Deep Learning - II

M. Vazirgiannis

LIX, Ecole Polytechnique

October 2018

- **MLP + CNN Applications**
  - Word embeddigs
- Recurrent NNs + LSTMs

Language model

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

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

e.g., $P\left(w_1 w_2 w_3\right) = P\left(w_1\right) P\left(w_2 \mid w_1\right) P\left(w_3 \mid w_1 w_2\right)$

- Traditional n-gram language model assumption:

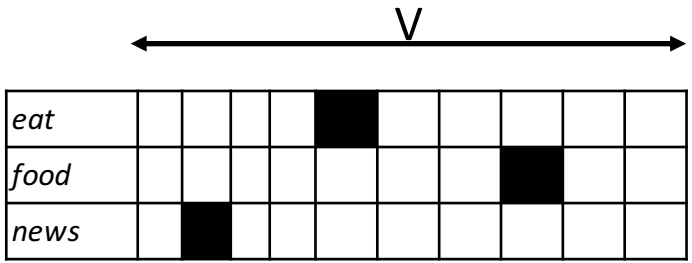
  "the probability of a word depends only on **context** of $n-1$ previous words"

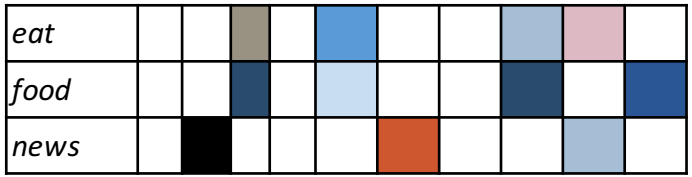
$$\Rightarrow \widehat{P}(s) = \prod_{i=1}^{k} P\left(w_i \mid w_{i-n+1} \dots w_{i-1}\right)$$

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

Representing Words

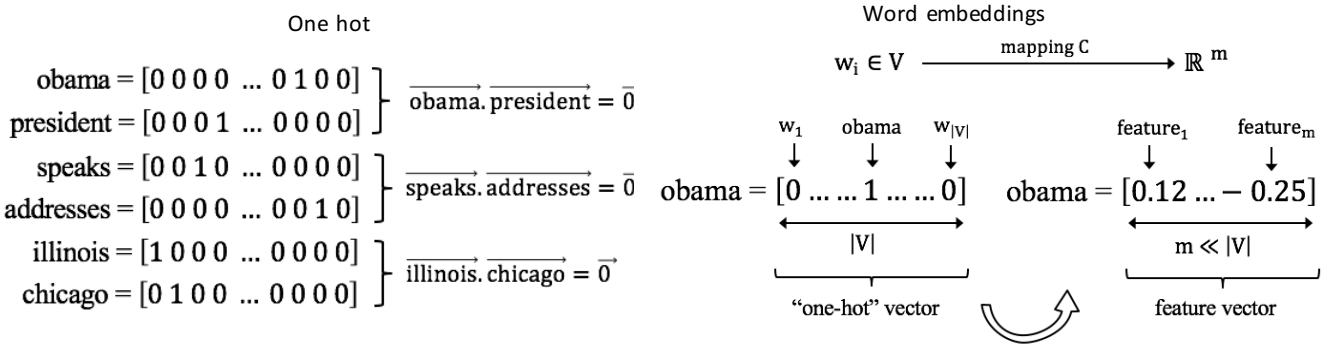
- **One-hot vector**
  - high dimensionality
  - sparse vectors
  - dimensions=|V| (10^6<|V|)
  - unable to capture semantic similarity between words
- **Distributional vector**
  - words that occur in similar contexts, tend to have similar meanings
  - each word vector contains the frequencies of all its neighbors
  - dimensions=|V|
  - computational complexity for ML algorithms





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



Co-occurrence matrix

- **I like to fly**
- **I enjoy learning to fly**
- **I like learning english**

	I	like	fly	enjoy	learning	english
I		1	1	1	1	1
like	1		1		1	1
fly	1			1		
enjoy	1		1		1	
learning	1	1	1			1
english	1	1			1	

- The matrix is large and sparse
  - Storage
  - Classification
- Learn a *low dimensional dense* representation of words

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**

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

Latent Semantic Indexing (LSI) - II

- 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 kxk - top-k singular values of the diagonal in matrix L,
- $U_{k,}$ mxk matrix - first k columns in U (left singular vectors)
- $V_k^{T,}$ kxn matrix - first k lines of V^T (right singular vectors)

Typical values for κ~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 $A_k = U_k L_k V_k^T$

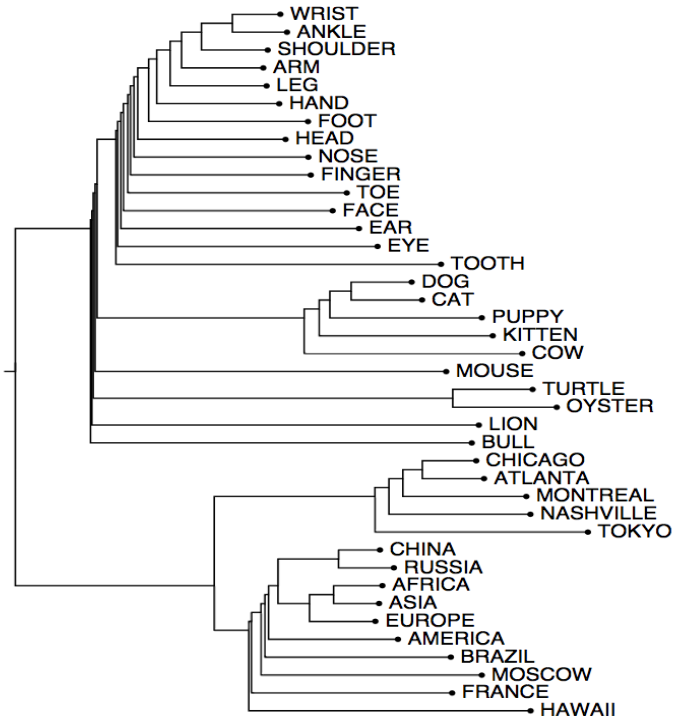
- Document-document similarity: $A_k^T A_k = V_k L_k^2 V_k^T$

- Term document similarity (as an element of the transformed – document matrix)

- Extended query capabilities transforming initial query q to q_n : $q_n = q^T U_k L_k^{-1}$

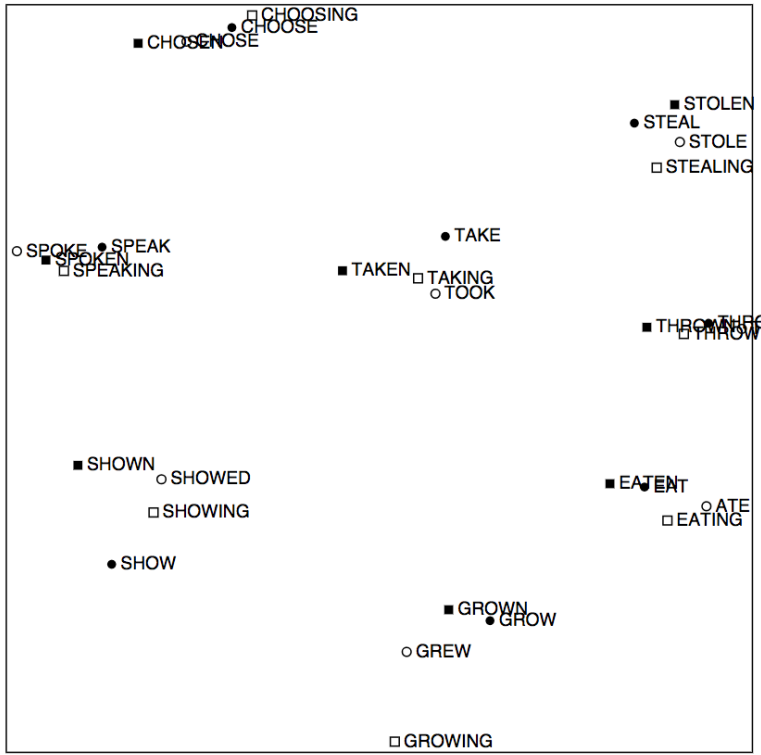
- Thus q_n can be regarded a line in matrix V_k

