Deep Learning - Embedding
深度学习 - 嵌入

Jian Li

IIIS, Tsinghua

Word2Vec

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

Build a cooccurance matrix (using a moving window)

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

- high dimensional, and sparse
- we want relatively low-dim representation

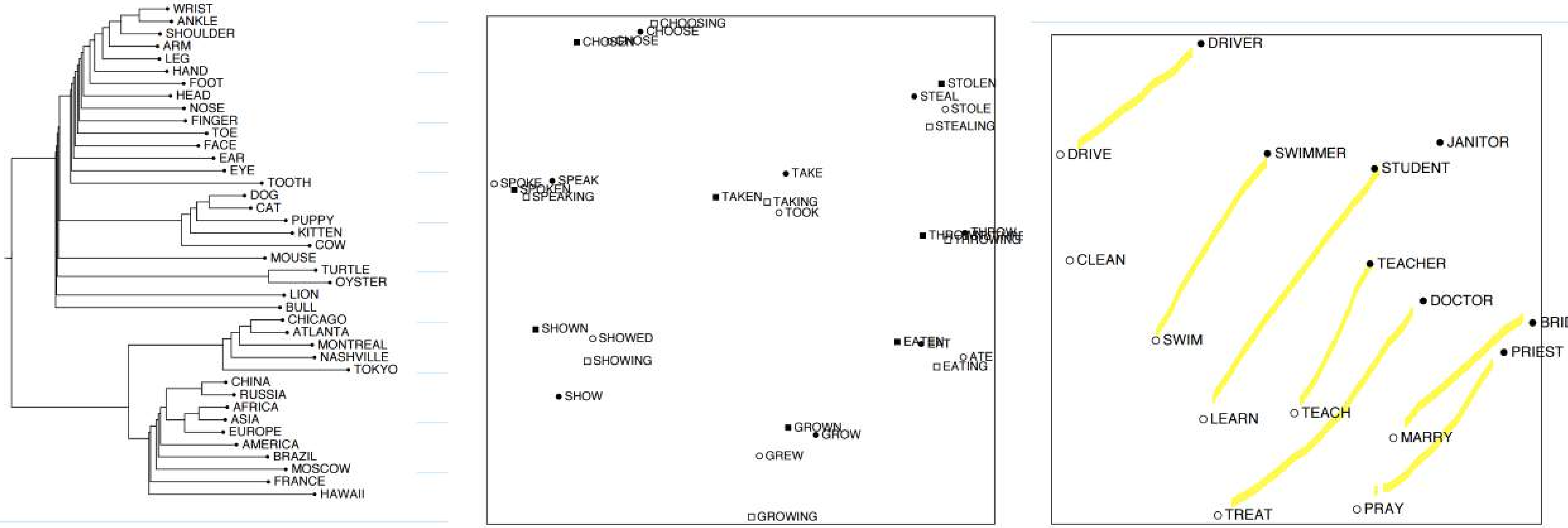
Word Embedding

- Map each word to a vector (in relatively low-dim space)

  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

approach 1 SVD/PCA

- Some notes: remove "of" "the" (syntatic words)
- Use Pearson correlations (rather than count) $\rho_{XY} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-EX)(Y-EY)]}{\sqrt{Var(X)Var(Y)}}$
    - set neg values to zero



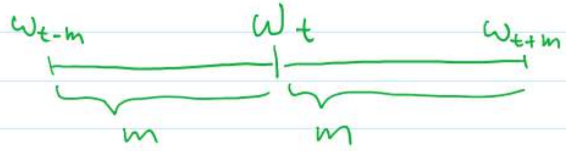
problem: SVD expensive for large vocabulary

special algorithm (doesn't fit into DL pipeline)

Word2Vec

word 2 Vec (Mikolov et al. 2013)

Predict surrounding words in a window (of len m) of every word



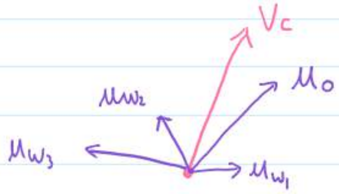
First define a probabilistic model

each word w has two vectors $\begin{cases} u_w & \text{(the embedding of } w \text{ when } w \text{ is outside)} \\ v_w & \text{(- - - - - - - - - - - - center)} \end{cases}$

We want to learn u_w, v_w for $w \in W$

$$P(o \mid c) = \frac{\exp(\langle u_o, v_c \rangle)}{\sum_{w=1}^{W} \exp(\langle u_w, v_c \rangle)} \quad \leftarrow \text{softmax/logistic regression}$$

outside word o   center word c



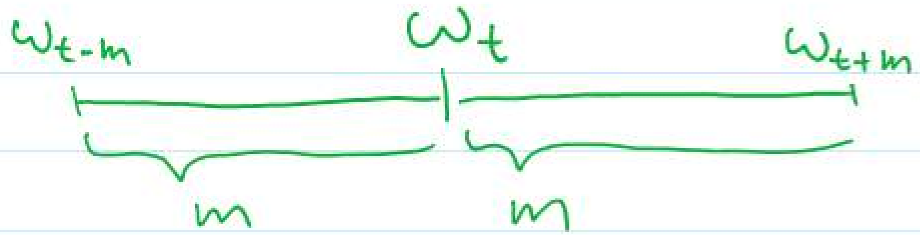
if u_o v_c in the same direction, $P(o|c)$ is large.

