Deep Learning Applications to Text Mining/NLP

M. Vazirgiannis

Data Science and Mining Team (DASCIM), LIX
École Polytechnique
http://www.lix.polytechnique.fr/dascim
Google Scholar: https://bit.ly/2rwmvQU
Twitter: @mvazirg

November, 2024

Outline

Text Representation for Information Retrieval

Text 1: “This is the text mining course lecture”
Text 2: “This is an introductory text course”

- Boolean model

	This	is	the	text	mining	course	Lecture	an	introductory
Text 1:	1	1	1	1	1	1	1	0	0
Text 2:	1	1		1	0	1	0	1	1

- Vector space model

	This	is	the	text	mining	course	Lecture	an	introductory
Text 1:	1/7	1/7	1/7	1/7	1/7	1/7	1/7	0	0
Text 2:	1/6	1/6	0	1/6	1/6	1/6	0	1/6	1/6

Bag of words models and td/idf

- Assume a query $q = \{w_i\}$ and a document $d = \{w_j\}$ in a corpus of N documents.
- We aim to find the similarity $sim(q, d)$ - serves as the score for documents ranking (i.e. as in web search)

$$sim(q,d) = \sum_{w_i \in q \cap d} (1 + log(f(w_i, d))) * log(\frac{N}{df(w_i)}) = \sum_{w_i \in q \cap d} tf(w_i, d) * idf(w_i) \tag{1}$$

- where: $tf(w_i, d)$: frequency of term w_i in document d and $df(t_i)$: number of documents containing w_i.

Language model

- Goal: determine $P(s = w_1 \dots w_k)$ in some domain of interest

$$P(s) = \prod_{i=1}^{k} P(w_i | w_1 \dots w_{i-1})$$

e.g. $P(w_1 w_2 w_3) = P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2)$

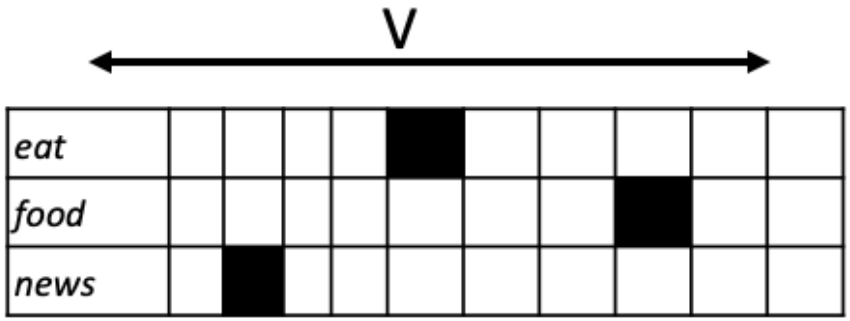
- Traditional n-gram language model assumption:"the probability of a word depends only on context of n-1 previous words"

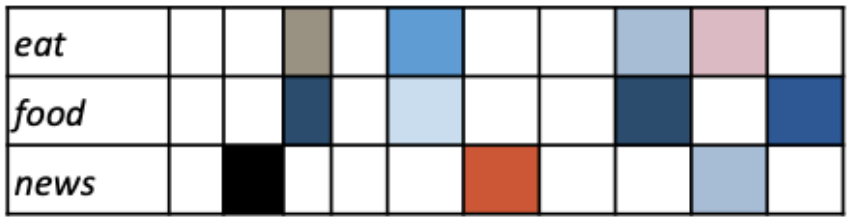
$$\Rightarrow \hat{P}(s) = \prod_{i=1}^{k} P(w_i | w_{i-n+1} \dots w_{i-1})$$

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

Word representations

- One hot encoding
  - sparse representation
  - problematic similarity computation
- 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





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

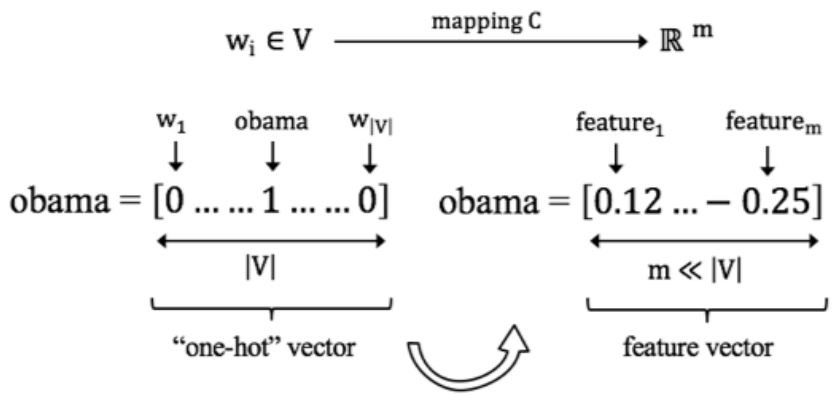
One hot

$$\left.\begin{array}{r} \text{obama} = [0\,0\,0\,0\ \ldots\ 0\,1\,0\,0] \\ \text{president} = [0\,0\,0\,1\ \ldots\ 0\,0\,0\,0] \end{array}\right\} \overrightarrow{\text{obama}}.\overrightarrow{\text{president}} = \vec{0}$$

$$\left.\begin{array}{r} \text{speaks} = [0\,0\,1\,0\ \ldots\ 0\,0\,0\,0] \\ \text{addresses} = [0\,0\,0\,0\ \ldots\ 0\,0\,1\,0] \end{array}\right\} \overrightarrow{\text{speaks}}.\overrightarrow{\text{addresses}} = \vec{0}$$

$$\left.\begin{array}{r} \text{illinois} = [1\,0\,0\,0\ \ldots\ 0\,0\,0\,0] \\ \text{chicago} = [0\,1\,0\,0\ \ldots\ 0\,0\,0\,0] \end{array}\right\} \overrightarrow{\text{illinois}}.\overrightarrow{\text{chicago}} = \vec{0}$$

Word Embedding



Word embeddings

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

SVD word embeddings

- Dimensionality reduction on co-occurrence matrix
- Create a $|V|x|V|$ word co-occurrence matrix X
- Apply SVD: $X = ULV^T$
- Chose top-k columns of U (how to choose k?)
- Use the k-dimensional vectors as representations for each word
- Able to capture semantic and syntactic similarity

Latent Semantic Indexing (LSI)

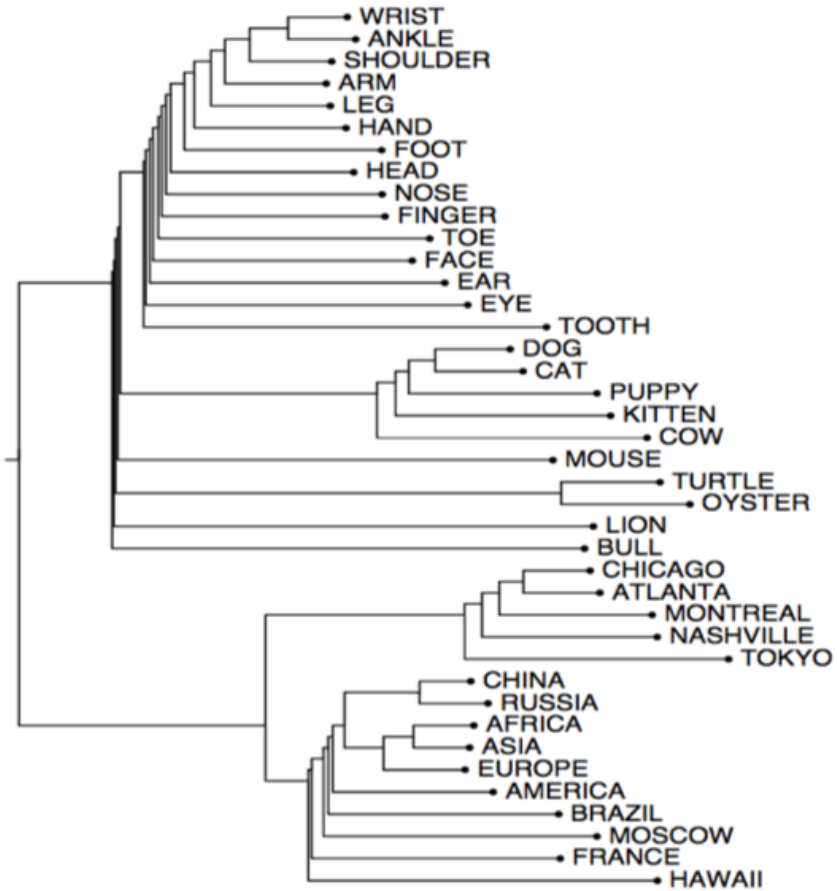
- The initial matrix is SVD decomposed as: $A = ULV^T$
- Choosing the top-k singular values from L we have: $A_k = U_k L_k V_k^T$
- L_k: square $k \times k$ - top-k singular values of the diagonal in matrix L
- U_k: $m \times k$ matrix - first k columns in U (left singular vectors)
- V_k^T: $k \times n$ matrix - first k lines of V^T (right singular vectors)

