UP AND DOWN FROM WORD EMBEDDINGS

Natural Language Processing

Sergey Nikolenko

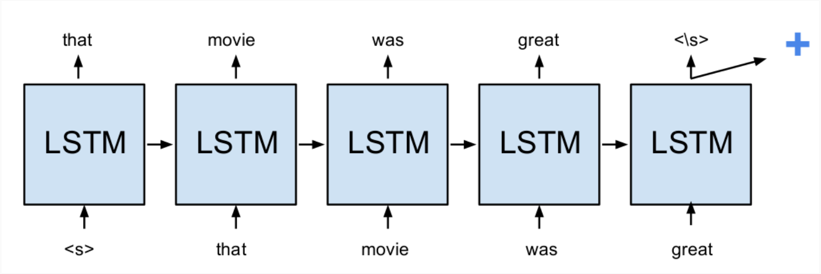
Harbour Space University, Barcelona, Spain

January 16, 2018

FROM WORD VECTORS TO PARAGRAPH VECTORS

HOW TO USE WORD VECTORS

- Next we can use recurrent architectures on top of word vectors.
- E.g., LSTMs for sentiment analysis:



- Train a network of LSTMs for language modeling, then use either the last output or averaged hidden states for sentiment.
- We will see a lot of other architectures later.

UP AND DOWN FROM WORD EMBEDDINGS

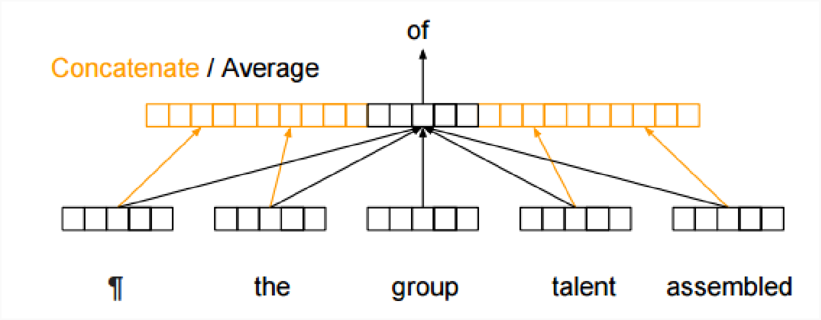
- Word embeddings are the first step of most DL models in NLP.
- But we can go both up and down from word embeddings.
- First, a sentence is not necessarily the sum of its words.
- Second, a word is not quite as atomic as the word2vec model would like to think.

SENTENCE EMBEDDINGS

- How do we combine word vectors into “text chunk” vectors?
- The simplest idea is to use the sum and/or mean of word embeddings to represent a sentence/paragraph:
    - a baseline in (Le and Mikolov 2014);
    - a reasonable method for short phrases in (Mikolov et al. 2013)
    - shown to be effective for document summarization in (Kageback et al. 2014).

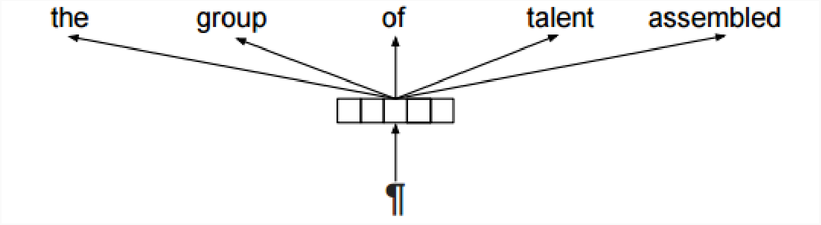
SENTENCE EMBEDDINGS

- How do we combine word vectors into “text chunk” vectors?
- Distributed Memory Model of Paragraph Vectors (PV-DM) (Le and Mikolov 2014):
    - a sentence/paragraph vector is an additional vector for each paragraph;
    - acts as a “memory” to provide longer context;



SENTENCE EMBEDDINGS

- How do we combine word vectors into “text chunk” vectors?
- Distributed Bag of Words Model of Paragraph Vectors (PV-DBOW) (Le and Mikolov 2014):
  - the model is forced to predict words randomly sampled from a specific paragraph;
  - the paragraph vector is trained to help predict words from the same paragraph in a small window.