Objective of Word2Vec

Objective : $$J = \frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P(\omega_{t+j} \mid \omega_t)$$

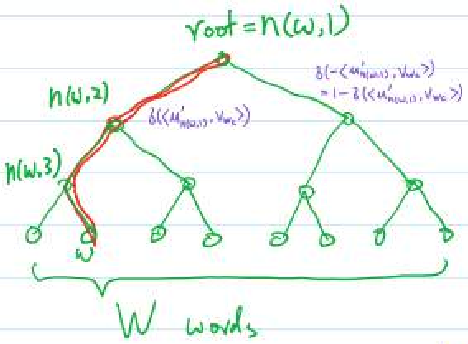
Sum over all word

log Prob of surrounding word of ω_t



training: $\nabla \log P(w_o | w_c)$ is expensive ($O(W)$ time)

① Hierarchical Softmax (it is an approximation)



$n(w,j)$: jth node on the root → w path

$L(w)$: length of root → w path

for every inner node, we also have a vector $u'_{n(w,j)}$

for every leaf word w. only one vector v_w

$$p(w|w_c) = \prod_{j=1}^{L(w)-1} \sigma\Big(\underbrace{\mathbb{1}}_{\pm 1 \text{ indicator}} \big\{ n(w,j+1) = \underbrace{\mathrm{ch}(n(w,j))}_{\text{left child of } n(w,j)} \big\} \langle u'_{n(w,j)}, v_{w_c} \rangle \Big)$$

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

$$\sum_{w=1}^{W} p(w|w_c) = 1$$ (easy to check)

— not necessarily a full binary tree. e.g. the paper uses Huffman tree (frequent word → shorter path)

— time. len of the path.

Training method 2

Negative Sampling.

Noise Contrastive Estimation (NCE) (Gutmann & Hyvarinen)

a good model should be able to differentiate data from noise (by logistic regression)

Negative sampling objective ($\sum \log \sigma(\ )$ vs $\log \sum \sigma(\ )$)

replace $\log P(w_o|w_c)$ by $\log \sigma(\langle u_{w_o}, v_{w_c}\rangle) + \sum_{i=1}^{K} \log \sigma(-\langle u_{w_i}, v_{w_c}\rangle)$

$w_i \sim P(w) \leftarrow$ noise distributions $\begin{cases} \text{uniform distr} \\ \text{Unigram distr } U(w) \text{ (i.e, word frequency)} \\ U(w)^{\frac{3}{4}}/Z \end{cases}$

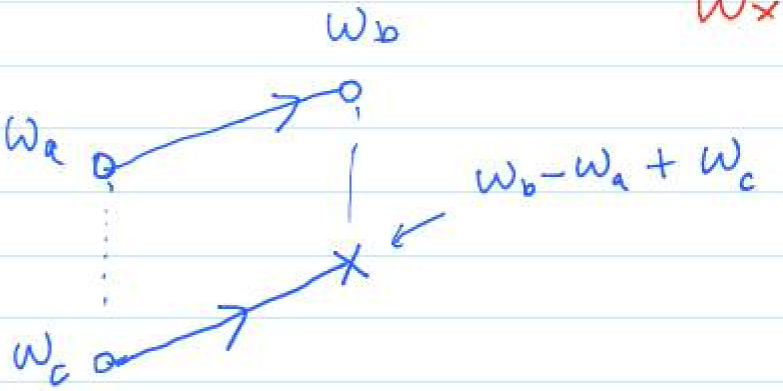
($K \sim 5\text{-}20$ typically. if data is large. $K \sim 2\text{-}5$)

- $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
- $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
- $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

$$\boxed{a:b :: c:?} \longrightarrow \boxed{d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}}$$

w_x produces the largest $\langle \nearrow , w_x \rangle$

direction of $w_b - w_a + w_c$



Notes

- Word2Vec: unsupervised learning
  - Huge amount of training data (no label is needed)

- Can be incorporated to deep learning pipeline
  - The corresponding layers is usually called the embedding layer
  - The resulting vectors obtained from word2vec can be used to initialize the parameters of NN
  - We can also get the embedding from training a specified DNN (for a specific task)
    - Computational complexity too high (much higher than word2vec)
    - The embedding may not be useful in other tasks
    - On the other hand, word2vec captures a lot of semantic information, which is useful in a variety of tasks

According to Mikolov:

CBOW (Continuous Bag of Words): Use context to predict the current word.

--several times faster to train than the skip-gram, slightly better accuracy for the frequent words

Skip-gram: Use the current word to predict the context.

--works well with small amount of the training data, represents well even rare words or phrases.

