Filip Graliński

Jak karmić tekstami
sztuczną sieć neuronową?

Część I

Kodowanie wejścia i wyjścia

Wejście *Skoro zostałem zaproszona niemal imiennie... Ja swoje akumulatory ładuję aktualnie geografią;) Karol, czyżbyś został wielbicielem starych filmów? I widzę, a przynajmniej tak mi się wydaje, że Szymborska natchnęła Cię do wierszy o przemijaniu. Jeśli tego tematu w tym wierszu nie ma i ja sam sobie go uroiłem - błagam, nie mów nikomu;)*

Wyjście F

Jak to zakodować wejście i wyjście?

Wejście *Skoro zostałem zaproszona niemal imiennie... Ja swoje akumulatory ładuję aktualnie geografią;) Karol, czyżbyś został wielbicielem starych filmów? I widzę, a przynajmniej tak mi się wydaje, że Szymborska natchnęła Cię do wierszy o przemijaniu. Jeśli tego tematu w tym wierszu nie ma i ja sam sobie go uroiłem - błagam, nie mów nikomu;)*

Wyjście F

Jak to zakodować wejście i wyjście?

- z wyjściem jest prosta sprawa:
  - zakodować klasę M jako 1, a F — jako 0
  - (albo jako 1 i -1, jeśli zamiast odwrotności logitu używamy tangensa hiperbolicznego)
- a wejście (czyli tekst do zaklasyfikowania???)

