# Job Recommender System Notes

**Overview:**

* The **big idea** is that you**represent documents as vectors of features**, and **compare documents by measuring the distance between these features**. There are **multiple ways** to **compute features** that capture the **semantics of documents**and multiple algorithms to capture **dependency structure of documents** to focus on meanings of documents. [link](https://medium.com/@adriensieg/text-similarities-da019229c894)

BLUF

* Use a simple **bag-of-words** approach to average sentence/document’s word vectors
  + Get it done, have a baseline <https://medium.com/huggingface/universal-word-sentence-embeddings-ce48ddc8fc3a>
* Improve with **Smooth Inverse Frequency** to not weight irrelevant words heavily
  + Cuts the noise
* Move forward to **FastText** (Word2Vec extension) to see out-of-vocab words
* Sentence encoders?
  + Google Sentence Encoder / FB Infersent?
* Step up from there
  + ELMo

Universal Sentence Encoder?

**Top Tier Language Representation Models**

* Best for contextual embedding of words (made a play at the plate versus broadway play versus children want to play)
* Contextually expensive
  + Perhaps able to overcome using pretrained + embedding all job descriptions ahead of time so you only need to embed and infer with one resume at a time?

BERT

* Bert-as-a-service is a Python library that enables us to deploy pre-trained BERT models in our local machine and run inference
  + Requires TF, bertr-as-a-service client and server
  + Computationally tough, but maybe a way around if you can have
* Transformer approach

XLNet

* Summer 2019. Beats BERT on some SOTA
* Computation
  + TPUs. Need 32-128 GPUs to reproduce…
* Transformer appraoch

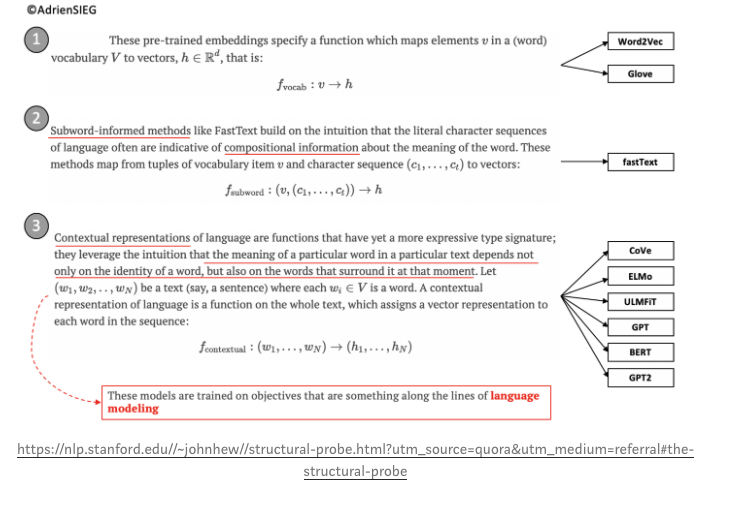
**Next up**

ELMo

* Contextual embeddings of words
* Possible to import pretrained ELMo model…still computationally tough but may be able to mitigate with batching
  + Requires TF?
* Bridges gap from static vector representations (GloVe, Word2Vec) that don’t have word context
* ELMo was trained to generate embeddings of words based on the context they were used in, so it solved both of these problems in one go. ELMo does compare favorably with the USE as a model that could be used for sentence similarity.
* does not use the transformer architecture, however, it struggles with context-dependency on larger sentences.

<https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d#e586>

Pre-trained word embeddings lead to faster training and lower final training loss. Glove 4B, FastText Wiki 16B tokens



<https://towardsdatascience.com/from-pre-trained-word-embeddings-to-pre-trained-language-models-focus-on-bert-343815627598>

* **ULMFiT** → Transfer by **Fine Tuning**
* **ELMo** → Transfer by **Features Extraction**
* **BERT** → Transfer by **Attention Extraction**

Key datasets to evaluate:

* SICK Dataset (Marelli et. al., 2014) consists of 10,000 pairs of sentences and related judgements.
* STS 2014 (Agirre, et al., 2014) consists of 3,750 pairs and ratings from six linguistic domains.

Questions?