GloVe

GloVe: Global Vectors for Word Representation

Two methodologies

① globle matrix factorization

② local context window (skip-gram)

predicting the context given a word

Ratio Matters

Co-occurrence matrix $X = \begin{bmatrix} X_{ij} \end{bmatrix}$ ← # times word j occurs in the context of word i

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} = \frac{X_{ij}}{\sum_j X_{ij}}$$

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

Solid is more relevant to ice than to steam

gas is less relevant to ice than to stream

Derivation

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

$\in \mathbb{R}^d$

Using the property of F, we try to derive the form of $\langle w_i, \tilde{w}_k \rangle$

want ① $F(w_i, w_j, \tilde{w}_k) = F(\langle w_i - w_j, \tilde{w}_k \rangle)$

enforce a linear structure

② Imagine we switch the role of a word & a context

i.e. $X \to X^T$

then we'd better have $w_i \to \tilde{w}_i$ (symmetric)

So we choose $F = \exp$

$$\left(\exp(\langle w_i - w_j, \tilde{w}_k \rangle) = \frac{\exp\langle w_i, \tilde{w}_k \rangle}{\exp\langle w_j, w_k \rangle} = \frac{P_{ik}}{P_{jk}} = \frac{X_{ik}/X_i}{X_{jk}/X_j}\right)$$

$$\Rightarrow \boxed{\langle w_i, \tilde{w}_k \rangle + b_i + \tilde{b}_k = \log(X_{ik})}$$

$\log X_i$ $\quad$ $\log X_j$

$$\left(\text{Verify} \quad \langle w_i - w_j, \tilde{w}_k \rangle = \log\left(\frac{X_{ik}}{X_{jk}}\right) - (b_i - b_j)\right)$$

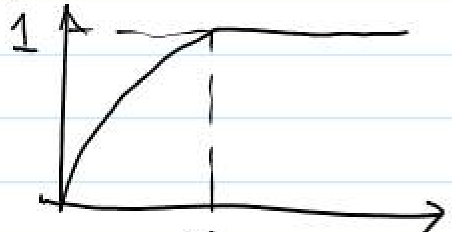
We try to factorize $\log X$ (i.e. $w\left(\,\right)^d \left(\ \tilde{w}\ \right) = \log\left(\quad\right)$)

Objective

Objective (weighted least square)

