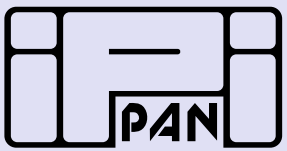
Ewaluacja polskich wektorów dystrybucyjnych w kontekście dezambiguacji morfoskładniowej i parsowania zależnościowego

Katarzyna Krasnowska-Kieraś
Piotr Rybak
Alina Wróblewska



INSTITUTE OF COMPUTER SCIENCE
POLISH ACADEMY OF SCIENCES
ul. Jana Kazimierza 5, 01-248 Warszawa

Warszawa, 23 października 2017

Spis treści

1. Wprowadzenie
2. Modele embeddingowe cech
3. Metodologia ewaluacji
4. Toygger: dezambiguacja morfoskładniowa
5. Parsowanie zależnościowe
6. Podsumowanie

Sieci neuronowe

- Głębokie sieci neuronowe to klasa algorytmów uczenia maszynowego z nadzorem, które:
  - uczą się na podstawie obserwacji,
  - uczą się reprezentować dane.
- Ważnym komponentem sieci neuronowych jest warstwa mapująca symbole (∼cechy) na gęste wektory w stosunkowo niskowymiarowej przestrzeni (tzw. *embedding layer*).
- *Feature embedding* to gęsty wektor cech danego typu.
- *Cecha* to element lingwistyczny:
  - forma słowa (np. segment, lemat),
  - część słowa (np. rdzeń, przyrostek),
  - część mowy,
  - znaczniki morfologiczne.

Wektory dystrybucyjne cech

- Różne typy cech są reprezentowane przez wektory z różną liczbą wymiarów:
  - formy lub części wyrazów – więcej wymiarów,
  - części mowy – mniej wymiarów.
- Cechy reprezentowane jako d-wymiarowe wektory są osadzone (ang. 'embedded') w d-wymiarowej przestrzeni.
- Cechy elementarne vs. cechy złożone.
- Zalety i wady reprezentacji wektorowych cech wyższego rzędu.

Koncepcja embeddingów dla cech

- Cechy współdzielące informacje powinny mieć podobne wektory.
- Przykład – gramatyczny aspekt dokonany i niedokonany
  - ZJADAĆ VS. ZJEŚĆ
  - Czasowniki z różnymi kategoriami aspektu mają podobną walencję i preferencje selekcyjne.
  - Czasowniki powinny być reprezentowane za pomocą podobnych wektorów.

Embeddingi cech w różnych językach

- Język angielski (izolacyjny) – modele embeddingowe są liczone i ewaluowane na segmentach (Mikolov et al., 2013, Pennington et al., 2014).
- Język rosyjski (fleksyjny) – modele embeddingowe:
  - liczone na rdzeniach (Leviant i Reichart, 2015) lub segmentach uproszczonych za pomocą reguł specyficznych dla rosyjskiego (Vulić et al., 2017),
  - ewaluowane (*in vitro*) na zbiorze testowym złożonym z par lematów z przypisanymi wagami podobieństwa/powiązania.
- Język polski (fleksyjny) – modele embeddingowe:
  - liczone na segmentach i lematach (Mykowiecka et al., 2017)
  - ewaluowane (*in vitro*) na parach lematów z dostępnych słowników semantycznych i zbiorze analogii (segmenty są automatycznie lematyzowane).

Problemy badawcze

- Jakie typy cech należy brać pod uwagę w liczeniu modeli embeddingowych języków fleksyjnych?
- Czy wyniki ewaluacji *in vitro* zawierają informacje niezbędne dla dalszych zastosowań?
- Czy modele embeddingowe powinny być ewaluowane *in vivo* w rzeczywistych zastosowaniach?

Spis treści

1. Wprowadzenie
2. Modele embeddingowe cech
3. Metodologia ewaluacji
4. Toygger: dezambiguacja morfoskładniowa
5. Parsowanie zależnościowe
6. Podsumowanie

Reprezentacje wektorowe segmentów

- Nazwa angielska: *token embedding model*.
- Oznaczenie: $\mathcal{T}$.
- Wymagania: duży zbiór tekstów i tokenizator.
- Wektor w przestrzeni embeddingowej (wektorowej) generowany jest dla każdego segmentu.
- Segmenty odpowiadające wszystkim synkretycznym formom fleksyjnym danego lematu, jak również segmenty homonimiczne różnych lematów mają taką samą reprezentację wektorową.

Przykłady (1)

$\mathcal{T}$
pociąg
pociągu
pociągowi
pociągiem
pociągi
pociągów
pociągom
pociągami
pociągach

Reprezentacje wektorowe lematów

- Nazwa angielska: *lemma embedding model*.
- Oznaczenie: $\mathcal{L}$.
- Wymagania: duży zlematyzowany korpus lub lematyzator
- Wektor generowany dla każdego lematu.
- Wszystkie segmenty należące do jednego lematu mają tę samą reprezentację wektorową.
- Jakość modelu zależy od jakości anotacji w korpusie lub lematyzatora.

Przykłady (2)

$\mathcal{T}$	$\mathcal{L}$
pociąg	POCIĄG
pociągu	
pociągowi	
pociągiem	
pociągi	
pociągów	
pociągom	
pociągami	
pociągach	

Reprezentacje wektorowe form fleksyjnych

- Nazwa angielska: *inflection embedding model*.
- Oznaczenie: $\mathcal{W}$.
- Wymagania: korpus zaanotowany morfoskładniowo lub tagger.
- Wektor generowany dla każdego segmentu połączonego z interpretacją morfoskładniową.
- Proces tworzenia tego typu wektorów jest najbardziej kosztowny i najbardziej podatny na propagowanie błędów.