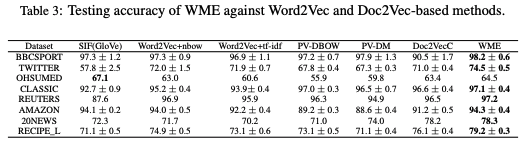
* Combining multipe methods?
* Does word order matter?
* What are the key things that define our dataset?
  + Unsupervised?
  + Multi-word tokenization needed?

The **big idea** is that you**represent documents as vectors of features**, and **compare documents by measuring the distance between these features**. There are **multiple ways** to **compute features** that capture the **semantics of documents**and multiple algorithms to capture **dependency structure of documents** to focus on meanings of documents. [link](https://medium.com/@adriensieg/text-similarities-da019229c894)

TDS embedding techniques article [here](https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d#fea6)

[Comparison of embedding methods on unsupervised data](https://www.aclweb.org/anthology/N16-1162.pdf)

Embedding Methods

* Bag of Words
  + Presented in [Harris, 1954], this method represents text as the bag ([multiset](https://en.wikipedia.org/wiki/Multiset)) of its words (losing grammar and ordering information). This is done by deciding on a set of n words that will form the vocabulary supported by the mapping, and assigning each word in the vocabulary a unique index. Then, each document is represented by a vector of length n, in which the i-th entry contains the number of occurrences of the word i in the document. One major downside of this approach is the non-linear dependency of the vocabulary size on the number of unique words, which can be very large for large corpora. Filtering techniques are commonly used to reduce the vocabulary size.
  + Term frequency (TF)
    - Good for text similarity
  + TF-IDF
    - Good for search query relevance
    - Good for classification documents as a whole, but word embeddings are good for identifying contextual content
    - Summary: This method re-weights the above word (or n-gram) frequency vectors with the inverse document frequency (IDF) of each word. The IDF of a word is simply the logarithm of the number of documents in the
* Word embeddings
  + Word2Vec
    - good for identifying contextual content
* GloVe
* Fast Text
* Fast Sent
* Latent Direct Allocation
  + Summary: LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word’s presence is attributable to one of the document’s topics. Topics are now characterized by distributions over words, while documents are distributions over topics. A main use case for LDA is unsupervised topic/community discovery, other cases include the use of the resulting latent topic space as an embedding space for the document corpus.
  + Basically makes intermediate layer of ‘topics’ (pets, news love) to represent a sentence or document.
* Neural Probabilistic Language Models
  + N-gram embeddings
    - [Mokolov et al 2013b](https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf): extend W2V’s skipgram to handle short phrases by treating 2-3 word phrases as individual tokens in W2V training. Less suitable for longer phrases. Bound not to generalize to unseen phrases as well as methods that follow it
    - Good TDS [article](https://towardsdatascience.com/word2vec-for-phrases-learning-embeddings-for-more-than-one-word-727b6cf723cf) explaining multi-word embeddings
  + Averaging word embeddings for a document into one document vector
    - Element-wise summation
      * [[Hill et al, 2016](https://www.aclweb.org/anthology/N16-1162)] compare a plethora of methods, including training CBOW and skip-gram word embeddings while optimizing for sentence representation (here using element-wise addition of word vectors).
        + Also do a big comparison
    - Siamese CBOW: Word embeddings for sentence representation
      * [paper](https://arxiv.org/pdf/1606.04640.pdf)
      * Summary: improves upon mere averaging of word vectors to make sentence vectors by
      * “Averaging the embeddings of words in a sentence has proven to be a surprisingly successful and efficient way of obtaining sentence embeddings. However, word embeddings trained with the methods currently available are not optimized for the task of sentence representation, and, thus, likely to be suboptimal. Siamese CBOW handles this problem by training word embeddings directly for the purpose of being averaged. The underlying neural network learns word embeddings by predicting, from a sentence representation, its surrounding sentences”
    - [[Arora et al, 2016](https://pdfs.semanticscholar.org/3fc9/7768dc0b36449ec377d6a4cad8827908d5b4.pdf)] [**SIF**](https://github.com/PrincetonML/SIF)
      * [**Github code**](file:///Users/richardkuzma/coding/NLP_projects/job_recommender_project/numpy,%20scipy,%20pickle,%20sklearn,%20theano%20and%20the%20lasagne%20library.) **for sentence2vec;** [**paper**](https://openreview.net/forum?id=SyK00v5xx) **links here**
      * Requirements: numpy, scipy, pickle, sklearn, theano and the lasagne library.
      * Finally, have further showed this approach to be a simple but tough-to-beat baseline when augmented with two small variations: (1) using a *smooth inverse frequency weighting scheme*, and (2) removing the common discourse component from word vectors; this component is found using PCA, and it is used as a correction term for the most frequent discourse, presumably related to syntax.
  + Word2Vec
  + Sentence2Vec
    - Sentence2vec as described in A SIMPLE BUT TOUGH TO BEAT BASELINE FOR SENTENCE EMBEDDING. [github](https://github.com/peter3125/sentence2vec) from Princeton SIF team
    - classic CBOW model of word2vec is both extended to include word n-grams and adapted to optimize the word (and n-grams) embeddings for the purpose of averaging them to yield document vectors [[Pagliardini et al, 2017](https://aclweb.org/anthology/N18-1049" \t "_blank)] and [[Gupta et al, 2019](https://www.aclweb.org/anthology/N19-1098)]
    - process of input subsampling is removed, considering the entire sentence as context instead. This means both that **(a)** the use of frequent word subsampling is discarded — so as not to prevent the generation of n-grams features — and **(b)** the dynamic context windows used by word2vec are made away with: the entire sentence is considered as the context window, instead of sampling the context window size for each subsampled word uniformly between 1 and the length of the current sentence.
