Enhancing Autonomous Spymaster and Guesser Algorithms for Codenames Gameplay

David Nguyen

Bachelors of Science In Computer Science
The University of Bath
2023-2024

Enhancing Autonomous Spymaster and Guesser Algorithms for Codenames Gameplay

Submitted by: David Nguyen

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

This dissertation proposes a novel approach to enhancing autonomous algorithms in Natural Language Processing (NLP) through the analysis of the popular word disambiguation game, Codenames. The study evaluates two key metrics: the average number of turns required to complete a game and the accuracy of identifying intended words based on provided clues.

Introducing Sense2Vec word embeddings as an alternative to conventional embeddings like Word2Vec, the research investigates their impact on Codenames, which have yet to incorporate such techniques.

The findings unveil significant insights. Employing Sense2Vec leads to superior performance compared to Word2Vec across various aspects. Sense2Vec consistently outperforms Word2Vec in word sense disambiguation, hitting fewer assassin words and completing more games, resulting in a substantially higher win percentage of 88.5% compared to Word2Vec's modest 80%.

Moreover, Sense2Vec demonstrates remarkable accuracy in deciphering clues, achieving a success rate of 86.6%, outshining Word2Vec's success rate of 84%. Notably, while Word2Vec excels in the first rounds of deciphering clues, achieving an impressive overall success rate of 90% compared to Sense2Vec's 86.9%, Sense2Vec showcases its ability in its overall performance.

Despite Word2Vec's advantage in completing games more swiftly, Sense2Vec's nuanced understanding of word senses contributes to its overall superiority. This research highlights the effectiveness of Sense2Vec in advancing NLP algorithms and solving the Word Sense Disambiguation problem present in the field NLP.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my gratitude to Prof Nello Cristianini, my supervisor, for his guidance and encouragement throughout my project.

I would also like to express my gratitude to George Clements, my friend, for his support and inspiration throughout my project.

I acknowledge that this work is my own, and I used ChatGPT 3.5 (Open AI, https://chat.openai.com/) to summarise my initial notes and to proofread my final draft only.

Chapter 1

Introduction

1.1 Codenames

Codenames, a word-association game outlined in Edition (2023), involves a team-based structure with participants divided into red and blue teams, each comprising a spymaster and one or more guessers. The shared game board encompasses 25 words, categorized as red, blue, neutral, or assassin. The spymasters possess knowledge of the word tags, unlike the guessers (refer to Fig 1.1). The primary responsibility of spymasters is to generate one-word clues alongside a numerical indicator, which indicates the number of associated words, guiding their team's guessers. The guessers' objective is to identify the associated words based on the provided clues given by their Spymaster. The Spymaster and Guesser on each team take turns to reveal words associated with their team on the board. If the words revealed are tagged opponent or neutral, this will immediately terminate their turn thus starting the opponent's turn. Termination of the game occurs when a team successfully guesses all its words ahead of the opponent (the blue team starts with 9 and the red team starts with 8 hidden words) or when a team mistakenly identifies the assassin word, leading to an immediate victory for the opposing team.

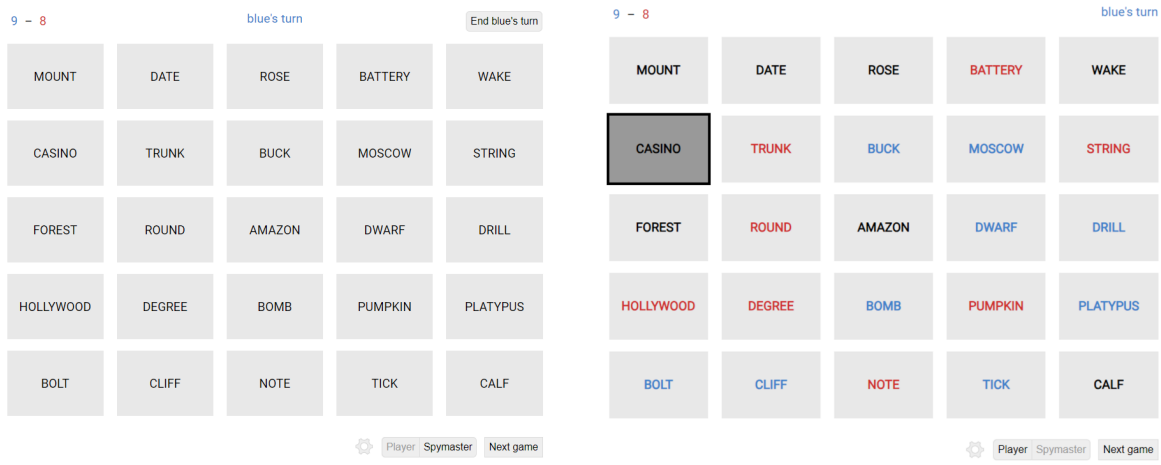

Figure 1.1: An example game board of the Guesser (left) and the Spymaster (right). Screenshot taken from Paste (2016)

The challenges for the Spymaster typically include crafting one-word clues that lead their team to select the correct words while avoiding words related to the opposing team's words or the "assassin" word. Spymasters must navigate the fine line between being too vague or too specific with their clues, all while considering the words in play.

The challenges for the Guessers typically involve deciphering the one-word clue that leads their team to select the correct words while avoiding words related to the opposing team's words or the "assassin" word. Agents must use their ability to think creatively and strategically to decipher the one-word clue.

1.2 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a dynamic field that addresses various ways in which computers analyze and process natural human language. The primary goal of an NLP system is to enable computers to understand, interpret, and generate human language in a way that is both meaningful and useful (Kochmar, 2022a). To achieve this goal, NLP employs a series of linguistic analyses, spanning from Raw Text Processing to Semantic Analysis, to bridge the gap between human communication and computer understanding.

Human language is complex. For instance, idioms present a unique challenge in language learning due to their non-literal meanings, such as the expression "piece of cake" meaning something is easy rather than referring to an actual dessert. This figurative language can complicate language understanding as they must discern the intended meaning beyond the literal interpretation. Conversely, non-stylistic language offers clarity and simplicity, aiming to convey information directly, as seen in sentences like "If your guinea pig is vomiting, contact your vet." In contrast, stylistic language, in sentences like "Who wants to volunteer like a human guinea pig?" employs creativity and literary devices to evoke emotions or convey ideas, making comprehension more challenging.

Explaining certain thoughts or emotions in words poses difficulties, particularly emotive ones like euphoria or terror. Additionally, cultural variability introduces nuances that may not readily translate across languages. For instance, the Japanese term "Tsundoku" meaning the habit of acquiring books and letting them pile up without reading them, reflects a cultural concept that may require contextual understanding for non-Japanese speakers to fully grasp. These complexities underscore the difficulties of language learning.

This creates numerous challenges for an NLP system to understand, generate, and interpret human language. A significant obstacle is Word Sense Disambiguation (Navigli, 2009). The same word or phrase can have multiple meanings, polysemy, making it difficult to understand the meaning of words in sentences. Language is highly context-dependent (Bach, 2014). Equally challenging is language's context dependency. The meaning of a word changes depending on the surrounding words. This makes it tough to grasp the intent of words in sentences.

1.3 NLP in Codenames

Codenames stand out as a challenge and a valuable testing ground for natural language processing systems, recently proposed in the Codenames AI Competition (Summerville, 2019). Its distinctive feature lies in the absence of contextual understanding, demanding a system

that excels in word sense disambiguation. In the game, each clue contains multiple possible senses; for example, the term "HOT" could mean the temperature or spiciness of food. NLP plays a pivotal role in assisting Spymasters in crafting effective one-word clues. Semantic analysis algorithms can be employed to understand the subtle nuances of word meanings and associations. By analyzing the semantic relationships between words, NLP can assist Spymasters in generating clues that are not only contextually relevant but also strategically aligned with the desired word associations on the game board. For the guessers, NLP contributes to deciphering the one-word clues provided by the Spymaster. Contextual understanding algorithms help agents interpret the subtle contextual cues embedded in the clues, making it easier for them to identify the intended associations and avoid potential mistakes related to opposing team words or the assassin word.

1.4 Dissertation Structure

The dissertation starts by exploring relevant literature, providing foundational knowledge essential for conducting this research. Subsequently, the methodology states the construction of the Codenames environment and identifies the methodologies outlined in the literature review to our specific context. These methodologies are then subjected to experimentation, following data collection and analysis. Finally, potential avenues for future work are discussed, resulting in conclusions derived from the findings of this study.

Chapter 2

Literature and Technology Survey

2.1 Semantics and Word Sense Representation

In the domain of Natural Language Processing (NLP), this research focuses on Semantics and Word Sense Representation to tackle playing Codenames. Semantics delves into the meaning/sense of words, while word sense representation tackles the task of encoding word meanings within computational systems. The ability to effectively represent words is crucial for finding the most appropriate sense of a word and allowing computations related to word similarities.