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

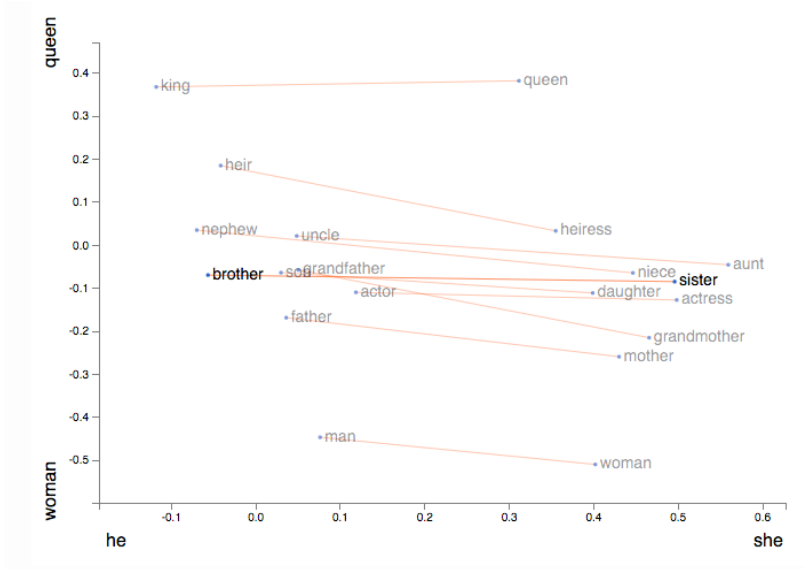
SVD problems

- 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. $argmin_w E_{w'}[\big(PMI(w'|w) - PMI(w'|queen)\big) - (PMI(w'|man) - PMI(w'|woman))]^2$

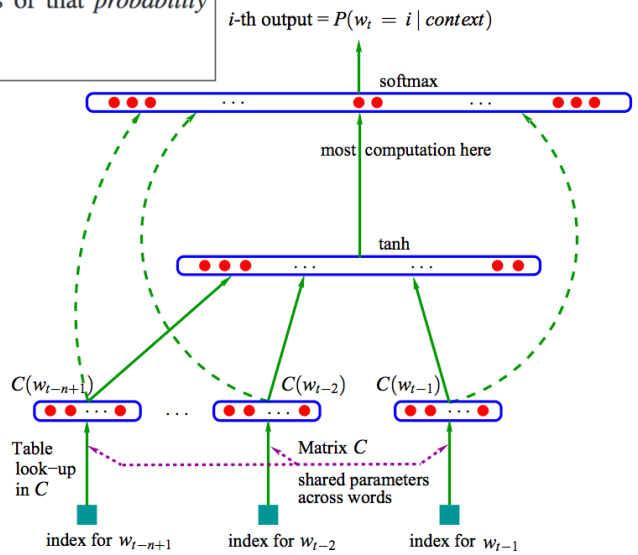
5. $argmin_w||\big(v_w - v_{queen}\big) - (v_{man} - v_{woman})||^2$
6. $v_w \approx v_{queen} - v_{woman} + v_{man}$ which is an analogy!

- *Arora et al (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<<|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 a 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 simultaneously the *word feature vectors* and the parameters of that *probability function*.



- Corpus containing T training words
- Objective: $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$
- 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)

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

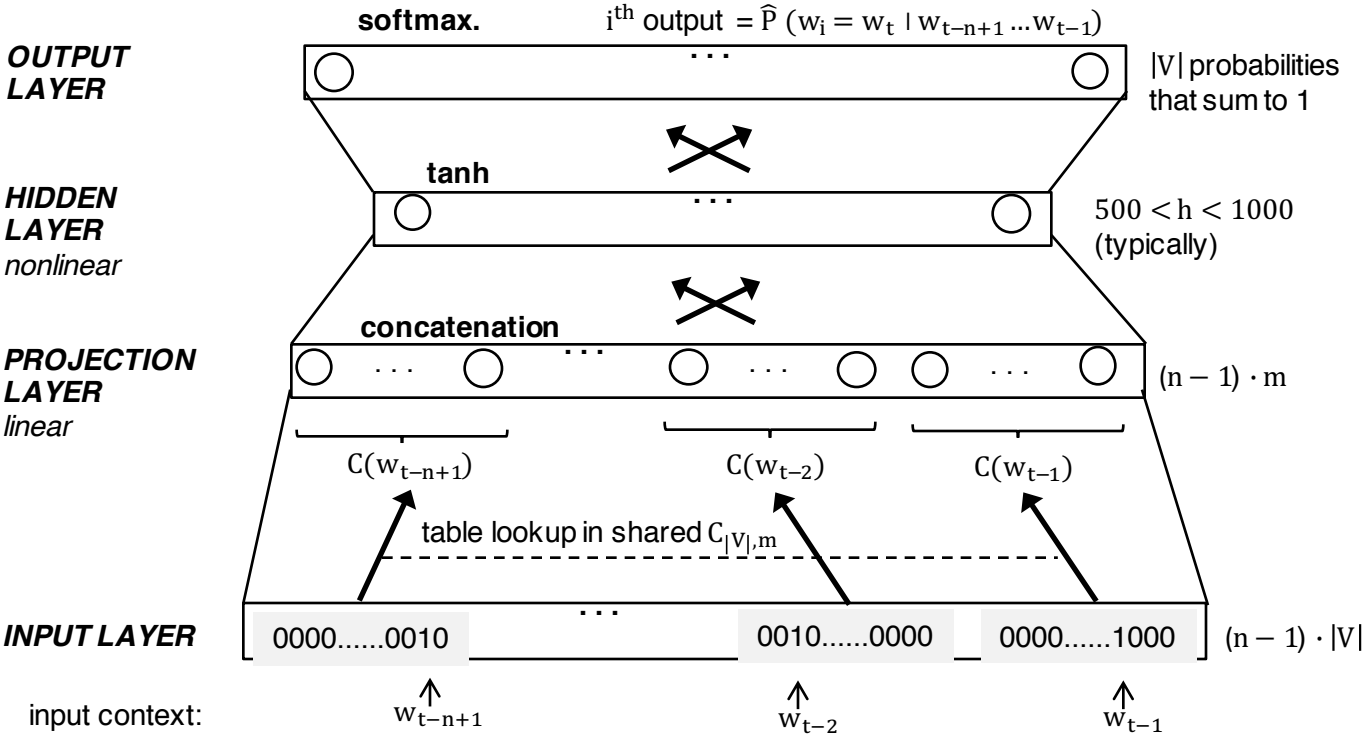
- C(i): i-th word feature vector (Word embedding)
- Objective function: $J = \frac{1}{T} \sum f(w_t, \ldots, w_{t-n+1})$

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 $E = -\log \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$



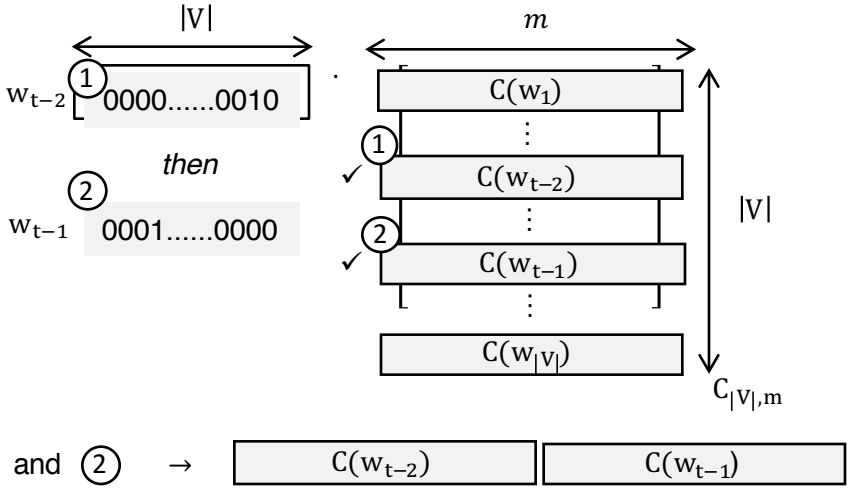
Objective function

- $E = -log\ \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$
- a probability between 0 and 1.
- On this support, the log is negative => –log term positive.
- makes sense to try to minimize it.
  - Probability 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	-0,15490196	0,15490196
0,2	-0,698970004	0,698970004