    - Another way to think of sent2vec is as an unsupervised version of fastText (see Figure 6), where the entire sentence is the context and possible class labels are all vocabulary words.
  + Paragraph vectors (doc2vec)
    - doc2vec, this method, presented in [[Le & Mikolov, 2014](https://cs.stanford.edu/~quocle/paragraph_vector.pdf)] is perhaps the first attempt to generalize word2vec to work with word sequences. The authors introduce two variants of the paragraph vectors model: Distributed Memory and Distributed Bag-of-Words.
    - [Le & Mikolov, 2014] demonstrated the use of paragraph vectors on several text classification and sentiment analysis tasks
    - [Dai et al, 2015] examined it in the context of document similarity tasks and [Lau & Baldwin, 2016] benchmarked it against a forum question duplication task and the [Semantic Textual Similarity (STS) SemEval](http://ixa2.si.ehu.es/stswiki/index.php/Main_Page) shared task
    - [Lau & Baldwin, 2016] benchmarked it against a forum question duplication task and the [Semantic Textual Similarity (STS) SemEval](http://ixa2.si.ehu.es/stswiki/index.php/Main_Page) shared task. [including code](https://github.com/jhlau/doc2vec))
    - Implementations
      * Python implementation with Gensim ackage
      * Pytorch implementation
      * Lau & Baldwin 2017 supplied their code
  + Doc2VecC (doc2vec corruption)
    - in contrast to paragraph vectors, which directly learns a unique vector for each document, Doc2VecC represents each document as an average of the embeddings of words randomly sampled from the document
    - Additionally, the authors choose to corrupt the original document by randomly removing a significant portion of words, representing the document by averaging only the embeddings of the remaining words. This corruption mechanism allows a speedup during training as it significantly reduces the number of parameters to update in back propagation. The authors also show how it introduces a special form of regularization, which they believe results in the observed performance improvement, benchmarked on a sentiment analysis task, a document classification task and a semantic relatedness task versus a plethora of state-of-the-art document embedding techniques.
      * C-based implementation of the method and code to reproduce the experiments in the paper can be found in [a public Github repository](https://github.com/mchen24/iclr2017).
    - Skip-thought vectors
      * Presented in [[Kiros et al, 2015](https://arxiv.org/abs/1506.06726" \t "_blank)], this is another early attempt to generalize word2vec, and was published with [an official pure Python implementation](https://github.com/ryankiros/skip-thoughts) (and recently also boasting implementations for [PyTorch](https://github.com/sanyam5/skip-thoughts" \t "_blank) and [TensorFlow](https://github.com/tensorflow/models/tree/master/research/skip_thoughts)). extends word2vec — specifically the skip-gram architecture — in another intuitive way: the base unit is now sentences, and an encoded sentence is used to predict the sentences around it. The vector representations are learned using an encoder-decoder model trained on the above task; the authors use an RNN encoder with GRU activations and RNN decoders with a conditional GRU. Two different decoders are trained for previous and next sentences.
  + FastSent
    - [[Hill et al, 2016](https://www.aclweb.org/anthology/N16-1162)] propose a significantly simpler variation on the skip-thoughts model; FastSent is a simple additive (log-bilinear) sentence model designed to exploit the same signal, but at a **much lower computational expense.** [**an official Python implementation**](https://github.com/fh295/SentenceRepresentation)**.**
  + Quick-thought vectors
    - [[Logeswaran & Lee, 2018](https://arxiv.org/pdf/1803.02893.pdf" \t "_blank)] reformulate the document embedding task — the problem of predicting the context in which a sentence appears — as a supervised classification problem. Given an input sentence, it is encoded by an encoder (RNNs, in this case), but instead of generating the target sentence, the model chooses the correct target sentence from a set of candidate sentences; the candidate set is built from both valid context sentences (ground truth) and many other non-context sentences. [an official Python implementation](https://github.com/lajanugen/S2V).
  + Word Mover’s Distance (WMD)
    - [[Kushner et al, 2015](http://proceedings.mlr.press/v37/kusnerb15.pdf)] presented Word Mover’s Distance (WMD); this measures the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to “travel” **in the embedding space** to reach the embedded words of another document
    - High computational cost O(L^3 log(L))
  + Word Mover’s Embedding (WME),
    - [[Wu et al, 2018b](https://arxiv.org/pdf/1811.01713v1.pdf)]. [An official C-based, Python-wrapped implementation is provided](https://github.com/IBM/WordMoversEmbeddings). WME builds on three components to learn continuous vector representations for texts of varying lengths:
      * The ability to learn high-quality word embedding in an unsupervised manner (e.g., using word2vec).
      * The ability to construct a distance measure for documents based on said embeddings using Word Mover’s Distance (WMD).
      * The ability to derive positive-definite kernel from a given distance function using D2KE.
      * This framework is extensible, since its two building blocks, word2vec and WMD, can be replaced by other techniques such as GloVe (for word embeddings)
      * authors evaluate WME on 9 real-world text classification tasks and 22 textual similarity tasks, and demonstrate that it consistently matches, and sometimes even outperforms, other state-of-the art techniques.
      * 
  + Sentence-BERT
    - [[Reimers & Gurevych, 2019](https://arxiv.org/pdf/1908.10084.pdf)] and accompanied by [a Python implementation](https://github.com/UKPLab/sentence-transformers), aims to adapt the BERT architecture by using siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity

