A Brief Survey of Word Embedding Methods

Kian Kenyon-Dean

Computational Linguistics Lab Meeting, August 6th 2018

What the heck is a word embedding?

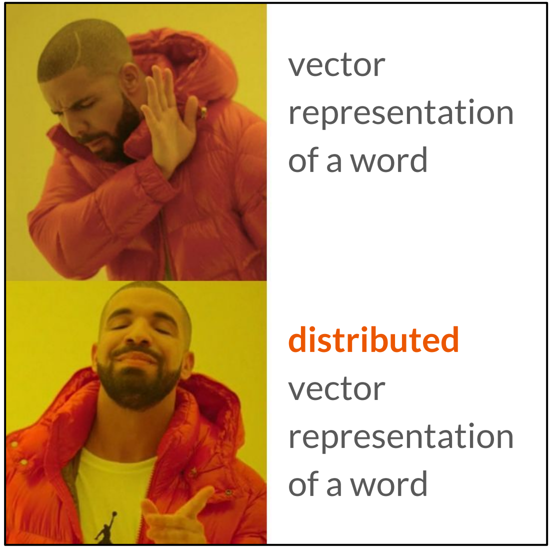
- A vector representation of a word

What the heck is a word embedding?

- ~~A vector representation of a word~~
  - **No!** Bag-of-words one-hot-encodings are not word embeddings.

What the heck is a word embedding?

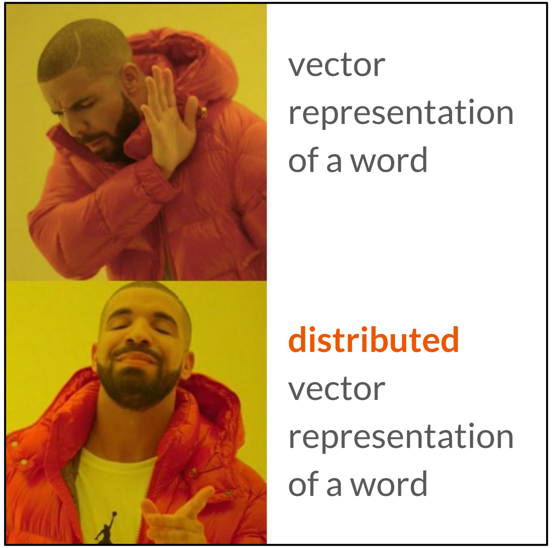
- ~~A vector representation of a word~~
- A **distributed** vector representation of a word.



What the heck is a word embedding?

A **distributed** vector representation of a word.

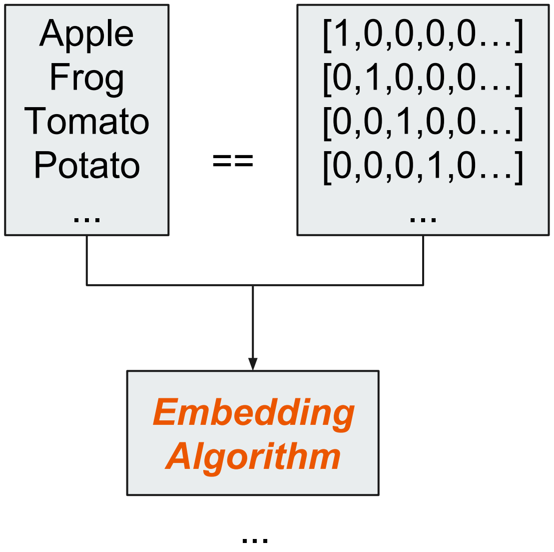
- A *mapping* from a *one-hot-encoded space* to a much lower dimensional *continuous space*.
- A *vocabulary* or *dictionary* is really a one-hot-encoded vector space.



What the heck is a word embedding?

A **distributed** vector representation of a word.

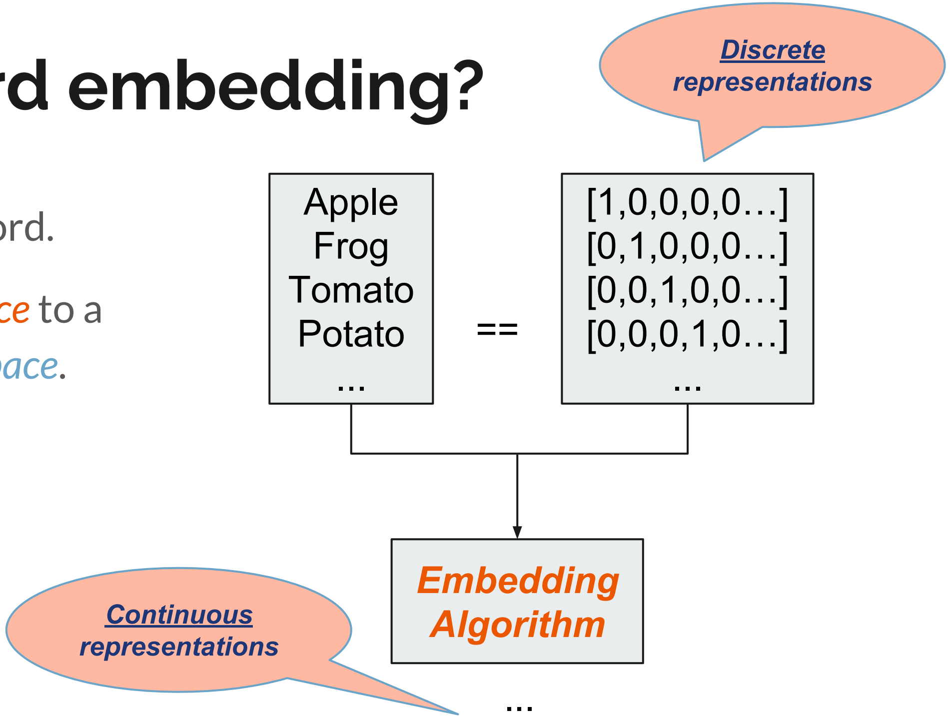
- A *mapping* from a *one-hot-encoded space* to a much lower dimensional *continuous space*.
- A *vocabulary* or *dictionary* is really a one-hot-encoded vector space.



What the heck is a word embedding?

A **distributed** vector representation of a word.

- A *mapping* from a *one-hot-encoded space* to a much lower dimensional *continuous space*.
- A *vocabulary* or *dictionary* is really a one-hot-encoded vector space.



Why do we want distributed representations?

Why do we want distributed representations?

- One-hot representations have **no linear expressivity**!
  - E.g., `cos([cat], [dog]) == cos([water], [hedgefund])`

Why do we want distributed representations?

- One-hot representations have **no linear expressivity**!
  - E.g., `cos([cat], [dog]) == cos([water], [hedgefund])`
- Way **too large** to be practical - dimensionality is size of vocabulary!

Why do we want distributed representations?

- One-hot representations have **no linear expressivity**!
  - E.g., `cos([cat], [dog]) == cos([water], [hedgefund])`
- Way **too large** to be practical - dimensionality is size of vocabulary!
- The **curse of dimensionality** is resolved (Bengio et al, 2003)
  - Distributed representations offer **local smoothness properties**, which can **generalize** a language model over syntactically/semantically related words.
  - Otherwise, to model a joint distribution of *10-word* sentences in a language with vocabulary size 100,000, there are **10^{50} free parameters**!