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) outputs positive numbers that sum to one (a multinomial probability distribution):

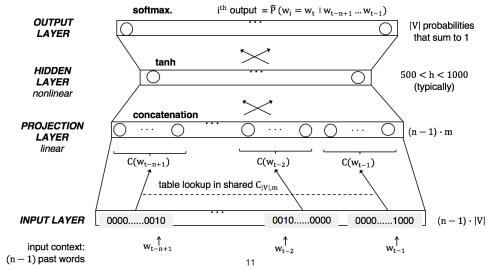
  i^{th} unit in the output layer: $\hat{P}(w_i = w_t \mid w_{t-n+1} \dots w_{t-1}) = \frac{e^{y w_i}}{\sum_{i'=1}^{|V|} e^{y w_{i'}}}$

  where:
  - $y = b + U.tanh(d + H.x)$
  - $\tanh$: 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 : $|V| * h$ matrix storing the *hidden-to-output* weights
  - H : $(h * (n-1)m)$ matrix storing the *projection-to-hidden* weights

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

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

    computational bottleneck: **nonlinear hidden layer** ($h * |V|$ term)



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

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

(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
- in terms of test *set perplexity*: geometric average of $1/\widehat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$
- Due to **complexity**, NNLM can't be applied to large data sets → 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 1000X speedup)

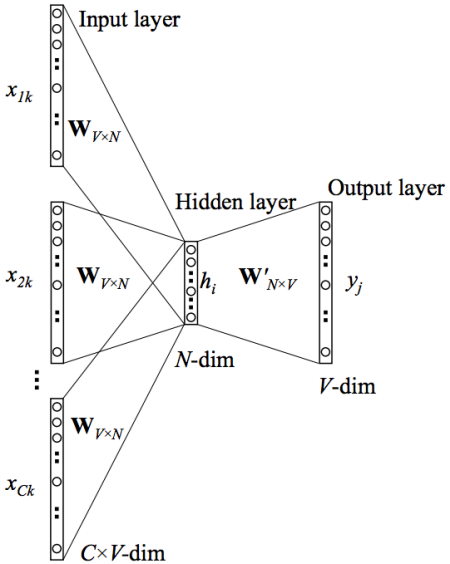
- projection layer is shared (not just the weight matrix) - C

- 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

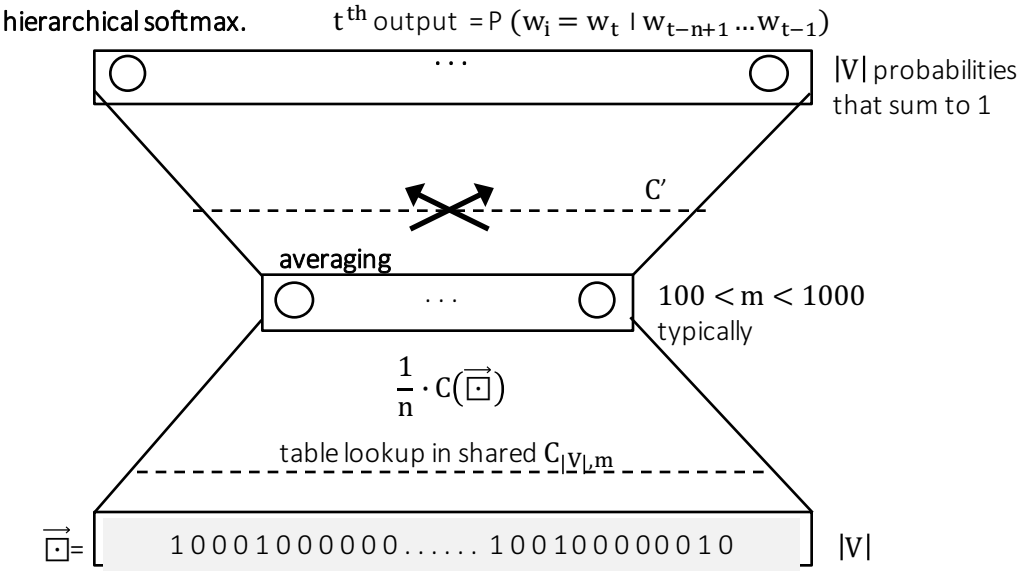


Efficient Estimation of Word Representations in Vector Space- Mikolov et al.

Continuous Bag-of-Words (CBOW)

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

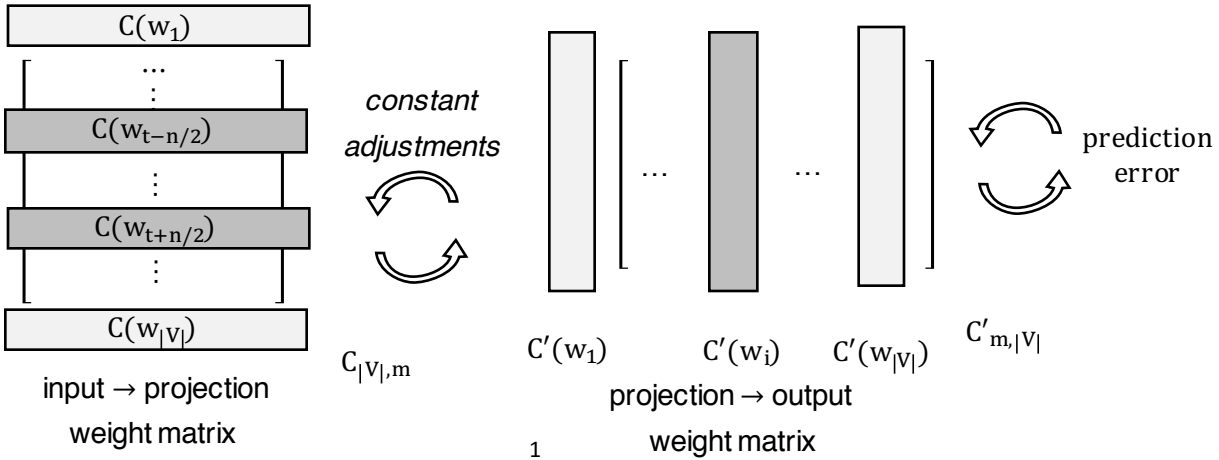
objective: minimize $-\log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$



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, in a context, the center word to predict the surrounding words

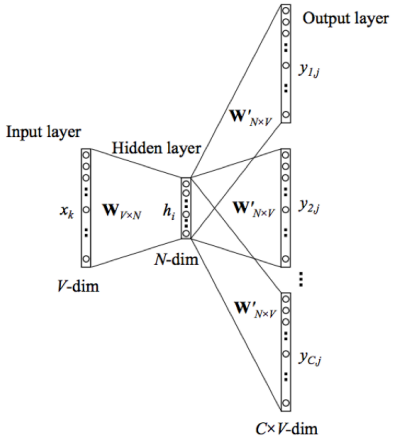
- instead of computing the probability of the target word w_t given its previous words, we calculate the probability of the surrounding word w_{t+j} given w_t

