Word embeddings
-----
an introduction and applications

P. Meladianos, A. Tixier, M. Vazirgiannis

Language model

- Goal: determine $\mathrm{P}(s = \mathrm{w}_1 \dots \mathrm{w}_\mathrm{k})$ in some domain of interest

$$\mathrm{P}(s) = \prod_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{P}\,(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_1 \dots \mathrm{w}_{\mathrm{i}-1})$$

e.g., $\mathrm{P}\,(\mathrm{w}_1\mathrm{w}_2\mathrm{w}_3) = \mathrm{P}\,(\mathrm{w}_1)\,\mathrm{P}\,(\mathrm{w}_2 \mid \mathrm{w}_1)\,\mathrm{P}\,(\mathrm{w}_3 \mid \mathrm{w}_1\mathrm{w}_2)$

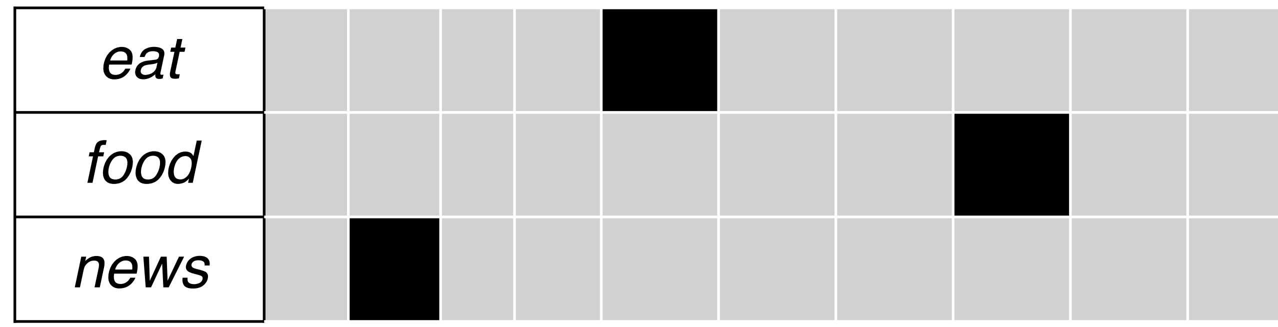
- Traditional n-gram language model assumption:
  "the probability of a word depends only on **context** of $\mathrm{n}-1$ previous words"

$$\Rightarrow \widehat{\mathrm{P}}(s) = \prod_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{P}\,(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1})$$

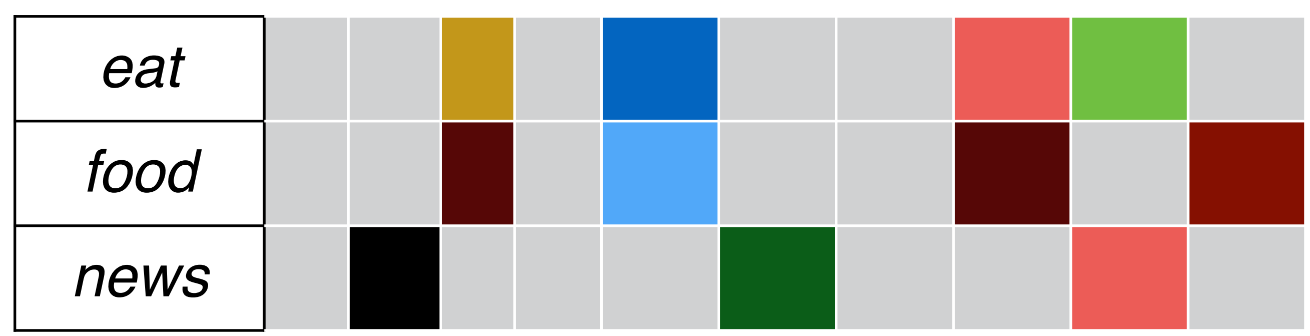
- Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute $\widehat{\mathrm{P}}\,(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1}) = \frac{\#\mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1}\mathrm{w}_\mathrm{i}}{\#\mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1}}$ on training corpus
  2. smooth to avoid zero probabilities

Representing Words

- **One-hot vector**
  - high dimensionality
  - sparse vectors
  - dimensions=|V| (10^6<|V|)
  - unable to capture semantic similarity between words



- **Distributional vector**
  - words that occur in similar contexts, tend to have similar meanings
  - each word vector contains the frequencies of all its neighbors
  - dimensions=|V|
  - computational complexity for ML algorithms



Representing Words

- **Word embeddings**
  - store the same contextual information in a low-dimensional vector
  - **densification** (sparse to dense)
  - **compression**
    - dimensionality reduction
    - dimensions=m 100<m<500
  - able to capture semantic similarity between words
  - learned vectors (unsupervised)
  - Learning methods
    - SVD
    - word2vec
    - GloVe

Example

- We should assign similar probabilities (discover similarity) to *Obama speaks to the media in Illinois* and the *President addresses the press in Chicago*

- This does not happen because of the “one-hot” vector space representation

One hot

$$\left.\begin{aligned} \text{obama} &= [0\,0\,0\,0\ \dots\ 0\,1\,0\,0] \\ \text{president} &= [0\,0\,0\,1\ \dots\ 0\,0\,0\,0] \end{aligned}\right\} \overrightarrow{\text{obama}}.\overrightarrow{\text{president}} = \vec{0}$$

$$\left.\begin{aligned} \text{speaks} &= [0\,0\,1\,0\ \dots\ 0\,0\,0\,0] \\ \text{addresses} &= [0\,0\,0\,0\ \dots\ 0\,0\,1\,0] \end{aligned}\right\} \overrightarrow{\text{speaks}}.\overrightarrow{\text{addresses}} = \vec{0}$$

$$\left.\begin{aligned} \text{illinois} &= [1\,0\,0\,0\ \dots\ 0\,0\,0\,0] \\ \text{chicago} &= [0\,1\,0\,0\ \dots\ 0\,0\,0\,0] \end{aligned}\right\} \overrightarrow{\text{illinois}}.\overrightarrow{\text{chicago}} = \vec{0}$$

Word embeddings

$$w_i \in V \xrightarrow{\text{mapping C}} \mathbb{R}^m$$

$$\underbrace{\text{obama} = \underset{|V|}{\underleftrightarrow{[\overset{\substack{w_1\\ \downarrow}}{0} \dots \dots \overset{\substack{\text{obama}\\ \downarrow}}{1} \dots \dots \overset{\substack{w_{|V|}\\ \downarrow}}{0}]}}}_{\text{“one-hot” vector}} \qquad \underbrace{\text{obama} = \underset{m \ll |V|}{\underleftrightarrow{[\overset{\substack{\text{feature}_1\\ \downarrow}}{0.12} \dots \overset{\substack{\text{feature}_m\\ \downarrow}}{-0.25}]}}}_{\text{feature vector}}$$

SVD word embeddings

- Dimensionality reduction on co-occurrence matrix
- Create a |V|x|V| word co-occurrence matrix X
- Apply SVD $X = USV^T$
- Take first k columns of U
- Use the k-dimensional vectors as representations for each word
- Able to capture semantic and syntactic similarity

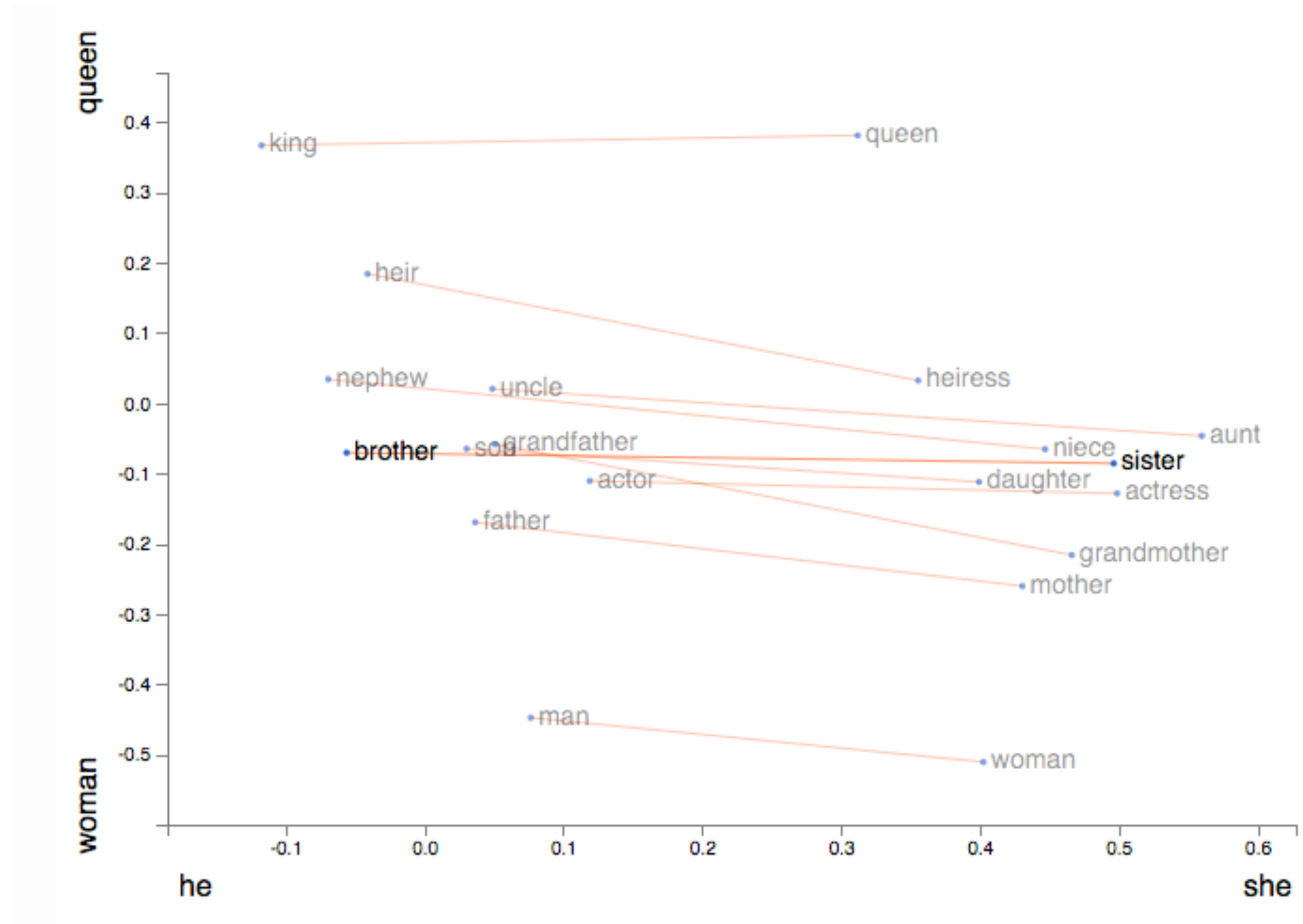
SVD problems

- The dimensions of the matrix change when dictionary changes
- The whole decomposition must be re-calculated when we add a word
- Sensitive to the imbalance in word frequency
- Very high dimensional matrix
- Not suitable for millions of words and documents
- Quadratic cost to perform SVD
- Solution: Directly calculate a low-dimensional representation

Word analogy

- ➢ Words with similar meaning end up laying close to each other
- ➢ Words that share similar contexts may be analogous
  - Synonyms
  - Antonyms
  - Names
  - Colors
  - Places
  - Interchangeable words

- ➢ Vector arithmetics to work with analogies
- ➢ i.e. **king - man + woman** = **queen**



https://lamyiowce.github.io/word2viz/

But why?

- what's an analogy?

$$\frac{p(w'|man)}{p(w'|woman)} \approx \frac{p(w'|king)}{p(w'|queen)}$$

Assume we have vectors s.t.

1. $PMI(w', w) \approx v_w v_{w'}$ *inner product*
2. Isotropic: $E_{w'}[(v_{w'} v_u)]^2 = ||v_u||^2$

Then

3. $argmin_w \, E_{w'}[ln \frac{p(w'|w)}{p(w'|queen)} - ln \frac{p(w'|man)}{p(w'|woman)}]^2$
4. $argmin_w \, E_{w'}[\big(PMI(w'|w) - PMI(w'|queen)\big) - (PMI(w'|man) - PMI(w'|woman))]^2$
5. $argmin_w ||\big(v_w - v_{queen}\big) - (v_{man} - v_{woman})||^2$
6. $v_w \approx v_{queen} - v_{woman} + v_{man}$ which is an analogy!