Why do we want distributed representations?

- One-hot representations have **no linear expressivity**!
  - E.g., `cos([cat], [dog]) == cos([water], [hedgefund])`
- Way **too large** to be practical - dimensionality is size of vocabulary!
- The **curse of dimensionality** is resolved (Bengio et al, 2003)
  - Distributed representations offer **local smoothness properties**, which can **generalize** a language model over syntactically/semantically related words.
  - Otherwise, to model a joint distribution of *10-word* sentences in a language with vocabulary size 100,000, there are 10^{50} **free parameters**!

Why do we want distributed representations?

Motivated by the **distributional hypothesis** (Harris, 1954):

> ***You shall know a word by the company it keeps.*** (Firth, 1957)

- Words with similar ***distributions*** will have similar meanings
- Words that appear in similar contexts have similar meanings

Language Modelling

Here, we would like to model the probability of a word given the previous sequence. This practically requires us to limit to the previous *m* words in the sequence.

$$p(w_n|w_1, w_2, \cdots, w_{n-1}) \approx p(w_n|w_{n-m}, \cdots, w_{n-2}, w_{n-1})$$

Language Modelling

Here, we would like to model the probability of a word given the previous sequence. This practically requires us to limit to the previous *m* words in the sequence.

$$p(w_n | w_1, w_2, \cdots, w_{n-1}) \approx p(w_n | w_{n-m}, \cdots, w_{n-2}, w_{n-1})$$

Traditional methods are *count-based*; e.g., for trigrams: $p(w_3 | w_1, w_2) = \frac{count(w_1, w_2, w_3)}{\sum_w count(w_1, w_2, w)}$

Language Modelling

We would like to model the probability of a word given the previous sequence. This practically requires us to limit to the previous *m* words in the sequence.

$$p(w_n|w_1, w_2, \cdots, w_{n-1}) \approx p(w_n|w_{n-m}, \cdots, w_{n-2}, w_{n-1})$$

Traditional methods are *count-based*; e.g., for trigrams: $p(w_3|w_1, w_2) = \frac{count(w_1,w_2,w_3)}{\sum_w count(w_1,w_2,w)}$

<u>Problems</u>: many **sequences will have 0 probability** (requires not great smoothing techniques),

Language Modelling

We would like to model the probability of a word given the previous sequence. This practically requires us to limit to the previous *m* words in the sequence.

$$p(w_n|w_1, w_2, \cdots, w_{n-1}) \approx p(w_n|w_{n-m}, \cdots, w_{n-2}, w_{n-1})$$

Traditional methods are *count-based*; e.g., for trigrams: $p(w_3|w_1, w_2) = \frac{count(w_1,w_2,w_3)}{\sum_w count(w_1,w_2,w)}$

Problems: many sequences will have 0 probability (requires not great smoothing techniques), **no generalization** for semantically similar words,

Language Modelling

We would like to model the probability of a word given the previous sequence. This practically requires us to limit to the previous *m* words in the sequence.

$$p(w_n | w_1, w_2, \cdots, w_{n-1}) \approx p(w_n | w_{n-m}, \cdots, w_{n-2}, w_{n-1})$$

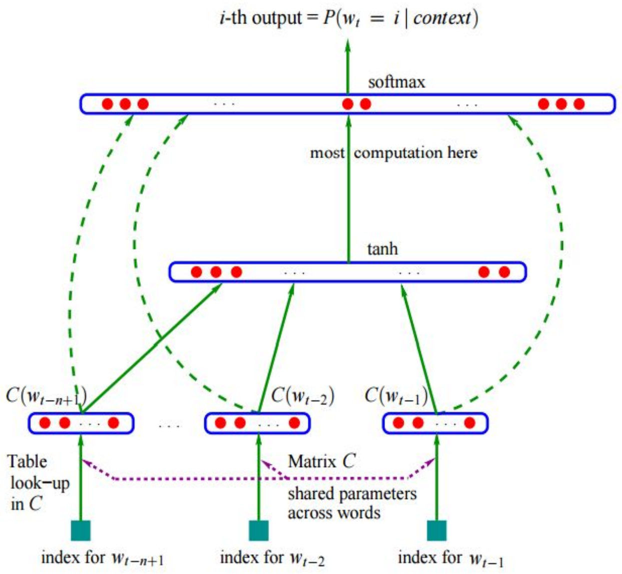
Traditional methods are *count-based*; e.g., for trigrams: $p(w_3 | w_1, w_2) = \frac{count(w_1, w_2, w_3)}{\sum_w count(w_1, w_2, w)}$

<u>**Problems**</u>: many sequences will have 0 probability (requires not great smoothing techniques), no generalization for semantically similar words, **depends on Markov assumption** (no longer than *m* dependency understanding).

Neural Language Modelling

Solution: represent words as *continuous vectors* in R^m, m << |V|.

- Learn vectors by building an **NNLM** (neural network language model)
- Objective: predict $P(w_t = i \mid \textit{context words})$
- i.e., map a sequence $w_{t-n+1}, \ldots, w_{t-1}$ to predict the probability that w_t is word *i*
- Key: represent words with vectors *C(i)* for word *i*



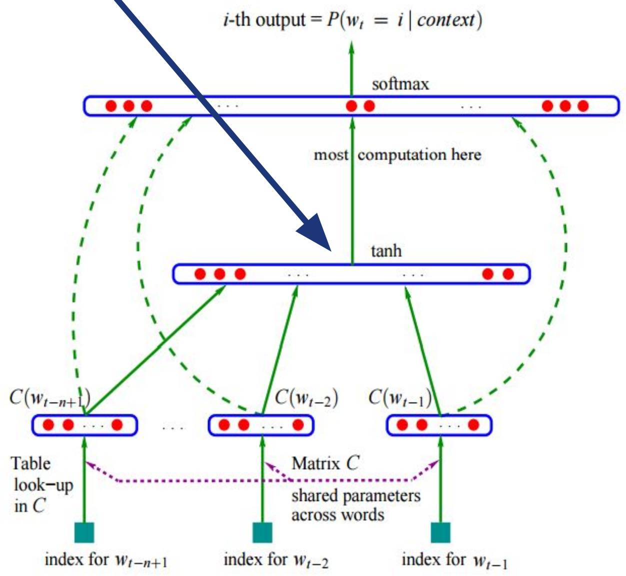
Neural Language Modelling

Solution: represent words as *continuous vectors* in R^m, m << |V|.

- Learn vectors by building an **NNLM** (neural network language model)
- Objective: predict $P(w_t = i \mid \textit{context words})$
- i.e., map a sequence $w_{t-n+1}, \ldots, w_{t-1}$ to predict the probability that w_t is word i
- Key: represent words with vectors $C(i)$ for word i

A *feed-forward* **NNLM** will *concatenate* the fixed number of context vectors.