Przykłady (3)

$\mathcal{T}$	$\mathcal{L}$	$\mathcal{W}$
pociąg	POCIĄG	*pociąg*.subst:sg:nom:m3
pociągu		*pociąg*.subst:sg:acc:m3
pociągowi		*pociągu*.subst:sg:gen:m3
pociągiem		*pociągu*.subst:sg:loc:m3
pociągi		*pociągu*.subst:sg:voc:m3
pociągów		*pociągowi*.subst:sg:dat:m3
pociągom		*pociągiem*.subst:sg:inst:m3
pociągami		*pociągi*.subst:pl:nom:m3
pociągach		*pociągi*.subst:pl:acc:m3
		pociągi.subst:pl:voc:m3
		pociągów.subst:pl:gen:m3
		pociągom.subst:pl:dat:m3
		pociągami.subst:pl:inst:m3
		pociągach.subst:pl:loc:m3

Reprezentacje wektorowe *n*-gramów znaków

- Nazwa angielska: *subword embedding model* lub *character n-gram embedding model*.
- Oznaczenie: $\mathcal{S}$.
- Wymagania: duży zbiór tekstów i tokenizator.
- Segmenty są reprezentowane jako ‚worki' *n*-gramów znaków.
- Segmenty odpowiadające wszystkim synkretycznym formom fleksyjnym danego lematu, jak również segmenty homonimiczne różnych lematów mają taką samą reprezentację wektorową.
- Można stworzyć reprezentację wektorową nieznanych segmentów, uśredniając wektory dla ich *n*-gramów.

Przykłady (4)

$\mathcal{T}$	$\mathcal{L}$	$\mathcal{W}$	$\mathcal{S}$ (dla $n = 5$)
pociąg	POCIĄG	*pociąg*.subst:sg:nom:m3	<*poci*+*pocią*+*ociąg*+*ciąg*>+<*pociąg*>
pociągu		*pociąg*.subst:sg:acc:m3	<*poci*+...+*iągu*>+<*pociągu*>
pociągowi		*pociągu*.subst:sg:gen:m3	<*poci*+...+*gowi*>+<*pociągowi*>
pociągiem		*pociągu*.subst:sg:loc:m3	<*poci*+...+*giem*>+<*pociągiem*>
pociągi		*pociągu*.subst:sg:voc:m3	<*poci*+...+*iągi*>+<*pociągi*>
pociągów		*pociągowi*.subst:sg:dat:m3	<*poci*+...+*ągów*>+<*pociągów*>
pociągom		*pociągiem*.subst:sg:inst:m3	<*poci*+...+*ągom*>+<*pociągom*>
pociągami		*pociągi*.subst:pl:nom:m3	<*poci*+...+*gami*>+<*pociągami*>
pociągach		*pociągi*.subst:pl:acc:m3	<*poci*+...+*gach*>+<*pociągach*>
		pociągi.subst:pl:voc:m3	
		pociągów.subst:pl:gen:m3	
		pociągom.subst:pl:dat:m3	
		pociągami.subst:pl:inst:m3	
		pociągach.subst:pl:loc:m3	

Technikalia

- Testujemy $\mathcal{T}$, $\mathcal{L}$ i $\mathcal{S}$.
- Modele są liczone na tym samym zbiorze tekstów z NKJP i Wikipedii.
- Wielkość słownika:
  - Słownik $\mathcal{T}$ – 2.12M
  - Słownik $\mathcal{L}$ – 1.55M
  - Słownik $\mathcal{S}$ – 5M

Trenowanie modeli $\mathcal{T}$ i $\mathcal{L}$

- Testujemy wektory $\mathcal{T}_d$ i $\mathcal{L}_d$[1], dla d będącego wielkością wektoru (Mykowiecka et al., 2017).
- Wektory były liczone za pomocą biblioteki *Gensim*[2] (Řehůřek i Sojka, 2010).
- Parametry słownika:
  - przycinanie słów z frekwencją mniejszą niż 5 w danych treningowych.
- Parametry trenowania:
  - wielkość wektorów: $d \in \{10, 25, 50, 100, 200, 300\}$
  - wielkość okna: 5
  - liczba iteracji (ang. *epochs*): 10
  - architektura sieci: CBOW i skip-gram
  - tryb uczenia: hierarchical softmax i negative sampling
  - liczba negatywnych przykładów: 5

[1] `dsmodels.nlp.ipipan.waw.pl`

[2] `https://radimrehurek.com/gensim`

Trenowanie modeli $\mathcal{S}$

- Wektory $\mathcal{S}_d$ były liczone za pomocą biblioteki *fastText*[3] (Bojanowski et al., 2016).
- Parametry słownika:
  - przycinanie słów z frekwencją mniejszą niż 5,
  - n-gramy znaków dla $3 \leqslant n \leqslant 6$
- Parametry trenowania:
  - wielkość wektorów: $d = 100$ lub $d = 300$
  - wielkość okna: 5
  - liczba iteracji: 10
  - architektura sieci: skip-gram
  - tryb uczenia: negative sampling
  - liczba negatywnych przykładów: 5

[3] `https://github.com/facebookresearch/fastText`

Spis treści

1. Wprowadzenie
2. Modele embeddingowe cech
3. Metodologia ewaluacji
4. Toygger: dezambiguacja morfoskładniowa
5. Parsowanie zależnościowe
6. Podsumowanie

Ewaluacja *in vitro*

- Angielska nazwa: *intrinsic evaluation*
- Idea: testowanie modelu w oparciu o zbiór testowy.
- Określenie podobieństwa/powiązania:
  - obliczenie bliskości semantycznej dwóch słów (np. cosinus kąta) i skorelowanie tego wyniku z oceną podobieństwa dokonaną przez człowieka,
  - metodologia testowania podobieństwa jest obecnie krytykowana (np. Faruqui et al., 2016, Chiu et al., 2016).
- Szukanie analogii:
  - wyszukanie wektora jednego słowa (np. *królowa*) na podstawie wektorów pozostałych słów (np. *król* - *mężczyzna* + *kobieta*),
  - metodologia tego testu jest również krytykowana (np. Drozd et al., 2016).

