Representation Discovery

(Slides by Piotr Mirowski, Hugo Larochelle, Omer Levy, Yoav Goldberg, Graham Neubig, and Tomas Mikolov)

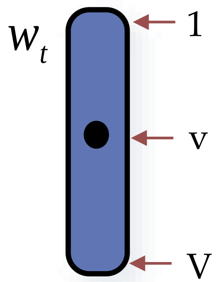
Distributed Representation

- Each word is associated with a continuous valued vector

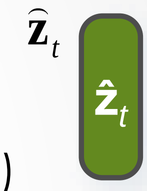
Word	w	$C(w)$
"the"	1	[0.6762, -0.9607, 0.3626, -0.2410, 0.6636]
"a"	2	[0.6859, -0.9266, 0.3777, -0.2140, 0.6711]
"have"	3	[0.1656, -0.1530, 0.0310, -0.3321, -0.1342]
"be"	4	[0.1760, -0.1340, 0.0702, -0.2981, -0.1111]
"cat"	5	[0.5896, 0.9137, 0.0452, 0.7603, -0.6541]
"dog"	6	[0.5965, 0.9143, 0.0899, 0.7702, -0.6392]
"car"	7	[-0.0069, 0.7995, 0.6433, 0.2898, 0.6359]

Vector-space representation of words

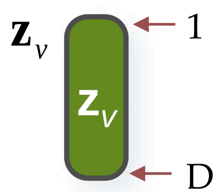
"**One-hot**" of "**one-of-V**" representation
of a word token at position *t*
in the text corpus,
with **vocabulary of size *V***



Vector-space representation
of the prediction
of **target word w_t**
(we predict a vector of size *D*)



Vector-space representation
of any word *v*
in the vocabulary
using a vector of **dimension *D***



Also called
distributed representation

Vector-space representation
of the **t^{th} word history**:
e.g., concatenation
of *n*-1 vectors of size *D*

Predictive

- Input:
    - word history/context (**one-hot** or **distributed representation**)
- Output:
    - target word(s) (**one-hot** or **distributed representation**)
- **Function** that **approximates word likelihood**:
    - **Continuous bag-of-words**
    - **Skip-gram**
    - …

Learning continuous space models

- How do we **learn the word representations z** for each word in the vocabulary?
- How do we **learn the model** that predicts a word or its representation $\hat{z}_t$ given a word context?

- Simultaneous learning of **model** and **representation**

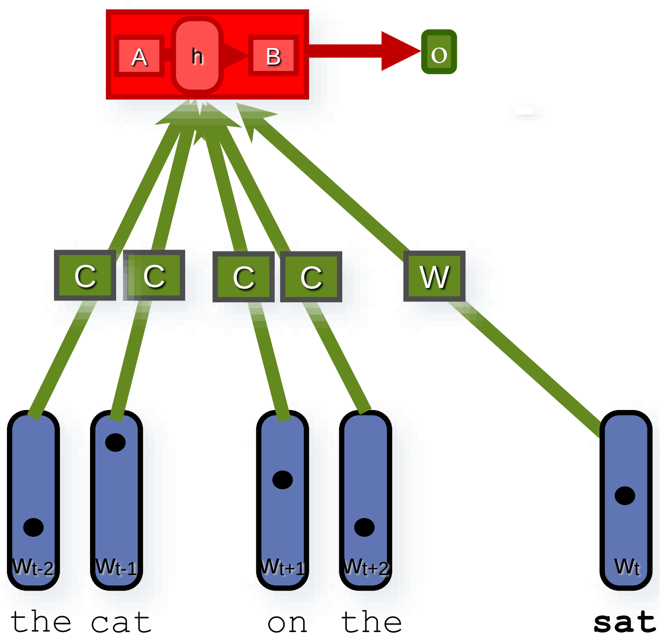
Collobert & Weston

Prediction network: 2 layer network outputting a scalar

word embedding space $\Re^D$ in dimension D=100 to 300

Word embedding matrices

discrete word space {1, ..., *V*} *V*>100k words



$$P\left(w_t \mid \mathbf{w}_{t-c}^{t-1}, \mathbf{w}_{t+1}^{t+c}\right) = \frac{e^{o(w)}}{\sum_v e^{o(v)}}$$

Solution: negative sampling
Max margin Loss:

max{0, 1-(o(w)-o(w'))}

Parameters: (2?)DxV + (2c+1)DxH + Hx1
Denominator: Iterate over V <then not feasible>

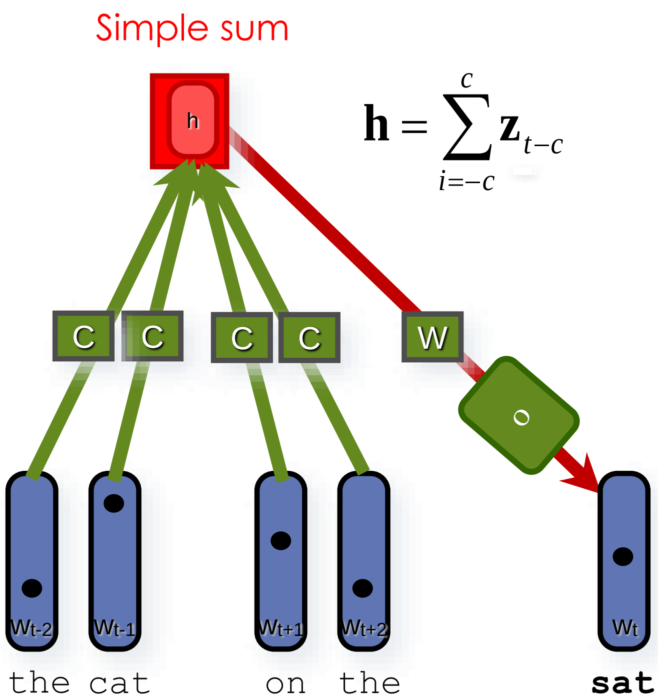
[Mikolov et al, 2013a; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Continuous Bag-of-Words

word embedding space $\Re^D$ in dimension D=100 to 300

Word embedding matrices

discrete word space {1, ..., V} V>100k words



$$\mathbf{h} = \sum_{i=-c}^{c} \mathbf{z}_{t-c}$$

$$\mathbf{o} = \mathbf{W}\mathbf{h}$$

$$P\left(w_t \mid \mathbf{w}_{t-c}^{t-1}, \mathbf{w}_{t+1}^{t+c}\right) = \frac{e^{o(w)}}{\sum_v e^{o(v)}}$$

Parameters: 2DxV

Problem: large output space!