Similarity (distance?) Calculations

* Jacard similarity
* Cosine
  + Requires vectors
    - Term frequency (TF)
    - TF-IDF
* Euclidean distance
  + Requires vectors
* Word Mover’s Distance

Pre-trained Weights

* **Averaging word vectors is a strong baseline**, so a good idea is to start your quest for good document embeddings by focusing on generating very good word vectors, and simply averaging them at first. Undoubtedly, much of the power of document embeddings comes from the word vectors upon which they are built, and I think it is safe to say there is a significant delta of information to optimize in that layer before moving forwards. You can try different pre-trained word embeddings, exploring which source domains and which methods (e.g. word2vec vs GloVe vs BERT vs ELMo) capture the type of information you need in a better way. Then, extending this slightly by trying different summarization operators or other tricks (like those in [[Arora et al, 2016](https://pdfs.semanticscholar.org/3fc9/7768dc0b36449ec377d6a4cad8827908d5b4.pdf)]) might prove to be enough.

Performance evaluation

* lean methods: [sent2vec](https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d#e3d4) and [FastSent](https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d" \l "e6e8),
* In contrast, the real-time vector representation inference required for each sentence when using doc2vec might prove costly given application constraints
* [SentEval, an evaluation toolkit for sentence representations](https://github.com/facebookresearch/SentEval) presented in [[Conneau & Kiela, 2018](https://arxiv.org/pdf/1803.05449.pdf" \t "_blank)], is a tool worth mentioning in this context.

