16. Vektorová reprezentace slov a její aplikace, Word2Vec a GloVe.

- Většina přístupů nakonec pracovala se slovy pouze jako s diskrétními objekty

Vektorová reprezentace slov a její aplikace

Diskrétní reprezentace slov

- Každé slovo reprezentováno samostatně

- Pro daný slovník zakódováno jako vektor nul s jednou jedničkou

„One-hot encoding“:

Petr = [0 0 ... 0 1 0 ... 0 0]

Pavel = [0 1 0 ...0 ....0]

- Celá řada problémů

    - Prostor má **obří dimenzi |V|**

    - Poloha vektorů a vzdálenosti mezi nimi nijak nesouvisí s významem slova

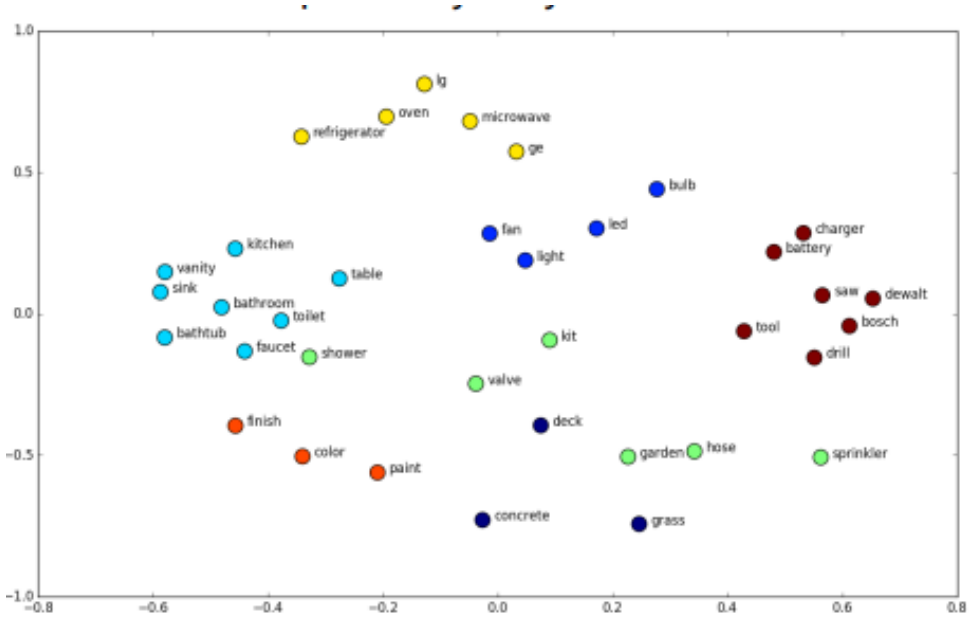
    - Podobná slova neleží ve stejné oblasti prostoru

Reprezentace přes podobnostní rozložení

- Cílem je vytvořit matematický model reprezentující podobnostní rozložení slov -> najít prostor, v němž jsou slova **rozložena podle významu**

- Každé slovo bude v tomto prostoru reprezentováno **vektorem**, který bude ležet blízko vektorů slov s podobným významem

- Prostor hledáme tak, že daný model maximalizuje pro všechna slova pravděpodobnost výskytu jejich okolí

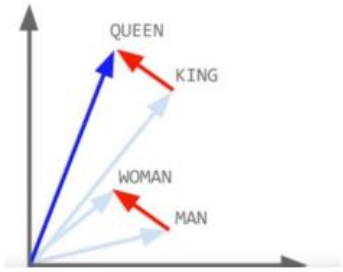


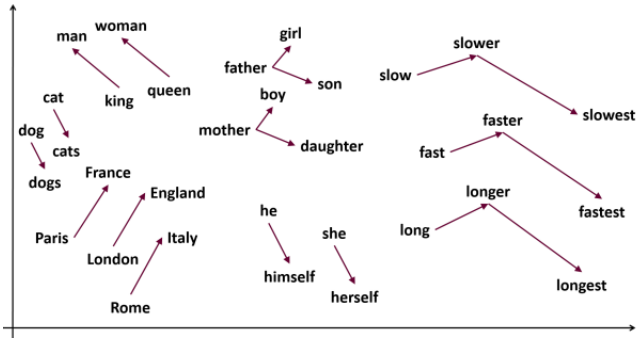
Word2Vec a GloVe

WORD2VEC

- Vektory reprezentující významově podobná slova tvoří shluky
  - Jsou blízko sebe
- Přičítáním a odčítáním vektorů je možné posouvat nebo přenášet význam:

$$v(king) - [v(man) - v(woman)] = v(queen)$$





Příklad trénování

- Slova ze slovníku zakódujeme pomocí „one-hot“ kódování
- Požadovaný vektor (zde pro slovo „šel“) přivedeme na vstup
- Na výstupu chceme získat postupně vektory odpovídající slovům „Petr“, „dnes“, „do“ a „kina“
- Například pro dvojici „šel“ a „Petr“ je
  - Vstupní vektor x = [0 0 1 0 0] a požadovaný výstupní vektory y = [1 0 0 0 0]
- Výstup ze sítě (funkce softmax) ovšem neodpovídá vždy přesně požadovanému
  - Může být například [0.7 0.1 0.1 0.05 0.05]
- Vznikne chyba, která se poté posčítá za všechna okolní slova
  - Na základě celkové chyby se přepočítají parametry modelu
    - Jde o algoritmus zpětné propagace (viz předmět USU)
    - Během trénování minimalizujeme cross-entropii mezi skutečným výstupem ze sítě a požadovanými hodnotami

