Neural Language Models and Representation Discovery

(Slides by Piotr Mirowski, Hugo Larochelle, Omer Levy and Tomas Mikolov)

Limitations of n-grams

- Conditional likelihood of seeing a sub-sequence of length *n* in available training data

`the cat sat on the` **`mat`**	$P(w_t \mid \mathbf{w}_{t-5}^{t-1}) = 0.15$
w_{t-5} w_{t-4} w_{t-3} w_{t-2} w_{t-1} w_t	
`the cat sat on the` **`hat`**	$P(w_t \mid \mathbf{w}_{t-5}^{t-1}) = 0.05$
`the cat sat on the` **`sat`**	$P(w_t \mid \mathbf{w}_{t-5}^{t-1}) = 0$

- Limitation: discrete model (each word is a token)
  - Incomplete coverage of the training dataset
    Vocabulary of size V words: V^n possible n-grams (exponential in *n*)

    `my` `cat sat on the` **`mat`** $P(w_t \mid \mathbf{w}_{t-5}^{t-1}) = ?$

  - Semantic similarity between word tokens is not exploited

    `the cat sat on the` ***`rug`*** $P(w_t \mid \mathbf{w}_{t-5}^{t-1}) = ?$

Outline

- Neural Probabilistic LMs
  - **Vector-space representation** of words
  - **Neural** probabilistic language model
  - Log-Bilinear (LBL) LMs (loss function maximization)
- Long-range dependencies
  - **Recurrent** Neural Networks (RNN)
- Bag-of-word-vector approaches
  - **Continuous bag-of-words** and **skip-gram** models
- Scalability with large vocabularies
  - Tree-structured LMs
  - Noise-contrastive estimation

Outline

- Neural Probabilistic LMs
  - **Vector-space representation** of words
  - **Neural** probabilistic language model
  - Log-Bilinear (LBL) LMs (loss function maximization)

- Long-range dependencies
  - **Recurrent** Neural Networks (RNN)

- Bag-of-word-vector approaches
  - **Continuous bag-of-words** and **skip-gram** models

- Scalability with large vocabularies
  - Tree-structured LMs
  - Noise-contrastive estimation

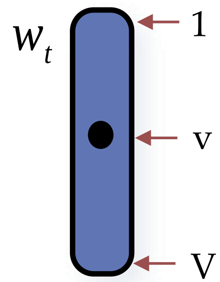
Distributed Representation

- Each word is associated with a continuous valued vector

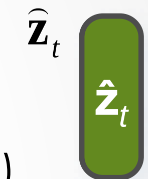
Word	w	$C(w)$
" the "	1	[0.6762, -0.9607, 0.3626, -0.2410, 0.6636]
" a "	2	[0.6859, -0.9266, 0.3777, -0.2140, 0.6711]
" have "	3	[0.1656, -0.1530, 0.0310, -0.3321, -0.1342]
" be "	4	[0.1760, -0.1340, 0.0702, -0.2981, -0.1111]
" cat "	5	[0.5896, 0.9137, 0.0452, 0.7603, -0.6541]
" dog "	6	[0.5965, 0.9143,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

Vector-space representation of words

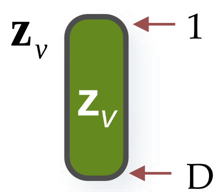
"**One-hot**" of "**one-of-V**" representation of a word token at position *t* in the text corpus, with **vocabulary of size *V***



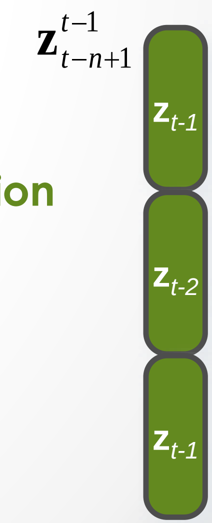
Vector-space representation of the prediction of **target word w_t** (we predict a vector of size *D*)



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



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



Also called
distributed representation

Learning continuous space language models

- Input:
  - word history (**one-hot** or **distributed representation**)
- Output:
  - target word (**one-hot** or **distributed representation**)
- **Function** that **approximates word likelihood**:
  - **Linear transform**
  - **Feed-forward neural network**
  - **Recurrent neural network**
  - **Continuous bag-of-words**
  - **Skip-gram**
  - …

Learning continuous space language models

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

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

Vector-space representation of words

- Compare two words using vector representations:
  - Dot product
  - Cosine similarity
  - Euclidean distance

- **Bi-Linear scoring function** at position *t*:

$$s\left(\mathbf{w}_1^{t-1}, v; \boldsymbol{\theta}\right) = s\left(\hat{\mathbf{z}}_t, v\right) = s_{\boldsymbol{\theta}}\left(v\right) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

  - **Parametric model θ** predicts next word
  - Bias b_v for word *v* related to unigram probabilities of word *v*
  - Given a **predicted vector $\hat{z}_t$**,
    the actual predicted word is the **1-nearest neighbour of $\hat{z}_t$**
  - Exhaustive search in large vocabularies (V in millions)
    can be computationally expensive…

[Mnih & Hinton, 2007]

Word probabilities from vector-space representation

- Normalized probability:
  - Using **softmax** function

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = \frac{e^{s(\hat{\mathbf{z}}_t, v)}}{\sum_{v'=1}^{V} e^{s(\hat{\mathbf{z}}_t, v')}}$$

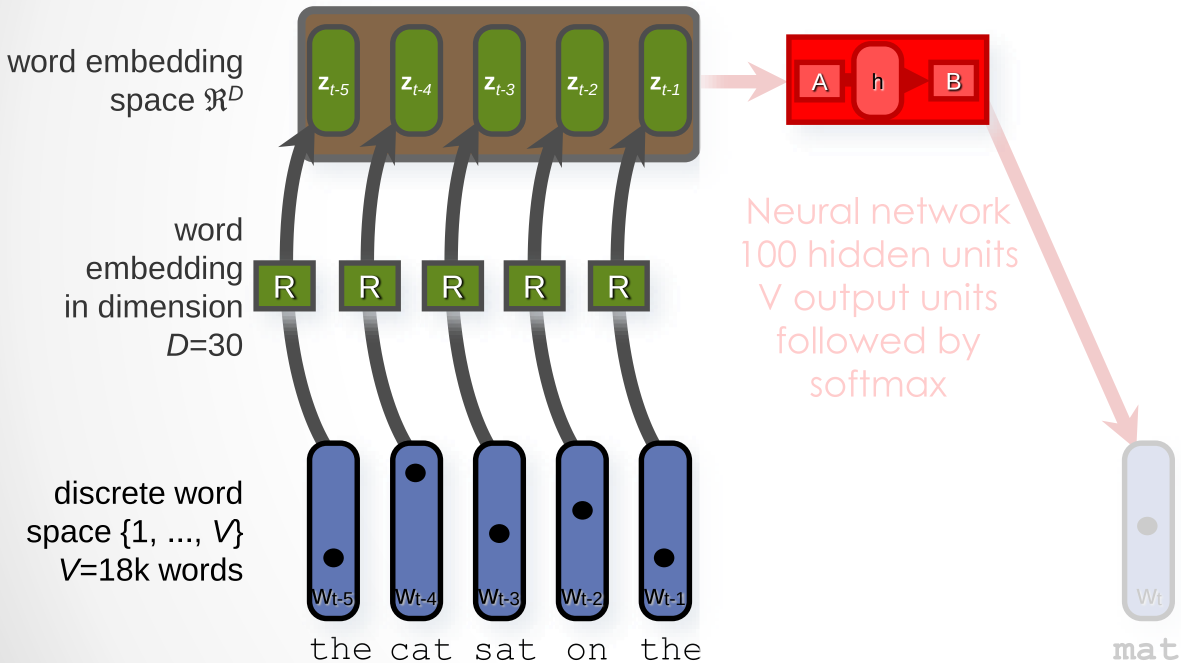
- **Bi-Linear scoring function** at position *t*:

$$s\left(\mathbf{w}_1^{t-1}, v; \boldsymbol{\theta}\right) = s\left(\hat{\mathbf{z}}_t, v\right) = s_{\boldsymbol{\theta}}\left(v\right) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

  - **Parametric model θ** predicts next word
  - Bias b_v for word *v* related to unigram probabilities of word *v*
  - Given a **predicted vector $\hat{z}_t$**,
    the actual predicted word is the **1-nearest neighbour of $\hat{z}_t$**
  - Exhaustive search in large vocabularies (V in millions)
    can be computationally expensive…

[Mnih & Hinton, 2007]

Neural Probabilistic Language Model




```
function z_hist = Embedding_FProp(model, w)
% Get the embeddings for all words in w
z_hist = model.R(:, w);
z_hist = reshape(z_hist, length(w)*model.dim_z, 1);
```

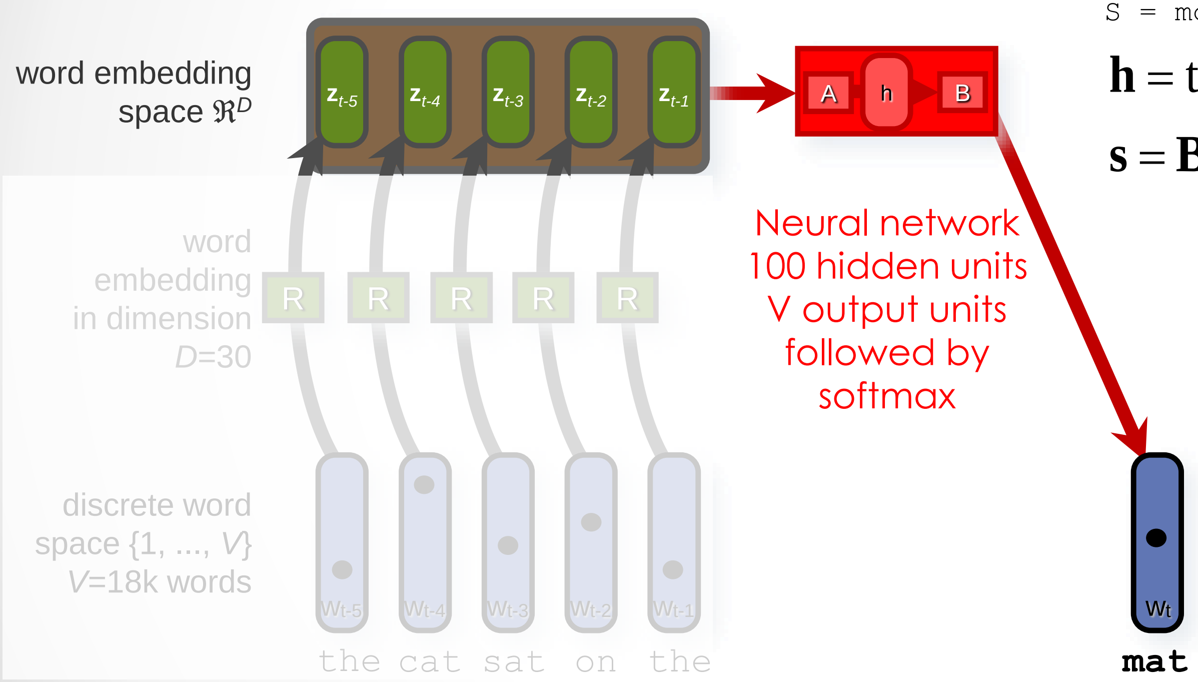
[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", *ICASSP* 2002]