- *Arora et al* shows that if (2) holds then (1) holds as well
- So we need to construct vectors from co-occurrence that satisfy (2)
- d<<|V| in order to have isotropic vectors

Learning Word Vectors

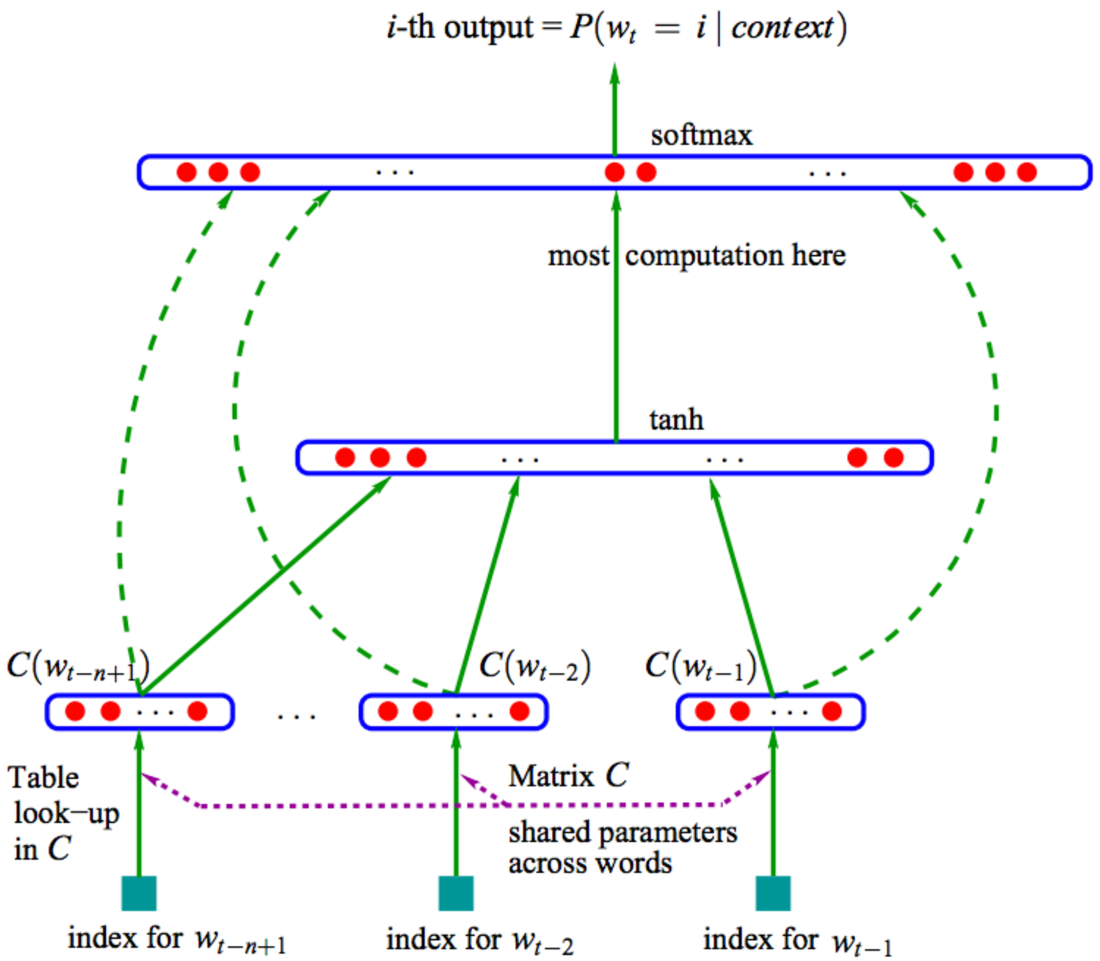
- Corpus containing a sequence of T training words
- Objective: $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$
- Decomposed in two parts:

$$w_i \in V \xrightarrow{\text{mapping C}} \mathbb{R}^m$$

  - Mapping C (1-hotv => lower dimensions)
  - Mapping any g s.t. (estimate prob t+1| t previous)

$$f(w_{t-1}, \cdots, w_{t-n+1}) = g(C(w_{t-1}), \cdots, C(w_{t-n+1}))$$

- C(i) is the i-th word feature vector (Word embedding)
- Objective function: $J = \frac{1}{T} \sum f(w_t, \ldots, w_{t-n+1})$

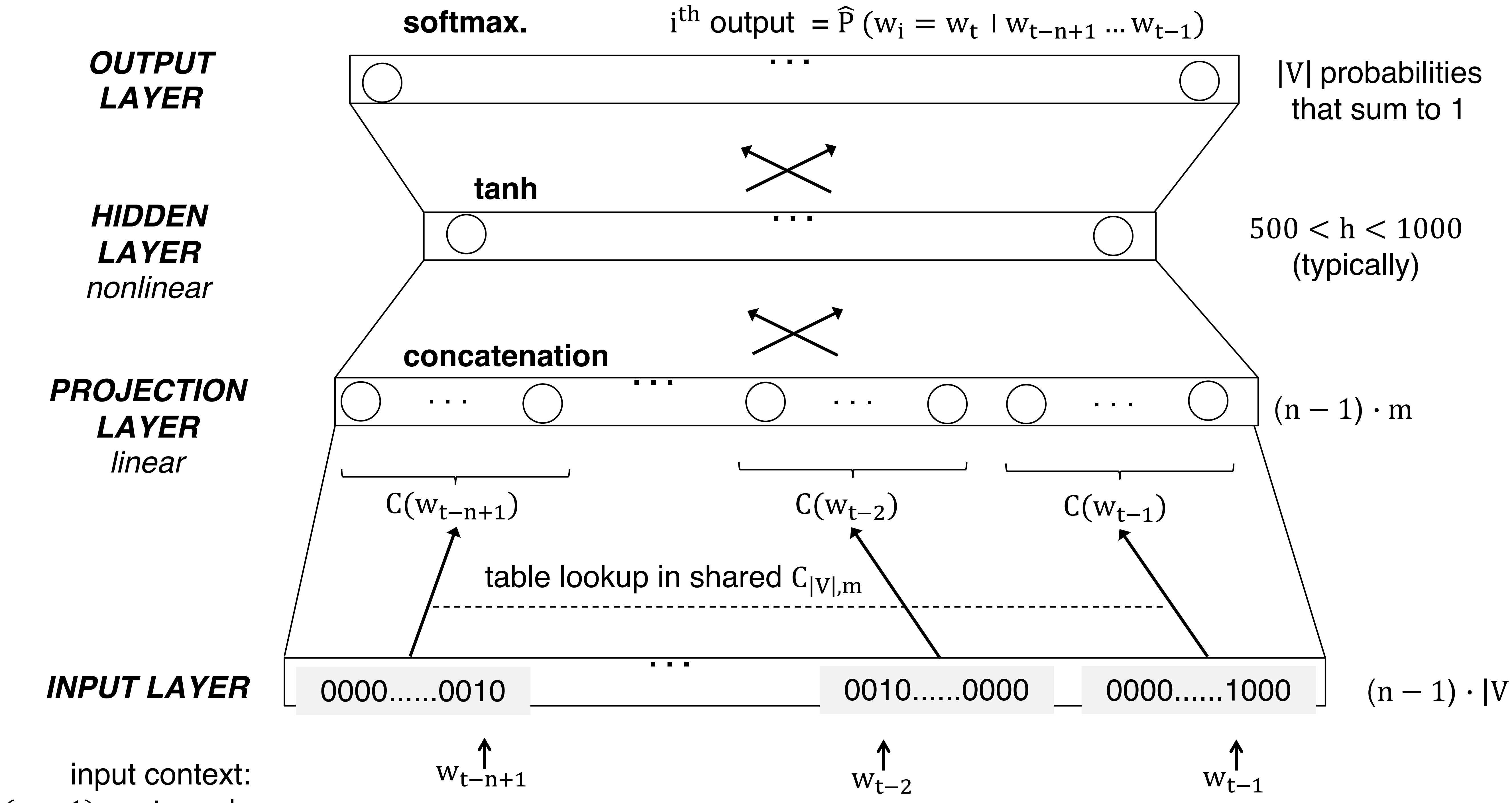


[Bengio, Yoshua, et al. "A neural probabilistic language model." *The Journal of Machine Learning Research* 3 (2003): 1137-1155.]

Neural Net Language Model

For each training sequence: input = (context, target) pair: $(w_{t-n+1} \dots w_{t-1}, w_t)$

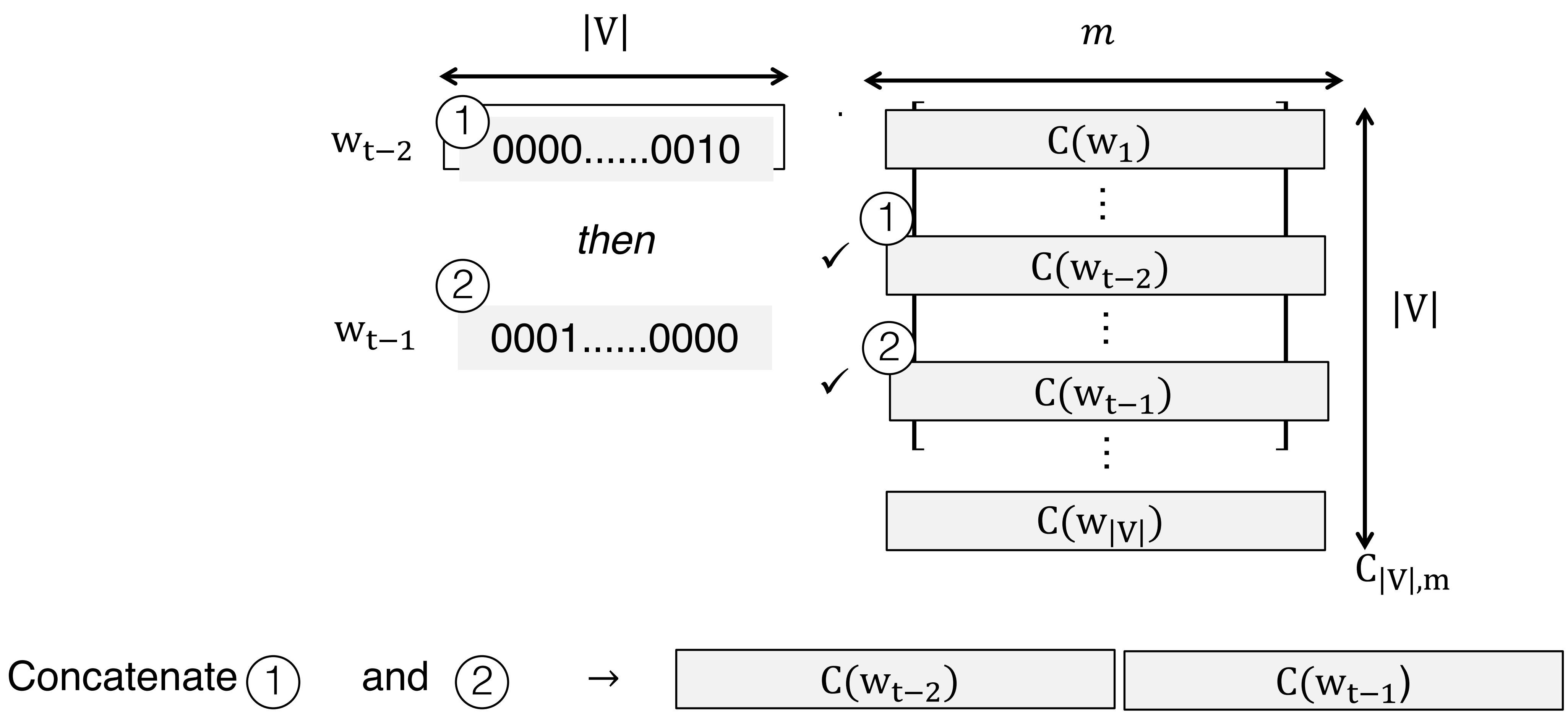
objective: minimize $E = -\log \widehat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$



