the words within the summaries and definitions. Using a bag-of-words model, it calculates TF-IDF scores for each word in each summary.

The Codemaster bot uses the TF-IDF scores to select a corpus document that contains the highest frequency of red words. The bot selects a word with high term frequency score from this document to use as clue word and the number of red words found within the document as the clue number.

The Guesser bot uses the same TF-IDF scores to do a reverse search within the set of corpuses to find the document containing the highest term frequency of the clue word and selects any board words found; ranking them based on their term frequency times inverse document frequency score.

Naive-Bayes Algorithm

The Naive-Bayes algorithm determines the probability of a word belonging to a particular class using Bayesian inference. It requires a corpus data set for the words, and class sets to categorize them into. Like the TF-IDF algorithm, we used tokenized Wikipedia summaries and dictionary definitions as the corpus set for the board words. For the "classes" representation, we used a list of 118 generic word categories (i.e. "animals", "countries", "food", etc) and retrieved their summaries and definitions to be used as training data.

The Codemaster bot uses Laplace smoothing in order to determine the probabilities of each red word belonging to these category "classes." When adding the probabilities of words in each class, the category with the highest probability is used as the clue word and the number of red words with a probability higher than a preset threshold value - the minimum possible Laplace distance - are used as the guess number.

The Guesser bot uses the same Laplace smoothing algorithm and dataset to determine the probability of each board word belonging to the specified class clue word. It ranks each board word based on their probabilistic Bayesian score calculated.

Transformer Algorithm

The transformer-based bot uses the pretrained GPT-2 word embedding model for the representation of the words on the board. GPT-2 uses stacks of transformer decoder blocks to build a highly contextualized Language Model designed to predict the next word in a sequence. (Radford et al. 2019). For the *Codenames* (Chvatil 2015) setup, the input consists of a set of single words, hence a context based fine tuning is not considered since a pretrained embedding suffices to reflect a general meaning for each word. Also, because a static representation was needed, only the first layer of the embedding was used. In order to minimize the curse of dimensionality, the small OpenAI GPT-2 English model (with a vocabulary size of 50257 and 768 dimensions) was selected (Openai 2019). Cosine Similarity metric was used to calculate distances between the words.

In the Codemaster bot, a subset of red words within a distance threshold is selected and a centroid that represents it is calculated. In an initial approach, the set of bad words - words with either a civilian label, an opponent team label, or an Assassin label - was also included in the centroid calculation (using negative weights, varying with the importance of the risk); the resulting centroid was markedly displaced from the subset neighborhood, resulting in a meaningless word search. Using the centroid and the vocabulary in the pretrained model, K nearest neighbors (KNN) was applied to find a list of recommended words. These words are located in the subset vicinity, offering a spatial, representation and meaning closeness. For the weighted version of the bot, this recommended list is sorted, using the relationship between the similarity with the red words and the similarity with the bad words to minimize the associated risk.

The Guesser bot uses KNN to search for K (number provided by the Codemaster) words in the board that are closer to the clue based on the pretrained GPT-2 embedding.

Bad Word Weighting

To extend upon each of these base algorithms, we implemented a weighting system for each of the words found on the board. Bad words are weighted based on their implicit risk factor: -1 for civilians, -2 for blue words and -3 for the assassin. These weights are applied by the Codemaster bots to avoid selecting these words when looking for a clue word.

Experiment Setup

The Codenames AI Competition Framework (Summerville et al.) is a single-team version of the game, where a Codemaster and a Guesser play together to find the red team's words in the minimum number of turns.

To assess the performance of the bots, a set of round-robin tournaments was used, matching pairs of Codemaster and Guessers bots and playing 30 games for each pair, with a fixed set of boards.

In a first tournament the unweighted version of our three Codemasters were used. A second contest was run using the weighted for bad words versions.

The team wins the game when it flips all red cards before flipping either the assassin or all the blue cards. Each time that a bad word is flipped, the red team losses the turn and the turn count in increased by one. The metrics used in the evaluation are the number of turns to win (Turns), both minimum and average -calculated for wining cases; the percentage of games won per each bot (Win Rate); and the number of good (red) and bad (either blue, civilian or assassin) words flipped during each game.

In a second stage of the experiment, the GPT-2 Transformer based bots were matched with the best performing bot found by Kim et al. (2019) built by concatenating word2vec and GloVe embeddings. The same metrics were observed and compared to evaluate the agents.

