Abstract
In this project, we produce a reverse dictionary, which allows one to find a word that they can’t remember by describing its meaning. A reverse dictionary takes in a word definition as an input query and returns the top-k candidate words that are most likely to match this definition. The input can be similar to a dictionary definition, or even a colloquial description of the desired word. For example, the input “a small vessel propelled on water” should yield an output of “boat”.

We compare multiple possible approaches to find the best way to implement a reverse dictionary, both in terms of accuracy and in terms of computational workload and speed. We find that an encoder-decoder model trained on a BERT encoder with either linear decoder layers or an LSTM decoder yields the best results, but similar accuracy can be obtained even with lightweight BERT models.

Methods
We tokenize all input queries, then pad and truncate to the same length. Tokens are passed into a BERT layer to generate the query encoding. We test two different model variants. First, we use a decoder stack composed of fully connected linear layers, ReLU, BatchNorm, and Dropout layers to train for 20 epochs. The final output is a 100-dimensional predicted word embedding.

We tokenize all input queries, then pad and truncate to the same length. Tokens are passed into a BERT layer to generate the query encoding. We test two different model variants. First, we use a decoder stack composed of fully connected linear layers, ReLU, BatchNorm, and Dropout layers to train for 20 epochs. The final output is a 100-dimensional predicted word embedding.

Experiments
After training both our model variants, we see that the linear decoder and LSTM decoder both reach similar accuracy levels on the validation set. However, LSTM reaches significantly higher accuracy on the training set.

Analysis
We find distribution of less among samples is similar, but the LSTM model is more robust with fewer outliers.

We also find that the top-1 and top-10 accuracy scores increase by nearly 3.5x if evaluated on just the 2000 most frequent words.

Conclusion
We find that our approach can accurately extract semantic meaning from a description of a word and provide a good prediction of target words. Overall, we believe the LSTM decoder has more robust behavior even if it’s slower to train.

Although our model finds the top 4 words out of a vocabulary of 400,000, a typical English speaker’s vocabulary is much more limited (order of a few thousand per day), our reverse dictionary could be even more accurate and useful if we removed rare and unused words.

Key References

Problem
A common phenomenon is the tip-of-the-tongue problem, where you can’t quite remember a specific word or phrase you are thinking of. We propose a “reverse dictionary” as a solution to this problem, which takes a word description or definition as an input query and returns a list of the top candidate words.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small amount</td>
<td>small, snap, snap, little, bit</td>
</tr>
<tr>
<td>Winter sport</td>
<td>skiing, snowboarding, sledding, ice skating, curling</td>
</tr>
</tbody>
</table>

Some use cases for a reverse dictionary:
- Recalling a word on the tip of your tongue
- Finding synonyms for a word
- Learning new words for non-native speakers

Background
The research space for reverse dictionaries is relatively sparse. There are very few papers that utilize recent state-of-the-art methods like BERT or neural networks[1]. Reverse dictionary implementations often compare word counts in inputs to known word definitions. As a result, there is still a lot of research that can be done to improve the performance of reverse dictionaries.

Dataset
For our dataset, we use word-definition pairs from WordNet and the Online Plain Text English Dictionary, combining for approximately 300,000 pairs. We construct an 80/20 training/validation split.

We use pre-trained GloVe embeddings[2] for our model. The pre-trained set contains 300-dimensional embeddings for 400 thousand words.