Ewaluacja *in vivo*

- Angielska nazwa: *extrinsic evaluation*.
- Idea: testowanie modelu poprzez jego integrację z systemem NLP i porównanie wyników przed i po dodaniu embeddingów.
- Dwa zadania:
  - dezambiguacja morfoskładniowa,
  - parsowanie zależnościowe.

Spis treści

1 Wprowadzenie

2 Modele embeddingowe cech

3 Metodologia ewaluacji

4 Toygger: dezambiguacja morfoskładniowa

5 Parsowanie zależnościowe

6 Podsumowanie

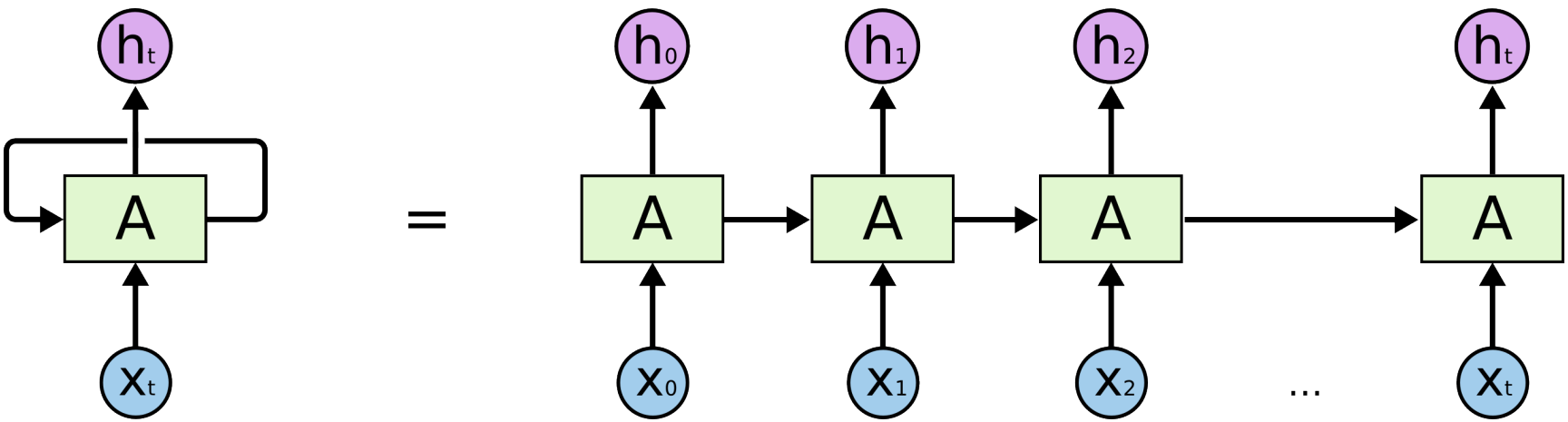
Zadanie

Dane: ciąg segmentów (tokenów); każdy segment opatrzony zbiorem możliwych znaczników morfoskładniowych albo specjalnym symbolem `ign`.

Wynik: przypisanie każdemu segmentowi wejściowemu dokładnie jednego znacznika (wybór spośród możliwych znaczników jeśli są dostępne, wygenerowanie znacznika dla segmentów ‚`ign`').

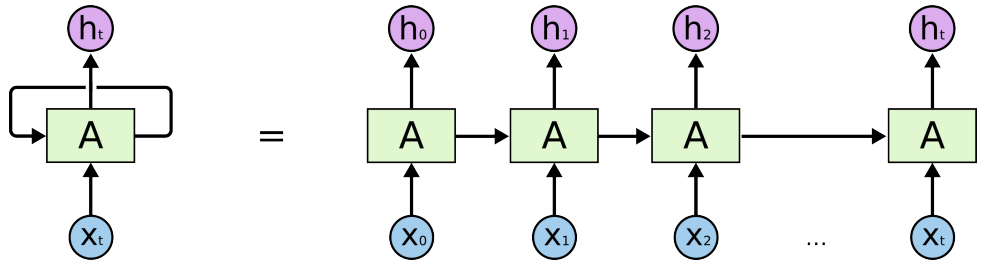
Nawracająca sieć neuronowa

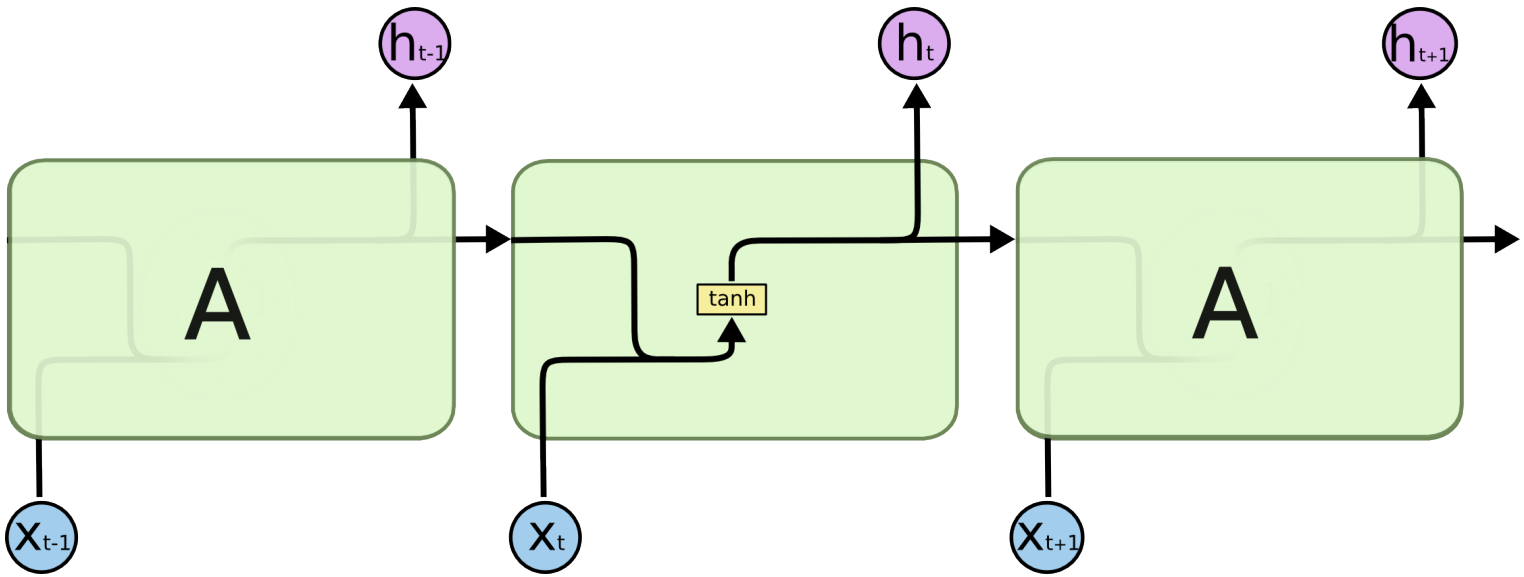
- ang. ***recurrent*** *neural network*,
- sieć operująca na danych o strukturze sekwencyjnej,
- $\neq$ ***recursive*** *neural network* (dane o strukturze rekurencyjnej, np. drzewa),
- skrót (niejednoznaczny): RNN.



