Gradient Descent (we never use)

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

problem: Calculating our objective function over all of our data for our model (即 training data for the model) would take very very long time

Stochastic (随机) Gradient Descent → SGD (always use for neural net)

↓

also referred to Mini Batch Gradient Descent

Repeatly sample windows, and update after each one

pick a very small subset of data

this is a noisy, inaccurate estimate of the gradient

虽然 SGD 每一步看上去是在"瞎猜"，但在统计意义上是正确的，这些 noisy gradient 的"长期平均方向"，正好就是批量梯度下降的准确方向

---

example : "the quick brown fox jumps" window size = 1 (center, context)

(quick, the) (quick, brown) (brown, quick) (brown, fox)

(fox, brown) (fox, jumps) (jumps, fox)

Traditional GD

1. input all the sample into model
2. caculate the average of the loss of all sample
3. caculate the gradient that loss to all parameters
4. update

SGD

1. pick first sample (brown, fox) randomly, calculate this sample's loss to all parameters then update.

PS: 虽然依旧是对 all parameters，但在例如 U the 之类的参数上，梯度会非常小或为 0

main gradient is focus on the Ubrown and Ufox

2. pick second sample (quick, the) randomly

3. Continue like this ....

(Bow)

Bag of words — Count word occurences, ignore order

↓

LSA — reduct dimension for Bow

↓

Glove (2014)

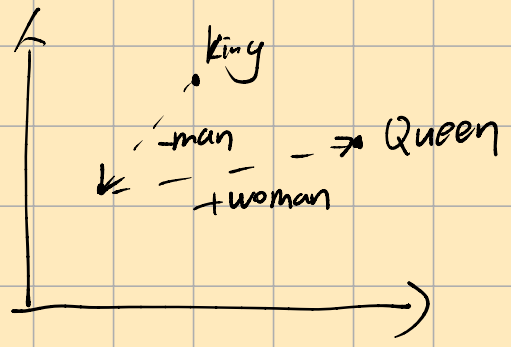
↓

Word2vec (2013) { skip-gram (→ SG) (跳字) — center word predict context

CBOW — context predict center word

↓

(Continue Bag of words)



analogy (类比)

Why have 2 vectors?

In math:

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

1 vec : $P(o|c) = \frac{\exp(w_o^T w_c)}{\sum_{w=1}^{V} \exp(w_w^T w_c)}$ ← Both 分子 & 分母 have "w_c"

Obviously, it's very complex if we want to calculate the particul derivative

① skip-gram model with negative sampling (负采样)

why use it ?

Cause in Softmax, it takes long time to calculate the value of $\sum_{w=1}^{V} \exp(u_w^T v_c)$, The value of V is very big

what is the idea ?

Maximize probability of real outside world

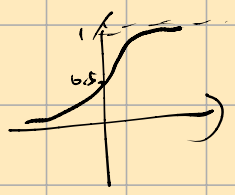
Minimize probability of random words

formula ?

make bigger ↓ make smaller ↓

$$J_{neg\text{-}sample}(u_o, v_c, U) = -\log \sigma(u_o^T v_c) - \sum_{k \in \{k \text{ sampled indices}\}} \log \sigma(-u_k^T v_c)$$

$\sigma(\cdots)$: sigmoid or logistic function



we learned from ML

$\sigma(x) = \frac{1}{1+e^{-x}}$

In this algorithm, we want Positive sample as big as possible

↓

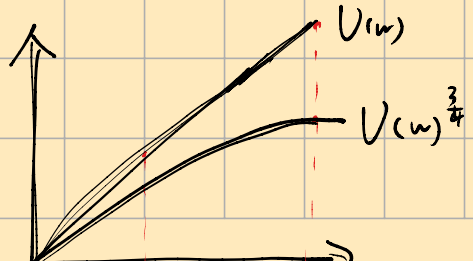
(probality of real outside world is "yes" bigger)

negative sample as big as possible too

(so we make $u_k^T v_c$ smaller to ensure $-u_k^T v_c$ bigger)

Unigram distribution (一元分布) 根据单个词的频率独立地采样

We always sample with $P(w) = U(w)^{\frac{3}{4}}/Z$, the unigram distribution $U(w)$ raised to the $\frac{3}{4}$ power



obviously, raising up the probability of the less frequent word

why raise to $\frac{3}{4}$ power?

make sampling more uniformly, do not overlay lean towards high-frequency words (such as 'the', 'a') [why? in math, example $\begin{cases} 10000^{0.75} \approx 5623 \\ 10^{0.75} \approx 5.6 \end{cases}$]

② Co-occurrence matrix (共计数矩阵)

eg: Window length = 1

corpus: I like you
I like him

	I	like	you	him
I	0	2	0	0
like	2	0	1	1
you	0	1	0	0
him	0	1	0	0

We find that the matrix is very high dimensional

So we want to reduce the dimensionality of that matrix (usually 25–1000 dimensions, similar to word2vec)

How to reduce?