Neural Probabilistic Language Model

```
function s = NeuralNet_FProp(model, z_hist)
% One hidden layer neural network
o = model.A * z_hist + model.bias_a;
h = tanh(o);
S = model.B * h + model.bias_b;
```

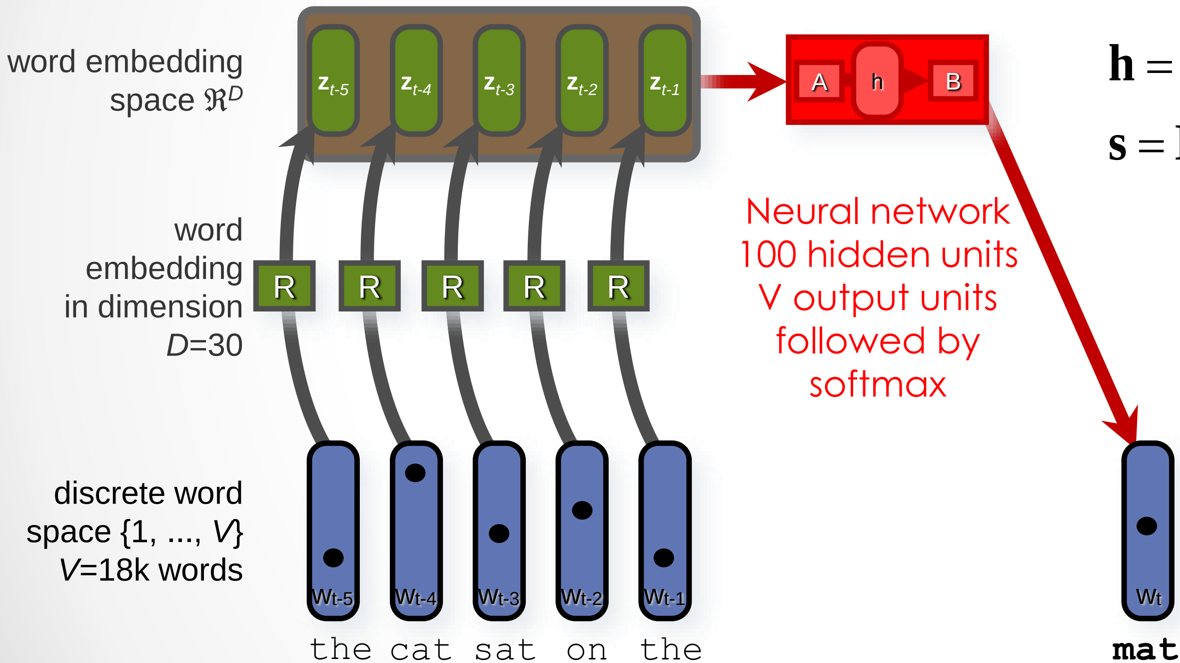
$$\mathbf{h} = \tanh\left(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a\right)$$

$$\mathbf{s} = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$



[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", *ICASSP* 2002]

Neural Probabilistic Language Model



$$\mathbf{h} = \tanh\left(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a\right)$$

$$\mathbf{s} = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

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

Outperforms best n-grams (Class-based Kneyser-Ney back-off 5-grams) by 7%

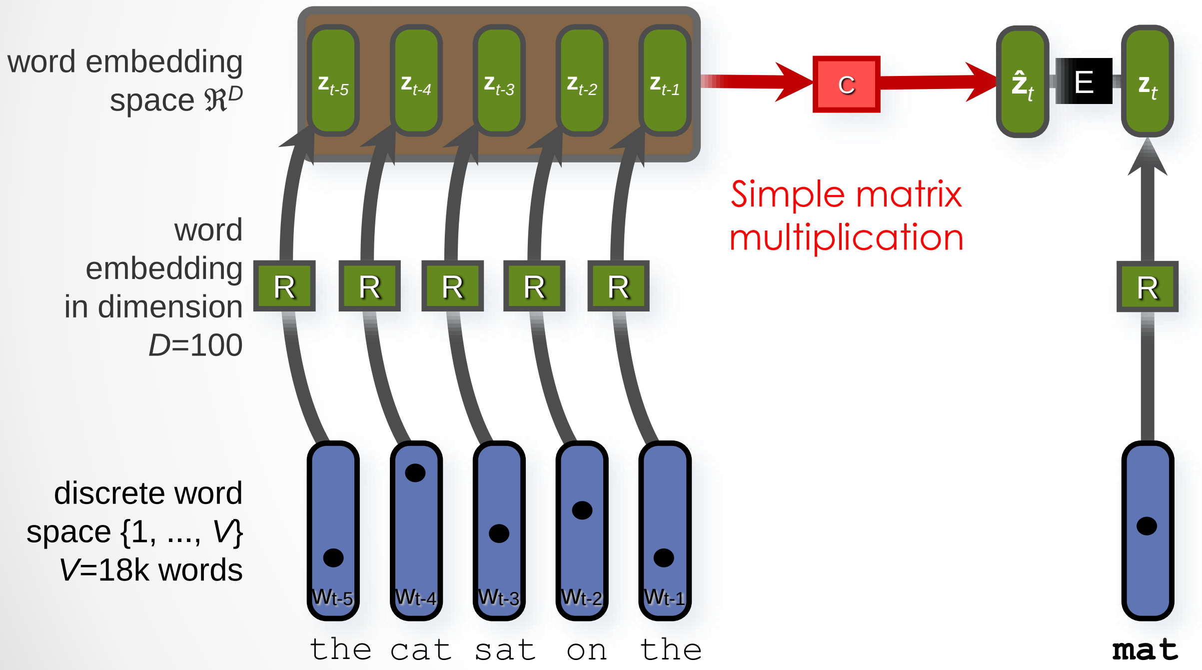
Took months to train (in 2001-2002) on AP News corpus (14M words)

Complexity: (n-1)×D + (n-1)×D×H + H×V

[Bengio et al, 2001, 2003; Schwenk et al, "Connectionist language modelling for large vocabulary continuous speech recognition", *ICASSP* 2002]

Log-Bilinear Language Model

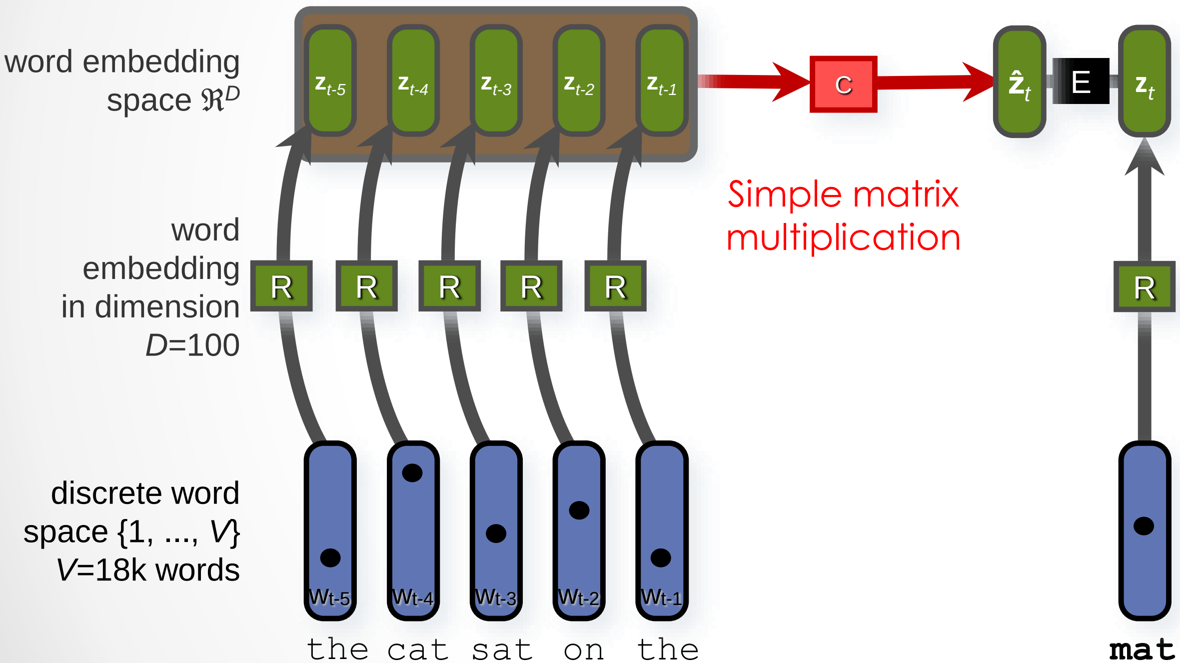
```
function z_hat = LBL_FProp(model, z_hist)
% Simple linear transform
Z_hat = model.C * z_hist + model.bias_c;
```



$$\hat{\mathbf{z}}_t = \mathbf{C}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_c$$

[Mnih & Hinton, 2007]

Log-Bilinear Language Model



$$\hat{\mathbf{z}}_t = \mathbf{C}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_c$$

$$s_{\boldsymbol{\theta}}(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

$$P\left(w_t \mid \mathbf{w}_{t-n+1}^{t-1}\right) = \frac{e^{s_{\boldsymbol{\theta}}(w_t)}}{\sum_v e^{s_{\boldsymbol{\theta}}(v)}}$$

