→ have a mathematical relationship with sparse embedding like PPMI)
↳ can be seen as optimization of PPMI)

→ SGNS

<u>Word2Vec</u> (Framework for learning word vectors

↳ embeddings : short dense vectors
  ↳ work better
    ↳ classifier requires to learn far fewer weights
    ↳ smaller parameter space possibly helps with generalization and avoid overfitting
    ↳ dense vector ⇒ capture synonyms better.
↳ fast, easier to train., <u>static embeddings</u>
  ↳ learns one fine embedding for each word.
↳ simpler model
  ↳ making binary prediction instead of prediction
↳ instead of counting how often each word $w_i$ occurs near a word $w$, train classifier to check how likely word $w_i$ will show up with $w$
↳ Don't actually care about prediction task. instead we take the learned classifier weighs as the word embeddings
↳ Use running text as implicitly supervised training data (<u>self supervision</u>)
  → 1st proposed in neural LM
  ↳ avoid the need of hand labeled supervision again.
↳ intuition (skip gram)
1) Treat target and a neighbouring words as +ve example
2) Sample other words randomly to get -ve sample
3) Use logistic regression.
4) Use learned weight as embedding

Sample with $P(\omega) = \underset{\substack{\uparrow \\ \text{unigram distribution}}}{U(\omega)^{3/4}} / Z$ ⇒ increasing the probability of less frequent words

$P(+|\omega, c)$ , $P(-|\omega, c) = 1 - P(+|\omega, c)$

$\Downarrow$ $\quad\quad$ $\Downarrow$

$\sigma(c \cdot \omega)$ $\quad\quad$ $\sigma(-c\omega)$

↳ makes the assumption that all context words are independent

$$\log P(+|\omega, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot \omega)$$

$$\sum \log \sigma(c_i \omega)$$

↳ store two embedding for each word
- ↳ as a target (input embedding)
- ↳ as a context (output embedding)

common add together $\omega_i = \omega_i + c_i$ throw away

↳ Learning skip-gram embeddings

→ Assigning random embedding vector for each of the N vocabulary word.

→ Iteratively shift the value to be more like embedding of ~~a~~ words that occur nearby and less like that don't occur nearby.

⇒ SGNS uses more -ve examples than +ve's (K being the ratio)

⇒ -ve example ⇒ noise words ⇒ choosen using $P_\alpha(\omega)$ $\quad \alpha = 0.75$

$$L = -\left[\log \sigma(c_{pos} \cdot \omega) + \sum_{i=1}^{K} \log \sigma(-c_{neg_i} \cdot \omega)\right]$$

loss function

Goal : 1) maximize similarity for $(c_{pos}, \omega)$

2) minimize similarity for $(\omega, c_{neg})$

$\boxed{\text{t-SNE}}$ → projection method ⇒ visualization of embeddings

→ minimizing loss function using stochastic gradient descent

→ tf-idf ⇒ choise of doc length, term frequency affects the performance

III[ly], word2vec ⇒ content window size L affects the performance. — tune it on devset

<u>Other kind of static embedding</u>

↳ fasttext ⇒ extension of word2vec ⇒ no good way to deal with unknown words

⇓

deals with languages with morphology using subword model

eg $n=3$ cohere

→ <wh, whe, her, ere, re>

↑ special boundary symbol

for each sub word a embedding is learned and the org word will be represented as the sum of all the embeddings

↳ GloVe ⇒ Global vectors → based of ratios of probabilities from w-w co-ocurrence matrix while also capturing linear structure used by word2vec

→ it is best to train multiple embeddings with bootstrap (randomly selecting subset) sampling over documents and average the results as embedding model suffers from inherent variability.

## Word2Vec: Objective function.

For each position $t = 1, --, T$, predict content words within a window of fixed size $m$, given center word $w_t$

Data likelihood: $L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} | w_t ; \theta)$

Objective function

$$J(\theta) = -\frac{1}{T} \log L(\theta) \rightarrow \begin{bmatrix} v_{w_1} \\ v_{w_2} \\ \vdots \\ u_{w_1} \\ u_{w_2} \end{bmatrix} \in \mathbb{R}^{2dV}$$

dimension of vectors

number of words

average

so that we dont have to deal with this enormous product

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

Use two vector per word $w$:

$V_w \rightarrow$ as a center word

$U_w \rightarrow$ as a content word

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

$\rightarrow$ calculating similarity using dot product

can be +ve or -ve

makes anything +ve

content word

center word

$\rightarrow$ which is softmax function

soft: because still assigns some probability to smaller $x_i$

max: because amplifies probability of largest $x_i$

$\rightarrow$ Different senses of words reside in a linear superposition (weighted sum) in standard word embeddings like word2Vec

$$V_w = \alpha_1 V_{w_{s_1}} + \alpha_2 V_{w_{s_2}} + \alpha_3 V_{w_{s_3}} + \dots$$

where $\alpha_1 = \frac{f_1}{f_1 + f_2 + \dots}$

Ideally after doing this we can't seperate out the individual senses but using "sparse coding" we can do it