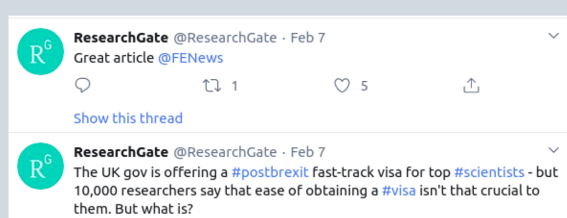
CS 3750: Word Models

PRESENTED BY: MUHENG YAN
UNIVERSITY OF PITTSBURGH
FEB.20, 2020

Is Document Models Enough?

- Recap: previously we have LDA and LSI to learn document representations
- What if we have very short documents, or even sentences? (e.g. Tweets)
- Can we investigate relationships between words/sentences with previous models?
- We need to model words individually for a better granularity



Distributional Semantics: from a Linguistic Aspect

Word Embedding, Distributed Representations, Semantic Vector Space... What are they?

A more formal term from linguistic: **Distributional Semantic Model**

"… quantifying and categorizing semantic similarities between linguistic items based on their distributional properties in large samples of language data." -- Wikipedia

--> Represent elements of language (word here) as distributions of other elements (i.e. documents, paragraphs, sentences, and words)

E.g. word 1 = doc 1 + doc 5 + doc 10 / word 1 = 0.5*word 12 + 0.7*word 24

Document Level Representation

Words as distributions of documents:

Latent Semantic Analysis/Indexing (**LSA/LSI**)

1. Build a co-occurrence matrix of word vs. doc *(n* by *d)*
2. Decompose the Word-Document matrix via SVD
3. Take the highest singular values to get the lower-ranked approximation of the w-d matrix, as the word representations

$$\begin{array}{cc} & \begin{array}{c} X \\ (\mathbf{d}_j) \\ \downarrow \end{array} \\ (\mathbf{t}_i^T) \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,j} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i,1} & \dots & x_{i,j} & \dots & x_{i,n} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,j} & \dots & x_{m,n} \end{bmatrix} \end{array} = (\hat{\mathbf{t}}_i^T) \rightarrow \begin{array}{c} U \\ \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} \end{array} \cdot \begin{array}{c} \Sigma \\ \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} \end{array} \cdot \begin{array}{c} V^T \\ (\hat{\mathbf{d}}_j) \\ \downarrow \\ \begin{bmatrix} [& \mathbf{v}_1 &] \\ & \vdots & \\ [& \mathbf{v}_l &] \end{bmatrix} \end{array}$$

Picture Credit: https://en.wikipedia.org/wiki/Latent_semantic_analysis

Word Level Representation

I. Counting and Matrix Factorization
II. Latent Representation
   I. Neural Network for Language Models
   II. CBOW
   III. Skip-gram
   IV. Other Models
III. Graph-based Models
   I. Node2Vec

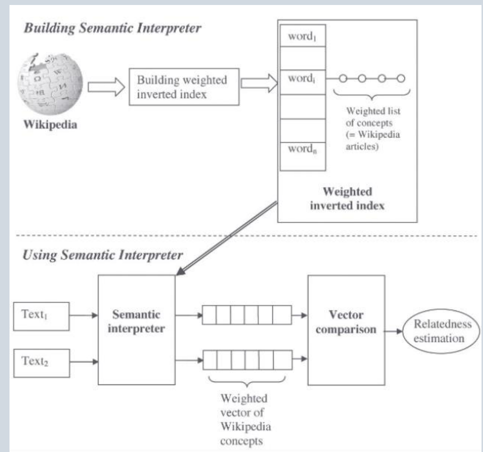
Counting and Matrix Factorization

- Counting methods start with constructing a matrix of co-occurrences between words and words (can be expanded to other levels, e.g. at document level it becomes LSA)
- Due to the high-dimensionality and sparcity, usually used with a dim-reduction algorithm (PCA, SVD, etc.)
- The rows of the matrix approximates the distribution of co-occurring words for every word we are trying to model

Example Models including: LSA, Explicit Semantic Analysis (ESA), Global vectors for word representation (GloVe)

Explicit Semantic Analysis

- Similar words most likely appear with the same distribution of topics
- ESA represents topics by Wikipedia concepts (Pages). ESA use Wikipedia concepts as dimensions to construct the space in which words will be projected
- For each dimension (concept), words in this concept article are counted
- Inverted index is then constructed to convert each word into a vector of concepts
- The vector constructed for each word represents the frequency of its occurrences within each (concept).