Open-source benchmarking

* **Open-source implementations are abundant**, so benchmarking different approaches against your task might be feasible.

[This](https://medium.com/@adriensieg/text-similarities-da019229c894) medium post’s rankings

* **Jaccard Similarity**☹☹☹
* Different embeddings**+** **K-means**☹☹
* Different embeddings**+** **Cosine Similarity**☹
* **Word2Vec** **+** **Smooth Inverse Frequency** + **Cosine Similarity**😊
  + [**SIF**](https://github.com/PrincetonML/SIF)
    - [**Github code**](numpy,%20scipy,%20pickle,%20sklearn,%20theano%20and%20the%20lasagne%20library.)**;** [**paper**](https://openreview.net/forum?id=SyK00v5xx) **links here**
    - Requirements: numpy, scipy, pickle, sklearn, theano and the lasagne library.
    - Finally, [[Arora et al, 2016](https://pdfs.semanticscholar.org/3fc9/7768dc0b36449ec377d6a4cad8827908d5b4.pdf)] have further showed this approach to be a simple but tough-to-beat baseline when augmented with two small variations: (1) using a *smooth inverse frequency weighting scheme*, and (2) removing the common discourse component from word vectors; this component is found using PCA, and it is used as a correction term for the most frequent discourse, presumably related to syntax.
* Different embeddings**+LSI** **+** **Cosine Similarity**☹
* Different embeddings**+ LDA** + **Jensen-Shannon distance**😊
  + Latent Direct Allocation
    - Summary: LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. For example, if observations are words collected into documents, it posits that each document is a mixture of a small number of topics and that each word’s presence is attributable to one of the document’s topics. Topics are now characterized by distributions over words, while documents are distributions over topics. A main use case for LDA is unsupervised topic/community discovery, other cases include the use of the resulting latent topic space as an embedding space for the document corpus.
    - Paper
    - Github
* Different embeddings**+** **Word Mover Distance** 😊😊
* Different embeddings**+ Variational Auto Encoder (VAE)**😊 😊
* Different embeddings**+ Universal sentence encoder**😊😊
* Different embeddings**+ Siamese Manhattan LSTM**😊😊😊
* BERT embeddings + Cosine Similarity ❤
* **Knowledge-based Measures** ❤

<https://medium.com/@adriensieg/text-similarities-da019229c894>

* **BoW**or**TF-IDF** is good for classification documents as a whole, but **word embeddings** is good for identifying contextual content

Comparison of Embedding Methods:

* <https://towardsdatascience.com/beyond-word-embeddings-part-1-an-overview-of-neural-nlp-milestones-82b97a47977f>

<https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec> and <https://cai.tools.sap/blog/glove-and-fasttext-two-popular-word-vector-models-in-nlp/>

Intuitively, a good language model should capture significant information about not only individual words but also relationships between words based on their positions in sentences relative to each other.

For words to be processed by machine learning models, they need some form of numeric representation that models can use in their calculation. Word2Vec showed that we can use a vector (a list of numbers) to properly represent words in a way that captures semantic or meaning-related relationships (e.g. the ability to tell if words are similar, or opposites, or that a pair of words like “Stockholm” and “Sweden” have the same relationship between them as “Cairo” and “Egypt” have between them) as well as syntactic, or grammar-based, relationships (e.g. the relationship between “had” and “has” is the same as that between “was” and “is”).

The field quickly realized it’s a great idea to use embeddings that were pre-trained on vast amounts of text data instead of training them alongside the model on what was frequently a small dataset