Typical values for $k \sim 200 - 300$ (empirically chosen based on experiments appearing in the bibliography)

LSI capabilities

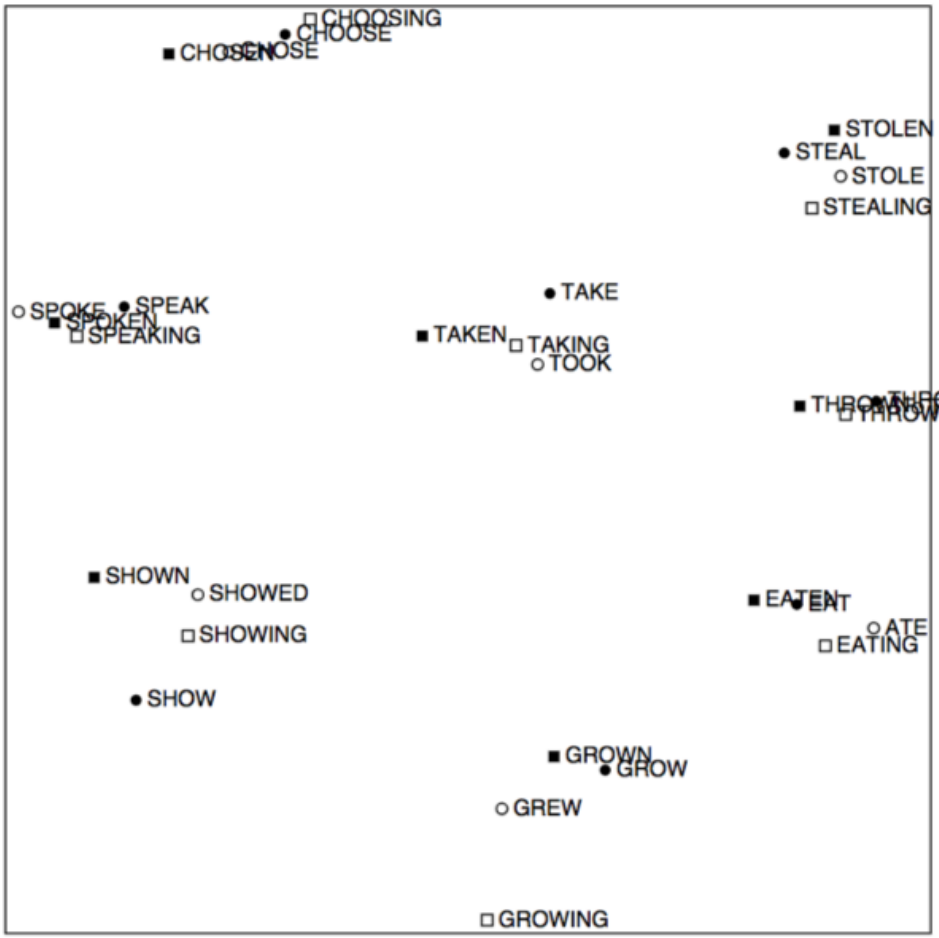
- Term to term similarity: $A_k A_k^T = U_k L_k^2 U_k^T$, where $A_k = U_k L_k V_k^T$
- Document-document similarity: $A_k^T A_k = V_k L_k^2 V_k^T$
- Term document similarity (as an element of the transformed – document matrix)
- Extended query capabilities transforming initial query q to q_n : $q_n = q^T U_k L_k^{-1}$
- Thus q_n can be regarded a line in matrix V_k

Patterns in SVD Space



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Patterns in SVD Space



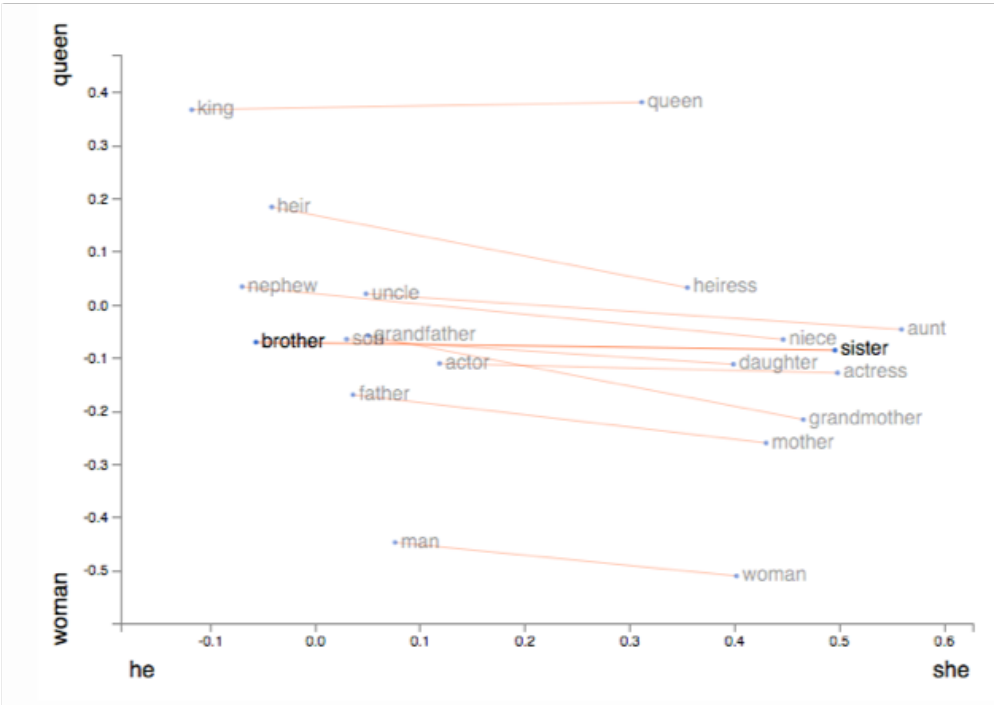
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

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 laying close to each other
- Words sharing similar contexts may be analogous
  - Synonyms
  - Antonyms
  - Names
  - Colors
  - Places
  - Interchangeable words
- Vector computations 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 PMI approximated by low rank approximation of the co-occurrence matrix.

1. $PMI(w', w) \approx v_w v_{w'}$ *inner product of the vector representations*
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. $argmin_w E_{w'}[(PMI(w'|w) - PMI(w'|queen) - PMI(w'|man) - PMI(w'|woman)]^2$
5. $argmin_w ||(v_w - v_{queen}) - (v_{man} - v_{woman})||^2$
6. $v_w \approx v_{queen} - v_{woman} + v_{man}$ which is an analogy!