[Mikolov et al, 2013a; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Aside

- Sum of vectors of words is a good baseline embedding for a short document
  - Short document = a bag of words since position information is lost

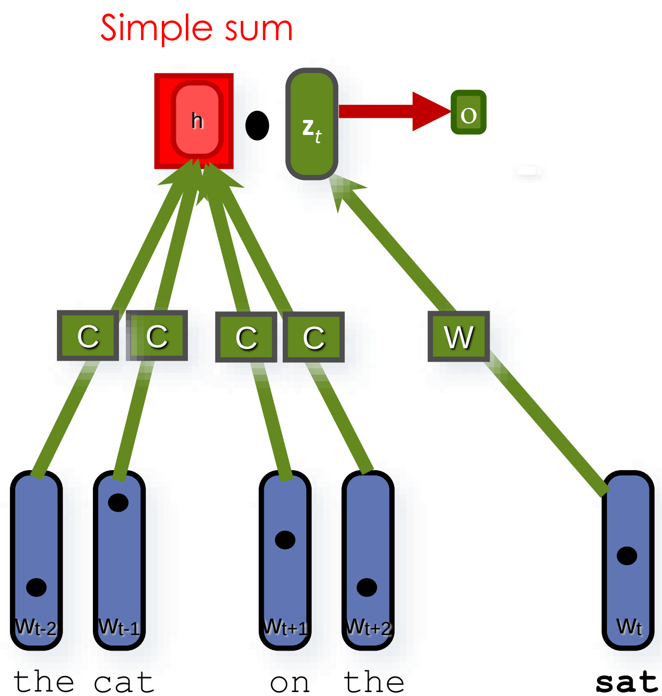
- See Section 11.6 (Goldberg) for the computation of document similarity

Continuous Bag-of-Words

word embedding space $\Re^D$ in dimension D=100 to 300

Word embedding matrices

discrete word space {1, ..., V} V>100k words



$$\mathbf{h} = \sum_{i=-c}^{c} \mathbf{z}_{t-c}$$

$$\mathbf{o}=\mathbf{h}.\mathbf{z}_t$$

Negative sampling for scalability (6B words)

$$\Pr(D=1|c)=\sigma(\mathbf{h}.\mathbf{w})$$
$$\Pr(D=0|c)=\sigma(-\mathbf{h}.\mathbf{w}')$$

Parameters: 2DxV

good word+context pairs

bad word+context pairs

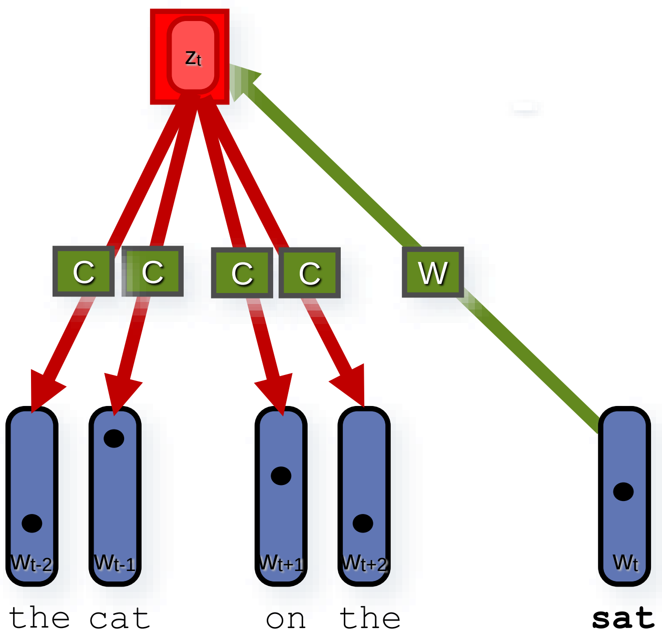
$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D=1|w,c) + \sum_{(w',c)\in \bar{D}} \log P(D=0|w',c)$$

Skip-gram

word embedding space $\Re^D$ in dimension D=100 to 1000

Word embedding matrices

discrete word space {1, ..., V} V>100k words



$o = z_{t+i} . z_t$ i=-2,-1,1,2

Parameters: 2DxV

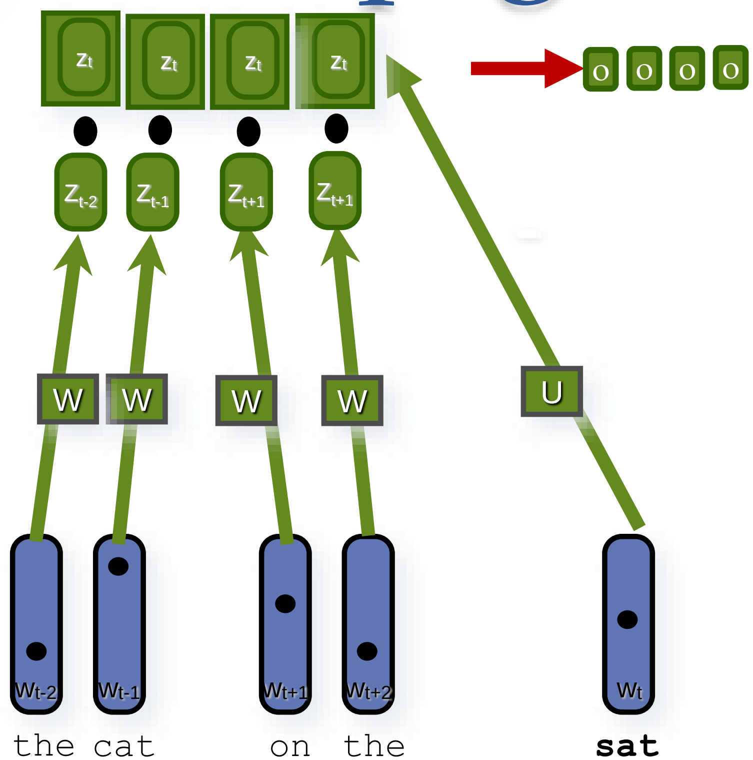
[Mikolov et al, 2013a, 2013b; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Skip-gram

word embedding space $\Re^D$ in dimension D=100 to 1000

Word embedding matrices

discrete word space {1, ..., V} V>100k words



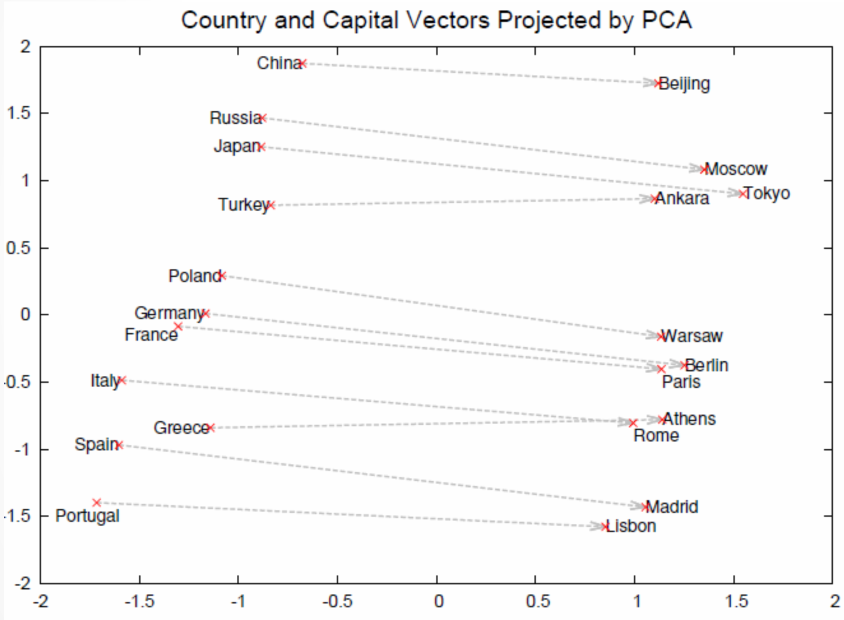
$o = z_{t+i} . z_t \quad i=-2,-1,1,2$

$Pr(D=1 | c) = \sigma(c_i . w)$
$Pr(D=0 | c) = \sigma(-c_i . w')$

Parameters: 2DxV
(Scales to 33B words)

[Mikolov et al, 2013a, 2013b; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Vector-space word representation without LM



[Image credits: Mikolov et al (2013) "Distributed Representations of Words and Phrases and their Compositionality", *NIPS*]

Word and phrase representation learned by skip-gram **exhibit linear structure** that enables **analogies with vector arithmetics**.

