Sistemas Inteligentes

Word Embeddings (Socher and Manning)

José Eduardo Ochoa Luna
Dr. Ciencias - Universidade de São Paulo

Maestría C.C. Universidad Católica San Pablo
Sistemas Inteligentes

23 de Octubre 2018

How do we have usable meaning in a computer

Common answer: Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets

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

Problems with this discrete representation

- Missing nuances: synonyms, adept expert, skillful

Problems with this discrete representation

- Missing nuances: synonyms, adept expert, skillful
- Missing new words (impossible to keep up to date): crack, ninja

Problems with this discrete representation

- Missing nuances: synonyms, adept expert, skillful
- Missing new words (impossible to keep up to date): crack, ninja
- Subjective

Problems with this discrete representation

- Missing nuances: synonyms, adept expert, skillful
- Missing new words (impossible to keep up to date): crack, ninja
- Subjective
- Requires human labor to create and adapt

Problems with this discrete representation

- Missing nuances: synonyms, adept expert, skillful
- Missing new words (impossible to keep up to date): crack, ninja
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity

Problems with this discrete representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols: hotel, conference, walk

Problems with this discrete representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols: hotel, conference, walk
- In vector space terms, this is a vector with one 1 and a lot of zeroes [00001000]

Problems with this discrete representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols: hotel, conference, walk
- In vector space terms, this is a vector with one 1 and a lot of zeroes [00001000]
- Dimensionality: 20K (speech), 500k(big vocab) 13M (Google 1T)

Problems with this discrete representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols: hotel, conference, walk
- In vector space terms, this is a vector with one 1 and a lot of zeroes [00001000]
- Dimensionality: 20K (speech), 500k(big vocab) 13M (Google 1T)
- We call this a one-hot representation

Problems with this discrete representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols: hotel, conference, walk
- In vector space terms, this is a vector with one 1 and a lot of zeroes [00001000]
- Dimensionality: 20K (speech), 500k(big vocab) 13M (Google 1T)
- We call this a one-hot representation
- It is a localist representation

From symbolic to distributed representations

In Web search

- if an user searches for [Dell notebook battery size], we would like to match documents with "Dell laptop battery capacity"
- if an user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"

but

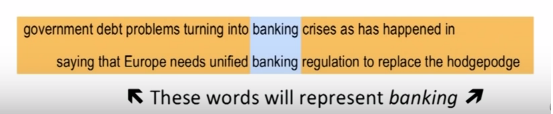
$$\mathit{motel}[0000000100]^T$$

$$\mathit{hotel}[0000100000] = 0$$

- Query and document vectors are orthogonal
- there is no natural notion of similarity in a set of one-hot vectors

Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors



Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors
- One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Word meaning is defined in terms of vectors

- We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context

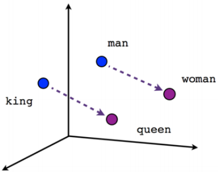
$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word meaning is defined in terms of vectors

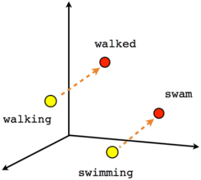
- We will build a dense vector for each word type, chosen so that it is good at predicting other words appearing in its context
- Those words also being represented by vectors

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

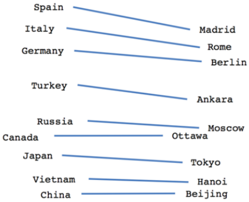
Example



Male-Female



Verb tense



Country-Capital

Word Embeddings Tasks

- In many NLP tasks, input is given as word sequences

Word Embeddings Tasks

- In many NLP tasks, input is given as word sequences
- Document classification

Word Embeddings Tasks

- In many NLP tasks, input is given as word sequences
- Document classification
- Part-of-speech tagging

Word Embeddings Tasks

- In many NLP tasks, input is given as word sequences
- Document classification
- Part-of-speech tagging
- Named Entity recognition

Word Embeddings Tasks

- In many NLP tasks, input is given as word sequences
- Document classification
- Part-of-speech tagging
- Named Entity recognition
- Question Answering

Word Embeddings Tasks

- In many NLP tasks, input is given as word sequences
- Document classification
- Part-of-speech tagging
- Named Entity recognition
- Question Answering
- Machine Translation

Basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word w_t and context words in terms of word vectors

$$p(context|w_t) = \ldots$$

which has a loss function, e.g.

$$1 - p(w_{-t}|w_t)$$

w_{-t} all other words in the context
we look at many positions t in a big language corpus
we keep adjusting the vector representations of words to minimize this loss

Word2vec

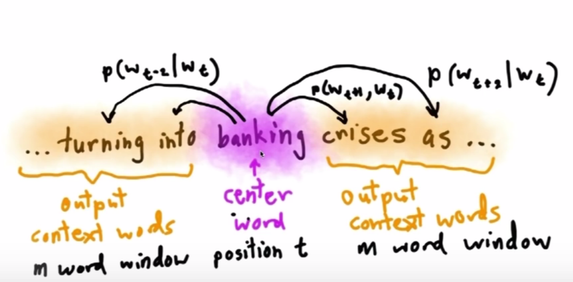
Predict between every word and its context words (Mikolov et al, 2013) Two algorithms