Picture and Content Credit: Ahmed Magooda

Global vectors for word representation (GloVe)

1. **Word-word co-occurrence with sliding window (|V| by |V|)** (and normalize as probability)
2. **Construct the cost as:**

$$J = \sum_{i,j}^{|V|} f(X_{i,j})\left(v_i^T v_j + b_i + b_j - \log(X_{i,j})\right)^2$$

3. **Use gradient descent to solve the optimization**

"I learn machine learning in CS-3750"

Window=2	I	learn	machine	learning
I	0	1	1	0
Learn	1	0	1	1
machine	1	1	0	2

GloVe Cont.

How the cost is derived?

Probability of word *i and k* appear together: $P_{i,k} = \frac{X_{ik}}{Xi}$

Using word *k* as a probe, the "ratio" of two word pairs: $ratio_{i,j,k} = \frac{P_{ik}}{P_{jk}}$

To model the ratio with embedding *v*: $J = \sum \left(ratio_{ijk} - g(v_i, v_j, v_k)\right)^2$ -> O(N^3)

Simplify the computation by design $g(\cdot) = e^{(v_i - v_j)^T v_k}$

Thus we are trying to make $\frac{P_{ik}}{P_{jk}} = \frac{e^\wedge(v_i^T v_k)}{e^\wedge(v_j^T v_k)}$

Value of ratio	J and k related	J and k not related
I and k related	1	Inf
I and k not related	0	1

Thus we have $J = \sum \left(\log P_{ij} - v_i^T v_j\right)^2$

To expand the object $\log P_{ij} = v_i^T v_j$, we have $\log(X_{ij}) - \log(X_i) = v_i^T v_j$, then $\log(X_{ij}) = v_i^T v_j + b_i + b_j$. By doing this, we solve the problem that $P_{ij} \neq P_{ji}$but $v_j^T v_i$

Then we come up with the final cost function $J = \sum_{i,j}^{|V|} \boldsymbol{f}(\boldsymbol{X_{i,j}})(\boldsymbol{v}_i^T \boldsymbol{v}_j + \boldsymbol{b}_i + \boldsymbol{b}_j - \log(\boldsymbol{X}_{i,j}))^2$, where $f(\cdot)$ is a weight function

Latent Representation

Modeling the distribution of context* for a certain words through a series of latent variables, by maximizing the likelihood ***P****(word | context)**

Usually fulfilled by neural networks

The learned latent variables are used as the representations of words after optimization

* context refers to the other words from the distribution of which we model the target word
** in some models it could be P(context | word), e.g. Skip-gram*

Neural Network for Language Model

Learning Objective (predicting next word w_j):

Find the parameter set θ to minimize

$L(\theta) = -\frac{1}{T}\left(\sum_j \log(P(w_j|w_{j-1}, \dots, w_{j-n+1}))\right) + R(\theta)$

Where $P(\cdot) = \frac{e^{y_{wi}}}{\sum_{i\neq j} e^{y_{wj}}}$, **Y** = b + $\boldsymbol{W}_{out}$tanh(d + $\boldsymbol{W}_{in}$**X**),

And **X** is the lookup results of the n-length sequence:

X = $[C(w_{j-1}), \dots, c(w_{j-n+1})]$

* **($\boldsymbol{W}_{out}$, b)** is the parameter set of output layer, **($\boldsymbol{W}_{in}$,** d**)** is the parameter set of hidden layer

In this mode we learn the parameters in **C** (|V| * |N|), $\boldsymbol{W}_{in}$ (n * |V| * hidden_size), and $\boldsymbol{W}_{out}$ (hidden_size * |V|)

