

NLP Week 1

Lecture 1 : Intro & Word Vectors

2. Human language and word meanings

GPT-3: A first step on the path to universal models

The SEC said, "Musk, your tweets are a blight. They really could cost you your job, if you don't stop all this tweeting at night."

Then Musk cried, "Why? The tweets I wrote are not mean, I don't use all-caps and I'm sure that my tweets are clean."

"But your tweets can move markets and that's why we're sore. You may be a genius and a billionaire, but it doesn't give you the right to be a bore!"

OpenAI

S: I broke the window.
Q: What did I break?
S: I gracefully saved the day.
Q: What did I gracefully save?
S: I gave John flowers.
Q: Who did I give flowers to?
S: I gave her a rose and a guitar.
Q: Who did I give a rose and a guitar to?

How many users have signed up since the start of 2020?
SELECT count(id) FROM users
WHERE created_at > '2020-01-01'

What is the average number of influencers each user is subscribed to?
SELECT avg(count) FROM (SELECT user_id, count(*)
FROM subscribers GROUP BY user_id)
AS avg_subscriptions_per_user

GPT-3

translate human language sentences into SQL

How do we represent the meaning of a word?

signifier (symbol) \Leftrightarrow signified (idea or thing)

problems with resources like WordNet?

- missing nuance / missing new meanings of the words

Representing words as discrete symbols

one-hot vectors

vector dimensions : number of words in voice

Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

```
motel = [0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 1 0 0 0 0 0 0 0]
```

These two vectors are **orthogonal**

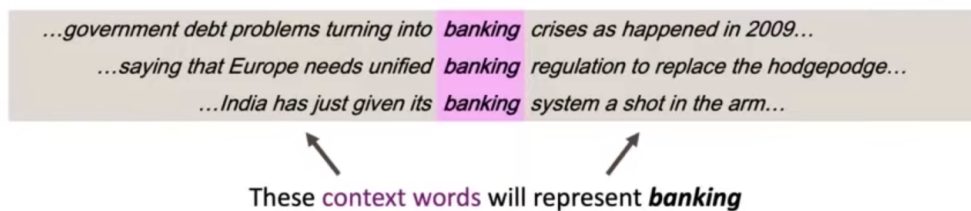
There is no natural notion of **similarity** for one-hot vectors!

Solution : learn to encode similarity in the vectors themselves

Representing words by their context

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

context : set of the words that appear nearby



Word vectors

word vectors = word embeddings = word representations

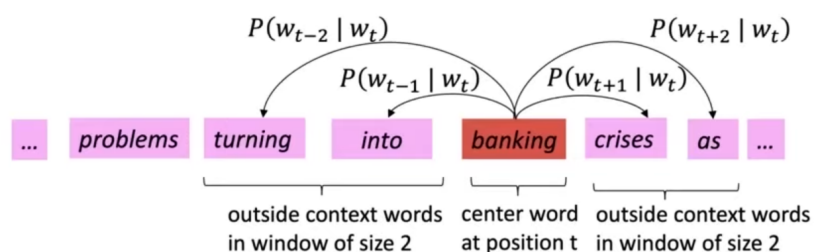
>they are distributed representation

3. Word2vec : Overview

idea :

- Use the **similarity** of c and o to **calculate the probability of o given c**
- **keep adjusting** the word vectors to maximize this probability

Example windows and process for computing $P(w_{t+j} | w_t)$



Word2vec: objective function

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

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

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

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

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

자막(c)

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

23

Then for a center word c and a context word o :

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

1. dot product compares similarity of o and c
2. exponentiation makes positive
3. normalize over entire voca to give probability distribution

This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$

Open region

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

The 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_i

But sort of a weird name because it returns a distribution!