2.1.1 Semantics by Contextual Understanding

Words can have different interpretations depending on their context. According to the hypothesis Kochmar (2022b), words that appear in similar contexts have similar meanings. For example, consider the meaning of the word "mulligatawny". For many, this would be challenging to deduce face on. However, if this appeared within a context such as the following sentence: "The chef's special, mulligatawny, tasted amazing!", it is much easier to deduce that "mulligatawny" is a type of dish. From this hypothesis, two points can be implicated:

- Word meaning can be learned with how often (co-occurrence) these neighboring words appear with the word at hand.
- Similar words should end up with similar representations. The word "mulligatawny" should have similar representations to the word "pizza" as they are both dishes.

Learning the context of a word can help with challenges raised by the sense of a word. Some challenges are:

- **Homonymy**: refers to the phenomenon where two or more words have the same form but different meanings that are unrelated. For example, "Bank" as in financial institution and "bank" as in the side of a river are homonyms
- **Polysemy**: refers to where a single word has multiple different meanings that are somewhat related. For example. "Run" can mean to move swiftly on foot or to operate something. The meanings are related to the concept of progression.
- **Homographs**: refers to words that have the same spelling but different meanings, often

with different pronunciations. For example, the word "Bass" can mean a low-pitched sound or a fish.

In Codenames, it is pivotal for both the Spymaster and the Guessers to carefully contemplate the challenges outlined above when formulating and deciphering clues. The potential for easy misinterpretation of clues underscores the importance of considering the nuanced meanings of words, as a single word may evoke multiple associations, potentially leading Guessers to select the wrong words.

2.1.2 Vector-based Representation

Given that contextual comprehension plays a pivotal role in determining word meaning, we can represent this relationship through vectors. The rationale behind employing vector representations lies in its computational efficiency and data-driven nature. Specifically, one vector representation approach is distributional vectors.

Distributional vectors (Kochmar, 2022b) are semantic representations of words that take the target words, the word at hand to find the meaning of, as rows, and the context words, the neighbouring words of the target words, as columns bounded by some context window. Distributional vectors have dimensions of |V| x |V|, where |V| represents the vocabulary size.

Context windows determine the extent to which surrounding words are considered relevant for establishing word relationships. A wider context window encompasses a broader range of words, potentially resulting in associations between less related words. Conversely, a narrower window limits the scope of surrounding words, potentially overlooking meaningful associations. A context window of size N means that there are a maximum N words on either side of the target word.

```
sentence 1: he used his COMPUTER on his desk
sentence 2: reached for an APPLE from the bowl
sentence 3: The BANANA he packed in his lunch had ripened
sentence 4: Access to reliable INFORMATION is essential
```

Figure 2.1: An illustration of context windows with a length of 3. The words 'computer,' and 'apple,' are surrounded by 6 context words, the 'information' is surrounded by 5 while the word 'banana' is surrounded by 4 context words.

	fruit	data	pie
apple	130	0	40
banana	100	0	20
computer	0	110	0
information	0	150	5

Table 2.1: An example distributional vectors where the rows are target words and the columns are the context words with some context windows. The entries are the co-occurrence counts between the target word and the context word i.e., how often do these two words appear together given a vocabulary size V. For example the pair apple and fruit co-occur, within a context window, 130 out of V times.

The co-occurrence counts within the matrix signify the frequency of each target word appearing alongside each context word within the designated window. An example distributional vector is outlined in Figure 2.1.

2.1.3 Measuring Semantic Similarity

In playing Codenames, understanding the similarity between words is essential for associating clues with the words on the board. This similarity isn't just about synonymy; it also involves semantic connections. Take, for instance, the following pairs of sentences:

```
He is a friendly person
He is a sweet person
```

In these sentences, "friendly" and "sweet" are synonymous; they can be used interchangeably.

However, consider another pair:

```
He is a cat person
He is a dog person
```

Here, "cat" and "dog" aren't synonymous; they can't be interchanged with each other since they have different meanings. However, they share semantic similarities as both refer to types of animals therefore links can be established for words that have different meanings.

Recognizing such semantic relationships is crucial for both the Spymaster and Guesser in grouping and identifying words that share connections that are beyond synonymy. A suitable similarity function that can effectively capture this is cosine similarity (Wikipedia contributors, 2024a). Cosine similarity is the cosine of the angle between vectors. It is a measure of similarity between two non-zero vectors in vector space.:

The cosine similarity between two vectors $\mathbf{A}$ and $\mathbf{B}$ is calculated as follows: (see Equation 2.1):

$$\text{similarity}(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{2.1}$$

where $\mathbf{A} \cdot \mathbf{B}$ represents the dot product of vectors $\mathbf{A}$ and $\mathbf{B}$, and $\|\mathbf{A}\|$ and $\|\mathbf{B}\|$ represent the magnitudes of vectors $\mathbf{A}$ and $\mathbf{B}$, respectively.

Now equipped with the capability to generate word vectors through distributional methods, we can assess their distances within vector space, commonly referred to as semantic space. As referenced in Table 2.1, it is observed that the word vectors for "computer" and "information" exhibit similarity. This is attributed to the frequent co-occurrence of the context word "data" compared to other context words associated with both target words. Consequently, these words are positioned closer to each other in semantic space, indicating their semantic similarity.

Distributional vectors offer a remarkable method for word representation, capable at encapsulating both synonymous and semantically related words. However, as vocabulary sizes expand, these vectors tend to become sparse rapidly, leading to a substantial number of entries close to 0. Consequently, computational complexity escalates, and similarity calculations via cosine may not adequately reflect the similarity between words due to the presence of large sparse vectors. A notable enhancement over distributional vectors is the utilization of word embeddings.

2.2 Word Embeddings

Word embeddings, introduced by Mikolov et al. (2013a) and Mikolov et al. (2013b), represent a form of vector-based encoding, characterized notably by their short length and density. This encoding strategy aims to reduce the computational complexities inherent in distributional vectors. Unlike distributional vectors, word embeddings are concise vectors typically ranging between 50 to 1000 entries, contrasting with the |V| entries in distributional vectors. Moreover, these embeddings are dense, with the majority of entries comprising real values, unlike distributional vectors that often contain numerous zero entries.

The distinctive features of word embeddings render them advantageous for neural network architectures such as Word2Vec. Their compact nature makes it easy to learn for classifiers due to the reduced number of weights and the minimized dimensionality mitigates the risk of overfitting. The condensed representations enable the capture of semantic similarities, notably through metrics like cosine similarity. However, a drawback of word embeddings lies in the potential loss of information stemming from the compact representation of words within dense vectors. Despite this limitation, the conventional Word2Vec methodology remains pivotal within word embedding techniques.

2.2.1 Word2Vec

Mikolov et al. (2013a) and Mikolov et al. (2013b) introduced Word2Vec, a neural network methodology designed to craft short and dense distributional vectors to represent words. At its core, Word2Vec aims to discern the patterns between context words and target words using neural networks, thereby encapsulating the contextual semantics of the language. The paper proposes two distinct architectures: Continuous Bag of Words (CBOW) and Skip Gram, each tailored to capture nuanced aspects of word co-occurrence dynamics.

Continuous Bag of Words (CBOW)

The first architecture of Word2Vec proposed is Continuous Bag of Words (CBOW). CBOW is a neural network approach that learns to predict the target word given its surrounding neighboring words (context words) bounded by some context window.

The Continuous Bag of Words (CBOW) (see Figure 2.2) model initiates by receiving a sequence of input words which are the neighboring context words of a target. These context words are typically encoded as one-hot vectors, where each word within a vocabulary of size V is represented by a vector of length V, with a singular entry denoting its presence and the rest set to zero. Subsequently, the input layer is subject to multiplication by a projection matrix within the hidden layer. This matrix serves the purpose of dimensionality reduction, transforming the high-dimensional one-hot vectors into dense representations capturing the contextual semantics of the surrounding words. The hidden layer aggregates the context words' representations, leading to a singular summation that encapsulates the semantic context provided by the input sequence. Finally, the output layer is subjected to an activation function, commonly softmax, which estimates the probability distribution across the vocabulary. Each word's likelihood is determined based on the accumulated context information, resulting in the prediction of the target word.

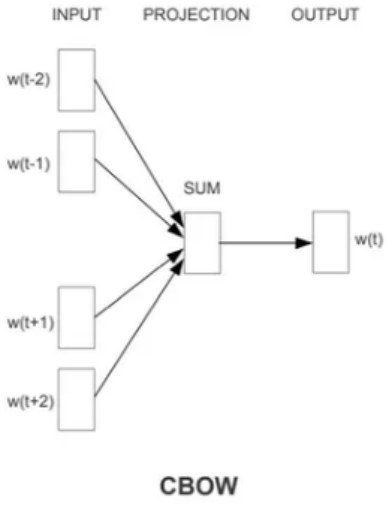

Figure 2.2: Continuous Bag of Words (CBOW) architecture. Image extracted from Kulshrestha (2010)

Skip Gram

The second type of architecture is the Skip Gram model. Skip Grams aims to train a binary classifier that takes the target words as input instead and outputs context words. The aim is to classify how likely a context word appears with a target word. The intention is to learn the classifier's weights which are used as the word embedding. The architecture 2.3 depicts this.

