Representation Learning for Text Data

Concepts & Techniques

Dipanjan (DJ) Sarkar

Session Agenda

Representation Learning - What & Why

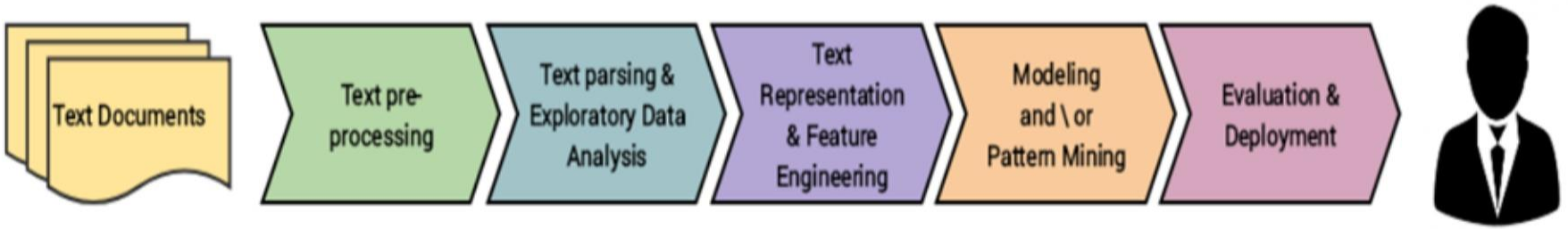
Text Representation Methodologies

Traditional Text Representation Models

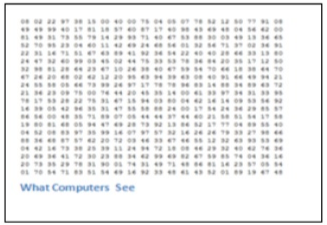
Representation Learning with Word Embeddings

Representation Learning
What & Why

NLP Workflow

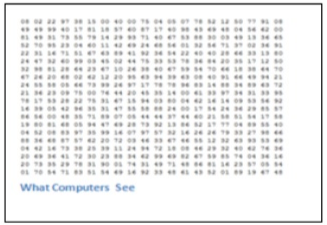


Why Representation Learning for Text



- Machine does not understand text directly
- We need a numeric representation to make any models train and learn
- Unlike images (RGB matrix), for text there is no obvious or direct mapping to numbers
- Essential part of the NLP pipeline before downstream applications

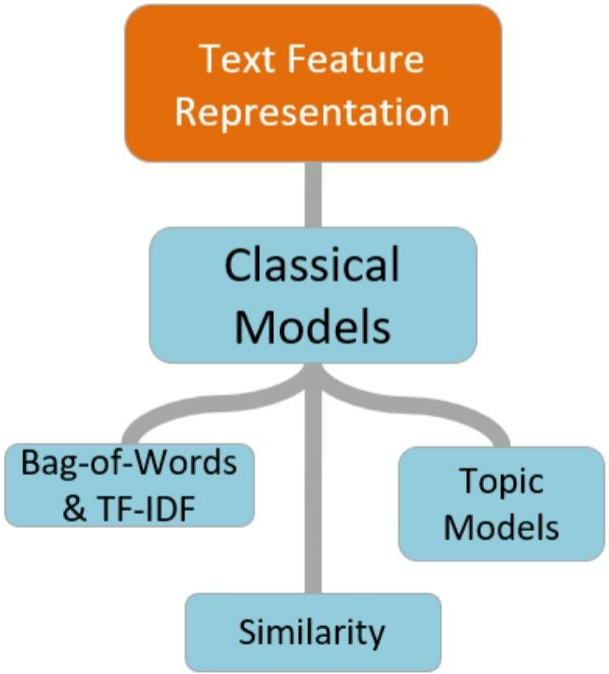
What & Why of Representation Learning

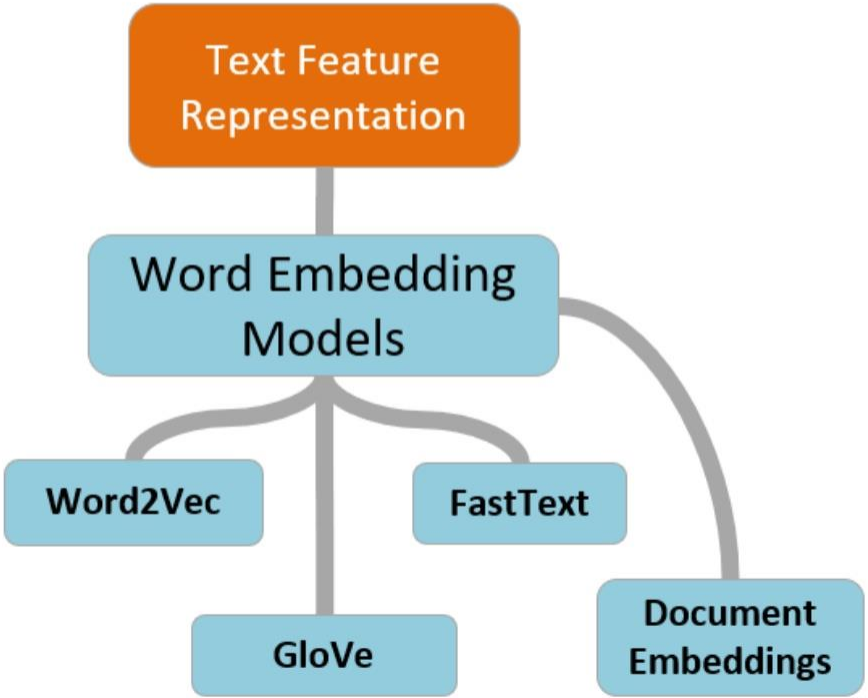


- Representation learning is a set of techniques that learn features from text by learning
- Transformation of the raw data into an effective representation that can used in downstream tasks
- Get rid of “hand-crafted” features and representation
- Unsupervised feature learning - eliminates the need of manual feature engineering