Classic method: Singular Value Decomposition (奇异值分解) I've learned from linear algebra

$\forall$ matrix $\Rightarrow X = U \Sigma V^T$

1. U & V are orthonormal
2. $\Sigma = \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & s_3 & \\ & & & \ddots \\ & & & & s_n \end{bmatrix}$ $s_1 > s_2 > s_3 > \cdots > s_n$ (It rules that s order in size)

then, we set the smallest singular vectors to 0

③ Global Vectors for Word Representation (GloVe model)

Crucial insight: Ratio of co-occurrence probabilities can encode meaning components

We want to capture them as linear meaning components in a word vector space

eg:

$\frac{P(x\|ice)}{P(x\|steam)}$	x = solid	x = gas	x = water / random
	large	small	1

this means if the ratio is large or small, X can distinguish ice and steam

if the ratio is 1, X can't distinguish them

How can we capture the ratios as linear meaning components?

using Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$ ← We assume it (cause dot product also represent the probability)

with vector differences: $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$ ← co-occurrence probability

$$\begin{cases} w_x \cdot w_a = \log P(x|a) \text{ ①} \\ w_x \cdot w_b = \log P(x|b) \text{ ②} \end{cases} \quad ① - ②$$

Loss: $J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$

④ how to evaluate word vectors?

A general concept of evaluation (in NLP)

Intrinsic (内部)

Evaluation on a specific / intermediate task

{ word vector distances
  human judgement
} to evaluate meaning similarity

Extrinsic (外部)

Evaluation on real task

one example could help evaluate named entity recognization (NER)

⑤ Word senses and word sense ambiguity

problem: most word have lots of meaning

how to solve?

Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters

↳ but we don't usually do

what's we usually done?

=> Different senses of a word reside in a linear superposition (叠加) (weighted sum 加权平均)

in standard word embeddings like word2vec

eg: $v_{pike} = \alpha_1 v_{pike_1} + \alpha_2 v_{pike_2} + \alpha_3 v_{pike_3}$

where $\sigma_i = \frac{f_i}{f_1+f_2+f_3}$ f ← frequency

⑥ Cross entropy

a concept from information theory

$$H(p,q) = -\sum_{c=1}^{C} p(c) \log q(c)$$

P : true, objective probability distribution

q : the probability distribution your model've predicted

if p and q are same, the cross-entropy will be small

if differ significantly the cross-entropy will be large

A special case:

when p is One-hot coding ($p=[0,1,0,0,1\cdots]$)

$$H(p,q) = -\sum_{c=1}^{C} p(c) \log q(c)$$ ← p(c) will be 0 except the only right one

$$= -\log q(C_r)$$ C_r is the right class which model predict

the essence of cross entropy is:

To measure the different between two probability distributions

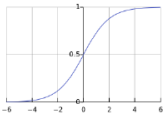
⑦ neural net work

NER: Binary classification for center word being location

- We do supervised training and want high score if it's a location

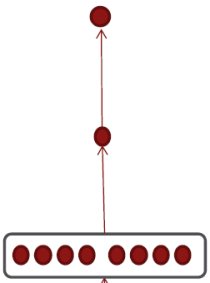
$$J_t(\theta) = \sigma(s) = \frac{1}{1+e^{-s}}$$

predicted model probability of class

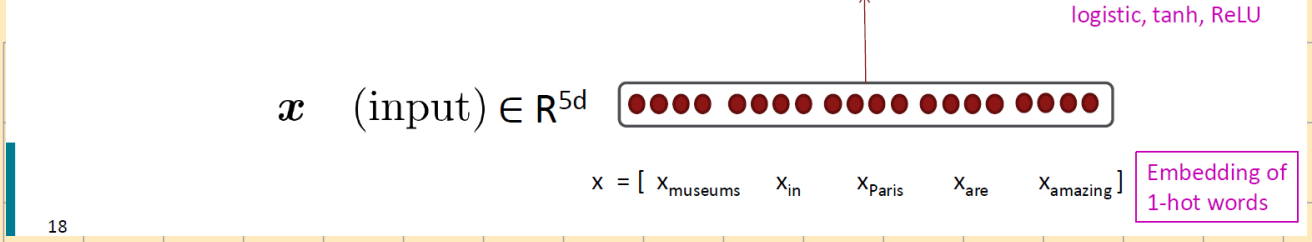


$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$



f = Some element-wise non-linear function, e.g.,


logistic, tanh, ReLU
$\boldsymbol{x}$ (input) $\in R^{5d}$
x = [$x_{museums}$ x_{in} x_{Paris} x_{are} $x_{amazing}$]
Embedding of 1-hot words