Enhancing Codenames Gameplay with Word Embeddings and Semantic Relatedness

Rehan Mullan
Tulane University of Louisiana / New Orleans, LA
rmullan@tulane.edu

Abstract

Codenames is a popular word-based board game that challenges players to make associations between words and uncover their team's secret cards only known by the spymaster. In this project, we present an innovative approach to enhancing the gameplay experience by leveraging word embeddings and semantic relatedness techniques from natural language processing (NLP).

Our spymaster creation utilizes the powerful word2vec model, which represents words as dense vector representations in a high-dimensional space, capturing their semantic and syntactic relationships. By leveraging these word embeddings, we can compute the similarity between words and identify meaningful associations that can aid players in decoding clues more effectively.

Additionally, we incorporate WordNet, a lexical database from the NLTK corpus library, to exploit the semantic relationships between words. WordNet organizes words into sets of synonyms (synsets) and provides hierarchical relationships, such as hypernyms (broader concepts) and hyponyms (narrower concepts). By traversing these semantic connections, our spymaster can uncover deeper conceptual links between words, enhancing the potential for more sophisticated and challenging clues.

Through the combination of word embeddings and WordNet, our spymaster creation aims to elevate the strategic depth of Codenames by providing players with more nuanced and thought-provoking clues. This approach not only enhances the overall gameplay experience but also introduces an intriguing intersection between natural language processing techniques and recreational word-based games.

1 Introduction

Codenames the popular word game consists of 2 teams, The red team and the blue team. Each team consists of a spy master elected by the team members. The job of the spymaster is to provide their team members with clues that denote which of the cards on the board is theirs. These clues can not consist of any of the words already present on the board. Valid clues could be synonyms, antonyms, or adjectives. The board is separated into 8 cards for the red team as they always go first, 7 for the blue team, 9 neutral cards and one Assassin card. Guessing the Assassin card is a direct game over for the team that guessed it. Guessing a neutral card ends your turn and guessing a correct card gives you an extra card. The goal of the game is to beat the opposing team by guessing all your team's cards before they do.

Stemming from my interest in automating word games, I had the idea of automating the spymaster for this game. The idea was to have an agent using one of these pre trained methods: transformers, word2vec, conceptnet, GloVe or Wordnet. All of these methods were tested and two of them were used, namely a pretrained Word2Vec model and WordNet using the Natural Language ToolKit (NLTK) corpus. The pre-trained Word2Vec model that was used is available through the gensim library. This library contains a lot of easy to use Word2Vec models trained on many different datasets. The WordNet dataset that was used is freely available from the Natural Language ToolKit (NLTK). A lexical database from the Natural Language Toolkit (NLTK) corpus, which organizes words into sets of synonyms (synsets) and provides hierarchical relationships, enabling the exploration of broader and narrower conceptual links.

By integrating the strengths of word embeddings and semantic relatedness models, this automated spymaster endeavors to augment the Codenames gameplay experience. Moreover, this project serves as a pioneering exploration into the potential of harnessing word embeddings and semantic relatedness

techniques for enhancing interactive word-based activities. While representing a significant stride towards creating a proficient automated spymaster, this study paves the way for further advancements and refinements.

2 Related Work

Initially going into this project I went into the mindset of wanting to use reinforcement learning to train an agent to play the game better. Upon further research I found out that this was in fact a more popular NLP problem than it was a reinforcement learning one even though reinforcement learning has been used previously (Siu, 2022). While conducting research and looking into different implementations I was able to find a few successful attempts at automating this game using various techniques such as transformer embedding, word2vec models, Naive bayes agents, Term Frequency-Inverse Document Frequency agents. The best of these implementations, done by a group of people from NYU, culminated in this paper named Word autobots (Jaramillo et al., 2020). This interesting paper has a lot of different agents trying to outplay the other. Through rigorous testing they find out using a transformer proves superior to the other implementations. Apart from that paper a lot of articles such as (Neiman, 2021) have been published and push for using Word2Vec models to play this game. As it is an easy out of the box implementation. A lot of other more effective methods use transformers and some such as (Thomasahle) use GloVe.

3 Methods

Initially while starting this project and looking into the successful transformer implementations, I wanted to create something similar as time and time again it has been shown how effective transformers are at performing this task. Upon testing with using the GPT2LMHead model and its tokenizer as well as other models like using sentence transformers and their respective tokenizers from hugging face I was not able to produce satisfactory results. The end weighted product of operations like cosine similarity were producing numbers too low and it was taking too long to figure out if this was because of my limited understanding of this method or just using the wrong models to begin with. From here on I ventured off to find something that could produce better results for my purpose. Here I stumbled onto ConceptNet, A Knowledge graph. ConceptNet is a large-scale knowledge graph that encodes commonsense knowledge using structured relationships between concepts, providing a semantic network for AI systems to understand human-like reasoning and context in natural language understanding tasks. This method seemed to prove very feasible but the challenge lay in understanding the implementation of the rest API. ConceptNet lets you make calls to its graph using JSON REST API. To use this alongside python you have to make use of the requests package and finally extract its information from the metadata you receive back. Initializing this and getting it to work also seemed like a time drain.