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

- v^T_{wt} is a column of W_{VxN} and $v'_{w_{t+j}}$ is a column of W'_{NxV}

- 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. 2013

word2vec facts

- Complexity is $\mathbf{n} * \mathbf{m} + \mathbf{m} * \mathbf{log}|\mathbf{V}|$ (Mikolov et al. 2013a)
- **n**:size of the context window (~10) **nxm**: dimensions of the projection layer, **|V|** size of the vocabulary
- 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/

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

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

- *Cast this into a least square problem:*

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

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}.$$

- X co-occurrence matrix
- f weighting function,
- b bias terms
- $w_i = word\ vector$
- $\widetilde{w_j} = context\ vector$

model that utilizes
- count data
- bilinear prediction-based methods like word2vec

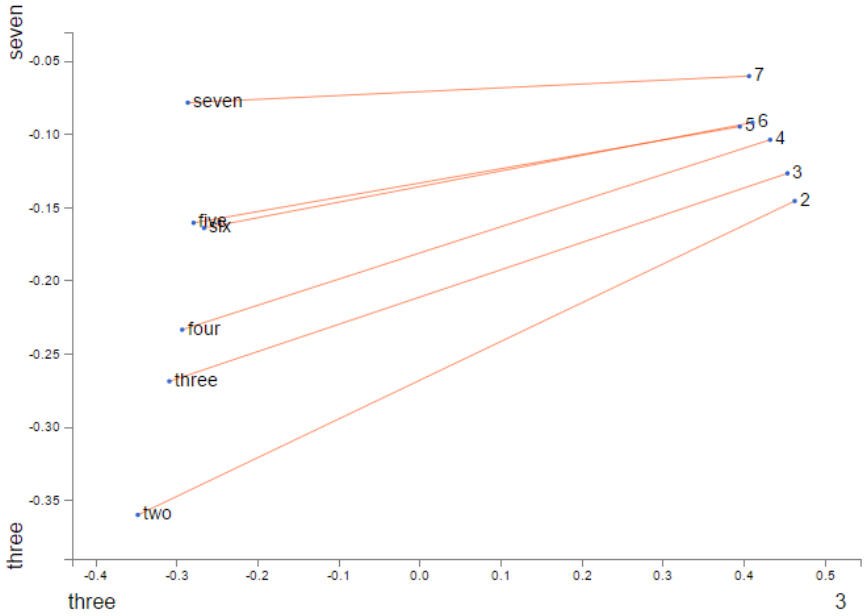
https://nlp.stanford.edu/pubs/glove.pdf

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

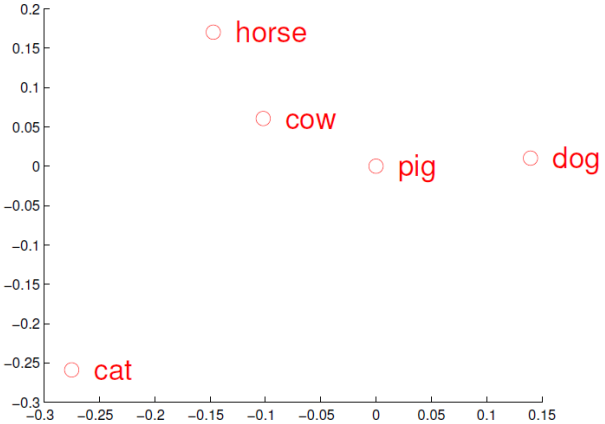
- Word analogies
- Find similar words
  - Semantic similarity
  - Syntactic similarity
- POS tagging
- Similar analogies for different languages
- Document classification

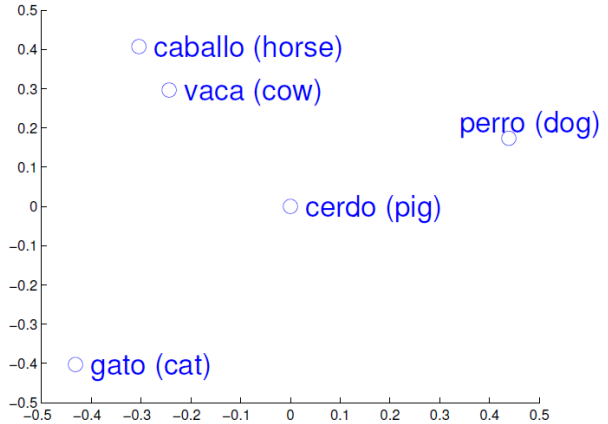


https://lamyiowce.github.io/word2viz/

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:

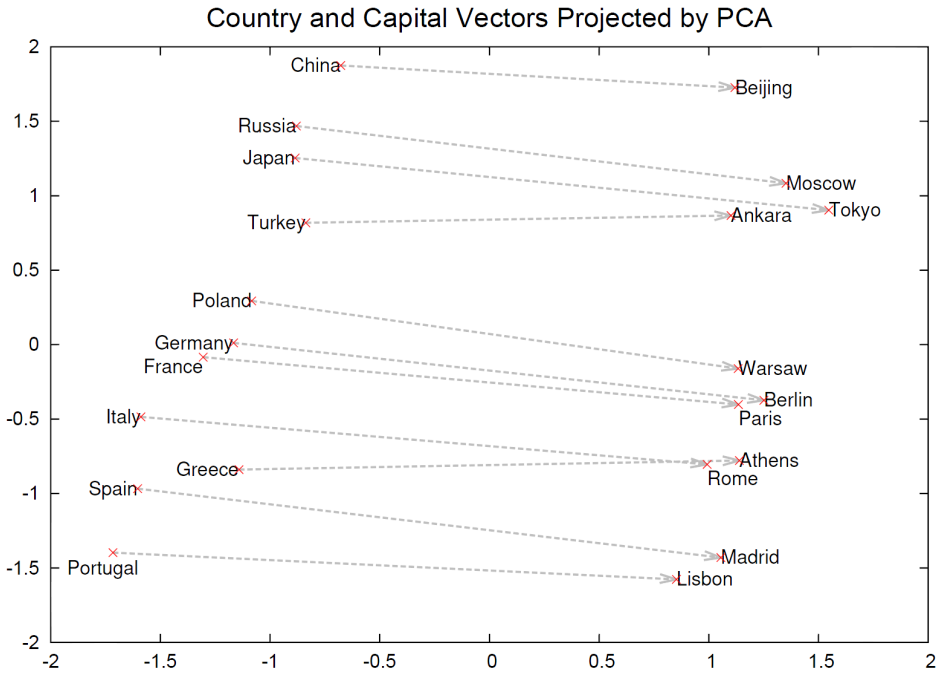




About 90% reported accuracy (Mikolov et al. 2013c)