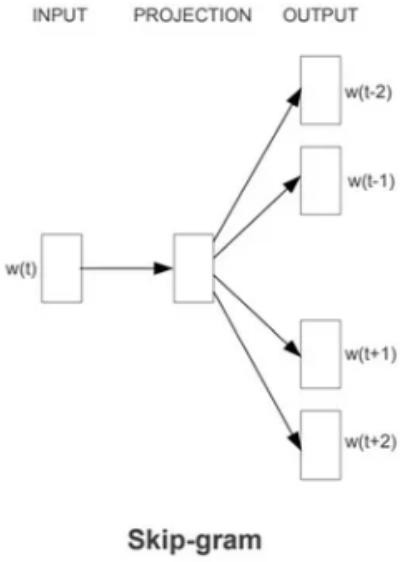

Figure 2.3: Skip Gram Architecture. Image extracted from Kulshrestha (2010)

Given a pair of target and context words, the goal is to return the probability such that the context word is *real* to the target word. By *real* we mean the context word belongs to the target word. For example, suppose the sentence below is present in the training data (the vocabulary used to construct the word embeddings):

```
my [annoying cat ate my homework] yesterday
```

Consider the target word "ate" with a context window of 2. Given the pair ("ate", "yesterday") it is unlikely that these two pairs are *real* since they do not appear in context of the training data (denoted in brackets). We can find the probabilities of the real pairs (also known as positive samples) by Equations 2.2 and 2.3:

$$P(+|w, c) \tag{2.2}$$

and the negative samples (the probabilities of pairs that are not "real") as:

$$P(-|w, c) = 1 - P(+|w, c) \tag{2.3}$$

The classifier estimates the probabilities of positive and negative samples by computing the embedding similarity. The intuition is that context words are likely to occur near a target word if close within semantic space i.e., the similarity. A simple similarity approach is to take the dot product between context words and target words. To generate probabilities, the dot product can be passed through as input to the sigmoid function (Wikipedia contributors, 2024b). The probability for the positive sample is shown in Equation 2.4 and the negative sample is shown in Equation 2.5.

$$P(+|w, c) = (c \cdot w) = \frac{1}{1 + exp(-c \cdot w)} \tag{2.4}$$

$$P(-|w, c) = 1 - P(+|w, c) = (-c \cdot w) = \frac{1}{1 + exp(c \cdot w)} \tag{2.5}$$

However, the probabilities above are only for a single context word. In training, there are many context words. By taking the assumption that all context words are independent, the probabilities can be multiplied as defined below (see Equation 2.6):

$$P(+|w, c1 : L) = \prod_{i=1}^{L} (-ci \cdot w) \tag{2.6}$$

The learning phase of the classifier aims to maximize the embedding similarity of the positive samples whilst simultaneously minimizing the similarities of the negative sample. A cross-entropy loss function can be defined in Equation 2.7 which the classifier aims to minimize.

$$\begin{aligned} \text{LCE} &= -\log\left(P(+|w, c_+)\right) + \sum_{i=1}^{k} \log\left(P(-|w, c_{-i})\right) \\ &= -\log\left(P(+|w, c_+)\right) + \sum_{i=1}^{k} \log\left(1 - P(+|w, c_{-i})\right) \\ &= -\log\left(\sigma(c_+ \cdot w)\right) + \sum_{i=1}^{k} \log\left(\sigma(-c_{-i} \cdot w)\right) \end{aligned} \tag{2.7}$$

The optimization of this function is achieved through stochastic gradient descent. In the Skip-gram model, two distinct embeddings are learned: one representing a word as a target and the other as a context. As a result, the skip-gram model aims to move the positive samples closer to each other in semantic space and further away from negative samples.

2.2.2 Sense2Vec

Sense2Vec, introduced by Trask (2016), represents a notable advancement over Word2Vec, a widely used word embedding technique. A common limitation shared by many word embedding methods is the encoding of words into single vectors, which poses challenges when dealing with words that have multiple senses. Word2Vec, in particular, encounters difficulties with polysemous, homonymous, and homographic words, as it accumulates all potential meanings into a single vector representation. To address this limitation, Sense2Vec offers a solution by introducing multiple vectors for a single word.

Sense2Vec employs a distinct training approach compared to Word2Vec. In Sense2Vec, the training data is labeled with both the word and its corresponding sense. For instance:

```
SENSE2VEC training_data:
"BANK|NOUN"
"BANK|VERB"
"BANK|PRONOUN"
"ANTARTICA|GPE"
"ANTARTICA|LOC"

WORD2VEC training_data:
"BANK"
"ANTARCTICA"
```

A binary classifier is trained using configurations of the Continuous Bag of Words (CBOW) or Skip-Gram models. Similar to Word2Vec, Sense2Vec aims to learn weights that serve as word embeddings. However, the primary divergence lies in the objective of the Skip-Gram models. Instead of predicting target words given context words (CBOW) or vice versa (Skip-Gram), Sense2Vec aims to identify and maximize the most probable word sense given surrounding word senses. This approach fundamentally alters the focus of the training process, enabling Sense2Vec to generate more nuanced and contextually informed word embeddings.

2.3 Previous Work

2.3.1 Word Association Games

Various word association games have been the focus of computational modelling efforts. Previous work have attempted to create algorithms to tackle word association games. Shen et al. (2018) delved into a specific board game centered on three-noun word associations, challenging players to guess two words by selecting from three corresponding adjectives. To tackle this, they investigated traditional methods using word embeddings such as GloVe (Pennington, Socher and Manning, 2014) and Word2Vec, coupled with a similarity function based on cosine similarity. Interestingly, their findings highlighted that modeling word co-occurrence proved most effective in simulating this particular game's dynamics.

Taboo emerges as another captivating word association game, raising the stakes by requiring players to give clues while avoiding specific "forbidden" words. Previous efforts, notably showcased in the Taboo Challenge Competition (Rovatsos, Gromann and Bella, 2018), have concentrated on algorithms that delve into the intricacy of semantic relationships. These approaches prioritize understanding synonyms, antonyms, hypernyms, hyponyms, and meronyms.

Successful algorithms have leveraged distributional semantics, employing vector-based representations to capture the subtle connections between words. Codenames introduce a higher level of complexity compared to Taboo. While similar rules apply, the key distinction lies in Codenames demanding the guesser to guess multiple words instead of single words. This involves the clue giver's intuition to group words effectively, devising clues fitting the largest groups while evading certain critical words known as "assassin" words, which result in instant loss if guessed.

The next section will focus on previous works on Codenames. Many people have successfully implemented algorithms to act as the Spymaster for the game Codenames. These works were able to capture word sense relationships using various techniques such as word embeddings, language graphs, and word co-occurrence. The following sections will also look into any related work to find any further improvements to aid computers in optimizing the word sense disambiguation problem.

2.3.2 Previous Codenames Implementation

Traditional Approach Spymasters

To model human word sense associations in Codenames, it's crucial to lay the groundwork by crafting a vocabulary that captures word meanings comprehensively. This foundation becomes the basis for refining algorithms, allowing for improvements that mimic human-like word associations accurately in games like Codenames.

Groundworks

Jeremy Neiman's study (Neiman, 2017) explores using Gensim's Word2Vec model to craft clues as a spymaster. Word2Vec, a neural network model, converts words into numerical vectors based on their context in text. Despite swiftly generating intuitive clues like "ambulance" suggesting "paramedics," it struggles to capture multiple words effectively. Ambiguity in interpreting clues calls for clearer indications of target words, advocating for clearer word-number pairings. Although improvements are needed, Neiman's work lays the groundwork for an automated clue-generating spymaster using Word2Vec.

Word2Vec, GloVe, WordNet Approach

Kim et al. (2019) was the first in optimal play for Codenames, employing a combination of three traditional word sense techniques: Word2Vec, GloVe, and WordNet (Miller, 1995). While cosine similarity was used for Word2Vec and GloVe, WordNet employed various similarity calculation methods including Path, Leacock, Wu-Palmer, and Resaik. Their experiment involved AI guessers trained on these word sense techniques, measuring game completion by calculating the average number of turns taken. Surprisingly, WordNet, relying on synsets (synonyms) for training, performed notably poorer, achieving a winning rate of 40 percent, contrasting the 80 percent winning rate of the other embedding techniques. Notably, the combined use of Word2Vec and GloVe yielded the best results, averaging 3.3 turns per game. Key takeaways from this study include the superiority of vectorial-based semantics over WordNet, indicating that humans establish word relationships based on contextual understanding rather than mere synonyms. Additionally, the success of Spymasters and Guesser pairs employing word sense techniques suggests that humans associate words more effectively if they've developed similar word meanings i.e., the same vocabulary. Furthermore, the study emphasizes the impact

of similarity functions, indicating the significance of how humans relate words in enhancing association performance. These insights are crucial in refining models for word association.

Further Algorithmic Improvements

Traditional approaches perform well however further improvements have been made to optimize computer word association. Below focuses on a range of other techniques to address creating an AI Spymaster. Jaramillo et al. (2020) focussed on using GPT-2 and achieved competitive results in terms of win rate and average turns taken. The research experimented with the codenames framework Word2Vec + GloVe spymaster and their implementation using GPT-2 pretrained word embedding model. GPT-2 uses stacks of transformers to predict the next word in a sequence given the context of preceding words. The transformers were weighted according to words on the board i.e., enemy and assassin words were higher weighted. From the study, a few things can be noted:

- Transformers perform better than the framework Word2Vec + GloVe. Performed on the same metric, Transformer to Transformer interaction averaged 3.2 turns per game.
- Introduction of weights of bad words significantly improved performance such as win rate and ability to avoid bad words This suggests that adding weights to assassin/opponent words improves performance and that this study reinforces that human learns better through context compared to synonyms.