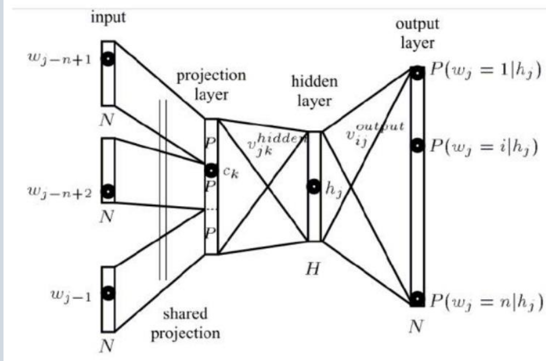


Image credit: Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010)

RNN for Language Model

Learning Objective: similar to NN for LM

Alter from NN:

- The hidden layer is now the linear combination of the input current word *t* and the hidden of previous word *t-1*:

$$s(t) = f(\boldsymbol{U}w(t) + \boldsymbol{W}s(t-1))$$

Where $f(\cdot)$ is the activation function

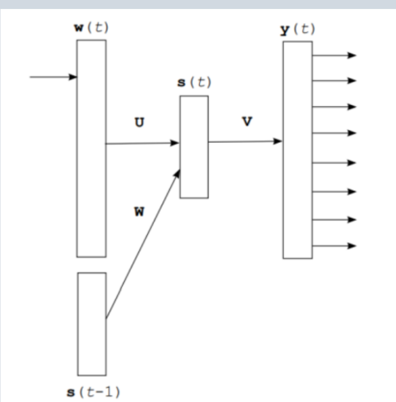


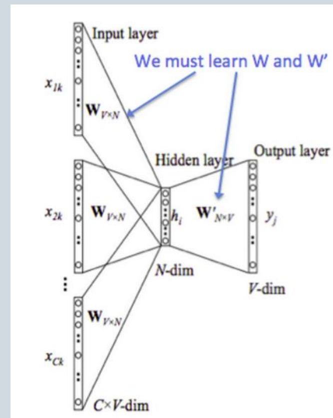
Image credit: Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010)

Continuous Bag-of-Words Model

Learning Objective: maximizing the likelihood of $P(word|context)$ for every word in a corpus

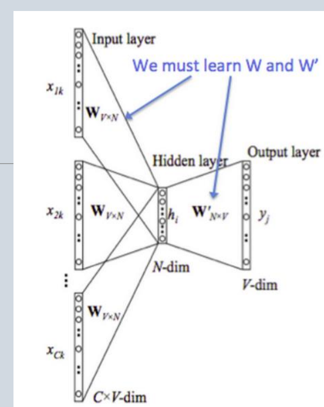
Similar to NN for LM, the inputs are one-hot vectors and the matrix $\boldsymbol{W}$ here is like the look-up matrix.

Differences compared to the NN for LM:
- Bi-directional: not predicting the "next", instead predicting the center word inside a window, where words from both directions are input
- Significantly reduced complexity: only learns 2 * |V| * |N| parameters



Picture Credit: Francois Chaubard, Rohit Mundra, Richard Socher, from https://cs224d.stanford.edu/lecture_notes/notes1.pdf

CBOW Cont.



Steps breakdown:

1. Generate the one-hot vectors for the context: $(\boldsymbol{x}^{c-m}, \ldots, \boldsymbol{x}^{c-1}, \boldsymbol{x}^{c+1}, \ldots, \boldsymbol{x}^{c+m} \in \boldsymbol{R}^{|V|})$, and lookup for the word vectors $\boldsymbol{v}^i = \boldsymbol{W}\boldsymbol{x}^i$
2. Average the vectors over contexts: $\boldsymbol{h}_c = \frac{v_{c-m} + \ldots + v_{c+m}}{2m}$
3. Generate the posterior $\boldsymbol{z}_c = \boldsymbol{W}'\boldsymbol{h}_c$, and turn it in to probabilities $\widehat{\boldsymbol{y}}_c = softmax(\boldsymbol{z}_c)$
4. Calculate the loss as cross-entropy: $\sum_{i=1}^{|V|} y_i \log(\hat{y}_i)$
   ➔ $P(w_c|w_{c-m}, \ldots w_{c+m})$

Notations:

m: half window size
c: center word index
w_i: word *i* from vocabulary V
$\boldsymbol{x}_i$: one-hot input of word *i*
$\boldsymbol{W} \in \boldsymbol{R}^{|V| \times n}$: *the context lookup matrix*
$\boldsymbol{W}' \in \boldsymbol{R}^{n \times |V|}$: *the center lookup matrix*

CBOW Cont.

