06/08/2020

Stanford CS224N

(1) How do we have usable meaning in a computer?

→ Use e.g. WordNet, a thesaurus containing lists of synonym sets and hypernyms ("is a" relationships)

Synonyms of "good"

```
from nltk.corpus import wordnet as wn
poses = {'n': 'noun', 'v': 'verb',
's': 'adj (s)', 'a': 'adj', 'r': 'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format
    (poses[synset.pos()], ", ".
    join([l.name() for l in
    synset.lemmas()])))
```

Output:

```
noun: good
noun: good, goodness
```

Hypernyms of "panda"

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

Output:

```
[Synset('procyonid.n.01'),
 Synset('carnivore.n.01'), ...]
```

Words can be represented by one-hot vectors

(2) Representing words by their context

→ • Distributional semantics: A word's meaning is given by the words that frequently appear close-by.

- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window)
- Use the many contexts of w to build up a representation of w.

③ Word vectors: We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts.

Word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

$$\text{banking} = \begin{bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

Word meaning a neural word vector-visualization

Word2vec : Overview

↳ Idea: A framework for learning word vectors

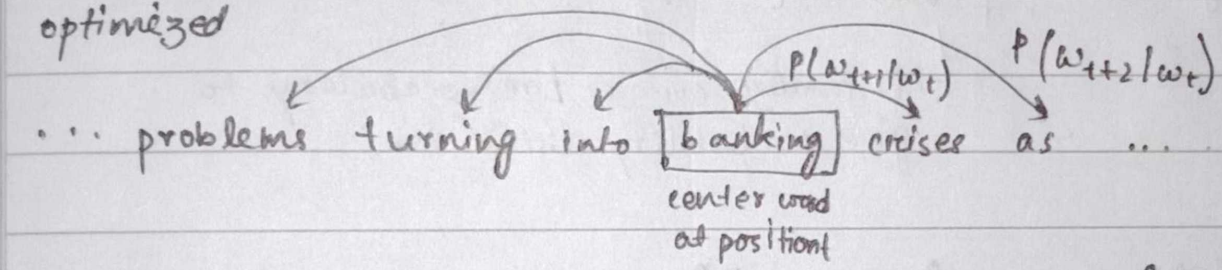
- We have a large corpus of text
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o.
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_j.

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized



Sometimes called cost or loss function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

If we minimize $J(\theta)$ that means we will be good at predicting words in context of another word.

(6) How to calculate $P(w_{t+j} | w_t ; \theta)$?

→ We will use two vectors per word w:

(i) v_w when w is a center word (c)

(ii) u_w when w is a context word (o)

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of o and c

$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

Exponentiation makes anything positive

↳ Normalize over entire vocabulary to give probability distribution.

(7) Softmax ~~max~~ function $R^n \to R^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

Softmax function maps arbitrary values x_i to a probability distribution p_i

- "max" because amplifies probability of largest x_i
- "soft" because still assigns some probability to smaller x
- frequently used in deep learning.

(8) Training a model by optimizing parameters: We adjust parameters to minimize a loss

$$\max J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w_{t+j} \mid w_t ; \theta)$$

We want to minimize $J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log p(w_{t+j} \mid w_t)$

for that we need to change θ

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_o^T v_c)}$$

$$\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_o^T v_c)}$$

$$\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_o^T v_c)$$

numerator — denominator

$$\frac{\partial}{\partial v_c} u_o^T v_c = u_o \quad \left(\because \frac{\partial}{\partial v_1} \{ u_{01} v_{c1} + u_{02} v_{c2} + \cdots \} = u_{01} \right.$$
$$\left. \frac{\partial}{\partial v_1} \frac{\partial}{\partial v_2} \frac{\partial}{\partial v_3} \cdots \{ u_o^T v_c \} = \{ u_{01} + u_{02} + \cdots \} = u_o \right)$$

↳ It's still a vector bec. we had a 100 dimensional representation of a word

$$\frac{\partial}{\partial v_c} \log \underbrace{\sum_{w=1}^{V} \exp(u_o^T v_c)}_{z(v_c)}$$

f (assume it a function)

$\therefore$ We will apply chain rule

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_o^T v_c)} \cdot \sum_{x=1}^{V} \frac{\partial}{\partial v_c} \exp(u_x^T v_c)$$

(derivative of log)

$$= \frac{1}{\sum_{w=1}^{V} \exp(u_o^T v_c)} \sum_{x=1}^{V} \exp(u_x^T v_c) \cdot u_x$$

$$\frac{\partial}{\partial v_c} \log p(o|c) = u_o - \frac{\sum_{x=1}^{V} \exp(u_x^T v_c) \cdot u_x}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

$$= u_o - \sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \cdot u_x$$

$$= u_o - \sum_{x=1}^{V} p(x|c) \cdot u_x$$

— x —

Posterior probability → Probability of an event occurring after taking into consideration new information. Posterior probability is calculated by updating the prior probability using Bayes' Theorem.

posterior ← likelihood ← Prior

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

$P(B)$ ← Normalizing constant

Word2vec → The Skip-Gram Model

For spam/ham classification, posterior probability is calculated for each unique word in the email. Plugging this into Bayes Rule, our formula will look:

$$P(\text{spam}|\text{word}) = \frac{P(\text{word}|\text{spam}) \times P(\text{spam})}{P(\text{word})}$$

$$P(\text{word}) = P(\text{word}|\text{spam}) \times P(\text{spam}) + P(\text{word}|\text{not spam}) \times P(\text{not spam})$$