Intuitively, when humans play Codenames, they opt to use a certain strategy. Our brains organize information such as ideas, feelings, and words by grouping them (Jung, 1910). Friedman and Panigrahi (2021) experiments with grouping words using a nearest-neighbors algorithm to generate clues to optimize the number of intended words. By experimenting with the size of clusters, the average number of turns to end the game changes with it. A few things can be noted:

- Increasing the number of clusters reduces the number of words hit with the clue word. This is understandable since more clusters separate words from others.
- Clusters produces far better results where it only averages two turns to end game

We can infer that the nearest neighbour improves the average number of turns per game. However, there is a lack of research on other clue-generation algorithms. Most Codenames implementation uses nearest neighbours to determine the number of intended words the clue gives. Their approach focuses more on finding the best word sense relation to associate words and then assigning groups based on the word's semantic meaning.

Other word embeddings

One of the central challenges in decoding words in games like Codenames arises from polysemy. Where a single word holds multiple meanings. Conventional word embedding methods like Word2Vec or GloVe represent words using a single vector, overlooking these diverse meanings within different contexts. Sense embeddings, as highlighted in Li and Jurafsky (2015), address this limitation by aiming to capture the diverse interpretations of words.

Instead of relying on a singular representation, sense embeddings assign multiple vectors to a word, each corresponding to a distinct meaning or sense. This could significantly enhance the existing Codenames Framework based on Word2Vec + GloVe implementation. Traditional

approaches like Word2Vec have limitations in capturing diverse contextual information due to their use of fixed context window sizes for all words during training. Polysemous or contextually versatile words might require larger or smaller context windows to capture their nuances effectively. Moreover, Word2Vec treats all words equally in terms of context window size, potentially overlooking the varying contextual dependencies of different word senses.

In contrast, AdaGram (Bartunov et al., 2016) represents an adaptation of Word2Vec embeddings that addresses these limitations. It diverges from the fixed context window approach by introducing dynamic context windows. AdaGram assesses the relevance of different contexts in defining word senses and assigns higher weights to contexts that contribute more significantly to understanding word senses. This adaptability aims to enhance word representations by capturing a more comprehensive range of contextual information, potentially improving the differentiation of diverse word senses within a corpus.

Kim presented a robust framework utilizing Word2Vec + GloVe, showcasing superior performance that exceeded human-level play, concluding with an average of 3.3 turns. Building upon this, Jaramillo introduced transformer models, which marginally improved the testing metric to 3.2 turns, demonstrating a notable leap but not significantly surpassing the traditional framework.

Final remarks

To optimize Codenames Gameplay further, exploring advancements in Word2Vec could be pivotal. Techniques like Sense Embeddings offer promising updates to the traditional approach. The next chapter will look at creating Codenames from scratch implementing both a Spymaster and Guesser that are autonomous using both the Word2Vec and the Sense2Vec embedding.

Chapter 3

Methodology

3.1 Codenames Game Board

The Codenames environment consists of a 25-word board, 2 spymasters, and at least one guesser on each team. The 25 words are randomly sampled from a list of 400 words (sagelga, 2021) used in Codenames. The Spymaster and Guesser then interact with the board, however they interact in different ways.

pants	millionaire	tooth	dress	gold
shark	alien	superhero	revolution	greece
well	glove	plate	square	horseshoe
berlin	time	sub	lead	game
cast	pirate	shot	track	pipe

Figure 3.1: A example game board of Codenames, created by author.

The Spymasters have access to the tag of these words i.e., whether the words are red, blue, neutral, or assassin. The tagged words are as follows:

```
tagged_words = {
    red: ['shadow', 'pit', 'water', 'bark', 'crown', 'bed', 'loch', 'line']
    blue: ['boot', 'penguin', 'stadium', 'foot', 'scientist', 'bond', 'tie',
    'crash', 'honey']
    neutral: ['tokyo', 'tail', 'laser', 'lemon', 'dress', 'tap', 'ice']
    assassin: ['crane']
}
```

The Guessers are not allowed to have access to the tagged words as their role involves identifying the tags. Therefore, the Guesser's representation of the board is as follows:

```
board_words = ['dress', 'stadium', 'line', 'honey', 'tie', 'lemon', 'boot',
'bark', 'Tokyo', 'bed', 'crash', 'ice', 'penguin', 'tap', 'crane', 'water',
'scientist', 'shadow', 'laser', 'loch', 'foot', 'crown', 'pit', 'bond', 'tail']
```

3.2 Spymaster Clue Generation

Now that the board representations have been initialized, the subsequent task involves formulating an algorithm to produce clues. Specific criteria govern what qualifies as an acceptable clue:

- It must consist of a single word
- The chosen word should not include the base or extended forms of any words already present on the board. For instance, if the word "shadow" is on the board, the clue "shadowing" would not be permissible, and vice versa.
- Each clue must differ from all previously given clues.
- The clue must be given with an intended number ranging from 1 to 3 which states how many words on the board are intended for the clue.

During the clue generation process, the Spymaster relies on word embeddings, such as Word2Vec or Sense2Vec, to acquire vector representations of words. This method helps transform semantic meanings into numerical vectors, aiding in the comprehension and analysis of word relationships. Training word embeddings from scratch can be demanding in terms of computational resources and time, particularly when dealing with large text corpora. By implementing pre-trained word vectors, the Spymaster mitigates these challenges, thereby optimizing efficiency. Consequently, utilizing pre-trained vectors, such as those derived from a subset of the Google News dataset (Google, 2013), is advantageous. This dataset comprises 100 billion words and generates 300-dimensional vectors for 3 million words and phrases. Training on such a large corpus enables the capture of semantic variability, polysemy, and rare words, enhancing the quality and robustness of the generated clues. Once an embedding model is established and words are represented as numerical vectors, it becomes trivial to calculate the similarity between words (via cosine similarity. See Equation 2.1).

Parameters	Value Used
Embedding Model	Word2Vec/Sense2Vec
Vocab Size	1500000
Cosine similarity difference	0.35
Top N words	2500

Table 3.1: Parameters used and value used to build the Codenames environment.

A straightforward algorithm for optimizing clue selection can be devised. Initially, the algorithm assesses the current state of the game board, identifying two distinct categories of words: "ally" words, representing words aligned with the team's words, and "bad" words, encompassing terms to be avoided, including words associated with the opposing team, neutral words, and the assassin word.

Subsequently, the algorithm proceeds to iterate through a predetermined vocabulary, delimited by a defined vocabulary size, which comprises the pre-trained vectors from the embedding model. Prior to this iteration, the vocabulary underwent preprocessing steps, including the removal of hyphenated terms (e.g., "mother-in-law," present in the embedding model), terms containing numerical characters (e.g., "2x2," also included in the embedding model), and terms containing underscores (e.g., "user_name," present in the embedding model). Once the vocabulary is cleansed, the iterative process of identifying the optimal clue can begin.

For each word in the vocabulary, it calculates a similarity score with respect to the good words, while considering the negative influence of the bad words. This heuristic-based methodology is outlined in Algorithm 1 (see figure 3.2) which determines the most advantageous clue.

Algorithm 1 Spymaster Clue Generator Algorithm

```
1: procedure SPYMASTERCLUEGENERATOR(V, A, B)        ▷ Input: vocab V, ally_words A, bad_words B
2:     best_clue ← None
3:     best_score ← −∞
4:     for v in V do
5:         sim_to_ally ← 0
6:         sim_to_bad ← 0
7:         for a in A do
8:             sim_to_ally += sim(v, a)             ▷ Calculates cosine similarity
9:         end for
10:        for b in B do
11:            sim_to_bad += sim(v, b)
12:        end for
13:        sim_score ← sim_to_ally - sim_to_bad
14:        if sim_score > best_score then
15:            best_clue ← v
16:            best_score ← sim_score
17:        end if
18:    end for
19:    intended_number ← CalculateIntendedNumber(A, best_clue) ▷ intended_number is generated from another procedure
20:    return best_clue, intended_number
21: end procedure
```

Figure 3.2: Spymaster Clue Generator Algorithm, created by author.

As outlined in Algorithm 1 (see Figure 3.2), determining the intended number of a clue involves another procedure called CalculateIntendedNumber. By considering the best clue and the current state of the game board, this can be deduced by examining the difference in similarity among ally words. The cosine similarity difference, as depicted in Table 3.1, serves to quantify this aspect. This parameter indicates the level of rigidness the Spymaster should apply when grouping very similar words. Algorithm 2 (see Figure 3.3) outlines this procedure. A lower difference implies that the Spymaster will adhere to a stricter criterion, selecting a smaller margin of ally words that closely align with the best clue. Conversely, a higher difference suggests a more lenient approach, allowing for a broader margin and greater flexibility in grouping ally words.