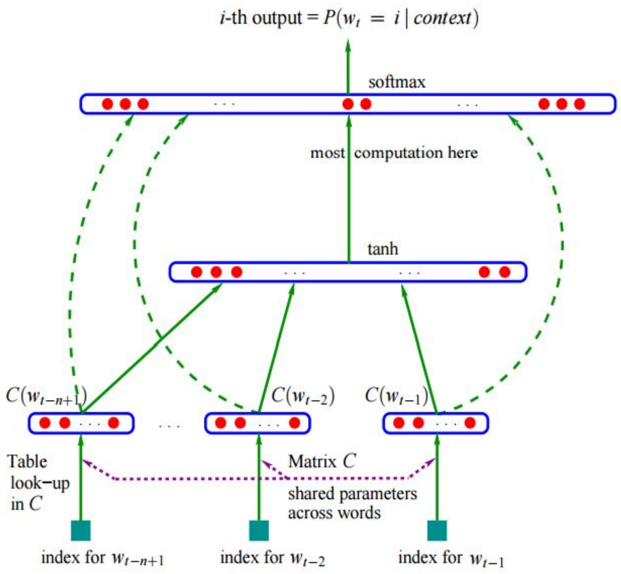
A *recurrent* **NNLM** uses an RNN to pool the vectors, thus incorporating word order.



Neural Language Modelling

Characteristics & Problems

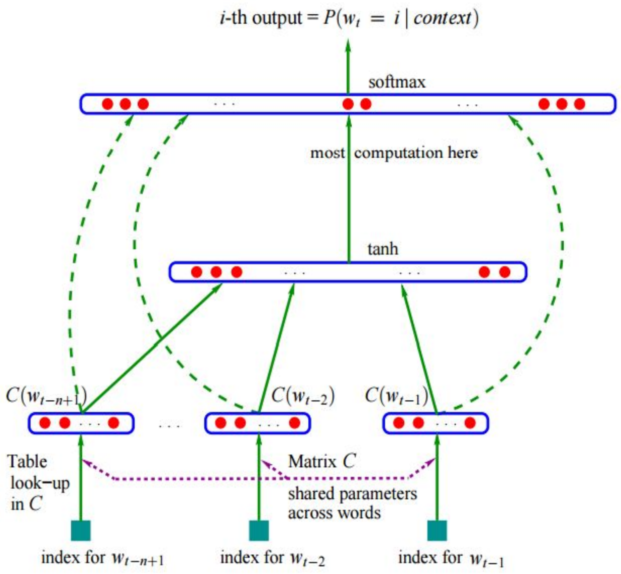
- *Softmax **output layer is huge**! |V| output neurons!*
- Feed-forward NNLM requires *fixed context length:*
  - **No generalization across words** in different positions on the parameters of the model;
  - Takes **a long time to train** (weeks!)
    - Bengio 2003: 3 weeks on 40 CPUs!
- Recurrent NNLM **generalizes model parameters**
  - Difficult to train, chaotic dynamical system
  - But, is **faster** to train than FFNN
  - Generally **performs better** than FFNN



Neural Language Modelling

Characteristics & Problems

- *Softmax **output layer is huge!** |V| output neurons!*
- Feed-forward NNLM requires *fixed context length:*
    - **No generalization across words** in different positions on the parameters of the model;
    - Takes **a long time to train** (weeks!)
        - Bengio 2003: 3 weeks on 40 CPUs!
- Recurrent NNLM **generalizes model parameters**
    - Difficult to train, chaotic dynamical system
    - But, is **faster** to train than FFNN
    - Generally **performs better** than FFNN



Matrix Factorization

- Global method to build low-rank approximations of a massive matrix of word co-occurence statistics in a corpus.
- Includes methods:
    - LSA (latent semantic analysis)
        - Word-document matrix. M_{ij} = # times word *i* appears in document *j*
    - HAL (hyperspace analogue to language)
        - Word-word matrix. M_{ij} = # times word *i* appears in **some local context** of another word *j*
    - [H]PCA (Hellinger principal component analysis)
        - PCA is a common method to factorize a matrix into smaller vectors.

Matrix Factorization

- Global method to build low-rank approximations of a massive matrix of word co-occurence statistics in a corpus.
- Includes methods:
    - **LSA** (latent semantic analysis)
        - Word-document matrix. M_{ij} = # times word *i* appears in document *j*
    - **HAL** (hyperspace analogue to language)
        - Word-word matrix. M_{ij} = # times word *i* appears in **some local context** of another word *j*
    - [H]**PCA** ([Hellinger] principal component analysis)
        - PCA is a common method to factorize a matrix into smaller vectors.

Matrix Factorization

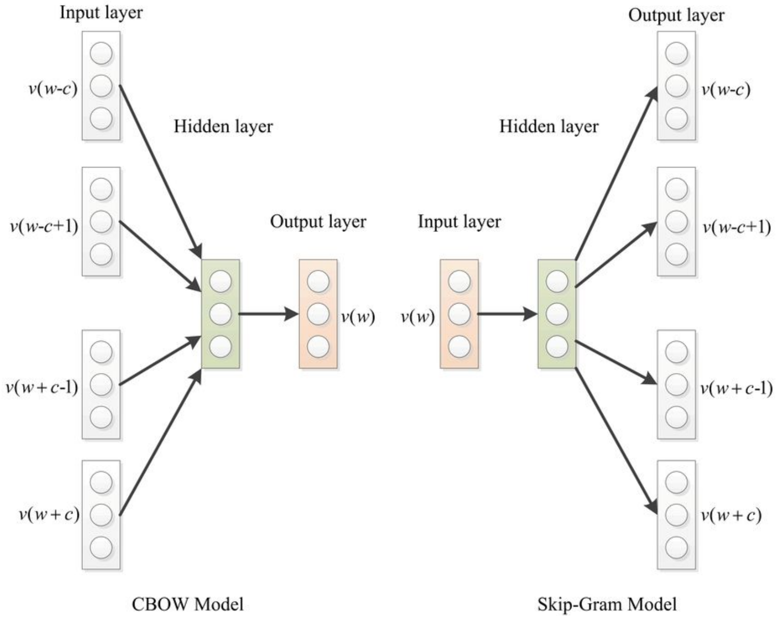
- Begin with matrix **M** with dimensions |V| x |D|
    - D is the set of *context words* that we care about (often just the vocabulary V)
- Define $\mathbf{M}_{ij}$ in some clever way:
    - E.g., $\mathbf{M}_{ij}$ = P(word *j in* "context of" word *i*); count this in training corpus
    - Can make it PPMI: divide by P(i)*P(j)
- Learn a set of |V| vectors **V** and |D| *"context vectors"* **W** with an objective function
    - Base form: $\mathbf{v}_i * \mathbf{w}_j = \mathbf{log}(\mathbf{M}_{ij})$
- Learning is done with matrix factorization algorithm

Matrix Factorization

- Begin with matrix **M** with dimensions |V| x |D|
    - D is the set of *context words* that we care about (often just the vocabulary V)
- Define $\mathbf{M}_{ij}$ in some clever way:
    - E.g., $\mathbf{M}_{ij}$ = P(word *j in* "context of" word *i*)
    - Can make it PPMI: divide by P(i)*P(j)
- Learn matrix of word vectors **V** (|V|x *d*) and *"context vectors"* **W** (*d* x |D|) with objective function
    - Base form: $\mathbf{v}_i * \mathbf{w}_j = \log(\mathbf{M}_{ij})$
- Learning is done with matrix factorization algorithm for factorizing: **log(M) = VW**