Schéma modelu

• Jde o model odpovídající NS s jednou skrytou vrstvou

• Parametry modelu jsou matice $\boldsymbol{W}1$ a $\boldsymbol{W}2$

• Na výstupu je softmax => krit. funkce má význam křížové entropie

• **x** = vektor reprez. vstupní slovo, obsahuje pouze jednu jedničku, dimenze [1, V]

• $\boldsymbol{W}1$ = matice vah skryté vrstvy o dimenzi [V, N], reprezentuje **word embedings**

- na řádku v je embedding pro v-té slovo ze slovníku
- vektor x obsahuje pouze jednu jedničku
- Součinem $\boldsymbol{x}^T \boldsymbol{W}_1$ proto vybereme vždy jeden příslušný řádek této matice

• $\boldsymbol{h}=\boldsymbol{x}\ \boldsymbol{T}\ \boldsymbol{W}1$

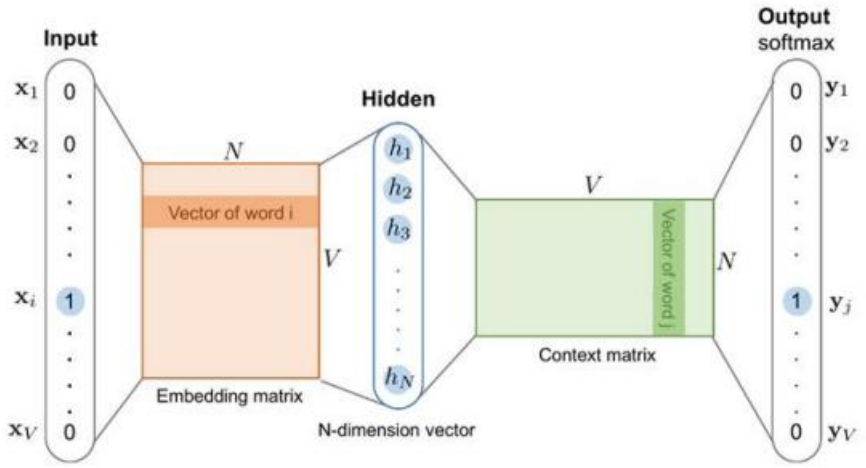
- vektor odpovídající embeddingu slova
- skrytá vrstva má lineární aktivační funkci
- tento embedding není nijak modifikován

• $\boldsymbol{W}2$

- "matice vah výstupní vrstvy dimenzi [N,V]"
- odpovídá kontextové matici

• $\boldsymbol{y}=\boldsymbol{softmax}(\boldsymbol{hT}\ \boldsymbol{W}2\ )$

- výstupní vektor, ideálně přesně odpovídá „one hot" zakódování požadovaného výstupní slova
- reálně se liší a podle odchylky se při trénování upraví všechny váhy



SKIP GRAM vs C-BOW

• U skip-gramu se sečte chyba na výstupu pro všechna predikovaná slova z okolí cílového slova

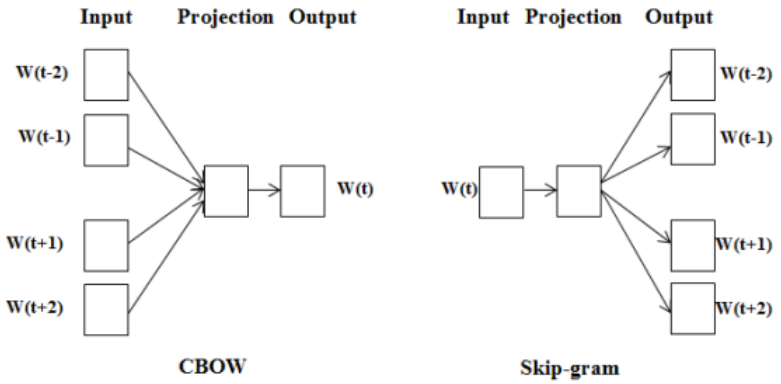
• U varianty CBOW se maximalizuje pravděpodobnost cílového slova na základě okolních slov

• Chyba za jednotlivá okolní slova se na výstupu sítě průměruje

• Pro obě metody obvykle platí, že čím větší N tím lepší výsledky

• Okolí se volí cca 10 pro skip-gram a 5 pro CBOW

Přístup: CBOW vs Skip-gram



Nevýhody popsaného způsobu trénování:

1. Výpočet softmaxu pro velké slovníku je náročný (exponenciála)

2. Pro dané cílové slovo se významně mění hodnoty pouze omezeného počtu vah, ostatní váhy se přesto aktualizují nadbytečně o zanedbatelně malé hodnoty

- Příklad
  - Předtrénovaný Google model má 3 miliony slov
    - Při trénování je pouze 1 slovo ze 3M cílové
    - Většinu času aktualizujeme váhy, které nesouvisí s cílovým slovem o malé hodnoty (viz 2.)