NNLM Projection layer

- Performs a simple table lookup in $C_{|V|,m}$: concatenate the rows of the shared mapping matrix $C_{|V|,m}$ corresponding to the context words

Example for a two-word context $w_{t-2}w_{t-1}$:



- $C_{|V|,m}$ is **critical**: it contains the weights that are tuned at each step. After training, it contains what we're interested in: the **word vectors**

NNLM hidden/output layers and training

- Softmax (log-linear classification model) is used to output positive numbers that sum to one (a multinomial probability distribution):

  for the i^{th} unit in the output layer: $\widehat{\mathrm{P}}(\mathrm{w_i} = \mathrm{w_t} \mid \mathrm{w_{t-n+1}} \ldots \mathrm{w_{t-1}}) = \frac{\mathrm{e}^{y_{w_i}}}{\sum_{i'=1}^{|V|} \mathrm{e}^{y_{w_{i'}}}}$

  Where:
  - $y = b + U.tanh(d + H.x)$
  - tanh : nonlinear squashing (link) function
  - x : concatenation $\mathrm{C(w)}$ of the context weight vectors seen previously
  - b : output layer biases ($|\mathrm{V}|$ elements)
  - d : hidden layer biases (h elements). Typically $500 < \mathrm{h} < 1000$
  - U : $|\mathrm{V}|$ * h matrix storing the *hidden-to-output* weights
  - H : (h * $(\mathrm{n}-1)\mathrm{m}$) matrix storing the *projection-to-hidden* weights

  $\rightarrow \boldsymbol{\theta} = (\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{U}, \boldsymbol{H}, \boldsymbol{C})$

  - Complexity per training sequence: $\mathrm{n} * \mathrm{m} + \mathrm{n} * \mathrm{m} * \mathrm{h} + \mathbf{h} * |\mathbf{V}|$

    computational bottleneck: **nonlinear hidden layer** ($\mathrm{h} * |\mathrm{V}|$ term)

- **Training** is performed via stochastic gradient descent (learning rate ε):

$$\theta \leftarrow \theta + \varepsilon \cdot \frac{\partial E}{\partial \theta} = \theta + \varepsilon \cdot \frac{\partial \log \widehat{P}\,(w_t \mid w_{t-n+1} \ldots w_{t-1})}{\partial \theta}$$

(weights are initialized randomly, then updated via backpropagation)

NNLM facts

- tested on Brown (1.2M words, $|V| \cong$ 16K) and AP News (14M words, $|V| \cong$ 150K reduced to 18K) corpuses
- Brown: $h = 100$, $n = 5$, $m = 30$
- AP News: $h = 60$, $n = 6$, $m = 100$, **3 week** training using **40 cores**
- 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs in terms of test *set perplexity*: geometric average of $1/\widehat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$
- Due to **complexity**, NNLM can't be applied to large data sets → poor performance on rare words
- Bengio et al. (2003) initially thought their main contribution was a more accurate LM. They let the interpretation and use of the word vectors as **future work**
- On the opposite, Mikolov et al. (2013) focus on the **word vectors**

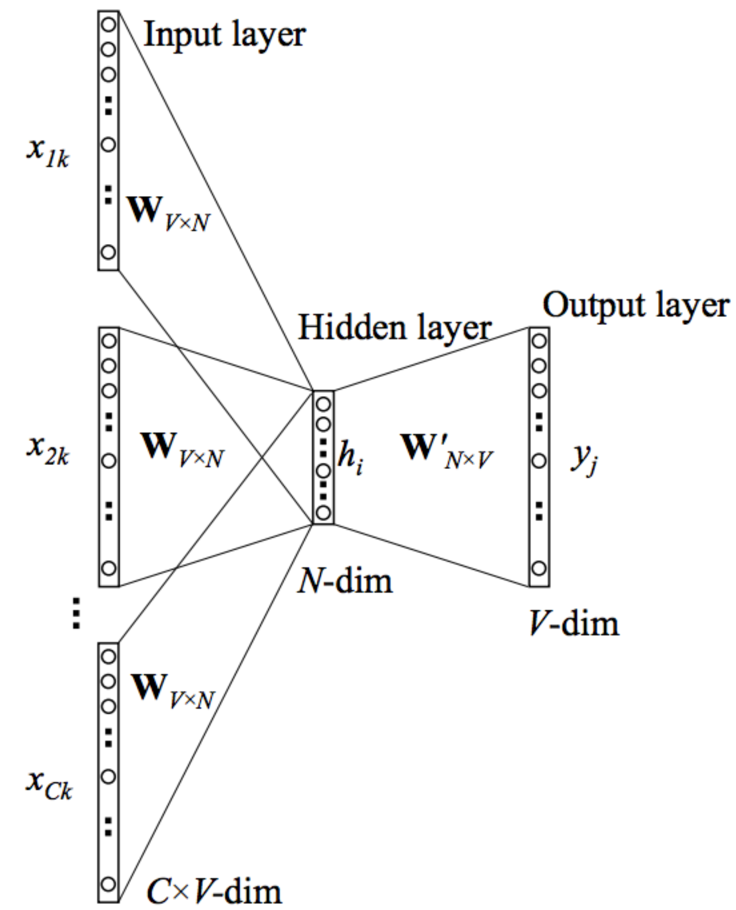
Word2Vec

- Mikolov et al. in 2013
- Key idea of word2vec: achieve better performance not by using a more complex model (i.e., with more layers), but by allowing a **simpler (shallower) model** to be trained on **much larger amounts of data**
- no hidden layer (leads to 1000X speedup)
- projection layer is shared (not just the weight matrix) - C
- context: words from both history & future:

- Two algorithms for learning words vectors:
  - **CBOW**: from context predict target
  - **Skip-gram**: from target predict context

CBOW

- continuous bag-of-words
- continuous representations whose order is of no importance
- uses the surrounding words to predict the center word
- n-words before and after the target word

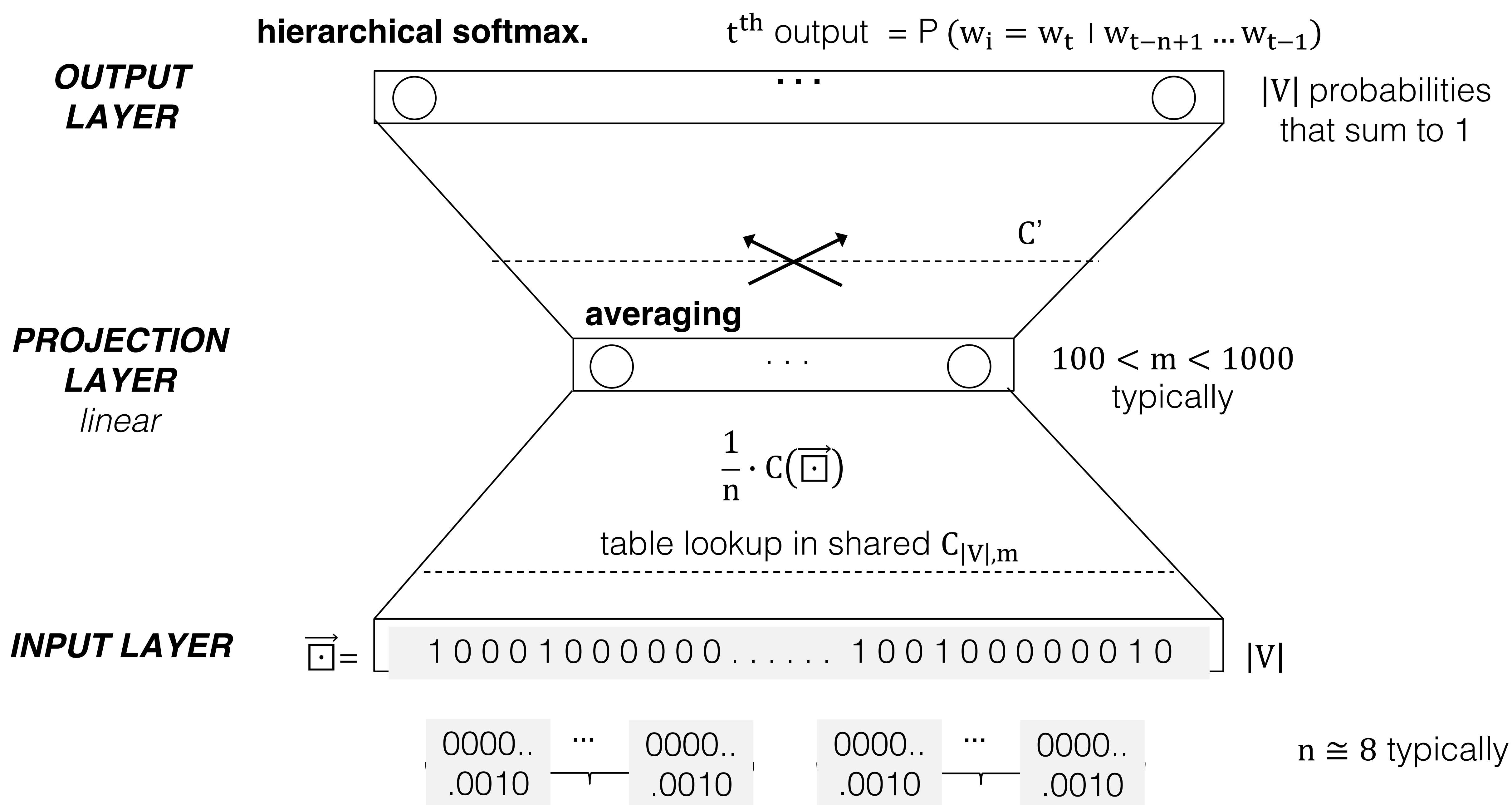


Efficient Estimation of Word Representations in Vector Space- Mikolov et al.

Continuous Bag-of-Words (CBOW)

For each training sequence: input = (context, target) pair: $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

objective: minimize $-\log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$

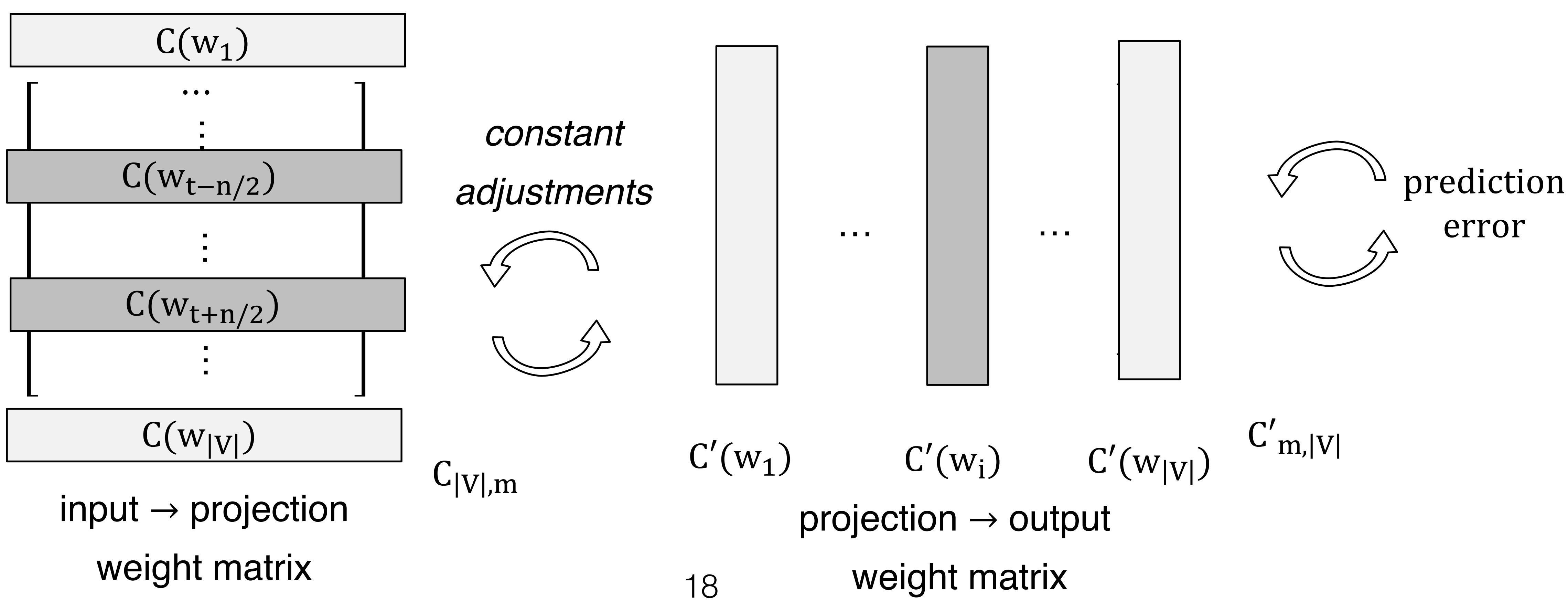


input context: $n/2$ history words: $w_{t-\frac{n}{2}} \dots w_{t-1}$ $n/2$ future words: $w_{t+1} + \cdots + w_{t+\frac{n}{2}}$