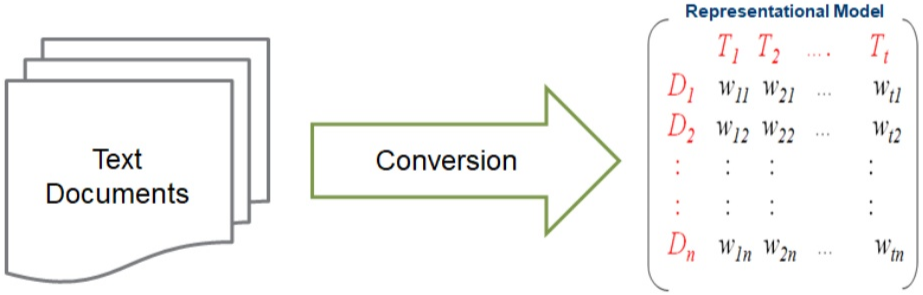
Text Representation Methodologies

Text Representation Models





Text Representation Models



- ML\DL models at heart are mathematical functions and cannot understand unstructured text
- Hence we need to convert text into some numeric representations which can be understood by machines
- Commonly known as Vector Space Models where text is converted to numeric vectors
  - Bag of Words, TF-IDF
  - Topic Models
  - Similarity
  - Word Embeddings - Word2Vec, GloVe, FastText etc.

Traditional Text Representation Models

Traditional Text Representation Models

1 Bag of Words

- Each document is represented by a vector (bag) of words
- Depicts the number of times each word occurs in that document

2 Bag of N-grams

- Same as the Bag of Words model
- Instead of words, we also have n-grams and counts for them in the vector

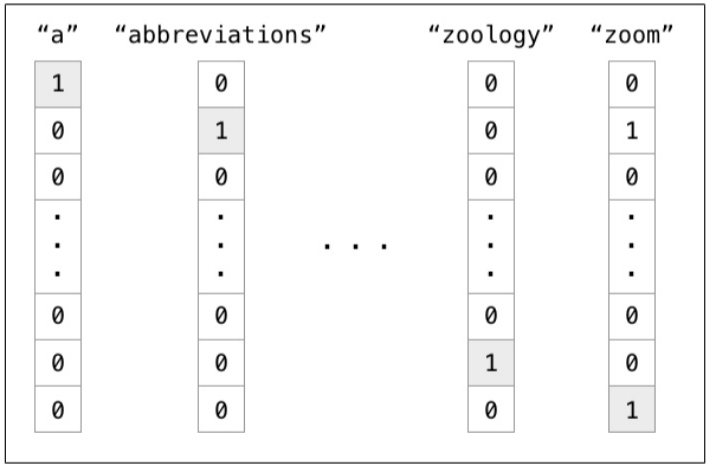
3 TF-IDF

- Similar to the basic Bag of Words (TF) model
- Normalizes counts using the inverse document frequency (IDF) to downplay effect of frequently occuring words

4 Document Similarity

- Derived attribute \ feature from bag of words based features
- Assign scores to each document w.r.t how similar it is to other documents (based on their BOW vectors)

Classical Representation Methods - One-hot encoding



- Assign every word to a position identifier (ID)
- Perform one-hot encoding of these words based on IDs
- Typically used for categorical data
- Poor representation of words - no context or semantics
- Unable to scale even with medium sized vocabulary

Classical Representation Methods - Bag of Words \ TF-IDF

Sentence 1: "The cat sat on the mat"
Sentence 2: "The dog and the cat and the mat"

Vocab = { the, cat, sat, on, mat, dog, and }

Sentence 1: { 2, 1, 1, 1, 1, 0, 0, 0 }
Sentence 2 : { 3, 1, 0, 0, 1, 1, 1, 2}

- Just takes into account word frequencies or normalized word frequencies
- No notion of word ordering is considered
- Contextual or semantic representations are not captured
- Sometimes can be too simplistic