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

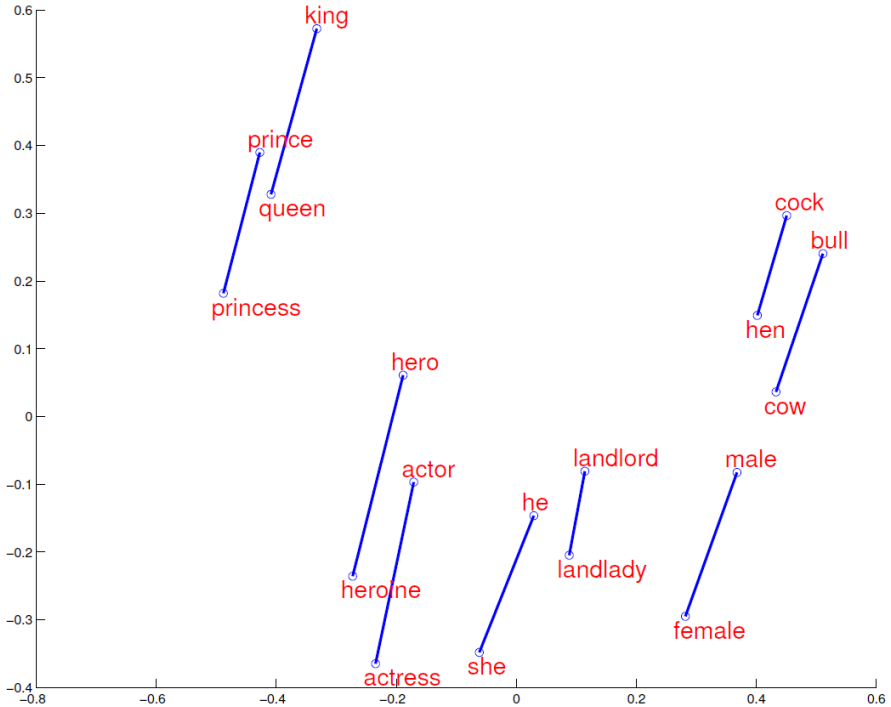
Remarkable properties of word vectors



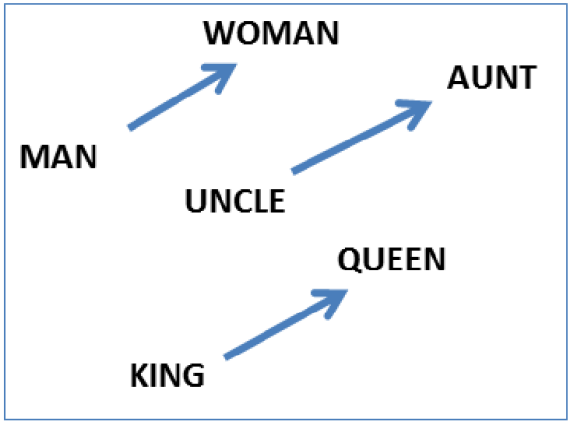
regularities between words are encoded in the difference vectors
e.g., there is a constant **country-capital** difference vector

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

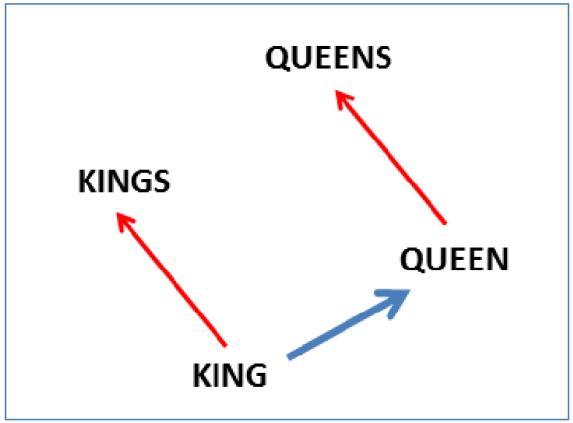
Remarkable properties of word vectors



constant **female-male** difference vector http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd



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/

CNN for text classification

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

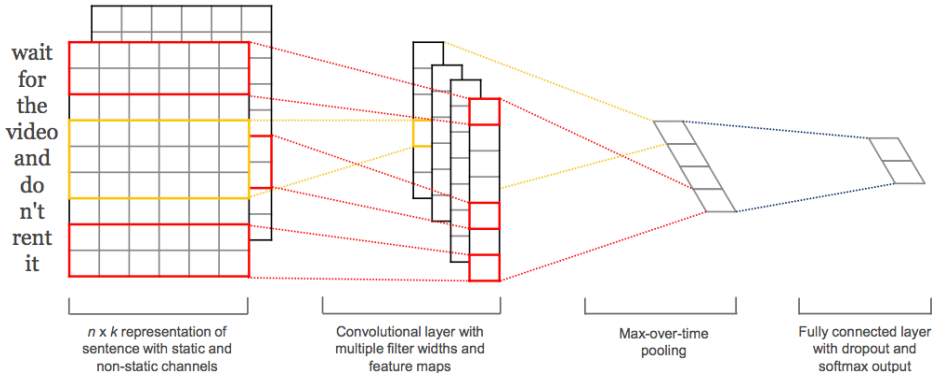
$$\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \ldots \oplus \mathbf{x}_n$$

- 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

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



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 back-propagation (CNN-rand)
- Modify existing vectors for the specific task through back-propagation(CNN-non-static)

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

CNN for text classification

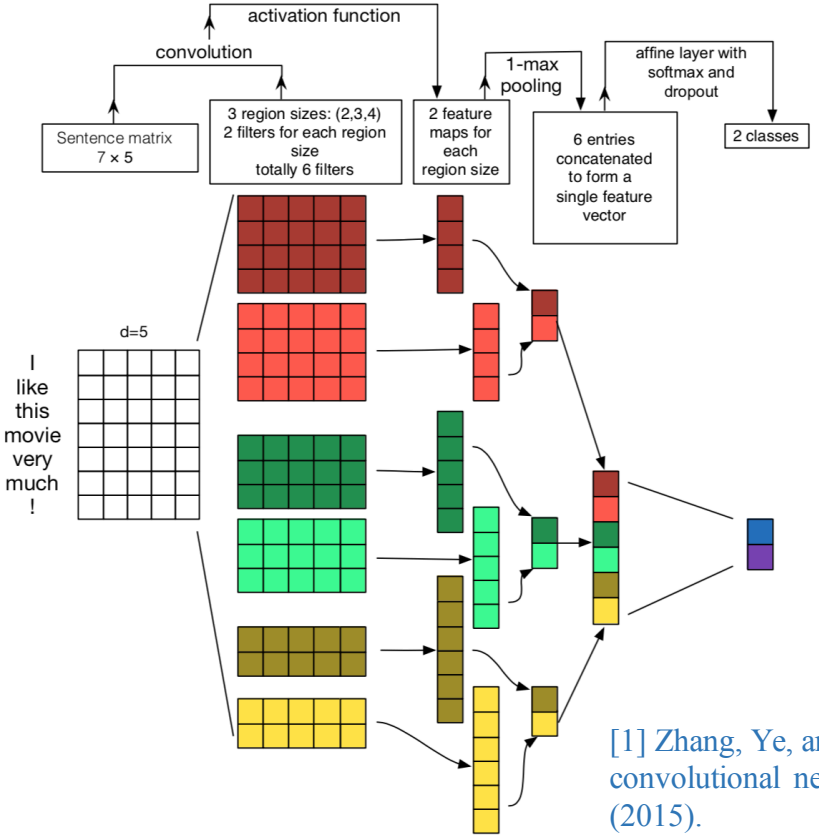
- Combine multiple word embeddings
- Each set of vectors is treated as a ‘channel’
- Filters applied to all channels
- Gradients are back-propagated only through one of the channels
- Fine-tunes one set of vectors while keeping the other static

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 architecture for (short) document classification [1]



- Data (text) only 1st 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 ~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 => classification via softmax

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

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

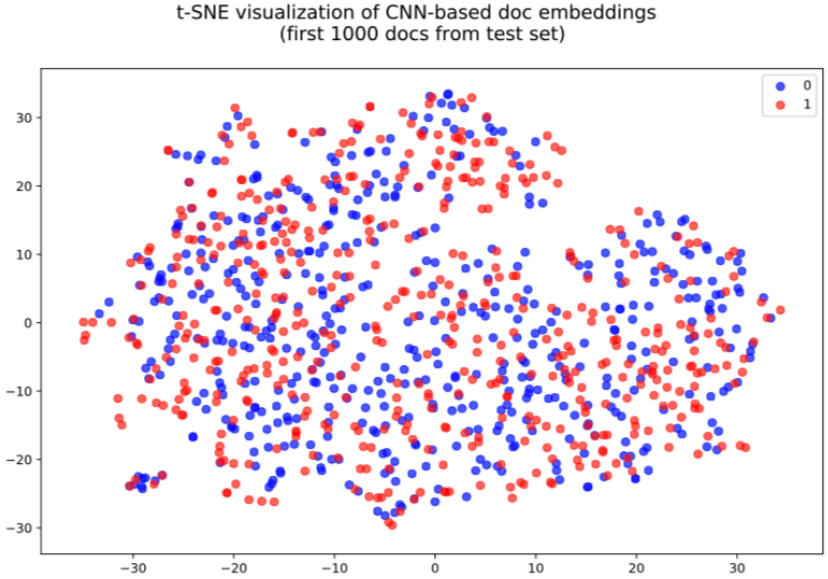


Figure 2: Doc embeddings before training.