A greater cosine similarity difference implies a riskier strategy, as it widens the gap between words to be avoided and those that may be guessed. While this approach increases the likelihood of quickly uncovering team words, it also heightens the risk of hitting undesirable ones. For instance, consider the cosine similarities provided below (see Figure 3.4):

Algorithm 2 CalculateIntendedNumber(A, C)

```
1: procedure CalculateIntendedNumber(A, C) ▷ Input: Sorted similarity Ally words A, clue C
2:     cosine_sim_difference ← 0.35
3:     max_intended_number ← 3          ▷ Maximum number of intended words allowed
4:     intended_number ← 1
5:     if len(A) == 1 then
6:         return intended_number
7:     else
8:         for i ← 0 to (len(A[: max_intended_number]) − 1) do
9:             if (A[i] − A[i + 1]) < cosine_sim_difference then
10:                intended_number ← intended_number + 1
11:            else
12:                break
13:            end if
14:        end for
15:    end if
16:    return intended_number
17: end procedure
```

Figure 3.3: An Algorithm which calculates the best intended number of words given the current best clue and the current state of the game board, created by author.

```
Cosine Similarity Difference: 0.05
[0.7, 0.65, 0.6] -> Intended number: 3
[0.7, 0.6, 0.5] -> Intended number: 1

Cosine Similarity Difference: 0.1
[0.7, 0.6, 0.5] -> Intended number: 3
[0.7, 0.6, 0.5] -> Intended number: 3
```

Figure 3.4: Example of Cosine Similarity Differences. By calculating the subsequent similarities of the next word similarity, if it is less than the cosine similarity difference it will not count as an intended word

Although the difference in cosine similarity between these scenarios is minimal, the resulting number of intended words varies significantly. Therefore, the Spymaster's decision regarding the cosine similarity difference parameter can significantly impact the dynamics of the game.

With the Spymaster now equipped to generate clues for the guessers, the concern shifts towards developing an algorithm tailored for the guesser. This algorithm aims to suggest clues to guessers by analyzing the Spymaster's clue, the intended number of words associated with it, and the current state of the game board.

Algorithm 3 Get Most Similar Words to Last Clue

```
procedure SUGGESTGUESS(B, C, I) ▷ Input: Current state of board B, Last clue C, Last Clue intended number I
    word_score ← {}
    for word in B do                                  ▷ Iterate through each word on board
        sim_score ← sim(C, word)                      ▷ Calculate similarity between clue and number
        word_score[word] ← sim_score
    end for
    sorted_word_score ← sorted(word_score)[: I]       ▷ Retrieve the first I similarity scores
    return argmax most_similar_words
end procedure
```

Figure 3.5: The algorithm for the guesser. Retrieves the most similar words on the board given the last clue, created by author.

3.3 Suggesting Guesses for Guessers

To decipher the intended words on the board given the last clue and the intended number, an algorithm can be devised. In the game setting, the Spymaster crafts clues that closely align with the ally words on the board. Consequently, an intuitive strategy for the guesser is to identify words that bear the highest resemblance to the provided clues. This can be achieved through iteratively evaluating the similarity between each word on the board and the given clue.

The algorithm, designed to address this task, iterates through the words on the board and calculates their similarity to the last clue. This computation serves to quantify the extent to which each word aligns with the semantic context implied by the clue. Subsequently, the algorithm identifies and returns the words with the highest similarity scores, therefore offering potential suggestions for the guesser. By leveraging this algorithm, the guesser can make informed decisions based on the semantic similarity between the clues provided by the Spymaster and the words presented on the board. This approach is outlined in Algorithm 3 (see figure 3.5).

3.4 Vocabulary Improvements

In Chapter 3.2, we previously discussed the Word2Vec word embeddings, which were pre-trained using the Google News Dataset—a corpus containing 100 billion words. This model generated 3 million words and phrases. For Sense2Vec, we opted for a different corpus: the Reddit Comments Corpus (AI, 2016). Despite the difference in training corpora, both models were adjusted to produce 1.5 million embeddings to enhance computational efficiency.

Nevertheless, even with this adjustment, computational efficiency remains a concern, particularly when generating clues. The Spymaster, tasked with matching the words on the board (|B|) with the vocabulary (|V|), faces a large task. This process entails $|V|^{|B|}$ comparisons, which becomes increasingly inefficient as the size of the vocabulary grows.

Fortunately, both models offer a solution by computing the |N| most similar words for each

word on the board. This parameter, referred to as Top N in Table 3.1, allows us to filter out irrelevant words from the vocabulary, focusing only on the most semantically similar ones. While this still involves $N^{|B|}$ comparisons, it significantly reduces the computational burden and improves the overall efficiency of the Spymaster's Clue Generation.

Chapter 4

Experiments, Results, and Discussion

With the framework established, the subsequent step involves designing experiments to extract meaningful results between the Word2Vec and Sense2Vec embedding models. Of particular interest are two key metrics:

- The average number of turns required to terminate a game
- The accuracy of correctly identifying intended words

By computing the average number of turns, we can assess the effectiveness of the Spymaster-Guesser interaction in efficiently terminating a game, to achieve results close to those reported by Kim (Kim et al., 2019). However, it is worth noting that even if the guessed words do not match the intended clues, it is still possible to correctly identify the team's word. Thus, to address the challenge of word sense disambiguation effectively, it is crucial to evaluate both the intended words for a clue and the guessed words associated with it. Additionally, a third metric is introduced which is the accuracy of correctly identifying intended words for a **single round**. By looking at a single round of words, the difficulty of disambiguating words increases since the Guesser will have a pool of 25 words to begin with. Further results are collected to further question whether Sense2Vec is better at quickly completing a game of Codenames and solving the Word Sense disambiguation problem.

4.1 Experimental Methodology

To collect the specified metrics, three distinct experiments were undertaken. A total of 540 Codenames games were played, with 270 games played with the Word2Vec model and 270 games with the Sense2Vec model. The utilization of three different random seeds (50, 100, 150) was implemented to enhance the breadth of captured data. Each experiment comprised of all three seeds accompanied by the embedding model Word2Vec (W2V) and Sense2Vec (S2V) i.e., Seed 50 - W2V/S2V, Seed 100 - W2V/S2V, Seed 150 - W2V/S2V.

For each experiment, the following data points were recorded:

```
Experiment 1:
- Average number of turns required
- Minimum number of turns taken
- Number of Assassin games
```

```
Experiment 2:
- Total number of clues given
- Number of correct guesses made
- Number of Assassin games

Experiment 3:
- Total number of clues given (Single Round)
- Number of correct guesses made (Single Round)
- Number of Assassin games
```

The assassin games are instances where the Guesser selects the assassin word resulting in immediate termination of the game and a loss. The same parameters were used for every game in these experiments, outlined in Table 3.1, with only the embedding models Word2Vec and Sense2Vec to observe.

4.2 Results

4.2.1 Experiment 1 Results

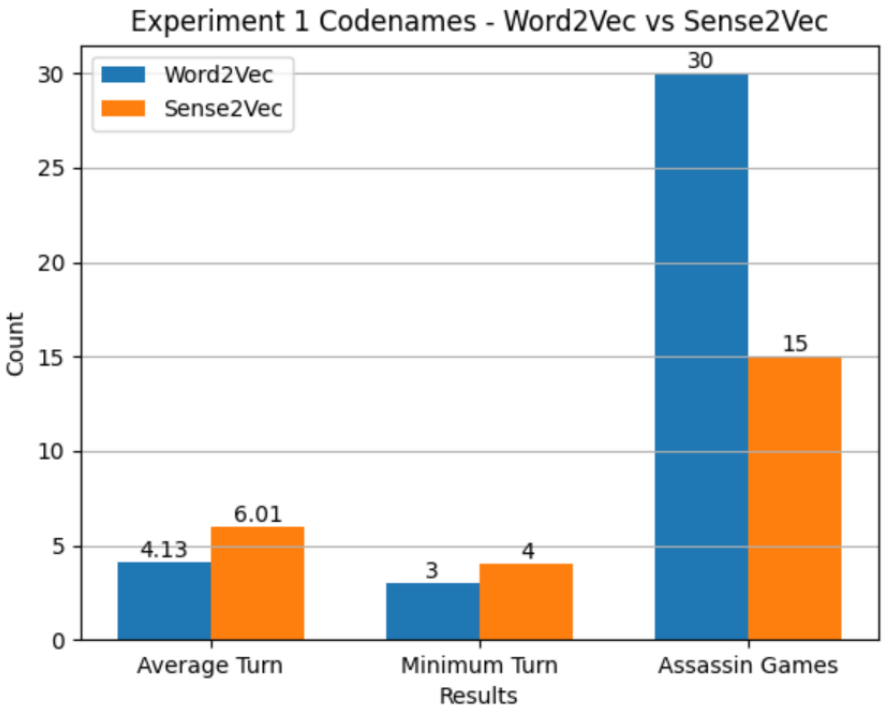

Figure 4.1: Results collected of each seed and embedding model combined visualized using a bar chart for Experiment 1. All the results were collected from each seed and the average was calculated for the average turns to complete the game. The minimum of turns and assassin games were also observed

From the analysis of Experiment 1, it becomes evident that the Word2Vec model adopts a more aggressive strategy compared to Sense2Vec in Codenames gameplay (outlined in Figure 4.1). Word2Vec demonstrates a notable pace, requiring an average of 4.13 turns to conclude a game, whereas Sense2Vec exhibits a more conscious approach, averaging 6.01 turns per game.