Weight updating

- For each (context, target=w_t) pair, only the word vectors from matrix C corresponding to the context words are updated
- Recall that we compute P $(w_i = w_t \mid \text{context})\ \forall\, w_i \in V$. We compare this distribution to the true probability distribution (1 for w_t, 0 elsewhere)
- **Back propagation**
- If P $(w_i = w_t \mid \text{context})$ is **overestimated** (i.e., > 0, happens in potentially $|V| - 1$ cases), some portion of $C'(w_i)$ is **subtracted** from the context word vectors in C, proportionally to the magnitude of the error
- Reversely, if P $(w_i = w_t \mid \text{context})$ is **underestimated** (< 1, happens in potentially 1 case), some portion of $C'(w_i)$ is **added** to the context word vectors in C

→ at each step the words move away or get closer to each other in the feature space → clustering



Skip-gram

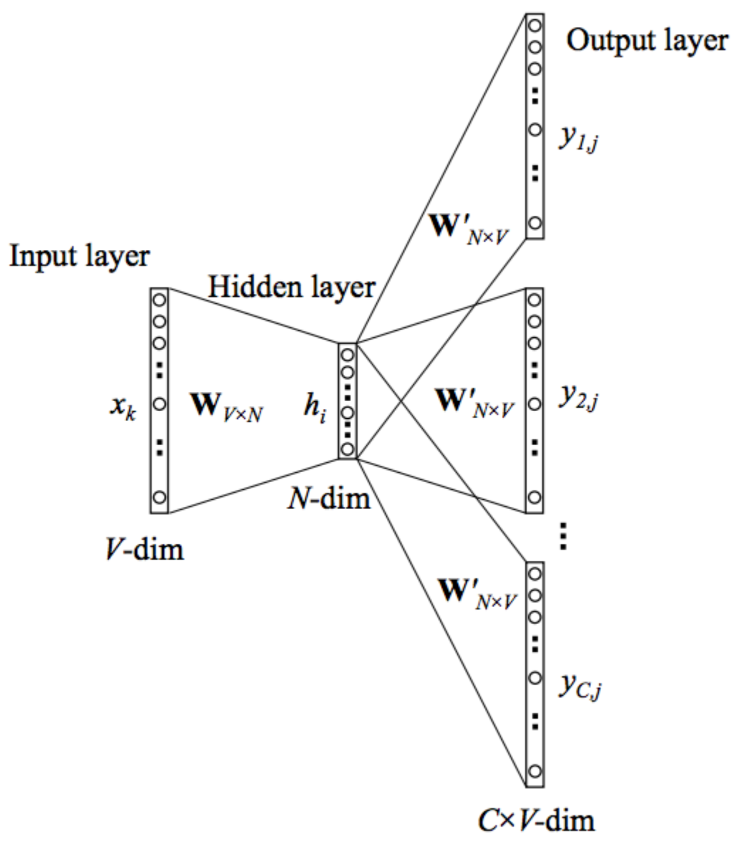
- skip-gram uses the center word to predict the surrounding words

- instead of computing the probability of the target word w_t given its previous words, we calculate the probability of the surrounding word w_{t+j} given w_t

- $$p(w_{t+j}|w_t) = \frac{\exp(v_{w_t}^T v'_{w_{t+j}})}{\sum_{w \in V} \exp(v_{w_t}^T v'_{w_{t+j}})}$$

- Objective function

$$J = \frac{1}{T}\sum_{t=1}^{T} \sum_{-n \le j \le n} \log p(w_{t+j}|w_t)$$



Efficient Estimation of Word Representations in Vector Space- Mikolov et al.

word2vec facts

- Complexity is $\mathrm{n} * \mathrm{m} + \mathrm{m} * \log|\mathbf{V}|$ (Mikolov et al. 2013a)
- On Google news 6B words training corpus, with $|\mathbf{V}| \sim 10^6$:
  - CBOW with $\mathrm{m} = 1000$ took **2 days** to train on **140 cores**
  - Skip-gram with $\mathrm{m} = 1000$ took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for $\mathrm{m} = 100$ only!
    (note that $\mathrm{m} = 1000$ was not reasonably feasible on such a large training set)
- word2vec training speed ≅ 100K-5M words/s

- Quality of the word vectors:
  - ↗ significantly with **amount of training data** and **dimension of the word vectors** (m), with diminishing relative improvements
  - measured in terms of accuracy on 20K semantic and syntactic association tasks.
    e.g., words in **bold** have to be returned:

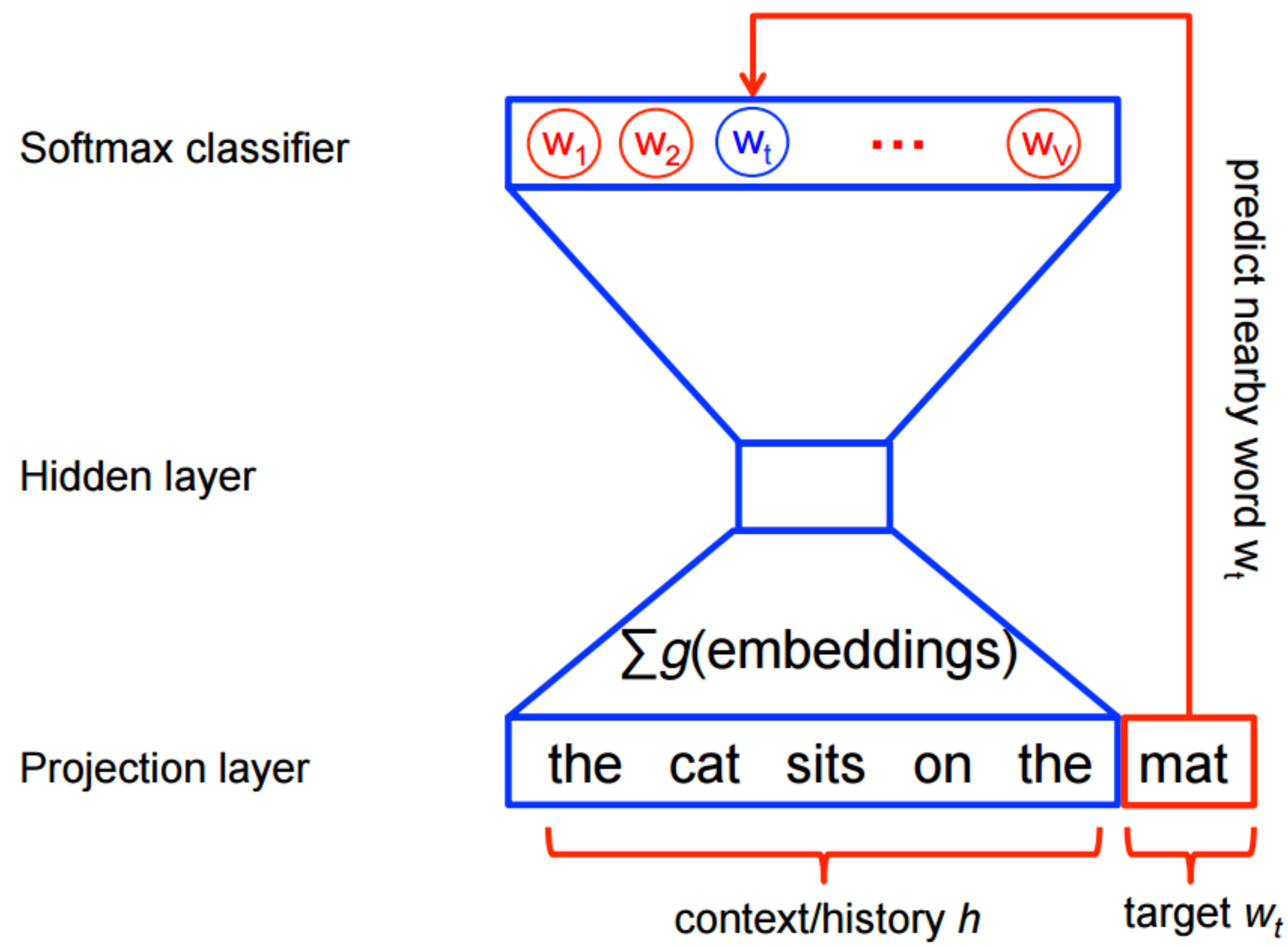
Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%

- References: http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd
  https://code.google.com/p/word2vec/

Softmax

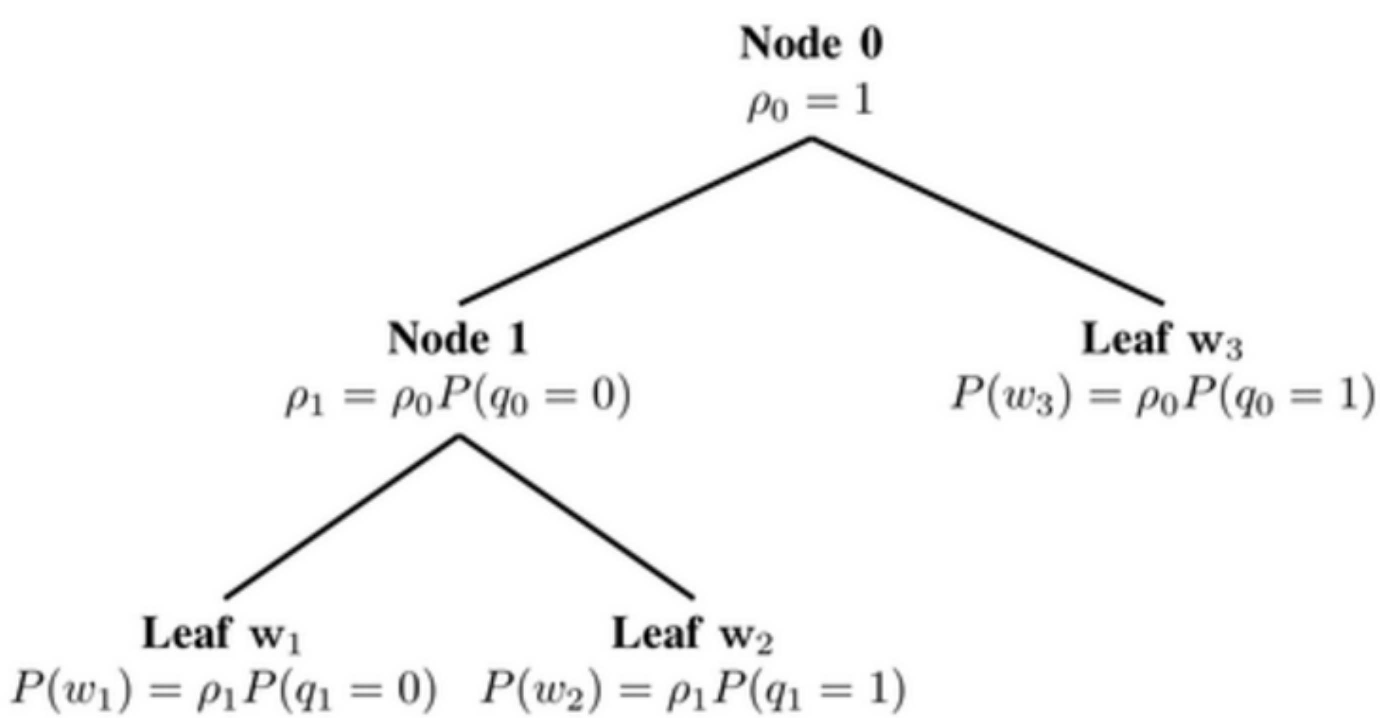
- Calculating the softmax is expensive
- Approximate the softmax
  - softmax-based
    - Hierarchical Softmax
    - Differentiated Softmax
    - CNN-Softmax
  - sampling-based
    - Importance Sampling
    - Adaptive Importance Sampling
    - Target Sampling
    - Noise Contrastive Estimation
    - *Negative Sampling*
    - Self-Normalisation
    - Infrequent Normalisation



https://www.tensorflow.org/tutorials/word2vec/

Hierarchical Softmax

- Inspired by binary trees
- Flat softmax layer -> hierarchical layer that has the words as leaves
- Morin and Bengio
- Skip the expensive normalization over all words
- Speed-up 50xM via log(V) search in the tree.