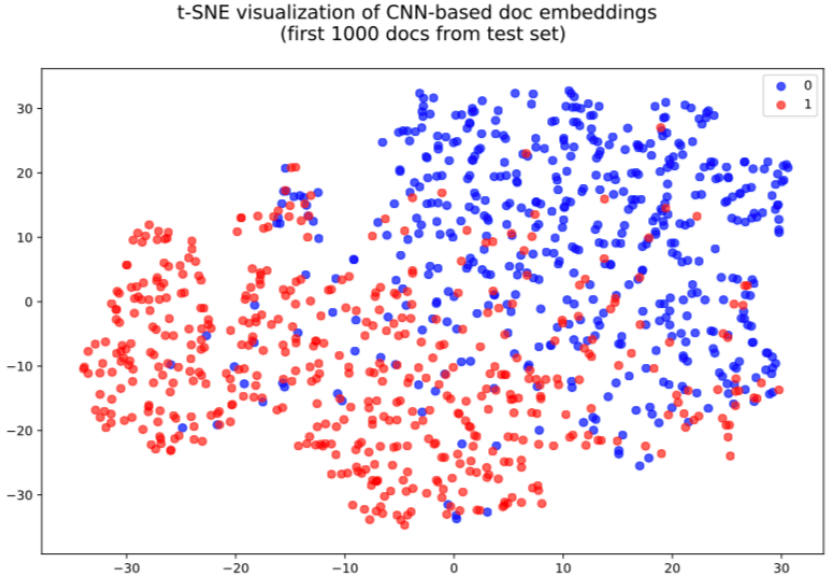


Figure 3: Doc embeddings after 2 epochs.

CNN architecture for (short) document classification - Saliency maps

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

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

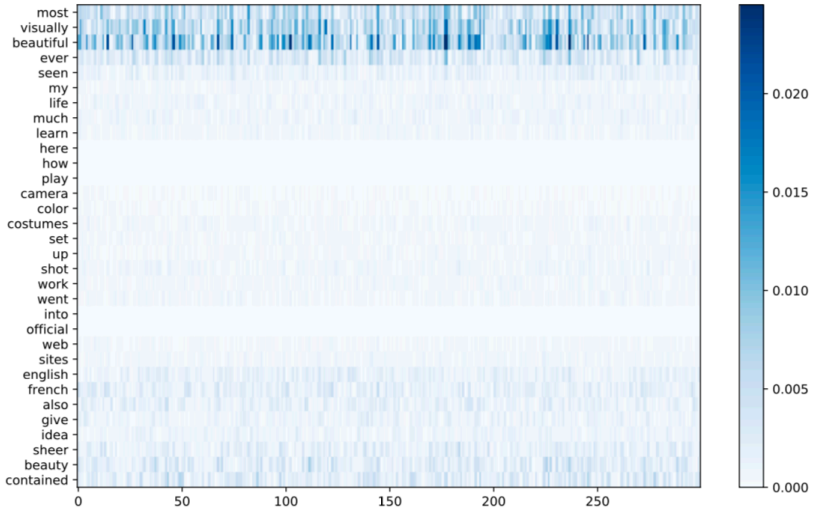


Figure 4: Saliency map for document 1 of the IMDB test set (true label: positive)

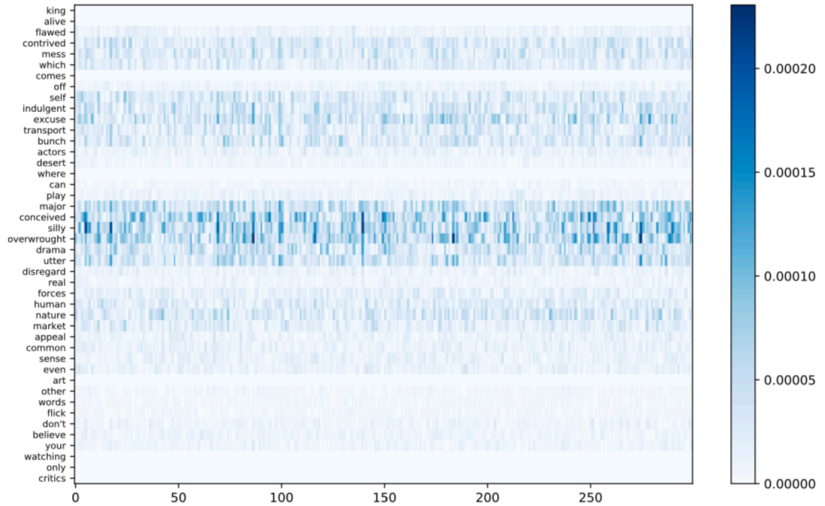
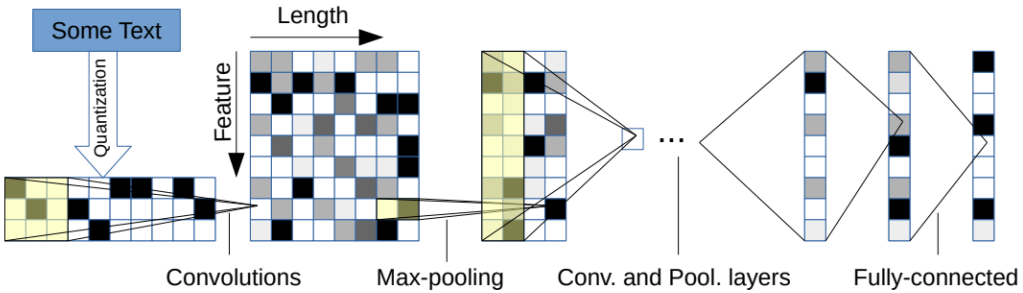


Figure 5: Saliency map for document 15 of the IMDB test set (true label: negative)

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

links

- http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
- https://arxiv.org/pdf/1509.01626.pdf
- http://www.aclweb.org/anthology/D14-1181
- http://cs231n.github.io/convolutional-networks/
- http://ufldl.stanford.edu/tutorial/supervised/Pooling/

- MLP + CNN Applications
  - Word embeddigs
- **Recurrent NNs + LSTMs**

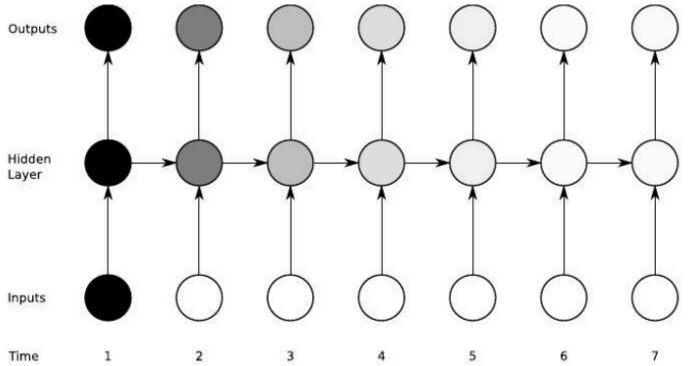
Recurrent Neural Networks

- CNNs are good at image classification
  - Input: an image
  - Output: probability of the class
    - $p(Red\ Panda|\ image) = 0,9$
    - $p(Cat\ |\ image) = 0,1$

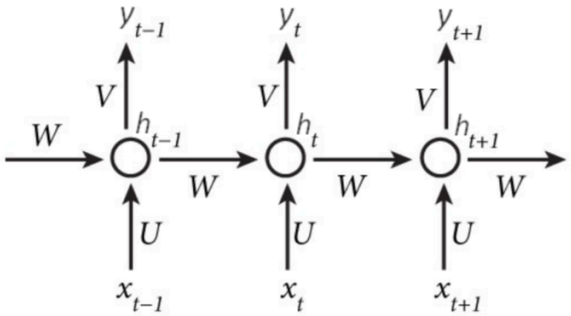
- Sequence learning: study of machine learning algorithms designed for sequential data (time series, language…)
  - Translate: "*machine learning is a challenging topic*" to French: "*L'apprentissage automatique est un sujet difficile*"
  - Predict the next word: "*John visited Paris, the capital of ….* "
  - Input and output strongly correlated within the sequence.