To understand how human intuitive are the clues and guessing of the Transformer and concatenated w2v+GloVe (w2v+GloVe) bots, a study with 10 human players was performed. Each participant played a total of 4 games, one with each model as both Codemaster and Guesser, for a total of 40 games.

Results

Table 1: Codenames Bots

Label	Bot
T	GPT-2 Transformer
TF	TF-IDF
NB	Naive-Bayes
WG	word2vec+GloVe concatenation
H	Human player

For the graphs shown in this section, the label scheme described in Table 1 is used. For each pair of graph labels, the first listed algorithm represents the Codemaster used and the second listed algorithm represents the Guesser used. For example, the label "TF-T" represents a TF-IDF Codemaster paired with a GPT-2 Transformer Guesser.

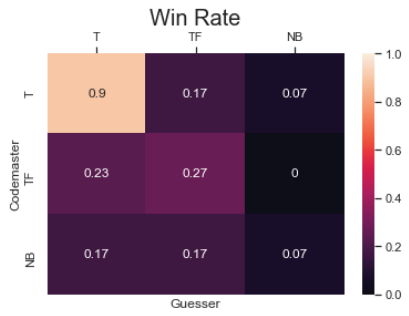
A link to the code repository for this project can be found in: https://github.com/MasterMilkX/codenames_autobots.

Win Rate

Figure 2 shows the win rates for each bot pair - with both the unweighted Codemasters examining only the good words and the enhanced Codemasters applying weights to the bad words. The Transformer self-pair had the highest win rate

percentage in both cases with 0.63 for the unweighted and 0.9 for the weighted bots. When considering mixed pairs, the TF-IDF Codemaster paired with the Transformer Guesser reached the best rate, with 0.33 for the unweighted and 0.23 for the weighted bots. Naive-Bayes got the poorest results both self paired and in mixed pairs.

(a) Unweighted Codemasters



(b) Weighted

Figure 2: Win Rate comparison for a good word exclusive Codemaster and a weighted bad word Codemaster

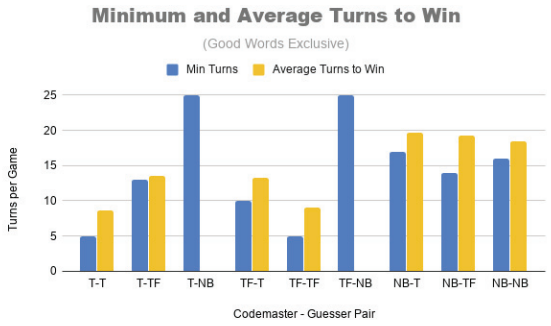
Average vs. Minimum

Figure 3 shows the minimum number of turns taken and the average number of turns taken to win for each bot pair. The Transformer self-pair had the smallest average number of turns at 8.58 turns with unweighted Codemasters and 7.96 turns with the weighted Codemasters. The TF-IDF self-pair tied with the Transformer self-pair for the minimum number of turns taken at 5 turns to win using the unweighted Codemasters, and the Transformer self-pair had a minimum of 4 turns with the weighted word Codemasters. Since there are a total of 8 red words per game that must be found, these results show that the bots are able to find more red words using fewer turns and use clue words that associate to multiple red words on the board.

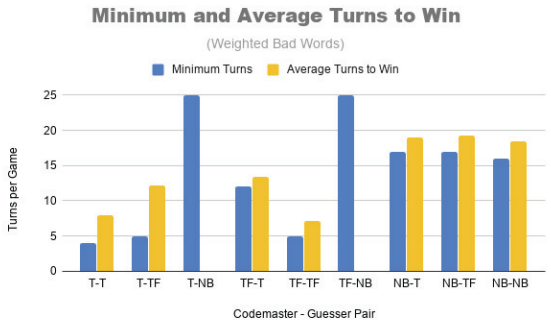
Again the Naive-Bayes bots perform poorly when paired with the other bots - requiring twice as many turns to guess or not being able to win at all.

Board Words Left on Completion

Figure 4 shows the average number of good and bad cards flipped on the board on completion (win or lose) for each game.