This is **due to training objective**, input and output (before softmax) are in **linear relationship**.

The sum of vectors in the loss function is the sum of log-probabilities (or log of product of probabilities), i.e., comparable to the AND function.

[Mikolov et al, 2013a, 2013b; http://code.google.com/p/word2vec]

Examples of Word2Vec embeddings

Example of word embeddings obtained using Word2Vec on the 3.2B word Wikipedia:

- Vocabulary V=2M
- Continuous vector space D=200
- Trained using CBOW

debt	aa	decrease	met	slow	france	jesus	xbox
debts	aaarm	increase	meeting	slower	marseille	christ	playstation
repayments	samavat	increases	meet	fast	french	resurrection	wii
repayment	obukhovskii	decreased	meets	slowing	nantes	savior	xbla
monetary	emerlec	greatly	had	slows	vichy	miscl	wiiware
payments	gunss	decreasing	welcomed	slowed	paris	crucified	gamecube
repay	dekhen	increased	insisted	faster	bordeaux	god	nintendo
mortgage	minizini	decreases	acquainted	sluggish	aubagne	apostles	kinect
repaid	bf	reduces	satisfied	quicker	vend	apostle	dsiware
refinancing	mortardepth	reduce	first	pace	vienne	bickertonite	eshop
bailouts	ee	increasing	persuaded	slowly	toulouse	pretribulational	dreamcast

[Mikolov et al, 2013a, 2013b; http://code.google.com/p/word2vec]

Semantic-syntactic word evaluation task

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

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
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

[Image credits: Mikolov et al (2013) "Efficient Estimation of Word Representation in Vector Space", *arXiv*]

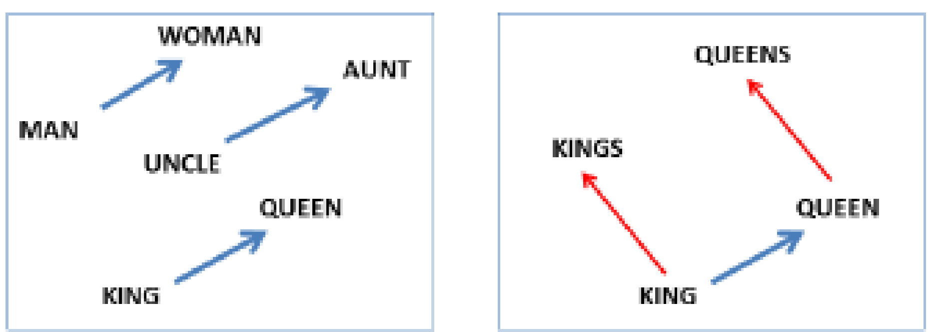
[Mikolov et al, 2013a, 2013b; http://code.google.com/p/word2vec]

Syntactic and Semantic tests

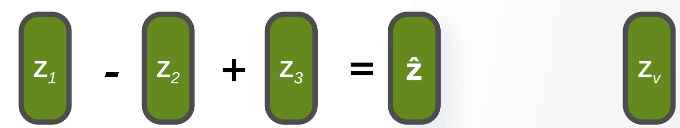
Observed that word embeddings obtained by RNN-LDA have linguistic regularities "a" is to "b" as "c" is to _

Syntactic: king is to kings as queen is to **queens**

Semantic: clothing is to shirt as dish is to **bowl**



Vector offset method



cosine similarity

$$\arg\max_{b^* \in V} \left(\cos\left(b^*, b - a + a^*\right)\right)$$

$$\arg\max_{b^* \in V} \frac{\cos\left(b^*, b\right)\cos\left(b^*, a^*\right)}{\cos\left(b^*, a\right) + \varepsilon}$$

$$\arg\max_{b^* \in V} \left(\cos\left(b^*, b\right) - \cos\left(b^*, a\right) + \cos\left(b^*, a^*\right)\right)$$

3]

Linguistic Regularities - Examples

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

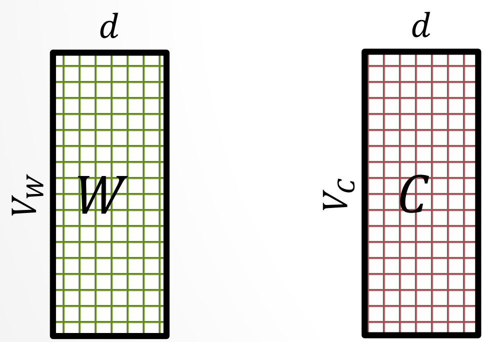
What is `word2vec`?

- `word2vec` is **not** a single algorithm
- It is a **software package** for representing words as vectors, containing:
  - Two distinct models
    - CBoW
    - **Skip-Gram** **(SG)**
  - Various training methods
    - **Negative Sampling** **(NS)**
    - Hierarchical Softmax
  - A rich preprocessing pipeline
    - Dynamic Context Windows
    - Subsampling
    - Deleting Rare Words

What is SGNS learning?

What is SGNS learning?

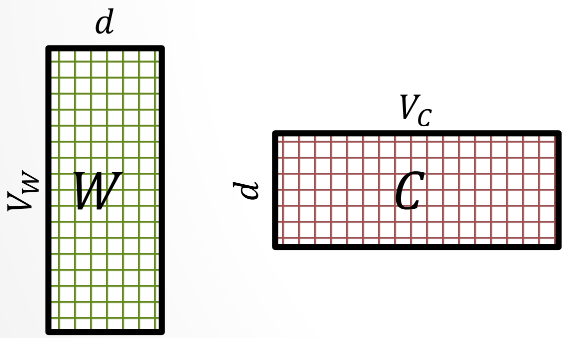
- Take SGNS's embedding matrices (W and C)



"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

What is SGNS learning?

- Take SGNS's embedding matrices (W and C)
- Multiply them
- What do you get?

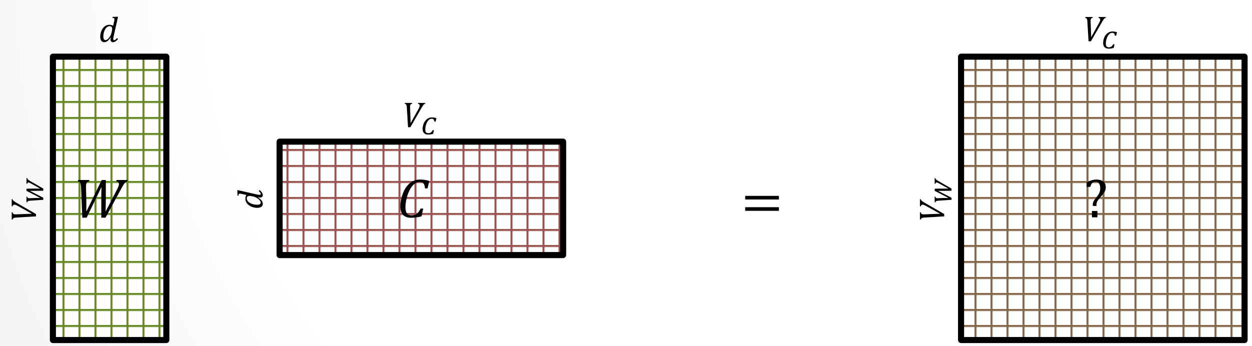


"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

What is SGNS learning?