Recurrent Neural Networks

- Update the hidden state in a deterministic nonlinear way.
- RNNs are powerful,
- Distributed hidden state that allows them to store a lot of information about the past efficiently.
- Non-linear dynamics that allows them to update their hidden state in complicated ways.
- No need to infer hidden state, pure deterministic.
- Weight sharing

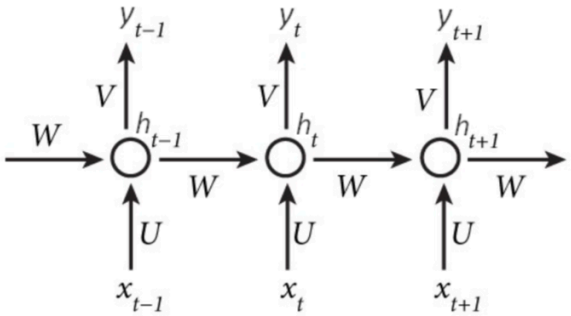


Recurrent Neural Networks



- input: ordered list of input vectors $x_1, \ldots, x_T$ initial hidden state h_0 initialized to all zeros,
- output
  - ordered list of hidden states $h_1, \ldots, h_T$,
  - ordered list of output vectors $y_1, \ldots, y_T$.
  - The output vectors may serve as input for other RNN units, when considering deep architectures .
  - The hidden states correspond to the “short-term” memory of the network.
- The last hidden state represents the encoding (embedding) of the time series

Recurrent Neural Networks



$$h_t = f(Ux_t + Wh_{t-1} + b)$$

- f a nonlinear function
- $x_t \in R^{d_{in}}, U \in R^{Hxd_{in}}, W \in R^{HxH}$,

Parameter matrices

- d_{in} size of the vocabulary
- H: dimension of the hidden layer (H~100)
- $y_t \in R^{d_{out}}$ transforms the current hidden state h_t to depend on the final task:
  - i.e. for classification $y_t = softmax(Vh_t)$
- $V \in R^{d_{out}\chi H}$ parameter matrix shared across all steps (i.e. for word level language model $dout=|V|$

Deep RNNs

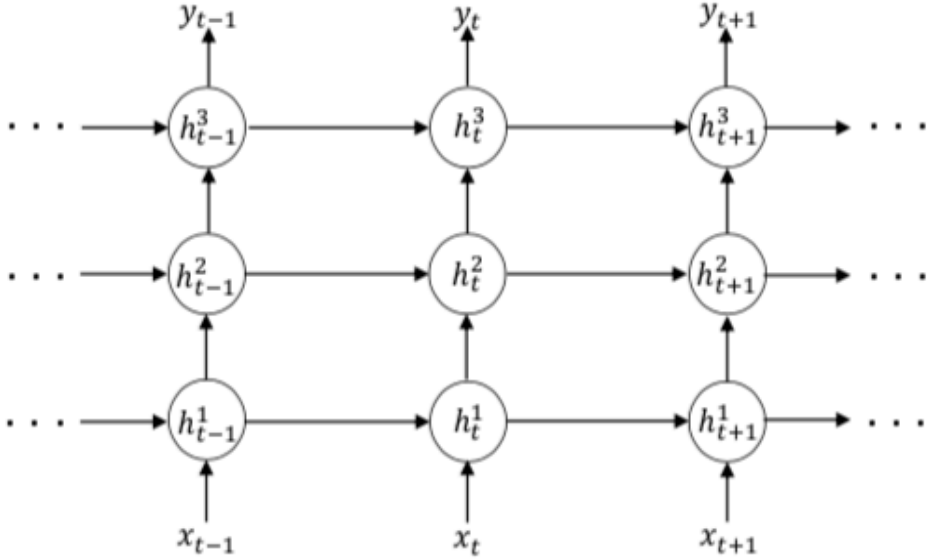
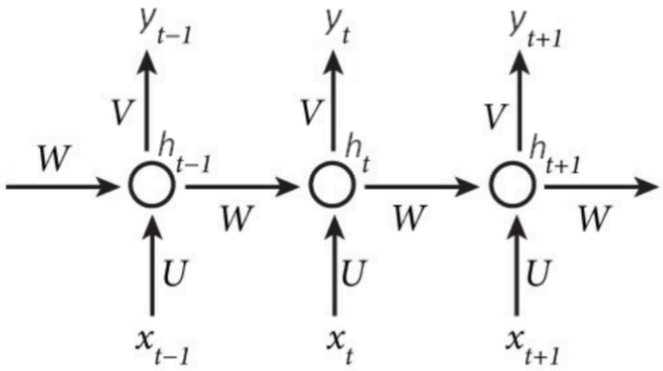


Figure 7: 3 steps of an unrolled deep RNN. Each circle represents a RNN unit. The hidden state of each unit in the inner layers (1 & 2) serves as input to the corresponding unit in the layer above.

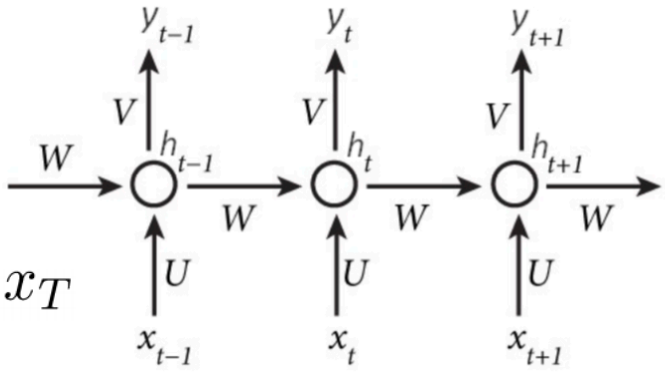
Recurrent Neural Networks



3 steps of an unrolled RNN

- CNNs are naturally efficient with grids[8],
- RNNs were specifically developed to be used with sequences
  - time series, or, in NLP, words (sequences of letters) or sentences (sequences of words).
  - language modeling P [$w_n | w_1, ..., w_{n-1}$].
  - RNNs trained with such objectives can be used to generate new and quite convincing sentences from scratch
  - RNN can be considered as a chain of simple neural layers that share the same parameters.

Learning in RNNs



- Assuming input $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- As a single time: $h_t = \sigma(W h_{t-1} + U x_t)$

$$y_t = softmax(V h_t)$$

Learning in RNNs

$$h_t = \sigma(W h_{t-1} + U x_t)$$
$$y_t = softmax(V h_t)$$

- Main idea: we use the same set of W,U,V weights at all time steps!
- $h_0 \in R^{HxH}$ initialization vector for the hidden layer at time step 0
- $\hat{y} \in R^{|V|}$ is a probability distribution over the vocabulary
- Loss function (@ time t): cross entropy predicting words instead of classes

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Learning in RNNs – bacpropagation in time

- Learning the weights of W:

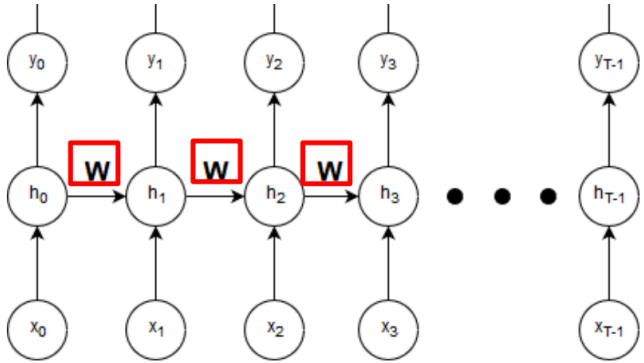
$$\mathbf{W} -> \mathbf{W} - \alpha \frac{\partial \mathbf{y}}{\partial \mathbf{W}}$$