Word2vec - Skipgram & CBOW

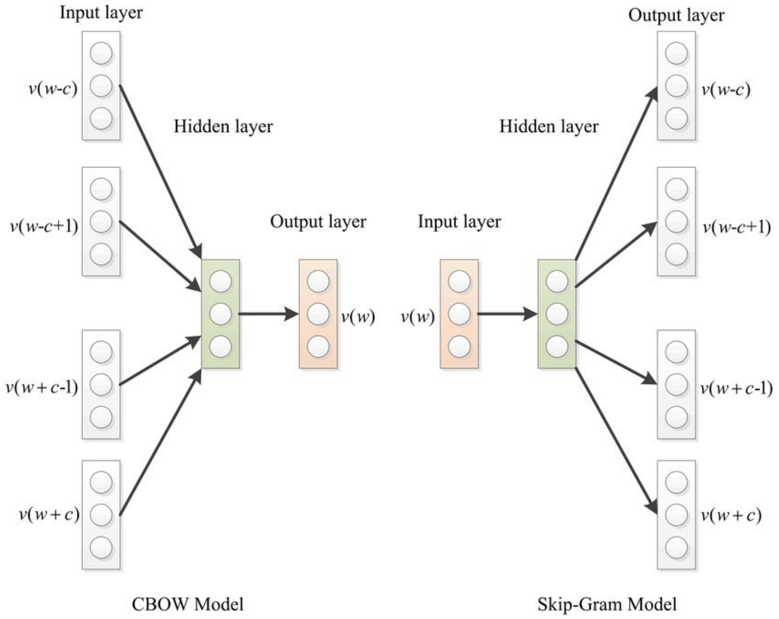
- Expressed as an online neural learner trained to *maximize the log-likelihood of the context given word*, for Skip-gram (vice versa for CBOW)
- But is actually "**log-linear**", no nonlinear projection layer (unlike NNLMs)
- *Linear makes sense*: using a nonlinear neural network "***would obfuscate the linear structure we are trying to capture***" [Glove paper]



Word2vec - Skipgram & CBOW

Unique characteristics

- Online learner, much much faster than NNLMs
- Uses *negative sampling* in the softmax
- Uses *hierarchical softmax* in the objective
- *Subsamples* frequent words to avoid pollution
- CBOW: predict **word** given **context**
- Skip-gram: predict **context** given **word**



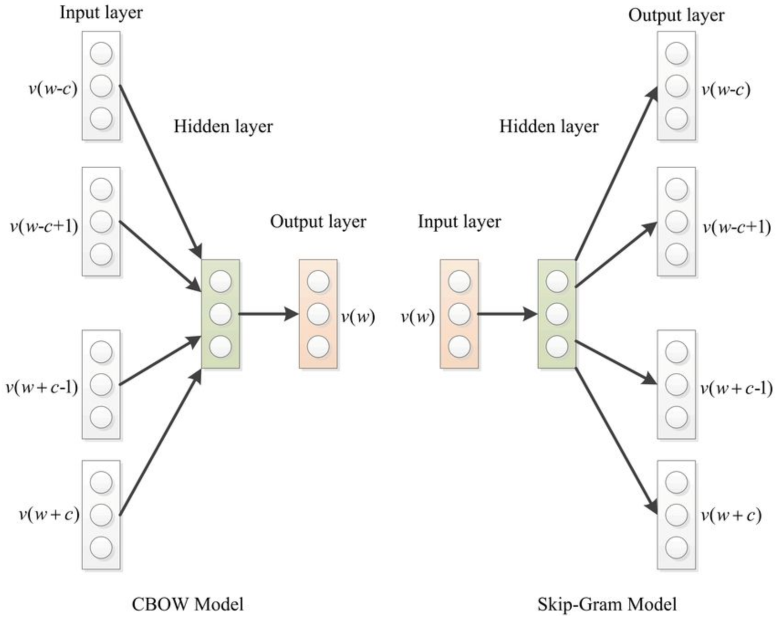
Word2vec - Skipgram & CBOW

Evaluation in the Papers

- All evaluations were on *semantic* and *syntactic* word analogy tasks & sentence completion
- They do better than NNLMs!
- *No downstream task evaluations!*

Skip-gram vs CBOW

- CBOW: *faster* to train, slightly **better on syntax**
- Skip-gram: better on *infrequent words* and **better for semantic** relationships



Glove - “Global Vectors”

- While Word2vec is presented as a *local context-window approach*, these authors proved that it really is **optimizing a global objective function** for *approximating corpus statistics.*

Glove - “Global Vectors”

- While Word2vec is presented as a *local context-window approach*, these authors proved that it really is **optimizing a global objective function** for *approximating corpus statistics.*
- Glove approximates global corpus statistics directly:
  - X_{ij} = # of times *j* occurs in context of *i*
  - Learn *vectors* and *covectors* (as word2vec)

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Glove - "Global Vectors"

- While Word2vec is presented as a *local context-window approach*, these authors proved that it really is **optimizing a global objective function** for *approximating corpus statistics.*
- Glove approximates global corpus statistics directly:
  - X_{ij} = # of times *j* occurs in context of *i*
  - Learn *vectors* and *covectors* (as word2vec)
  - Real objective:
    - $v_i * v_j = \log X_{ij}$
- So, **the dot-product between two vectors should equal the log of their co-occurence.**

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

Glove - "Global Vectors"

- While Word2vec is presented as a *local context-window approach*, these authors proved that it really is **optimizing a global objective function** for *approximating corpus statistics.*
- Glove approximates global corpus statistics directly:
  - X_{ij} = # of times *j* occurs in context of *i*
  - Learn *vectors* and *covectors* (as word2vec)
  - Real objective:
    - $v_i * v_j = \log X_{ij}$

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- So, **the dot-product between two vectors should equal the log of their co-occurence.**
- So, there is **no mystery in word embeddings** - they are optimized to measure frequency!

Fasttext - “Vectors with Subword Information”

- Also by Mikolov (this time, Facebook)
- **Capture morphology**: allows for rare words to be represented more confidently based on n-grams from n=3 to 6.
- Simply use Skip-gram as well as Word2vec, no real differences in learning model

Problems with these Word Embeddings

- No differentiation between **conceptual similarity** and **semantic similarity**!
- **No semantic/syntactic relations** other than measuring corpus co-occurence **are directly imposed**!
- Inability to capture/represent **polysemy** (no solutions for this presented today)

Retrofitting (Faruqui et al., NAACL 2015)

- A **post-processing** step to augment *pre-trained word embeddings.*
- Main idea: use an **ontology** (i.e.., an undirected graph) to encode **semantic relations** that *should* be captured by your word embeddings.
    - Ontology: e.g., Wordnet, defining a graph with vertices as *words*, edges as *relations*
    - Semantic relation: *anything you want to represent* - synonymy, hyponymy, "is bigger than"-omy ...