* + Word2Vec
    - Continuous bag of words – order of context words doesn’t influence prediction. Averages context words for predictio. Faster.
    - Continuous Skip-gram – weighs nearby context words more heavily than more distant context words. Order not captured, but context vectors weighed independently. Better for infrequent words
    - Fails to provide vector representation for words not in model dictionary
  + GloVe – takes advantage of –not just local— context. GloVe vectors faster to train but neither GloVe nor W2V is definitely better
    - Fails to provide vector representation for words not in model dictionary
  + FastText – builds on Word2Vec by learning vector representations for each word and n-grams found within each word. Higher computation cost than Word2Vec but more accurate by a number of measures
    - Instead of learning vectors for words directly, fastText represents each word as an n-gram of characters. So, for example, take the word, “*artificial*” with n=3, the fastText representation of this word is <*ar, art, rti, tif, ifi, fic, ici, ial, al*>
    - This helps capture the meaning of shorter words and allows the embeddings to understand suffixes and prefixes
    - fastText works well with rare words. So even if a word wasn’t seen during training, it can be broken down into n-grams to get its embeddings.
  + ELMo
    - *contextualized* word-embeddings were born
    - model generates embeddings for a word based on the context it appears thus generating slightly different embeddings for each of its occurrence
    - “play” as in ‘to play’ (verb) or ‘broadway play’ (theater production) has the same embedding in GloVe, FastText, Word2Vec. Not so in ELMo, it can tell the difference.
    - promising implications for preforming transfer learning on out of domain datasets
    - ELMo provided a significant step towards pre-training in the context of NLP. The ELMo LSTM would be trained on a massive dataset in the language of our dataset, and then we can use it as a component in other models that need to handle language.
  + BERT
    - [BERT](https://arxiv.org/abs/1810.04805) is different from ELMo and company primarily because it targets a different training objective. The main limitation of the earlier works is an inability to take into account both left and right contexts of the target word, since the language model objective is generated from left to right, adding successive words to a sentence. Even ELMo, which uses a bidirectional LSTM, simply concatenated the left-to-right and right-to-left information, meaning that the representation couldn’t take advantage of both left and right contexts simultaneously.
    - BERT replaces language modeling with a modified objective they called “masked language modeling”. In this model, words in a sentence are randomly erased and replaced with a special token (“masked”) with some small probability, 15%. Then, a Transformer is used to generate a prediction for the masked word based on the unmasked words surrounding it, both to the left and right.
    - Computationally expensive…64GB RAM on TPU or GPU. Out of memory errors with 16GB
  + GPT
    - Transformer instead of bi-LSTM for GPT
  + ULMFit
    - Discriminative fine-tuning
    - ULM-FiT introduced methods to effectively utilize a lot of what the model learns during pre-training – more than just embeddings, and more than contextualized embeddings. ULM-FiT introduced a language model and a process to effectively fine-tune that language model for various tasks.
    - NLP finally had a way to do transfer learning probably as well as Computer Vision could.
  + Transformers vs LSTMs

SIF + Glove for sentence/document embeddings 🡪 <https://openreview.net/pdf?id=SyK00v5xx>

A interesting feature of our method is that it ignores the word order. This is in contrast to that RNN’s and LSTM’s can potentially take advantage of the word order. The fact that our method achieves better or comparable performance on these benchmarks raise the following question: is word order not important in these benchmarks? We conducted an experiment suggesting that word order does play some role. We trained and tested RNN/LSTM on the supervised tasks where the words in each sentence are randomly shuffled, and the results are reported in Table 3.8 It can be observed that the performance drops noticeably. Thus our method —which ignores word order—must be much better at exploiting the semantics than RNN’s and LSTM’s. An interesting future direction is to explore if some ensemble idea can combine the advantages of both approaches.

<https://github.com/PrincetonML/SIF>

<https://jalammar.github.io/illustrated-bert/>

* <https://www.quora.com/What-are-the-main-differences-between-the-word-embeddings-of-ELMo-BERT-Word2vec-and-GloVe>
* <https://www.reddit.com/r/MachineLearning/comments/aptwxm/d_what_are_the_main_differences_between_the_word/>
* Bert <https://arxiv.org/abs/1810.04805>
  + <https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b>
* <https://medium.com/@Intellica.AI/comparison-of-different-word-embeddings-on-text-similarity-a-use-case-in-nlp-e83e08469c1c>
* Systemic approach to select pre-trained embeddings for downstream task: <https://arxiv.org/pdf/1903.04433.pdf>

Embedding strategies for specialized domains: <https://www.aclweb.org/anthology/P19-2041.pdf>