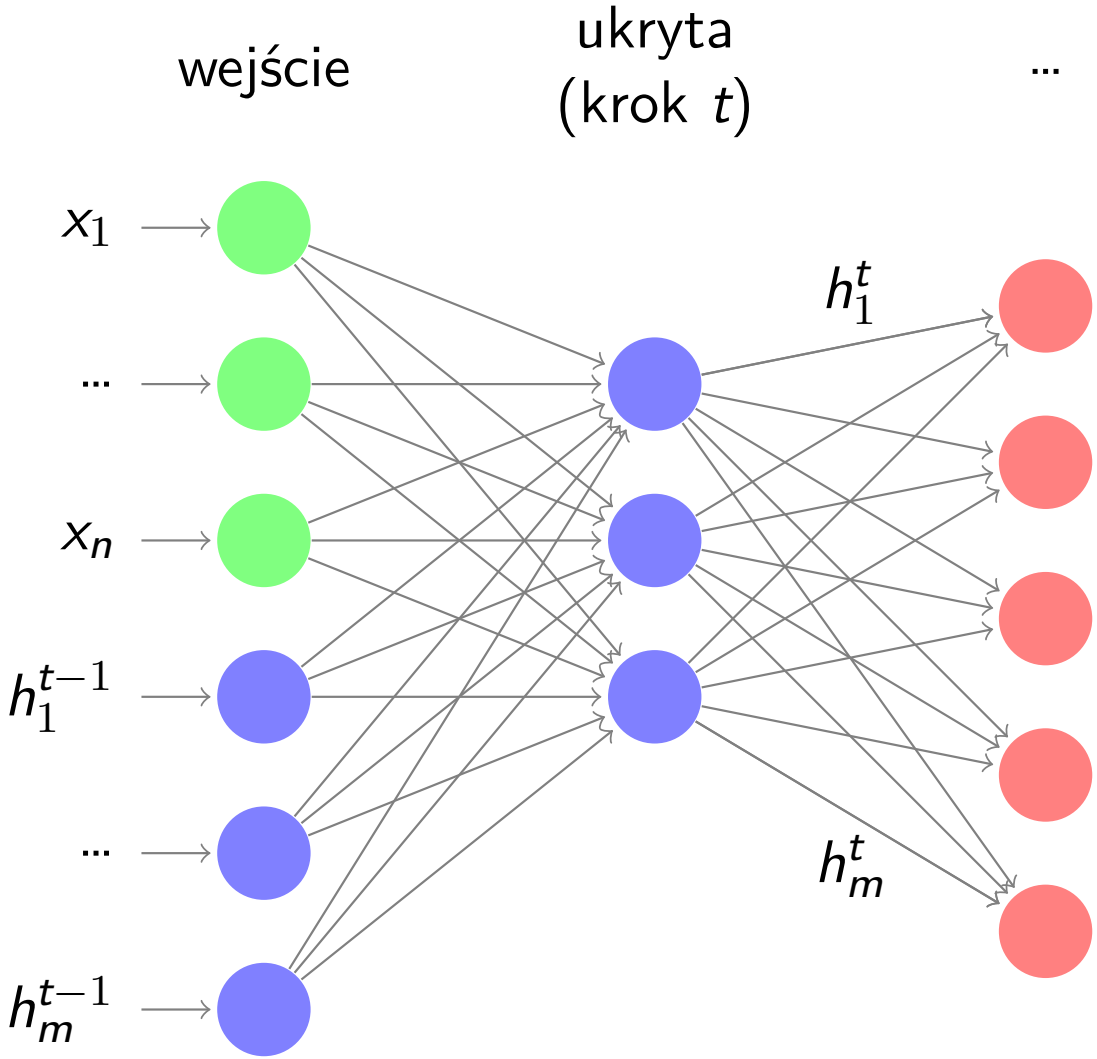
Kodowanie wejścia

Tekst to ciąg (sekwencja) dyskretnych jednostek języka.
Dwa problemy:

1. Jak zakodować sekwencję jednostek?
2. Jak dobrać jednostki języka? Jak je zakodować?

Rekurencyjna sieć neuronowa

Rekurencyjna sieć neuronowa to sieć, która radzi sobie z sekwencją.