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

weight



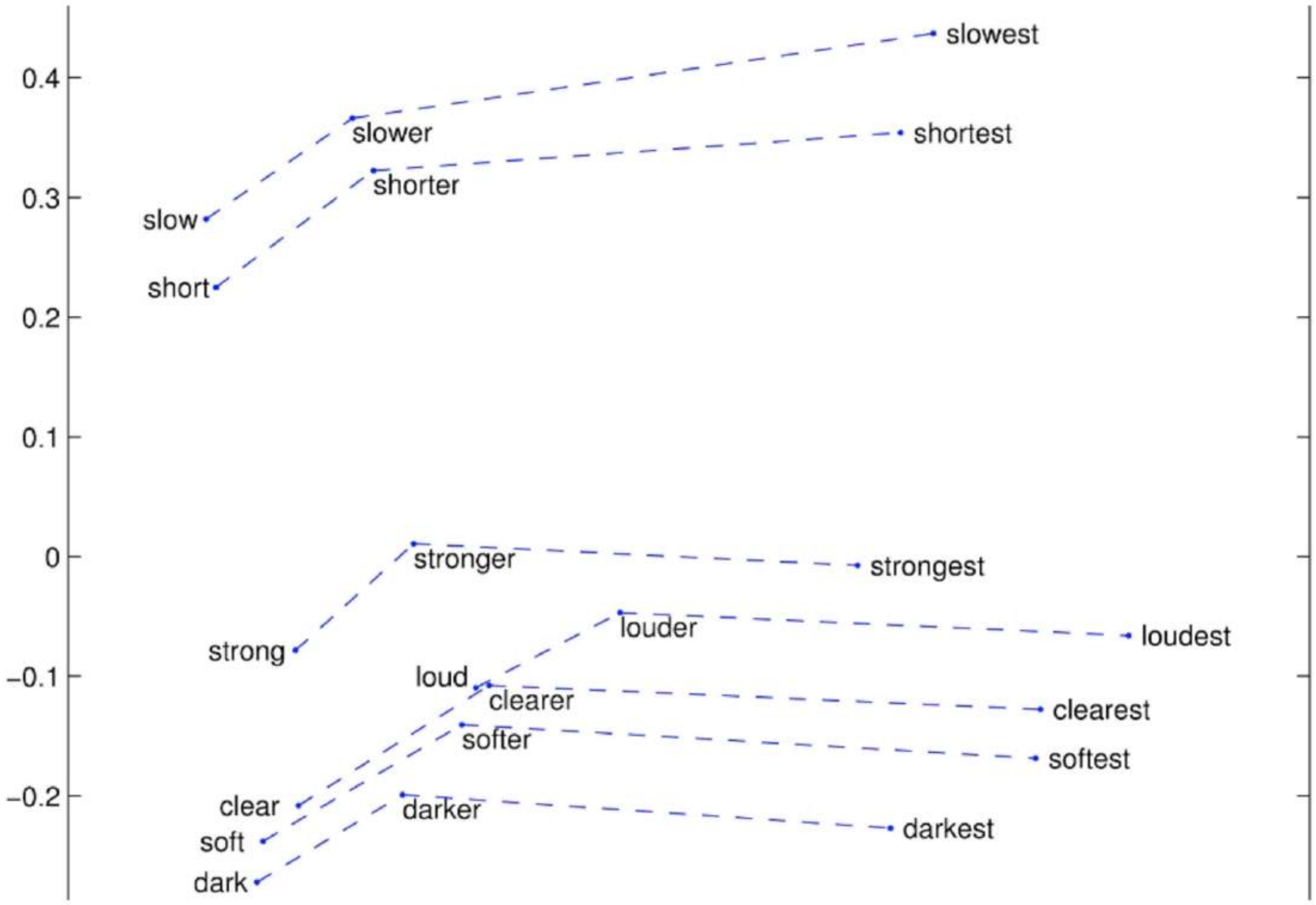
$X_{max} = 100$

$(\alpha = 3/4)$

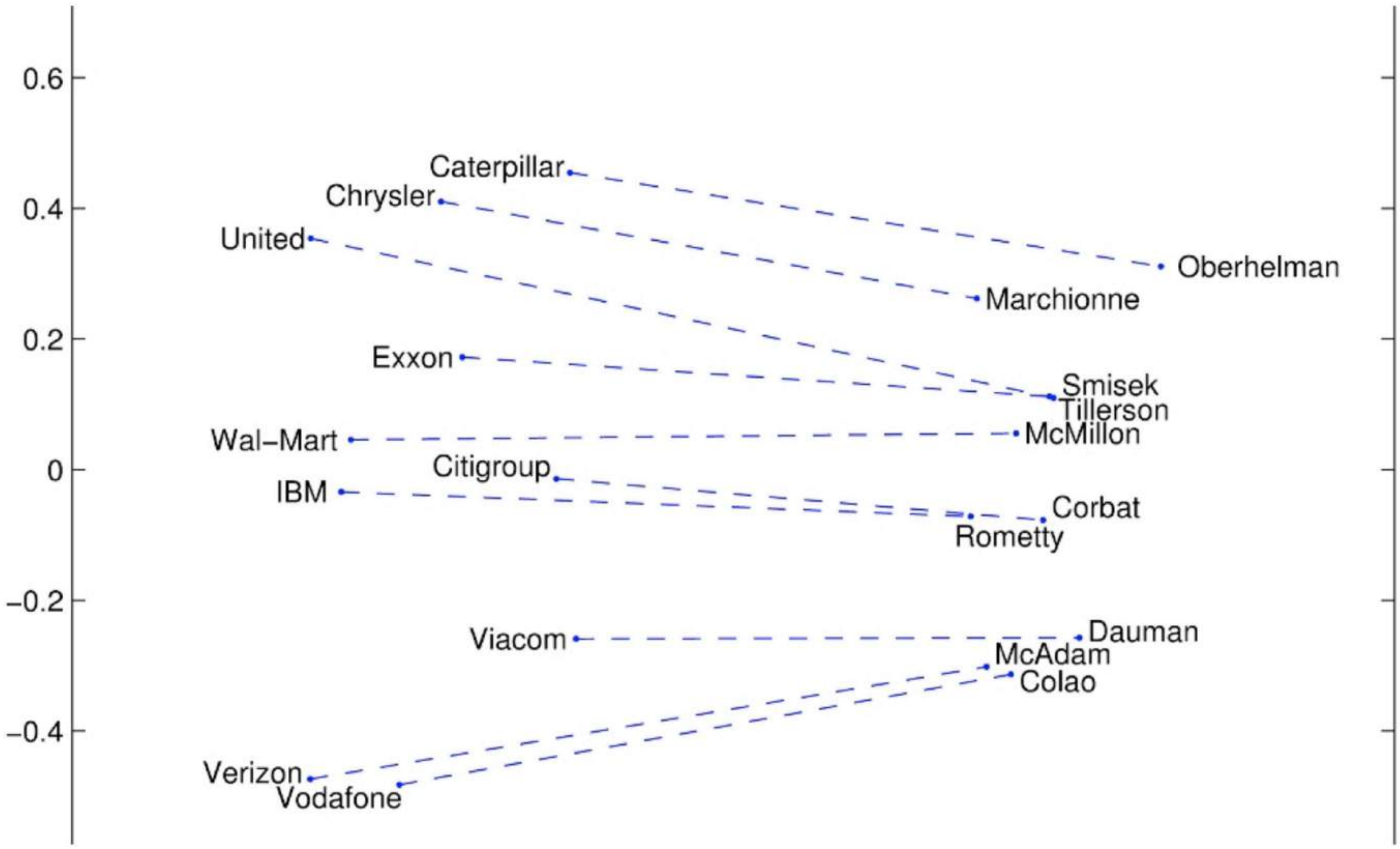
$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & x < x_{max} \\ 1 & \text{o.w.} \end{cases}$$

Training: SGD, Adagrad

Glove Visualizations: Superlatives



Glove Visualizations: Company - CEO



Glove results

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Reference

- Baroni, Marco, Georgiana Dinu, and Germán Kruszewski. Don‘t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. ACL 14
  - An early paper claims that the prediction formulation (like word2vec) is better than factorizing a co-occurrence matrix
- Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings.