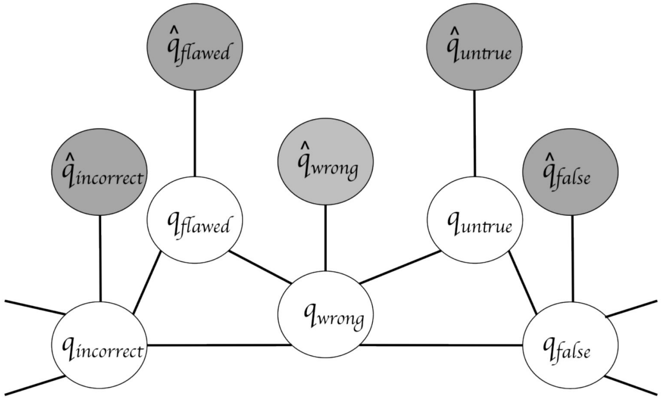
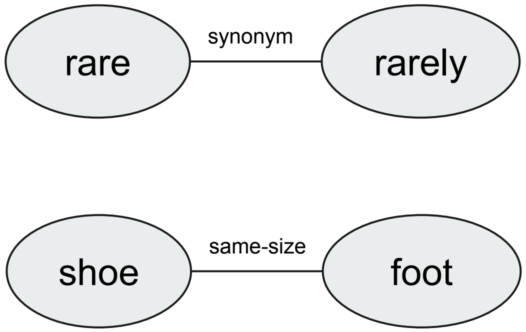


Figure 5: Word graph with edges between related words showing the observed and the inferred word vector representations.

Retrofitting (Faruqui et al., NAACL 2015)

- A **post-processing** step to augment *pre-trained word embeddings.*
- Main idea: use an **ontology** (i.e.., an undirected graph) to encode **semantic relations** that *should* be captured by your word embeddings.
    - Ontology: e.g., Wordnet, defining a graph with vertices as *words*, edges as *relations*
    - Semantic relation: *anything you want to represent* - synonymy, “same size as”-omy ...



Retrofitting (Faruqui et al., NAACL 2015)

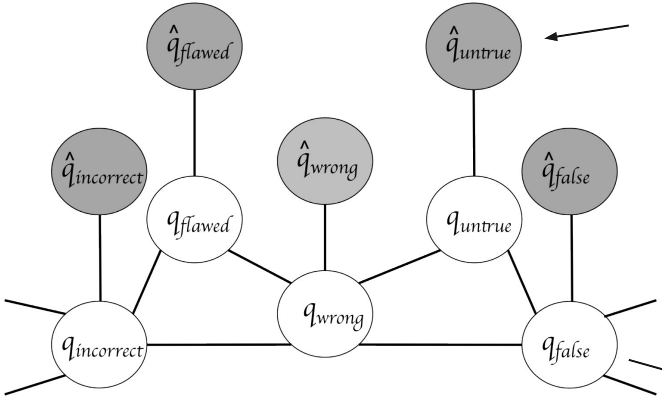


Figure 5: Word graph with edges between related words showing the observed and the inferred word vector representations.

Original embedding space structure we observe

$$\Psi(\mathbf{Q}) = \sum_{i=1}^{n} \left[\alpha_i \|\mathbf{q}_i - \hat{\mathbf{q}}_i\|^2 + \sum_{(i,j)\in E} \beta_{ij} \|\mathbf{q}_i - \mathbf{q}_j\|^2 \right]$$

Latent embedding space structure we seek to impose

Retrofitting (Faruqui et al., NAACL 2015)

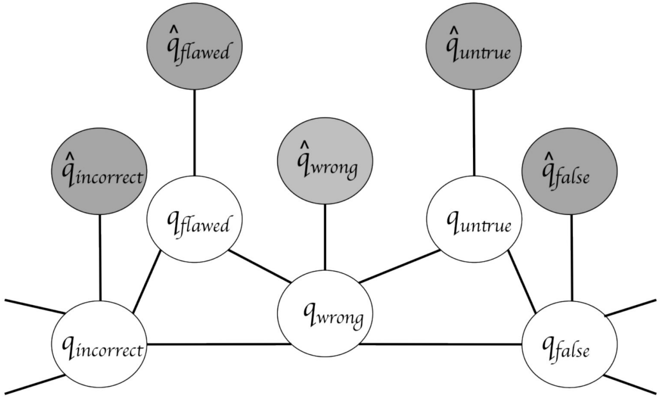


Figure 5: Word graph with edges between related words showing the observed and the inferred word vector representations.

New embeddings

$$\Psi(\mathbf{Q}) = \sum_{i=1}^{n} \left[\alpha_i \|\mathbf{q}_i - \hat{\mathbf{q}}_i\|^2 + \sum_{(i,j)\in E} \beta_{ij} \|\mathbf{q}_i - \mathbf{q}_j\|^2 \right]$$

1. Retain old structure (be close “enough” to original)

2. Reflect graph structure with the new embeddings

Retrofitting (Faruqui et al., NAACL 2015)

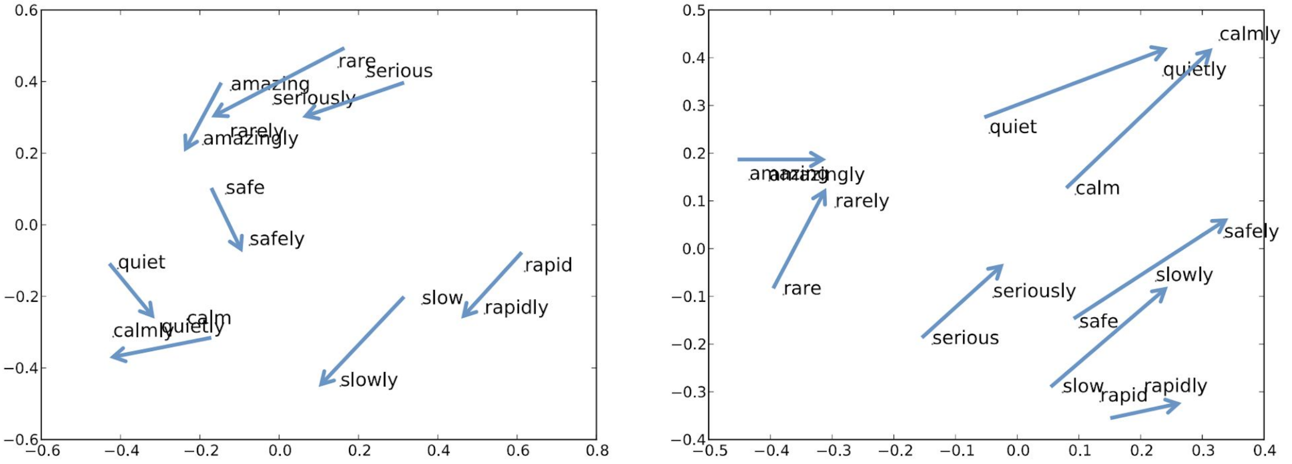


Figure 7: Two-dimensional PCA projections of 100-dimensional **SG** vectors of syntactic analogy “adjective to adverb” relation, before (left) and after (right) retrofitting.