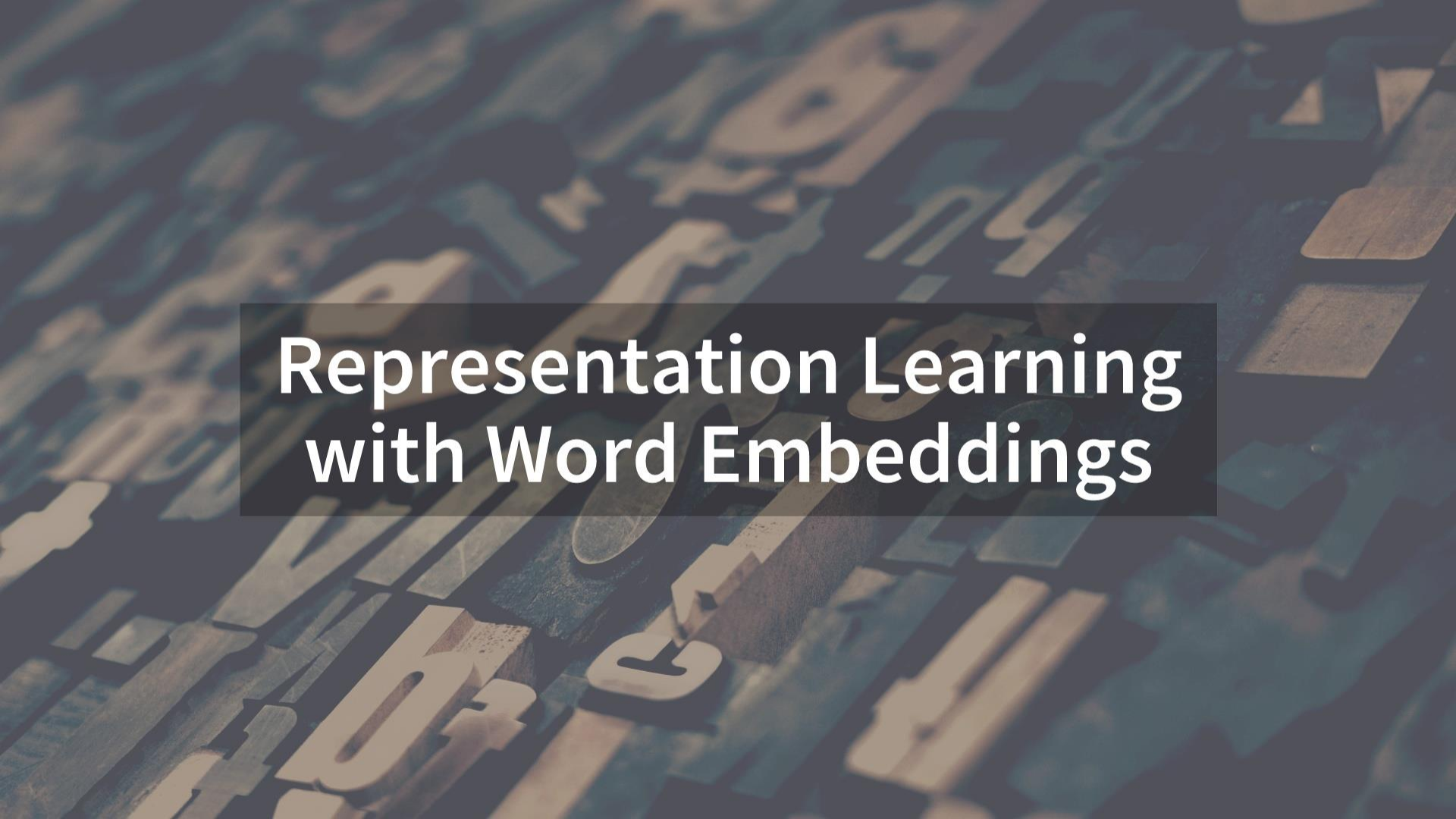
Classical Representation Methods - Bag of N-grams

Sentence 1: "The cat sat on the mat"
Sentence 2: "The dog and the cat and the mat"

Vocab = { the cat, cat sat, sat on, on the, the mat,
the dog, dog and, and the, cat and, and the}

Sentence 1: { 1, 1, 1, 1, 1, 0, 0, 0, 0, 0}
Sentence 2 : { 1, 0, 0, 0, 0, 1, 1, 1, 1, 1}

- Similar to Bag of Words
- Tries to incorporate word ordering to some extent
- Contextual or semantic representations are not captured
- Unable to capture representations based on longer sequences

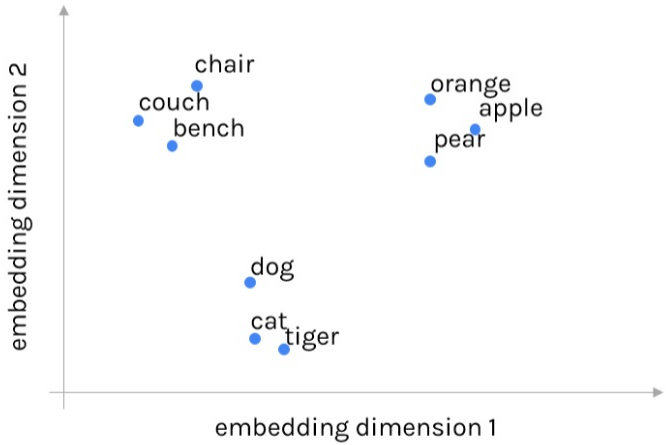

Representation Learning
with Word Embeddings

Why Word Embeddings?

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

- Use unsupervised or semi-supervised learning to automatically learn numeric representations
- Represent each word with a fixed sized vector of numbers - this is known as a word embedding
- A word is known by the company it keeps!
- Various concepts or dimensions of a word are represented by this fixed-width dense vector

More about word embeddings



- **Map linguistic units into a continuous vector space with a lower dimension (dense representation)**

  Linguistic unit can be - word, characters, n-grams, sentences, paragraphs, documents

- **The meaning of the linguistic unit is represented by the multi-dimensional numeric vector (embedding)**
- **Based on the distributional hypothesis - linguistic units with similar distributions have similar meanings**
- **The distributional property is usually induced from document or context or textual vicinity (like a sliding window)**

Word Embedding representation properties

$$\text{good} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

- Embeddings are compact, dense and low dimensional representations
- Build efficient dense representations using co-occurences and context
- Each single component of the embedding vector representation does not have any meaning of its own
- Meaning of a word is smeared across all dimensions of its embedding
- The interpretable features (word contexts) are hidden and distributed among uninterpretable embedding components

Word Embedding Methodologies

1 Word2Vec

- Generates high quality dense vector representations of words
- Looks at each word and their surrounding words to generate representations