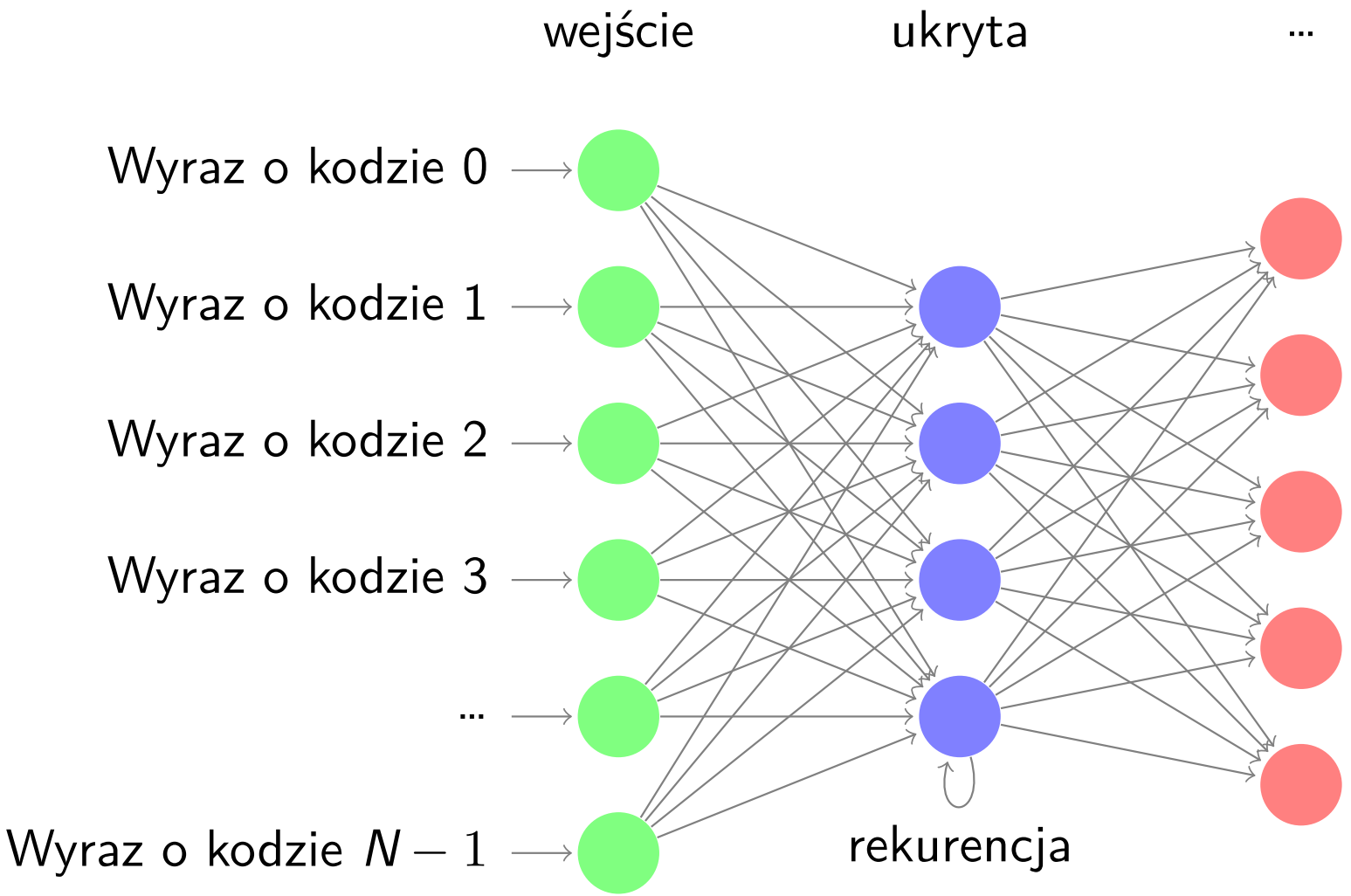


O rekurencyjnych sieciach neuronowych i ich wariantach/konkurentach powiemy na kolejnym wykładzie. Dzisiaj skupiamy się na pytaniu 2.

Część II

Jak podać wyrazy na wejściu sieci neuronowej?

One-hot encoding (kodowanie z gorącą jedynką)



$N = |V|$ – rozmiar słownika

One-hot encoding (kodowanie z gorącą jedynką) cd.

- wyrazom przypisujemy kody $0, 1, \ldots, N-1$ w sposób arbitralny
  - można posortować wyrazy alfabetycznie
  - ...ale nie jest to konieczne
- ...innymi słowy, każdemu wyrazowi (z całego słownika, nie tylko z tekstu wejściowego!) odpowiada dokładnie jeden neuron na wejściu
- dla danego wyrazu podajemy 1 dla „jego" neuronu („gorąca jedynka"), 0 – dla wszystkich pozostałych

Część III

Na czym polega problem?

- słów jest bardzo dużo
  - w słowniku PoliMorf: 3,8 mln (!) form fleksyjnych, 315 tys. lematów
    - *a*, *aa*, *AA*, *Aachen*, *Aalborg*, *AAP*, *Aar*, *Aara*, *Aaron*, *aaronowy*, ...
  - w korpusie „płci” w zbiorze uczącym: 2,6 mln „wyrazów”
    - µ, ˇ, □, ∫, □, ɛʀ, ɛ̈ʒ, ɛ□□, a, µa, ɚa, a□, ª, á, ...
    - w dev-secie: 413 tys. wyrazów, z czego 71 tys. spoza słownika (zbioru uczącego) – *out of vocabulary*, *OOV*: ʌ, א, aaaaaaaaaaaaaale, aaaaaaaaaaaalllllllllllllllleeeeeeeeeeżżżżżz, aaaaaaaahh, aaaaalutka, aaaabsolutna, aaaczy, aaaobserwować, aaguzjakiedy, aalek, aaronshay

Liczba wyrazów w PoliMorfie:

```
zcat PoliMorf-0.6.7.tab.gz | cut -f 1 | sort -u | wc
```

Liczba wyrazów w korpusie płci:

```
zcat train.tsv.gz| perl -C -ne 'eval{print lc($&),"\n" while /\p{L}+/g}' | sort -u | wc -l
```

słów jest bardzo dużo ...

- sieci neuronowe tego nie lubią
  - dużo neuronów na wejściu = dużo wag do wyuczenia między warstwą wejścia a pierwszą warstwą ukrytą
  - może nie zmieścić się w pamięci GPU
- sytuację pogarsza rozkład słów
  - mało częstych słów, długi ogon rzadkich słów
  - ...czy wręcz *hapaxów*
  - w zbiorze testowym będzie dużo słów OOV
    - ostatecznie wszystkie wylądują w jednym worze
    - ...bo cóż innego można z nimi zrobić?
- patrzenie na kształt wyrazów?
  - *tapet* nie ma nic wspólnego z *tapetą*
  - *sowa* nie wygląda jak *ptak*, *puszczyk*, *jastrząb*, *kura* itd.
  - ...choć OK, *dom* ma coś wspólnego z *domem*, *domkiem*, *domownikiem*, *domowym* i *udomowieniem* (czy *sęp* ma coś wspólnego z *zasępieniem*, a *słoń* z *zasłonieniem*?)
    - to fleksja i słowotwórstwo, w językach takich jak angielski i chiński – jeszcze mniejsze znaczenie

Część IV

„Proste” rozwiązania

Rozwiązanie 1. Rozpatrywać mniej słów

Rozpatrywać tylko N najczęstszych słów.

```
zcat train.tsv.gz| perl -C -ne 'eval{print lc($&),"\n" while /\p{L}+/g}' \
 | sort | uniq -c | sort -k 1rn \
 | head -n 10 | cut -b 9- | sort
```

Dla $N = 50000$ pierwsze 10 słów w porządku alfabetycznym:
a, *ă*, *â*, *ä*, *aa*, *aaa*, *aaaa*, *aaaaa*, *aaron*, *ab*.