- A $V_W \times V_C$ matrix
- Each cell describes the relation between a specific word-context pair

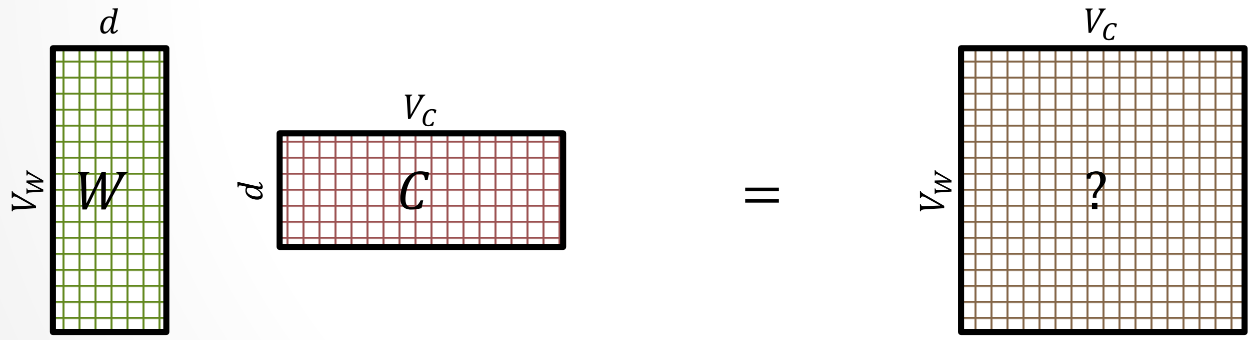
$$\vec{w} \cdot \vec{c} = ?$$



"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

What is SGNS learning?

- We **prove** that for large enough d and enough iterations

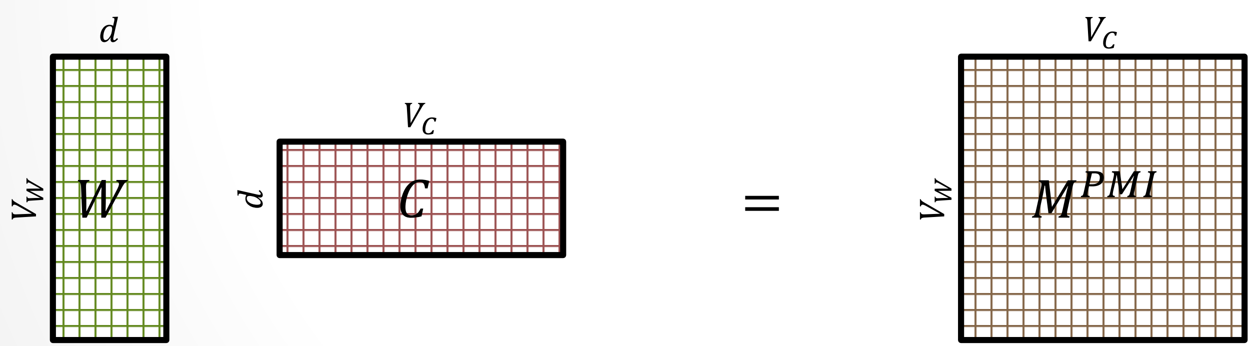


"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

What is SGNS learning?

- We **prove** that for large enough d and enough iterations
- We get the word-context PMI matrix

$$\text{PMI (w, c)} = \frac{log\#(\text{w,c})}{\log(\#w){*}\log(\#c)}$$

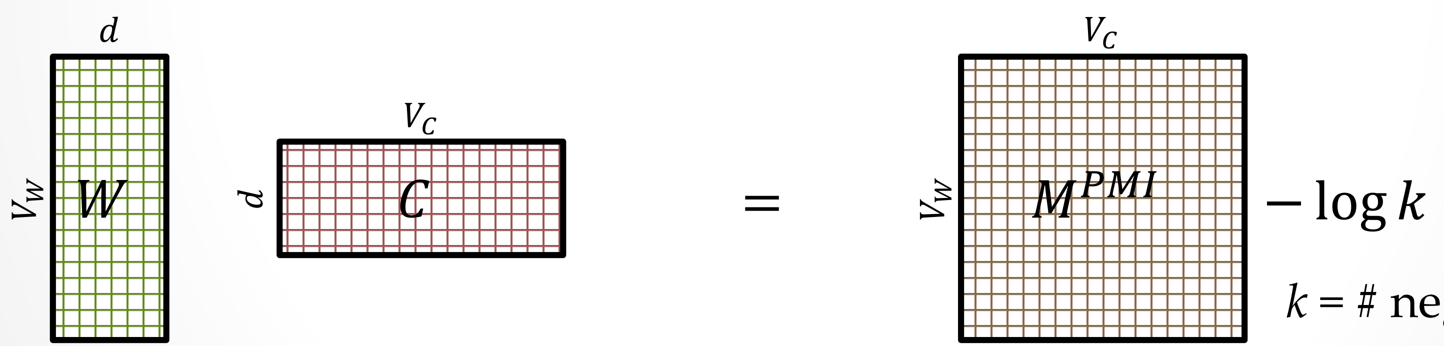


"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

What is SGNS learning?

- We **prove** that for large enough d and enough iterations
- We get the word-context PMI matrix, shifted by a global constant

$$Opt(\vec{w} \cdot \vec{c}) = PMI(w, c) - \log k$$



k = # neg. samples

"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

GLOVE

- SGNS

$$\vec{w} \cdot \vec{c} = \mathrm{PMI}(w, c) - \log k$$

$$\ell = \sum_{w \in V_W} \sum_{c \in V_C} \#(w,c) \left(\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)]\right)$$

- GLOVE

$$\vec{w} \cdot \vec{c} + b_w + b_c = \log\left(\#(w,c)\right) \quad \forall (w,c) \in D$$

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

Follow up work

Baroni, Dinu, Kruszewski (2014): Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors

- Turns out neural based approaches are very close to traditional distributional semantics models
- Luckily, word2vec significantly outperformed the best previous models across many tasks ☺

- How to reconcile good results ???