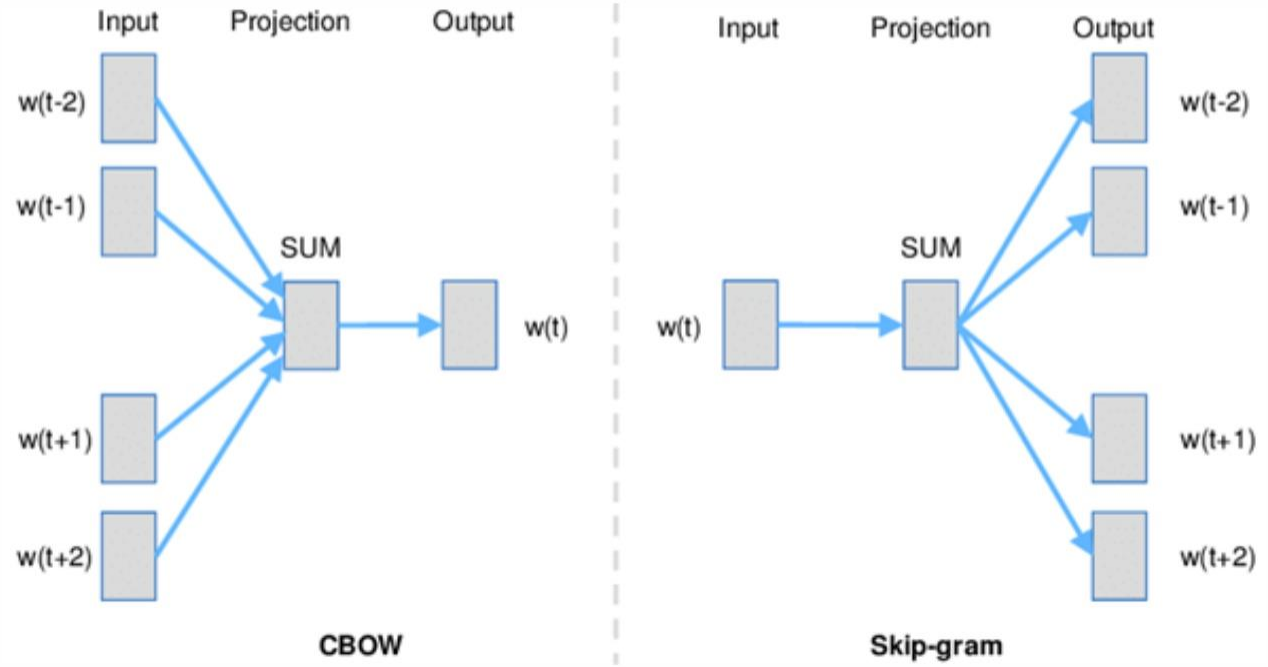
2 GloVe

- Unsupervised learning model which can be used to obtain dense word vectors
- Considering **the Word-Context (WC) matrix**, **Word-Feature (WF) matrix** and **Feature-Context (FC) matrix**, we try to factorize **WC = WF x FC**

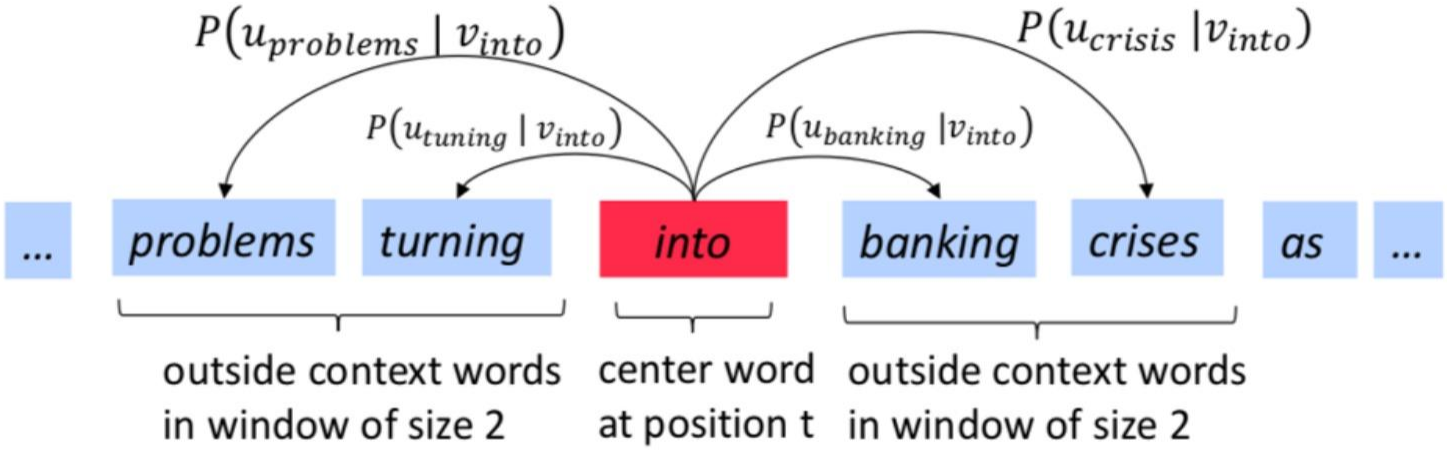
3 FastText

- Considers each word as a Bag of Character n-grams to generate vector representations
- Taking the word **where** and **n=3** (tri-grams) as an example, it will be represented by the character n-grams: **<wh**, **whe**, **her**, **ere**, **re>** and the special sequence **< where >**