- Skip-grams (SG): predict context words given target (position independent)
- Continuous Bag of Words (CBOW): predict target word from bag-of-words context

Two training methods

- Hierarchical softmax
- Negative sampling

Skip-gram prediction



Details of word2vec

For each word $t = 1 \ldots T$, predict surrounding words in a window of "radius" m of every word

Objective function: Maximize the probability of any context word given the current center word:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m_{j \neq 0}} p(w_{t+j} | w_t; \theta)$$

Negative log likelihood

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m_{j \neq 0}} \log p(w_{t+j} | w_t; \theta)$$

where θ represents all variables we will optimize

The objective function - details

- Terminology: loss function = cost function = objective function

The objective function - details

- Terminology: loss function = cost function = objective function
- Usual loss for probability distribution: Cross-entropy loss

Details of Word2vec

Predict surrounding words is a window of radius m of every word
For $p(w_{t+j}|w_t)$ the simplest first formulation is

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

where o is the outside (output) word index, c is the center word index, v_c and u_o are center and outside vector of indice c and o
Softmax using word c to obtain probability of word o

Dot products

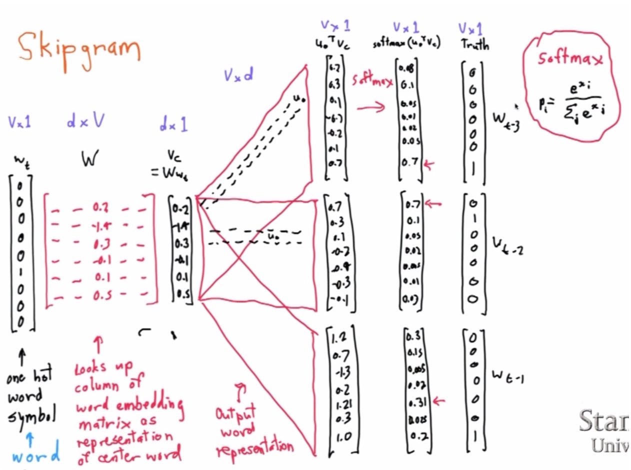
Dot product $u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$ Bigger if u and v are more similar

Iterate over $w = 1 \ldots W : u_w^T v$ means: work out how similar each word is to v!

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

Softmax function turns out numbers in a probability distribution

Skip Gram


Skipgram
V×1
d×V
d×1
V×d
w_t
W
v_c = W w_t
one hot word symbol
word
Looks up column of word embedding matrix as representation of center word
Output word representation
V×1
u_o^T v_c
V×1
softmax(u_o^T v_c)
V×1
Truth
softmax
Softmax
p_i = e^{x_i} / Σ_j e^{x_j}
w_{t-3}
w_{t-2}
w_{t-1}

Training

- We often define the set of all parameters in a model in terms of one longe vector θ
- With d-dimensional vector and V many words. Then optimize

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Every word has two vectors

Recall

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log p(w_{t+j}|w_t)$$

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

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

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

Gradients

1)

$$\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$$

2)

$$\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c) =? \text{ (chain rule)}$$

Second term - Chain Rule

$$
\begin{aligned}
\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c) &= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c) \\
&= \ldots [\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \exp(u_x^T v_c)] \\
&= \ldots [\sum_{x=1}^{V} exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c] \\
&= \ldots [\sum_{x=1}^{V} exp(u_x^T v_c) u_x] \\
&= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} [\sum_{x=1}^{V} exp(u_x^T v_c) u_x]
\end{aligned}
$$

Final Formula

$$\begin{aligned}\frac{\partial}{\partial v_c}\log\sum_{w=1}^{V}\exp(u_w^T v_c) &= \sum_{x=1}^{V}\frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V}\exp(u_w^T v_c)}u_x \\ &= \sum_{x=1}^{V}p(x|c)u_x\end{aligned}$$

Final formula

$$\frac{\partial}{\partial v_c}\log p(o|c) = \underbrace{u_o}_{\text{Observed}} - \underbrace{\sum_{x=1}^{V}p(x|c)u_x}_{\text{Expectation}}$$

Gradient Descent

We will optimize (maximize or minimize) our objective / cost functions

Updates would be for each element of θ :

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

In matrix notation for all parameters:

$$\theta^{new} = \theta^{old} - \alpha \frac{\partial}{\partial \theta^{old}} J(\theta)$$

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

Stochastic Gradient Descent

- But corpus may have 40B tokens and windows
- You would wait a very long time before making a single update
- instead: we will update parameters after each window t: stochastic gradient descent (SGD)

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J_t(\theta)$$

Tarea

Implementar lo solicitado en NeuralNetworkSemana3.ipynb