The Big Impact of "Small" Hyperparameters

- `word2vec` & GloVe are more than just algorithms…
- Introduce **new hyperparameters**
- May seem minor, but **make a big difference** in practice

New Hyperparameters

- **Preprocessing** **(word2vec)**
  - Dynamic Context Windows
  - Subsampling
  - Deleting Rare Words

- **Postprocessing** **(GloVe)**
  - Adding Context Vectors

- **Association Metric** **(SGNS)**
  - Shifted PMI
  - Context Distribution Smoothing

New Hyperparameters

- **Preprocessing** **(word2vec)**
  - Dynamic Context Windows
  - Subsampling
  - Deleting Rare Words

- **Postprocessing** **(GloVe)**
  - Adding Context Vectors

- **Association Metric** **(SGNS)**
  - Shifted PMI
  - Context Distribution Smoothing

New Hyperparameters

- **Preprocessing** **(word2vec)**
  - Dynamic Context Windows
  - Subsampling
  - Deleting Rare Words
- **Postprocessing** **(GloVe)**
  - Adding Context Vectors
- **Association Metric** **(SGNS)**
  - Shifted PMI
  - Context Distribution Smoothing

New Hyperparameters

- **Preprocessing** **(word2vec)**
  - Dynamic Context Windows
  - Subsampling
  - Deleting Rare Words
- **Postprocessing** **(GloVe)**
  - Adding Context Vectors
- **Association Metric** **(SGNS)**
  - Shifted PMI
  - Context Distribution Smoothing

Dynamic Context Windows

Marco saw a furry little wampimuk hiding in the tree.

Dynamic Context Windows

saw a furry little wampimuk hiding in the tree.

Dynamic Context Windows

saw a furry little wampimuk hiding in the tree.

word2vec:	$\frac{1}{4}$	$\frac{2}{4}$	$\frac{3}{4}$	$\frac{4}{4}$		$\frac{4}{4}$	$\frac{3}{4}$	$\frac{2}{4}$	$\frac{1}{4}$
GloVe:	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{1}$		$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$
Aggressive:	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{1}$		$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

The Word-Space Model *(Sahlgren, 2006)*

Adding Context Vectors

- SGNS creates word vectors $\vec{w}$
- SGNS creates auxiliary context vectors $\vec{c}$
  - So do GloVe and SVD

Adding Context Vectors

- SGNS creates word vectors $\vec{w}$
- SGNS creates auxiliary context vectors $\vec{c}$
  - So do GloVe and SVD

- Instead of just $\vec{w}$
- Represent a word as: $\vec{w} + \vec{c}$

- Introduced by Pennington et al. (2014)
- Only applied to GloVe

Context Distribution Smoothing

- SGNS samples $c' \sim P$ to form **negative** (w, c') examples

- Our analysis assumes P is the unigram distribution

$$P\left(c\right) = \frac{\#c}{\sum_{c' \in V_C} \#c'}$$

Context Distribution Smoothing

- SGNS samples $c' \sim P$ to form **negative** (w, c') examples
- Our analysis assumes P is the unigram distribution
- In practice, it's a **smoothed** unigram distribution

$$P^{0.75}(c) = \frac{(\#c)^{0.75}}{\sum_{c' \in V_C} (\#c')^{0.75}}$$

- This little change makes a big difference

Context Distribution Smoothing

- We can **adapt** context distribution smoothing to PMI!
- Replace $P(c)$ with $P^{0.75}(c)$:

$$PMI^{0.75}(w,c) = \log \frac{P(w,c)}{P(w) \cdot \boldsymbol{P^{0.75}(c)}}$$

- Consistently improves **PMI** on **every task**
- **Always use Context Distribution Smoothing!**

Comparing Algorithms

Controlled Experiments

- Prior art was unaware of these hyperparameters
- Essentially, comparing “apples to oranges”
- We allow **every algorithm** to use **every hyperparameter**

Controlled Experiments

- Prior art was unaware of these hyperparameters
- Essentially, comparing “apples to oranges”
- We allow **every algorithm** to use **every hyperparameter***

* If transferable

Systematic Experiments

- 9 Hyperparameters
  - 6 New
- 4 Word Representation Algorithms
  - PPMI (Sparse & Explicit)
  - SVD(PPMI)
  - SGNS
  - GloVe
- 8 Benchmarks
  - 6 Word Similarity Tasks
  - 2 Analogy Tasks
- **5,632 experiments**

Systematic Experiments

- 9 Hyperparameters
  - 6 New
- 4 Word Representation Algorithms
  - PPMI (Sparse & Explicit)
  - SVD(PPMI)
  - SGNS
  - GloVe
- 8 Benchmarks
  - 6 Word Similarity Tasks
  - 2 Analogy Tasks
- **5,632 experiments**

Hyperparameter Settings

Classic Vanilla Setting

(commonly used for distributional baselines)

- Preprocessing
  - <None>
- Postprocessing
  - <None>
- Association Metric
  - Vanilla PMI/PPMI

Hyperparameter Settings

Classic Vanilla Setting

(commonly used for distributional baselines)

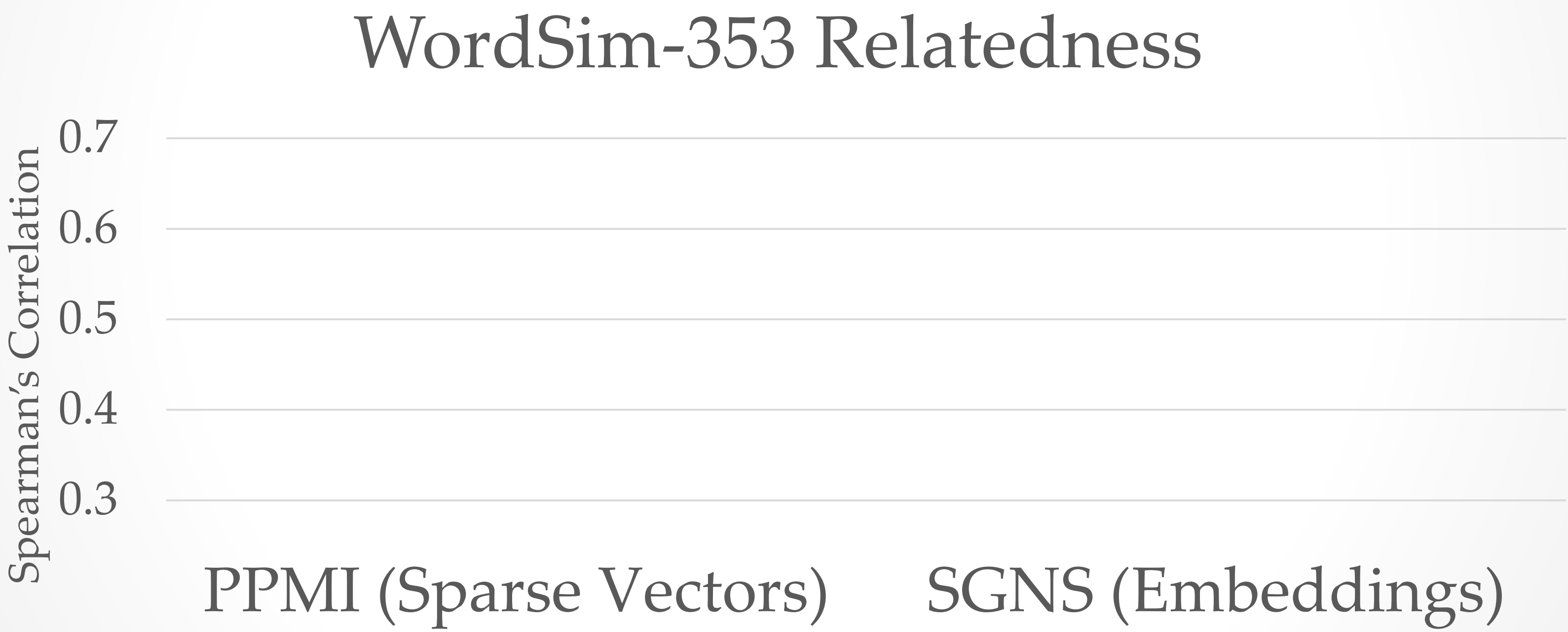
- Preprocessing
  - <None>
- Postprocessing
  - <None>
- Association Metric
  - Vanilla PMI/PPMI