Stephan Gouws **(Quora)**

Sampling-based Approaches

- Approximate the normalization of the softmax with some cheap to compute loss at training time
- The update rule consists of:
  - positive reinforcement for the target word
  - negative reinforcement for all other words
- approximate this negative reinforcement in less complex manner
- Without calculating the sum over the probabilities for all words in V but doing i.e. Monte Carlo

1. $J = -log \frac{\exp(h^T v'_{w_t})}{\sum_{w \in V} \exp(h^T v'_{w_i})}$

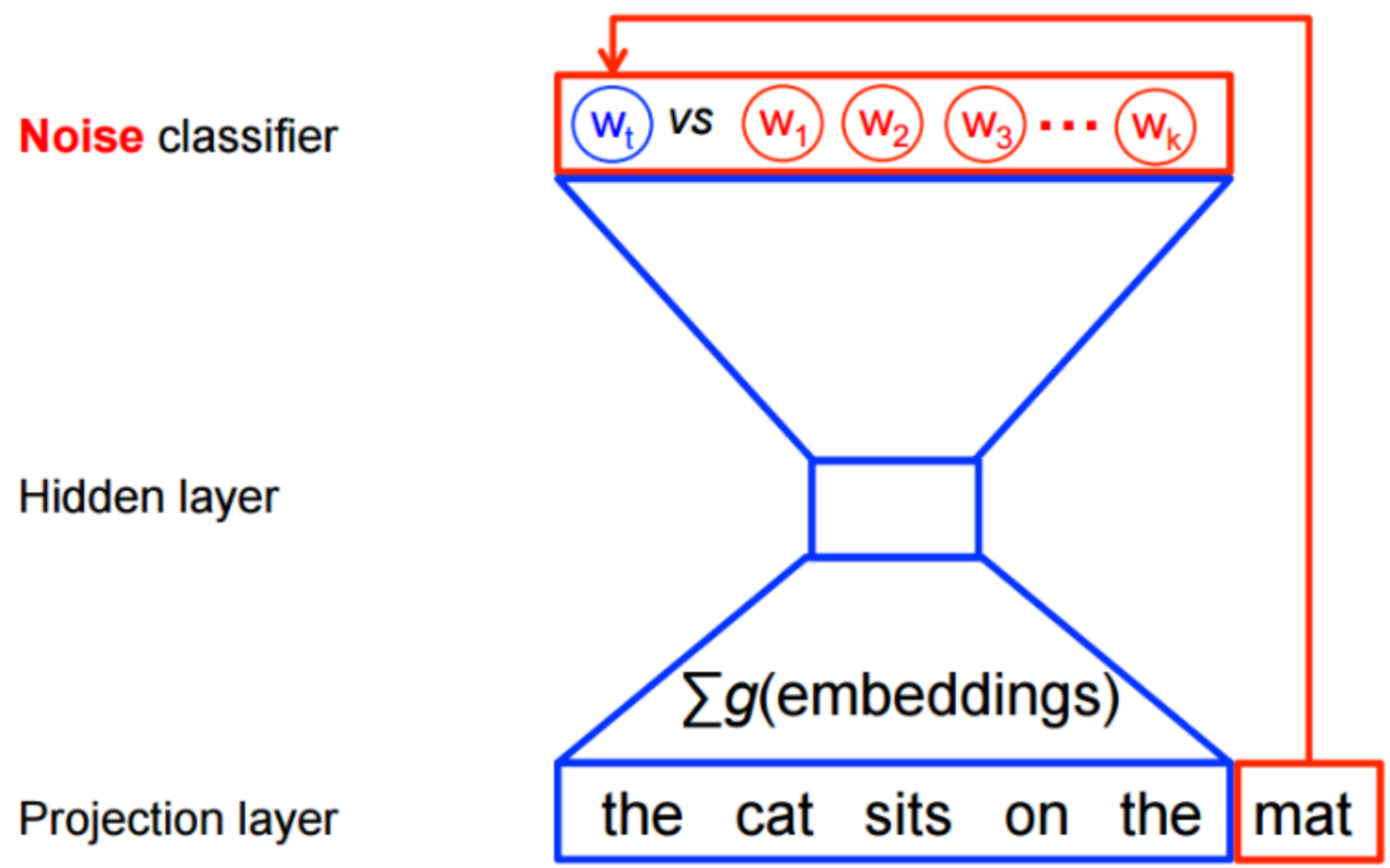
$\mathcal{E}(w) = -h^T v'_{w_t}$ and P=softmax probability

2. $\nabla \mathrm{J} = \nabla \mathcal{E}(w) - \sum_{w \in V} P(w_i) \nabla \mathcal{E}(w_i)$

3. $\sum_{w \in V} P(w_i) \nabla \mathcal{E}(w_i) = \mathbb{E}_{w_i \sim P} \nabla \mathcal{E}(w_i)$

Negative Sampling

- For every word w_i given its context c_i generate k noise samples $\widetilde{w_{ik}}$ from a noise distribution Q
- Separate the correct words from the noise(binary classification)
- logistic regression to minimize the negative log-likelihood
- Also optimizes the goal of maximizing the probability of correct words
- Replacing the more expensive softmax



https://www.tensorflow.org/tutorials/word2vec/

Which Approach?

Approach	Speed-up factor	During training?	During testing?	Performance (small vocab)	Performance (large vocab)	Proportion of parameters
Softmax	1x	-	-	very good	very poor	100%
Hierarchical Softmax	25x (50-100x)	X	-	very poor	very good	100%
Differentiated Softmax	2x	X	X	very good	very good	< 100%
CNN-Softmax	-	X	-	-	bad - good	30%
Importance Sampling	(19x)	X	-	-	-	100%
Adaptive Importance Sampling	(100x)	X	-	-	-	100%
Target Sampling	2x	X	-	good	bad	100%
Noise Contrastive Estimation	8x (45x)	X	-	very bad	very bad	100%
Negative Sampling	(50-100x)	X	-	-	-	100%
Self-Normalisation	(15x)	X	-	-	-	100%
Infrequent Normalisation	6x (10x)	X	-	very good	good	100%

GloVe

- Count-based model
- Ratio of co-occurrence probabilities best distinguishes relevant words
- Pennington, et al
- Weighted *least squares regression* model
- X co-occurrence matrix
- f weighting function,
- b bias terms
- $w_i = word\ vector$
- $\widetilde{w_j} = context\ vector$

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

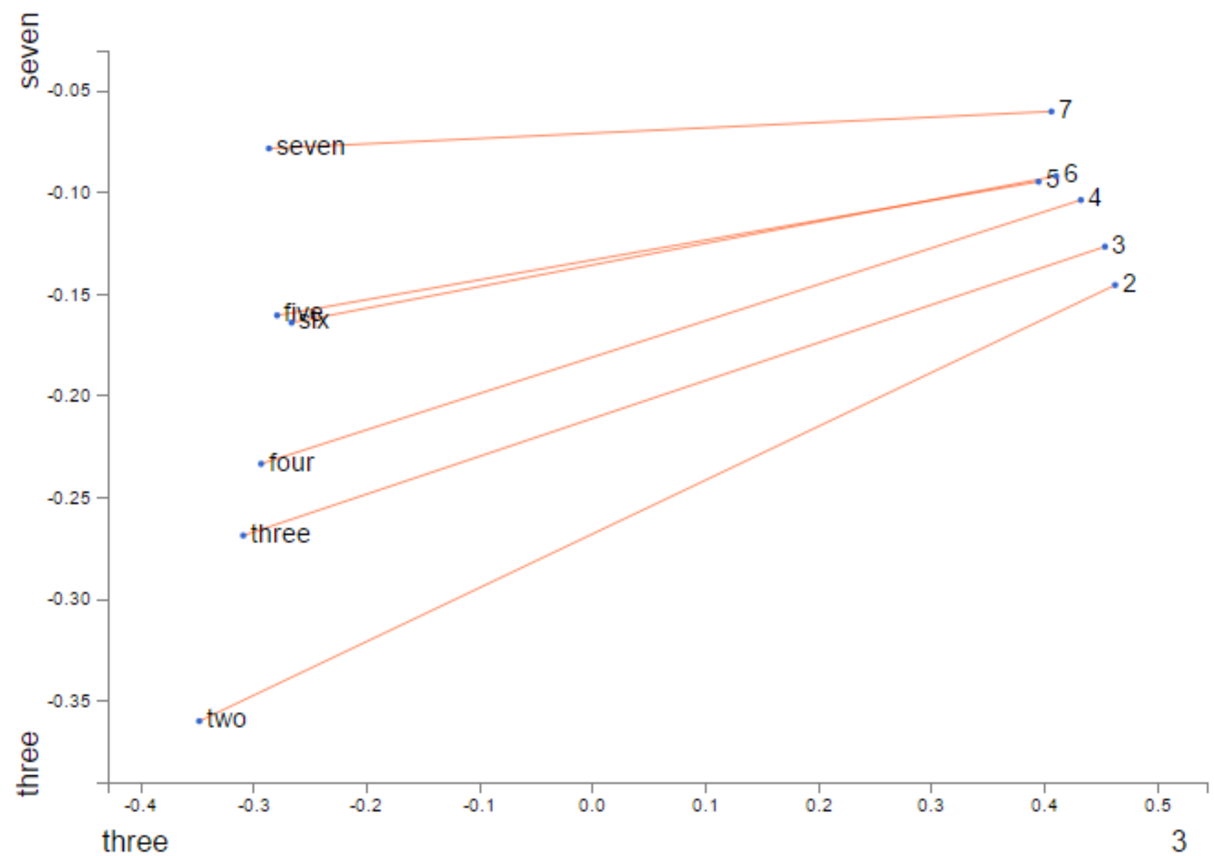
Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Which is one is better?

