Final problem:

Statement: find the “answer” to a question of the type “2\*man – 3\*woman + tree = ?” (you may have seen “king – man + woman =… (queen)”.

What we need: database, model, live or not?

Ideal product: how to impress people with this? Give them an UI to enter their own queries!

* Box for query entry
* Area for displaying the answer
* (optional): some “music” while they wait (if needed)

Tasks / chunks of work to get there?

1. Language choice (assumptions / limitations)
   1. For query – English
   2. For coding – Python
   3. No other choices
   4. Learn further: language detection from short passages (recent article?)
2. (UI) “Webpage” with:
   1. box for text entry (e.g. “king – man + woman”)
   2. area for display of answer
   3. need these two to communicate with a computing environment behind the scenes, for:
      1. text processing
      2. calculations / selection / ranking of potential results
      3. command to display the answer
   4. **someone will have to do this (back-up plan: use a string in Spyder/Jupyter notebook)**
3. (Backend) Python to do what:
   1. (assume) the text we entered is stored in a string – assume it is UTF8 (Mark to explain)
   2. Parse it – why? To find the words AND the coefficients (with sign) (or weights)
      1. E.g.: our string would look like: +[king] –[man] +[woman]
      2. Even better: two list:
         1. Words list: [“king”, “man”, “woman”]
         2. Weights list: [+1, -1, +1]
      3. **Someone will have to do this**
   3. Need arithmetic operations between words - ????
   4. How?
      1. Context is king – man and woman could be antonyms (e.g. on binary gender forms) or much closer to synonyms (e.g. when talking about humans vs trees)
      2. Dummy variables: one dummy for each word (also potentially for part of speech) – this is a large space 200K for words x 2-ish for PoS => close to 400K dummies. This is essentially one-hot encoding
         1. OK but: performing arithmetic with 3.b.ii leaves us with a 400K-ish dimensional vector with mostly 0 (very sparse), two 1’s and one -1. Now what?
         2. Problem: with one-hot encoding, every word is equidistant from any other word. So “closest” is not working very well. BUT it provides a good direction – use a vector to represent each word, because then you can do arithmetic on vectors!
      3. **Word embeddings:** 
         1. Usually fixed length numeric vectors, e.g. 50, 300, 600 dimensions
         2. Obtained as a byproduct of an “encoder-decoder” architecture (usually NN)
         3. Idea: use a LOT of text (think Wikipedia, AP, project Gutenberg, tweets, financial results, news, etc.). Chunk this into sequences of 5 words, then somehow force a neural network to: input the middle word, and output the other 4 words (word2vec setup, more or less).
            1. RV to share some links for CBOW vs this setup
         4. Word embeddings are available, pre-computed, based on English language publicly available text, from e.g. GloVe (gensim package)
            1. **Someone will have to get familiar with this**
            2. **And prepare some smaller datasets to facilitate search**
            3. **Think functions:**

**Word\_to\_vector(string)**

Yell and scream if: string has more than one word, or the word is not found!

**Vector\_find\_closest(vector, metric)**

Talk to the next guys

* + 1. Let’s say we have these word embeddings
       1. More precisely, a dictionary: input word, output vector
       2. Now what?
          1. Take each word in the words list, lookup its embedding as a vector
          2. Use the weights list to compute the linear combination of the vectors with those weights
          3. We get what? A vector of the same dimension!
          4. Ideal situation: it matches exactly one (and only one) of the vectors in the dictionary => we read the key => done
          5. Real situation: need to find the closest vector, or maybe the top 5 closest vectors
          6. Problem: what is “closest”?

Usually cosine similarity: disregards vector length (normalizing them), then computes the dot product => back up the angle between the vectors

Highest cosine similarity ⬄ smallest angle ⬄closest

Another choice: mean square error between vector components. What’s the problem with this? Complexity – may be more expensive than dot product.

Or Word Movers’ Distance

**Someone (or two) will have to look at these distances, and implement / include them**

**Think about parsing (speed) the 400K (?) vectors**

**Talk to the UI guy to either implement a drop-down selection of similarity metric, OR to send 3 answers, one for each similarity**

* + - * 1. Do we have enough computing power for this?

400K words x 300 dimensions x 4 bytes / dim => 500MB dataset (likely more – see gensim)

Test speed of various “closest” / distance metrics above

* + 1. Last: send the word/answer to the UI to be displayed
       1. Suggest a dictionary: {metric\_used: answer\_word}
       2. Even better: {metric\_used: answer\_word\_top5\_list}
       3. Best: {metric\_used: {word:distance, word:distance,…}}