Neural Word Representation

MLP LAB 임경태

Contents

01. Word Representation

Word Representation

- Symbolic한 문자를 연산 가능한 숫자로 표현하는 방법

Word Representation

- Symbolic한 문자를 연산 가능한 숫자로 표현하는 방법
- 단순히 숫자로만 표현하면 될까? 사람은 어떨까? “Tree”라는 단어를 떠올려보자

• signifier (symbol) ⟺ signified (idea or thing)

= denotational semantics

tree ⟺ { 🌳 , 🌲 , 🌴 ,…}

- 단어를 표현할 때 단어의 **의미(semantics)**를 내포해야 하겠네?

Ref: Starford cs224n

Word Representation(one-hot)

- One-hot 은 단어의 의미(semantics)를 표현하고 있을까?

  - 단어사전 V = {cat, fat, mat, sat, the, on}
  - cat = [1, 0, 0, 0, 0, 0]
  - fat = [0, 1, 0, 0, 0, 0]
  - mat = [0, 0, 1, 0, 0, 0]
  - sat = [0, 0, 0, 1, 0, 0]
  - the = [0, 0, 0, 0, 1, 0]
  - on = [0, 0, 0, 0, 0, 1]

사람: “The fat cat sat on the mat”

컴퓨터: “32 832 561 634 6132 32 565”

?!??!

  - |V|의 크기가 매우 크므로 매우 큰 차원의 sparse vector가 필요
  - 단어의 의미가 포함되어 있을까??

Ref: 자연어처리바이블

One-hot encoding의 문제점

- One-hot은 local representation으로 단어의 의미를 표현하지 못한다

Example: in web search, if a 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 0 1 0 0 0 0]
hotel = [0 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:

- Could try to rely on WordNet's list of synonyms to get similarity?
  - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

Ref: Starford cs224n

해결책

- Dimensionality: Sparse한 표현을 Dense한 표현으로 변경해서 차원을 줄인다
- 의미의 내포: 유사한 단어는 유사한 벡터 값으로 표현하게 한다

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

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 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:

motel = [0.34 -1.22 3.31]
hotel = [0.30 -1.01 3.31]

now "motel" and "hotel" is similar

Ref: Starford cs224n

의미의 내포: 유사한 단어는 유사한 벡터 값으로 표현하게 한다

이거 어떻게함?

주변 Context를 고려한 word embedding

Representing words by their context

- Distributional semantics: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

The goal of this lecture

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

banking = $\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$

monetary / *화폐의* = $\begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$

Note: word vectors are also called (word) embeddings or (neural) word representations They are a distributed representation

Ref: Starford cs224n

One-hot VS. Word embedding

	원-핫 벡터	임베딩 벡터
차원	고차원	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습
값의 타입	1과 0	실수

02. What the word embedding is learning?

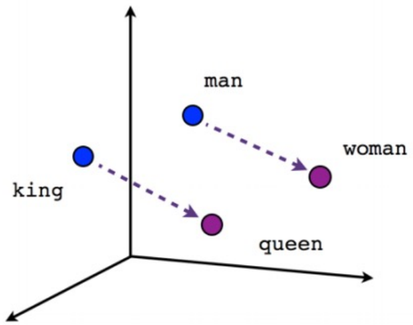
그런데 word embedding에

정말 단어의 의미가 표현 되는 거야?

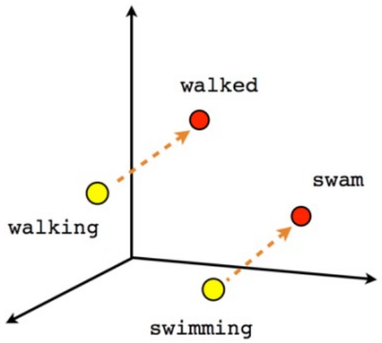
Investigation of Word Embedding

- 의미의 내포: 유사한 단어는 유사한 벡터 값으로 표현하게 한다

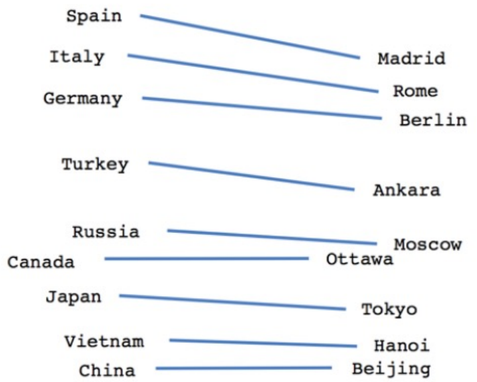
  = **‘비슷한 위치에서 등장하는 단어들은 비슷한 의미를 가진다’** (Distributional Hypothesis)