- Open question
- SVD vs word2vec vs GloVe
- All based on co-occurrence
- Levy, O., Goldberg, Y., & Dagan, I. (2015)
  - SVD performs best on similarity tasks
  - Word2vec performs best on analogy tasks
  - *No single algorithm consistently outperforms the other methods*
  - *Hyperparameter tuning is important*
  - 3 out of 6 cases, tuning hyperparameters is more beneficial than increasing corpus size
  - word2vec outperforms GloVe on all tasks
  - *CBOW is worse than skip-gram on all tasks*

Applications

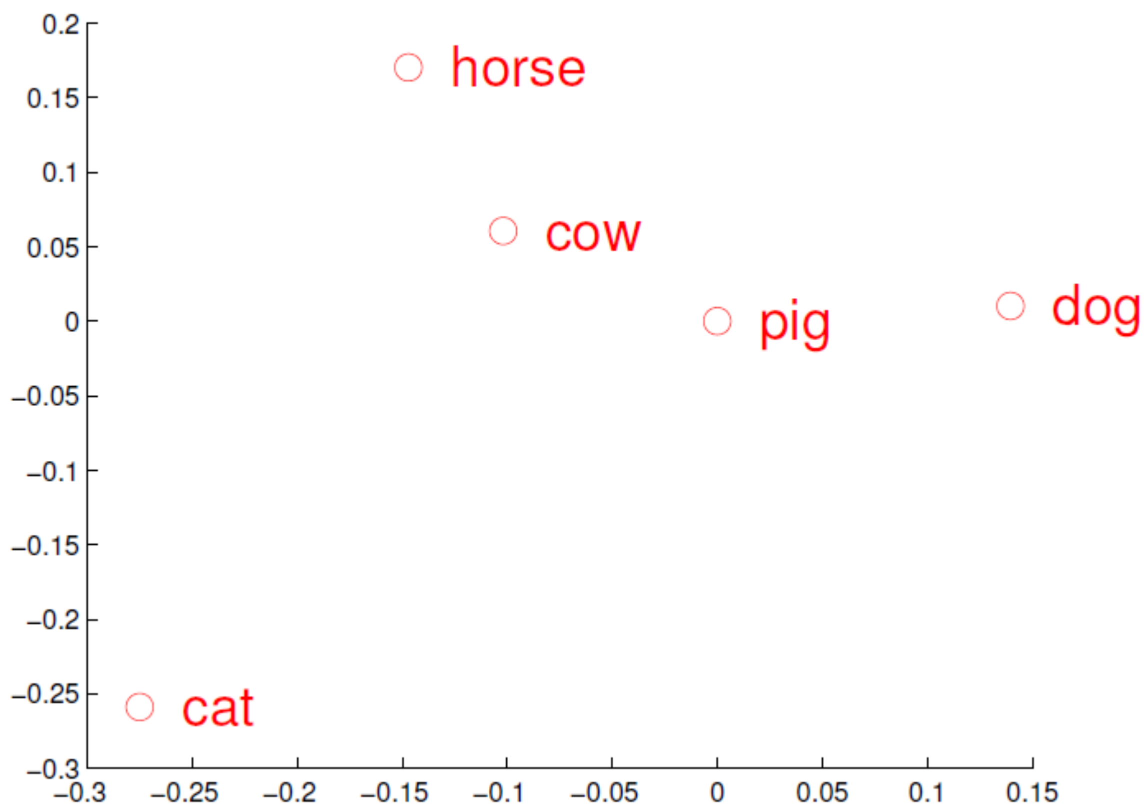
- Word analogies
- Find similar words
  - Semantic similarity
  - Syntactic similarity
- POS tagging
- Similar analogies for different languages
- Document classification

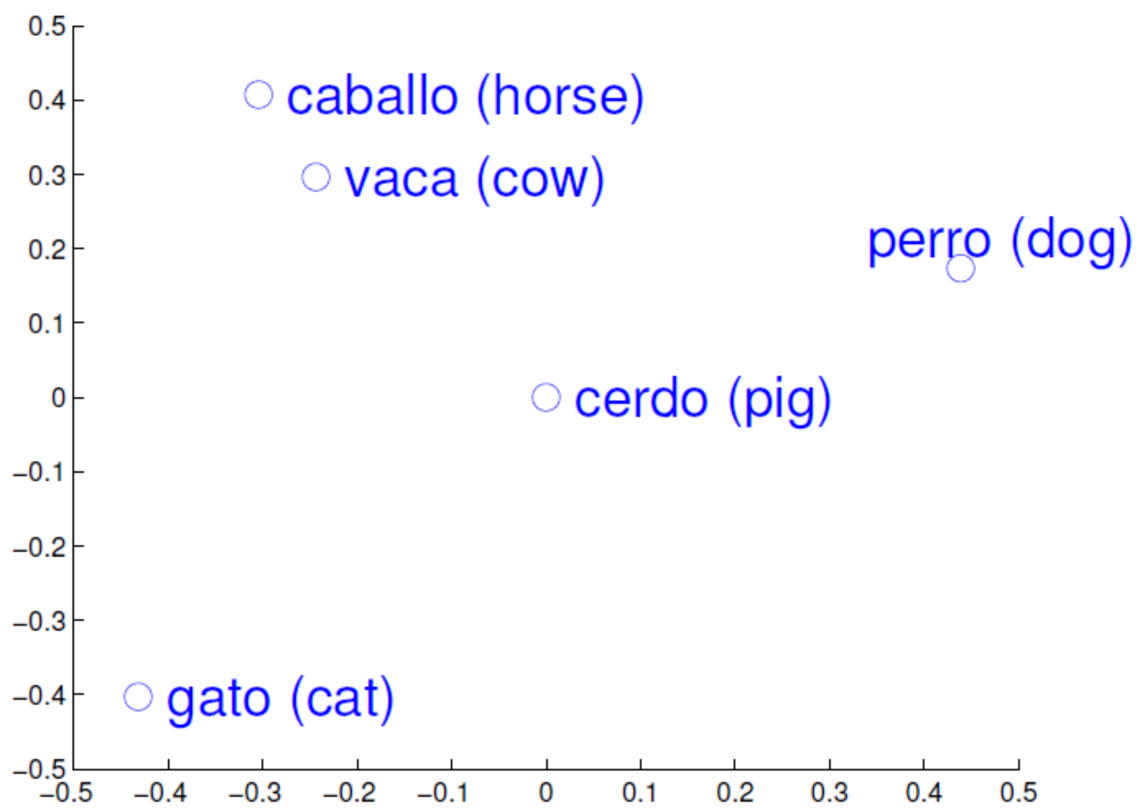


https://lamyiowce.github.io/word2viz/

Applications

- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval…
- Example for English to Spanish machine translation:

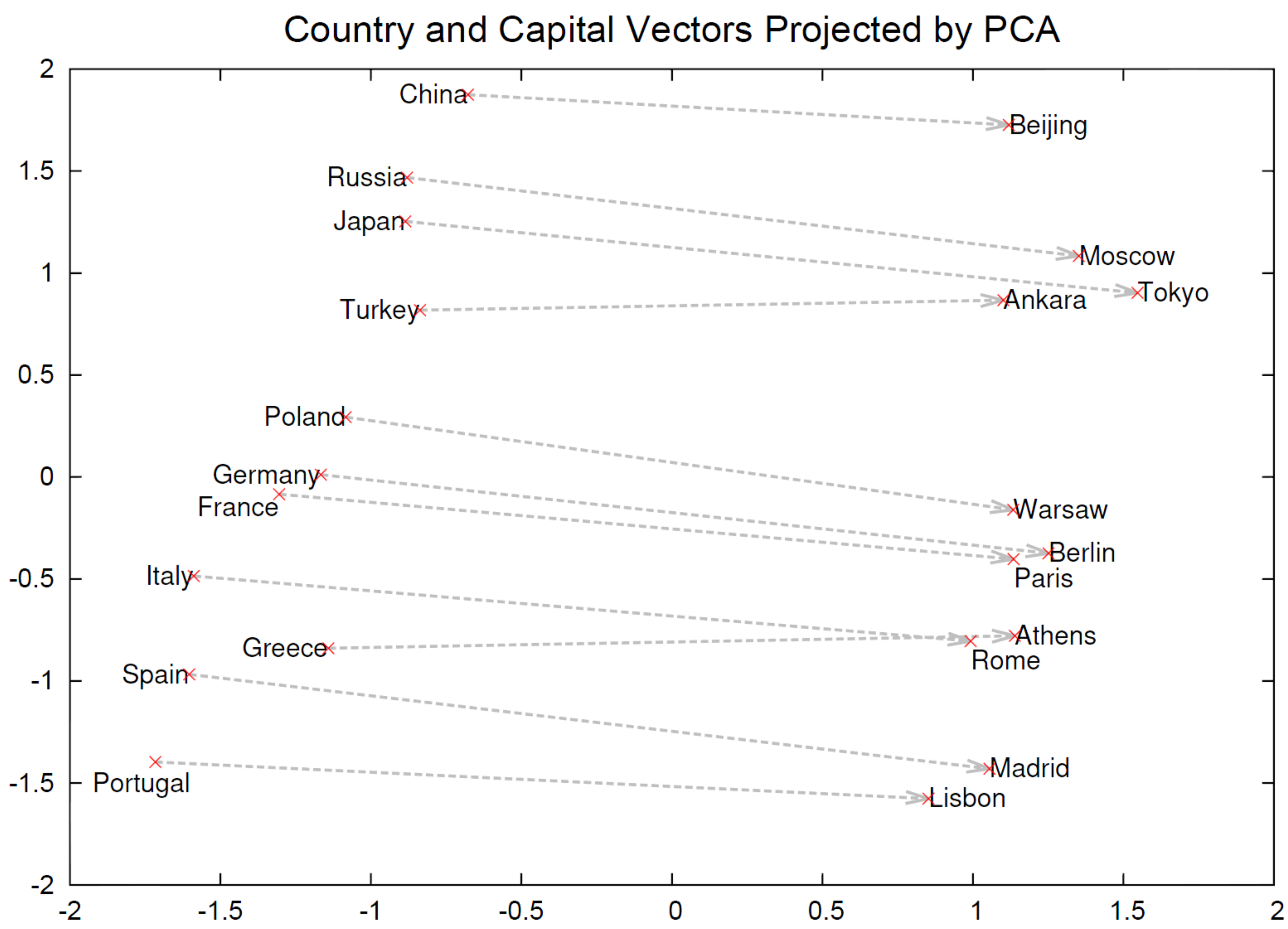




About 90% reported accuracy (Mikolov et al. 2013c)

Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168.*

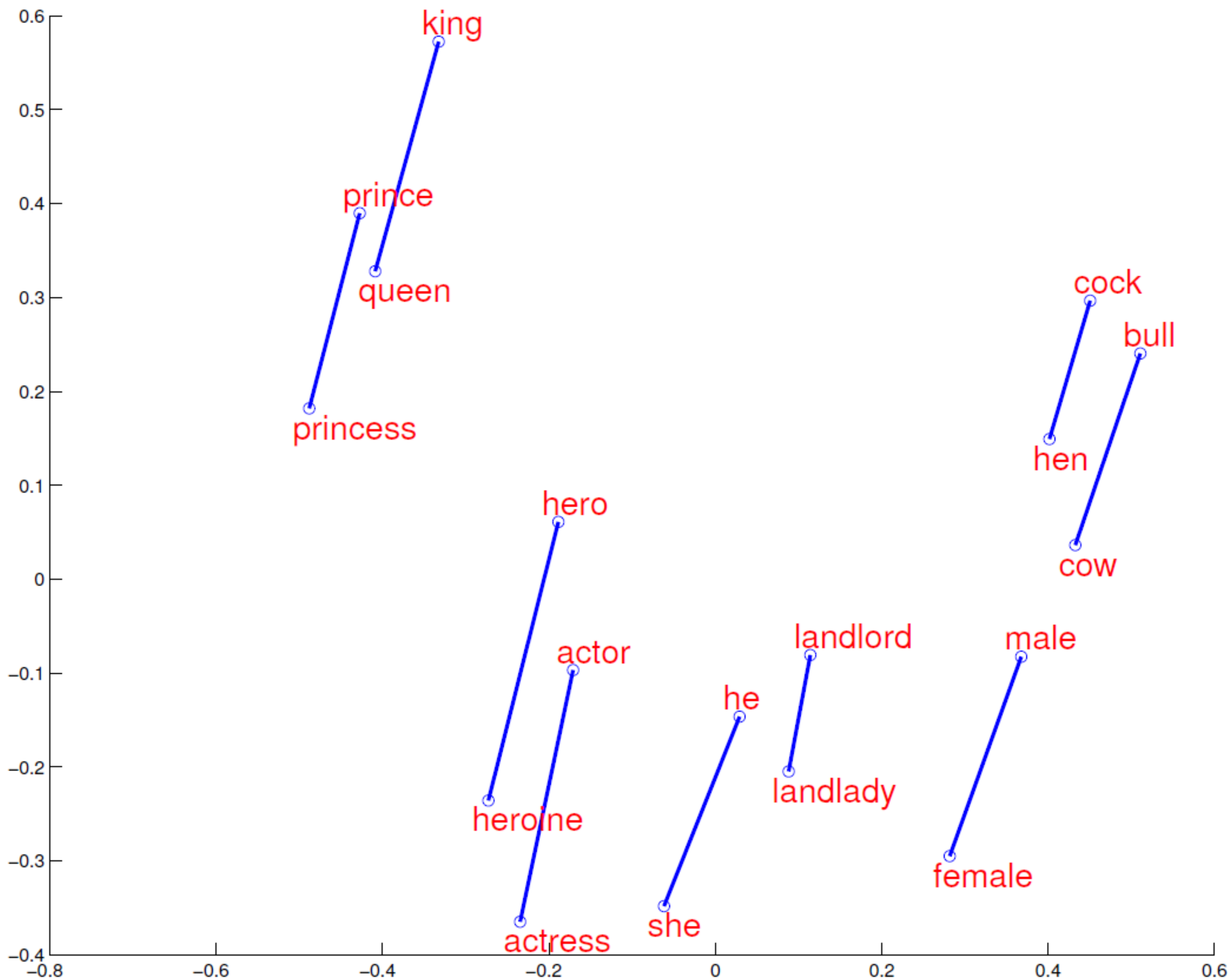
Remarkable properties of word vectors



regularities between words are encoded in the difference vectors
e.g., there is a constant **country-capital** difference vector