- Arora et al (ACL 2016) shows that if (2) holds then (1) holds as well
- So we need to construct vectors from co-occurrence that satisfy (2)
- $d \ll |V|$ in order to have isotropic vectors

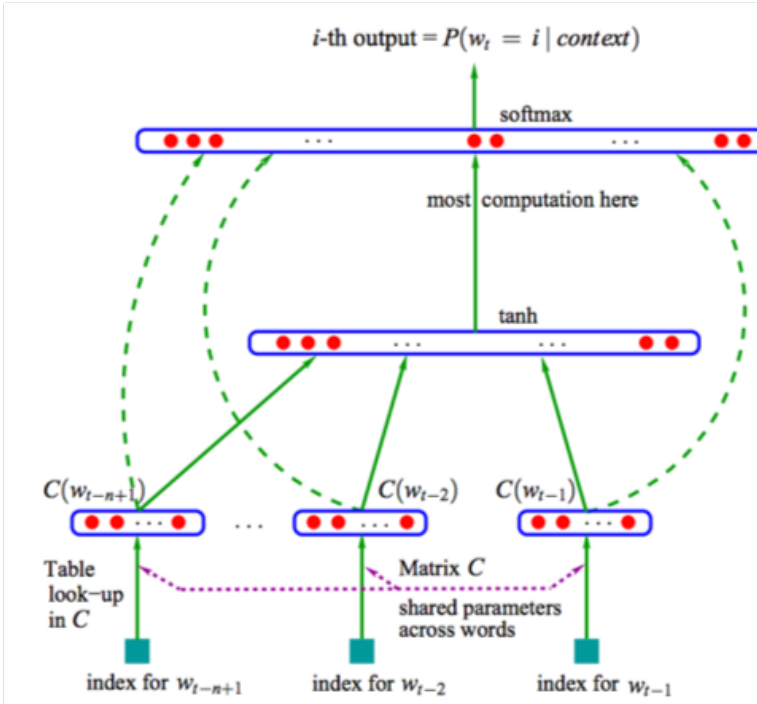
A Latent Variable Model Approach to PMI-based Word Embeddings, Arora et al (ACL 2016)

Learning Word Vectors

1. associate with each word in the vocabulary distributed *word feature* vector (a real-valued vector in $\mathbb{R}^m$)
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneourly the *word feature vecotrs* and the parameters of that *probability function*

- Corpus containing T training words
- Objective :
  $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t | w_{t-n+1}, \ldots, w_{t-1})$
- Decomposed in two parts: $w_i \in V \xrightarrow{\text{mapping C}} \mathbb{R}^m$
  - Mapping **C** (1-hot v $\Rightarrow$ lower dimensions)
  - Mapping any **g** s.t. estimate prob $t+1|t$ previous)

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

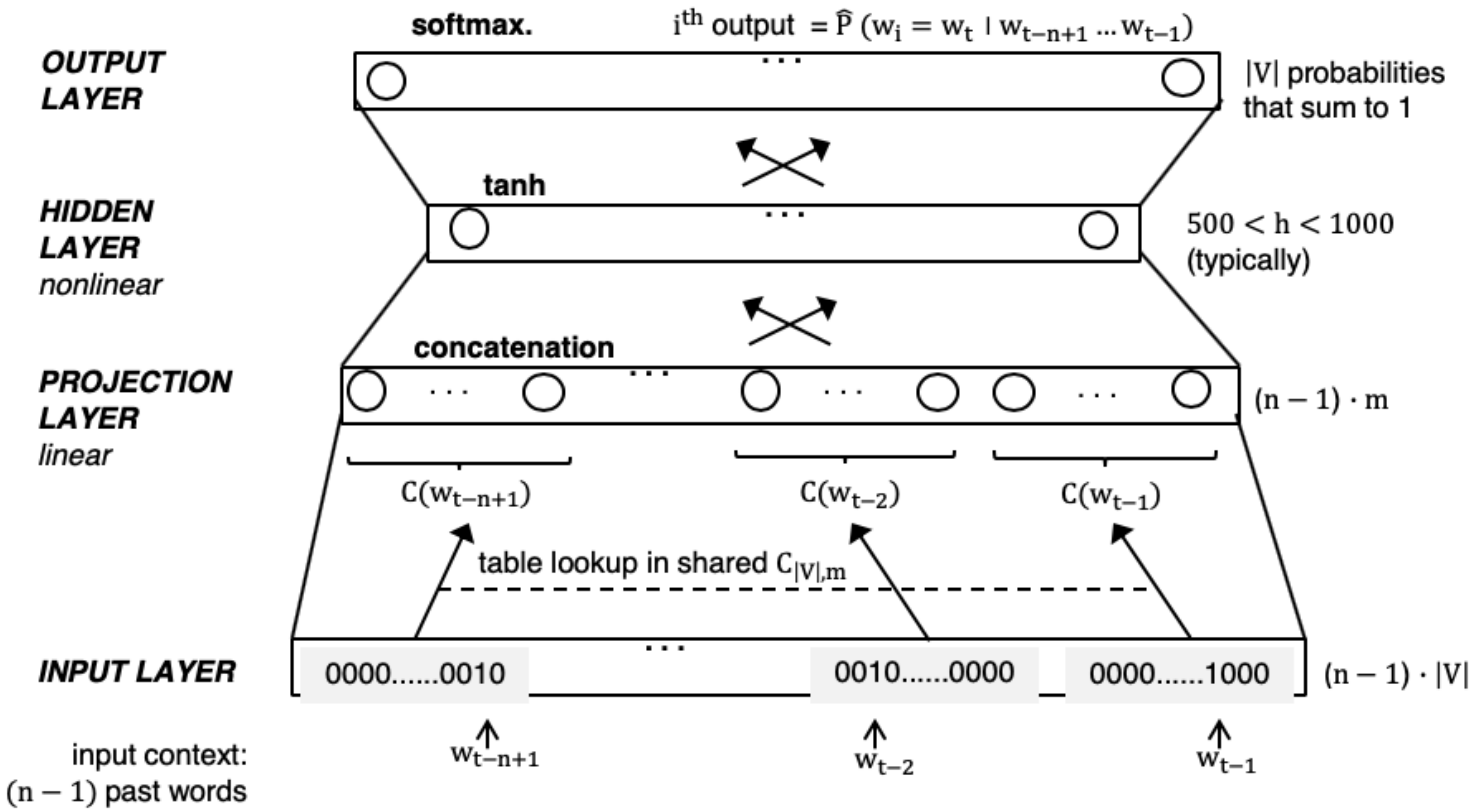


Bengio, Yoshua, et al. "A neural probabilistic language model." The Journal of Machine Learning Research 3 (2003): 1137-1155.

Neural Net Language Model

For each training sequence:

- input: (context, target)
- pair: $(w_{t-n+1}, \ldots, w_{t-1}, w_t)$
- objective: minimize $E = -log\hat{P}(w_t|w_{t-n+1}, \ldots, w_{t-1})$



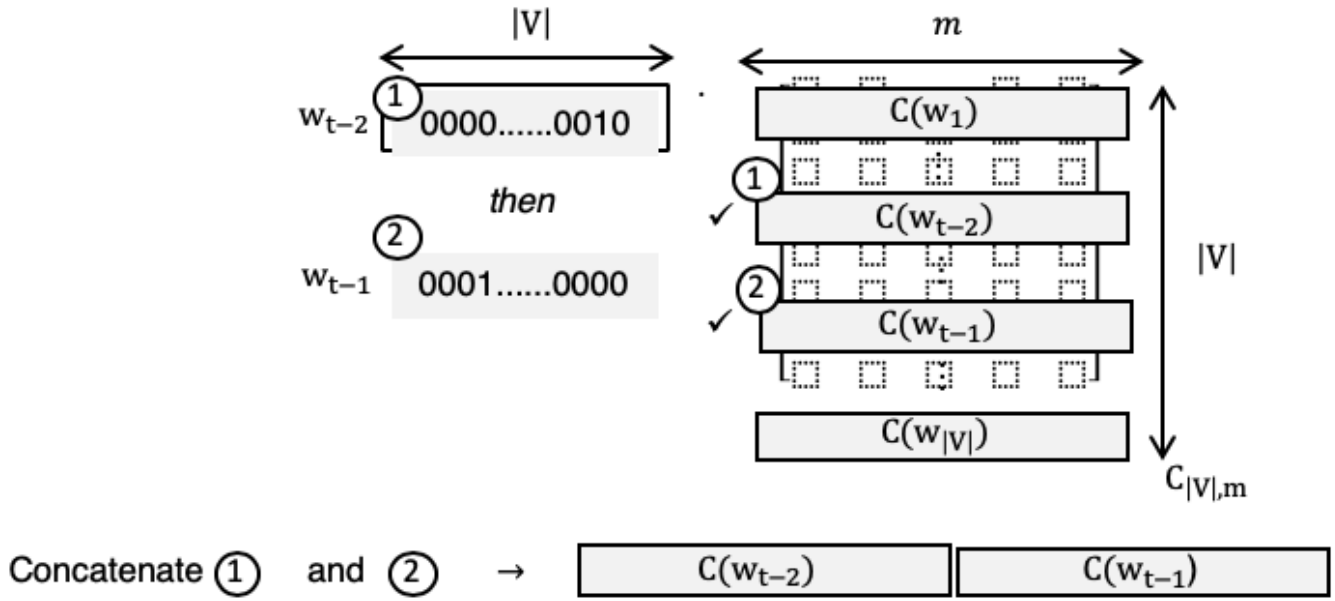
Objective function

- $E = -log\hat{P}(w_t|w_{t-n+1}, \ldots, w_{t-1})$
- $\hat{P}$: probability between 0 and 1.
- log of $\hat{P}$ is negative $\Rightarrow$ –log $\hat{P}$ term positive.
- makes sense to try to minimize E.
  - Probability $\hat{P}$ of word given the context be as high as possible (1 for a perfect prediction).
  - case the error is equal to 0 (global minimum).