* General thoughts:
  + Compare embedding types
  + What is SOTA embedding method?
  + How to use pre-trained embeddings?
  + How to use embedding method training on your data? Costs?
  + Off-the-shelf embeddings pro/con
  + Off-the-shelf + static embeddings on small in-domain corpus
* d
* d
* d
* d
* d
* d

Key paper: An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation, [arXiv](https://arxiv.org/pdf/1607.05368.pdf), [github](https://github.com/jhlau/doc2vec)

* Reqs
  + Supports python2 only
  + Uses personal forked Gensim library
* Application results
  + Used to detect duplicate questions from stackoverflow forums
  + Semantic Textual Similarity (STS) of sentences where 0 is no similarity and 5 is semantic equivalence
    - Tokenize using Stanford’s coreNLP: split, lower, tokenize
  + Dbow better than dmpv
* Referenced by <https://github.com/tsandefer/dsi_capstone_2> implementation

Skill clustering paper (R)

* <https://nycdatascience.com/blog/student-works/using-machine-learning-measure-job-skill-similarities/> And <https://github.com/brettamdur/skillSimilarities>
* Word2vec embedding
* K means clustering for skills
* Hierarchical clustering with a dendogram
* Latent Dirichlet Allocation (LDA) + visualization

Comparing Similarity of Popular Books

* <https://www.kaggle.com/currie32/comparing-books-with-word2vec-and-doc2vec/data#Doc2Vec>
* <https://github.com/Currie32/Comparing-Books>
* More Project Gutenberg texts (check pre-processing)
  + <https://web.eecs.umich.edu/~lahiri/gutenberg_dataset.html> (3k)
  + <https://github.com/pgcorpus/gutenberg> (50k)

Comparing similarity of Quora questions

* <https://www.kaggle.com/currie32/predicting-similarity-tfidfvectorizer-doc2vec/comments>
* <https://github.com/Currie32/Predicting-Similar-Questions>

Text clusterization using python and doc2vec

* <https://medium.com/@ermolushka/text-clusterization-using-python-and-doc2vec-8c499668fa61>

Multi-class classification using doc2vec (analogous to job type or branch classification?)

* <https://towardsdatascience.com/implementing-multi-class-text-classification-with-doc2vec-df7c3812824d>

Lit review of document embedding techniques

* <https://towardsdatascience.com/document-embedding-techniques-fed3e7a6a25d#0242>

Research:

## Python version

## Packages (good comparison [here](https://medium.com/activewizards-machine-learning-company/comparison-of-top-6-python-nlp-libraries-c4ce160237eb))

* + PDF to text extractor
    - PDFminer
    - PyPDF2
  + NLP Libraries
    - [Gensim](https://github.com/RaRe-Technologies/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](https://www.python.org/), tested with versions 2.7, 3.5, 3.6 and 3.7.
        + [NumPy](https://www.numpy.org/) for number crunching.
        + [smart\_open](https://pypi.org/project/smart_open/) for transparently opening files on remote storages or compressed files.
      * [Youtube video](https://www.youtube.com/watch?v=Otde6VGvhWM&list=PLZoTAELRMXVMdJ5sqbCK2LiM0HhQVWNzm&index=7) for word2vec w/Gensim library and NLTK
    - [Stanza](https://github.com/stanfordnlp/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](https://www.nltk.org/))
    - [SpaCy](https://spacy.io/)
      * 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](https://towardsdatascience.com/do-the-keywords-in-your-resume-aptly-represent-what-type-of-data-scientist-you-are-59134105ba0d))
  + Labeling entities / unlabeled key phrases
    - [Snorkel](https://github.com/snorkel-team/snorkel)

## Data

* + Transfer learning
    - Penn Treebank (PTB) (Mikolov)
      * 2500 stories from WSJ [link](https://catalog.ldc.upenn.edu/LDC99T42)
      * 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?)