Mikolov et al. (2013b)
Distributed representations of words and phrases and their compositionality

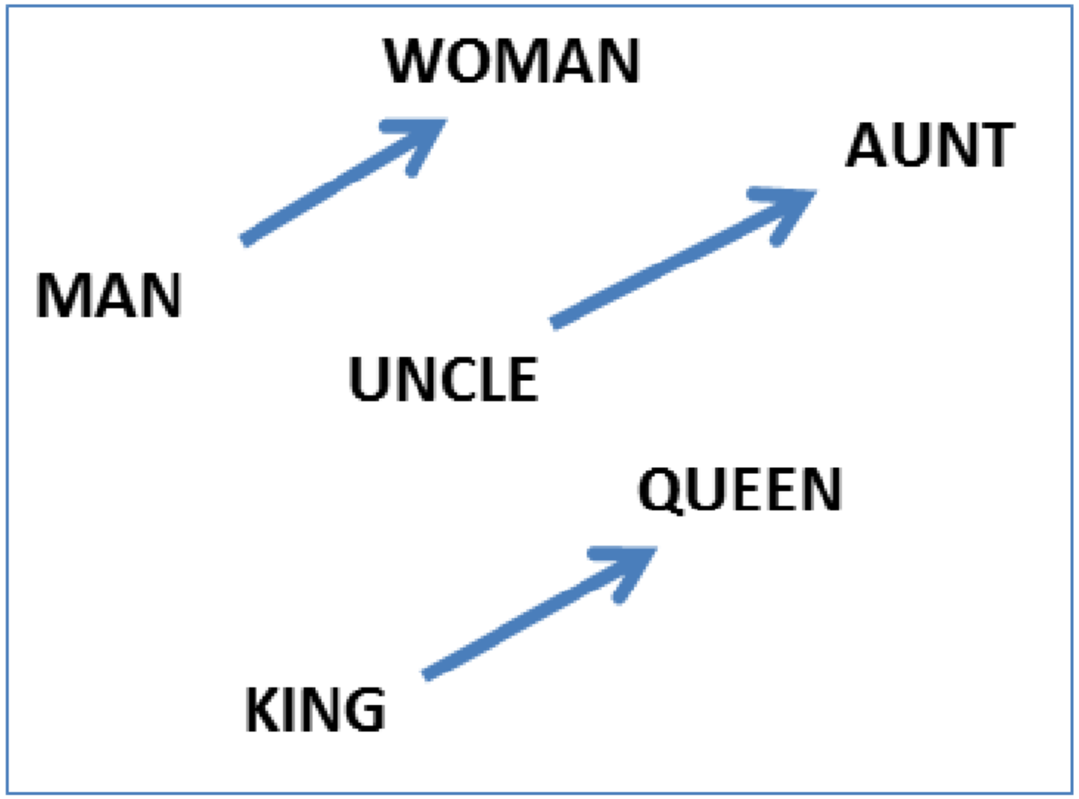
Remarkable properties of word vectors



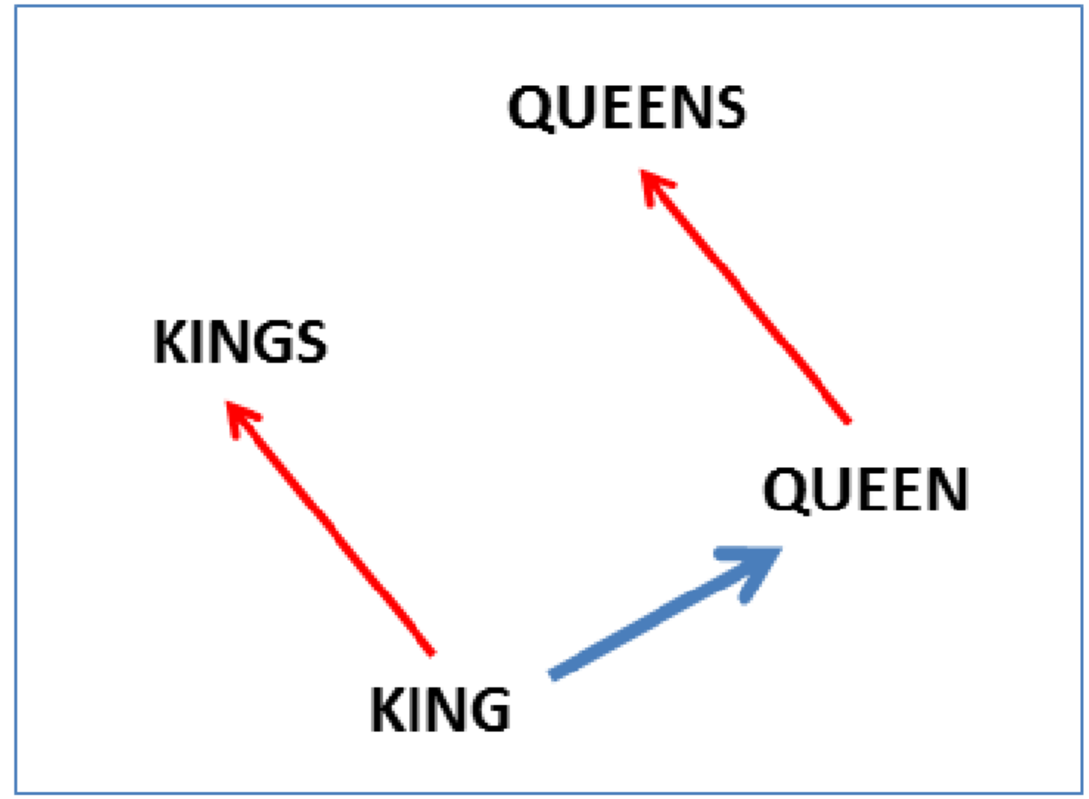
constant **female-male** difference vector

http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd

Remarkable properties of word vectors



constant **male-female** difference vector



constant **singular-plural** difference vector

- Vector operations are supported and make intuitive sense:

$$w_{king} - w_{man} + w_{woman} \cong w_{queen}$$

$$w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso}$$

$$w_{paris} - w_{france} + w_{italy} \cong w_{rome}$$

$$w_{his} - w_{he} + w_{she} \cong w_{her}$$

$$w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$$

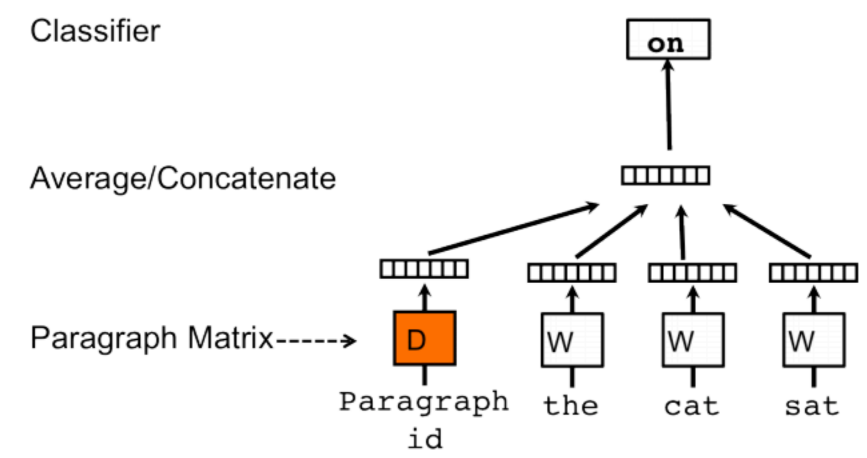
$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

- Online demo (scroll down to end of tutorial)

http://rare-technologies.com/word2vec-tutorial/

Distributed Representations of Sentences and Documents

- **Doc2vec**
- Paragraph or document vectors
- Capable of constructing representations of input sequences of variable length
- Represent each document by a dense vector
- Trained to predict words in the document
- paragraph vector and word vectors are averaged or concatenated to predict the next word in a context
- can be thought of as another word shared across all contexts in document