Notably, Word2Vec impressively achieves a minimum game termination in just 3 turns. This is the optimal minimum number of games due to the fact that a game starts with 8 or 9 tagged words with only a maximum intended number of 3 words per turn.

Conversely, Sense2Vec, while exhibiting a slower pace, demonstrates superiority in avoiding the revelation of the assassin's word. Over the course of 270 games, Sense2Vec only exposes 15 assassin words, highlighting its adeptness at minimizing risky clues. In contrast, Word2Vec inadvertently reveals the assassin's word twice as much in 30 instances across the same number of games. This aspect holds importance in Codenames gameplay, as the identification of the assassin's word immediately terminates the game. Hence, Sense2Vec's ability to avoid the assassin word underscores its strategic advantage in this regard.

Overall, while Word2Vec showcases a tendency for fast gameplay, Sense2Vec showcases a tendency to avoid the assassin word. This approach signifies Sense2Vec effectiveness in navigating the complexities of Codenames gameplay.

4.2.2 Experiment 2 Results

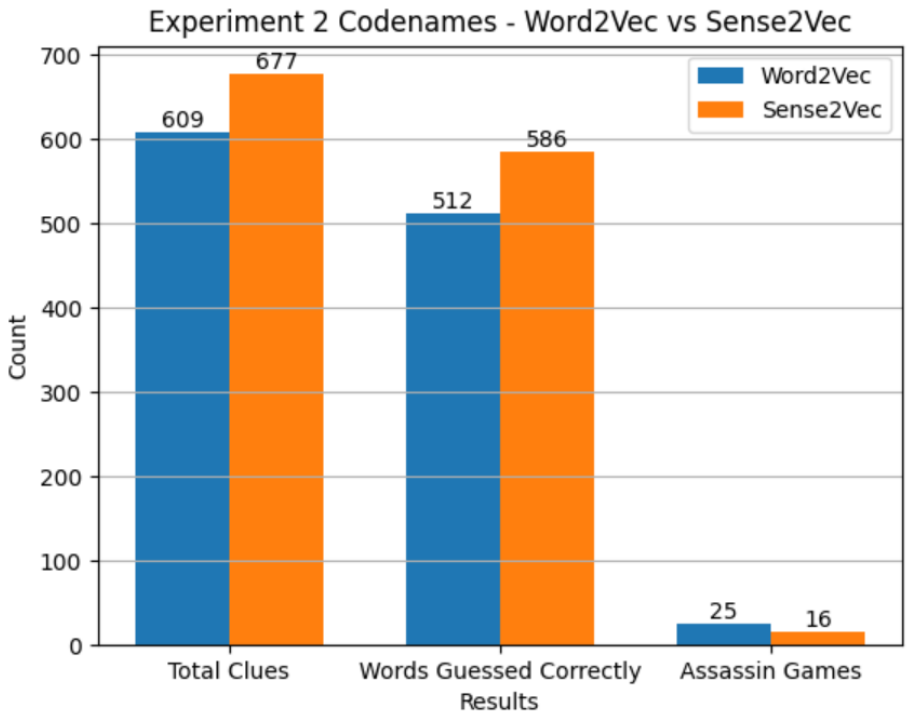

Figure 4.2: Results collected of each seed and embedding model combined visualized using a bar chart for Experiment 2. All the results were collected from each seed and the total number of clues and words guessed correctly were observed. The assassin games were also observed.

In Experiment 2 (see Figure 4.2), it was evident that the Sense2Vec model excelled over Word2Vec in both delivering and deciphering clues effectively. Over the span of 270 games, the Sense2Vec Spymaster provided a total of 677 clues, boasting an impressive 86.6% success rate, with 586 of these clues being correctly guessed. In comparison, the Word2Vec Spymaster provided fewer clues, totaling 609, with a slightly lower success rate of 84%, as 512 of these clues were accurately guessed. Notably, Sense2Vec exhibited an average of 2.5 intended

words per clue, while Word2Vec averaged 2.25 intended words per clue, with the latter being influenced by a higher occurrence of assassin games.

Furthermore, the frequency of assassin games was re-evaluated, aligning with the methodology employed in Experiment 1. The analysis revealed a consistent trend wherein Word2Vec consistently exposed the assassin word more frequently than Sense2Vec. Specifically, Word2Vec disclosed the assassin word 25 times, whereas Sense2Vec revealed it 16 times, across 270 games for each model.

4.2.3 Experiment 3 Results

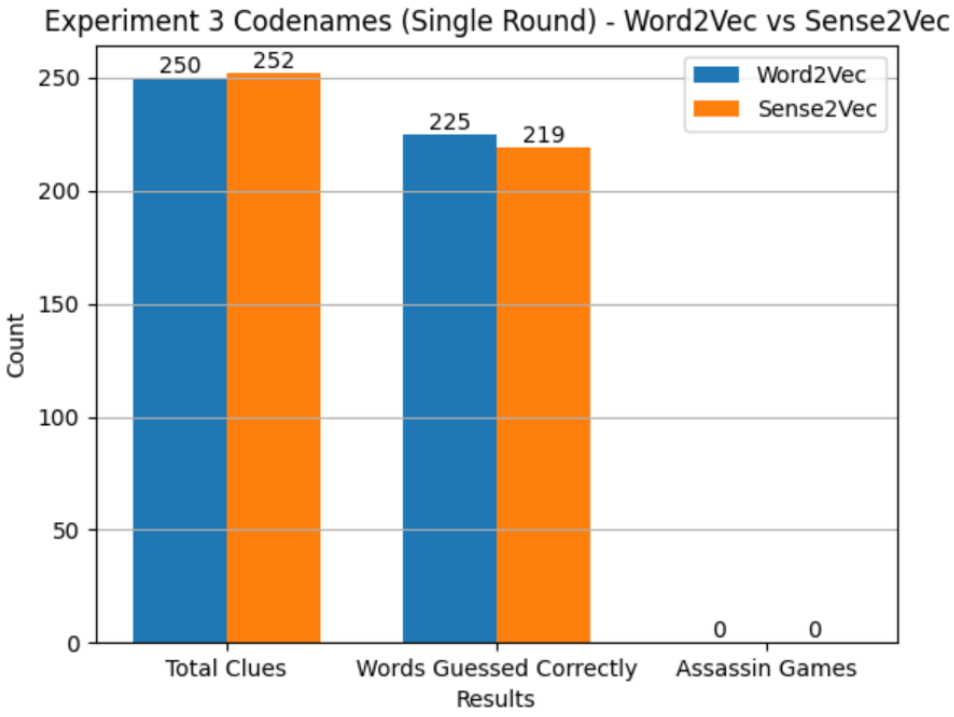

Figure 4.3: Results collected of each seed and embedding model combined visualized using a bar chart for Experiment 3. All the results were collected from each seed and the total number of clues and words guessed correctly were observed for a single round. The assassin games were also observed.

Experiment 3 (see Figure 4.3), focuses on the accuracy of guesses during the initial round of Codenames, which inherently presents the most challenging task due to the need to decipher clues from a pool of 25 words. By scrutinizing the correctness of guesses in this critical phase, insights into the guessers' ability to disambiguate words can be obtained.

From the findings of Experiment 3, both Sense2Vec and Word2Vec demonstrated commendable performance by successfully avoiding any hits on assassin words during the initial round. However, Word2Vec exhibited superior proficiency in deciphering clues compared to Sense2Vec. Sense2Vec provided a total of 252 clues, with 219 intended words guessed correctly, yielding an 86.9% success rate. In contrast, Word2Vec showcased enhanced performance in clue interpretation during the first round. It provided a total of 250 clues, with 225 of them guessed correctly, resulting in a superior success rate of 90%.

These results underscore Word2Vec's effectiveness in generating accurate clues and subsequent word associations.

