# NLP Homework 1 Report

## What embedding model do you use?

I used the `Word2Vec` model from the `gensim` library to do word embedding.

## What are the pre-processing steps?

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
analyze = count_vect.build_analyzer()

def pre_process(line):
    # Remove punctuation and tokenize
    line = re.sub(r"[^\w\s]", "", line)
    # Remove stopwords, tokenization and lowercasing
    words = analyze(line)
    return words
```

1. Removed punctuation

   I used regex to remove all punctuations in a line.

2. Remove stop words, tokenization and lowercasing

   By referencing the official documentation of `build_analyzer` in `sklearn`, the analyze method can splits the line into tokens, convert all characters in the text to lowercase, and remove stop words such as 'the', 'that', etc.

## What are the hyperparameter settings?

The model was instantiated with the following command.

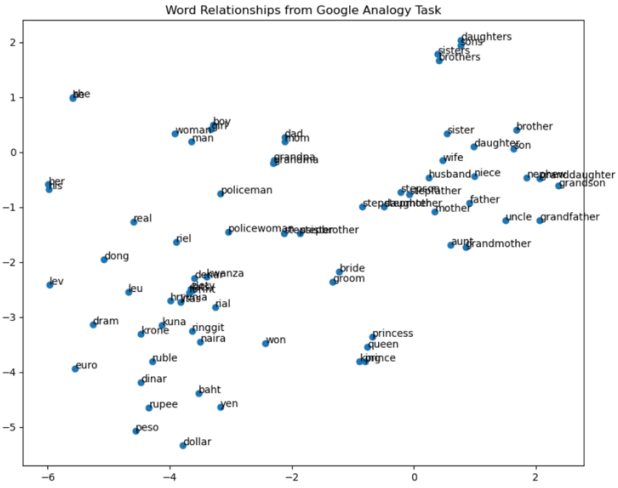
```
model = Word2Vec(sentences, vector_size=100, window=5, min_count=1, workers=4)
```

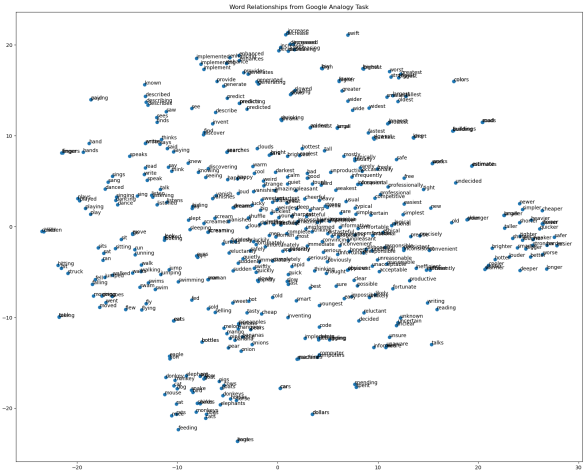
The hyper parameter settings are:

- `sentences`: each sentence in this list is preprocessed into word vectors
- `vector_size=100`: this specifies that the dimensionality of the word vectors is 100
- `window=5`: this defines the maximum distance between the current and predicted word within a sentence
- `min_count=1`: this means that the model includes all the words in the corpus, even if the word only appeared once
- `workers=4`: this means that four cores are used to train the model

## What is the performance for different categories or subcategories?

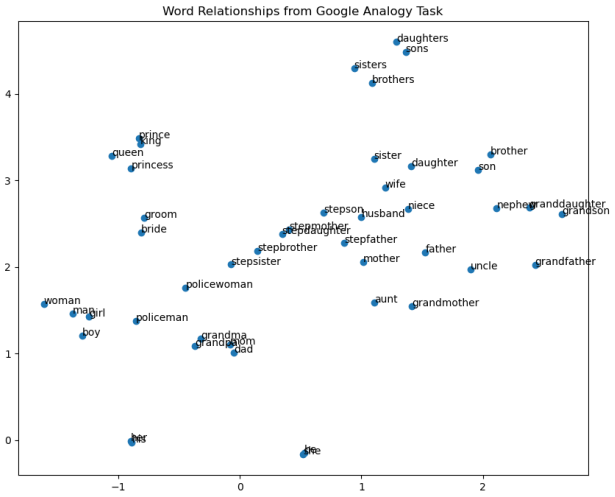
### 1 Different categories

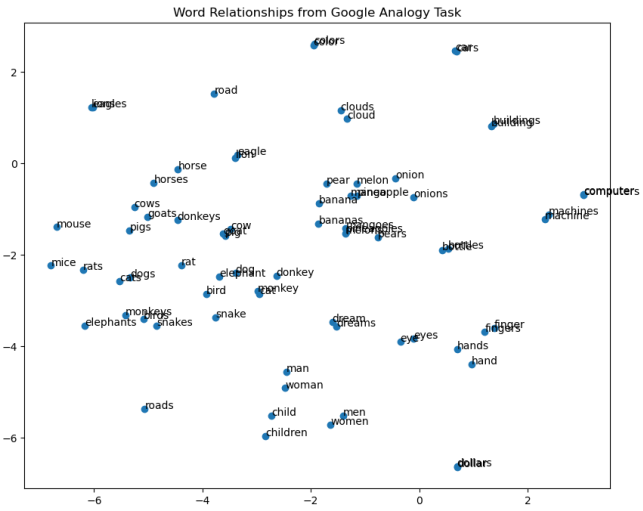



The "Semantic" category outperformed the "Syntactic" category. Look at the plots above: the one on the left was created using the "Semantic" category, while the other one used the "Syntactic" category. Clearly, the data points on the right-hand side are more scattered, indicating weaker relationships between the word vectors. In contrast, in the "Semantic" plot, you can definitely see some kind of relationship between words like "her" and "his", “bride” and “groom”, etc.

2 Different subcategories




The two plots above were created using the “: family” and “: gram8-plural” sub-categories, from left to right, respectively. In my humble opinion, the model explained the “: gram8-plural” category better. You can tell by looking at the singular and plural forms in the dataset, where the dots representing these word pairs often overlap or are very close to each other. This indicates that the model captures the relationship between singular and plural forms effectively. On the other hand, in the “: family” category, the data points are more scattered, showing less consistency in capturing the relationships between family-related terms.

Note that there is something I tried in main.py that didn’t work. My attempt to use the “: currency” sub-category failed because of the error: `ValueError: perplexity must be less than n_samples`. I believe this happened because my vocabulary dataset doesn't contain enough words related to "currency." Therefore, there aren’t enough words to show the relationships between word vectors.