## 

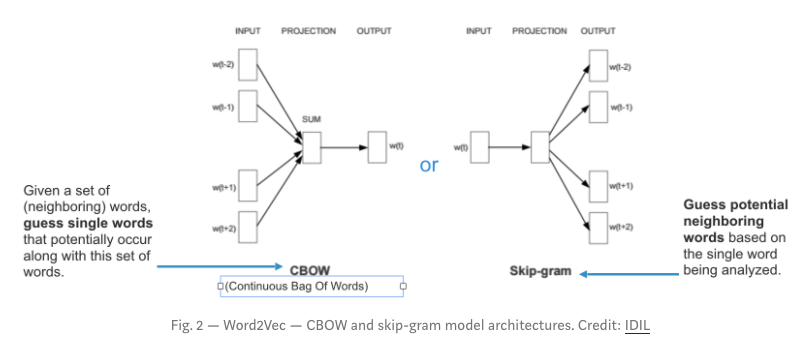
## Techniques

* + **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](https://www.youtube.com/watch?v=pzyIWCelt_E))
      * 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](https://realpython.com/build-recommendation-engine-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](https://towardsdatascience.com/word-embeddings-exploration-explanation-and-exploitation-with-code-in-python-5dac99d5d795)
* KD nuggets [pre-processing](https://www.kdnuggets.com/2018/03/text-data-preprocessing-walkthrough-python.html)
* Medium post on phrase-matching in resumes using Spacy package ([use case here](https://towardsdatascience.com/do-the-keywords-in-your-resume-aptly-represent-what-type-of-data-scientist-you-are-59134105ba0d))
* Automated Resume Screening System project ([github](https://github.com/JAIJANYANI/Automated-Resume-Screening-System))
* Word2Vec sentiment analysis tweets [implementation](https://www.ahmedbesbes.com/blog/sentiment-analysis-with-keras-and-word-2-vec)
* Resume job-matching project ([github](https://github.com/binoydutt/Resume-Job-Description-Matching))
  + 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](https://towardsdatascience.com/an-implementation-guide-to-word2vec-using-numpy-and-google-sheets-13445eebd281)
  + 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
    - 
* Doc2vec gensim implementation: [gensim](https://radimrehurek.com/gensim/models/doc2vec.html)

### Papers

* Mikolov et al., SEP2013, Efficient Estimation of Word Representations in Vector Space, [arXiv](https://arxiv.org/pdf/1301.3781.pdf)
* Mikolov et al., OCT2013, Distributed Representations of Words and Phrases and their Compositionality, [arXiv](https://arxiv.org/pdf/1310.4546.pdf)
* Le and Mikolov, MAY2014, Distributed Representations of Sentences and Documents, [arXiv](https://arxiv.org/pdf/1405.4053.pdf)
* Schmitt et al, Matching Jobs and Resumes: a Deep Collaborative Filtering Task, [EasyChair](https://easychair.org/publications/open/Jwh)
* Stanford NLP Stanza, [arXiv](https://arxiv.org/pdf/2003.07082.pdf)
* An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation, [arXiv](https://arxiv.org/pdf/1607.05368.pdf)
  + Provide public code w/doc2vec pretrained on large corpus
  + Identify and compare to other embeddings:
    - skip-thought (Kiros et al., 2015)
    - paragram-phrase (pp: Wieting et al. (2016))
      * PP outperforms Lau & Baldwin’s d2v on STS, not on finding duplicates
      * D2V improves with pretraining on wikipedia
      * Given that pp is based on word vector averaging, these observations support the conclusion that vector averaging methods works best for shorter documents, while dbow handles longer documents better.
* TOWARDS UNIVERSAL PARAPHRASTIC SENTENCE EMBEDDINGS [arXiv](https://arxiv.org/pdf/1511.08198.pdf)

### Notes on Topics

Word embeddings exploration explanation ([towardsdatascience](https://towardsdatascience.com/word-embeddings-exploration-explanation-and-exploitation-with-code-in-python-5dac99d5d795))

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
   1. Note, they don’t remove **stop words** like ‘and’ and ‘is’
   2. [KDnuggets](https://www.kdnuggets.com/2018/03/text-data-preprocessing-walkthrough-python.html) article on text pre-processing
   3. Gensim library also provides a function to perform simple text preprocessing using [gensim.utils.simple\_preprocess](https://radimrehurek.com/gensim/utils.html#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
   1. Window\_size = 2
   2. N = 10 (number of dimensions of word embeddings aka size of hidden layer. It’s your vocabulary size)
   3. Epochs = 50
   4. 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
   1. One-hot encoding = Dummy vars for words
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