(a) Unweighted Codemasters



(b) Weighted Codemasters

Figure 3: Average and Minimum Turns to win comparison for a good word exclusive Codemaster and a weighted bad word Codemaster

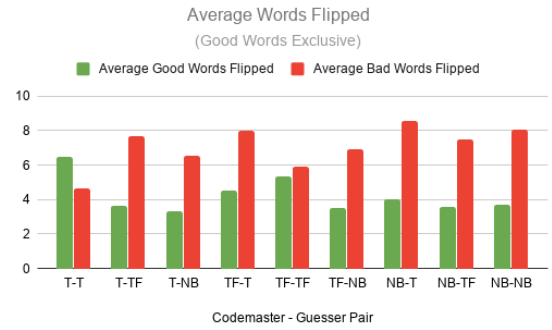
The Transformer self-pair is the only pair to have a higher average of good words flipped than bad words - meaning the Transformer bots had a higher precision; the Codemaster for selecting clue words closely related to the red words and the Guesser for correctly interpreting these clue words to be associated with red words more than bad words.

The TF-IDF self-pair had the closest ratio for average good words to average bad words flipped. Since there are twice as many bad words on the board than good words, it can be noted that this self-pair still performs well at selecting good words over bad words. The Naive-Bayes Codemaster caused the highest averages for selecting bad words than good words - resulting in the least amount of precision for all the pairings and having only half of the good words revealed by the end of the game.

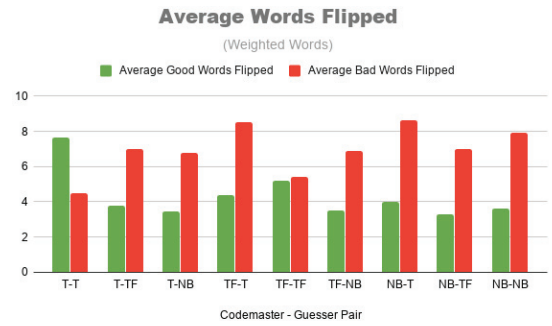
With the weighted Codemasters, bad words that were flipped were reduced for all pairings, meanwhile the number of good words that were flipped remained the same except for the Transformer self-paired bots.

Human Playability

Transformer bot has a better performance as Codemaster when paired with a human Guesser: wining rate is 0.6 vs.0.4; the average number of good words flipped is 6.7 vs. 5.9, while the average number of bad words flipped is 5.9 vs. 6.1; additionally, the average number of turns is 11 vs.



(a) Unweighted Codemaster



(b) Weighted Codemaster

Figure 4: Average Word Flip comparison for a good word exclusive Codemaster and a weighted bad word Codemaster

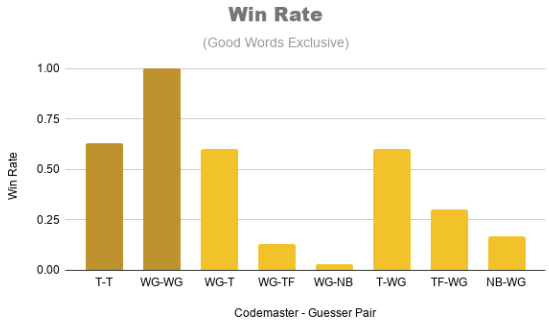
12.25, while the minimum number of turns to win is 6 for both. In the case of human Codemaster, the concatenated w2v+GloVe Guesser bot outperformed the Transformer one: the wining rate is 0.8 vs. 0.3; the average number of good words flipped is 7.1 vs. 6; and the average number of bad words flipped was 4.2 vs. 7.4. Both the minimum number of turns and the average number of turns to win were lower with 7 vs. 11 and 10.75 vs. 12.67 respectively. Qualitative findings from the human playability test included:

- Bots provide a best clue based on the board space and can keep repeating the same clue in cases where the person was not able to guess properly.
- Humans rely on abstract concepts and subtle relationships that non-human agents have a harder time to identify.

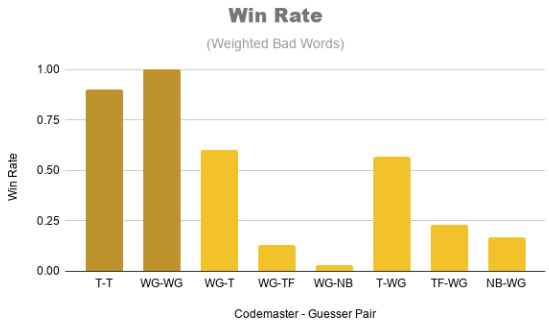
Word2vec+GloVe Comparison

Win Rate Figure 5 shows the win rates for each bot pairing. The self-paired w2v+GloVe has a 100% win rate (Kim et al. 2019). When paired with other bots, however, it reaches a win rate less than that of the Transformer self-paired bot. When the w2v+GloVe concatenation bot is paired with a Transformer bot, it performs equally as well as both a Codemaster and a Guesser with a 60% win rate. When paired with the TF-IDF bots and the Naive-Bayes bots it has 30% and below for a win rate.