```
ValueError: perplexity must be less than n_samples
```
*Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...*

## What do you believe is the primary factor causing the accuracy differences for your approach?

I believe that the primary factor is the hyperparameter values when constructing the model:

`model = Word2Vec(vector_size=100, window=5, min_count=1, workers=4)`

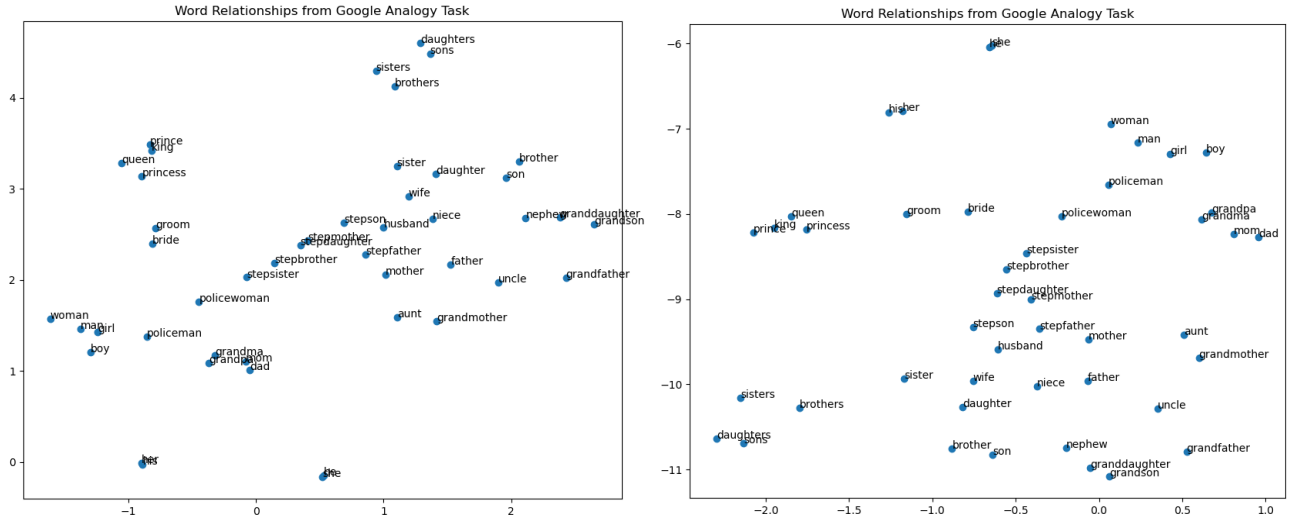
A vector size of 100 might not be able to capture the complex relationships between words, a bigger window may not capture long-distance word relationships, and the min count might be too low to include unrelated words to train the data. By tuning these values, my model could perform differently.

## What's your discovery from your t-SNE visualization plots?

When I compared the plots of my word embeddings with those from a pretrained model, I found the points in my plot were closer together. This implies that my model isn't as good at separating different words compared to the pretrained model. The pretrained embeddings show better separation, likely because they were trained on more data and learned more detailed differences between words. The tighter clusters in my model's plot might mean it doesn't distinguish between words as well.

## What's the difference in word representations if you increase the amount of training data?

The plot on the left was created using a model trained with 20% of the data, while the one on the right was trained with 30% of the data. Even though both plots show tighter clusters compared to the pretrained model, there is a slight improvement with the 30% sample. In the right plot, the model separated the word vectors more effectively. However, I think there are no significant differences between the two models.




## Why did you train your model in batches of data?

When I was doing the homework, I ran into the issue of not having enough RAM. I researched the problem and learned that if I read or write the entire text file at once, it would overload the RAM. To avoid this, I decided to process the data in batches, which allowed me to handle the large dataset without running out of memory.

Environment:

- Running Environment:
  - Operating System: macOS
  - Processor: Apple Silicon (M1 pro)
  - Architecture: ARM64
- Python Version: 3.10.15

AI Usage Explanation

1. Plt