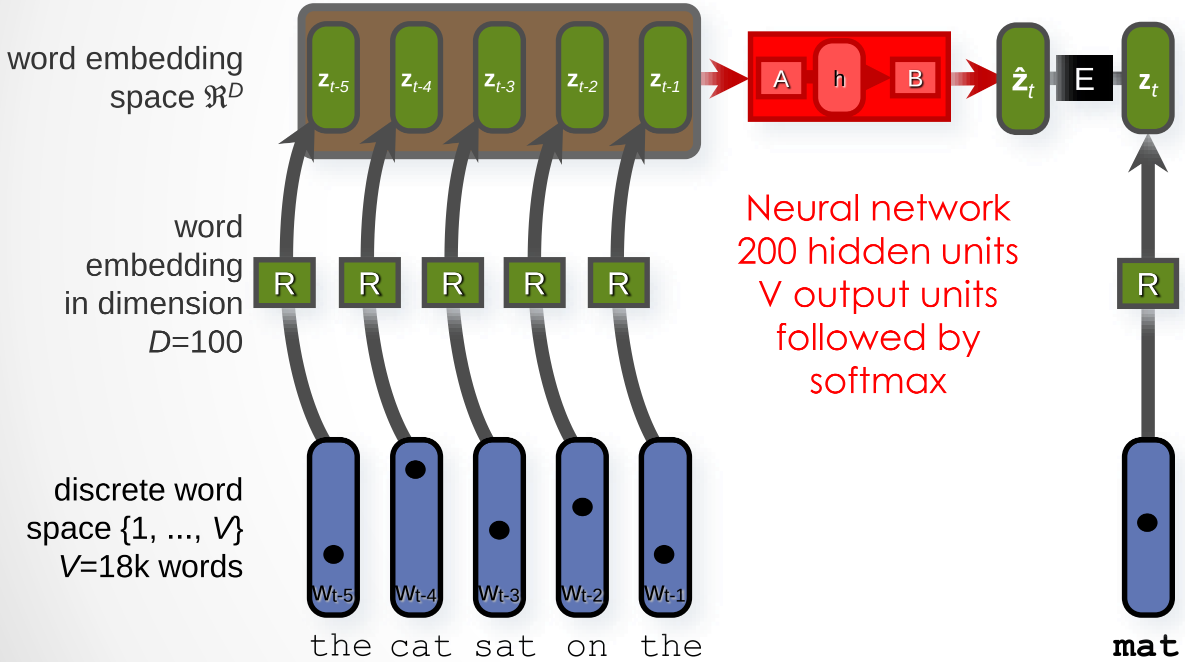
Slightly better than best n-grams (Class-based Kneyser-Ney back-off 5-grams)

Takes days to train (in 2007) on AP News corpus (14 million words)

Complexity: $(n\text{-}1)\times D + (n\text{-}1)\times D\times D + D\times V$

[Mnih & Hinton, 2007]

Nonlinear Log-Bilinear Language Model



$$\mathbf{h} = \tanh\left(\mathbf{A}\mathbf{z}_{t-n+1}^{t-1} + \mathbf{b}_a\right)$$

$$\hat{\mathbf{z}}_t = \mathbf{B}\mathbf{h} + \mathbf{b}_b$$

$$s_\theta(v) = \hat{\mathbf{z}}_t^T \mathbf{z}_v + b_v$$

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

Outperforms best n-grams (Class-based Kneyser-Ney back-off 5-grams) by 24%

Took weeks to train (in 2009-2010) on AP News corpus (14M words)

Complexity: $(n\text{-}1)\times D + (n\text{-}1)\times D\times H + H\times D + D\times V$

[Mnih & Hinton, *Neural Computation*, 2009]

Limitations of these neural language models

- **Computationally expensive** to train
  - Bottleneck: need to **evaluate probability** of each word over the **entire vocabulary**
  - Very slow training time (days, weeks)

- **Ignores long-range dependencies**
  - Fixed time windows
  - **Continuous version of n-grams**

More Observations

- There is no knowledge built in that most recent context word should generally be more informative than earlier ones – this has to be learned

- Parameters of the model are hard to interpret
  - L2 norm of A_j for different context words j correspond to importance of history position j
  - Individual word embeddings can be clustered and dimensions can be analyzed (Tsvetkov et al 2015)

- Architectures are non-intuitive

- Still, perplexity gains substantial…

Outline

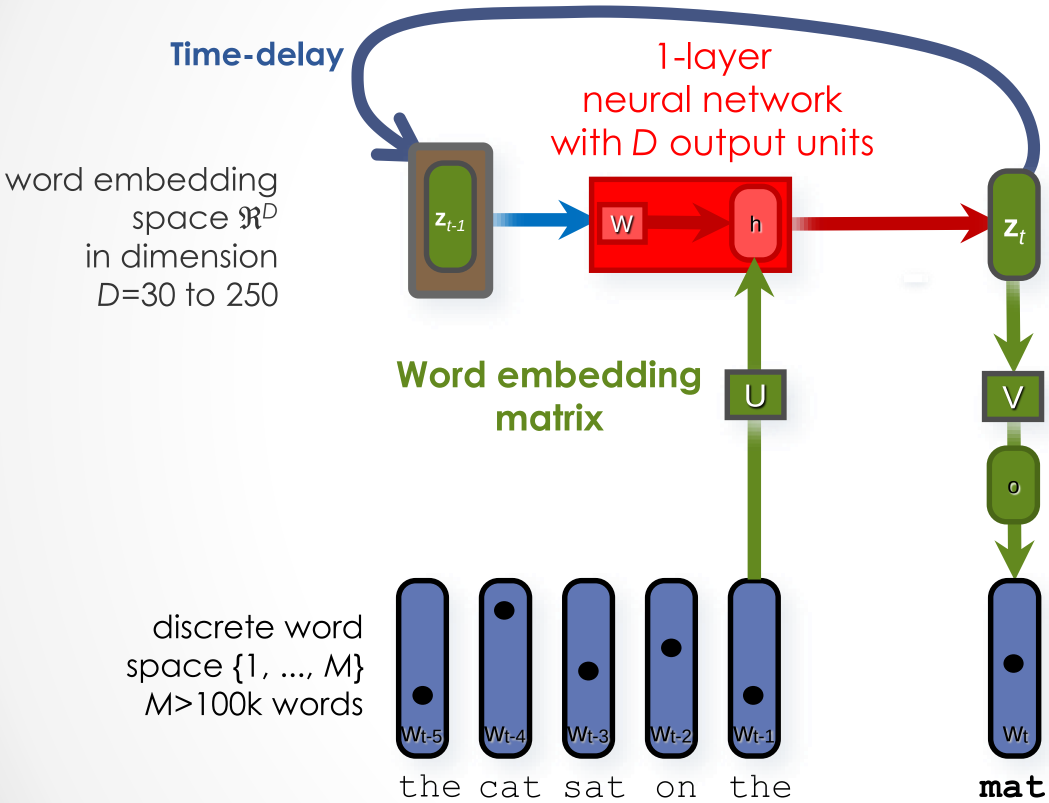
- Neural Probabilistic LMs
  - **Vector-space representation** of words
  - **Neural** probabilistic language model
  - Log-Bilinear (LBL) LMs (loss function maximization)

- Long-range dependencies
  - **Recurrent** Neural Networks (RNN)

- Bag-of-word-vector approaches
  - **Continuous bag-of-words** and **skip-gram** models

- Scalability with large vocabularies
  - Tree-structured LMs
  - Noise-contrastive estimation

Recurrent Neural Net (RNN) language model



$$\mathbf{z}_t = \sigma\left(\mathbf{W}\mathbf{z}_{t-1} + \mathbf{U}\mathbf{w}_t\right)$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\mathbf{o} = \mathbf{V}\mathbf{z}_t$$

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

Handles **longer word history** (~10 words) as well as 10-gram feed-forward NNLM

Training algorithm: BPTT
Back-Propagation Through Time

Complexity: $D \times D + D \times D + D \times V$

[Mikolov et al, 2010, 2011]

Word embeddings obtained on Reuters

- Example of word embeddings obtained using our language model on the Reuters corpus (not RNN) (1.5 million words, vocabulary V=12k words), vector space of dimension D=100
- For each word, the 10 nearest neighbours in the vector space retrieved using cosine similarity:

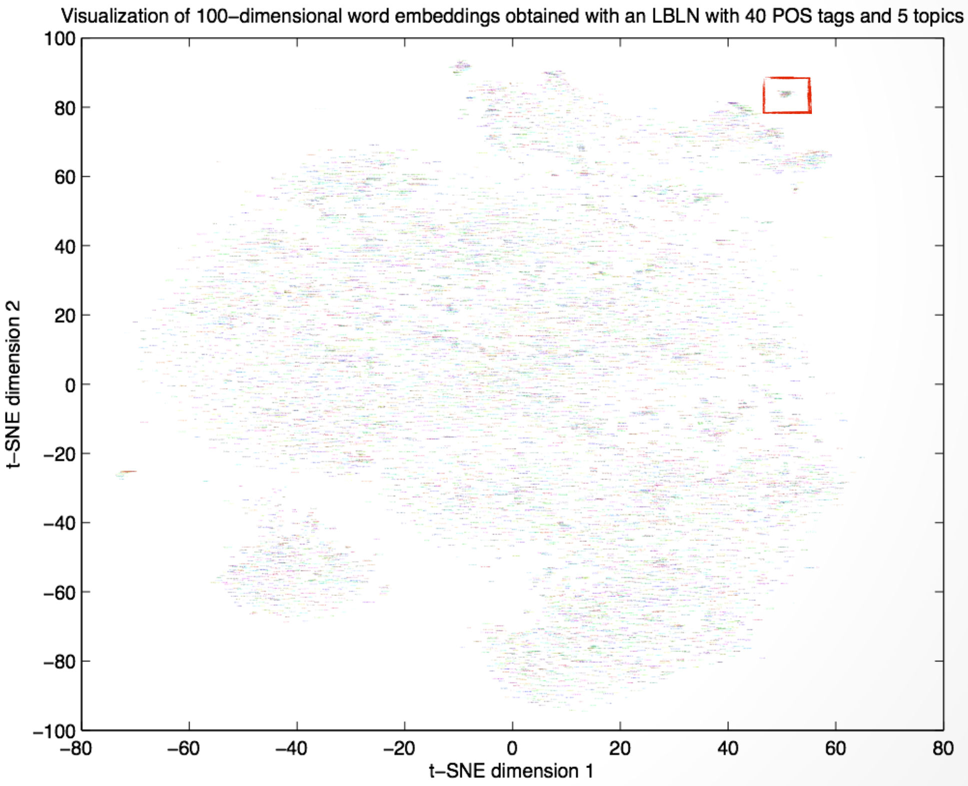
debt	aa	decrease	met	slow
financing	aaa	drop	introduced	moderate
funding	bbb	decline	rejected	lower
debts	aa-minus	rise	sought	steady
loans	b-minus	increase	supported	slowing
borrowing	a-1	fall	called	double
short-term	bb-minus	jump	charged	higher
indebtedness	a-3	surge	joined	break
long-term	bbb-minus	reduction	adopted	weaker
principal	a-plus	limit	made	stable
capital	a-minus	slump	sent	narrow

