Job Recommender System Notes

Research:

- **Python version**
- **Packages** (good comparison here)
    - PDF to text extractor
        - PDFminer
        - PyPDF2
    - NLP Libraries
        - Gensim
            - (doc2vec)
            - provides tf-idf vectorization, word2vec, doc2vec, latent semantic analysis, latent Dirchlet allocation
            - Doesn't have enough for full NLP pipeline so must combine with spaCy or NLTK
            - most commonly used for topic modeling and similarity detection
            - Dependencies
                - Python, tested with versions 2.7, 3.5, 3.6 and 3.7.
                - NumPy for number crunching.
                - smart_open for transparently opening files on remote storages or compressed files.
            - Youtube video for word2vec w/Gensim library and NLTK
        - Stanza
            - pipeline will include all processors, including tokenization, multi-word token expansion, part-of-speech tagging, lemmatization, dependency parsing and named entity recognition (for supported languages). However, you can always specify what processors you want to include with the processors argument.
        - Natural Language toolkit (nltk)
        - SpaCy
            - spaCy has support for word vectors whereas NLTK does not.
            - NLTK is a string processing library. It takes strings as input and returns strings or lists of strings as output. Whereas, spaCy uses object-oriented approach. When we parse a text, spaCy returns document object whose words and sentences are objects themselves.

        - Gives you fastest model for each thing you want to do. Faster than nltk
        - Slower sentence tokenization than NLTK
        - Phrase matching (use case here)
    - Labeling entities / unlabeled key phrases
        - Snorkel

- **Data**
    - Transfer learning
        - Penn Treebank (PTB) (Mikolov)
            - 2500 stories from WSJ link
            - No case, punctuation, numbers
        - Wikitext
            - Larger, retains original case, punctuation, numbers
            - WikiText-2 (2x larger than PTB)
            - WikiText-103 (110x larger than PTB)
- Existing models (and transfer learning?)

- **Techniques**
    - **Hierarchical softmax** (hs) (instead of vanilla softmax)
        - Hierarchical is much faster to train with SGD
            - Would use SGD instead of GD if large data set (vocab)
        - Can be used as long as there's softmax function on activation layer
            - 4 or 5 orders of magnitude decrease in compute costs
        - Requires binary tree
            - Huffman tree works for word2vec (chrismccormickai)
            - Rare words deeper in tree, common words shallow
        - Alternative is negative sampling…….
            - Gives up on softmax
            - Goes to sigmoid
            - Instead of training on 100,000 outputs, takes one positive for word and chooses 5 or so words to negatively add
    - Word vs subword information
    - Collaborative filtering
        - Known preferences of set of users to predict unknown preferences for new users if users are similar
            - Fastai library?
    - Content-based recommendation (CBR)

Links

- Word embeddings exploration explanation towardsdatascience
- KD nuggets pre-processing
- Medium post on phrase-matching in resumes using Spacy package (use case here)
- Automated Resume Screening System project (github)
- Word2Vec sentiment analysis tweets implementation
- Resume job-matching project (github)
    - Like this a lot.
    - Scraper of jobs
    - Scraper of resumes
    - Word2vec + tf-idf transform + add (or average?) for both job and resume
-

Isaac links:

- Word2Vec NumPy implementation: towardsdatascience
    - Embedding = transforming words into vectors
    - 2 Methods:
        - Continuous Bag of Words (CBOW) -- guess target word from neighboring words
        - Skip-Gram (SG) -- guess context words from target word

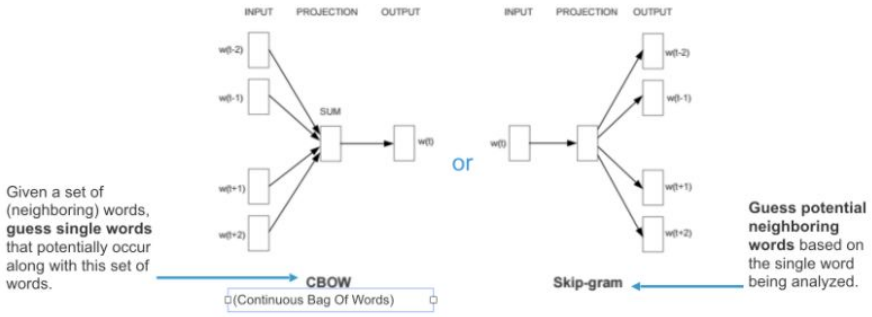

Fig. 2 — Word2Vec — CBOW and skip-gram model architectures. Credit: IDIL

-

- Doc2vec gensim implementation: gensim

Papers

- Mikolov et al., SEP2013, Efficient Estimation of Word Representations in Vector Space, arXiv
- Mikolov et al., OCT2013, Distributed Representations of Words and Phrases and their Compositionality, arXiv
- Le and Mikolov, MAY2014, Distributed Representations of Sentences and Documents, arXiv
- Schmitt et al, Matching Jobs and Resumes: a Deep Collaborative Filtering Task, EasyChair
- Stanford NLP Stanza, arXiv

Notes on Topics

Word embeddings exploration explanation (towardsdatascience)

One-hot encoding (CountVectorizing)

- The most basic and naive method for transforming words into vectors is to count occurrence of each word in each document, isn't it? Such an approach is called countvectorizing or one-hot encoding (dependent on the literature).

TF-IDF transforming

- weighting by exploitation of useful statistical measure called tf-idf. Having a large corpus of documents, words like 'a', 'the', 'is', etc. occur very frequently, but they don't carry a lot of information.

Word2Vec parameter learning explained

- One word per context aka CBOG
    - One-hot encoded vector as the input size of Vx1, input--> hidden layer weights matrix W of size VxN, hidden layer-->output layer weights matrix W' of size NxV and softmax function as final activation function.
    - Goal to calculate P(wj | wi) for word with index i.
- Multi-word context
    - No differences except type of probability distribution we want to obtain and type of hidden layer. Multiple word input → 1 target prediction.
- Skip-gram
    - Opposite
    - Predict c tonext words having one target word as input

GloVe (global vectors for word representation)

- GloVe model trains on global co-occurrence counts of words and makes a sufficient use of statistics by minimizing least-squares error and, as result, producing a word vector space with meaningful substructure

TowardsDataScience Implementation of W2V (SG)

1. Data Preparation — Define corpus, clean, normalise and tokenise words
    a. Note, they don't remove **stop words** like 'and' and 'is'
    b. KDnuggets article on text pre-processing
    c. Gensim library also provides a function to perform simple text preprocessing using gensim.utils.simple_preprocess where it converts a document into a list of lowercase tokens, ignoring tokens that are too short or too long.
2. Hyperparameters — Learning rate, epochs, window size, embedding size
    a. Window_size = 2
    b. N = 10 (number of dimensions of word embeddings aka size of hidden layer. It's your vocabulary size)
    c. Epochs = 50
    d. Learning_rate = 0.01
3. Generate Training Data — Build vocabulary, one-hot encoding for words, build dictionaries that map id to word and vice versa
    a. One-hot encoding = Dummy vars for words
    b.
4. Model Training — Pass encoded words through forward pass, calculate error rate, adjust weights using backpropagation and compute loss
5. Inference — Get word vector and find similar words
6. Further improvements — Speeding up training time with Skip-gram Negative Sampling (SGNS) and Hierarchical Softmax