(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Nawracająca sieć neuronowa

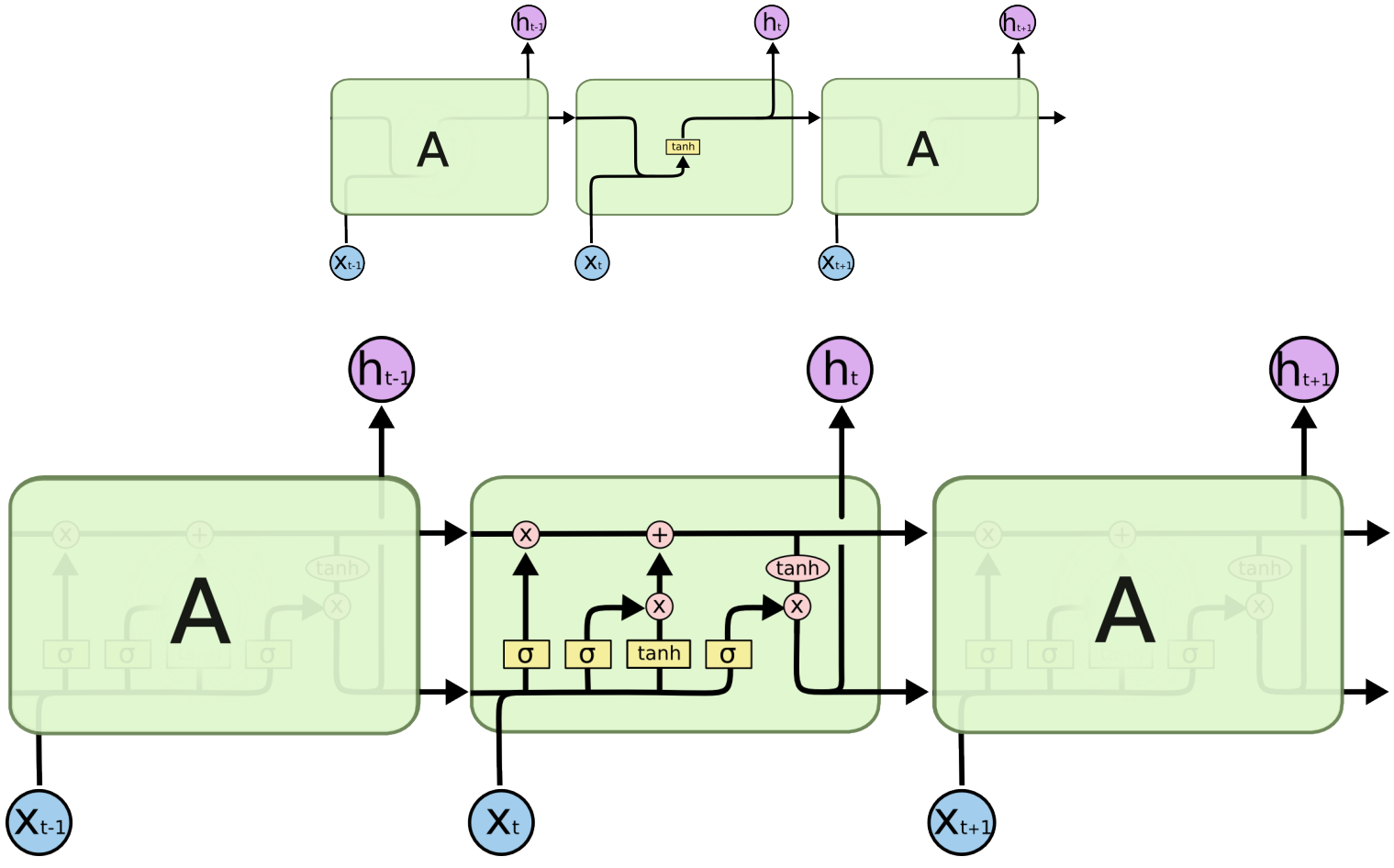




(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

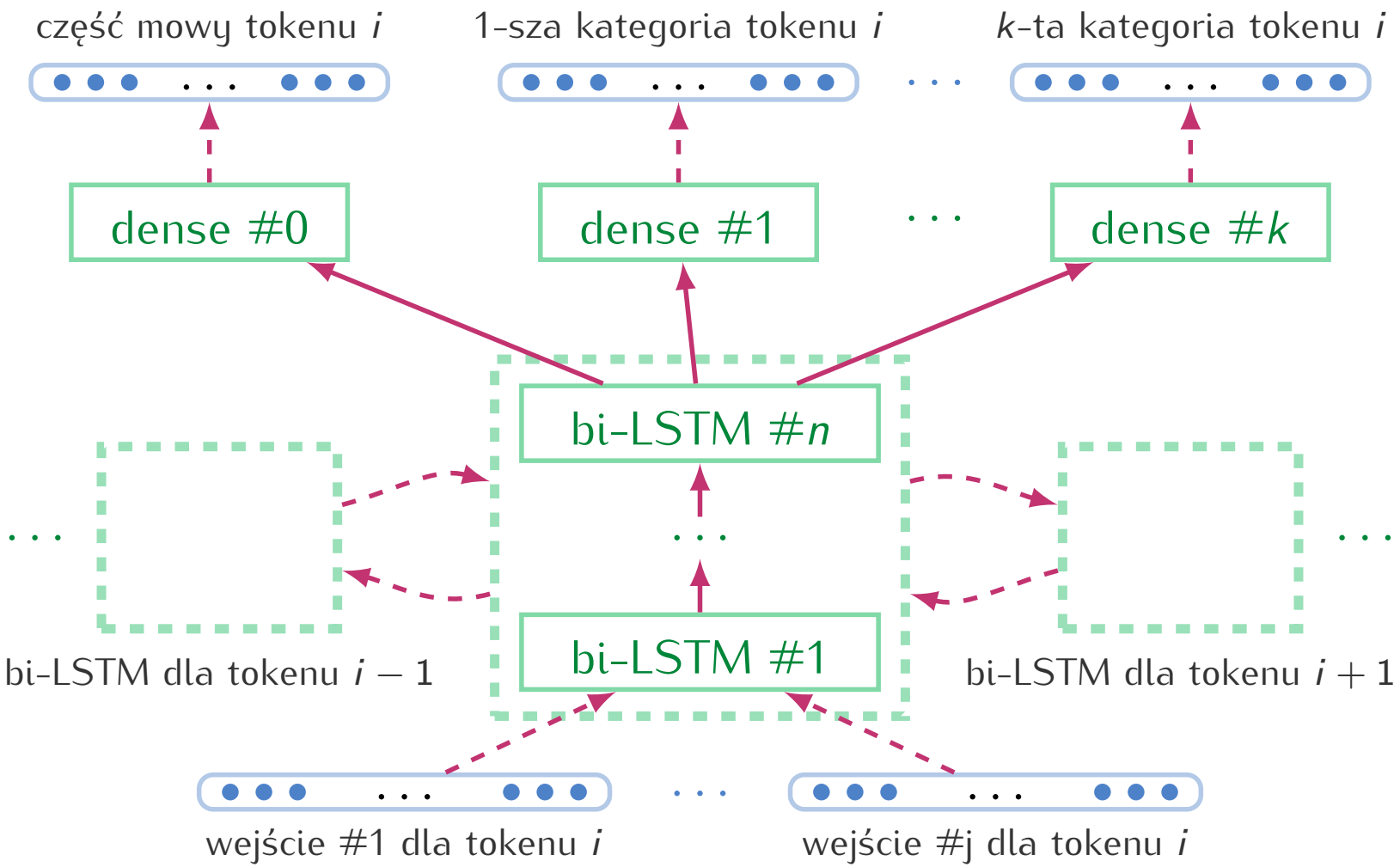
Komórka LSTM (Hochreiter i Schmidhuber, 1997)

- ang. *Long Short Term Memory*,
- ma zapobiegać „zapominaniu” odległych informacji.



(http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Schemat sieci



Implementacja

- Python 2.7
- Keras (`https://keras.io/`)
- TensorFlow (`https://www.tensorflow.org/`)

Reprezentacja danych: morfologia

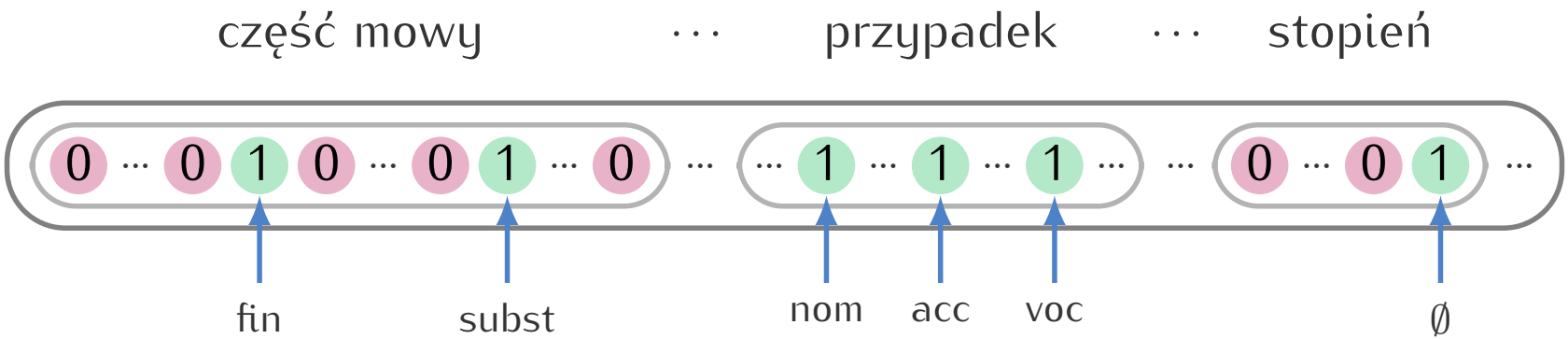
Możliwe znaczniki dla tokenu ***cielę***:

- `subst:sg:nom:n`
- `subst:sg:acc:n`
- `subst:sg:voc:n`
- `fin:sg:pri:imperf`

„Worek znaczników”:

- `subst`, `sg`, `nom`, `acc`, `voc`, `n`, `fin`, `pri`, `imperf`

Wektor morfologiczny:



Reprezentacja danych: wektory słów

Wektory z modeli $\mathcal{T}$ lub $\mathcal{S}$ z heurystyką dla słów nieznanych modelowi:

- znajdź w słownictwie modelu zbiór S_w słów dzielących najdłuższy (niepusty) sufiks z nieznanym słowem w,
- jeśli $S_w \neq \emptyset$, wybierz zbiór $S'_w \subseteq S_w$ słów o najmniejszej odległości Levenshteina od w; średnia wektorów słów z S'_w jest przybliżeniem wektora dla w,
- jeśli $S_t = \emptyset$, użyj losowego wektora.