[Mirowski, Chopra, Balakrishnan and Bangalore (2010) "Feature-rich continuous language models for speech recognition", *SLT*]

Word embeddings obtained on AP News

Example of word embeddings obtained using our LM on AP News (14M words, *V*=17k), *D*=100

The word embedding matrix R was projected in 2D by Stochastic t-SNE [Van der Maaten, *JMLR* 2008]

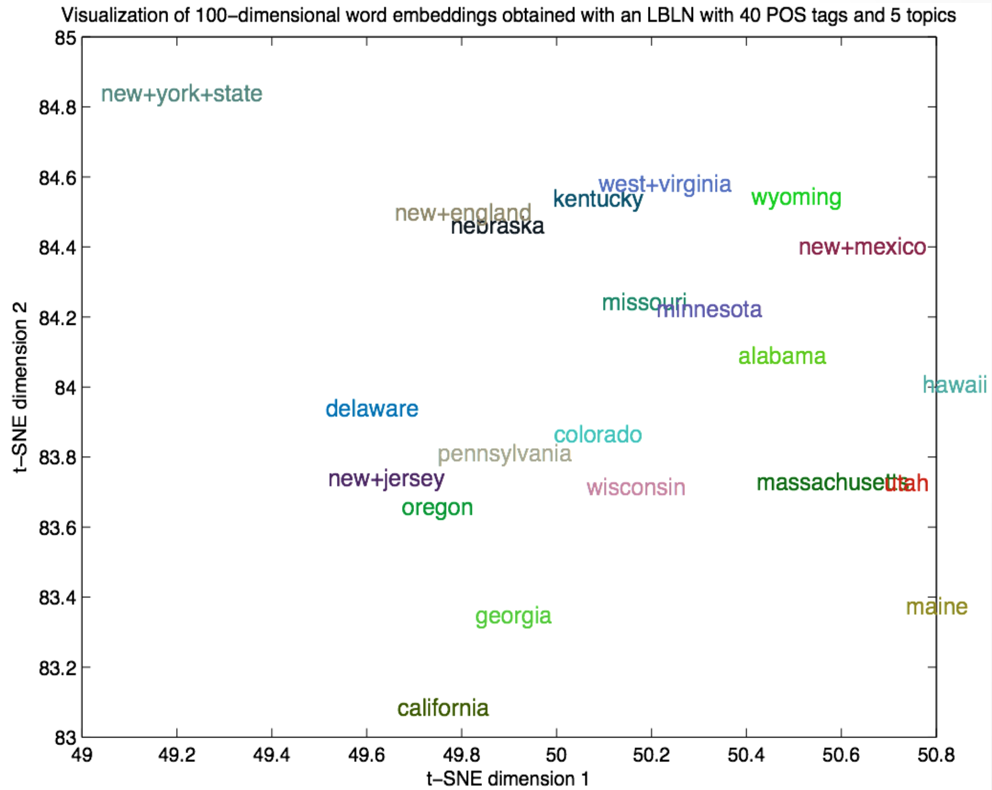


[Mirowski (2010)
"Time series modelling with hidden variables and gradient-based algorithms", *NYU PhD thesis*]

Word embeddings obtained on AP News

Example of word embeddings obtained using our LM on AP News (14M words, V=17k), D=100

The word embedding matrix R was projected in 2D by Stochastic t-SNE [Van der Maaten, *JMLR* 2008]

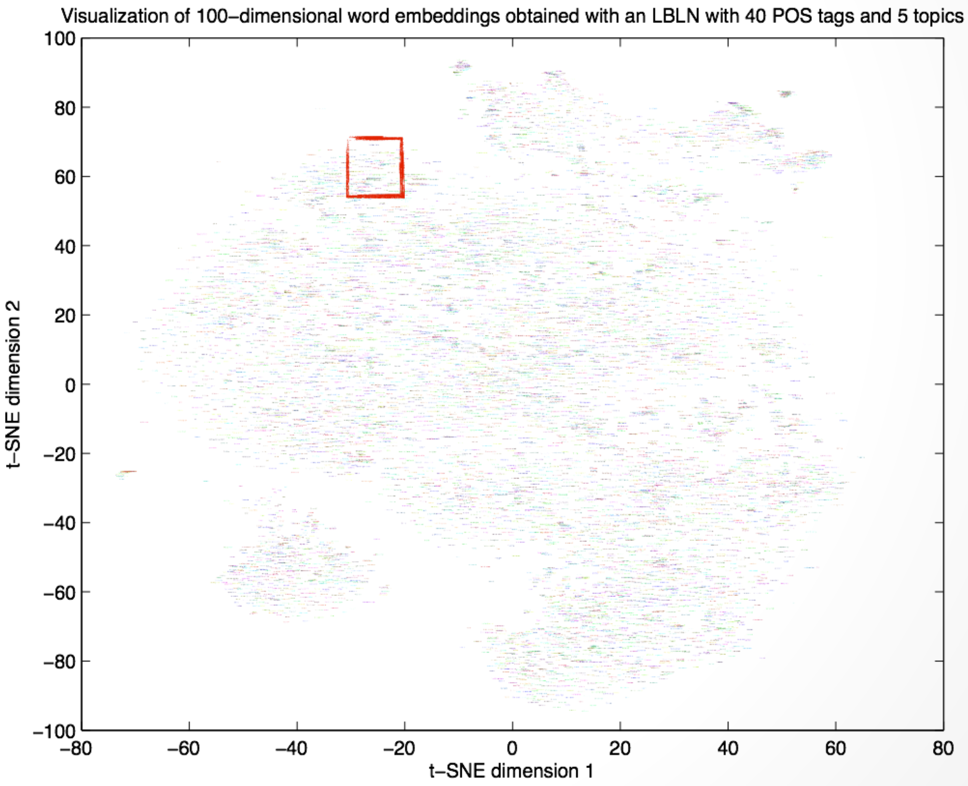


[Mirowski (2010) "Time series modelling with hidden variables and gradient-based algorithms", *NYU PhD thesis*]

Word embeddings obtained on AP News

Example of word embeddings obtained using our LM on AP News (14M words, V=17k), D=100

The word embedding matrix R was projected in 2D by Stochastic t-SNE [Van der Maaten, *JMLR* 2008]

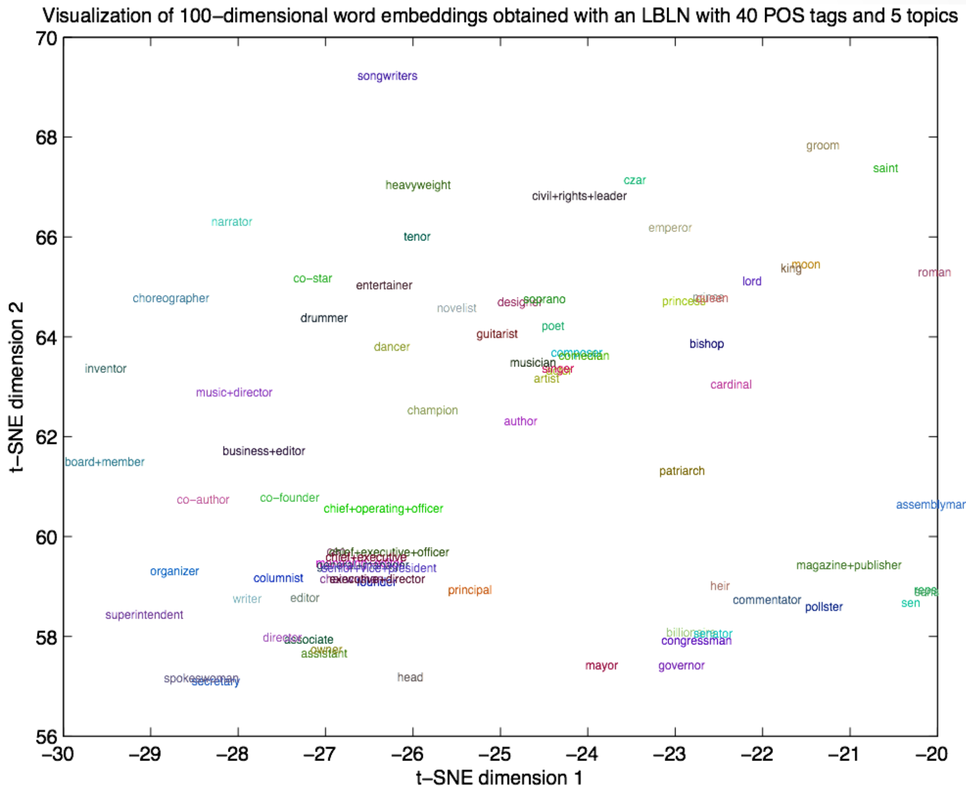


[Mirowski (2010)
"Time series modelling with hidden variables and gradient-based algorithms", *NYU PhD thesis*]

Word embeddings obtained on AP News

Example of word embeddings obtained using our LM on AP News (14M words, *V*=17k), *D*=100

The word embedding matrix R was projected in 2D by Stochastic t-SNE [Van der Maaten, *JMLR* 2008]



[Mirowski (2010)
"Time series modelling with hidden variables and gradient-based algorithms", *NYU PhD thesis*]

Outline

- Neural Probabilistic LMs
  - **Vector-space representation** of words
  - **Neural** probabilistic language model
  - Log-Bilinear (LBL) LMs (loss function maximization)

- Long-range dependencies
  - **Recurrent** Neural Networks (RNN)

- Bag-of-word-vector approaches
  - **Continuous bag-of-words** and **skip-gram** models