However, some of the win rates decrease when the W2V Guesser is paired with Codemasters using weighted bad words compared to the Codemasters not examing the bad words. The Weighted Transformer Codemaster results in a 57% win rate, the Weighted TF-IDF Codemasters results in a 23% win rate, and the Weighted Naive-Bayes Codemaster win rate remains unchanged at 17%.



(a) Unweighted Codemasters



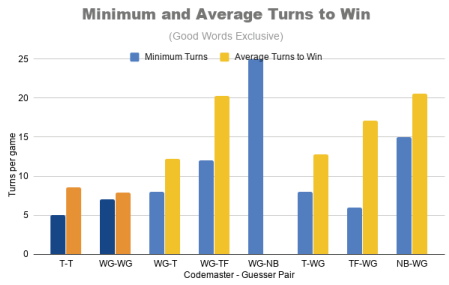
(b) Weighted Codemasters

Figure 5: Win Rate comparison for the w2v+GloVe concatenation bot with Unweighted and Weighted Codemasters

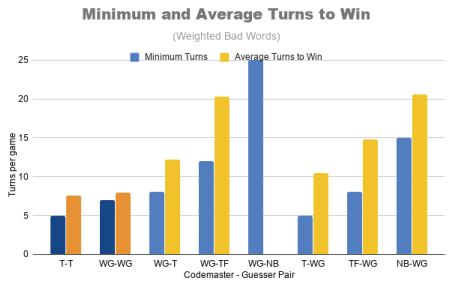
Minimum and Average Turns to Win The unweighted Transformer self-pair bots performed better than the w2v+GloVe concatenated bots by winning the game in fewer turns - see Figure 6 with a lower minimum turn score of 5 vs. 7. When paired with the weighted Codemasters, the average turns taken decreased for both the Transformer and TF-IDF bots. The average number of turns decreased to 7.6 turns for the Transformer Codemaster - less than the w2v+GloVe self-pairing with 7.93 in average.

It can be observed that the weighted Transformer bot was able to give more clues within a single turn and search for the word association and similarities between more red words better than the w2v+GloVe bot self-pairing.

Average Words Flipped Figure 7 reveals that the w2v+GloVe self-pairing has perfect precision in determining good words from bad words, as mentioned by Kim et al. (2019). However, when paired with other bots its precision falls, guessing more bad words on average than the Transformer self-pairing. It guesses the most good words when paired with the Transformer bot, either as Codemaster or

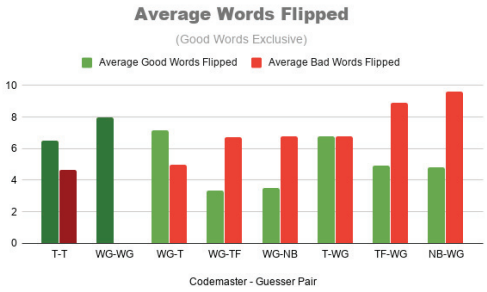


(a) Unweighted Codemasters

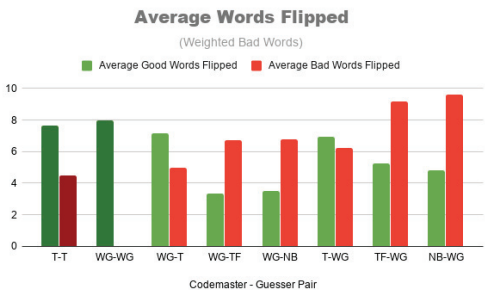


(b) Weighted Codemasters

Figure 6: Minimum and Average Turns comparison for the w2v+GloVe concatenation bot with Unweighted and Weighted Codemasters



(a) Unweighted Codemasters



(b) Weighted Codemasters

Figure 7: Average Words Flipped comparison for the w2v+GloVe concatenation bot with Unweighted and Weighted Codemasters

Guesser. As Guesser, it is able to correctly guess at least half of the red words on average. When paired with the Transformer as a Guesser, it picks just as many red words as bad words.