Loss fuction:

$For\ all\ w_c \in V\ , minimize$

$$J(\cdot) = logP(w_c|w_{c-m}, \ldots w_{c+m})$$
$$\Rightarrow -\frac{1}{|V|}\sum logP(\boldsymbol{W}_c|\boldsymbol{h}_c)$$
$$= -\frac{1}{|V|}\sum log\frac{e^{{w'_c}^T \boldsymbol{h}_c}}{\sum_{j=1}^{|V|} e^{{w'_j}^T \boldsymbol{h}_c}}$$
$$= -\frac{1}{|V|}\sum -\boldsymbol{w}_c'^T\boldsymbol{h}_c + \log(\sum_{j=1}^{|V|} e^{{w'_j}^T \boldsymbol{h}_c})$$

Optimization: use SGD to update all relevant vectors $\boldsymbol{w}_c'$ and $\boldsymbol{w}$

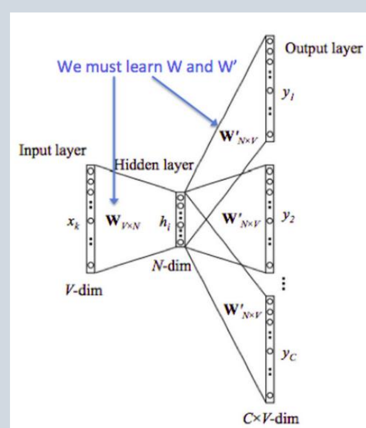
Skip-gram Model

Learning Objective: maximizing the likelihood of $P(context|word)$ for every word in a corpus

Steps Breakdown:

1. Generate one-hot vector for the center word $\boldsymbol{x} \in \boldsymbol{R}^{|V|}$, and calculate the embedded vector $\boldsymbol{h}_c = \boldsymbol{Wx} \in \boldsymbol{R}^n$
2. Calculate the posterior $\boldsymbol{z}_c = \boldsymbol{W}'\boldsymbol{h}_c$
3. For each word j in the context of the center word, calculate the probabilities $\widehat{\boldsymbol{y}}_c = softmax(\boldsymbol{z}_c)$
4. We want the probabilities $\hat{y}_{cj}$ in $\widehat{\boldsymbol{y}}_c$ match the true probabilities of the contexts which are $y^{c-m}, \ldots, y^{c+m}$

Cost function constructed similarly to the CBOW model



Picture Credit: Francois Chaubard, Rohit Mundra, Richard Socher, from https://cs224d.stanford.edu/lecture_notes/notes1.pdf

Skip-gram Cont.

Cost Function:

$for\ every\ center\ word\ w_c\ in\ |V|\ ,minimize:$

$$J(\cdot) = -logP(w_{c-m}, \dots w_{c+m}|w_c)$$
$$= -log\prod_{j=0,j\neq m}^{2m} P(w_{c-m+j}|w_c)$$
$$= -log\prod P(\boldsymbol{w}'_{c-m+j}|\boldsymbol{h}_c)$$
$$= -\log\prod\frac{e^{\boldsymbol{w}'^{T}_{c}\boldsymbol{h}_c}}{\sum_{j=1}^{|V|} e^{\boldsymbol{w}'^{T}_{j}\boldsymbol{h}_c}}$$

Skip-gram with Negative Sampling

An alternative way of learning skip-gram:

From the previous learning method, we have looped heavily on negative samples when summing over |V|

Alternatively, we can reform the learning objective in order to enabling "negative sampling", where we only take a few negative samples in each epoch

Alternative Objective: maximize the likelihood of *P(D=1|w, c)* if the word pair (*w, c*) is from the data, and minimize the likelihood of *P(D=0|w, c)* if (*w, c*) is not from the data

