3. Deep Learning in NLP

a. Synonym Expansion - **Chapter 2**
   i. A synonym filter in the pipeline uses a neural network to generate synonyms
   ii. Improvement in Recall - Reduce the number of queries that generate 0 results
   iii. The overall idea of synonym expansion is that when the search engine receives a stream of terms, it can enrich them by adding their synonyms, if they exist, at the same position.
   iv. The simplest and most common approach for implementing synonyms is based on feeding the search engine a vocabulary that contains the mapping between all the words and their related synonyms
   v. The reason for this is that you don't need synonym expansion at both query time and index time. For two synonyms to match, it's sufficient to do expansion once. (Do it only once)
   v. WordNet recommendation for Syn Exp
   vi. Synonyms based on Context
      1. One approach to overcoming these limitations is to have a way to generate synonyms from the data to be ingested. The basic concept is that it should be possible to extract the nearest neighbors of a word by looking at the context of the word, which means analyzing the patterns of surrounding words that occur together with the word itself. A nearest neighbor of a word in this case should be its synonym, even if it's not strictly a synonym from the grammar perspective.
   vii. A short text has a relatively poor probability of being hit with a query, because it's composed of only a few terms. So you might decide to focus on such documents and expand their synonyms, rather than focusing on longer documents. On the other hand, the "informativeness" of a document doesn't only depend on its size. Thus you might use other techniques, such as looking at term weights (the number of times a term appears in a piece of text) and skipping those that have a low weight.
   viii. Pick synonyms for words based on
      1. The similarity to the nearest neighbors
         a. One thresholding idea - Drop everything below a threshold (0.1) from the max value - Max sim to nearest neighbor is 0.7 drop everything below 0.6
      2. PoS tags - if it is an ADJ then expand
      3. Length or nature of the document and possible context
   x.

xi.

b. Word2vec Fast Trivia **Section 2.4.1**
    i. With the CBOW model, the target word is used as the output of the network, and the remaining words of the text fragment (the context) are used as inputs. The opposite is true with the continuous skip-gram model: the target word is used as input and the context words as outputs (as in the example). In practice, both work well, but skip-gram is usually preferred because it works slightly better with infrequently used words.
    ii. Words that don't have similar meanings might have same or similar context words appearing around them, causing word2vec models to place them near by. E.g Imagine that you have the following sentences: "I like pizza," "I hate pizza," "I like pasta," "I hate pasta," "I love pasta," and "I eat pasta." This would be a small set of sentences for word2vec to use to learn accurate embeddings in real life. But you can clearly see that the terms "I" on the left and "pizza" and "pasta" on the right all share verbs in between. Because word2vec learns word embeddings using similar text fragments, you may end up with similar word vectors for the verbs "like," "hate," "love," and "eat." So word2vec may report that "love" is close to "like" and "eat" (which is fine, given that the sentences are all related to food) but also to "hate," which is definitely not a synonym for "love."
c. Automatic query expansion is the name of the technique of generating (portions of) queries under the hood to maximize the number of relevant results for the end user.
    i. Generate new words using RNNs or other methods to add to the query - rewrite the query
    ii. Query Time
    iii. OR or some way to merge the original query and the new rewritten query.
    iv. You don't want to generate an alternative query representation that's perfect but not useful.
    v. In practice, it's common to use query logs for such
    vi. learning tasks because
    vii. Query logs reflect the behavior of users on that specific system, so the resulting model will behave relatively close to the actual users and data. Using or generating other datasets may incur additional costs while possibly training a model that's based on different data, users, domains, and so on.
    viii. correlating queries generate similar search result sets, that come from the same users in specific time windows, or that contain similar search terms - Can be used as dataset for training
        1. The first approach is to group queries that share a portion of their associated search results

2. The second potential approach relies on the assumption that users search for similar things in small time windows.
3. Correlate queries that share terms

ix. If you plan to use synonym expansion, you probably should not expand on every possible synonym; you could instead do so only for input queries that don't have a corresponding alternative query,

x. Use Indexed Data
1. Doing so will likely help reduce the number of queries with zero results, because the hints in the alternative queries don't come from the user-generated queries but rather from the text of relevant documents.

xi. The key takeaway is that you aren't limited to using queries to train the neural network, as long as the target output is correlated with the input in a way that's useful for representing an alternative query.

xii. Unsupervised

d. RNNs - LSTMs - Revision - Section 3.3 - 3.4
e. AutoComplete Section 4.3 - UI Discussion too - **Read this section**
f. It's interesting to note that I also got the following:
1. An infix suggestion (a suggestion string containing new tokens placed infix—between two existing tokens of the original string). In the "books about google search" suggestion, the word "google" is between "about" and "sear" in the query I typed. Keep this in mind, because this is something you'll want to achieve later; but we'll skip it for now.
2. A suggestion that skipped the word "about" (the last three, "books search…"). Keep this in mind also; you can discard terms from the query while giving suggestions. -

ii. There's no point when the end user can't count on suggestions, even if they aren't particularly accurate. Remove terms from the suggester when they have no possible match at search time.

iii. Stupid Backoff?

iv. You can combine the results of the original suggester with word vectors to augment the diversity of the suggestions.

f. Ranking
i. It's often mentally easier for a user to write a better query than to scroll down and click the Page 2 button on the results page

ii. a smart search engine should consider the following: User history—Record the past activity of a user and take it into consideration when ranking. For example, recurring terms in past queries may indicate a user's interest in a certain topic, so search results on that same topic should have a higher ranking.

iii. All possible context clues—Look for signals to provide more context to the query. For example, look at the search logs to see whether a query was previously performed; if so, check the next query in the search log to see if there are any shared results, and give them a higher ranking.

iv. Overall, the (simplified) idea is that a document that's relevant with respect to a certain query should be returned even if there's no exact match between the query and the indexed terms.

v. A good retrieval model should consider semantics. As you can imagine, this semantic perspective applies to ranking documents, as well. For example:

1. When ranking a result whose matching terms came from one of the alternative queries generated by a LSTM network, should such documents score differently than documents that matched based on terms from the original user query?
2. If you plan to use representations generated via deep learning (for example, thought vectors) to capture user intent, how do you use them to retrieve and rank results?
3. **probability ranking principle**: if retrieved documents are ordered by decreasing probability of relevance to the data available, then the system's effectiveness is the best that can be obtained for the data

vi. w2v based ranking

1. Avg words into Passage/Doc vector
2. Cosine with Query
3. Term Smoothing using TF or TF-IDF
4. Other means - HM so lower values can bring the mean down

vii. Lucene4IR?

viii. Theoretical results and evaluations must always be measured against real-life usage of your search engine.

ix. Additionally, a good solution that's also supported by recent research can be to mix classic and neural ranking models by using multiple scoring functions at the same time.

g. Document/Paragraph vectors - **Chapter 6 - Section 6.1 - 6.2**

h. Recommendations - More like this

i. LMDirichletSimilarity

Ii.