p	log(p)	-log(p)
0.7	-0.154	0.154
0.2	-0.698	0.698

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**

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

NNLM hidden/output layers and training

- Softmax (log-linear classification model) outputs positive numbers that sum to one (a multinomial probability distribution): i^{th} unit in the output layer:
  $\hat{P}(w_i = w_t | w_{t-n+1}, \ldots, w_{t-1}) = \frac{e^{yw_i}}{\sum_{i'=1}^{|V|} e^{yw_{i'}}}$

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

  $\boldsymbol{\theta = (b, d, U, H, C)}$
  complexity per training sequence: $n * m + n * m * h + \boldsymbol{h * |V|}$
  computational bottleneck: **nonlinear hidden layer** ($h * |V|$ term)

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

$$\theta \leftarrow \theta + \epsilon \frac{\partial E}{\partial \theta} = \theta + \epsilon \frac{\partial log \hat{P}(w_t | w_{t-n+1}, \ldots, w_{t-1})}{\partial \theta}$$

  (weights are initialized randomly, then updated via backpropagation)

NNLM facts

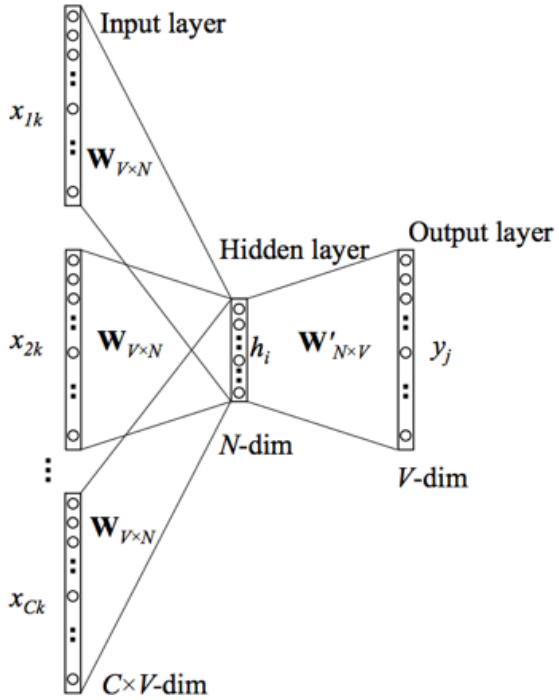
- tested on Brown (1.2M words, $|V| \cong 16K$) and AP News (14M words, $|V| \cong 150K$ reduced to 18K) corpuses
- h: # hidden units, n= # words in context, m: dimensionality of word vectors.
- Brown: h=100, n=5, m=30
- AP News: h=60, n=6, m=100, **3 week** training using **40 cores**
- 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs
- Due to **complexity**, NNLM can't be applied to large data sets $\rightarrow$ poor performance on rare words
- Bengio et al. (2003) claims 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 1000 $\times$ 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
- uses the surrounding words to predict the center word
- n-words before and after the target word

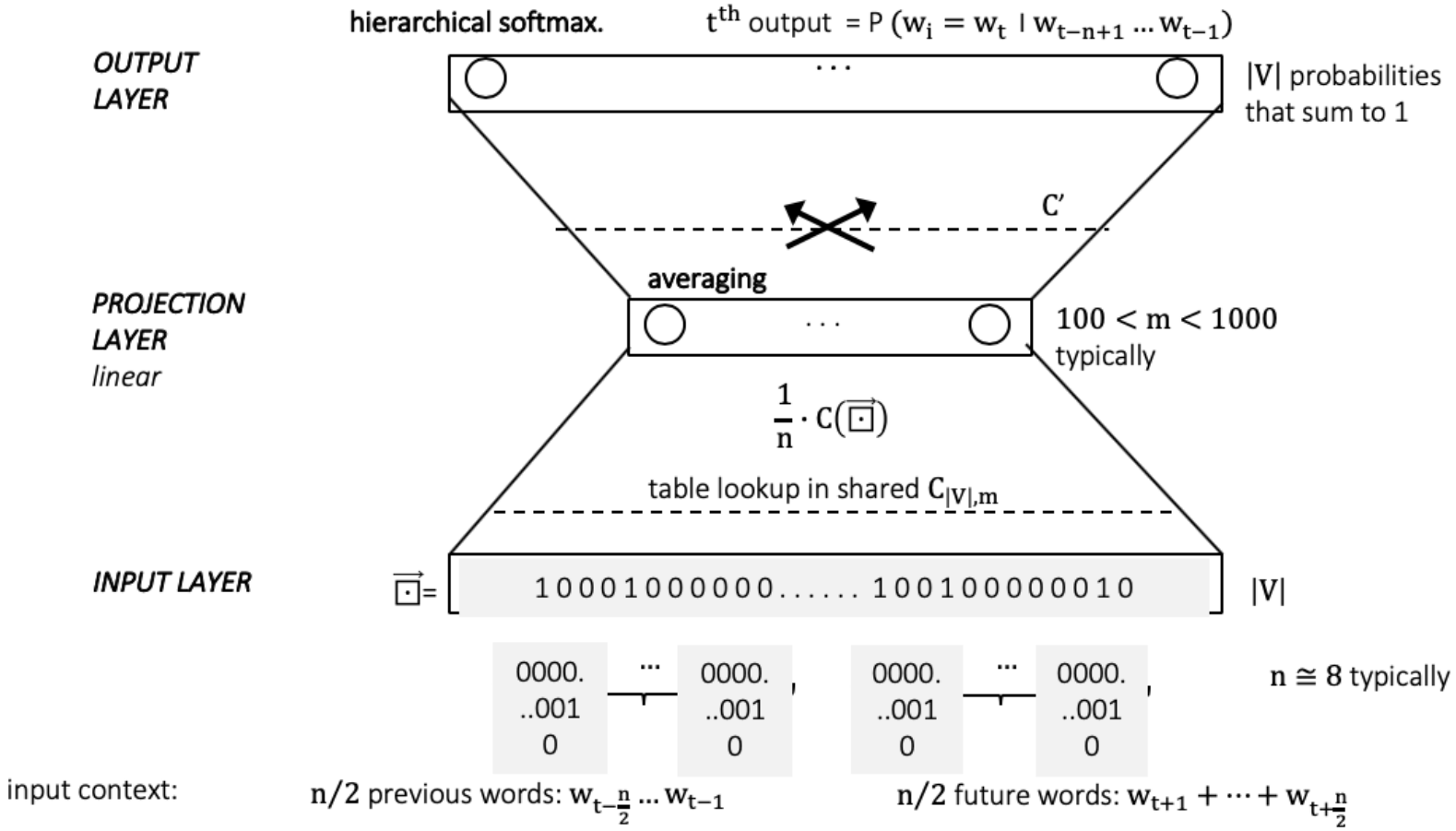


Efficient Estimation of Word Representations in Vector Space- Mikolov et al.