Word2Vec - Model Architectures



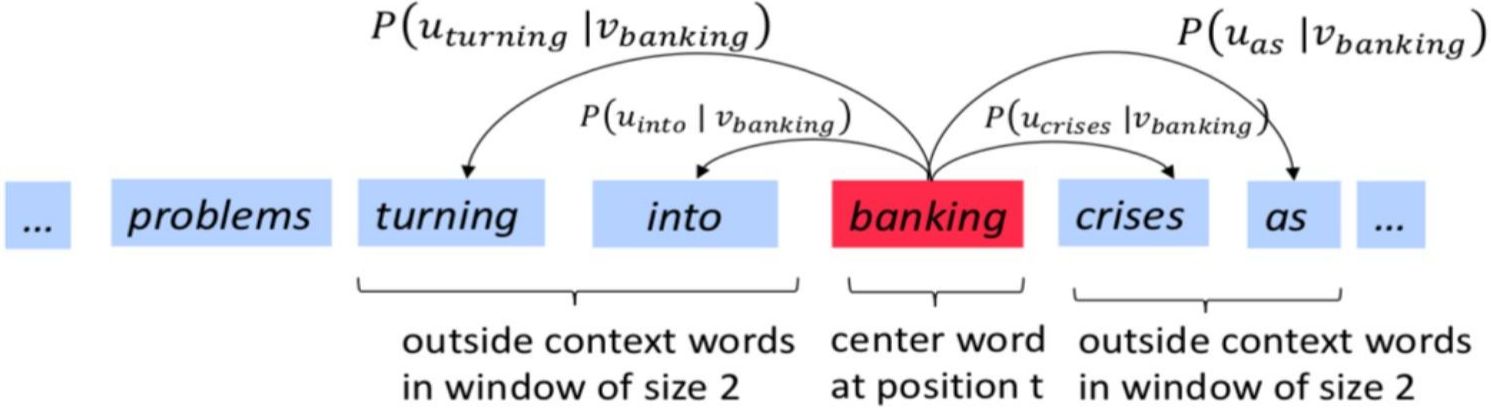
Word2Vec - Training Principles



- Example windows and processes for computing $\mathbf{P(w_{t+j} \mid w_t)}$

- $\mathbf{P}(\boldsymbol{u}_{\mathbf{problems}} \mid \boldsymbol{v}_{\mathbf{into}})$: Probability of finding the word 'problems' in the search window, given that the center word is 'into'

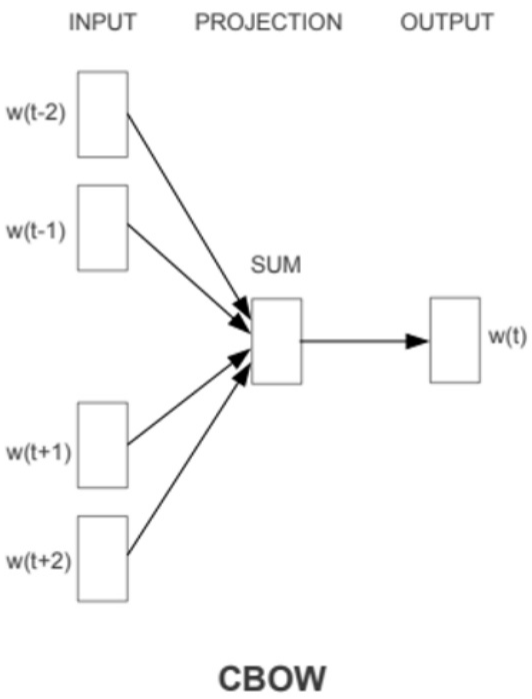
Word2Vec - Training Principles

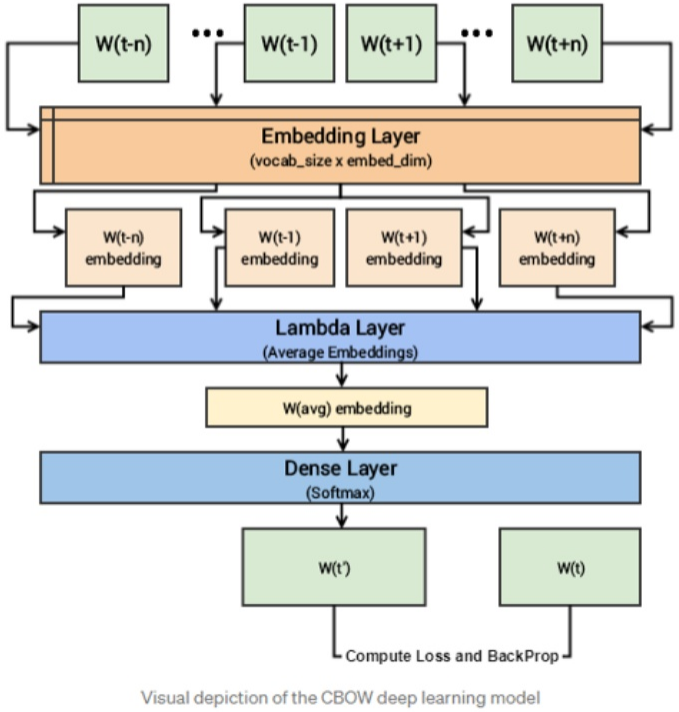


- Example windows and processes for computing $\mathbf{P(w_{t+j} \mid w_t)}$

- $\mathbf{P(u_{turning} \mid v_{banking})}$: Probability of finding the word 'turning' in the search window, given that the center word is 'banking'