50000 najczęstszych słów (1,9% typów) pokrywa jaki odsetek wystąpień?

Rozwiązanie 1. Rozpatrywać mniej słów

Rozpatrywać tylko N najczęstszych słów.

```
zcat train.tsv.gz| perl -C -ne 'eval{print lc($&),"\n" while /\p{L}+/g}' \
 | sort | uniq -c | sort -k 1rn \
 | head -n 10 | cut -b 9- | sort
```

Dla $N = 50000$ pierwsze 10 słów w porządku alfabetycznym:
a, *ă*, *â*, *ä*, *aa*, *aaa*, *aaaa*, *aaaaa*, *aaron*, *ab*.

50000 najczęstszych słów (1,9% typów) pokrywa jaki odsetek wystąpień? 89,6%

Rozwiązanie 2. Lematyzacja/rdzeniowanie

- wymaga wiedzy językowej
  - lematyzacja: reguły fleksji + lista wyjątków albo baza form fleksyjnych
  - rdzeniowanie: reguły (zob. stemer Snowball)
- lematyzacja: daje niejednoznaczny wynik (chyba że wyuczymy model ujednoznaczniania)
- rdzeniowanie: może wrzucać za dużo wyrazów do tej samej przegródki

Rozwiązanie 2. Lematyzacja/rdzeniowanie

- wymaga wiedzy językowej
  - lematyzacja: reguły fleksji + lista wyjątków albo baza form fleksyjnych
  - rdzeniowanie: reguły (zob. stemer Snowball)
- lematyzacja: daje niejednoznaczny wynik (chyba że wyuczymy model ujednoznaczniania)
- rdzeniowanie: może wrzucać za dużo wyrazów do tej samej przegródki
- ...zazwyczaj niewiele dają :(

Rozwiązanie 3. Znaki

- pojedynczy znak alfabetu nic nie znaczy
- rozmiar wejścia przy kodowaniu gorącą jedynkę dramatycznie się zmniejsza
- może działać, jeśli dodać wielowarstwową sieć neuronową
- może być bardzo kosztowne obliczeniowo

Xiang Zhang, Yann LeCun, *Character-level Convolutional Networks for Text Classification*, 2015.

- 9 warstw!

Część V

Kodowanie digramami

Kodowanie digramami (*byte pair encoding, BPE*)

Ani znaki, ani wyrazy – coś pomiędzy: jednostki podwyrazowe (*subword units*).
Lista jednostek jest automatycznie indukowana na podstawie tekstu. Ich liczba musi być z góry określona.

wejście `to be or not to be that is the question`

oznacz końce `to$ be$ or$ not$ to$ be$ that$ is$ the$ question$`

najczęstsza para `to$ b`[`e$`]$_1$ `or$ not$ to$ b`[`e$`]$_1$ `that$ is$ th`[`e$`]$_1$ `question$`

najczęstsza para `to$ b`[`e$`]$_1$ `or$ not$ to$ b`[`e$`]$_1$ [`th`]$_2$`at$ is$` [`th`]$_2$[`e$`]$_1$ `question$`

najczęstsza para `to$` [`b`[`e$`]]$_3$ `or$ not$ to$` [`b`[`e$`]]$_3$ [`th`]$_2$`at$ is$` [`th`]$_2$[`e$`]$_1$ `question$`

najczęstsza para `t`[`o$`]$_4$ [`b`[`e$`]]$_3$ `or$ not$ t`[`o$`]$_4$ [`b`[`e$`]]$_3$ [`th`]$_2$`at$ is$` [`th`]$_2$[`e$`]$_1$ `question$`

Kodowanie digramami (*byte pair encoding, BPE*), cd.

A zatem dla liczby jednostek równej 4 otrzymamy następujący słownik jednostek podwyrazowych:

```
1   e$
2   th
3   b1
4   to
```

Zamiana tekstu na jednostki podwyrazowe przebiega podobnie:

```
      wejście tom will be the best
oznacz końce tom$ will$ be$ the$ best$
  szukaj par 4m$ will$ be1 21 best$
  szukaj par 4m$ will$ 3 21 best$
```

Na wejściu wchodzą kody (oraz ewentualnie pozostałe znaki).