Continuous Bag-of-Words (CBOW)

For each training sequence:

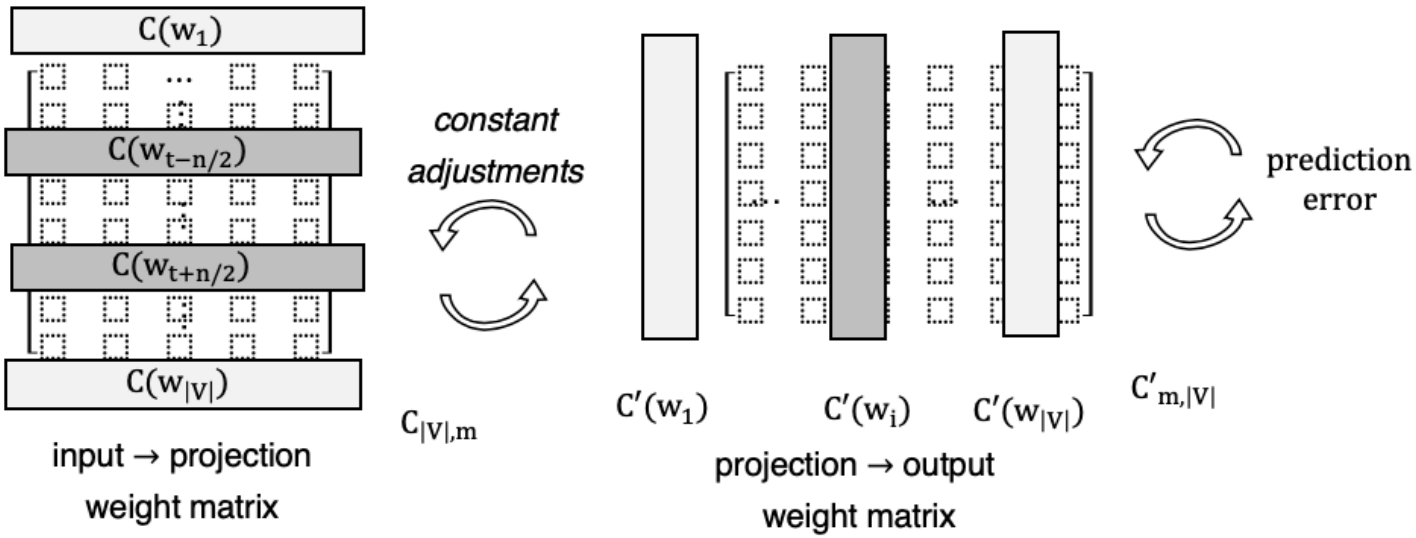
- input = (context, target)
- pair: $(w_{t-\frac{n}{2}}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+\frac{n}{2}}, w_t)$
- minimize $-log\hat{P}(w_t|w_{t-n+1}, \ldots, w_{t-1})$



Weight updating

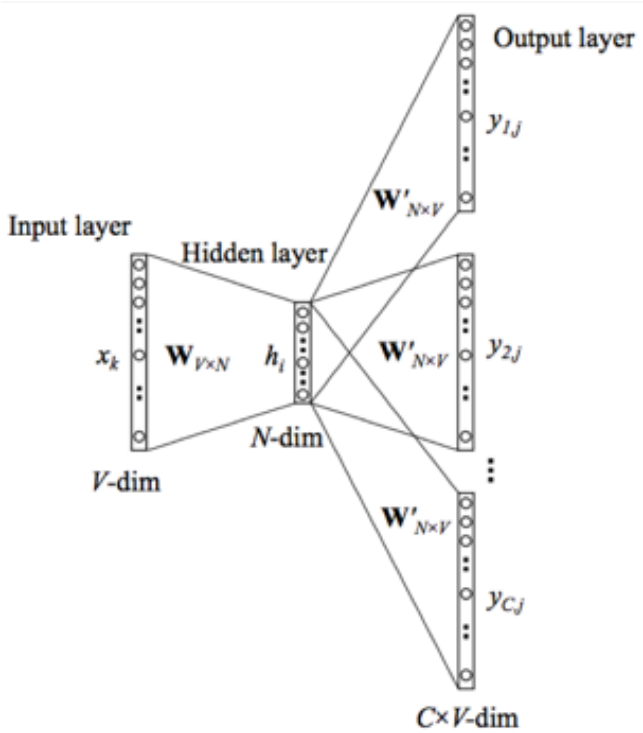
- For each (context, target=w_t) pair, only the word vectors from matrix C corresponding to the context words are updated
- Recall that we compute $P(w_i = w_t|context) \quad \forall w_i \in V$. We compare this distribution to the true probability distribution (1 for w_t, 0 elsewhere)
- **Back propagtion**
  - $P(w_i = w_t|context)$ 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 $P(w_i = w_t|context)$ **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 $\rightarrow$ clustering



Skip-gram

- skip-gram uses, in a context, 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
- $P(w_{t+j}|w_t) = \frac{\exp({v_{w_t}}^T v'_{w_{t+j}})}{\sum_{w \in V} \exp({v_{w_t}}^T v'_{w_{t+j}})}$
- ${v_{w_t}}^T$ is a column of $W_{V \times N}$ and $v'_{w_{t+j}}$ is a column of $W'_{N \times V}$
- objective function $J = \frac{1}{T} \sum_{t=1}^{T} \sum_{-n \leqslant j \leqslant n} \log p(w_{t+j}|w_t)$



Efficient Estimation of Word Representations in Vector Space- Mikolov et al. 2013

word2vec facts

- Complexity is $\boldsymbol{n * m + m * log|V|}$ (Mikolov et al. 2013a)
- n:size of the context window ($\sim$ 10), $\boldsymbol{n \times m}$: dimensions of the projection layer, $\boldsymbol{|V|}$: size of the vocabulary
- On Google news 6B words training corpus, with $|V| \sim 10^6$:
  - CBOW with m=1000 took **2 days** to train on **140 cores**
  - Skip-gram with m=1000 took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for m=100 only! (note that m=1000 was not reasonably feasible on such a large training set)
- word2vec training *speed* $\cong 100K - 5M$ words/s
- Quality of the word vectors:
  - $\uparrow$ 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: **un-ethical**

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

---

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

https://code.google.com/p/word2vec/

Glove

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

- Ratio of co-occurrence probabilities best distinguishes relevant words

  $F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \Rightarrow w_i^T \tilde{w}_k + b_i + \tilde{b_k} = log(X_{ik})$

- Cast this into a least square problem: $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - logX_{ij})^2$
- X: co-occurrence matrix
- f: weighting function $f(x) = \begin{cases} (x/x_{max})^{\alpha} & if \quad x < x_{max} \\ 1 & otherwise \end{cases}$
- b: bias terms
- w_i = word vector
- $\tilde{w}_j$ = context vector