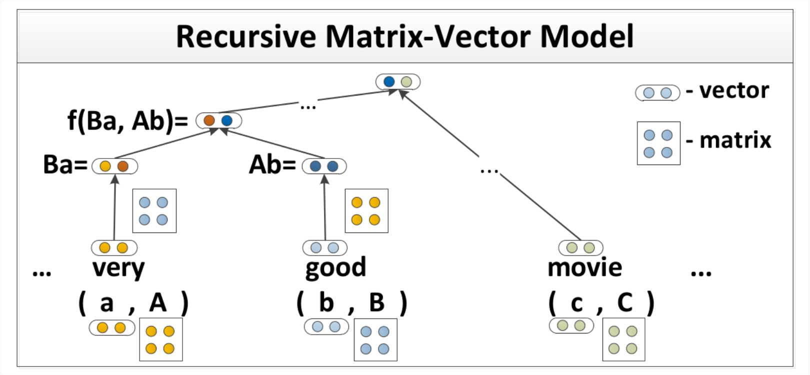


SENTENCE EMBEDDINGS

- How do we combine word vectors into “text chunk” vectors?
- A number of convolutional architectures (Ma et al., 2015; Kalchbrenner et al., 2014).
- (Kiros et al. 2015): skip-thought vectors capture the meanings of a sentence by training from skip-grams constructed on sentences.
- (Djuric et al. 2015): model large text streams with hierarchical neural language models with a document level and a token level.

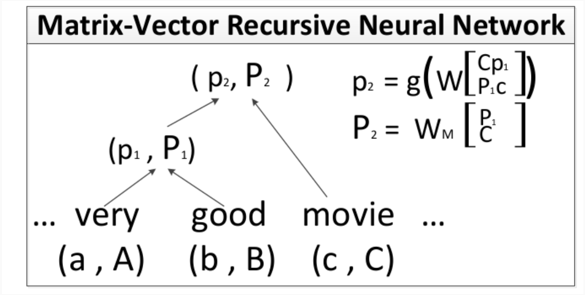
SENTENCE EMBEDDINGS

- How do we combine word vectors into “text chunk” vectors?
- *Recursive neural networks* (Socher et al., 2012):
    - a neural network composes a chunk of text with another part in a tree;
    - works its way up from word vectors to the root of a parse tree.



SENTENCE EMBEDDINGS

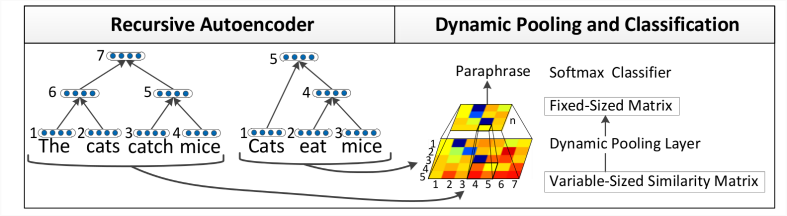
- How do we combine word vectors into “text chunk” vectors?
- *Recursive neural networks* (Socher et al., 2012):
  - by training this in a supervised way, one can get a very effective approach to sentiment analysis (Socher et al. 2013).



Matrix-Vector Recursive Neural Network
(p_2, P_2)
$p_2 = g\left(W\begin{bmatrix} Cp_1 \\ P_1c \end{bmatrix}\right)$
$P_2 = W_M \begin{bmatrix} P_1 \\ C \end{bmatrix}$
(p_1, P_1)
... very good movie ...
(a, A) (b, B) (c, C)

SENTENCE EMBEDDINGS

- How do we combine word vectors into “text chunk” vectors?
- A similar effect can be achieved with CNNs.
- *Unfolding Recursive Auto-Encoder* model (URAE) (Socher et al., 2011) collapses all word embeddings into a single vector following the parse tree and then reconstructs back the original sentence; applied to paraphrasing and paraphrase detection.