Retrofitting (Faruqui et al., NAACL 2015)

Results & comments

- Improved results for Glove and Skip-gram embeddings for *instrisic evaluation*
    - E.g., semantic and syntactic analogies
- Marginal improvements in *one downstream task* - sentiment analysis (+1%)

Retrofitting (Faruqui et al., NAACL 2015)

Results & comments

- Improved results for Glove and Skip-gram embeddings for *instrisic evaluation*
    - E.g., semantic and syntactic analogies
- Marginal improvements in *one downstream task* - sentiment analysis (+1%)
- **Only factors in similarity!**
    - All semantic relations are encoded and optimized in the same way: *minimize the squared euclidean distance between their vectors*
- **Only is undirected!**
    - What about *directed relations* - hyper/hyponymy, etc.?

Counterfitting (Mrkšić et al., NAACL 2016)

- **Extension** of retrofitting
- Model synonymy *and antonymy*
- Works on level of **cosine similarity**, not euclidean distance.
    - **Attract** *synoynyms* together;
    - **Repulse** *antonyms* from each other;
    - **Preserve** the *semantic information* of *original vectors* as much as possible.

	east	**expensive**	**British**
Before	west	pricey	American
	north	cheaper	Australian
	south	costly	Britain
	southeast	overpriced	European
	northeast	inexpensive	England
After	eastward	costly	Brits
	eastern	pricy	London
	easterly	overpriced	BBC
	-	pricey	UK
	-	afford	Britain

Table 1: Nearest neighbours for target words using GloVe vectors before and after counter-fitting

Counterfitting (Mrkšić et al., NAACL 2016)

- **Extension** of retrofitting
- Model synonymy *and antonymy*
- Works on level of **cosine similarity**, not euclidean distance.
    - **Attract** *synoynyms* together;
    - **Repulse** *antonyms* from each other;
    - **Preserve** the *semantic information* of *original vectors* as much as possible.

Minimize distance

$$\mathrm{SA}(V') = \sum_{(u,w)\in S} \tau\left(d(\mathbf{v}'_u, \mathbf{v}'_w) - \gamma\right)$$

Maximize distance

$$\mathrm{AR}(V') = \sum_{(u,w)\in A} \tau\left(\delta - d(\mathbf{v}'_u, \mathbf{v}'_w)\right)$$

Preserve original distances

$$\mathrm{VSP}(V, V') = \sum_{i=1}^{N} \sum_{j\in N(i)} \tau\left(d(\mathbf{v}'_i, \mathbf{v}'_j) - d(\mathbf{v}_i, \mathbf{v}_j)\right)$$

Words in radial neighborhood

$$C(V, V') = k_1\mathrm{AR}(V') + k_2\mathrm{SA}(V') + k_3\mathrm{VSP}(V, V')$$

Counterfitting (Mrkšić et al., NAACL 2016)

Results & Commentary

- Only evaluated on an *intrinsic evaluation*
  - But improves performance on it
  - A dataset with 0.67 annotator agreement, they get 0.74 score...
- Reveals that *distributional hypothesis* is problematic because it will tend to conflate *semantic similarity* with *conceptual association*

	east	expensive	British
Before	west	pricey	American
	north	cheaper	Australian
	south	costly	Britain
	southeast	overpriced	European
	northeast	inexpensive	England
After	eastward	costly	Brits
	eastern	pricy	London
	easterly	overpriced	BBC
	-	pricey	UK
	-	afford	Britain

Table 1: Nearest neighbours for target words using GloVe vectors before and after counter-fitting

Functional Retrofitting (COLING 2018, Lengerich et al.)

- **Generalize** retrofitting & counterfitting!
- Encode **directed** relations in any **knowledge graph**!
- Represent & model **any type of relation**, *without a priori knowledge*
  - **Learns** the parameters of a semantic relation!
  - No longer impose everything to be similar!

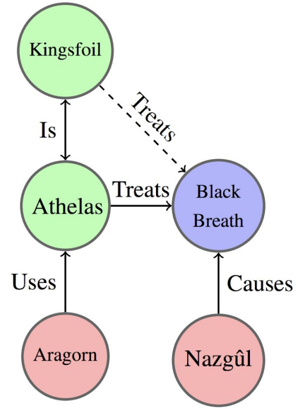


Figure 1: Toy knowledge graph with diverse relation types that connect treatments (green), diseases (blue), and persons (red) by known (solid) and unknown (dashed) relations. Traditional methods, which assume that all relations imply similarity, would retrofit Aragorn and Nazgûl toward similar embeddings.

Functional Retrofitting (COLING 2018, Lengerich et al.)

- Given a **knowledge graph** with directed edges of any kind of relations encoded:
  - G = (V, E) s.t. nodes *i* are words, and all edges *e = (i, j, r)* defines a specific directed *relation link r* between two words *i* and *j*
- **Objective:** augment set of word embeddings to *represent **and learn how to** represent* the relations in G

Functional Retrofitting (COLING 2018, Lengerich et al.)

- Given a **knowledge graph** with directed edges of any kind of relations encoded:
  - G = (V, E) s.t. nodes *i* are words, and all edges *e = (i, j, r)* defines a specific directed *relation link r* between two words *i* and *j*
- **Objective**: augment set of word embeddings to *represent **and learn how to represent*** the relations in G

Functional Retrofitting (COLING 2018, Lengerich et al.)

- Given a **knowledge graph** with directed edges of any kind of relations encoded:
  - G = (V, E) s.t. nodes *i* are words, and all edges *e = (i, j, r)* defines a specific directed *relation link r* between two words *i* and *j*
- **Objective**: augment set of word embeddings to *represent **and learn how to represent*** the relations in G

$$\Psi_{\mathcal{G}}(\mathcal{Q};\mathcal{F}) = \sum_{i\in\mathcal{Q}} \alpha_i ||\boldsymbol{q_i} - \hat{\boldsymbol{q}}_{\boldsymbol{i}}||^2 + \sum_{(i,j,r)\in\mathcal{E}} \beta_{i,j,r} f_r(\boldsymbol{q_i},\boldsymbol{q_j}) - \sum_{(i,j,r)\in\mathcal{E}^-} \beta_{i,j,r} f_r(\boldsymbol{q_i},\boldsymbol{q_j}) + \sum_{r\in\mathcal{R}} \rho_\lambda(f_r)$$

Preserve the original
vector space

Functional Retrofitting (COLING 2018, Lengerich et al.)

- Given a **knowledge graph** with directed edges of any kind of relations encoded:
  - G = (V, E) s.t. nodes *i* are words, and all edges *e = (i, j, r)* defines a specific directed *relation link r* between two words *i* and *j*
- **Objective**: augment set of word embeddings to *represent **and learn how to represent*** the relations in G

$$\Psi_{\mathcal{G}}(\mathcal{Q};\mathcal{F}) = \sum_{i\in\mathcal{Q}} \alpha_i ||\boldsymbol{q_i} - \boldsymbol{\hat{q}_i}||^2 + \sum_{(i,j,r)\in\mathcal{E}} \beta_{i,j,r} f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) - \sum_{(i,j,r)\in\mathcal{E}^-} \beta_{i,j,r} f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) + \sum_{r\in\mathcal{R}} \rho_\lambda(f_r)$$

Preserve the original vector space

Capture relations of the knowledge graph

Don’t capture nonexistent relations

Functional Retrofitting (COLING 2018, Lengerich et al.)

