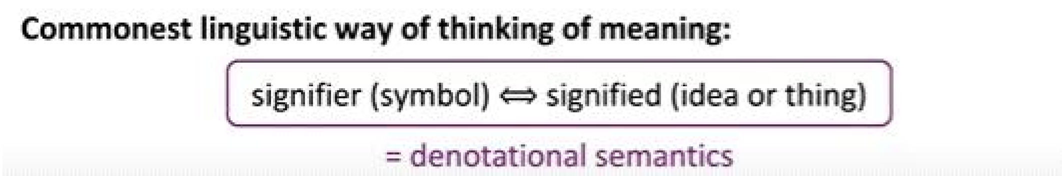
NLP Lecture 1 - Intro & Word Vectors

Key question for AI&Human-computer interaction
-> how to get computers to be able to understand the information conveyed in human languages.
-> how to build on virtuous cycle.

GPT-3: A first step on the path to universal models
-> to detect spam, pronography.. whatever
--> predicts following words!

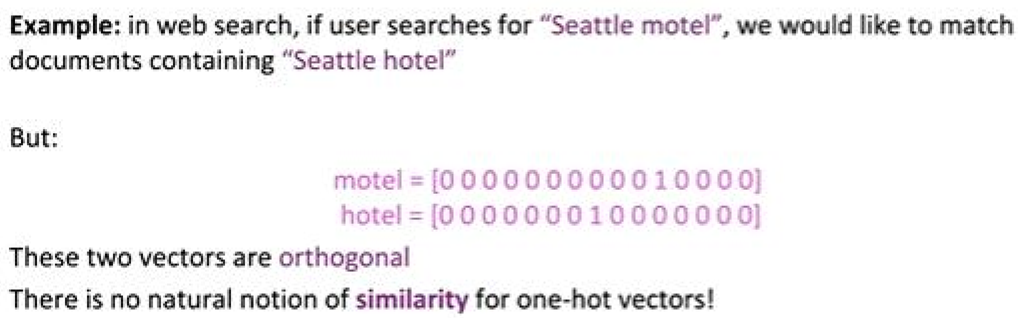
How do we represent the meaning of a word?
-> denotational semantics

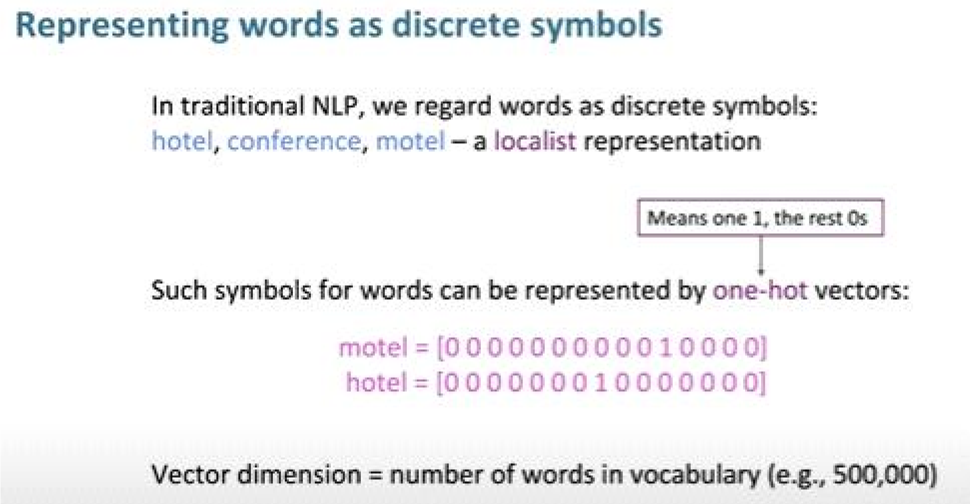


Common NLP solution: WordNet(-> which organized words and terms into both synonyms sets of words that can mean the same thing and hypernyms which correspond to ISA relationships.), a thesaurus containing lists of synonym sets and hypernyms

Problems with resources like WordNet (deficiency)
-> it lacks a lot of nuances
“proficient” is listed as a synonyms for “good” -> This is only correct in some contexts

The problem of the traditional NLP
-> words are regarded as discrete symbols.