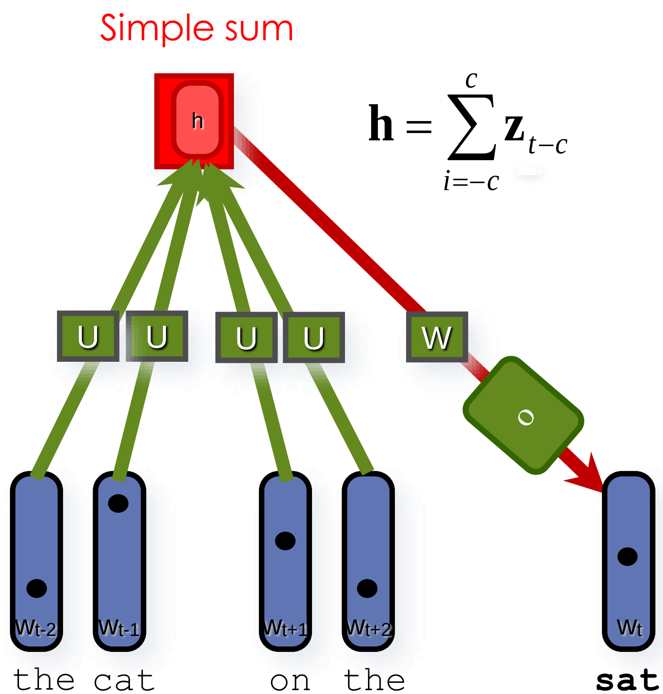
- Scalability with large vocabularies
  - Tree-structured LMs
  - Noise-contrastive estimation

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



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

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

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

Extremely efficient estimation of **word embeddings** in matrix **U** **without a Language Model**.
Can be used as input to neural LM.
Enables much larger datasets, e.g., Google News (6B words, V=1M)

Complexity: 2C×D + D×V

Complexity: 2C×D + D×log(V) (**hierarchical softmax using tree factorization**)

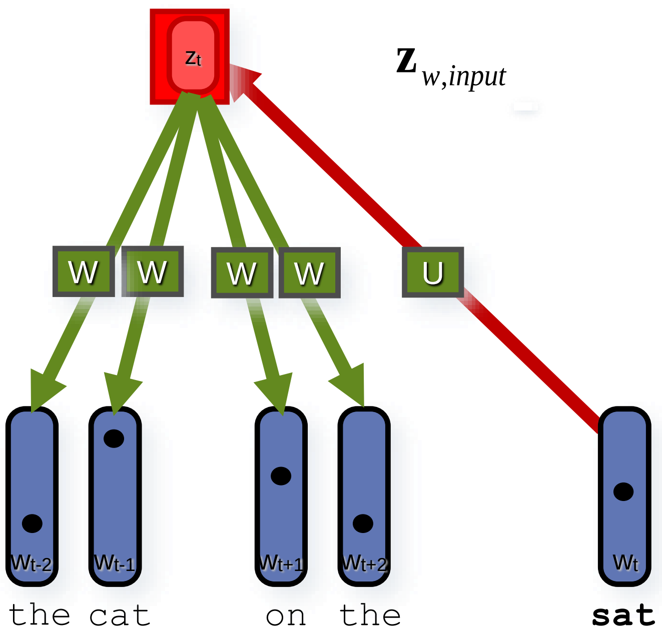
[Mikolov et al, 2013a; Mnih & Kavukcuoglu, 2013; http://code.google.com/p/word2vec]

Skip-gram

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

Word embedding matrices

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



$$s_\theta(w,c) = \mathbf{z}_{c,output}\mathbf{z}_{w,input}$$

$$P(w_{t+c} \mid w_t) = \frac{e^{s_\theta(w,c)}}{\sum_v e^{s_\theta(v,c)}}$$

Extremely efficient estimation of **word embeddings** in matrix **U** **without a Language Model**.
Can be used as input to neural LM.
Enables much larger datasets, e.g., Google News (33B words, V=1M)

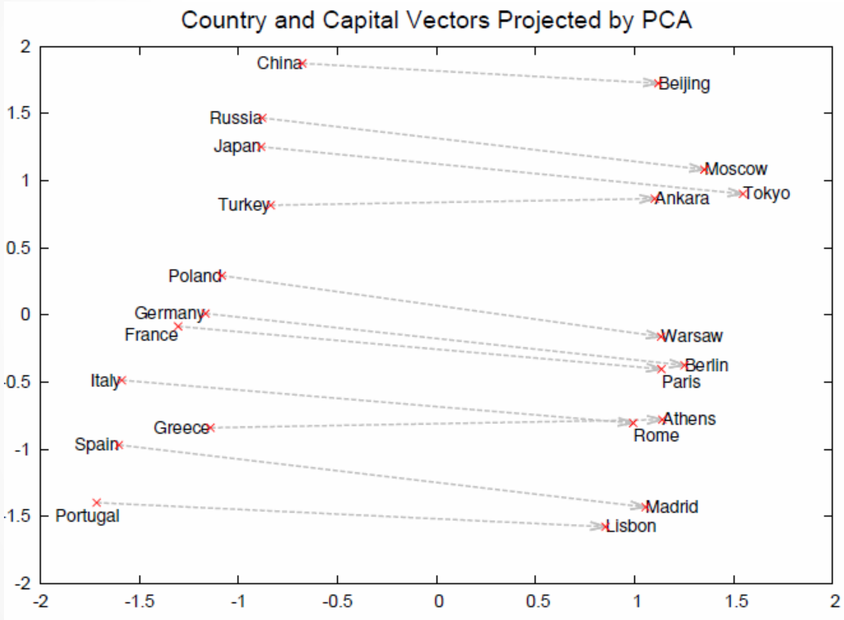
Complexity: 2C×D + 2C×D×V

Complexity: 2C×D + 2C×D×log(V) (**hierarchical softmax using tree factorization**)

Complexity: 2C×D + 2C×D×(k+1) (**negative sampling with k negative examples**)

[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]

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*]

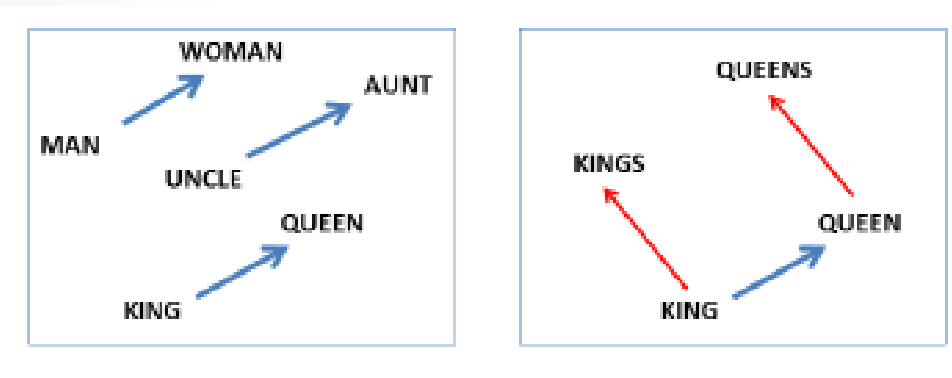
[Mikolov et al, 2013a, 2013b; http://code.google.com/p/word2vec]

Syntactic and Semantic tests with RNN

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

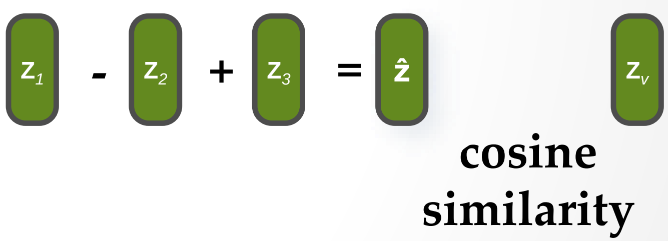
Syntactic: king is to kings as queen is to **queens**

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



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

Vector offset method



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

Semantic-syntactic word evaluation task

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

[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]

Outline

- Neural Probabilistic LMs
  - **Vector-space representation** of words
  - **Neural** probabilistic language model
  - Log-Bilinear (LBL) LMs (loss function maximization)

- Long-range dependencies
  - **Recurrent** Neural Networks (RNN)

- Bag-of-word-vector approaches
  - **Continuous bag-of-words** and **skip-gram** models

- Scalability with large vocabularies
  - Tree-structured LMs
  - Noise-contrastive estimation

Computational bottleneck of large vocabularies

target word	$w(t)$
word history	$\mathbf{w}_1^{t-1}$
scoring function	$s_v(t) = s(\mathbf{w}_1^{t-1}, v)$
softmax	$g(s_v) = \frac{e^{s_v}}{\sum_{v'=1}^{V} e^{s_{v'}}}$

$$P(w_t = v \mid \mathbf{w}_1^{t-1}) = g(s_v(t))$$

- Bulk of computation at **word prediction** and at **input word embedding** layers
- Large vocabularies:
  - AP News (14M words; V=17k)
  - HUB-4 (1M words; V=25k)
  - Google News (6B words, V=1M)
  - Wikipedia (3.2B, V=2M)
- Strategies to compress output softmax

Reducing the bottleneck of large vocabularies

- **Replace rare words**, numbers by <unk> token
- **Subsample frequent words** during training
  - Speed-up 2x to 10x
  - Better accuracy for rare words

- **Hierarchical Softmax** (HS)
- **Noise-Contrastive Estimation** (NCE) and **Negative Sampling** (NS)

[Morin & Bengio, 2005, Mikolov et al, 2011, 2013b; Mnih & Teh 2012, Mnih & Kavukcuoglu, 2013]

Hierarchical softmax by grouping words

target word $w(t)$

word history $\mathbf{w}_1^{t-1}$

scoring function $s_{\boldsymbol{\theta}}(v) = s\left(\mathbf{w}_1^{t-1}, v; \boldsymbol{\theta}\right)$

softmax $g\left(s_{\boldsymbol{\theta}}(v)\right) = \dfrac{e^{s_{\boldsymbol{\theta}}(v)}}{\sum_{v'=1}^{V} e^{s_{\boldsymbol{\theta}}(v')}}$

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = g\left(s_{\boldsymbol{\theta}}(v)\right)$$

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = \boxed{P\left(c \mid \mathbf{w}_1^{t-1}\right)} \times \boxed{P\left(v \mid \mathbf{w}_1^{t-1}, c\right)}$$