  (comparing several SNSG, GloVe, and SVD)

A very reasonable blog discussing the relations between different models

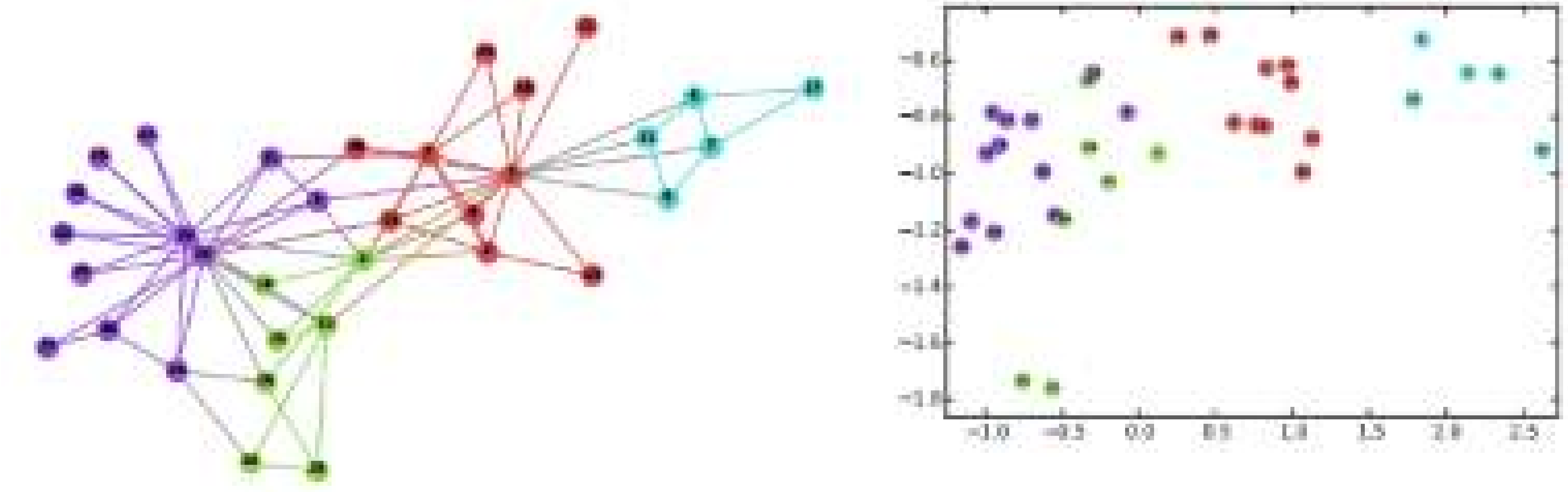
http://sebastianruder.com/secret-word2vec/index.html

- O Levy, Y Goldberg. Neural Word Embedding as Implicit Matrix Factorization. NIPS 14. (in blackboard)

- Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, Jie Tang. The 11th ACM International Conference on Web Search and Data Mining (WSDM 2018).

Deep Walk
-embedding node in a social network

Embedding node (using pairwise relations)



(a) Input: Karate Graph

(b) Output: Representation

Key Idea

treat vertex as words , random walks as sentences

Algorithm 1 DEEPWALK(G, w, d, γ, t)

Input: graph $G(V, E)$
window size w
embedding size d
walks per vertex γ
walk length t

Output: matrix of vertex representations $\Phi \in \mathbb{R}^{|V| \times d}$ ← all parameters

d

$\Phi = |V|$

Initialization: Sample Φ from $\mathcal{U}^{|V| \times d}$
Build a binary Tree T from V
for $i = 0$ to γ **do**
  $\mathcal{O} = \text{Shuffle}(V)$ ← random shuffle of all vertices. Speed up SGD
  for each $v_i \in \mathcal{O}$ **do**
    $\mathcal{W}_{v_i} = RandomWalk(G, v_i, t)$ ← Generate a random walk from v_i of length t
    SkipGram(Φ, $\mathcal{W}_{v_i}$, w) ← SGD update step
  end for
end for

Algorithm 2 SkipGram(Φ, $\mathcal{W}_{v_i}$, w)

for each $v_j \in \mathcal{W}_{v_i}$ **do** ← look at each vertex in the random walk v_j
  for each $u_k \in \mathcal{W}_{v_i}[j - w : j + w]$ **do** ← look at each word in the window
    $J(\Phi) = -\log \Pr(u_k \mid \Phi(v_j))$
    $\Phi = \Phi - \alpha * \frac{\partial J}{\partial \Phi}$
  end for
end for

} → SGD

Prob of u_k (context word) conditioning on the vector of v_j (center word)

Use Hierarchical softmax to approximate.