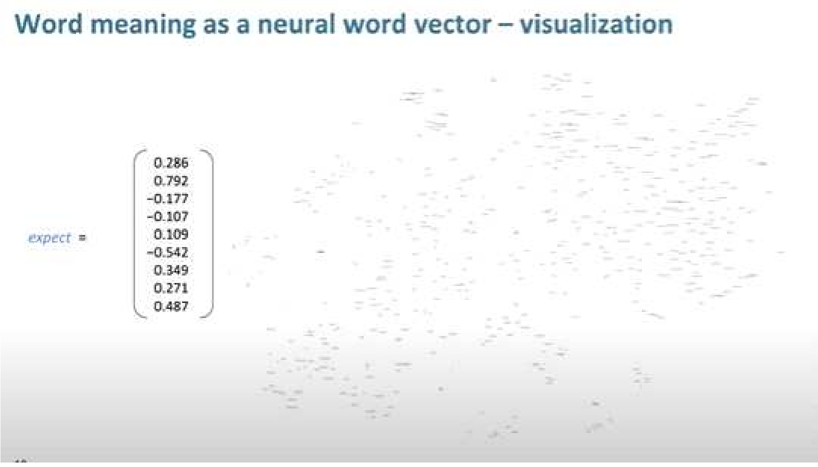




one-hot 인코딩 방식으로는 “Seattle motel”과 “Seattle hotel” 두 단어 간 유사성을 파악할 수 없음.

-Solution
 -> WordNet의 list에 의존? -> fail badly: incompleteness
 -> Instead, learn to encode similarity in the vectors themselves. --> Distributional semantics

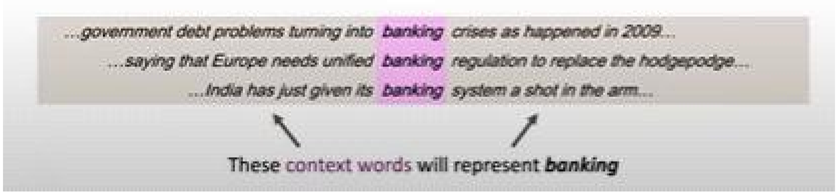
Distributional semantics: A word’s meaning is given by the words that frequently appear close-by (문장 속 특정 단어 앞 뒤(주변)에 위치하는 단어들 <- 그 단어를 설명하는, 유사한 단어일 것이라는 아이디어!



Word embedding

Word2vec: framework for learning word vectors

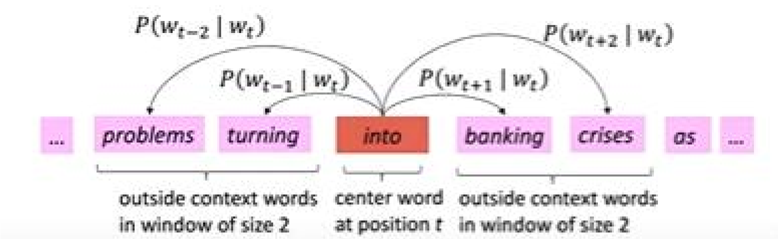
- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

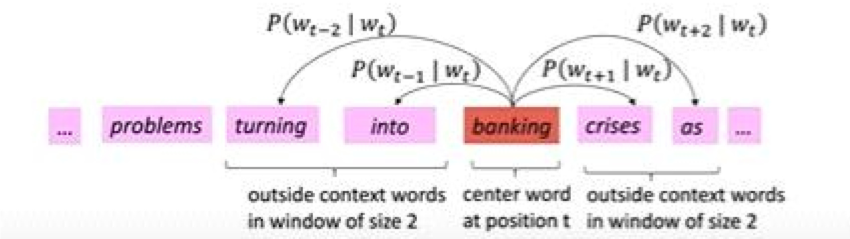


-
- large corpus("body") of text
- Every word in a fixed vocabulary is represented by a vector
- Keep adjusting the word vectors to maximize this probability
-

- Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
- Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)

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





- cost of loss function: objective function을 최소화하는 것과 예측정확도를 최대화하는 것 간의 충돌!
- dot product : o와 c의 유사성을 구하는 수학적 방법(큰 값일수록 확률 커짐)
- 이때 확률을 구해야하므로 음수값 나오지 않게, similarity 강조위해 exponentiate & 확률은 1 이하의 값이 나와야하므로 normarliztion(-> use softmax function)

Word2vec: objective function

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

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

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:

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

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

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

24

- loss minimize 위해 derivation 사용해 모든 vector gradients를 점진적으로 구함.

개념적으로 “observed - expect”

Word2vec: prediction function

② Exponentiation makes anything positive

① Dot product compares similarity of o and c.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

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

③ Normalize over entire vocabulary
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
  - Frequently used in Deep Learning

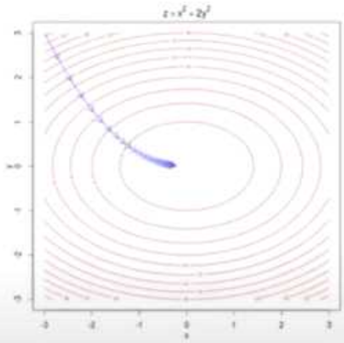
But sort of a weird name because it returns a distribution!

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

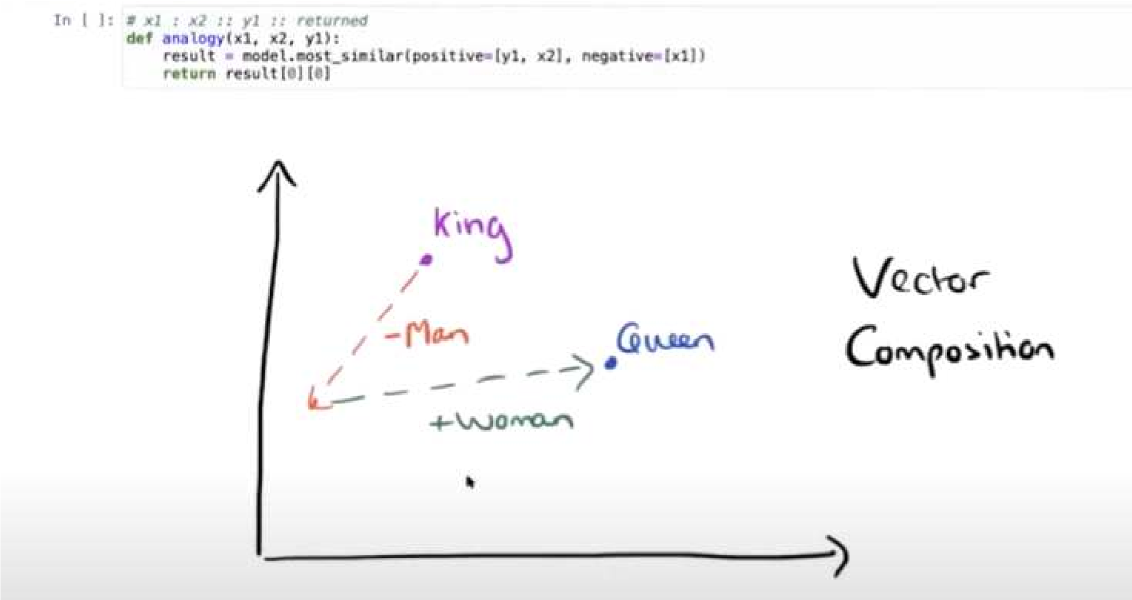
To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d-dimensional vectors and V-many words, we have:
- Remember: every word has two vectors

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



- We optimize these parameters by walking down the gradient (see right figure)
- We compute all vector gradients!



analogy function - word2vec 원리 이용, 유사한 단어를 예측하는 알고리즘