Reprezentacja danych: wektory słów

Wektory z modeli $\mathcal{T}$ lub $\mathcal{S}$ z heurystyką dla słów nieznanych modelowi:

- znajdź w słownictwie modelu zbiór S_w słów dzielących najdłuższy (niepusty) sufiks z nieznanym słowem w,
- jeśli $S_w \neq \emptyset$, wybierz zbiór $S'_w \subseteq S_w$ słów o najmniejszej odległości Levenshteina od w; średnia wektorów słów z S'_w jest przybliżeniem wektora dla w,
- jeśli $S_t = \emptyset$, użyj losowego wektora.

Przykład:

- słowo ***Kotkowice*** nieznane modelowi,
- w modelu: *Motkowice*, *Gotkowice*, *Młotkowice*, *Katowice*,
- wynik: średnia wektorów dla *Motkowice* i *Gotkowice*.

Reprezentacja danych: wektory końcówek

Dane uczące PolEval[4] użyte w eksperymentach:

- 143473 różnych tokenów; 8,5 wystąpień na token,
- 25931 sufiksów 4-literowych; 46,9 wystąpień na sufiks.

Pomysł: wytrenować embeddingi dla sufiksów razem z modelem do dezambiguacji.

Potencjalne zalety:

- mniejszy model embeddingów,
- niepotrzebne duże dane uczące,
- embeddingi dostosowane do zadania,
- porównanie z wynikami dla „dużych” modeli.

[4] `http://poleval.pl`

Wyjścia sieci

- Dla każdego tokenu sieć produkuje $k+1$ wektorów odpowiadających:
  - części mowy,
  - 1-szej kategorii gramatycznej tagsetu,
  - ...
  - k-tej kategorii gramatycznej tagsetu.
- Warstwy gęste produkujące wektory korzystają z funkcji aktywacji *softmax*:
  - $(x_1, x_2, ..., x_n) \mapsto \left(\frac{e^{x_1}}{\sum_{i=1}^{n} e^{x_i}}, \frac{e^{x_2}}{\sum_{i=1}^{n} e^{x_i}}, ..., \frac{e^{x_n}}{\sum_{i=1}^{n} e^{x_i}} \right)$
  - można traktować jako rozkład prawdopodobieństwa.
- Wyjście wzorcowe: wektory 'one-hot' (rozkłady jednopunktowe) odpowiadające właściwemu znacznikowi.
- Funkcja straty – entropia krzyżowa:
  - $H(p, q) = -\sum_x p(x) \log q(x)$,
  - p – wektor wzorcowy, q – wyjście sieci.

Wybór znacznika

- Wartości (inne niż $\emptyset$) o najwyższych p-stwach w poszczególnych wektorach wyjściowych połączone dwukropkiem.
- Wynik może:
  - nie należeć do możliwych znaczników danego segmentu,
  - w ogóle nie należeć do tagsetu.
- Ostateczna odpowiedź to znacznik najbardziej podobny do wyniku spośród:
  - możliwych znaczników segmentu, jeśli są dostępne,
  - całego tagsetu dla `ign`-ów.

Podobieństwo znaczników

- Intuicja: „normalizacja” podobieństw tekstowych między pozycjami znaczników.
- Mapowanie poszczególnych wartości na dwuznakowe napisy:
  - pierwszy znak wspólny dla wszystkich wartości tej samej kategorii,
  - drugi znak inny dla każdej wartości,
  - np. `nom` $\mapsto$ `T#`, `acc` $\mapsto$ `T$`, `pos` $\mapsto$ `!c`, `com` $\mapsto$ `!v`.
- Odległość Levensteina między tak przekształconymi znacznikami.

Wyniki

reprezentacja danych	acc_G	acc_D	acc	e.r.	Δ
tylko morfologia	38.73%	89.23%	91.59%	0.0%	2.6
+RANDOM$_{100}$	52.80%	90.95%	93.11%	18.1%	2.8
+SUFFIX$_3$	62.84%	91.06%	93.59%	23.8%	3.7
+SUFFIX$_5$	60.51%	91.65%	93.81%	26.5%	4.9
+SUFFIX$_4$	62.97%	91.54%	93.85%	26.9%	4.3
+$\mathcal{S}_{100}$–SKIPG–NS	64.21%	92.38%	94.36%	33.0%	2.5
+$\mathcal{T}_{100}$–SKIPG–HS	64.18%	92.71%	94.54%	35.2%	2.6
+$\mathcal{T}_{100}$–SKIPG–NS	65.72%	93.01%	94.77%	37.9%	2.5
+$\mathcal{S}_{300}$–SKIPG–NS	66.97%	92.98%	94.81%	38.3%	2.7
+$\mathcal{T}_{100}$–CBOW–NS	65.91%	93.23%	94.90%	39.4%	**1.9**
+$\mathcal{T}_{300}$–SKIPG–HS	66.85%	93.28%	94.97%	40.2%	2.7
+$\mathcal{T}_{100}$–CBOW–HS	66.77%	93.36%	95.01%	40.6%	2.2
+$\mathcal{T}_{300}$–CBOW–NS	67.54%	93.55%	95.14%	42.2%	2.2
+$\mathcal{T}_{300}$–SKIPG–NS	**68.18%**	93.53%	95.15%	42.4%	2.7
+$\mathcal{T}_{300}$–CBOW–HS	68.14%	**93.66%**	**95.22%**	43.2%	2.4

acc_G – dokł. dla `ign`, acc_D – dokł. dla segmentów nietrywialnych, **e.r.** – redukcja błędu, Δ – przeuczenie

Spis treści

1. Wprowadzenie
2. Modele embeddingowe cech
3. Metodologia ewaluacji
4. Toygger: dezambiguacja morfoskładniowa
5. Parsowanie zależnościowe
6. Podsumowanie

Zadanie

Dane: sekwencja segmentów (zdanie); każdy z przypisaną częścią mowy, lematem i charakterystyką morfologiczną.