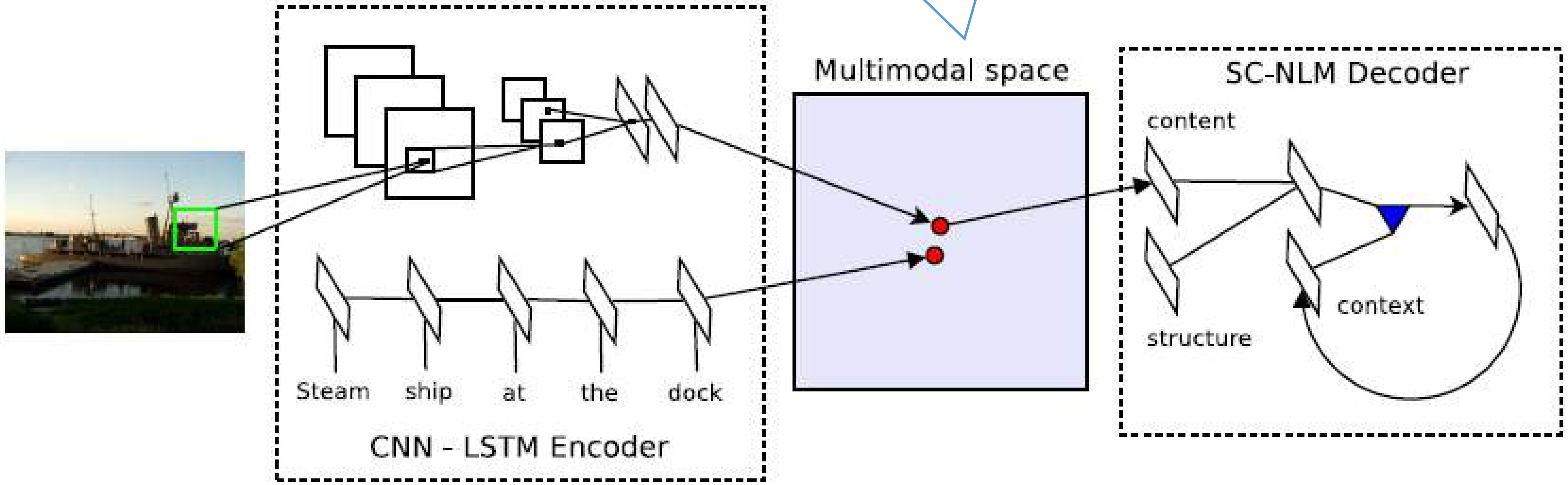
Multimodal representation learning
---Image Caption 2

Kires et al. Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models

Figure 1: Sample generated captions. The bottom row shows different error cases. Additional results can be found at http://www.cs.toronto.edu/~rkiros/lstm_scnlm.html