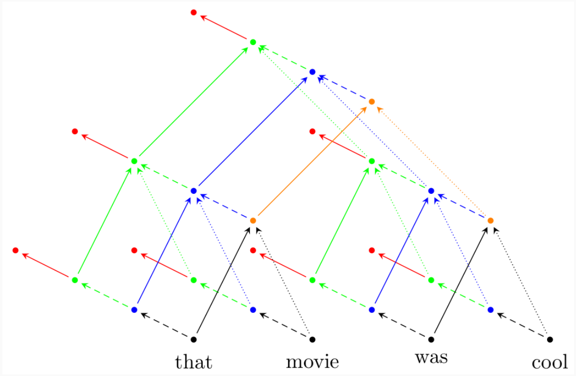
DEEP RECURSIVE NETWORKS

- Deep recursive networks for sentiment analysis (Irsoy, Cardie, 2014).
- First idea: decouple leaves and internal nodes.
- In recursive networks, we apply the same weights throughout the tree:
$$\mathbf{x}_v = f(W_L \mathbf{x}_{l(v)} + W_R \mathbf{x}_{r(v)} + \mathbf{b}).$$
- Now, we use different matrices for leaves (input words) and hidden nodes:
    - we can now have fewer hidden units than the word vector dimension;
    - we can use ReLU: sparse inputs and dense hidden units do not cause a discrepancy.

DEEP RECURSIVE NETWORKS

- Second idea: add depth to get hierarchical representations:

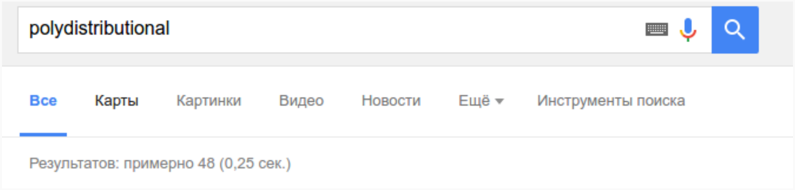
$$h_v^{(i)} = f(W_L^{(i)} h_{l(v)}^{(i)} + W_R^{(i)} h_{r(v)}^{(i)} + V^{(i)} h_v^{(i-1)} + \mathbf{b}^{(i)}).$$



- An excellent architecture for sentiment analysis... if you have the parse trees.

CHARACTER-LEVEL MODELS

- Word embeddings have important shortcomings:
  - vectors are independent but words are not; consider, in particular, morphology-rich languages like Russian/Ukrainian;
  - the same applies to out-of-vocabulary words: a word embedding cannot be extended to new words;
  - word embedding models may grow large; it's just lookup, but the whole vocabulary has to be stored in memory with fast access.
- E.g., "polydistributional" gets 48 results on Google, so you probably have never seen it, and there's very little training data:


polydistributional

Результатов: примерно 48 (0,25 сек.)

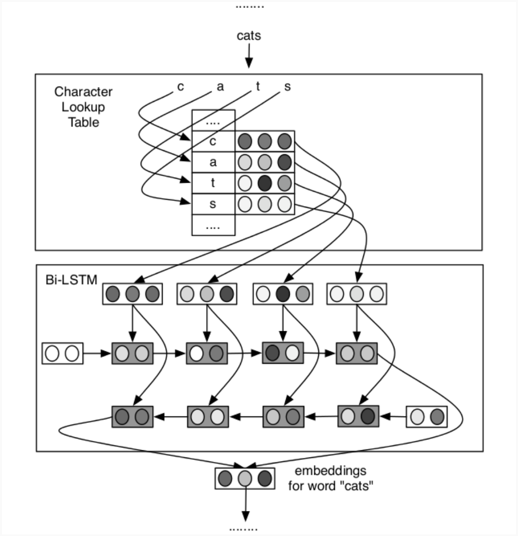
- Do you have an idea what it means? Me too.

CHARACTER-LEVEL MODELS

- Hence, *character-level representations*:
    - began by decomposing a word into morphemes (Luong et al. 2013; Botha and Blunsom 2014; Soricut and Och 2015);
    - but this adds errors since morphological analyzers are also imperfect, and basically a part of the problem simply shifts to training a morphology model;
    - two natural approaches on character level: LSTMs and CNNs;
    - in any case, the model is slow but we do not have to apply it to every word, we can store embeddings of common words in a lookup table as before and only run the model for rare words – a nice natural tradeoff.

