WordNet (Discrete representation)	**One-hot representation** (count vectorizing)	**Bag of words**	**TF-IDF**
+ Good as a resource	+ Simplest to implement	+ captures word frequency	+ weights to represent whether a word is significant
- Missing **nuances** (e.g. synonyms) - Missing **new words** - Needs **human** labor to create & adapt - Hard to compute accurate **word similarity** - Regards words as atomic symbols	- **Unordered**, context of words is lost - vector rep is in binary form, no **freq** info considered - Really **huge** and **sparse** vectors	- Can't capture word order - Can't capture similarities (e.g. boy, girl, giraffe. BOW cannot say who are more similar) - High dimensional & sparse	- high dimensional - don't capture semantic relatedness

http://hunterheidenreich.com/blog/intro-to-word-embeddings/

Dense Vector Embeddings

(Singular value decomposition etc)

Window-based **Co-occurrence Matrix** Word-doc CM gives general topics - WBCM -> symmetric (irrelevant whether left or right context)	**t-SNE (For visualization) use after SVD** (rep multi-dim vectors in 2D space; suited for visualization) (optimizes st similar points in high-dim space are still closeby in low-dim space)
+ captures both syntactic (POS) & semantic info	
- Increase size with vocab - Very high dim - Subsequent classification model will have sparsity issue (less robust) Sol: Low dimensional vectors Store 'most' of the impt info in a fixed small # dim Reduce dimensionality via **SVD**	**SVD** - For n x m matrix: cost = $O(mn^2)$; bad when dealing with millions of words / docs - Hard to incorporate new words/doc Let's directly learn low-dim word vectors!

Word representations

Traditional Method - Bag of Words Model	Word Embeddings
• **one hot** encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . 0 0 0 0 • Context information is not utilized	• Each word is a **point in *n*-dimensional space**, represented by a **vector of length *n***, (*n* ~ 100 - 300) • Trained by using big text dataset • For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

Directly learning word vectors

Word2vec	Negative Sampling as objective	GloVe
Word2vec Efficient embedding model Instead of capturing co-occurrence counts directly, **predict surround words of each word** - Maximise P(context | focus word) - 2 options (both 1 layer): CBOW, Skip-gram (SG more popular) - **CBOW**: predict word given context, **SG** opposite	**Negative Sampling as objective** - new formulation of objective that implements binary logistic regression to classify between data & noise samples	**GloVe** (efficient count-based model) Combines elements from 2 main word embedding models Instead of learning raw co-occurrence probs, **learn ratios of these co-occurrence prob** **Noise points:** words that don't help us distinguish b/w i and j (use ratio b/w CO prob to filter out these noise points) $P(k \mid j) / P(k \mid i)$ with different k words. **If value is high -> k is useful to distinguish i and j**
To train this neural network, we need to 1. Decide window size (dependent on dataset size / quality) 2. Decide embedding size (#nodes in network) 3. Size of vocab Training obj? Softmax cross-entropy? -> hard to compute Solution: **Negative Sampling**	Results: + Fast & Accurate predictive model + Captures semantic content + Pretrained models available + Super fast to train *(these models try to encode some meaning in the words so that you can use it in your eventual classification model)*	+ Fast training + Scalable to huge corpus, good perf + Easier to parallelize than word2vec, but comparable accuracy + Ratio of co-occurrence probabilities can encode meaning

Negative Sampling for Word2Vec:

Instead of considering all words in vocabulary, consider a few 'negative samples' for each 'positive sample'. Randomly sample 5 negative (false) contexts for each positive (correct) (word, context) in the dataset.

Evaluation of word vectors

1. Intrinsic evaluation: it encodes semantic information (e.g. similarity)
2. Extrinsic evaluation: is it useful for other NLP tasks

Word similarity task, Word analogy task

Extrinsic evaluation: Part of Speech tagging, Named Entity recognition

Application for data science:

- Input to other models
- Sentiment analysis

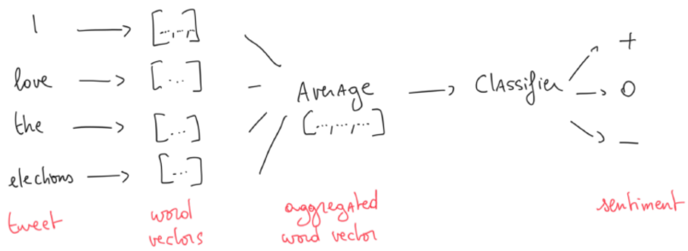
Sentiment Analysis:

1. Preprocessing: remove URLs, weird characters, code tags, etc

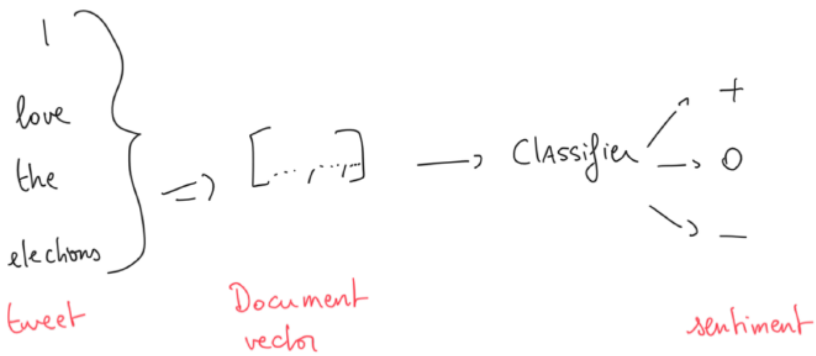
2. Tokenize (i.e. split text into tokens like words)
3. Get embeddings for words (average the vectors to overcome variable length input)

Does not work well…

Example : tweet sentiment prediction



→ problem: does not work super well..



→ Doc2vec

Use **Doc2vec**
-> Generalise word2vec to whole documents
-> Provides fixed-length vector, one vector for your document
+ Faster and consumes less memory than word2vec
+ Typically obtains better results