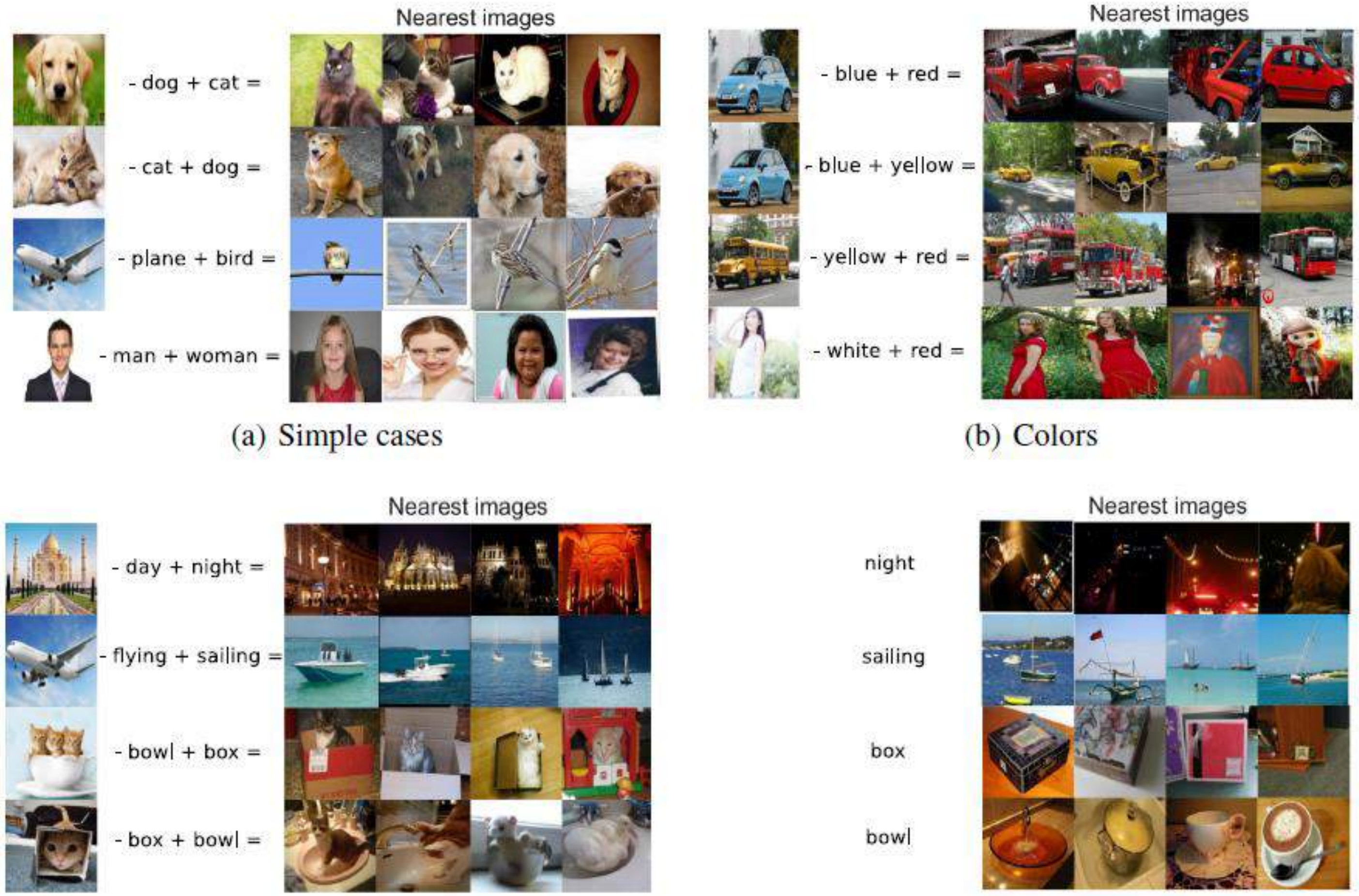
Overview

Map CNN codes and RNN code to a common space



Details of SC-NLM. Please see the paper

Figure 2: **Encoder:** A deep convolutional network (CNN) and long short-term memory recurrent network (LSTM) for learning a joint image-sentence embedding. **Decoder:** A new neural language model that combines structure and content vectors for generating words one at a time in sequence.



(a) Simple cases

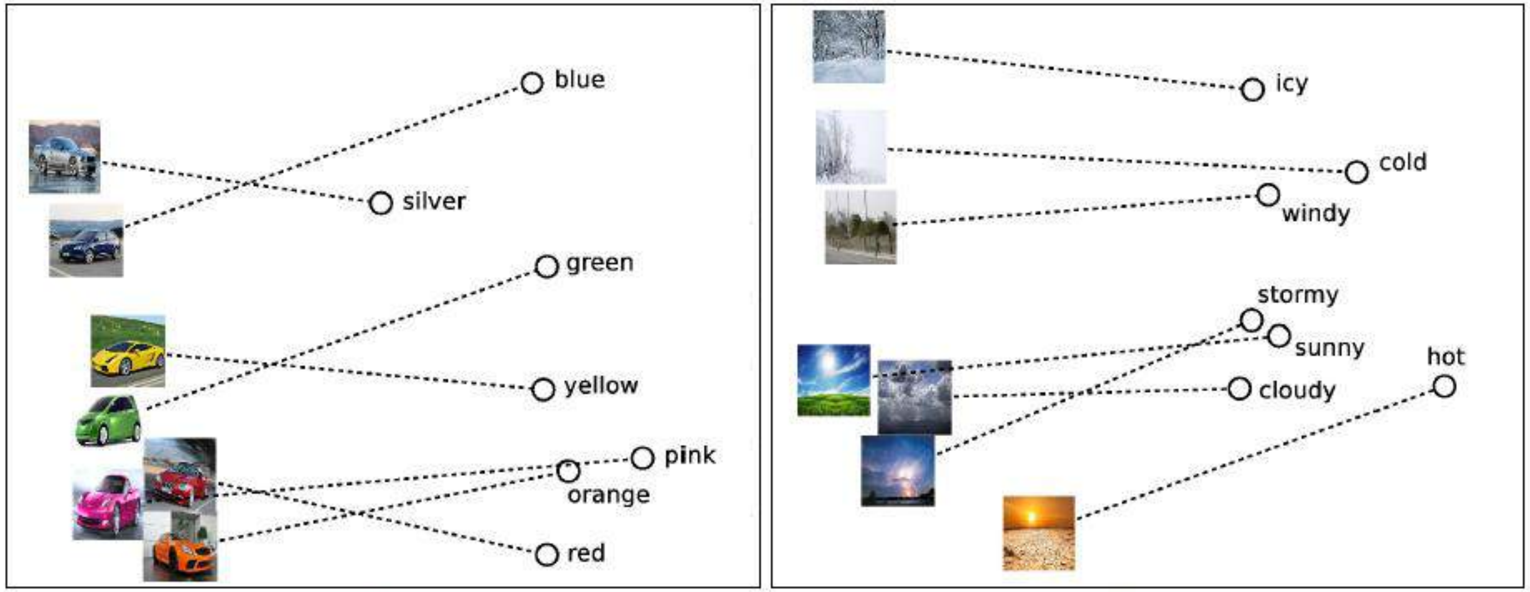
(b) Colors

(c) Image structure

(d) Sanity check

Figure 4: Multimodal vector space arithmetic. Query images were downloaded online and retrieved images are from the SBU dataset.

$$
\begin{aligned}
\mathbf{v}_{car} &\approx \mathbf{I}_{bcar} - \mathbf{v}_{blue} \\
\mathbf{v}_{red} + \mathbf{v}_{car} &\approx \mathbf{I}_{bcar} - \mathbf{v}_{blue} + \mathbf{v}_{red} \\
\mathbf{I}_{rcar} &\approx \mathbf{I}_{bcar} - \mathbf{v}_{blue} + \mathbf{v}_{red}
\end{aligned}
$$



(a) Colors (b) Weather

Figure 5: PCA projection of the 300-dimensional word and image representations for (a) cars and colors and (b) weather and temperature.