• Řešení ve výběru trénovací metody

1. Hierarchical Softmax - Složitější na implementaci

2. **Negative sampling** - Dosahuje lepších výsledků

Negative Sampling #1

- Umožňuje pro každý vzorek modifikovat pouze malou část vah
- Problém se reformuluje
  1. Místo predikce slova „šel" na základě okolních slov se trénuje model predikující, zda je nebo není slovo „šel" sousední se slovy „Petr", „do" ...
     - Klasifikaci 1 z V převedeme na V binárních klasifikací
  2. Trénování binárního klasifikátoru pro dané cílové slovo se dále zjednoduší tak, že se kromě slova v okolí cílového slova vybere i několik (typicky 5) náhodně vybraných slov, negativních případů, která v daném okolí zrovna nejsou
     - Dvojici (šel;Petr) doplníme např. o dvojice (šel;dům), (šel;včera)
     - Pro slovo v okolí chceme na výstupu NS hodnotu 1
     - Pro negativní vzorky hodnotu 0

• Chyba se pak propaguje zpět pouze pro použitá slova a zároveň se nevyčísluje softmax pro V slov ⇒ zrychlení výpočtu

• Slova se jako negativní vzorky vybírají náhodně s pravděpodobností $P=u(w)$ 3/4, kde $u(w)$ je unigram daného slova

• Četnější slova budou vybrána častěji než méně četná

• Mocnina ¾ zvýhodňuje méně četná slova oproti samotnému unigramovému rozložení

Skip-gram nebo CBOW (a Negative sampling)?

- Skip-gram je pomalejší + lepší pro málo četná slova
- CBOW je rychlejší + lepší pro četná slova
- Softmax je pomalejší + lepší pro málo četná slova
- Negative sampling je rychlejší + lepší pro četná slova a málo dimenzionální vektory

Využití:

• Překlad z jazyka do jazyka

• Analýza sentimentu

• Klasifikace textu

• Automatická sumarizace

• Identifikace jazyka z textu

Glove

• Global Vectors for Word Representation

Intuice

• Word2Vec nepřímo modeluje společný výskyt slov

• Společný výskyt slov můžeme spočítat přímo v prvním kroku

2 možnosti počítání slov

• Okénko -> Vybereme n slov okolo slova a v tomto okénku počítáme společný výskyt -> Zachycuje syntaktickou i sémantickou informaci

• Dokument -> Místo počítání v okénku počítáme v celém dokumentu (LSA)

počítání s okénkem

- Text:
  - I like NLP.
  - I like deep learning.
- Okénko: 1 slovo (v okolí)

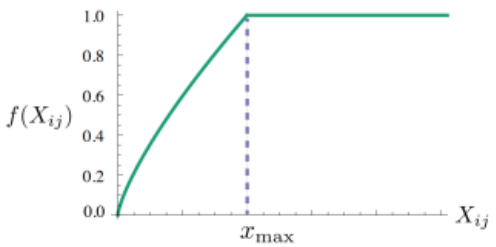
	I	like	NLP	deep	learning
I	0	2	0	0	0
like	2	0	1	1	0
NLP	0	1	0	0	0
deep	0	1	0	0	1
learning	0	0	0	1	0

- Problémy takového počítání
  - Matice se zvětšuje s velikostí slovníku
  - V matici se vyskytuje velké množství nul
  - Často vyskytující se slova budou mít velký počet společného výskytu
- Vylepšení
  - Použití Singular Value Decomposition (SVD) pro redukci dimenze
  - Velkou četnost společného výskytu omezíme maximální hodnotou
    - Např. min(100, count)
  - Použití vzdálenosti při počítání v okénku
    - „I like NLP ...“
      - X(I, like) += 1
      - X(I, NLP) += $\frac{1}{2}$
      - X(I, ...) += $\frac{1}{3}$
- Posledním vylepšením je získat každý počet v rozsahu 0-1
  - Lepší pro učení NN

$$f(X_{ij}) = \left(\frac{X_{ij}}{x_{max}}\right)^{\alpha} \text{ if x} < \text{xmax, else 1}$$

α = 0,75; x_{max} = 100

X_{ij} – hodnota z matice počtů společného výskytu



- SVD se složitěji optimalizuje
  - Problém s časovou náročností pro větší matice

- Výsledné řešení
  - Dvě matice vah *u* a *v* (dimenze [V, N]) a 2× bias vektor
    - Podobně jako u Word2Vec
  - Učí se pomocí váženého MSE

$$J = \frac{1}{2}\sum_{i,j=1}^{V} f(X_{ij})\left(u_i^T v_j + bi + bj - \log X_{ij}\right)^2$$

- Funguje dobře na menším množství dat i s menší velikostí embeddingu