After this I finally settled to test out using a pretrained Word2Vec model from Gensim. Gensim is a Python library for topic modeling, document similarity analysis, and natural language processing tasks, providing efficient implementations of algorithms like Word2Vec. It enables users to analyze and extract semantic meaning from large text corpora through intuitive interfaces and scalable implementations. While starting out and using the different pre trained Word2Vec models through the Gensim downloader module I quickly learned that datasets like the conceptnet-numberbatch-17-06-300, glove-wiki-gigaword-100 as well as the word2vec-google-news-300 all of these did not seem to contain words that were commonly present in the codenames wordpool or even give relatively decent clues even if they did. So I eventually settled for downloading the GoogleNews-vectors-negative300 which ended up having a lot of the words and produced decent clues while using the most similar function that under the hood uses cosine similarity.

Upon using this new dataset and testing it on the randomized dictionary representing the board state the results were still not satisfactory. The Word2Vec model frequently gave clues that blatantly contained the string it wanted me to guess. To demonstrate: Word to guess “Dress”, Clue given “Dressing”. To combat this issue my proposed solution was to make use of the synsets given to me from the Natural Language ToolKit. This function is able to extract and give synonyms by extracting it from the semantic meaning of the word. Seeing the result of this on each individual word gave me words that were really good clues for one particular word. To reduce repetition of similar meaning words from the synsets I then extracted the lemmas.

Lemmas are the base or dictionary forms of words, capturing their canonical or root form, often used for normalization in natural language processing tasks. And then dropped words that were either hyphenated, contained two separate words, or contained any special characters. Doing so and creating a set out of this dropped any repetition and gave me words that were substantially better clues than before.

4 Results

After repeating the experiment multiple times. Using these methods of abstraction does somehow prove to give you good clues at times. Since the method of board generation is random each time it is usually hard to have a baseline. Given enough trials you start to see patterns in the clues it gives you. At other times the bot gives you a clue, a very popular one for this instance being the clue "significantly", that even though it has a very high similarity metric it's hard to denote what word this clue is actually trying to get you to pick. Apart from that after multiple runs of the bot you start to see a pattern given the board state. For example whenever the word "Circle" is on the board the top clue is always the word "laps". Whenever the word "Bugle" is on the board the model spits out the clue "melodies". Multiple instances like this and it is easy to spot trends.

5 Discussion

As we can see the model does work better than a standard Word2Vec approach. It does spit out better clues. However sometimes these clues do not make complete sense. Through multiple runs one can start to see patterns in the clues given and given enough runs one can learn some of the clues to board relations. The fact that the board is randomized does help the model at times but at times it also does hurt it. The randomization helps the model if the right type of board is given to the agent. Given the right kinds of randomization the agent will be really helpful giving you some helpful hints. But for the most part the clues are hard to decipher. The times where the randomization hurts to board are when there is almost no relation between the words on the board or if there are too many proper nouns on the board. This model can definitely be worked on more and improved on in many aspects. Starting with finding a better dataset. Given a better dataset this model will be able to produce better clues. Apart from the dataset, if the model used more of WordNet, specifically functions like hypernyms, and maybe even some sort of knowledge graph this model could definitely perform better. Other approaches should also be looked at for future work such as successfully implementing a transformer and using the embeddings from it.

6 Conclusion

This project leveraged word embeddings from a pretrained Word2Vec model and semantic relations from WordNet to develop an automated spymaster agent for the word association game Codenames. The hybrid approach could identify meaningful connections between words to generate relevant clues without directly including the target words. Over multiple trials, patterns emerged where certain clue words consistently mapped to specific board words, though some clues lacked clear associations.

While representing progress in automating the Codenames spymaster, there is room for improvement. Incorporating larger word embedding datasets, exploiting additional semantic relationships from WordNet, and investigating techniques like transformers and knowledge graphs could enhance clue quality and contextual understanding.

Ultimately, this study demonstrated the potential of combining word embeddings and semantic models for creating more engaging interactive word-based experiences. Continued advancement in these NLP techniques holds promise for intellectually stimulating applications within word games and beyond.

7 Division of Labor

This project was conducted by myself with the help of ideas from various articles, githubs and papers some of which are included in the reference section below.

References

Catalina Jaramillo, Melanie Charity, Rodrigo Canaan, and Julian Togelius. 2020. Word autobots: using transformers for word association in the game codenames. *Proceedings*, 16(1):231–237.

Jeremy Neiman. 2021. How to create a Codenames Bot Part 1: Word2Vec - towards data science.

Sherman Siu. 2022. Towards automating Codenames spymasters with deep reinforcement learning.

Thomasahle. GitHub - thomasahle/codenames: Codenames AI using Word Vectors.