Word2Vec - CBOW Architecture



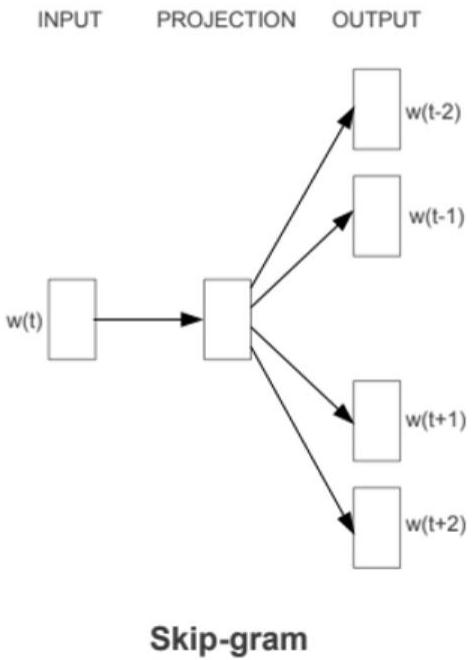


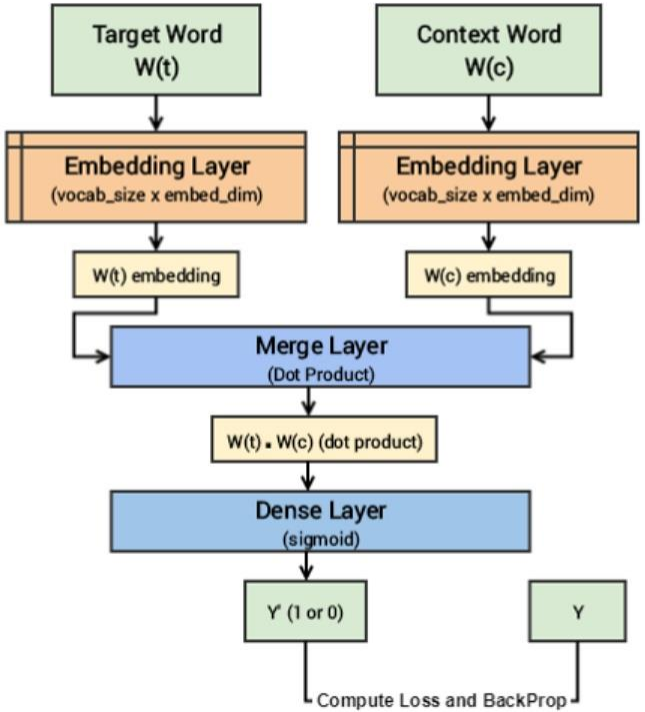
Visual depiction of the CBOW deep learning model

Word2Vec - CBOW Architecture

- Our inputs will be our context words which are passed to an embedding layer (initialized with random weights)
- The word embeddings are propagated to a lambda layer where we average out the word embeddings
- Called CBOW because we don't really consider the order or sequence in the context words when averaged
- We pass this averaged context embedding to a dense softmax layer which predicts our target word
- We match this with the actual target word, compute the loss by leveraging the categorical_crossentropy loss and perform backpropagation

Word2Vec - Skipgram Architecture





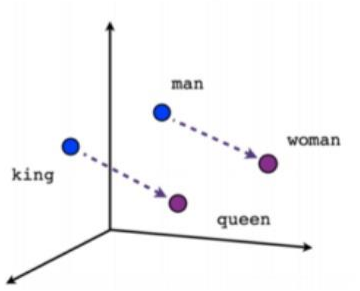
Visual depiction of the Skip-gram deep learning model

Word2Vec - Skipgram Architecture

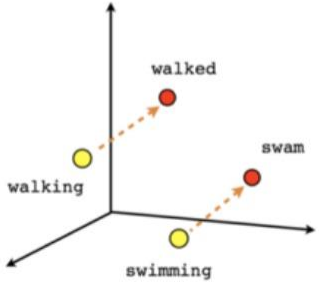
- We feed our skip-gram model pairs of ***(X, Y)*** where ***X*** is our input and ***Y*** is our label.
- We do this by using ***[(target, context), 1]*** pairs as positive input samples where ***target*** is our word of interest and ***context*** is a context word occurring near the target word and the positive label **1** indicates this is a contextually relevant pair.
- We also feed in ***[(target, random), 0]*** pairs as negative input samples where ***target*** is again our word of interest but ***random*** is just a randomly selected word from our vocabulary which has no context or association with our target word. Hence the negative label **0** indicates this is a contextually irrelevant pair (negative sampling)
- We do this so that the model can then learn which pairs of words are contextually relevant and which are not and generate similar embeddings for semantically similar words.

Word2Vec - General Model Principles

- A neural network is used to train all words in our dictionary to get vectors (embeddings)
- We do have a **forward pass** and **back propagation** to get to the final word embeddings after convergence
- A word embedding exists for every word in the vocabulary (obtained from corpus)
- We get similar vectors for similar words



Male-Female



Verb tense

Word2Vec - Model Settings

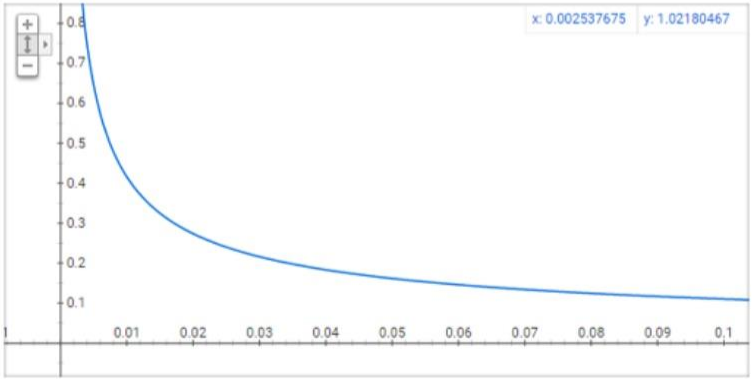
Sampling Rate: Subsampling and removing frequent words

wi is the word, **z(wi)** is the fraction of the total words in the corpus that are that word. For example, if the word "peanut" occurs 1,000 times in a 1-billion-word corpus, then **z('peanut') = 1E-6**.

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

- $P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.
  - This means that only words which represent more than 0.26% of the total words will be subsampled.
- $P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.
- $P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.
  - That is, if the corpus consisted entirely of word w_i, which of course is ridiculous.