To train the model : Optimize value of parameters to minimize loss

gradually adjust parameters to minimize a loss by walking down the gradient

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

$$\downarrow \frac{d}{dV_C} \log$$

$$\frac{d}{dV_c} \log \exp(u_0^T \cdot u_c) - \frac{d}{dV_c} \log \sum_{w=1}^V \exp(u_w^T \cdot V_c)$$

① $\frac{d}{dV_C} u_0^T V_C = u_0$ V_C : vector

$$\textcircled{2} \quad \frac{\partial}{\partial \mathbf{V}_c} \log \sum_{w=1}^V \exp(\mathbf{U} w^T \cdot \mathbf{V}_c) \quad f(g(\mathbf{x}))$$

$$= \frac{1}{\sum_{n=1}^N \exp(u_n^T \cdot v_c)} \cdot \frac{d}{dv_c} \underbrace{\sum_{n=1}^N \underbrace{\exp(u_n^T \cdot v_c)}_{f}}_{Z = g(v_c)} \rightarrow \sum_{n=1}^N \exp(u_n^T \cdot v_c) \cdot \underbrace{\frac{d}{dv_c} u_n^T v_c}_{u_n}$$

$$\begin{aligned} \frac{d}{dV_c} \log p(O|C) &= u_0 - \frac{\sum_{x=1}^V \exp(u_x^T \cdot V_c)}{\sum_{w=1}^T \exp(u_w^T \cdot V_c)} u_x \\ &= u_0 - \sum_{x=1}^V \frac{\exp(u_x^T \cdot V_c)}{\underbrace{\sum_{w=1}^T \exp(u_w^T \cdot V_c)}_{\text{Softmax}}} u_x = u_0 - \sum_{x=1}^V p(x|C) u_x \\ &= u_0 - \underbrace{\sum_{x=1}^V p(x|C) u_x}_{\text{expectation}} \\ &= \text{observed} - \text{expected} \end{aligned}$$

Gensim word vector visualization of various word vectors

```
In [7]: ! pip install gensim
```

```
Requirement already satisfied: gensim in /Users/songhyejun/conda/envs/Hello/lib/python3.8/site-packages (4.2.0)
Requirement already satisfied: smart-open>=1.8.1 in /Users/songhyejun/conda/envs/Hello/lib/python3.8/site-packages
(from gensim) (6.0.0)
Requirement already satisfied: numpy>=1.17.0 in /Users/songhyejun/conda/envs/Hello/lib/python3.8/site-packages (fro
m gensim) (1.23.1)
Requirement already satisfied: scipy>=0.18.1 in /Users/songhyejun/conda/envs/Hello/lib/python3.8/site-packages (fro
m gensim) (1.8.1)
WARNING: There was an error checking the latest version of pip.
```

```
In [9]: import numpy as np

%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('ggplot')

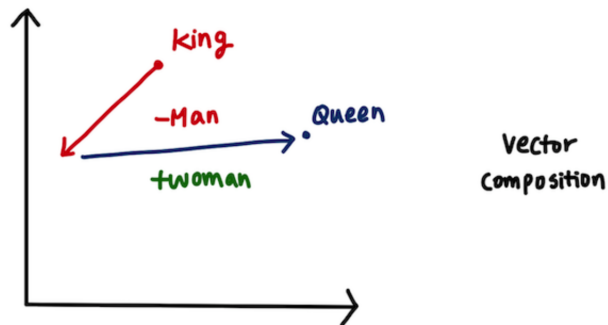
from sklearn.decomposition import PCA

import gensim.downloader as api
from gensim.models import KeyedVectors
```

```
In [11]: model = api.load("glove-wiki-gigaword-100")
print(type(model))

[=====] 100.0% 128.1/128.1MB downloaded
<class 'gensim.models.keyedvectors.KeyedVectors'>
```

```
In [17]: def analogy(x1,x2, y1):
result = model.most_similar(positive=[y1,x2], negative=[x1])
return result[0][0]
```



```
In [20]: analogy('man', 'king', 'woman')
```

```
Out[20]: 'queen'
```