model that utilizes

- count data
- bilinear prediction-based methods like word2vec

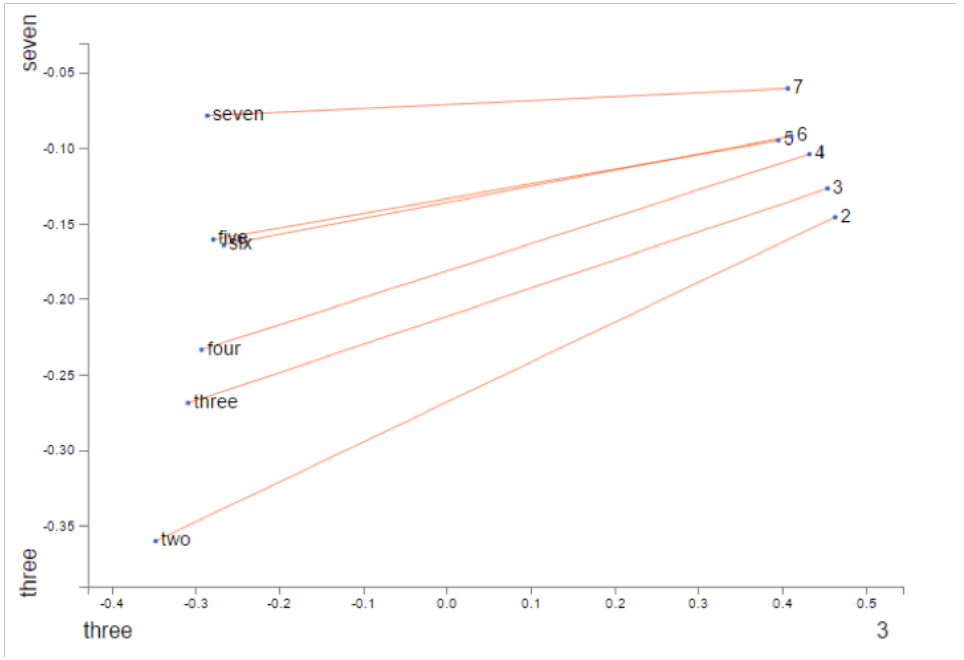
https://nlp.stanford.edu/pubs/glove.pdf

Which 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

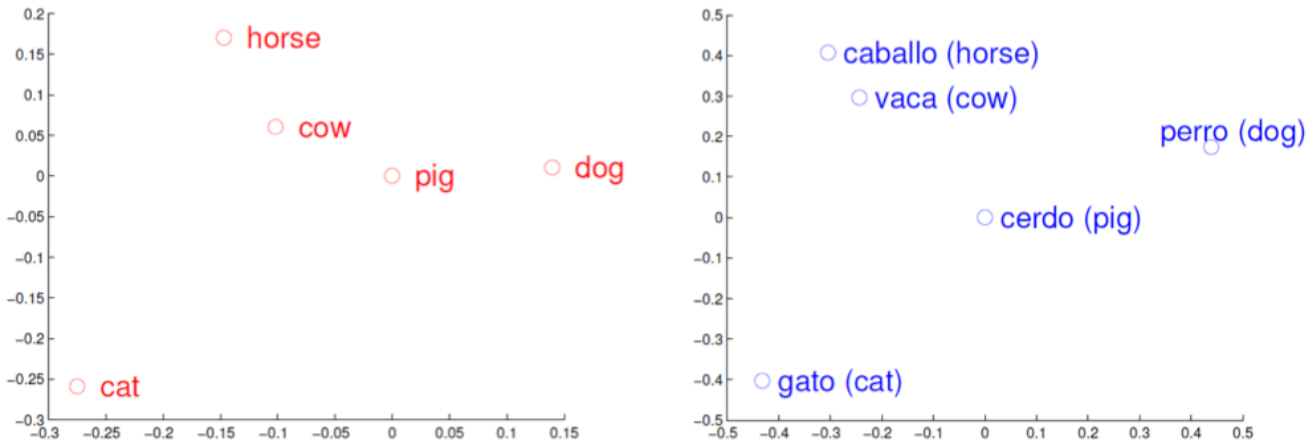
- Word analogies
- Find similar words
  - Semantic similarity
  - Syntactic similarity
- POS tagging
- Similar analogies for different languages
- Document classification



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

Applications

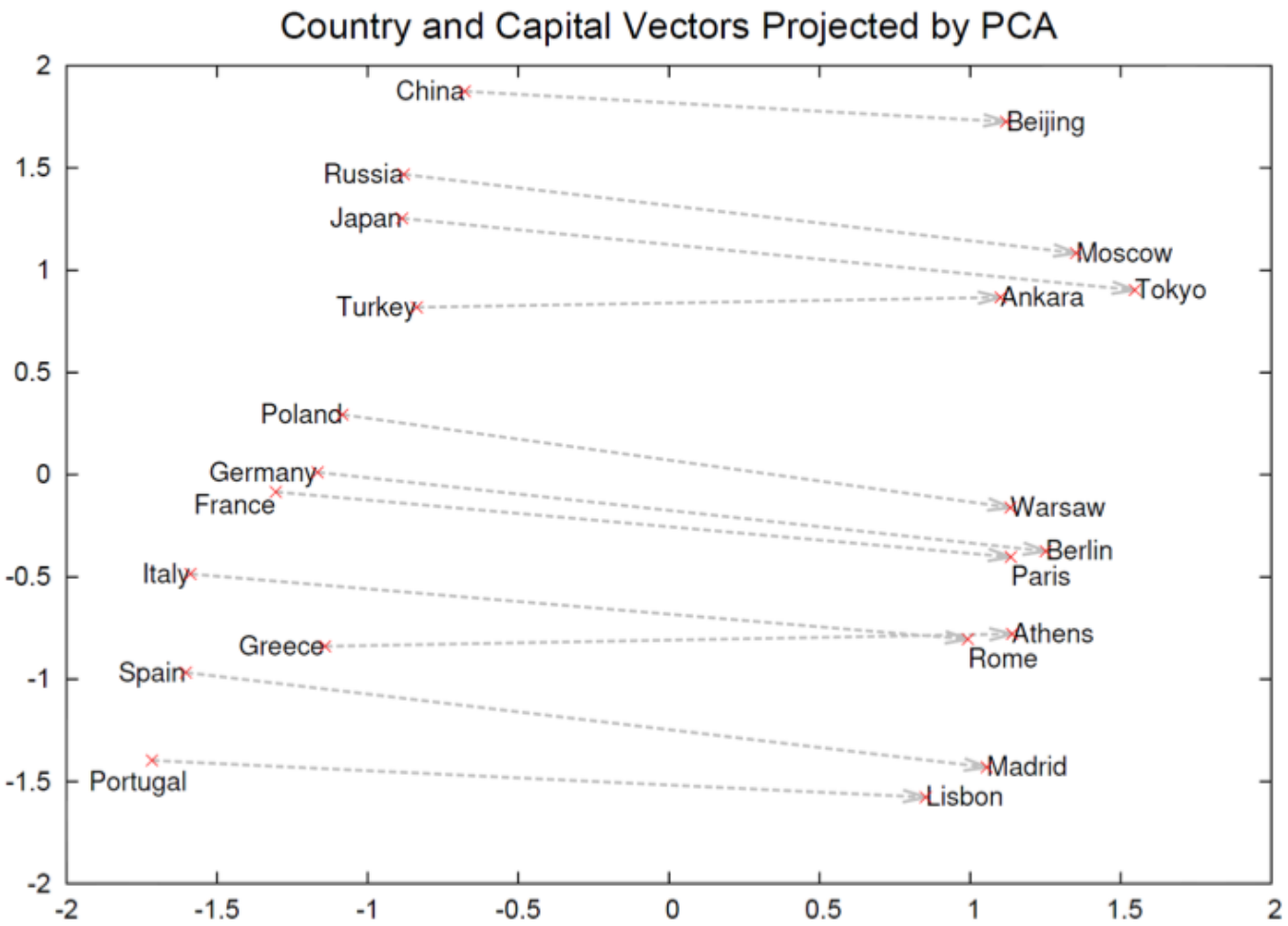
- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval...
- Example for English to Spanish machine translation:



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.