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = \boxed{g\left(s_{\boldsymbol{\theta}}(c)\right)} \times \boxed{g\left(s_{\boldsymbol{\theta}}(c, v)\right)}$$

- Group words into **disjoint classes**:
  - E.g., 20 classes with frequency binning
  - Use unigram frequency
  - Top 5% words ("the") go to class 1
  - Following 5% words go to class 2
- Factorize word probability into:
  - **Class probability**
  - **Class-conditional word probability**
- Speed-up factor:
  - O(|V|) to O(|C|+max|VC|)

[Mikolov et al, 2011, Auli et al, 2013]

Hierarchical softmax by grouping words

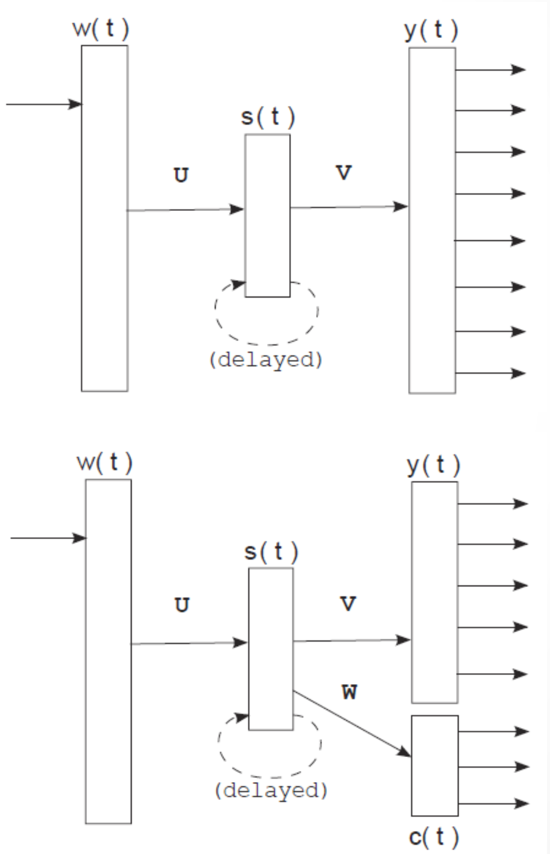
target word $w(t)$

word history $\mathbf{w}_1^{t-1}$

scoring function $s_\theta(v) = s\left(\mathbf{w}_1^{t-1}, v; \theta\right)$

softmax $g(s_\theta(v)) = \frac{e^{s_\theta(v)}}{\sum_{v'=1}^{V} e^{s_\theta(v')}}$

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = g(s_\theta(v))$$



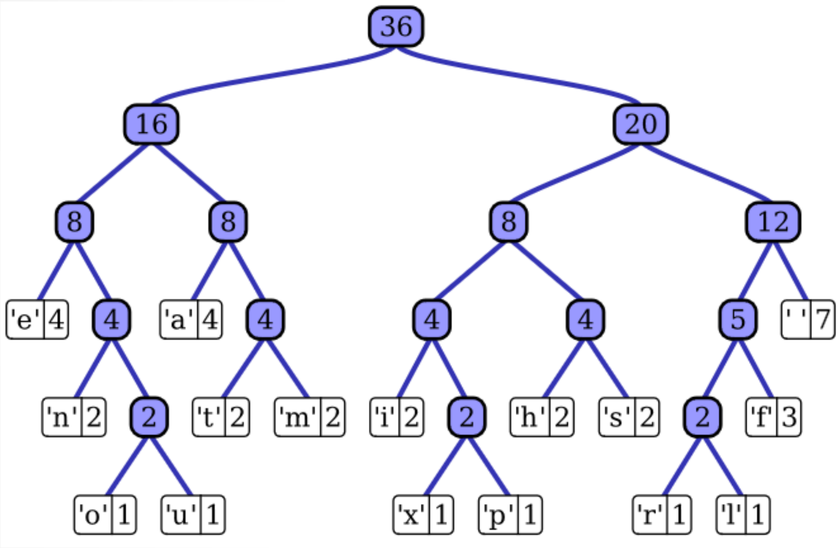
[Image credits: Mikolov et al (2011) "Extensions of Recurrent Neural Network Language Model", *ICASSP*]

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = \boxed{P\left(c \mid \mathbf{w}_1^{t-1}\right)} \times \boxed{P\left(v \mid \mathbf{w}_1^{t-1}, c\right)}$$

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = \boxed{g(s_\theta(c))} \times \boxed{g(s_\theta(c, v))}$$

[Mikolov et al, 2011, Auli et al, 2013]

Hierarchical softmax using Huffman trees



"this is an example of a huffman tree"

[Image credits: Wikipedia, Wikimedia Commons http://en.wikipedia.org/wiki/File:Huffman_tree_2.svg]

- Frequency-based binning

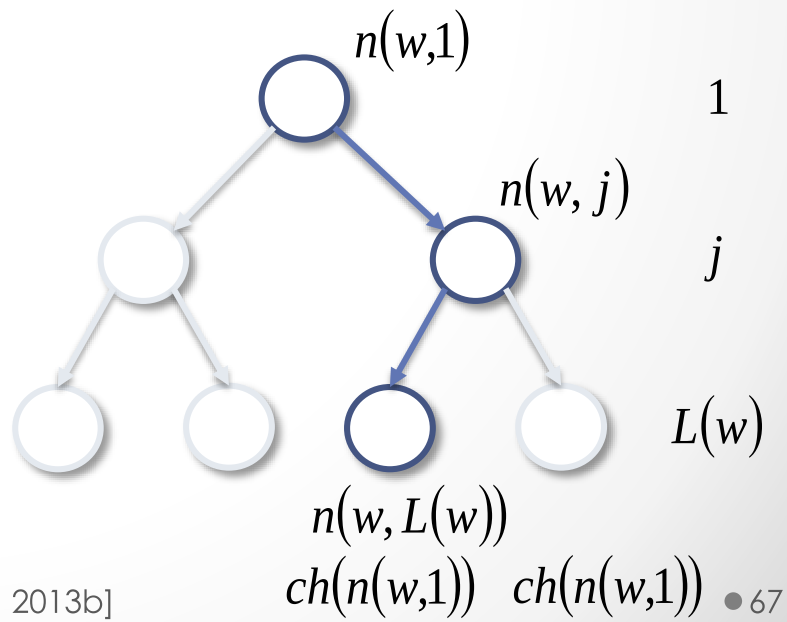
[Mikolov et al, 2013a, 2013b]

Hierarchical softmax using Huffman trees

target word	$w(t)$
word history	$\mathbf{w}_1^{t-1}$
path to target word at node j	$n\left(w(t), j\right)$
predicted word vector	$\hat{\mathbf{z}}(t)$
vector at node j of target word	$\mathbf{z}_{n(w,j)}$
sigmoid	$\sigma(x) = \dfrac{1}{1+e^{-x}}$

$$P\left(w_t = v \mid \mathbf{w}_1^{t-1}\right) = \prod_{j=1}^{L(w)-1} \sigma\left(\pm 1_{n(w,j+1)=ch(n(w,j))} \mathbf{z}_{n(w,j)}^T \hat{\mathbf{z}}\right)$$

- Replace comparison with ***V* vectors** of **target words** by comparison with **log(*V*) vectors**



[Mikolov et al, 2013a, 2013b]

Noise-Contrastive Estimation

- **Conditional probability of word** w in the data:

$$P\left(w_t = w \mid \mathbf{w}_1^{t-1}\right) = \frac{\boxed{e^{s_\theta(w)}}}{\boxed{\sum_{v=1}^{V} e^{s_\theta(v)}}}$$

- **Conditional probability that word** w comes from **data D** and **not from the noise distribution**:

$$P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right) = \frac{\boxed{P_d^{\mathbf{w}_1^{t-1}}(w)}}{\boxed{P_d^{\mathbf{w}_1^{t-1}}(w)} + \boxed{kP_{noise}(w)}} \qquad P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right) = \frac{\boxed{e^{s_\theta(w)}}}{\boxed{e^{s_\theta(w)}} + \boxed{kP_{noise}(w)}}$$

  - Auxiliary binary classification problem:
    - **Positive examples (data)** vs. **negative examples (noise)**
  - **Scaling factor** *k*: **noisy samples** k times more likely than data samples
    - **Noise distribution**: based on **unigram word probabilities**
  - Empirically, model can cope with un-normalized probabilities:

$$P_d^{\mathbf{w}_1^{t-1}}(w) \leftarrow P\left(w \mid \mathbf{w}_1^{t-1}, \boldsymbol{\theta}\right) \approx e^{s_\theta(w)}$$

[Mnih & Teh, 2012; Mnih & Kavukcuoglu, 2013; Mikolov et al, 2013a, 2013b]

Noise-Contrastive Estimation

- **Conditional probability that word** w comes from **data D** and **not from the noise distribution**:

$$P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right)=\frac{\boxed{e^{s_\theta(w)}}}{\boxed{e^{s_\theta(w)}}+\boxed{kP_{noise}(w)}}$$

  - Auxiliary binary classification problem:
    - **Positive examples (data)** vs. **negative examples (noise)**
  - **Scaling factor** *k*: **noisy samples** k times more likely than data samples
    - **Noise distribution**: based on **unigram word probabilities**
  - Introduce log of difference between:
    - **score of word** w **under data distribution**
    - and **unigram distribution score of word** w

$$\Delta s_\theta(w)=\boxed{s_\theta(w)}\boxed{-\log kP_{noise}(w)}$$

$$P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right)=\sigma\left(\Delta s_\theta(w)\right) \qquad \sigma(x)=\frac{1}{1+e^{-x}}$$

[Mnih & Teh, 2012; Mnih & Kavukcuoglu, 2013; Mikolov et al, 2013a, 2013b]

Negative sampling

- **Noise contrastive estimation**

$$L_t' = \boxed{E_{P_d^{\mathbf{w}_1^{t-1}}}}\left[\log P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right)\right] + \boxed{kE_{P_{noise}}}\left[\log P\left(D=0 \mid w, \mathbf{w}_1^{t-1}\right)\right]$$

$$P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right) = \frac{\boxed{e^{s_\theta(w)}}}{\boxed{e^{s_\theta(w)}} + \boxed{kP_{noise}(w)}}$$