Recommended word2vec Setting

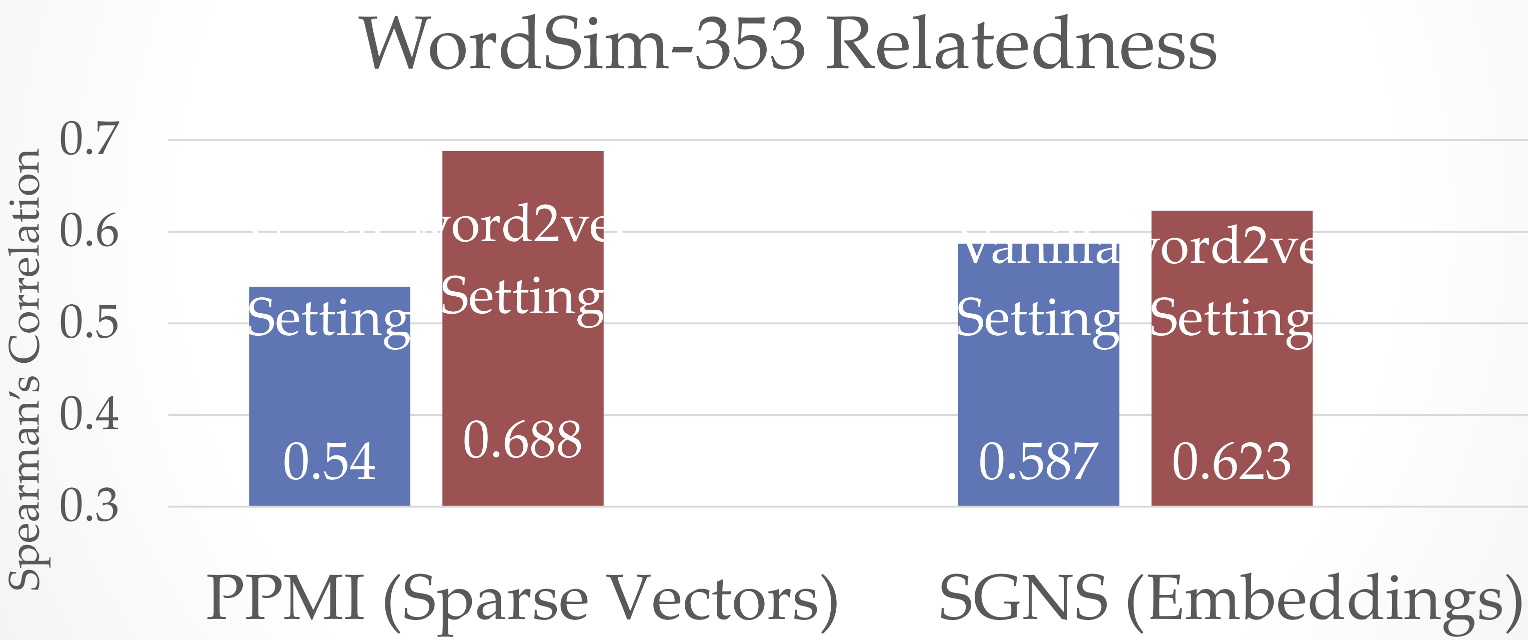
(tuned for SGNS)

- Preprocessing
  - Dynamic Context Window
  - Subsampling
- Postprocessing
  - <None>
- Association Metric
  - Shifted PMI/PPMI
  - Context Distribution Smoothing

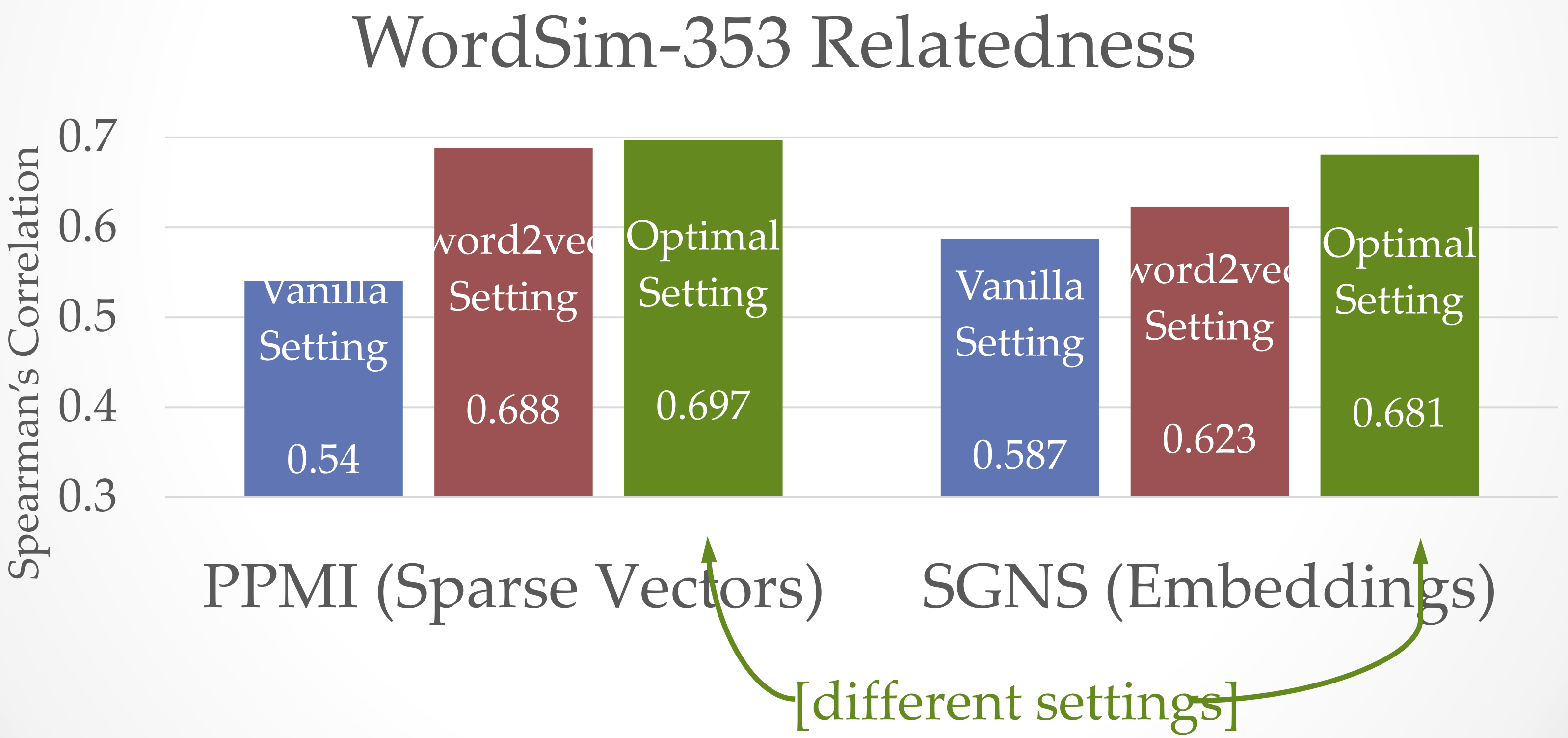
Experiments



Experiments: "Oranges to Oranges"



Experiments: Hyperparameter Tuning



Overall Results

- **Hyperparameters** often have stronger effects than **algorithms**
- **Hyperparameters** often have stronger effects than **more data**
- **Prior superiority claims** were not exactly accurate

Note on Dot Product

- We have been using $c^T w$ as the similarity score
- In case c and w come from different spaces
  one can use $c^T U w$ as the score
  where parameters of U matrix are also learnt
- Equivalent to projecting c in w space.