- Given a **knowledge graph** with directed edges of any kind of relations encoded:
  - G = (V, E) s.t. nodes *i* are words, and all edges *e = (i, j, r)* defines a specific directed *relation link r* between two words *i* and *j*
- **Objective**: augment set of word embeddings to *represent **and learn how to represent*** the relations in G

$$\Psi_{\mathcal{G}}(\mathcal{Q};\mathcal{F}) = \sum_{i\in\mathcal{Q}} \alpha_i ||\boldsymbol{q_i} - \hat{\boldsymbol{q}}_{\boldsymbol{i}}||^2 + \sum_{(i,j,r)\in\mathcal{E}} \beta_{i,j,r} f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) - \sum_{(i,j,r)\in\mathcal{E}^-} \beta_{i,j,r} f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) + \sum_{r\in\mathcal{R}} \rho_\lambda(f_r)$$

Preserve the original vector space

Capture relations of the knowledge graph

Don't capture nonexistent relations

Regularize the functional relation parameters

Functional Retrofitting (COLING 2018, Lengerich et al.)

- Given a **knowledge graph** with directed edges of any kind of relations encoded:
  - G = (V, E) s.t. nodes *i* are words, and all edges *e = (i, j, r)* defines a specific directed *relation link r* between two words *i* and *j*
- **Objective**: augment set of word embeddings to *represent **and learn how to represent*** the relations in G

$$\Psi_{\mathcal{G}}(\mathcal{Q};\mathcal{F}) = \sum_{i\in\mathcal{Q}}\alpha_i||\boldsymbol{q_i}-\hat{\boldsymbol{q}}_{\boldsymbol{i}}||^2 + \sum_{(i,j,r)\in\mathcal{E}}\beta_{i,j,r}f_r(\boldsymbol{q_i},\boldsymbol{q_j}) - \sum_{(i,j,r)\in\mathcal{E}^-}\beta_{i,j,r}f_r(\boldsymbol{q_i},\boldsymbol{q_j}) + \sum_{r\in\mathcal{R}}\rho_\lambda(f_r)$$

Preserve the original vector space

Capture relations of the knowledge graph

Don’t capture nonexistent relations

Regularize the functional relation parameters

Functional Retrofitting (COLING 2018, Lengerich et al.)

- *Main idea*: **learn** the relations as functions, *simultaneously with learning the embeddings!*
- Relations can have different parameterizations:
    - Linear - learn A_r and b_r

      $$f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) = \|\boldsymbol{A_r q_j} + \boldsymbol{b_r} - \boldsymbol{q_i}\|^2$$

    - "Neural" - learn A_r

      $$f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) = \sigma(\boldsymbol{q_i}^T \boldsymbol{A_r q_j})$$

Functional Retrofitting (COLING 2018, Lengerich et al.)

- *Main idea*: **learn** the relations as functions, *simultaneously with learning the embeddings!*
- Relations can have different parameterizations:
  - Linear - learn A_r and b_r

    $$f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) = \|\boldsymbol{A_r}\boldsymbol{q_j} + \boldsymbol{b_r} - \boldsymbol{q_i}\|^2$$

  - "Neural" - learn A_r

    $$f_r(\boldsymbol{q_i}, \boldsymbol{q_j}) = \sigma(\boldsymbol{q_i}^T \boldsymbol{A_r}\boldsymbol{q_j})$$

If $\boldsymbol{A_r} = \boldsymbol{I}$ and $\boldsymbol{b_r} = \boldsymbol{0}$, we have the original *retrofitting* method of Faruqui et al. 2015!

If $\boldsymbol{A_r} = -\boldsymbol{I}$ and $\boldsymbol{b_r} = \boldsymbol{0}$, we (more or less) have *counterfitting*!

What will the representations of the learned functional relations look like? Future work.

Functional Retrofitting (COLING 2018, Lengerich et al.)

Results & Commentary

- Many experiments were done using relations from *Framenet* and *Wordnet*
  - Evaluated on *link prediction* on the knowledge graphs
  - Linear and Neural seem better than baselines
  - This is an *intrinsic evaluation* on how well embeddings capture what they're trained for
- Very interesting proposal, but presented results are not very interesting!
  - What is the character of the learned relations?
    - E.g., how do learned hypo/hypernymy and syn/antonymy relations differ in weights?
    - Do they make sense?
  - Does this improve performance for downstream tasks?

Functional Retrofitting (COLING 2018, Lengerich et al.)

Results & Commentary

- Many experiments were done using relations from *Framenet* and *Wordnet*
  - Evaluated on *link prediction* on the knowledge graphs
  - Linear and Neural seem better than baselines
  - This is an *intrinsic evaluation* on how well embeddings capture what they're trained for
- Very interesting proposal, but presented results are not very interesting!
  - What is the character of the learned relations?
    - E.g., how do learned hypo/hypernymy and syn/antonymy relations differ in weights?
    - Do they make sense?
  - Does this improve performance for downstream tasks?

Problems with Evaluating Word Embeddings

REPEVAL 2016 - workshop on word representations, many influential papers

- *Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance*, Chiu et al.
    - Intrinsic evaluation = *analogy questions* ("man is to woman as king is to ___"), *word similarity* problems, 8 datasets tested
    - **Zero (or negative correlation)** between *intrinsic performance* and *downstream performance* on POS-tagging, NER, and chunking
        - *Except on* ***SimLex-999***, high correlation!
        - Likely because SimLex-999 distingustishes between *conceptual relatedness* and *semantic similarity*. E.g., (film, cinema) = related, *not* similar; (male, man) = both
- *Problems with Evaluation of Word Embeddings Using Word Similarity Tasks*, Faruqui et al.
    - Review of the problems of these datasets, discuss the problems of ***polysemy***, *subjectivity of "similarity"*, ***low correlation with extrinsic***, *frequency problems of embeddings*

Problems with Evaluating Word Embeddings

REPEVAL 2016 - workshop on word representations, many influential papers

- *Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance*, Chiu et al.
    - Intrinsic evaluation = *analogy questions* ("man is to woman as king is to ___"), *word similarity* problems, 8 datasets tested
    - **Zero (or negative correlation)** between *intrinsic performance* and *downstream performance* on POS-tagging, NER, and chunking
        - *Except on **SimLex-999***, high correlation!
        - Likely because SimLex-999 distingustishes between *conceptual relatedness* and *semantic similarity*. E.g., (film, cinema) = related, *not* similar; (male, man) = both
- *Problems with Evaluation of Word Embeddings Using Word Similarity Tasks*, Faruqui et al.
    - Review of the problems of these datasets, discuss the problems of ***polysemy***, *subjectivity of "similarity"*, ***low correlation with extrinsic***, *frequency problems of embeddings*