- **Negative sampling**
  - Remove normalization term in probabilities

$$P\left(D=1 \mid w, \mathbf{w}_1^{t-1}\right) = \sigma\left(s_\theta(w)\right)$$

$$L_t' = \boxed{\log \sigma\left(s_\theta(w)\right)} + \boxed{\sum_{i=1}^{k} E_{P_{noise}}\left[\log \sigma\left(-s_\theta(v_i)\right)\right]}$$

- Compare to **Maximum Likelihood learning**:

$$L_t = \boxed{s_\theta(w)} - \boxed{\log \sum_{v=1}^{V} e^{s_\theta(v)}}$$

[Mnih & Teh, 2012; Mnih & Kavukcuoglu, 2013; Mikolov et al, 2013a, 2013b]

Speed-up over full softmax

LBL with **full softmax**, trained on APNews data, **14M words**, **V=17k** **7days**

Skip-gram (context 5) with phrases, trained using **negative sampling**, on Google data, **33G** words, **V=692k + phrases** **1 day**

Model (training time)	Redmond	Havel	ninjutsu	graffiti	capitulate
Collobert (50d) (2 months)	conyers lubbock keene	plauen dzerzhinsky osterreich	reiki kohona karate	cheesecake gossip dioramas	abdicate accede rearm
Turian (200d) (few weeks)	McCarthy Alston Cousins	Jewell Arzu Ovitz	- - -	gunfire emotion impunity	- - -
Mnih (100d) (7 days)	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	- - -	anaesthetics monkeys Jews	Mavericks planning hesitated
Skip-Phrase (1000d, 1 day)	Redmond Wash. Redmond Washington Microsoft	Vaclav Havel president Vaclav Havel Velvet Revolution	ninja martial arts swordsmanship	spray paint grafitti taggers	capitulation capitulated capitulating

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

LBL (2-gram, 100d) with **full softmax**, **1 day**

LBL (2-gram, 100d) with **noise contrastive estimation** **1.5 hours**

RNN (100d) with **50-class hierarchical softmax** **0.5 hours** (own experience)

Training algorithm	Number of samples	Test PPL	Training time (h)
ML		163.5	21
NCE	1	192.5	1.5
NCE	5	172.6	1.5
NCE	25	163.1	1.5
NCE	100	159.1	1.5
RNN (HS)	50 classes	145.4	0.5

Penn TreeBank data (900k words, V=10k)

[Image credits: Mnih & Teh (2012) "A fast and simple algorithm for training neura probabilistic language models", *ICML*]

[Mnih & Teh, 2012; Mikolov et al, 2010-2012, 2013b]

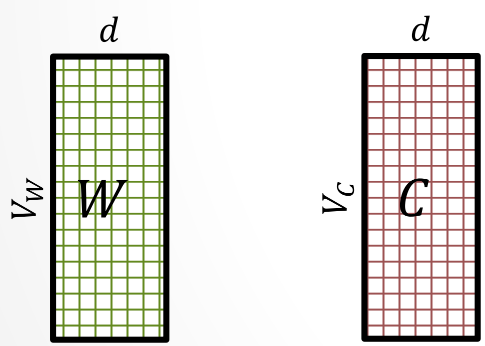
What is `word2vec`?

- `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?

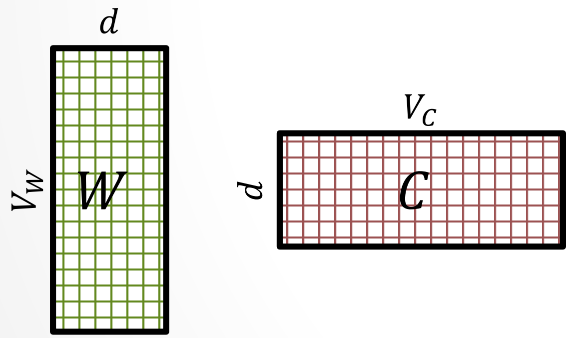
- Take SGNS's embedding matrices (W and C)



"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?

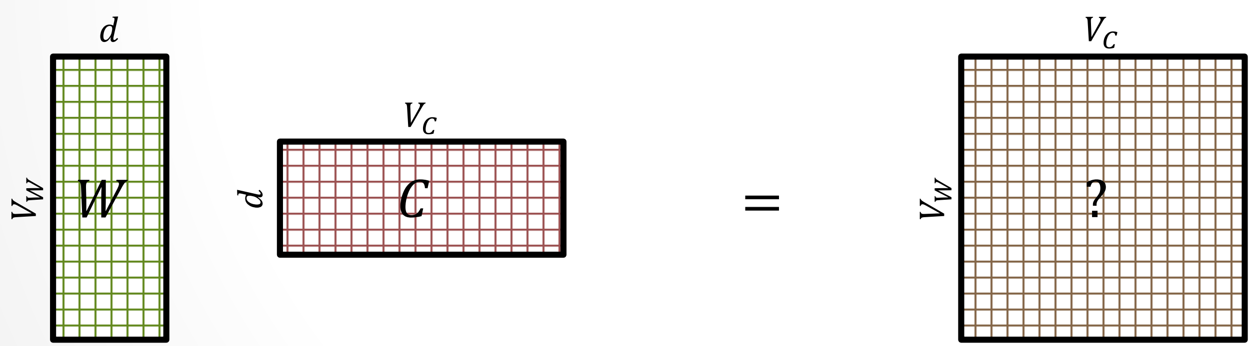


"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

What is SGNS learning?

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

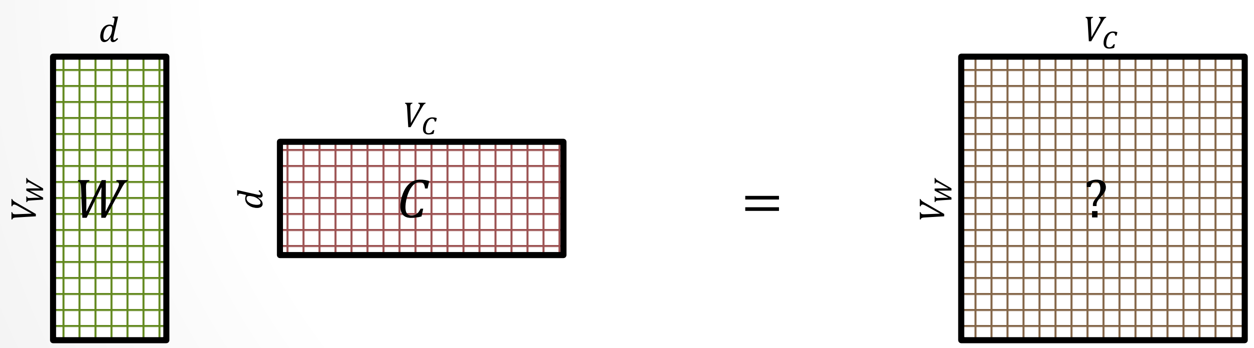
$$\vec{w} \cdot \vec{c} = ?$$



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

What is SGNS learning?

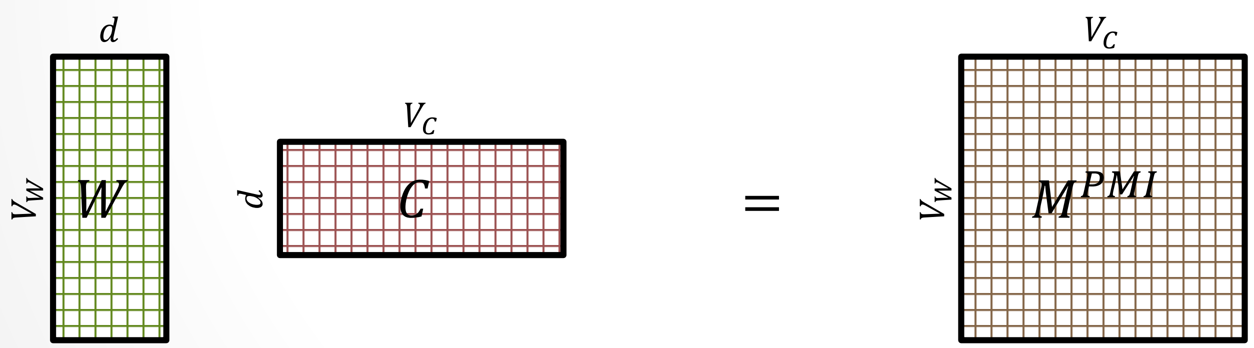
- We **prove** that for large enough d and enough iterations



"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

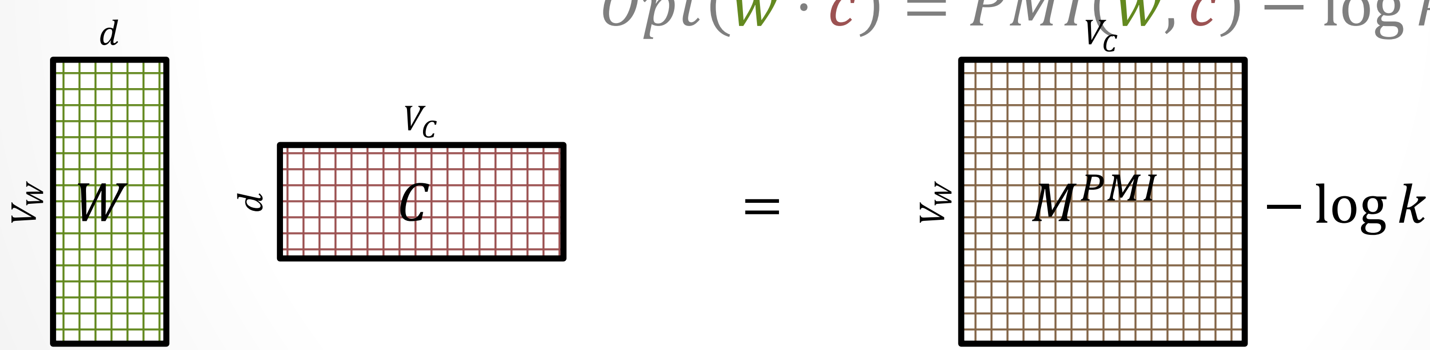


"Neural Word Embeddings as Implicit Matrix Factorization"
Levy & Goldberg, NIPS 2014

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

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



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

GLOVE

- SGNS

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

- GLOVE

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

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

Mar 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

Adapting Hyperparameters across Algorithms

Context Distribution Smoothing

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

- Our analysis assumes P is the unigram distribution

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

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)