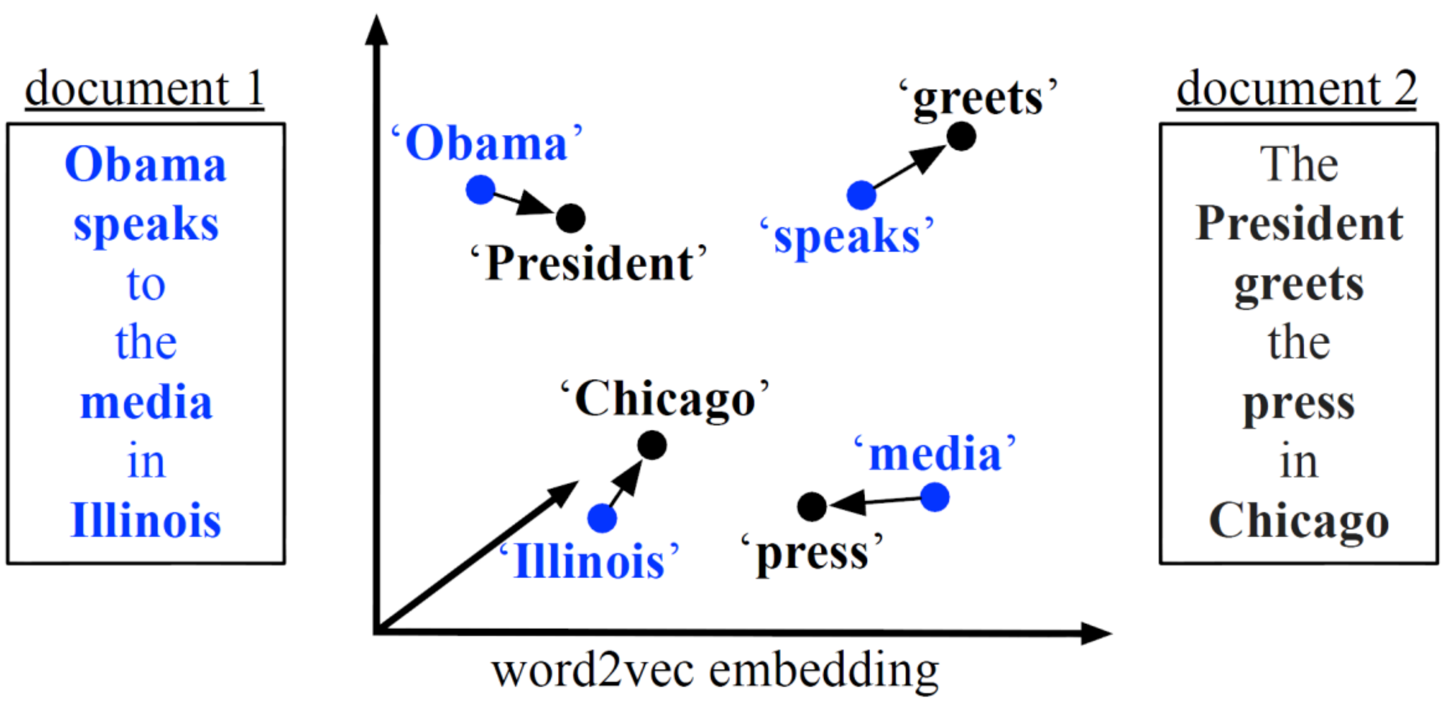


Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	**12.2%**	**51.3%**

https://cs.stanford.edu/~quocle/**paragraph_vector**.pdf

Word Mover's distance

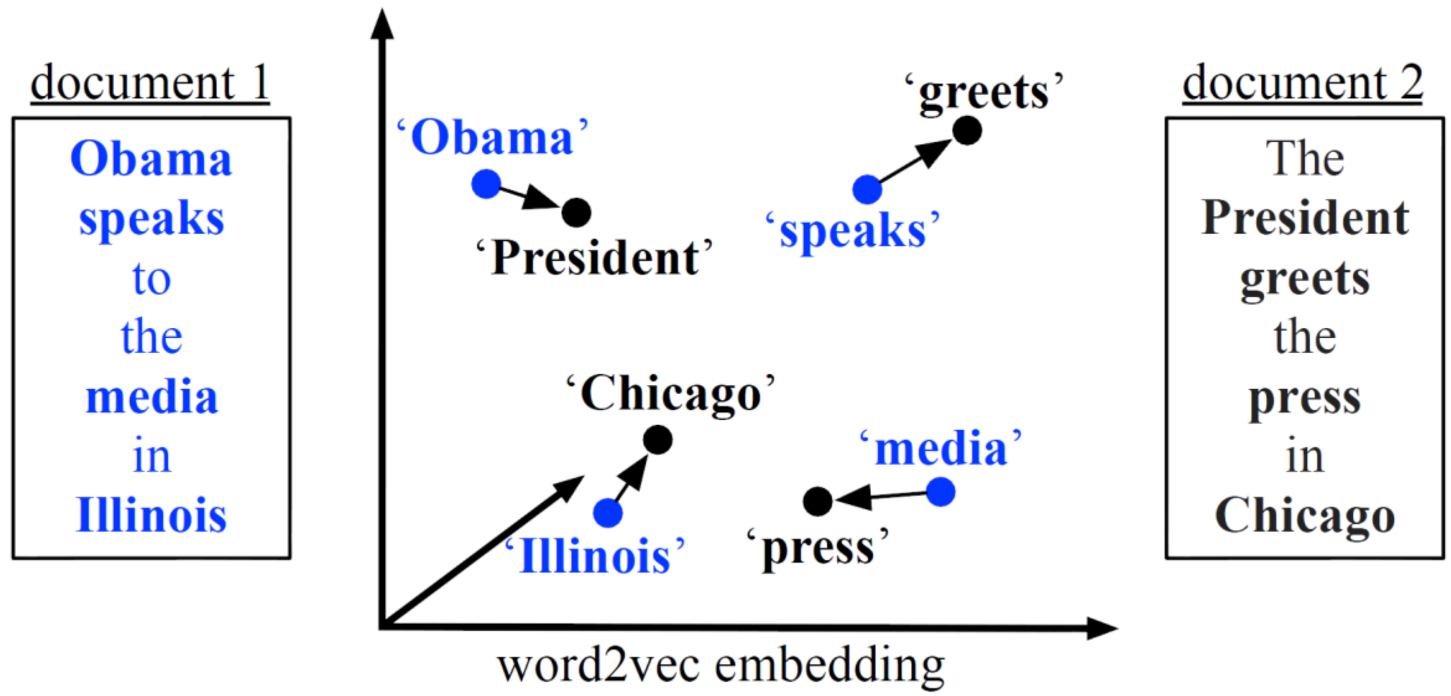
- Edit distance of 2 documents
- Based on word embedding representations
- Incorporate semantic similarity between individual word pairs into the document distance metric
- Based on "travel cost" between two words
- Calculates the cost of moving d to d'
- hyper-parameter free
- highly interpretable
- high retrieval accuracy

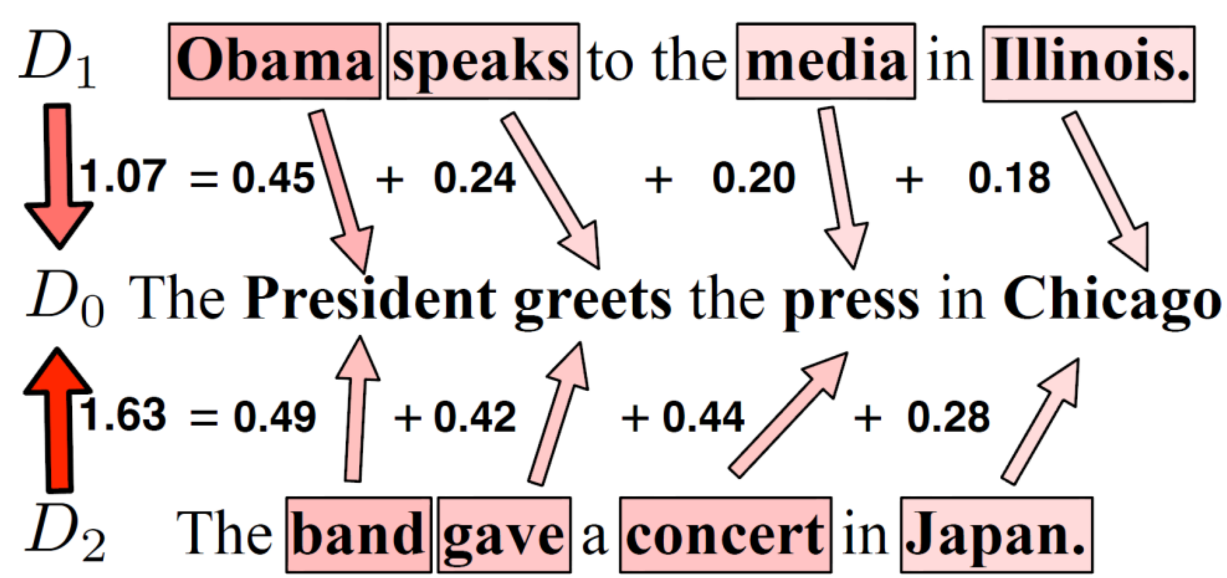


"minimum cumulative distance that all words in document 1 need to travel to exactly match document 2"

Word Mover's distance example

With the BOW representation D_1 and D_2 are at equal distance from D_0. Word embeddings allow to capture the fact that D_1 is closer.





[Kusner, M. J., Sun, E. Y., Kolkin, E. N. I., & EDU, W. From Word Embeddings To Document Distances. Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.]

CNN for document classification

- Use the high quality embeddings as input for Convolutional Neural Network
- Input must be fixed size
- max-pooling deals with variable document lengths
- Applies multiple filters to concatenated word vectors

$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$$

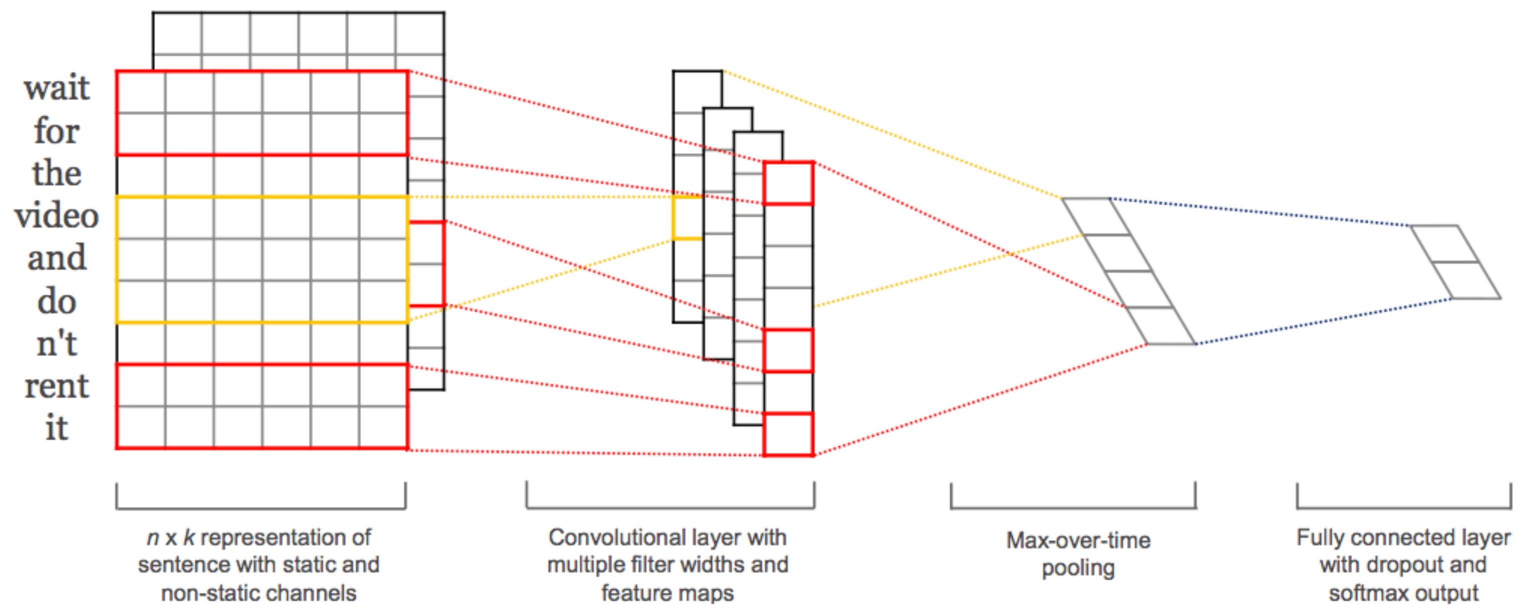
- Produces new features for every filter

$$c_i = f(\mathbf{w} \cdot \mathbf{x}_{i:i+h-1} + b)$$

- And picks the max as a feature for the CNN

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}]$$

$$\hat{c} = \max\{\mathbf{c}\}$$



Yoon Kim - Convolutional Neural Networks for Sentence Classification

CNN for document classification

- Many variations of the model
- use existing vectors as input (CNN-static)
- learn vectors for the specific classification task through backpropagation (CNN-rand)
- Modify existing vectors for the specific task through backpropagation(CNN-non-static)
- Combine multiple word embeddings
    - Each set of vectors is treated as a ‘channel’
    - Filter is applied to both channels
    - Gradients are backpropagated only through one of the channels
    - Fine-tunes one set of vectors while keeping the other static

CNN for document classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

References

- Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. The Journal of Machine Learning Research, 3, 1137–1155. http://doi.org/10.1162/153244303322533223
- Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Proceedings of the International Conference on Learning Representations (ICLR 2013), 1–12.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS, 1–9.
- Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing. Proceedings of the 25th International Conference on Machine Learning - ICML '08, 20(1), 160–167. http://doi.org/10.1145/1390156.1390177
- Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). Character-Aware Neural Language Models. AAAI. Retrieved from http://arxiv.org/abs/1508.06615
- Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., & Wu, Y. (2016). Exploring the Limits of Language Modeling. Retrieved from http://arxiv.org/abs/1602.02410
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. Journal of Machine Learning Research, 12 (Aug), 2493–2537. Retrieved from http://arxiv.org/abs/1103.0398
- Chen, W., Grangier, D., & Auli, M. (2015). Strategies for Training Large Vocabulary Neural Language Models, 12. Retrieved from http://arxiv.org/abs/1512.04906

More References

- Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings. Transactions of the Association for Computational Linguistics, 3, 211–225. Retrieved from https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/570
- Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543. http://doi.org/10.3115/v1/D14-1162
- Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. ACL, 238–247. http://doi.org/10.3115/v1/P14-1023
- Levy, O., & Goldberg, Y. (2014). Neural Word Embedding as Implicit Matrix Factorization. Advances in Neural Information Processing Systems (NIPS), 2177–2185. Retrieved from http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization
- Hamilton, W. L., Clark, K., Leskovec, J., & Jurafsky, D. (2016). Inducing Domain-Specific Sentiment Lexicons from Unlabeled Corpora. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Retrieved from http://arxiv.org/abs/1606.02820
- Hamilton, W. L., Leskovec, J., & Jurafsky, D. (2016). Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. arXiv Preprint arXiv:1605.09096.

References- blogs

- Sebastian Ruder blog series on Word Embeddings, http://sebastianruder.com/
- Andy Jones blog on word2vec, http://andyljones.tumblr.com/post/111299309808/why-word2vec-works
- Arora et al, https://arxiv.org/pdf/1502.03520v7.pdf
- Piotr Migdał , http://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html