Wynik: przypisanie każdemu segmentowi wejściowemu jego rodzica w drzewie zależnościowym (czyli innego segmentu ze zdania albo korzenia ROOT) oraz typu zależności.

Parser stosowy (ang. *Transition-based parser*)

Inicjujemy parser:

- Wejście (ang. *Buffer*) = [segment1, segment2, ...]
- Stos (ang. *Stack*) = []

W każdym kroku wykonujemy jedną operację:

- Shift - przenieś segment z wejścia na stos,
- Left-Arc - połącz pierwszy z drugim segmentem ze stosu, wyrzuć drugi ze stosu, zapamiętaj krawędź,
- Right-Arc - połącz drugi z pierwszym segmentem ze stosu, wyrzuć pierwszy ze stosu, zapamiętaj krawędź.

Klasyfikator

- Konwersja danych treningowych do postaci:
  - Cechy opisujące aktualny stan parsera,
  - Optymalna operacja do wykonania.
- Trening dowolnego modelu uczenia maszynowego,
- Klasyfikator wybierający optymalną operację.

MaltParser (Nivre et al., 2016)

Model cech:

- dla N górnych segmentów ze stosu i wejścia:
  - forma,
  - lemat,
  - część mowy,
  - charakterystyka morfologiczna,
  - przewidziane wcześniej typy zależności,
- wybrane interakcje pomiędzy powyższymi cechami,
- wartości embeddingu dla górnych segmentów.

Klasyfikator:

- regresja logistyczna.

SyntaxNet (Andor et al., 2016)

Cechy:

- dla N górnych segmentów ze stosu i wejścia:
  - forma,
  - lemat,
  - część mowy,
  - przewidziane wcześniej typy zależności,

Klasyfikator:

- Dwuwarstwowa sieć neuronowa.

BistParser (Kiperwasser i Goldberg, 2016)

Zmienne:

- BiLSTM po całym zdaniu,
- stan ukryty jako embedding dla formy,
- stan ukryty jako embedding części mowy.

Klasyfikator:

- BiLSTM.

Porównanie

Właściwości parsera	MaltParser	SyntaxNet	BIST
Forma	TAK	TAK	TAK
Lemat	TAK	TAK	NIE
Część mowy	TAK	TAK	TAK
Morfologia	TAK	NIE	NIE
Typy krawędzi	TAK	TAK	NIE
Ręczne interakcje	TAK	NIE	NIE
Model liniowy	TAK	NIE	NIE

parser	LAS		UAS	
	macro	micro	macro	micro
Malt	81.56%	77.78%	86.07%	82.56%
SyntaxNet	81.64%	77.91%	87.55%	84.23%
Bist	**83.01%**	**79.70%**	**89.07%**	**85.95%**

MaltParser: wyniki

Malt	LAS		UAS	
	macro	**micro**	**macro**	**micro**
Baseline	81.56%	77.78%	86.07%	82.56%
Base+$\mathcal{T}_{50}$	**82.02%**	**78.32%**	**86.34%**	**82.90%**
Base+$\mathcal{T}_{100}$	81.65%	77.97%	86.08%	82.64%
Base+$\mathcal{T}_{200}$	81.46%	77.78%	85.93%	82.49%
Base+$\mathcal{T}_{300}$	80.75%	76.95%	85.43%	81.88%

BistParser: ewaluacja typu embeddingu

Bist	LAS		UAS	
	macro	**micro**	**macro**	**micro**
$\mathcal{T}_{intern}$	83.01%	79.70%	89.07%	85.95%
$\mathcal{T}_{100}$	**85.70%**	**82.68%**	**90.29%**	**87.52%**
$\mathcal{L}_{intern}$	82.94%	79.85%	89.07%	86.06%
$\mathcal{L}_{100}$	84.24%	81.27%	89.64%	86.78%
$\mathcal{S}_{100}$	85.48%	82.48%	90.07%	87.25%

$\mathcal{X}_{intern}$ – wewnętrzny 100-wymiarowy embedding, $\mathcal{X}_{100}$ – zewnętrzny 100-wymiarowy embedding.

BistParser: ewaluacja wymiaru

Bist	LAS		UAS	
	macro	**micro**	**macro**	**micro**
$\mathcal{T}_{10}$	84.15%	81.02%	89.62%	86.67%
$\mathcal{T}_{25}$	84.86%	81.77%	89.90%	87.03%
$\mathcal{T}_{50}$	85.22%	82.17%	90.02%	87.20%
$\mathcal{T}_{100}$	**85.70%**	**82.68%**	**90.29%**	**87.52%**
$\mathcal{T}_{200}$	85.54%	82.44%	90.10%	87.28%
$\mathcal{T}_{300}$	85.52%	82.46%	90.09%	87.30%

BistParser: ewaluacja architektury i trybu uczenia

Bist $+\mathcal{T}_{100}$	LAS		UAS	
	macro	**micro**	**macro**	**micro**
(cbow-ns)	85.57%	82.49%	90.18%	87.36%
(cbow-hs)	85.45%	82.38%	90.02%	87.21%
(skipg-ns)	**85.70%**	**82.68%**	**90.29%**	**87.52%**
(skipg-hs)	85.25%	82.18%	89.93%	87.14%