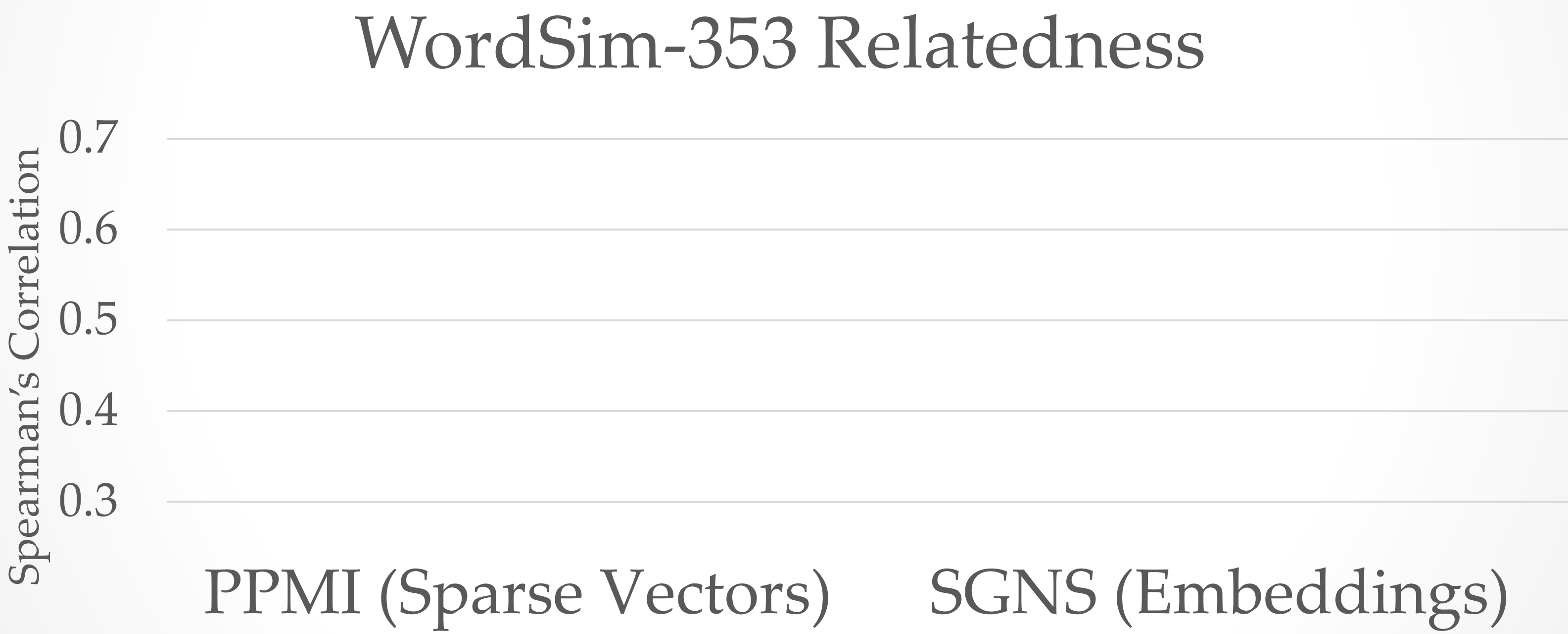
- Preprocessing
  - <None>
- Postprocessing
  - <None>
- Association Metric
  - Vanilla PMI/PPMI

Recommended word2vec Setting

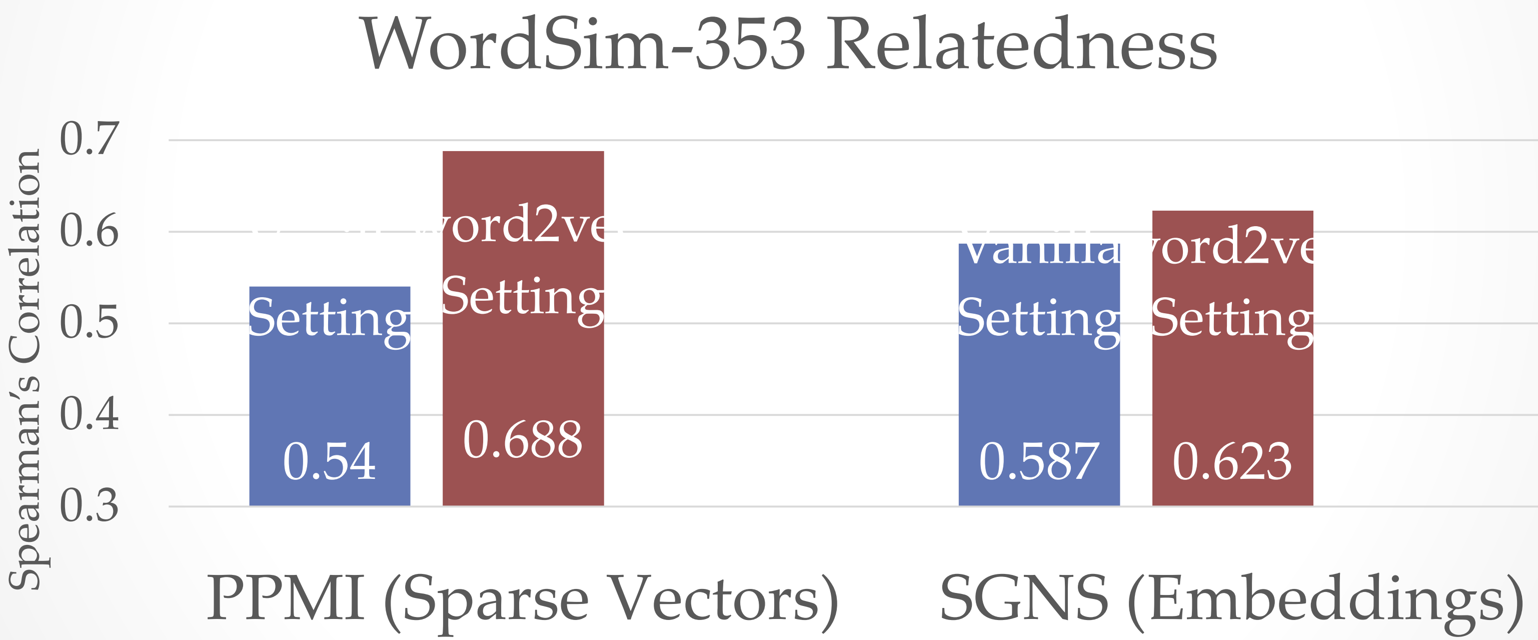
(tuned for SGNS)

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

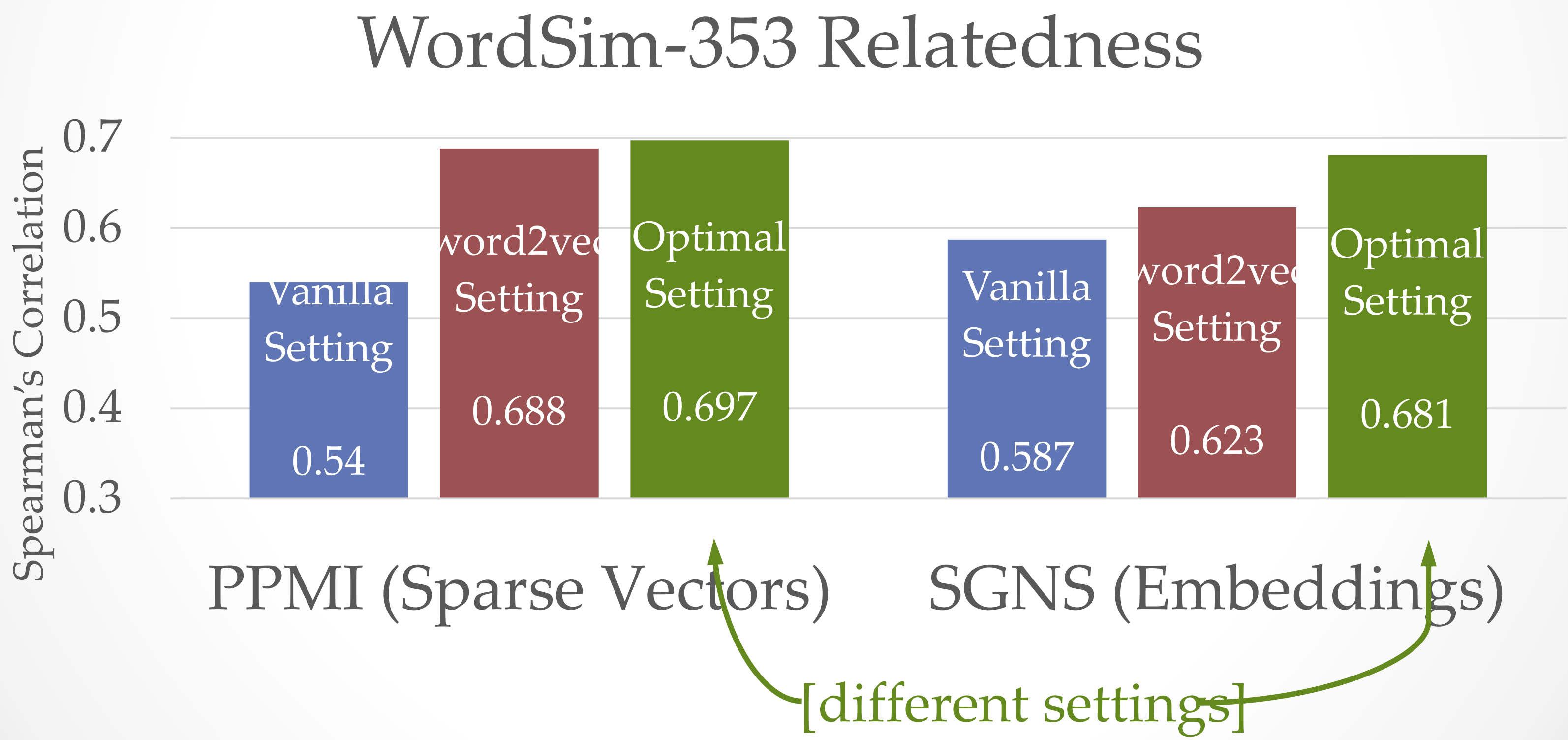
Experiments



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
- **Not exactly true for NEURAL LANGUAGE MODELS**