Graph for (sqrt(x/0.001)+1)*0.001/x



Source: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

Word2Vec - Model Settings

- The size of the embedding dimension itself is also a hyperparameter
  - Shorter corpora can be represented with smaller dimension embeddings (e.g., 50-100)
  - A standard dimension is 300 (available in most pre-trained embeddings)

- The size of the context window depends on the application
  - Shorter windows (1-5): Capture more syntactic information
  - Longer windows (5-10): Can capture more semantic meaning and longer dependency-based relationships

Word2Vec - Model Settings

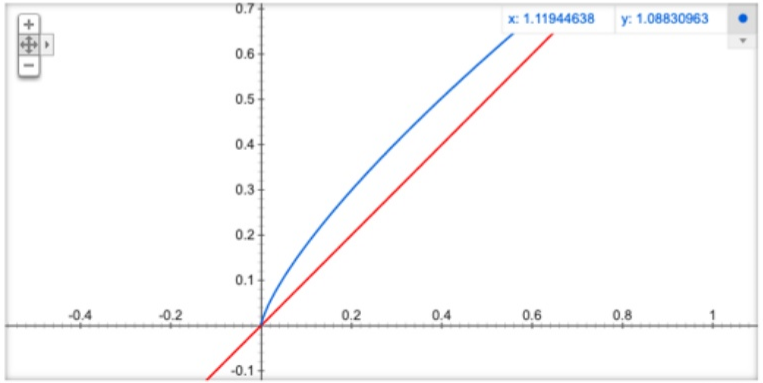
Negative Sampling

- Millions of weights trained
- Modify only a small percentage of weights with each input
- Randomly sample a small number of negative examples (with zero output probability) to update weights i.e., for negative words
- More frequent words more likely selected as negative samples
- *f(wi) = frequency of word i*

The authors state in their paper that they tried a number of variations on this equation, and the one which performed best was to raise the word counts to the 3/4 power:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4} \right)}$$

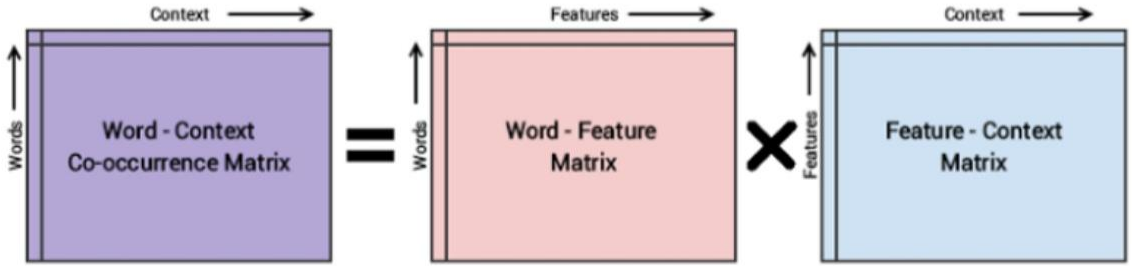
Graph for x^(3/4), x



Source: http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

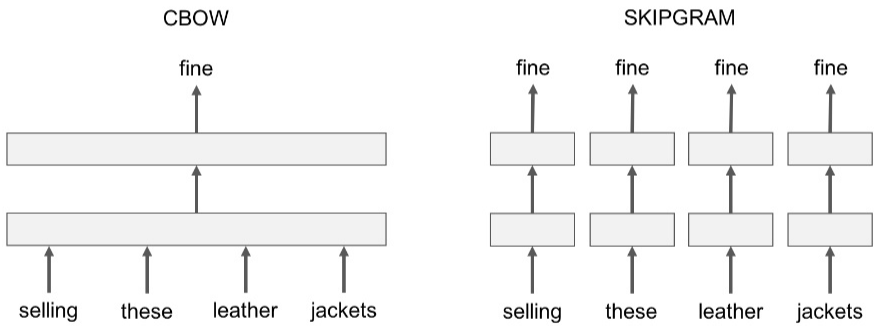
Global Vectors (GloVe) Model

- Word-Context (WC) co-occurence matrix consisting of (word, context) pairs such that each element in this matrix represents how often a word occurs with the context (can be a sequence of words too)
- **Word-Feature (WF) matrix: features = embedding vectors for words**
- Feature-Context (FC) matrix: features = embedding vectors for context
- Apply matrix factorization to approximate WC = WF x FC
- *We typically initialize WF and FC with some random weights and attempt to multiply them to get WC' (an approximation of WC)*



Conceptual model for the GloVe model's implementation

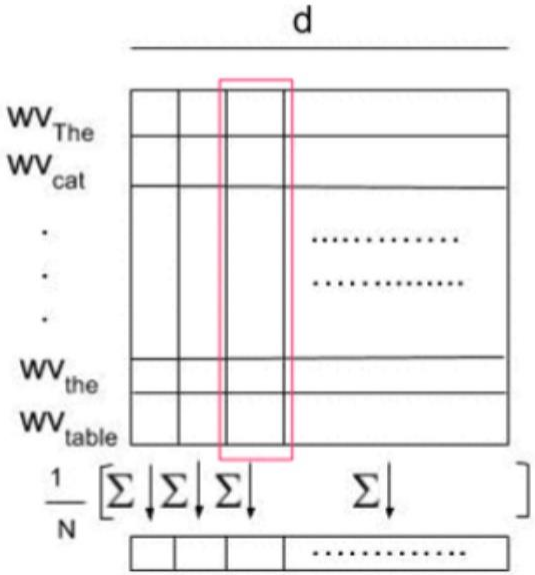
FastText Embeddings



- **FastText has similar training architectures as Word2Vec**
- **Considers each word as a bag of character n-grams**
- **Taking the word where and n=3 (tri-grams) as an example:**

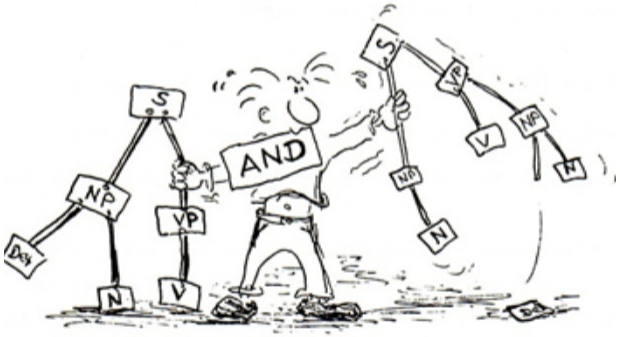
  Word is represented by the character n-grams: **<wh, whe, her, ere, re>** and the special sequence **<where>** representing the whole word