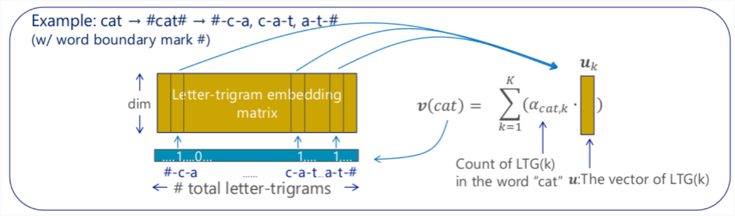
CHARACTER-LEVEL MODELS

- C2W (Ling et al. 2015) is based on bidirectional LSTMs:



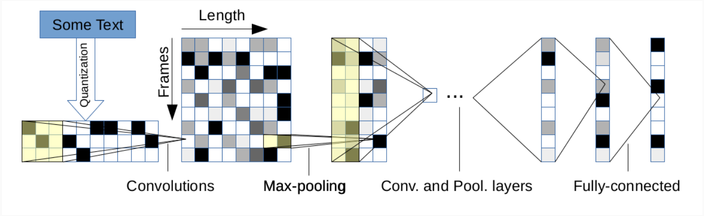
CHARACTER-LEVEL MODELS

- The approach of *Deep Structured Semantic Model* (DSSM) (Huang et al., 2013; Gao et al., 2014a; 2014b):
  - sub-word embeddings: represent a word as a bag of trigrams;
  - vocabulary shrinks to $|V|^3$ (tens of thousands instead of millions), but collisions are very rare;
  - the representation is robust to misspellings (very important for user-generated texts).


Example: cat → #cat# → #-c-a, c-a-t, a-t-#
(w/ word boundary mark #)
dim
Letter-trigram embedding matrix
....1....0...
1....
1....
#-c-a
......
c-a-t
a-t-#
← # total letter-trigrams →
u_k
$v(cat) = \sum_{k=1}^{K} (\alpha_{cat,k} \cdot \quad)$
Count of LTG(k) in the word "cat"
u:The vector of LTG(k)

CHARACTER-LEVEL MODELS

- ConvNet (Zhang et al. 2015): text understanding from scratch, from the level of symbols, based on CNNs.



- Character-level models and extensions to appear to be very important, especially for morphology-rich languages like Russian or Spanish.

FASTTEXT

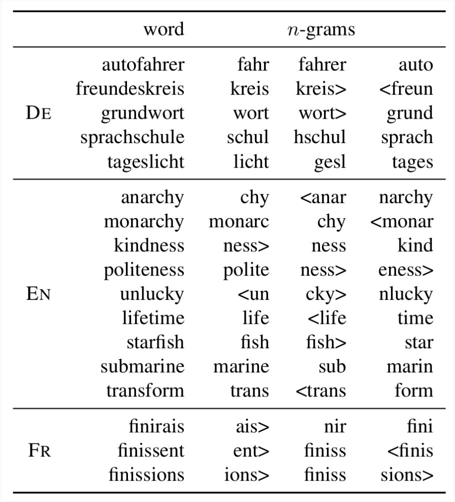
- FastText – character-based word vectors.
- Represent words as n-grams of characters:
  - a word is a bag of n-grams plus the word itself:

$$\text{\#mo}, \text{mod}, \text{ode}, \text{del}, \text{el\#}, \text{\#model\#}$$

  - the word vector is the sum of the vectors of its components: n-grams from $n = 3$ to $n = 6$;
  - and then it's just regular skip-gram on top of these words.
- Significant improvements for many tasks like word analogy.

FASTTEXT

- Most important n-grams:



	word	n-grams		
DE	autofahrer	fahr	fahrer	auto
	freundeskreis	kreis	kreis>	<freun
	grundwort	wort	wort>	grund
	sprachschule	schul	hschul	sprach
	tageslicht	licht	gesl	tages
EN	anarchy	chy	<anar	narchy
	monarchy	monarc	chy	<monar
	kindness	ness>	ness	kind
	politeness	polite	ness>	eness>
	unlucky	<un	cky>	nlucky
	lifetime	life	<life	time
	starfish	fish	fish>	star
	submarine	marine	sub	marin
	transform	trans	<trans	form
FR	finirais	ais>	nir	fini
	finissent	ent>	finiss	<finis
	finissions	ions>	finiss	sions>

THANK YOU!

Thank you for your attention!