history

Work on trying to do things with human language started in the 1950s

really in the last decade with neural networks that just enormous strides of progress have been made

- first big breakthroughs => using neural NLP systems for machine translation
  - 始于2014，到2016年已经部署于google
- GPT-2 2019s => first time a LLM could generate fluent text
- 2022 — now ChatGPT (from GPT3 to GPT 4)

Word2Vec

very common traditional solution => WordNet

↓

one hot vectors

eg motel = [0 0 0 1 0 0 0 0]
hotel = [0 0 1 0 0 0 0 0]

word vectors

$$\text{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \end{pmatrix} \qquad \text{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \end{pmatrix}$$

对应项点积大（同正负号）表更有关系

embedding : learn good word vectors

thinking : embedding each work as a position in high dimentional space

QA.

1. how to visualize

t-SNE : nonlinear dimensionality reduction (可视化降维)

↳ better than PCA

2. how to know how many dimensions is optimal to produce like a dense enough but not too sparse

↳ people works on it

depends on how much data to make your representations

100 → 300 → 1000 ~2000

↙ 获得文档 ↓ best previously ↘ LLM

3. why from -1 to 1, should we normalize?

↳ allways need, and people use regularization (正则化)

4. 词是否有一个 embedding

↳ 我们数是的

eg: bank { river bank / the financial institution

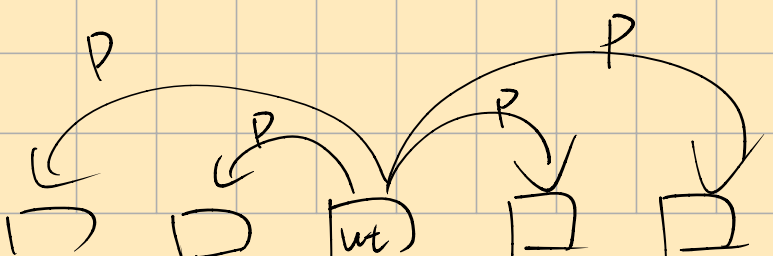
事实上，它是某种程度上不同词义的平均 (an average evoked by both senses)

正式进入 Word2vec

large amount of text

in NLP called corpus (语料库)

①



m m

θ are the parameters of our model

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

$\prod_{t=1}^{T}$: T个单词　$\prod_{-m \le j \le m, j \neq 0}$: w_t 周围2m长的窗口的"上下文单词"

Target = make $L(\theta)$ as big as possible

2 tricks:

1. put a minus sign　因为我们比起最大化，更善长最小化
2. introduce a logarithm (对数), make $\prod_{t=1}^{T}$ to $\sum_{t=1}^{T}$
   求似然的常规操作

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

因为要加等式 $J(\theta)$ 变得很大，因此我们的目标函数是平均负对数似然 (average negative log likelihood)

(quantity)

② Q: how to calculate the probability?

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

explanation:

1. $u_o^T v_c$ is dot product

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

Larger dot product = larger probability

2. $\exp(x)$: exponentiate a number $= e^x$

Why? To get something positive out

3. $\frac{\exp(\ )}{\sum \exp(\ )}$: normalization, make probabilities in my vocabulary will sum up to 1.

1 + 2 + 3 : Softmax function (I've learned from ML)

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

Can get a probability distribution

① It was discovered after using it that softmax function can amplifies the probability of the largest value, to make model more confident (自信)

② Why it can in math?

Set $z_1 = z_2 + \Delta$

$$\frac{p_1}{p_2} = e^{z_1 - z_2} = e^{\Delta}$$ ← exponential

③ Train the model = optimize value of parameters to minimize loss

θ : represents all the parameters in one long vector

$\theta = [v_{aardvark}, v_a \cdots v_{zebra}, v_{aardvark}, v_a \cdots v_{zebra}]^T$

length(θ) = length of (aardvark to zebra) × 2 × word number

aardvark/zebra is the first/last word of vocabulary

center word

context word

Remember: every word has two vectors

but why we say a word have only a embedding?

embedding = the average of two vectors

We optimize parameters by walking down the gradient

how to do it in math

we've learned from nL

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

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta) \quad \begin{cases} \nabla_{v_c} J \\ \nabla_{u_w} J \end{cases}$$

$$J = -\log P(o|c) = -u_o^T v_c + \log \sum_{w=1}^{V} \exp(u_w^T v_c)$$

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

$$\nabla_{v_c} J = -u_o + \sum_{w=1}^{V} P(w|c) \cdot u_w$$

$$\Rightarrow \begin{cases} \nabla_{u_o} J = (P(o|c) - 1) \cdot v_c \\ \nabla_{u_w} J = P(w|c) \cdot v_c \end{cases}$$

$$\Rightarrow \begin{cases} u_w = u_w - \alpha (P(w|c) \cdot v_c) \\ u_o = u_o - \alpha (P(o|c) - 1) \cdot v_c \end{cases}$$

v_c : center vec

u_o : Outside (context) vec

u_w : context vec

$\nabla_{v_c} = \frac{\partial}{v_c}$

chain rule : $\frac{\partial f}{\partial v_c} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial v_c}$

$\star \quad \frac{\partial}{\partial v_c} u_x^T v_c = u_x$

$(u_1 v_1, u_2 v_2, \dots)$ $(u_1, u_2, \dots)$

$z = \sum_{w=1}^{V} \exp(u_w^T v_c)$, also $z = \exp(u_o^T v_c) + \sum_{w \neq o} \exp(u_w^T v_c)$

$\nabla_{u_o} J = -v_c + \frac{1}{z} \cdot \nabla_{u_o} z$

$= -v_c + \frac{\exp(u_o^T v_c)}{z} \cdot v_c$

we need 2 times for chain rule

1: to caculate $\log\left[\sum_{w=1}^{V} \exp(u_w^T v_c)\right]$ (f, g)

2: to caculate $\sum_{w=1}^{V} \exp(u_w^T v_c)$ (f, g)

Why $J(\theta)$ called loss ???

1. we want $L(\theta)$ (likelihood) as big as possible
2. To simplify calculation, we use minus logarithm
3. think that we want something to be as small as possible

   loss happen to correspond with this concept

   也就是说损失刚好符合这个概念

What's more?

because minus logarithm is mathematically equivalent to cross entropy

(when distribution $p(y|x)$ is one-hot)

$$p(y|x) = \begin{cases} 1 & \text{if } y = y^* \\ 0 & \text{otherwise} \end{cases}$$

$$H(p,q) = -\sum_y p(y|x)\log q(y|x) = -\log q(y^*|x)$$

So we also compute cross Entropy to get the loss

1. **起点：One-Hot表示**
   - **想法**：最简单的表示，一个词就是一个长度为词汇表$|V|$的向量，只有一位是1。
   - **问题**：这导致了**维度灾难**和**语义鸿沟**。模型无法理解“猫”和“狗”有什么关系。
2. **核心突破：词嵌入**
   - **想法**：我们不把词看作孤立的符号，而是用一**个低维、稠密的向量**来表示它。核心假设是：**相似上下文的词有相似的语义。**
   - **实现方法：Word2Vec**
     - 这是一个**无监督**算法，目标不是解决具体任务，而是**学习词嵌入本身。**
     - 它通过 **Skip-gram** 或 **CBOW** 架构，让模型学习用中心词预测上下文（或反之），从而在向量空间中捕捉语义关系。
   - **至此，我们获得了语言的“原子”：有意义的词向量。**