Skip-gram with Negative Sampling

We model the probability as:

$$P(D=1|w,c,\theta) = sigmoid(\boldsymbol{h}_c^T\ \boldsymbol{h}_w) = \frac{1}{1+e^{-h_c^T h_w}}$$

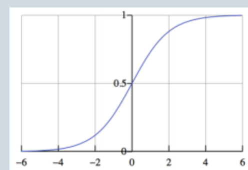
And the optimization of the loss would be:

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in Data} P(D=1|w,c,\theta) \prod_{(w,c)\notin Data} P(D=0|w,c,\theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in Data} P(D=1|w,c,\theta) \prod_{(w,c)\notin Data} (1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum \log \frac{1}{1+e^{-h_c^T h_w}} \sum \log(1 - \frac{1}{1+e^{-h_c^T h_w}})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum \log \frac{1}{1+e^{-h_c^T h_w}} \sum \log \frac{1}{1+e^{h_c^T h_w}}$$

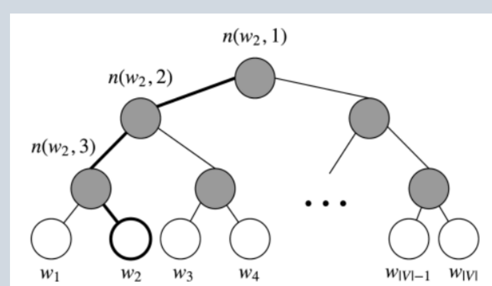


Hierarchical Softmax and FastText

Hierarchical Softmax:

An alternative way to solve the dimensionality problem when softmaxing through $\boldsymbol{y}$:

1. Build a Binary Tree of words in V, each non-leaf nodes are associated with a pseudo-output to learn.
2. Define the loss for word c as the path to the word from root
3. The probability of $P(w_c|context)$ now becomes
   $\prod_{j=1}^{length\ of\ path} sigmoid([n(w,j+1)\ is\ the\ childern] \cdot \boldsymbol{h}_{(w,j)}^T \boldsymbol{h}_c)$



FastText: Sub-word n-grams + Hierarchical Softmax
"apple" and "apples" are referring to the same semantic, yet word model ignores such sub-word features.

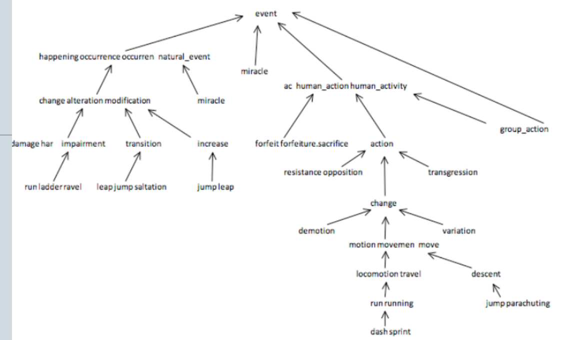
FastText model introduces sub-word n-gram inputs, while having a similar architecture as skip-gram models. This expands the dimension of $\boldsymbol{y}$ to a even larger number. Thus it adopts the Hierarchical Softmax to speed up the computation

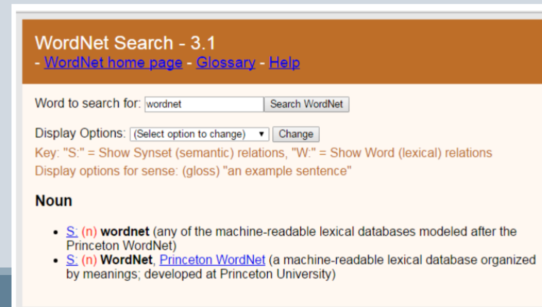
Graph-based Models

WordNet:

- A large lexical database of words. Words are grouped into set of synonyms (synsets), each expressing a distinct concept
- Provides Word senses, Part of Speech, semantic relationships between words
- From which we can construct a real "net" of words where words are nodes and word relationships defined by synsets as edges

Image credit: University of Princeton




WordNet Search - 3.1
- WordNet home page - Glossary - Help

Word to search for: wordnet Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **wordnet** (any of the machine-readable lexical databases modeled after the Princeton WordNet)
- S: (n) **WordNet**, Princeton WordNet (a machine-readable lexical database organized by meanings; developed at Princeton University)