Load Time

The Transformer algorithm had the shortest average load time (from command run to third clue output), 31.25 s, followed by Naive-Bayes with 59.75 s and TF-IDF with 60.25 s. Although the w2v+GloVe algorithm has the highest win rate and creates the most precise clues and guesses it takes over 5 times as long to load (163.75).

Conclusion and Future Work

In this experiment, we challenged the Codenames framework bots w2v+GloVe with our own bots using alternative algorithms . We used two non-vector based approaches, TF-IDF and Naive-Bayes. Both failed to achieve good results. We also created alternative word embedding search bots using GPT-2 and achieved competitive results in terms of win rate and average turns taken and lower minimum turns taken and load time. Key takeaways include:

- When bots are self-paired (based on the same method), the average performance is better.
- Bots that consider the risk of choosing a bad word in the selection of the clue, provide better results compared with the unweighted version of the bots.
- The weighted transformer pair outperformed the other bots developed by the authors in all the metrics: it triples the wining rate of the next pair, has a better minimum number of turns to win, and is the only one that reached an average number of good words higher than the number of bad words flipped during the game.
- The transformer bot is able to combine several red words in one single clue, reducing the number of turns needed to play the game.

When compared the w2v+GloVe bot, the transformer has:

- Lower average and minimum number of turns to win.
- Lower average good words flipped.
- A 90% wining rate (compared with 100% for the w2v+GloVe bot). But the weighted transformer pair has a considerably lower computational cost.

In conclusion, the transformer bot offers a promising performance, and further work can be done aimed to keep improving it. Some suggested work includes:

- Compare different threshold values for the selection of red words subset used in searching for recommended words.
- Compare results with different weights for the bad words and different threshold for the red words subset selection.

Acknowledgments

Rodrigo Canaan grate-fully acknowledges the financial support from Honda Research Institute Europe (HRI-EU).

References

Bard, N.; Foerster, J. N.; Chandar, S.; Burch, N.; Lanctot, M.; Song, H. F.; Parisotto, E.; Dumoulin, V.; Moitra, S.; Hughes, E.; et al. 2020. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence* 280:103216.

Brown, T. B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chvatil, V. 2015. Codenames. Board Game.

Fellbaum, C. 2012. Wordnet. *The encyclopedia of applied linguistics*.

Gaina, R. D.; Pérez-Liébana, D.; and Lucas, S. M. 2016. General video game for 2 players: Framework and competition. In *2016 8th Computer Science and Electronic Engineering (CEEC)*, 186–191. IEEE.

Jurafsky, D., and Martin, J. H. 2014. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Dorling Kindersley Pvt, Ltd.

Kim, A.; Ruzmaykin, M.; Truong, A.; and Summerville, A. 2019. Cooperation and codenames: Understanding natural language processing via codenames. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, 160–166.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Openai. 2019. openai/gpt-2-output-dataset.

Pennington, J.; Socher, R.; and Manning, C. D. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1(8):9.

Resnick, C.; Eldridge, W.; Ha, D.; Britz, D.; Foerster, J.; Togelius, J.; Cho, K.; and Bruna, J. 2018. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*.

Summerville, A.; Kim, A.; Ruzmaykin, M.; and Truong, A. The codenames ai competition. https://sites.google.com/view/the-codenames-ai-competition.

Urbanek, J.; Fan, A.; Karamcheti, S.; Jain, S.; Humeau, S.; Dinan, E.; Rocktaschel, T.; Kiela, D.; Szlam, A.; and Weston, J. 2019. Learning to speak and act in a fantasy text adventure game. *arXiv preprint arXiv:1903.03094*.

Walton-Rivers, J.; Williams, P. R.; and Bartle, R. 2019. The 2018 hanabi competition. In *2019 IEEE Conference on Games (CoG)*. IEEE.

Walton, N. 2019. Ai dungeon. = https://aidungeon.io/.

Wu, H. C.; Luk, R. W. P.; Wong, K. F.; and Kwok, K. L. 2008. Interpreting tf-idf term weights as making relevance decisions. *ACM Transactions on Information Systems* 26(3):1–37.

Yannakakis, G. N., and Togelius, J. 2018. *Artificial intelligence and games*. Springer.