Domain Adaptation of Embeddings

- Pretrained embeddings W
  - And small new corpus
- Method 1
  - Fine tune all embeddings of W in a task-specific manner
  - Problem: only words in small dataset get changed
- Method 2
  - Learn a projection T. W' = WT
  - Problem: can't separate close-by words
- Method 3
  - Learn a full new vector U. W' = WT+U
  - Problem: need more data

Other Details

- Padding
  - Zero
  - Padding embedding
- Unknown Words
  - Unk embedding
- Word Dropout
  - randomly replace words with Unk
  - Use a/(a+#w) as dropout rate
- Word Dropout as regularization
  - Dropout rate not dependent on #w

Limitations of Distributional Similarity

- What kind of similarity is hard to ~control?
  - Small context: more syntax-based embedding
  - Large context: more topical embeddings
  - Context based on parses: more functional embeddings
- Sensitive to superficial differences
  - Dog/dogs
- Black sheep
  - People don't say the obvious
- Antonyms
- Corpus bias
  - "encode every kind of psychological bias we can look for"
  - Females<->family and not career;
- Lack of context
  - See Elmo [2018]
- Not interpretable

Retrofitting Embeddings

- Additional evidence – e.g., Wordnet
- Graph: nodes – words, edges – related
- New objective: find matrix $\hat{W}$ such that
  - $\hat{w}$ is close to W for each word
  - $\hat{w}$ of words related in the graph is close

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

De-biasing Embeddings

(Bolukbasi etal 16)

Extreme *she*	**Extreme *he***
1. homemaker	1. maestro
2. nurse	2. skipper
3. receptionist	3. protege
4. librarian	4. philosopher
5. socialite	5. captain
6. hairdresser	6. architect
7. nanny	7. financier
8. bookkeeper	8. warrior
9. stylist	9. broadcaster
10. housekeeper	10. magician

	Gender stereotype *she-he* analogies	
sewing-carpentry	registered nurse-physician	housewife-shopkeeper
nurse-surgeon	interior designer-architect	softball-baseball
blond-burly	feminism-conservatism	cosmetics-pharmaceuticals
giggle-chuckle	vocalist-guitarist	petite-lanky
sassy-snappy	diva-superstar	charming-affable
volleyball-football	cupcakes-pizzas	lovely-brilliant
	Gender appropriate *she-he* analogies	
queen-king	sister-brother	mother-father
waitress-waiter	ovarian cancer-prostate cancer	convent-monastery

Identify pairs to "neutralize", find the direction of the trait to neutralize, and ensure that they are neutral in that direction

Issues with Word2Vec and Glove

- Learning one embedding for each word in training data
- What to do with words missing in training data?

Issues with Word2Vec and Glove

- Learning one embedding for each word in training data

- What to do with words missing in training data?

- Option 1: Learn UNK embedding
- Replace words occurring only once or twice in the training data with UNK

Issues with Word2Vec and Glove

- Option 1: Learn UNK embedding
- Replace words occurring only once or twice in the training data with UNK

Issues with Word2Vec and Glove

- Option 1: Learn UNK embedding
- Replace words occurring only once or twice in the training data with UNK

- Issues:
- Loss of information
- Not using rich internal structure present in words - Morphology

- We can have a rough idea of Embedding('*taller*') from Embedding('*tall*')

Fasttext Representations

Enriching Word Vectors with Subword Information

Piotr Bojanowski* and **Edouard Grave*** and **Armand Joulin** and **Tomas Mikolov**
Facebook AI Research
{bojanowski,egrave,ajoulin,tmikolov}@fb.com

Fasttext Representations

- Train embedding for character n-grams
    - artificial: <ar, art, rti, tif, ifi, fic, ici, ial, al>

Fasttext Representations

- Train embedding for character n-grams
- Embedding of word = Sum of embedding of character n-grams

Fasttext Representations

- Train embedding for character n-grams
- Embedding of word = Sum of embedding of character n-grams
- Train skip-gram model based on these embeddings

Fasttext Representations

- Train embedding for character n-grams
- Embedding of word = Sum of embedding of character n-grams
- Train skip-gram model based on these embeddings
- Output: Learnt character n-gram embeddings

Fasttext Representations

- Train embedding for character n-grams
- Embedding of word = Sum of embedding of character n-grams
- Train skip-gram model based on these embeddings
- Output: Learnt character n-gram embeddings
- Unknown words - divide into constituent character n-grams
- Sum their embeddings

Document Embeddings

Document as Bag of Word Vectors

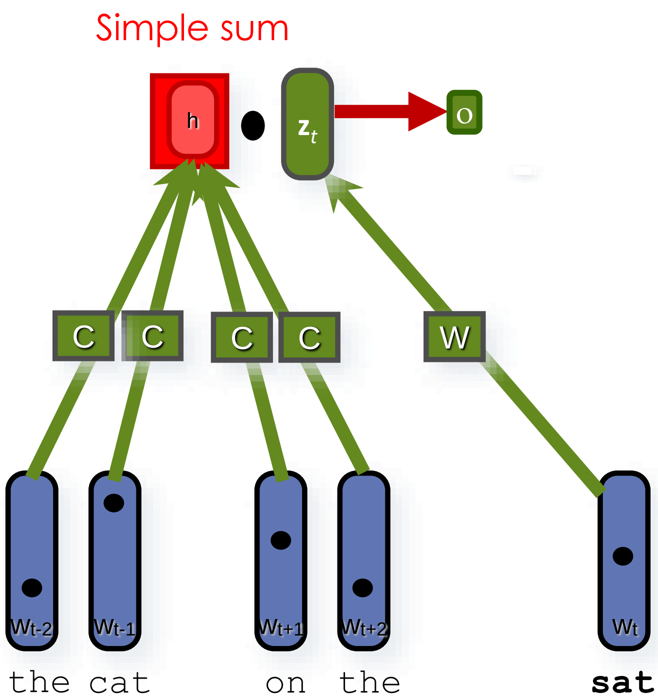
- Sum of all word vectors
- Average of all word vectors
- (see Deep Sets 2017)

- Each input x is transformed (possibly by several layers) into some representation $\phi(x)$.
- The representations are added up and their output is the processed using the ρ network very much in the same manner as in any deep network (e.g. fully connected layers, nonlinearities, etc.).