Przykład

`https://github.com/rsennrich/subword-nmt`

Zastosowanie na korpusie płci (50000 jednostek)

```
pair 0: i e -> ie (frequency 32747908)
pair 1: a </w> -> a</w> (frequency 24651991)
pair 2: o </w> -> o</w> (frequency 20748719)
pair 3: m </w> -> m</w> (frequency 17271135)
pair 4: e </w> -> e</w> (frequency 16108344)
pair 5: i </w> -> i</w> (frequency 16105444)
pair 6: ie </w> -> ie</w> (frequency 15877548)
pair 7: y </w> -> y</w> (frequency 13640651)
pair 8: c z -> cz (frequency 11171809)
pair 9: s t -> st (frequency 10502604)
pair 10: p o -> po (frequency 9936396)
pair 11: n ie</w> -> nie</w> (frequency 9584354)
pair 12: c h -> ch (frequency 9038815)
pair 13: r z -> rz (frequency 8795769)
pair 14: s z -> sz (frequency 8505324)
pair 15: d z -> dz (frequency 8161356)
pair 16: e m</w> -> em</w> (frequency 7971682)
pair 17: r a -> ra (frequency 7654533)
pair 18: n i -> ni (frequency 7526619)
pair 19: o w -> ow (frequency 7465858)
pair 20: ę </w> -> ę</w> (frequency 7429832)
```

Przykład (cd.)

Pierwsze 10 posortowanych alfabetycznie: *aa*, *aaa*, *aaaa*, *aaaaaaaa*, *aaaaa$*, *aaaa$*, *aaa$*, *aardzo$*, *aa$*, *ab*

Wyjście:

> *zze nie obcią@@ ża włosów do@@ ve he@@ at defen@@ se the@@ ra@@ py m@@ gie@@ łka do włosów naraż@@ onych na działanie wysokiej temperatury ok zł ml muszę przyznać że się rozczarowałem nie dość że włosy mi się niemiłosiernie pu@@ szyły to jeszcze jakby się wysu@@ szyły więce nie obcią@@ ża włosów do@@ ve he@@ at defen@@ se the@@ ra@@ py m@@ gie@@ łka do włosów naraż@@ onych na działanie wysokiej temperatury ok zł ml muszę przyznać że się rozczarowałem nie dość że włosy mi się niemiłosiernie pu@@ szyły to jeszcze jakby się wysu@@ szyły więc*

Część VI

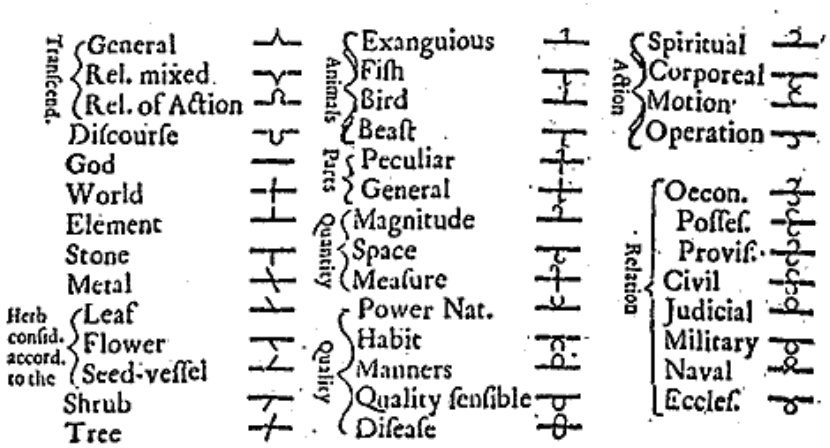
Ciągłe reprezentacje słów

Jeszcze raz, jaki mamy problem ze słowami?

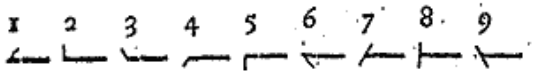
- *tapet* nie ma nic wspólnego z *tapetą*
- *sowa* nie wygląda jak *ptak*, *puszczyk*, *jastrząb*, *kura* itd.