4.2.4 Final Results

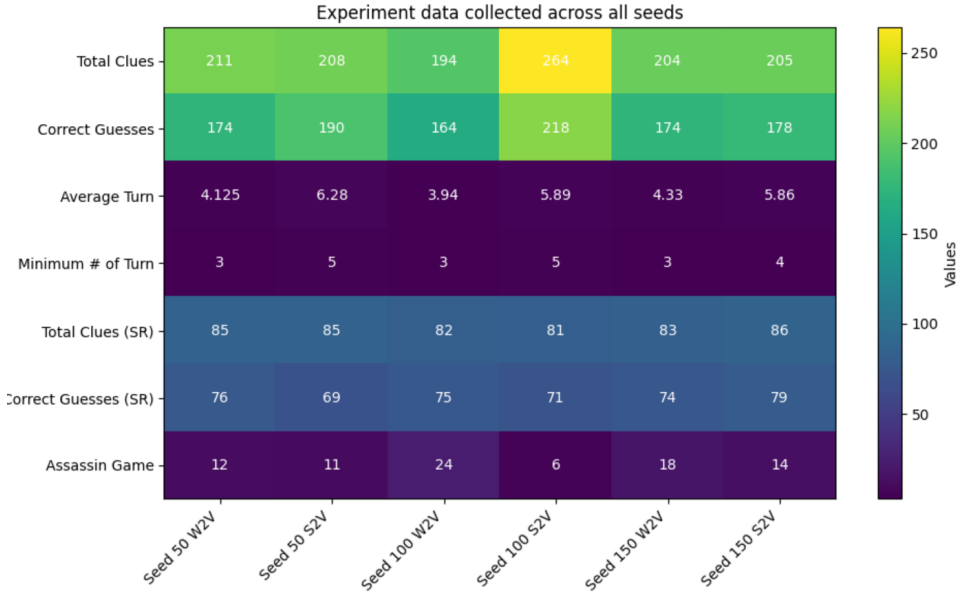

Figure 4.4: Final results collected of each Seed and embedding model visualized using a heat map matrix. Each Seed consisted of 90 games where the 7 following metrics were observed. "SR" denotes "Single Round"

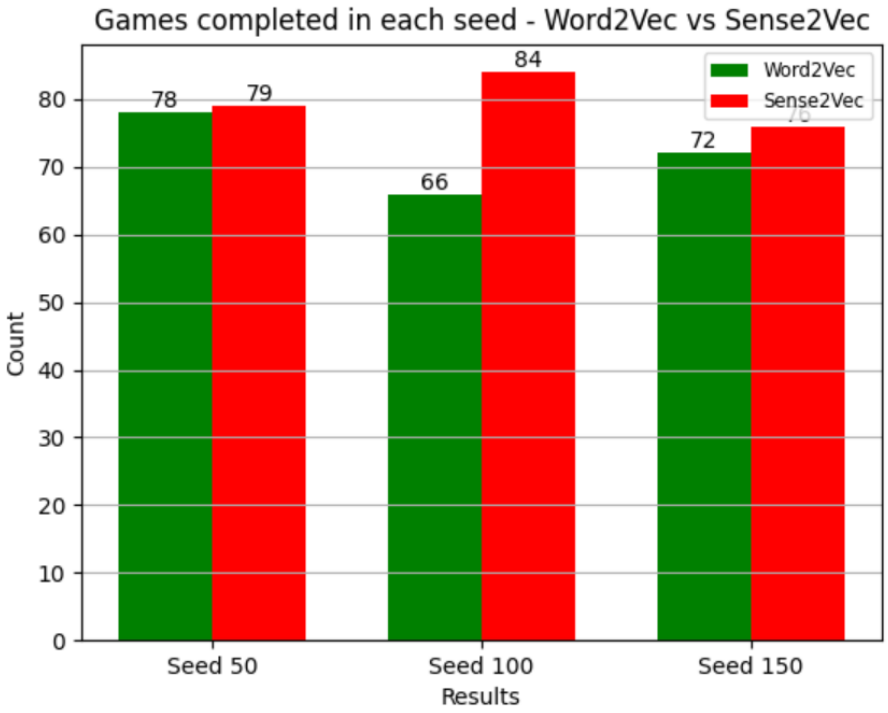

Figure 4.5: Results collected of games that were completed for each seed visualized using a bar chart. Each seed consisted of 180 games.

4.3 Overall Discussion

The findings from the experiments comparing Sense2Vec and Word2Vec in the context of Codenames gameplay consistently demonstrate the superiority of Sense2Vec in word sense disambiguation. Across over 540 games, with 270 games employing different embedding models, Sense2Vec consistently outperformed Word2Vec. Specifically, Sense2Vec hit fewer assassin words and completed more games, resulting in a significantly higher win percentage of 88.5% compared to Word2Vec's win percentage 80% (percentages calculated from Figure 4.5).

In terms of deciphering clues, Sense2Vec exhibited a remarkable success rate of 86.6%, significantly surpassing Word2Vec's success rate of 84%. However, an intriguing observation arose in Experiment 3, where Word2Vec excelled in deciphering clues in the first round, achieving an impressive overall success rate of 90% compared to Sense2Vec's 86.9%.

Despite these nuances, Word2Vec demonstrated a strategic advantage in completing games more swiftly than Sense2Vec. Across all seeds (see Figure 4.4), Word2Vec consistently achieved a minimum of 3 turns, showcasing a more aggressive gameplay approach whereas Sense2Vec achieved a minimum of 4 turns. Word2Vec averaged 4.13 turns per game and Sense2Vec averaged 6.01 per game. However, both models fell short of achieving the remarkable performance reported by Kim et al. (2019), who averaged 3.3 turns per game. This disparity may be attributed to various factors such as algorithmic differences, vocabulary corpus, vocabulary sizes, and specific parameters outlined in Table 3.1.

Nevertheless, the research establishes Sense2Vec's superiority in word sense disambiguation over Word2Vec, despite Word2Vec's tactical advantage in completing Codenames. This conclusion holds significant implications for applications requiring precise word interpretations and strategic gameplay.

Chapter 5

Future Work

For future work, exploring alternative word embedding models like AdaGram (Bartunov et al., 2016), which adaptively adjusts the context window, holds promise. Such models offer a more nuanced understanding of word embeddings, potentially leading to improved senses of words. Furthermore, investigating diverse corpus types could provide insights into varying contextual understandings. Additionally, exploring clustering techniques and implementing weighted approaches for assassin words may further enhance word sense disambiguation strategies in Codenames gameplay.

It is worth mentioning that in Codenames, clue-giving is a shared activity between both teams. Consequently, each turn provides an opportunity for guessers to use the opponent's guessed words as clues, enabling them to strategically navigate away from those words. This tactic offers a notable advantage in terms of reducing the average number of turns needed to complete a game and increasing the likelihood of correctly guessing the intended words, as players can leverage the opponent's clues in each round. Future work can account for such notable advantage.

While this study yielded promising results, it is essential to acknowledge that the research environment does not mirror the nuanced interactions between human spymasters and guessers. This difference arises from the assumption within the research that both parties share an identical vocabulary, thereby possessing perfect knowledge of each other's context. However, this assumption overlooks how individuals learn, interpret, and expand their vocabularies. For instance, consider the scenario of an adult playing Codenames with a child, highlighting the inherent discrepancies in shared linguistic understanding.

Consequently, it becomes evident that two humans cannot possess identical contextual frameworks. Future research endeavors within the realm of Codenames should consider these factors, exploring how differences in vocabularies and vocabulary sizes influence the dynamics between spymasters and guessers.

Chapter 6

Conclusion

In this study, we investigated the effectiveness of two distinct word embedding models within the context of Codenames gameplay. Our analysis incorporated existing literature utilizing two embedding models. The performance evaluation of the two embedding models centered on their autonomous gameplay dynamics, particularly focusing on the interaction between the Spymaster and Guesser. Following 540 gameplay sessions, it became evident that Sense2Vec showed superiority in word sense disambiguation. Sense2Vec demonstrated a notably lower frequency of hitting assassin words, coupled with a higher win percentage of 88.5%, in contrast to Word2Vec's 80% win rate. Additionally, Sense2Vec displayed enhanced accuracy in deciphering clues to identify the intended words with an accuracy of 86.6% and Word2Vec accuracy of 84%.

However, it is crucial to acknowledge that Word2Vec showcased swifter gameplay, achieving the best minimal turn of 3 in completing the game. It was also worth noting that various parameters influence clue generation, such as the choice of vocabulary corpus. Notably, Word2Vec was trained on the Google News corpus, while Sense2Vec utilized Reddit Comments. The impact of vocabulary size and algorithmic differences will further contribute to the observed changes in performance between the embedding models.

Further research delving into these factors, alongside future avenues of investigation, particularly concerning the word sense disambiguation challenge intrinsic to Codenames gameplay, holds promise for advancing our comprehension of Natural Language Processing applications within gaming contexts.

Number of words until this point, excluding front matter: 7291.

Bibliography

AI, E., 2016. sense2vec. https://github.com/explosion/sense2vec?tab=readme-ov-file#pretrained-vectors.

Bach, K., 2014. Context dependence. *The bloomsbury companion to the philosophy of language* [Online], pp.153–184. Available from: https://philpapers.org/rec/BACCDS.

Bartunov, S., Kondrashkin, D., Osokin, A. and Vetrov, D., 2016. Breaking sticks and ambiguities with adaptive skip-gram [Online]. In: A. Gretton and C.C. Robert, eds. *Proceedings of the 19th international conference on artificial intelligence and statistics*. Cadiz, Spain: PMLR, *Proceedings of Machine Learning Research*, vol. 51, pp.130–138. Available from: https://proceedings.mlr.press/v51/bartunov16.html.

Edition, C.G., 2023. Codenames [Online]. Available from: https://codenames.game/.

Friedman, D. and Panigrahi, A., 2021. Algorithms for codenames [Online]. Available from: https://www.cs.princeton.edu/~smattw/Teaching/521FA21/FinalProjectReports/FriedmanPanigrahi.pdf.

Google, 2013. word2vec [Online]. Available from: https://code.google.com/archive/p/word2vec/.

Jaramillo, C., Charity, M., Canaan, R. and Togelius, J., 2020. Word autobots: Using transformers for word association in the game codenames [Online]. *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. vol. 16, pp.231–237. Available from: https://ojs.aaai.org/index.php/AIIDE/article/view/7435.

Jung, C.G., 1910. The association method. *The american journal of psychology* [Online], 21(2), pp.219–269. Available from: https://www.jstor.org/stable/1413002.