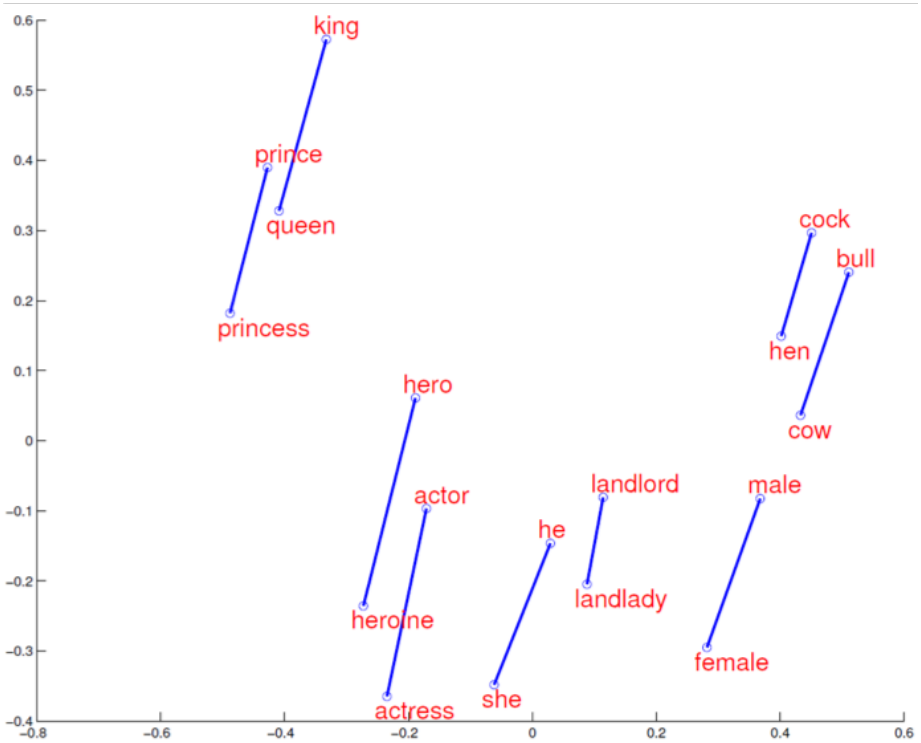
Remarkable properties of word vectors



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

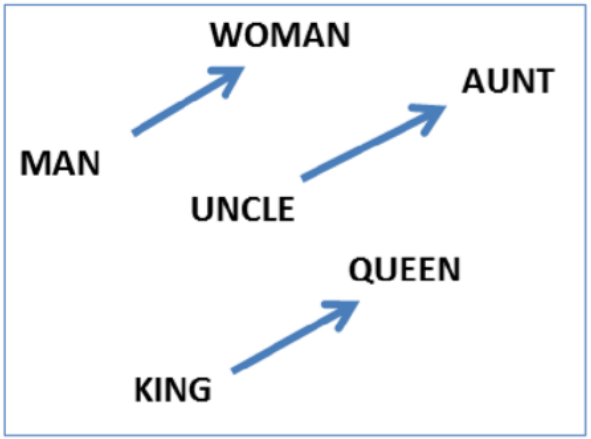
Remarkable properties of word vectors



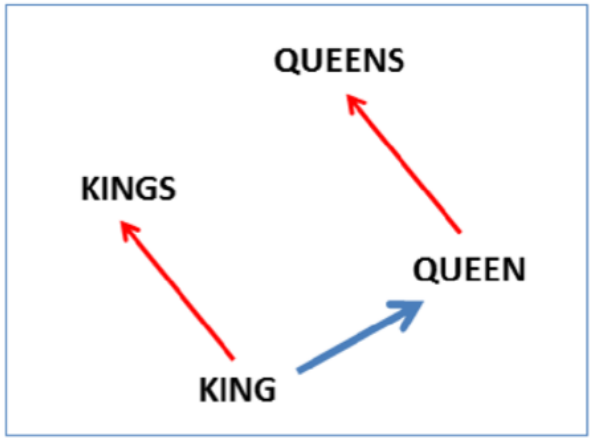
constant **female-male** difference vector

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

Remarkable properties of word vectors



constant **female-male** difference vector



constant **singular-plural** difference vector

- Vector operations are supported and make intuitive sense:

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

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

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

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

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

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

- Online demo (scroll down to end of tutorial)

---

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

Outline

CNN for text classification

- Use the high quality embeddings as input for Convolutional Neural Network
- Applies multiple filters to concatenated word vector

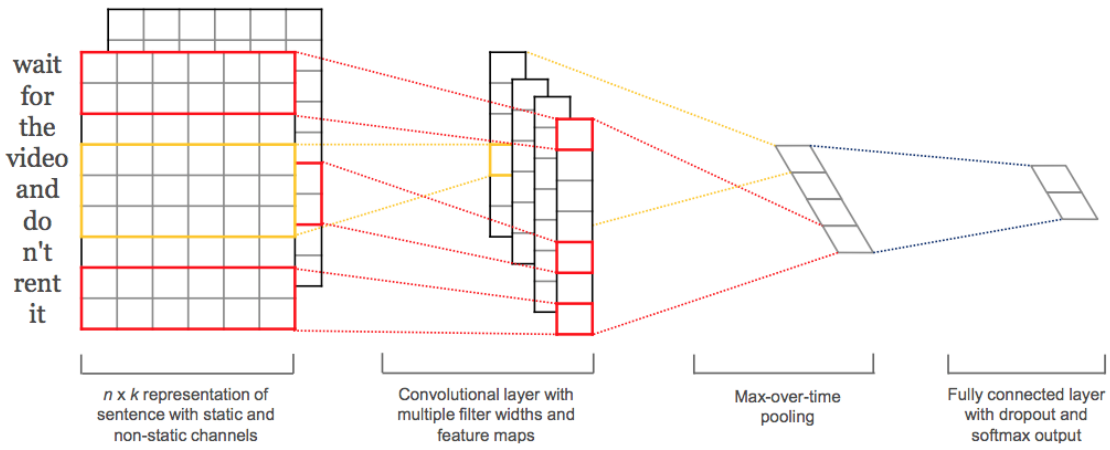
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \ldots \mathbf{x}_n$$

- 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

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



Yoon Kim - Convolutional Neural Networks for Sentence Classification

CNN for text classification

Many variations of the model [1]

- 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)

[1] Y. Kim, Convolutional Neural Networks for Sentence Classification, EMNLP 2014

CNN for text classification

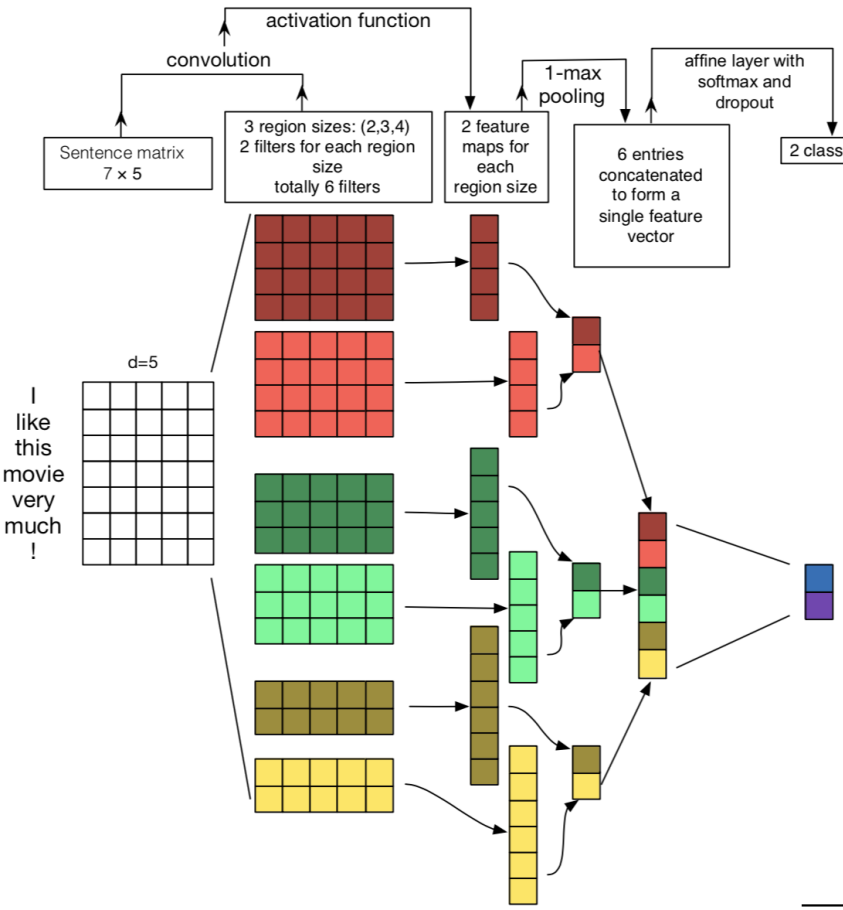
- Combine multiple word embeddings
- Each set of vectors is treated as a ‘channel’
- Filters applied to all channels
- Gradients are back-propagated only through one of the channels
- Fine-tunes one set of vectors while keeping the other static

CNN for text 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**	–	–

Accuracy scores (Kim et al vs others)

CNN for text classification



- Data (text) only 1^{st} column of input
- Rest of each row: embedding (in images 2D+RGB dimension)
- Filters of different sizes (4x5, 3x5 etc.)
  - Each size captures different features (need $\sim 10^2$ filters/size)
- Feature maps:
  - As many as the times filter fits on data matrix
- Max pooling maintains the "best features"
- Global feature map $\Rightarrow$ classification via softmax

---

[1] Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) conv classification." arXiv preprint arXiv:1510.03820 (2015).

CNN architecture for (short) document classification – T-SNE visualization

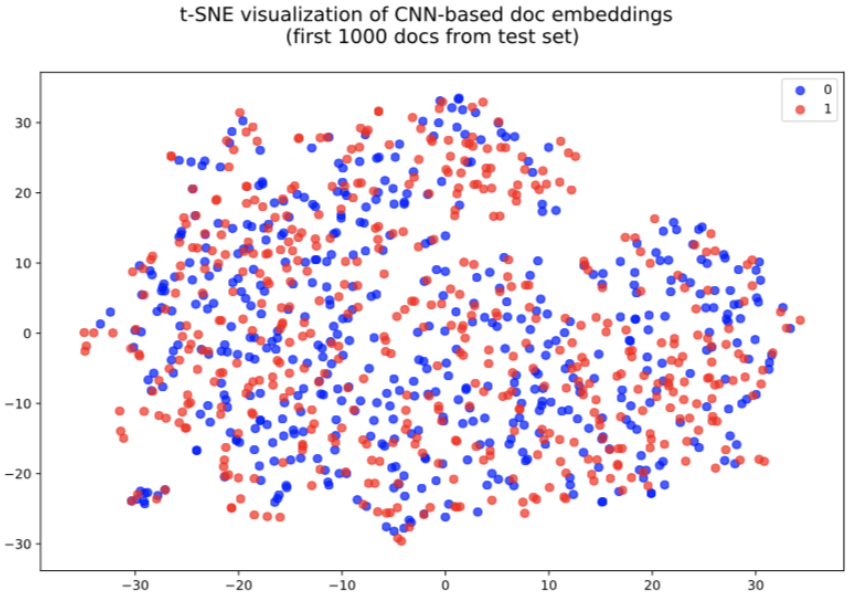


Figure 2: Doc embeddings before training.