```
# These two lines are generated by ChatGPT
plt.figure(figsize=(10, 8))
plt.scatter(word_vectors_in_2d[:, 0], word_vectors_in_2d[:, 1])
```

Since this is the first time for me to use a plotting library, I consulted with ChatGPT and learned from asking it. The followings are my understanding about the code:

1 The first line creates a new plot with a size of 10x8 inches.

2 The second line creates a scatter plot where:

2.1 `word_vectors_in_2d[:, 0]` provides the x-coordinates

2.2 `word_vectors_in_2d[:, 1]` provides the y-coordinates

2. Vocabulary buiding

```
# Reference: https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html
# Also, I asked ChatGPT for help on the code block below.
with open("output_20.txt", "r", encoding="utf-8") as infile:
    for line in tqdm(infile, desc="Processing lines for vocabulary build"):
        processed_line = pre_process(line)
        if processed_line:
            sentences.append(processed_line)

        if not vocab_built and len(sentences) >= batch_size:
            model = Word2Vec(vector_size=100, window=5, min_count=1, workers=4)
            model.build_vocab(sentences)
            vocab_built = True
            break
```

Since I am not sure about the vocabulary building process, I asked ChatGPT for help and here is what I learned:

This part initializes the model and builds its vocabulary. The second if statement is to make sure that when enough sentences are processed, the building process would stop. Also note that the vocabulary was constructed from the processed sentences.