Continuous Bag-of-Words

word embedding space $\Re^D$ in dimension D=100 to 300

Word embedding matrices



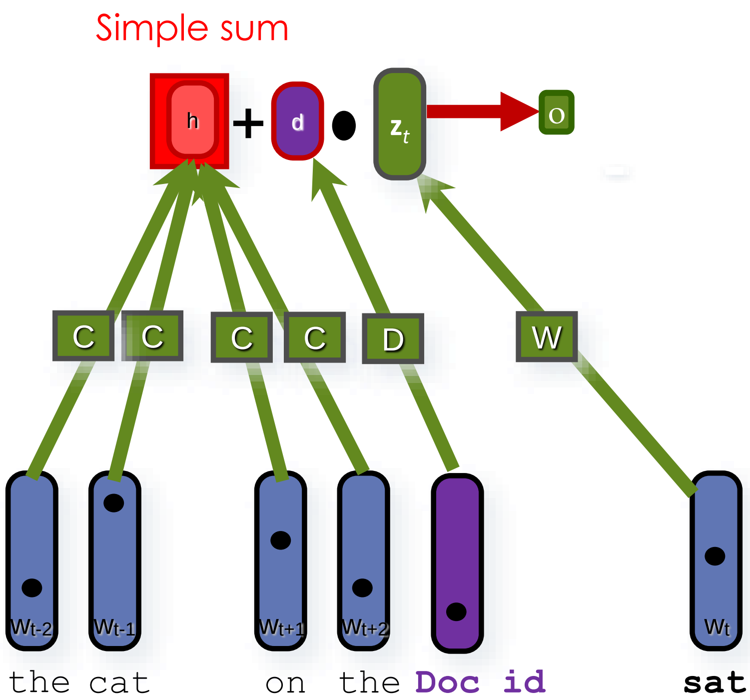
$$\mathbf{h} = \sum_{i=-c}^{c} \mathbf{z}_{t-c}$$

$$\mathbf{o{=}h.z_t}$$

CBOW Paragraph Vector

word embedding space $\Re^D$ in dimension D=100 to 300

Word embedding matrices

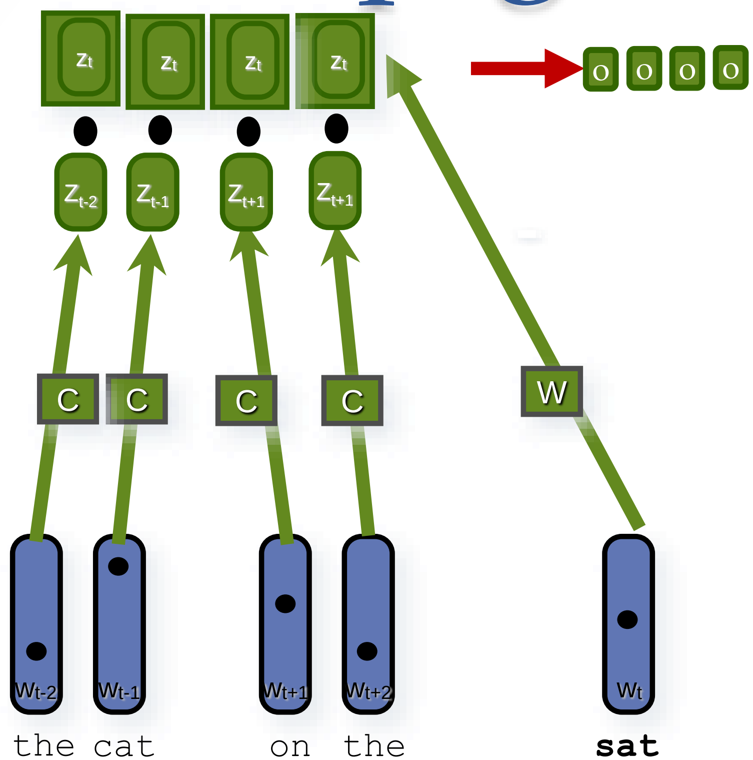


$$\mathbf{h} = \sum_{i=-c}^{c} \mathbf{z}_{t-c}$$

$$\mathbf{o}=(\mathbf{h}+\mathbf{d})\mathbf{z}_t$$

Skip-gram

word embedding space $\Re^D$ in dimension D=100 to 1000

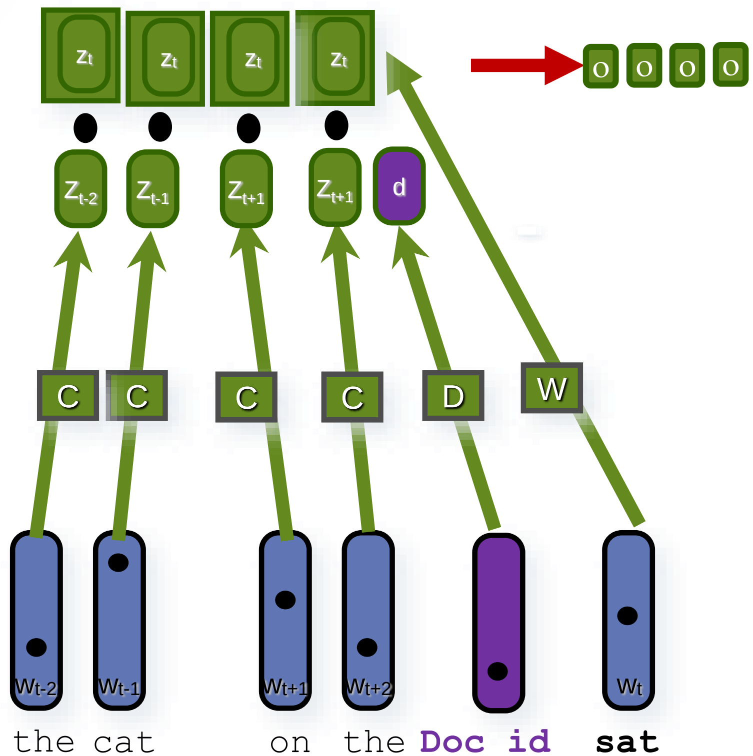


$o = z_{t+i} . z_t \quad i = -2, -1, 1, 2$

[Mikolov et al, 2013a, 2013b; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Skip-gram Paragraph Vector

word embedding space $\Re^D$ in dimension D=100 to 1000



$$o=(z_{t+i}+d)z_t \quad i=-2,-1,1,2$$

[Mikolov et al, 2013a, 2013b; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

New Document

- Keep U, w, etc fixed.
- Just relearn d parameters via backprop

Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

More Reading resources

- https://web.stanford.edu/~jurafsky/li15/lec3.vector.pdf
- https://ruder.io/word-embeddings-1/
- https://ruder.io/word-embeddings-softmax/index.html
- https://ruder.io/secret-word2vec/index.html

Finally, for the brave-hearted…

- Word2Vec - highly optimized C code:
- https://github.com/tmikolov/word2vec
- Note of Caution: Lots of malloc, calloc

- Readable version of the code:
- https://github.com/chrisjmccormick/word2vec_commented

- Python implementation:
- https://github.com/RaRe-Technologies/gensim

Pytorch Worksheet

- Link: https://colab.research.google.com/drive/1_2Ge4OLWj6I8O9Odp-OGYzHmr04tKC96?usp=sharing
- Contains 7 problems with varying levels of difficulty
- Will help improve your understanding of Pytorch