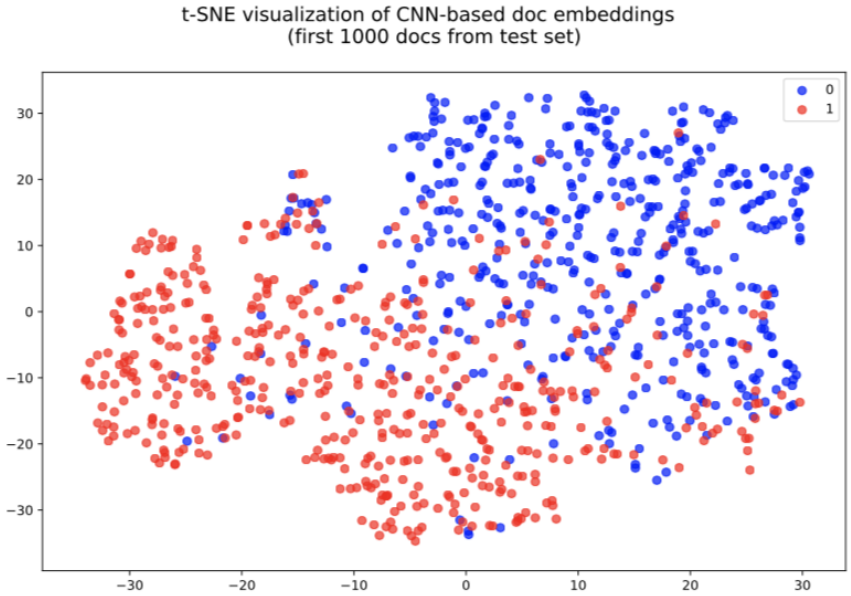


Figure 3: Doc embeddings after 2 epochs.

CNN architecture for (short) document classification - Saliency maps

- Words most related to changing the doc classification
- A in R^{sxd}, s:# sentence words, d:size of embeddings

$$saliency(a) = \left| \frac{\partial(CNN)}{\partial a} | a \right|$$

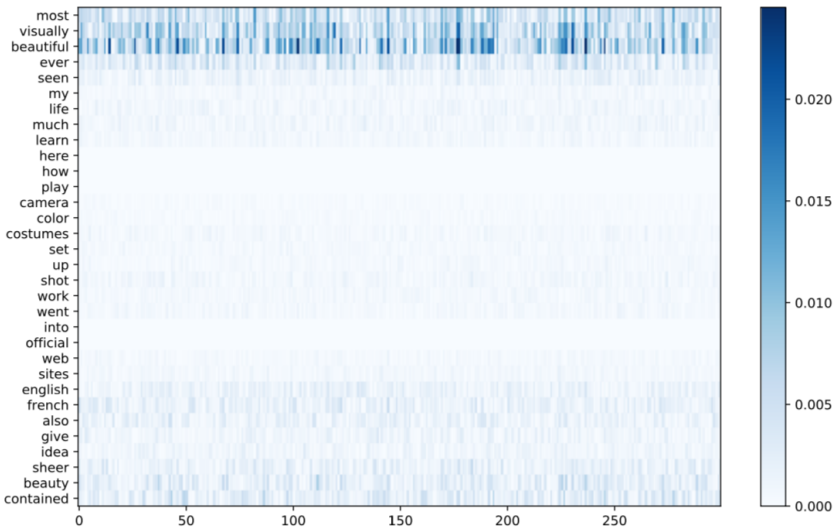


Figure 4: Saliency map for document 1 of the IMDB test set (true label: positive)

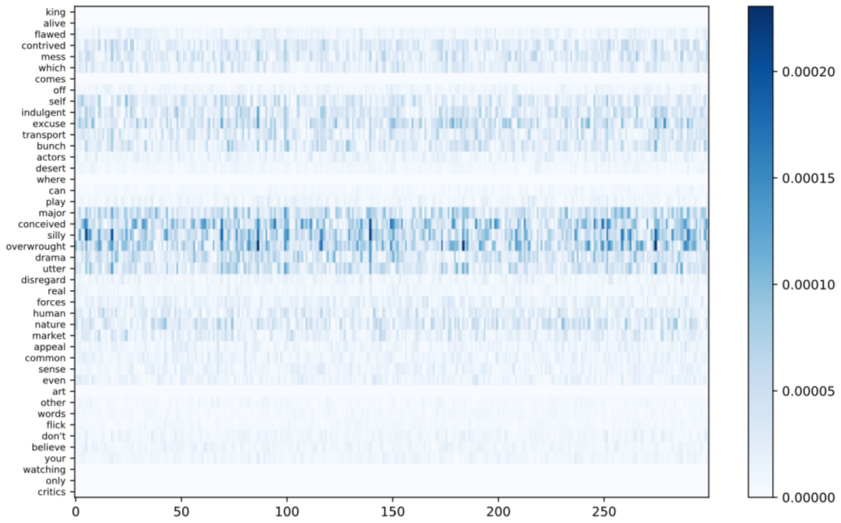
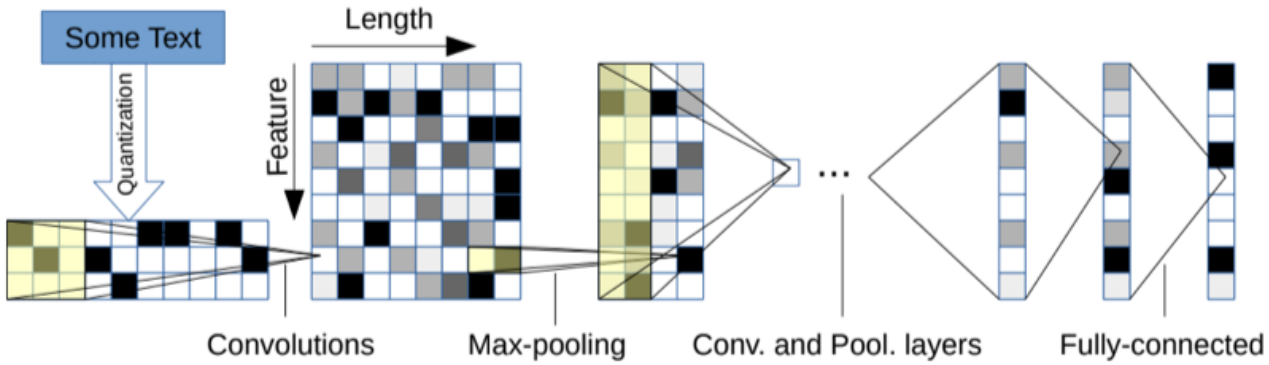


Figure 5: Saliency map for document 15 of the IMDB test set (true label: negative)

Character-level CNN for Text Classification

- Input: sequence of encoded characters
- quantize each character using “one-hot” encoding
- input feature length is 1014 characters
- 1014 characters able capture most of the texts of interest
- Also perform Data Augmentation using Thesaurus as preprocessing step

Model Architecture



- 9 layers deep
- 6 convolutional layers
- 3 fully-connected layers
- 2 dropout modules in between the fully-connected layers for regularization

Model Comparison

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg. w2v Conv.	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm. w2v Conv.	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg. w2v Conv. Th.	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm. w2v Conv. Th.	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg. Lk. Conv.	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm. Lk. Conv.	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg. Lk. Conv. Th.	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

Testing errors for all models: Blue→best, Red→worst

Some References

- Bengio, Yoshua, et al. ”A neural probabilistic language model.” The Journal of Machine Learning Research 3 (2003): 1137-1155.
- Rong, X. (2014). word2vec Parameter Learning Explained. arXiv preprint arXiv:1411.2738.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).
- Convolutional Neural Networks for Sentence Classification, Yoon Kim, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014
- Character-level Convolutional Networks for Text Classification, Xiang Zhang Junbo Zhao Yann LeCun, in proceedings of NIPS 2015.