Rozwiązanie 4. Analityczny język Johna Wilkinsa

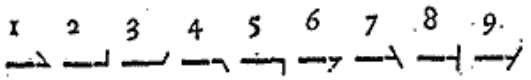
- *de* — żywioł
  - *deb* — ogień
    - *debα* — płomień
  - *det* — meteor (zjawisko meteorologiczne)
    - *detα* — tęcza



The Differences are to be affixed unto that end which is on the left ſide of the Character, according to this order;



The Species ſhould be affixed at the other end of the Character according to the like order.



... czyli dzisiaj sieci słów (*wordnets*)

polski Słowosieć (`http://plwordnet.pwr.wroc.pl`)

angielski Princeton Wordnet (`https://wordnet.princeton.edu/`) ...i Słowosieć!

zjawisko$_1$

- zjawisko naturalne$_1$
  - płomień$_2$
  - zjawisko atmosferyczne$_1$
    - tęcza$_1$

- arbitralność klasyfikacji

  [...] zwierzęta dzielą się na:

  a. *należące do Cesarza,*
  b. *zabalsamowane,*
  c. *tresowane,*
  d. *prosięta,*
  e. *syreny,*
  f. *fantastyczne,*
  g. *bezpańskie psy,*
  h. *włączone do niniejszej klasyfikacji,*
  i. *miotające się jak szalone,*
  j. *niezliczone,*
  k. *narysowane cienkim pędzelkiem z wielbłądziego włosia,*
  l. *et caetera,*
  m. *które właśnie rozbiły wazon,*
  n. *które z daleka podobne są do much.*

- w praktyce nie zawsze zastosowanie słowosieci wiele daje

Ciągła reprezentacja słów?

- słowosieć to dyskretna, hierarchiczna reprezentacja (znaczenia) słów
  - można mierzyć odległość między słowami (liczba kroków w górę do najbliższego wspólnego hiperonimu)
    - „skokowe”
    - arbitralne (bo słowosieć arbitralna)
- a może reprezentować słowa w sposób ciągły?
  - rzutujemy każde słowa na wektory 100-1000 liczb
  - mała zmiana wartości w pojedynczym (lub każdym) wymiarze oznacza małą zmianą w znaczeniu słowa
  - odległość indukowana w sposób naturalny

Rozwiązanie 5. *Word embeddings*

tylko jak?

Rozwiązanie 5. *Word embeddings*

tylko jak?

word2vec

Rozwiązanie 5. *Word embeddings*

tylko jak?

word2vec

1. Znajdź duży korpus tekstu
2. Stokenizuj go i przerób na małe litery (zazwyczaj niewiele więcej trzeba robić)
3. `word2vec -train corpus.txt -output vectors.bin -binary 1 -size 200`
4. ...magia ...
5. `word-analogy vectors.bin`
   - `berlin : germany warsaw : ?`

word2vec – historia

- wcześniej znane były inne sposoby reprezentowania słów w sposób ciągły: Latent Semantic Analysis (LSA) i Latent Dirichlet Allocation (LDA)
  - word2vec daje lepsze wyniki (?? — trwają spory)
  - mniej efektywne
- neuronowy model języka to sieć neuronowa, która podaje prawdopodobieństwo następnego słowa
  - np.: 2-warstwowa RNN z gorącą jedynką na wejściu i softmaksem na wyjściu
- ...(Mikolov i in. 2013) przypadkiem odkrywają, że pierwsza ukryta warstwa takiego neuronowego modelu języka ma ciekawe własności
  - (hmmm „z gorącą jedynką na wejściu" – zatoczyliśmy koło do rozwiązania nr 1!)

word2vec – teraźniejszość

- (Mikolov i in. 2013) przerabiają neuronowy model języka:
  - w bardzo efektywny sposób (bez GPU!)
  - z dwiema możliwymi architekturami

    Bag-of-Words odgadujemy słowo na podstawie 4 poprzednich i 4 nastepnych

    Skip-Gram odgadujemy kontekst na podstawie słowa (lepsze, ale wolniejsze niż Bag-Of-Words)
  - powstaje otwartoźródłowy word2vec (C)
- powstają klony (i konkurenci) word2vec
  - gensim (Python)
    - także doc2vec, który daje wektory dla zdań/dokumentów
  - Deeplearning4j / Word2Vec (Java/Scala)
  - GloVe (C)
- obecnie powszechne w sieciach neuronowych używanych do tekstu
  - wektory są wstępnie uczone (*pre-trained*)
  - ...ale mogą być strojone i poprawiana przez back-propagation!