Problems with Evaluating Word Embeddings

REPEVAL 2016 - workshop on word representations, many influential papers

- *Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance*, Chiu et al.
  - Intrinsic evaluation = *analogy questions* ("man is to woman as king is to ___"), *word similarity* problems, 8 datasets tested
  - **Zero (or negative correlation)** between *intrinsic performance* and *downstream performance* on POS-tagging, NER, and chunking
    - *Except on **SimLex-999***, high correlation!
    - Likely because SimLex-999 distingustishes between *conceptual relatedness* and *semantic similarity*. E.g., (film, cinema) = related, *not* similar; (male, man) = both
- *Problems with Evaluation of Word Embeddings Using Word Similarity Tasks*, Faruqui et al.
  - Review of the problems of these datasets, discuss the problems of ***polysemy***, *subjectivity of "similarity"*, ***low correlation with extrinsic***, *frequency problems of embeddings*

Frequency “Pollution” in Word Embeddings

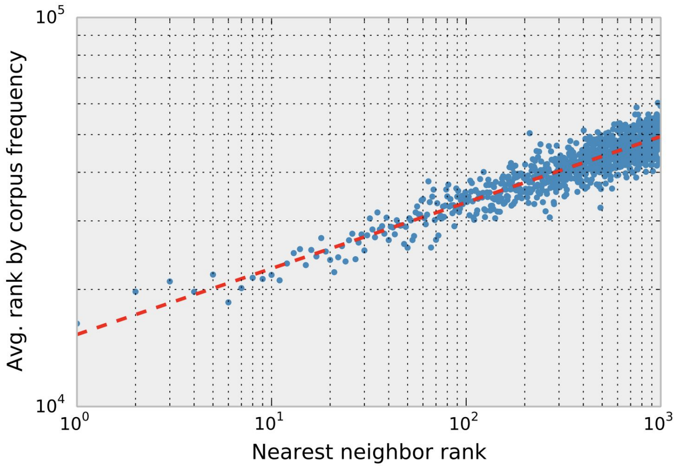


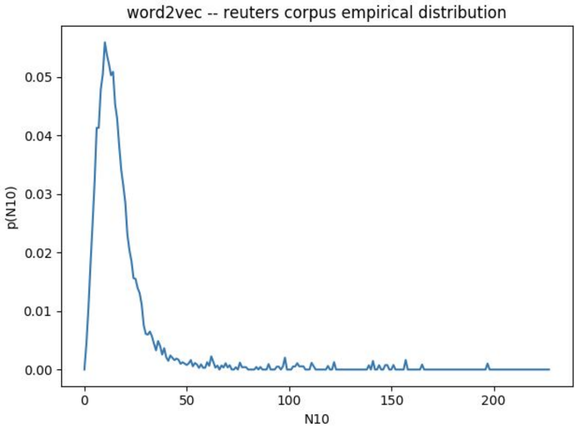
Figure 4: Avg. word rank by frequency in training corpus vs. nearest-neighbor rank in the C&W embedding space.

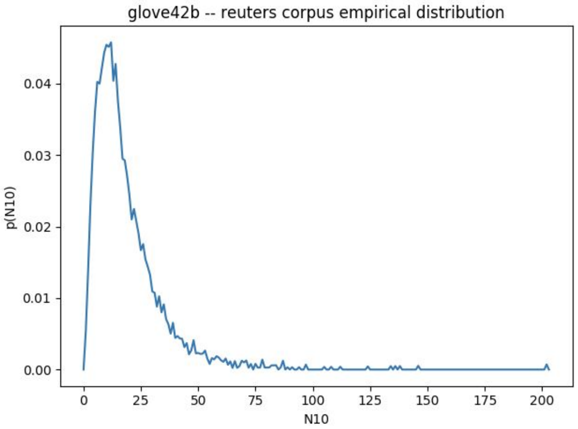
Translation: “words tend to be surrounded in the embedding space by words with similar frequencies in the corpus they are trained upon.”

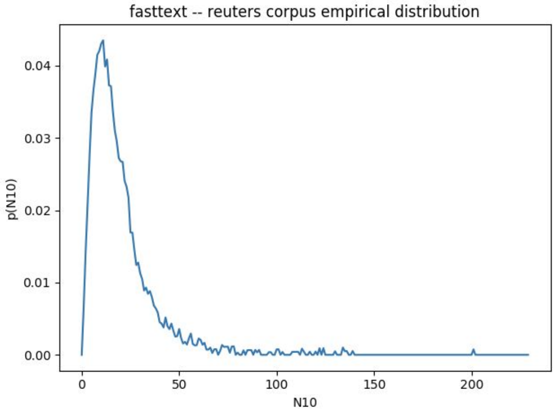
a power law relationship for C&W embeddings between a word’s nearest neighbor rank (w.r.t. a query) and the word’s frequency rank in the training corpus (nn-rank $\sim 1000 \cdot$ corpus-rank$^{0.17}$).

Hubness “Pollution” in Word Embeddings

In a KNN graph of an embedding space, there exist **hubs**, words that are the nearest neighbor to *many many* other words - often tend to be *names* and *helping words* (“really”, “anyway”, etc.)







Different methods, summary

- Matrix factorization: if you want to be super mathematical and make proofs
- Neural language models: if you want to spend a lot of time and obfuscate linearity (shouldn't do this)
- Word2vec, **CBOW**: need good **syntactic** representations, need **fast** training
- Word2vec, Skip-gram: need good **semantic** representations, need good representations of **rare words**
- Glove: need good, **well-rounded** representations -- great default embeddings
- Fasttext: have lots of **morphological** data, need to capture this well
- Retrofitting: need to capture **basic** features of a **knowledge graph** in your embeddings
- Counterfitting: need to capture **synonymy** and **antonymy** explicitly in your embeddings
- Functional retrofitting: need to capture **complex** features of a **knowledge graph** with many relations

References

- *HAL (matrix factorization)*: Producing high-dimensional semantic spaces from lexical co-occurrence, Lund & Burgess (1996).
- *HPCA (matrix factorization)*: Word Embeddings through Hellinger PCA, Lebret & Collobert (2014).
- *NNLMs*: A Neural Probabilistic Language Model, Bengio et al. (2003). Good *blog post* about this & new techniques (2017).
- *Word2vec*: ICLR paper with CBOW (2013); NIPS paper with Hierarchical Softmax and Negative Sampling (2013).
- *Glove*: Global Vectors for Word Representation, Stanford. (2015)
- *Fasttext*: Enriching Word Vectors with Subword Information, Facebook. (2017)
- *Retrofitting*: Retrofitting Word Vectors to Semantic Lexicons, Faruqui et al. (2015)
- *Counterfitting*: Counter-fitting Word Vectors to Linguistic Constraints, Mrksic et al. (2016)
- *Functional retro*: Retrofitting Distributional Embeddings to Knowledge Graphs with Functional Relations, Lengerich et al. (2018)
- *Problems 1*: Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance, Chiu et al. (2016)
- *Problems 2*: Problems With Evaluation of Word Embeddings Using Word Similarity Tasks, Faruqui et al. (2016)
- *All the problems!* RepEval 2016, famous workshop on word embeddings, accepted papers.
- *Evaluation methods*: Evaluation methods for unsupervised word embeddings, Schnabel et al. (2016)

Thank you!

- Kian