Details

- LSTM notations used in this work

Let $\mathbf{X}_t$ denote a matrix of training instances at time t. In our case, $\mathbf{X}_t$ is used to denote a matrix of word representations for the t-th word of each sentence in the training batch. Let $(\mathbf{I}_t, \mathbf{F}_t, \mathbf{C}_t, \mathbf{O}_t, \mathbf{M}_t)$ denote the input, forget, cell, output and hidden states of the LSTM at time step t. The LSTM architecture in this work is implemented using the following equations:

$$
\begin{aligned}
\mathbf{I}_t &= \sigma(\mathbf{X}_t \cdot \mathbf{W}_{xi} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{hi} + \mathbf{C}_{t-1} \cdot \mathbf{W}_{ci} + \mathbf{b}_i) && (1)\\
\mathbf{F}_t &= \sigma(\mathbf{X}_t \cdot \mathbf{W}_{xf} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{hf} + \mathbf{C}_{t-1} \cdot \mathbf{W}_{cf} + \mathbf{b}_f) && (2)\\
\mathbf{C}_t &= \mathbf{F}_t \bullet \mathbf{C}_{t-1} + \mathbf{I}_t \bullet tanh(\mathbf{X}_t \cdot \mathbf{W}_{xc} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{hc} + \mathbf{b}_c) && (3)\\
\mathbf{O}_t &= \sigma(\mathbf{X}_t \cdot \mathbf{W}_{xo} + \mathbf{M}_{t-1} \cdot \mathbf{W}_{ho} + \mathbf{C}_t \cdot \mathbf{W}_{co} + \mathbf{b}_o) && (4)\\
\mathbf{M}_t &= \mathbf{O}_t \bullet tanh(\mathbf{C}_t) && (5)
\end{aligned}
$$

where (σ) denotes the sigmoid activation function, $(\cdot)$ indicates matrix multiplication and $(\bullet)$ indicates component-wise multiplication. [1]

Details

Let $\mathbf{q} \in \mathbb{R}^D$ denote an image feature vector

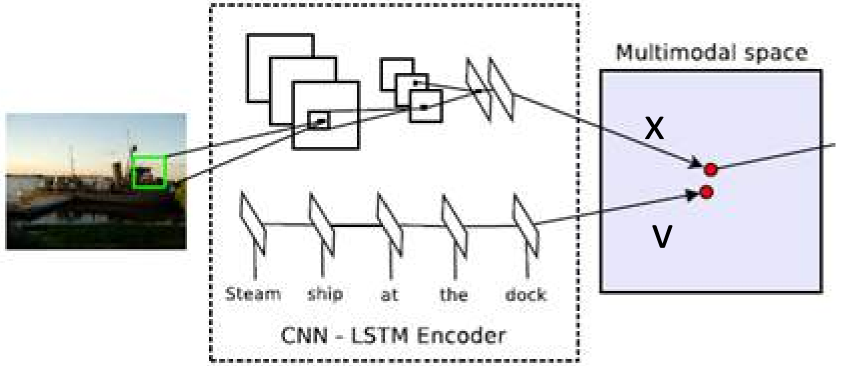
- D: length of the CNN code (CNN can be AlexNet, VggNet, or ResNet)

$\mathbf{x} = \mathbf{W}_I \cdot \mathbf{q} \in \mathbb{R}^K$ be the image embedding

image description $S = \{w_1, \ldots, w_N\}$ with words $w_1, \ldots, w_N$

$\{\mathbf{w}_1, \ldots, \mathbf{w}_N\}, \mathbf{w}_i \in \mathbb{R}^K, i = 1, \ldots, n$ denote the corresponding word representations to words $w_1, \ldots, w_N$ (entries in the matrix $\mathbf{W}_T$). The representation of a sentence $\mathbf{v}$ is the hidden state of the LSTM at time step N (i.e. the vector $\mathbf{m}_t$).

W_T: precomputed using e.g. word2vec



Details

- Optimize pairwise rank loss (θ:parameters needed to be learnt: W_I and LSTM parameters) (similar to negative sampling in spirit)

Vector for image

Vector for sentence

Vector for contrastive sentence

Vector for contrastive image

$$\min_{\boldsymbol{\theta}} \sum_{\mathbf{x}} \sum_{k} \max\{0, \alpha - s(\mathbf{x}, \mathbf{v}) + s(\mathbf{x}, \mathbf{v}_k)\} + \sum_{\mathbf{v}} \sum_{k} \max\{0, \alpha - s(\mathbf{v}, \mathbf{x}) + s(\mathbf{v}, \mathbf{x}_k)\}$$

Max-margin formulation. α margin
We have nonzero loss if
s(x,v) is less than s(x,vk)+\alpha

scoring function $s(\mathbf{x}, \mathbf{v}) = \mathbf{x} \cdot \mathbf{v}$.
$\mathbf{v}_k$ is a contrastive (non-descriptive) sentence for image embedding $\mathbf{x}$, and vice-versa with $\mathbf{x}_k$.