Content Credit: Ahmed Magooda

Node2Vec

A model to learn representations of nodes:

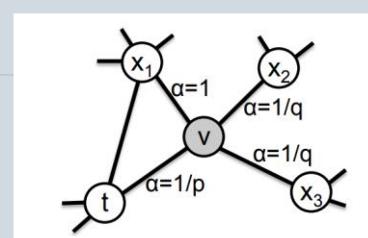
- Applied to any graphs including word nets
- Turns graph topology into sequences with the random walk algorithm:
- Start from a random node *v*, the probability of travel to another node is:

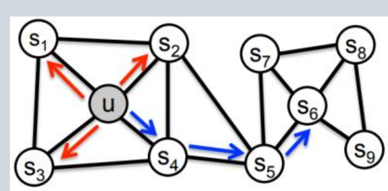
$$P(x|v) = \begin{cases} \frac{\pi_{vx}}{Z} & if\ (v,x) \in E \\ 0 & otherwise \end{cases}$$

- The transition probability is defined as:

$$\pi_{vx} = \begin{cases} \frac{1}{p} & if\ x\ is\ the\ previous\ node\ t \\ 1 & if\ x\ is\ the\ neighbor\ of\ t \\ \frac{1}{q} & if\ x\ is\ neighbor\ of\ neighbor\ of\ t \end{cases}$$

- *p* and *q* controls the strategy of breath-first search or depth-first search
- *With the sequence generated, we can embed nodes as we did in language models*





Evaluation of Word Representations

- **Intrinsic Evaluation:**
  - How good are the representations?
- **Extrinsic Evaluation:**
  - How effective are the learned representations in other downstream tasks?

Content Credit: Ahmed Magooda

Intrinsic Evaluations

Word Similarity Task

- Calculate the similarity of word pairs from the learned vectors through a various distance metrics (e.g. euclidean, cosine, etc.)
- Compare the calculated word similarities with human-annotated similarities
- Example test set: word-sim 353 (http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/)

Analogy Task:

- Proposed by Mikolov et.al (2013)
- For a specific word relation, given *a, b, y*; find *x* so that "*a* is to *b* as *x* is to *y*"
  - "*man is to king as woman is to queen*"

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter

"Analogy task" Content Credit: Ahmed Magooda

Extrinsic Evaluations

Learned word representations can be taken as inputs to encode texts in downstream NLP tasks, including:

- Sentiment Analysis, POS tagging, QA, Machine Translation...
- **GLUE benchmark**: a collection of dataset including 9 sentence language understanding tasks (https://gluebenchmark.com/)

Summary

What we have covered:

	Document-level	Word-level
Count & Decomposition	- LSA	- GloVe
Latent Vector Representation		- NN for LM - CBOW, Skip-gram, FastText - Node2Vec

- "words as distributions"
- evaluations of the word representations

Software at Fingertips

LSA : manual through **sklearn** or **Gensim**

CBOW, Skip-gram: **Gensim**

Neural-networks: **Torch, Tensorflow**

Pre-trained word representations:

- **Word2vec:** (https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit)
- **Glove:** (https://nlp.stanford.edu/projects/glove/)
- **Fasttext:** (https://fasttext.cc/)

References

Word2Vec:

- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality.
- Mikolov, T., Yih, W. T., & Zweig, G. (2013, June). Linguistic Regularities in Continuous Space Word Representations.
- Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, 135-146.

Counting:

- Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).
- Gabrilovich, E., & Markovitch, S. (2007, January). Computing semantic relatedness using wikipedia-based explicit semantic analysis.
- Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis.

NN for LM:

- Mikolov, T., Kopecky, J., Burget, L., & Glembek, O. (2009, April). Neural network based language models for highly inflective languages.
- Mikolov, T., Karafiát, M., Burget, L., Černocký, J., & Khudanpur, S. (2010). Recurrent neural network based language model.

Graph-based:

- Princeton University "About WordNet." WordNet. Princeton University. 2010. http://wordnet.princeton.edu
- Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 855-864).