- Computation involves summation over all paths regarding
- i. time

$$\frac{\partial \mathbf{y}}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \frac{\partial \mathbf{y}_j}{\partial \mathbf{W}}$$

- ii. Levels of hidden layers

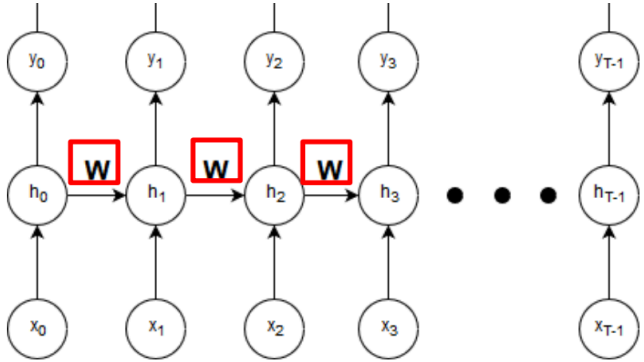
$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



Learning in RNNs – backpropagation in time

- Therefore:

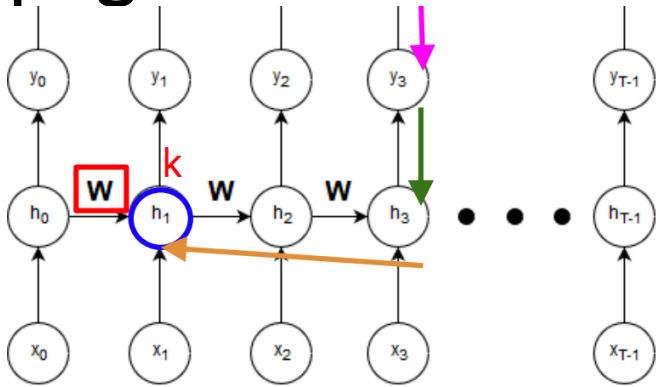
$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \frac{\partial h_j}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}}$$



- Indirect dependency. One final use of the chain rule:

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}}$$

Learning in RNNs – back propagation in time



$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

Learning in RNNs – vanishing/exploding gradients

$$\frac{\partial y_j}{\partial \mathbf{W}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W}}$$

$$h_m = f(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)$$

$$\frac{\partial h_m}{\partial h_{m-1}} = \mathbf{W}_h^T \mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

$$\frac{\partial h_j}{\partial h_k} = \prod_{m=k+1}^{j} \mathbf{W}_h^T \mathrm{diag}\left(f'(\mathbf{W}_h h_{m-1} + \mathbf{W}_x x_m)\right)$$

Weight Matrix

Derivative of activation function

- Repeated matrix multiplications leads to vanishing and exploding gradients.

Learning in RNNs – vanishing/exploding gradients

- Initialization of W with 1s
- Using relu as activation function $f(z) = rect(z) = max(z, 0)$
- Using - clip gradients to a maximum value [Mikolov]

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

LSTM Unit

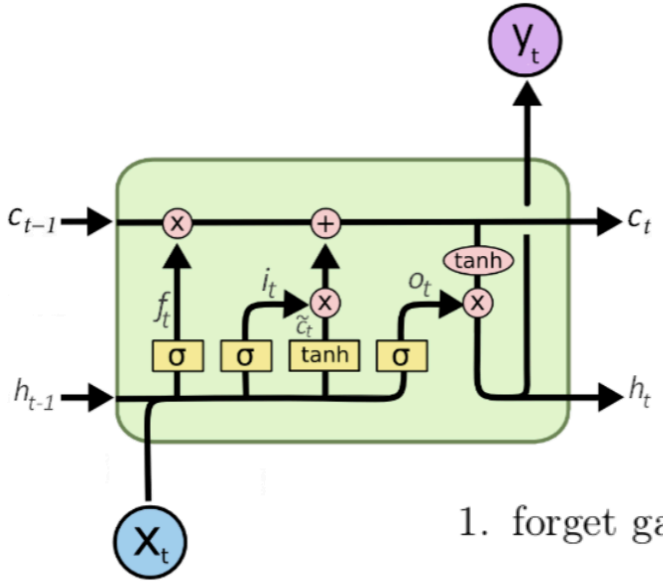
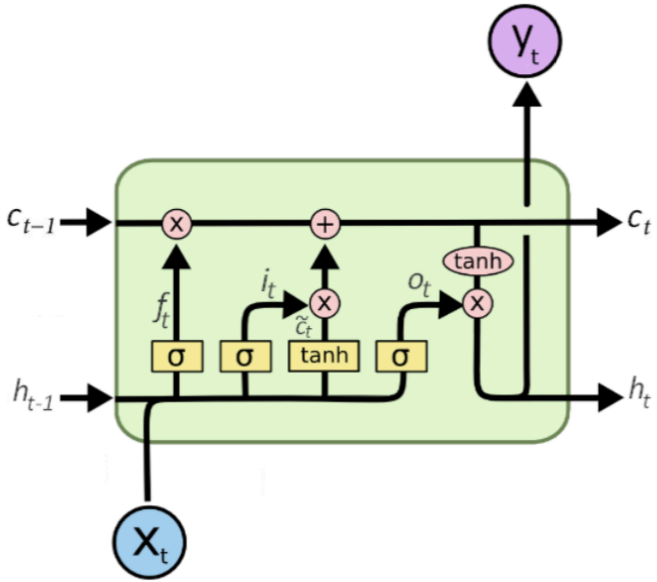


Figure 8: The LSTM unit. Adapted fro

- To tackle RNN problems of i. vanishing gradients and ii. Keep track of information for longer term.
- There is a "memory bus" Ct on which we chose how much to write and read

1. forget gate layer: $f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right)$
2. input gate layer: $i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right)$
3. candidate values computation layer: $\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$
4. output gate layer: $o_t = \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right)$

LSTM Unit



1. forget gate layer: $f_t = \sigma\left(U_f x_t + W_f h_{t-1} + b_f\right)$
2. input gate layer: $i_t = \sigma\left(U_i x_t + W_i h_{t-1} + b_i\right)$
3. candidate values computation layer: $\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$
4. output gate layer: $o_t = \sigma\left(U_o x_t + W_o h_{t-1} + b_o\right)$

Assume a new training example x_t and the current hidden state h_{t-1},

- forget gate layer f_t determines how much of the previous cell state c_{t-1} should be forgotten (what fraction of the memory should be freed up),
- input gate layer decides how much of the candidate values $\tilde{c_t}$ should be written to the memory: how much of the new information should be learned. Combining the output of the two filters updates the cell state:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = \tanh(c_t) \circ o_t$$

$$y_t = \text{softmax}(V h_t)$$

References (1/2)

1. Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.
2. Hadsell, Raia, Sumit Chopra, and Yann LeCun. "Dimensionality reduction by learning an invariant mapping." *Computer vision and pattern recognition, 2006 IEEE computer society conference on*. Vol. 2. IEEE, 2006.
3. Luong, Minh-Tang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).
4. Mueller, J., & Thyagarajan, A. (2016, February). Siamese Recurrent Architectures for Learning Sentence Similarity. In *AAAI* (pp. 2786-2792).
5. Course slides "Recurrent Neural Networks" Richard Socher, Stanford, 2016
6. Recurrent Neural Network Architectures Abhishek Narwekar, Anusri Pampari, University of Illinois, 2016
7. Neculoiu, Paul, Maarten Versteegh, and Mihai Rotaru. "Learning text similarity with siamese recurrent networks." *Proceedings of the 1st Workshop on Representation Learning for NLP*. 2016.
8. Rush, Alexander M., Sumit Chopra, and Jason Weston. "A neural attention model for abstractive sentence summarization." arXiv preprint arXiv:1509.00685 (2015).
9. Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.
10. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).
11. Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical attention networks for document classification. NAACL 2016 (pp. 1480-1489).