Kim, A., Ruzmaykin, M., Truong, A. and Summerville, A., 2019. Cooperation and codenames: Understanding natural language processing via codenames [Online]. Available from: https://ojs.aaai.org/index.php/AIIDE/article/view/5239.

Kochmar, E., 2022a. Getting started with natural language processing [Online]. Available from: https://learning.oreilly.com/library/view/getting-started-with/9781617296765/?sso_link=yes&sso_link_from=UnivOfBath.

Kochmar, E., 2022b. Welcome to week 8: Semantics and meaning representation [Online]. Available from: https://bath-prod-sss1.s3.eu-west-1.amazonaws.com/0e/fa/0efa3b81be3ce79809ea6cadd8c48f00045bb53b?response-content-disposition=inline%3B%20filename%3D%22Handout_Week8.pdf%22&response-content-type=application%2Fpdf&X-Amz-Content-Sha256=UNSIGNED-PAYLOAD&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAJBFBMNJTZPM2NVZA%

2F20240501%2Feu-west-1%2Fs3%2Faws4_request&X-Amz-Date=20240501T175453Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21547&X-Amz-Signature=5961b18d19585d8e07be9c527e8de1adf5ead026244483ed7ec64a0640b9e9af.

Kulshrestha, R., 2010. Nlp 101: Word2vec — skip-gram and cbow [Online]. Available from: https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314.

Li, J. and Jurafsky, D., 2015. Do multi-sense embeddings improve natural language understanding? *arxiv preprint arxiv:1506.01070* [Online]. Available from: https://arxiv.org/abs/1506.01070.

Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013a. Efficient estimation of word representations in vector space. *arxiv preprint arxiv:1301.3781* [Online]. Available from: https://arxiv.org/abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013b. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems* [Online], 26. Available from: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.

Miller, G.A., 1995. Wordnet: a lexical database for english. *Communications of the acm* [Online], 38(11), pp.39–41. Available from: https://dl.acm.org/doi/abs/10.1145/219717.219748.

Navigli, R., 2009. Word sense disambiguation: A survey. *Acm computing surveys (csur)* [Online], 41(2), pp.1–69. Available from: https://dl.acm.org/doi/abs/10.1145/1459352.1459355?casa_token=cf74KRtTIEMAAAAA:3HPDIqSCcyTYR526PEa9g1PKAZq9WPQD3i-wNG6rkPtavzeH8nu_3PcCtkT8YTL0xVwJp_ARkw.

Neiman, J., 2017. How to create a codenames bot part 1: Word2vec. *Medium* [Online]. Available from: https://towardsdatascience.com/how-to-create-a-codenames-bot-part-1-word2vec-62701de38e66.

Paste, H., 2016. Codenames - play online [Online]. Available from: https://www.horsepaste.com/.

Pennington, J., Socher, R. and Manning, C.D., 2014. Glove: Global vectors for word representation [Online]. *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)*. pp.1532–1543. Available from: https://aclanthology.org/D14-1162.pdf.

Rovatsos, M., Gromann, D. and Bella, G., 2018. The taboo challenge competition. *Ai magazine* [Online], 39(1), pp.84–87. Available from: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2779.

sagelga, 2021. Codenames word list [Online]. Available from: https://github.com/sagelga/codenames/blob/main/wordlist/en-EN/default/wordlist.txt.

Shen, J.H., Hofer, M., Felbo, B. and Levy, R., 2018. Comparing models of associative meaning: An empirical investigation of reference in simple language games. *arxiv preprint arxiv:1810.03717* [Online]. Available from: https://arxiv.org/abs/1810.03717.

Summerville, A., 2019. The codenames ai competition [Online]. Available from: https://sites.google.com/view/the-codenames-ai-competition.

Trask, A., 2016. Sense2vec - a fast and accurate method for word sense disambiguation in neural word embeddings. [Online]. Available from: https://arxiv.org/pdf/1511.06388.

Wikipedia contributors, 2024a. Cosine similarity — Wikipedia, the free encyclopedia [Online]. Available from: https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1207802784.

Wikipedia contributors, 2024b. Sigmoid function — Wikipedia, the free encyclopedia [Online]. Available from: https://en.wikipedia.org/w/index.php?title=Sigmoid_function&oldid=1220784304.

Appendix A

Code

The link to the code is available on Bath One Drive:
https://computingservices-my.sharepoint.com/:u:/g/personal/dn444_bath_ac_uk/EWkKyfbyq4lBsUeoEeYwXicBv59TURayRTMUxT2kWl6ZXg?e=t57874

Appendix B

Raw Code Output

```
Creating board...
------------------------------
------ Starting Codenames------
------------------------------


-----OPTIONS TO INPUT-----
pgb: prints the guesser's board
guess (your word): make a guess on the board
sg: give suggestions for current clue and number
pch: print past clues given



==========
BLUE TURN
==========

Scores: ['red': 8, 'blue': 9]
```

calf	amazon	ghost	superhero	york
soul	time	giant	sink	alps
green	dinosaur	berlin	chick	fence
thief	america	thumb	club	degree
square	england	olive	czech	port

```
Generating clue...

Blue: burglars, 1
>>>
```

Figure B.1: An illustrative depiction of an initial Codenames gameboard. The available commands for user input are displayed. The output includes the team commencing the turn, the current score, and the automatically generated clue along with its intended number

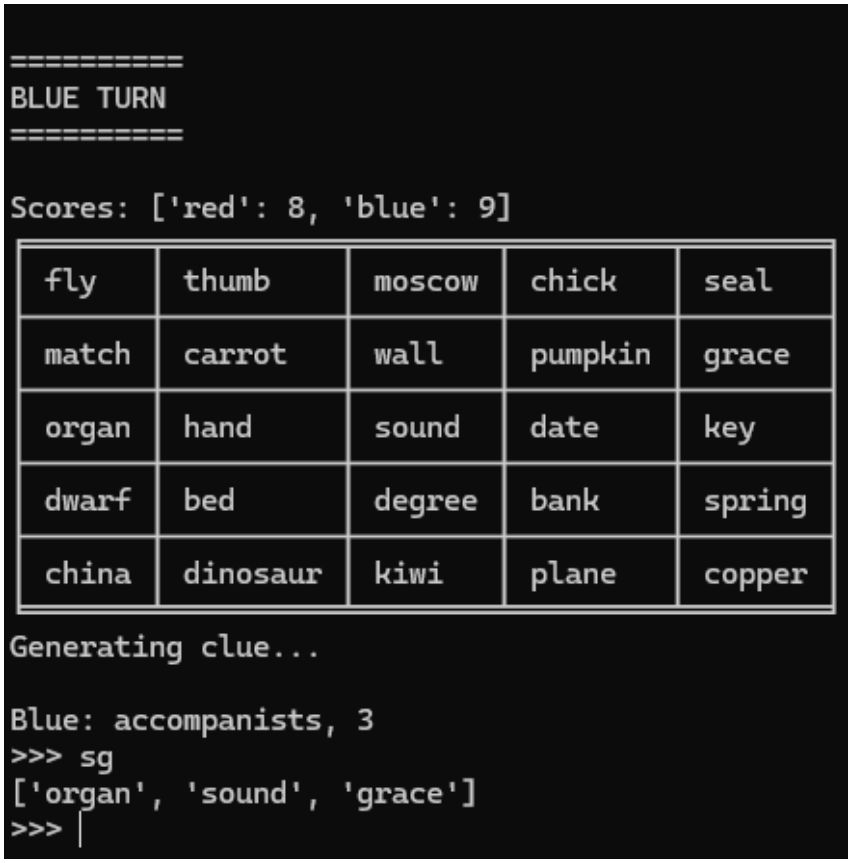

Figure B.2: An illustrative representation of guesser's word suggestions. In response to the clue "accompanists" with an intended number of 3, the displayed words on the board include "organ," "sound," and "grace

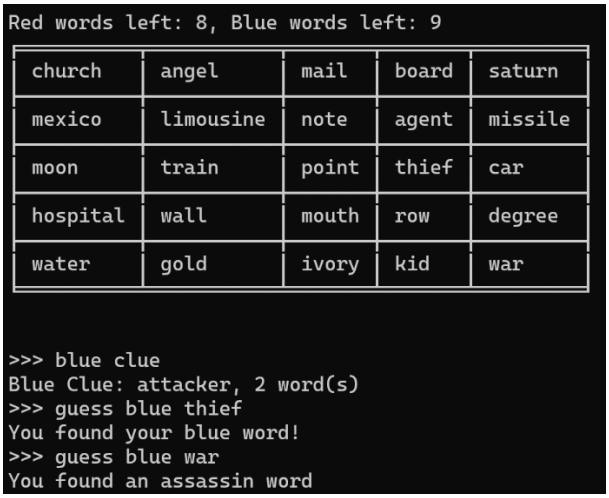

Figure B.3: An instance showcasing an ineffective clue offered by a Spymaster during the first round. The clue provided was "attacker," which led the guesser to inadvertently select the assassin word, resulting in an immediate loss

Figure B.4: An instance of an effective clue presented by a Spymaster during gameplay. In this instance, the clue provided was "milan." As the guesser, all three associated words were accurately identified and revealed