Spis treści

1. Wprowadzenie
2. Modele embeddingowe cech
3. Metodologia ewaluacji
4. Toygger: dezambiguacja morfoskładniowa
5. Parsowanie zależnościowe
6. Podsumowanie

Podsumowanie

- Ewaluacja *in vivo*:
  - dezambiguacja morfoskładniowa – embeddingi segmentowe zwiększają dokładność o ok. 4 pp, co prowadzi do zmniejszenia liczby błędów o 43%,
  - parsowanie zależnościowe – embeddingi segmentowe zwiększają jakość parsowania zależnościowego,
  - embeddingi 300-wymiarowe są lepsze w dezambiguacji, a embeddingi 100-wymiarowe w parsowaniu,
  - embeddingi lematów są bezużyteczne w dezambiguacji i są mniej przydatne w parsowaniu,
  - wybór parametrów liczenia embeddingów ma drugorzędne znaczenie.
- Dezambiguator morfologiczny Toygger (Krasnowska-Kieraś, 2017) – zwycięzca zdania 1(A) w pierwszej edycji konkursu PolEval.

Podsumowanie

- Ewaluacja *in vivo*:
  - dezambiguacja morfoskładniowa – embeddingi segmentowe zwiększają dokładność o ok. 4 pp, co prowadzi do zmniejszenia liczby błędów o 43%,
  - parsowanie zależnościowe – embeddingi segmentowe zwiększają jakość parsowania zależnościowego,
  - embeddingi 300-wymiarowe są lepsze w dezambiguacji, a embeddingi 100-wymiarowe w parsowaniu,
  - embeddingi lematów są bezużyteczne w dezambiguacji i są mniej przydatne w parsowaniu,
  - wybór parametrów liczenia embeddingów ma drugorzędne znaczenie.
- Dezambiguator morfologiczny Toygger (Krasnowska-Kieraś, 2017) – zwycięzca zdania 1(A) w pierwszej edycji konkursu PolEval.

Dziękujemy za uwagę!

Podziękowania

Przedstawione badania były finansowane ze środków projektu SONATA 8 pt. „Kompozycyjno-dystrybucyjne modelowanie semantyki języka polskiego” (grant nr 2014/15/D/HS2/03486) z Narodowego Centrum Nauki.

Bibliografia

- Andor, D., C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, 2016. Globally Normalized Transition-Based Neural Networks. In Proceedings of ACL 2016.
- Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov, 2016. Enriching Word Vectors with Subword Information. arXiv preprint arXiv:1607.04606.
- Chen, D. and C. Manning, 2014. A Fast and Accurate Dependency Parser using Neural Networks. In Proceedings of EMNLP 2014.
- Chiu, B., A. Korhonen, and Sampo S. Pyysalo, 2016. Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance. In Proceedings of RepEval 2016.
- Drozd, A., A. Gladkova, and S. Matsuoka, 2016. Word Embeddings, Analogies, and Machine Learning: Beyond King – Man + Woman = Queen. In Proceedings of COLING 2016.
- Faruqui, M., Y. Tsvetkov, P. Rastogi, and C. Dyer, 2016. Problems With Evaluation of Word Embeddings Using Word Similarity Tasks. In Proceedings of RepEval 2016.
- Finkelstein, L., E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin, 2002. Placing Search in Context: The Concept Revisited. ACM Transactions on Information Systems, 20(1):116–131.
- Hill, F., R. Reichart, and A. Korhonen, 2015. SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. Computational Linguistics, 41(4):665–695.
- Iyyer, M., V. Manjunatha, J. Boyd-Graber, and H. Daumé III, 2015. Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In Proceedings of ACL-IJCNLP 2015.
- Hochreiter, S and J. Schmidhuber, 1997. Long Short-term Memory. In Neural computation, 9:1735-80.
- Kiperwasser, E. and Y. Goldberg, 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. Transactions of the Association for Computational Linguistics, 4:313–327.
- Kobyliński, Ł. and W. Kieraś, 2016. Part of speech tagging for Polish: State of the art and future perspectives. In Proceedings of CICLing 2016.

Bibliografia

- Krasnowska-Kieraś, K., 2017. Morphosyntactic disambiguation for Polish with bi-LSTM neural networks. In Proceedings of LTC 2017.
- Leviant, I. and R. Reichart, 2015. Separated by an Uncommon Language: Towards Judgment Language Informed Vector Space Modeling. CoRR, abs/1508.00106.
- Linzen, T., 2016. Issues in evaluating semantic spaces using word analogies. In Proceedings of RepEval 2016.
- Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, 2013a. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NISP 2013.
- Mikolov, T., W. Yih, and G. Zweig, 2013b. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT 2013.
- Mykowiecka, A., M. Marciniak, and P. Rychlik, 2017. Testing word embeddings for Polish. Cognitive Studies. Submitted.
- Nivre, J., J. Hall, and J. Nilsson, 2006. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In Proceedings of LREC 2006.
- Ogrodniczuk, M., Ł. Kobyliński, and A. Wawer, 2017. Results of the poleval 2017 competition: Part-of-speech tagging and sentiment analysis shared tasks. In Proceedings of LTC'17. Submitted.
- Pennington, J., R. Socher, and C. Manning, 2014. GloVe: Global Vectors for Word Representation. In Proceedings of EMNLP 2014.
- Řehůřek, R. and P. Sojka, 2010. Software Framework for Topic Modelling with Large Corpora. In Proceedings of the Workshop on New Challenges for NLP Frameworks.
- Vulić, I., N. Mrkšić, R. Reichart, D. Ó Séaghdha, S. Young, and A. Korhonen, 2017. Morph-fitting: Fine-Tuning Word Vector Spaces with Simple Language-Specific Rules. In Proceedings of ACL 2017.
- Wróblewska, A., 2014. Polish Dependency Parser Trained on an Automatically Induced Dependency Bank. Ph.D. dissertation, ICS PAS, Warsaw.