Document Embeddings - Aggregations



- Sentence (S) - “The cat sat on the table”
- How do you represent the embedding of an entire sentence/document/paragraph?
- Aggregation of word embeddings
  - Sum
  - Weighted Mean
  - Average

Document Embeddings - Doc2Vec



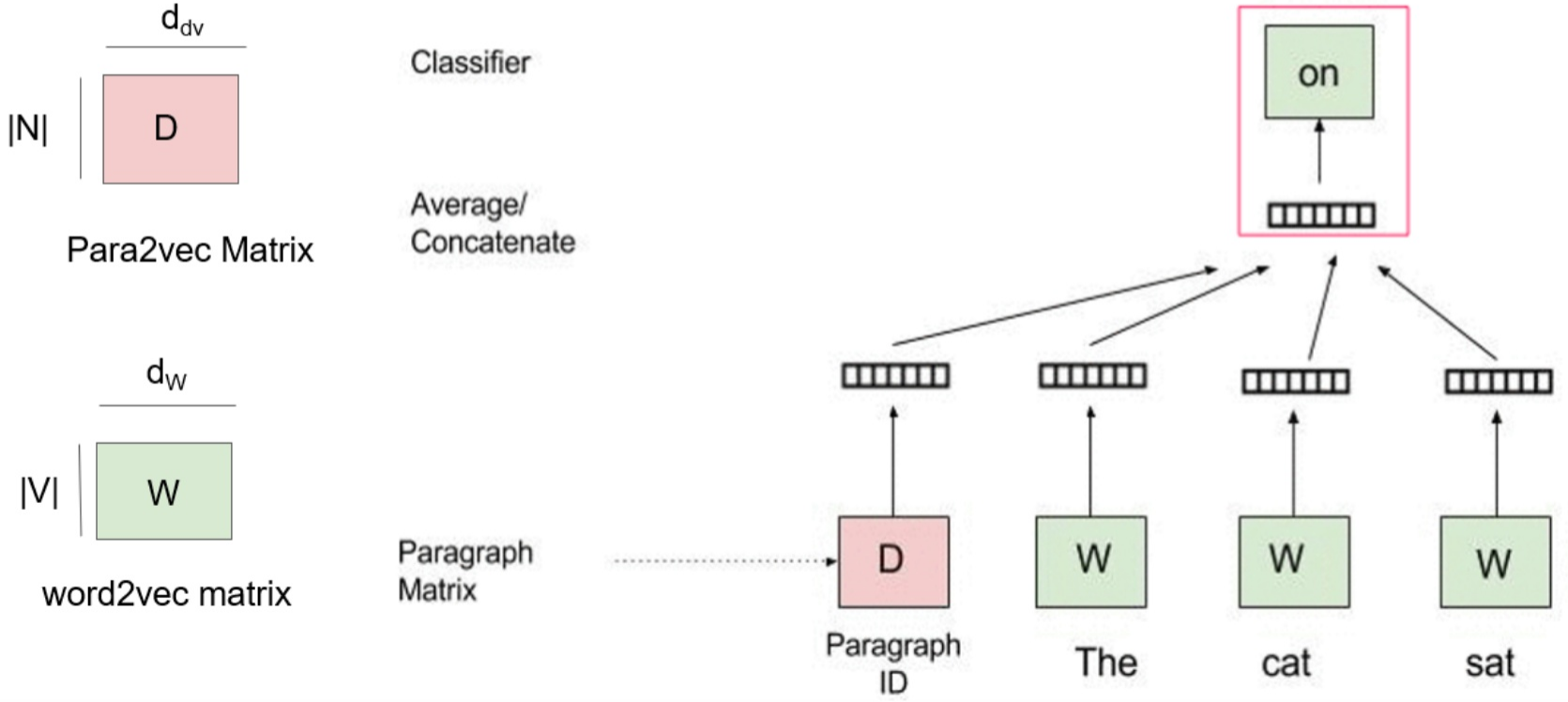
- **Build embedding representation at sentence/paragraph/document level such that it has syntactic and semantic properties**
- **Syntactic properties**
  - Ordering of words
- **Semantic properties**
  - Sentences that have the same meaning should come together
  - Capturing negation
- **Provide fixed length representation for variable length text documents**

Doc2Vec - Distributed Memory Model

- We saw that word2vec uses context words to predict the target word.
- In the Doc2Vec Distributed Memory (DM) model, we simply extend the above idea
- We use a *document vector* along with context word vectors to predict the *next* word.

- S = "***The cat sat on the table***"
- $(S_v, wv_{The}, wv_{cat}, wv_{sat})$ wv_{on}

Doc2Vec - Distributed Memory Model



Doc2Vec - Distributed Bag of Words Model

- We saw that word2vec uses target word to predict the context words.
- In Doc2Vec Distributed Bag of Words (DBOW) model we simply extend the above idea
- In distributed memory model, we simply extend the above idea - we use *document vector* to predict the *context* (surrounding) words.

- S = ***"The cat sat on the table"***

(S_v) (wv_{The}, wv_{cat}, $wv_{sat}wv_{on}$)

Doc2Vec - Distributed Bag of Words Model

- Words and the ordering of the words uniquely define a paragraph.
- Reversing this : a paragraph uniquely defines the words and their ordering present in the paragraph.
- Thus, given a paragraph representation, we should be able to predict the words in the paragraph
- This is precisely what DBOW does.