Male-Female



Verb tense



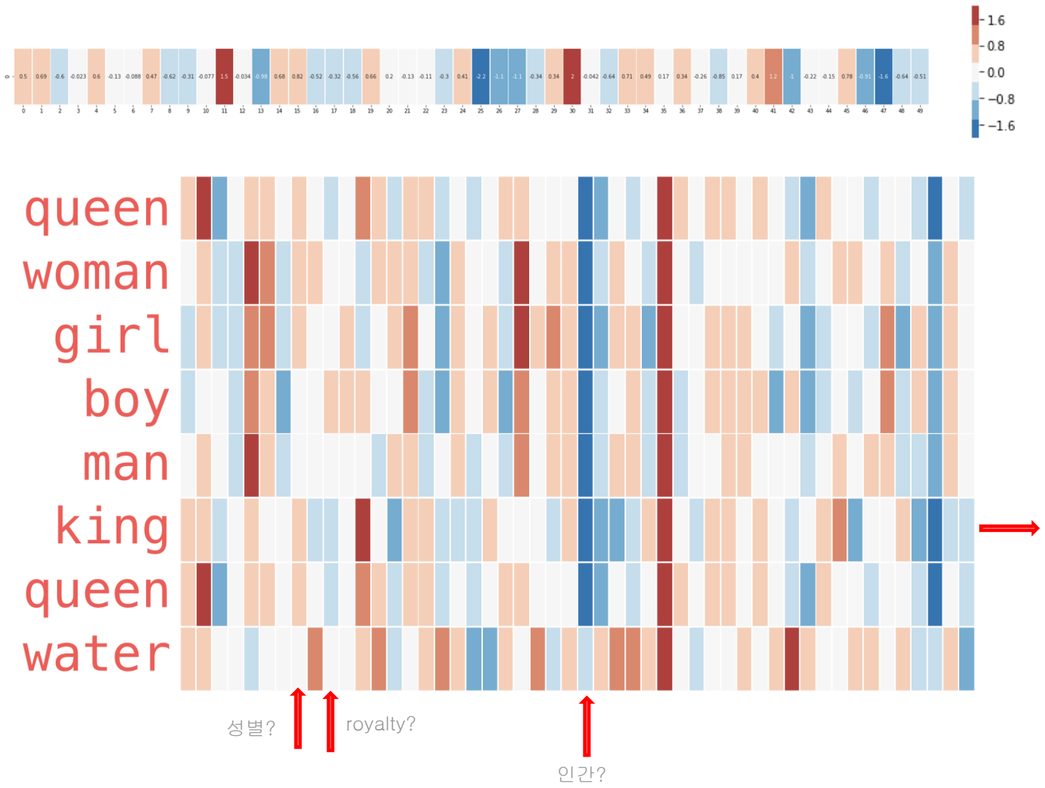
Country-Capital

실습 http://w.elnn.kr/search/

Investigation of Word Embedding

- 학습이 완료된 word embedding의 숫자는 무엇을 배운 결과인가?

  - 각 차원이 나타내는 특징을 명확하게 알 수 없지만 비교를 통해 guessing할 수 있음.



[관측포인트]
1. 각각의 컬럼이 어떤 의미를 함축하고 있나 알 수 없음
2. Man과 boy의 유사성과 woman과 girl의 유사성
3. Water와 비교 시 사람을 함축하는 컬럼
4. Royalty로 가정할 수 있는 컬럼이 있음

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 ,
-0.076666, 1.493 , -0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961
, -0.13495 , -0.11476 , -0.30344 , 0.41177 , -2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 ,
-0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -0.8523 , 0.1661 , 0.40102 , 1.1685 ,
-1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

참조 https://jalammar.github.io/illustrated-word2vec/

이제 word embedding을 학습하는

알고리즘만 알면 되겠네?

03. Word2Vec

Word2Vec Overview

- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

Idea:

- We have a large corpus (“body”) of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- 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)
- Keep adjusting the word vectors to maximize this probability

Ref: Starford cs224n

Word2Vec Overview

- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors
- There are two different type of Word2Vec

Skipgram: 중간에 있는 단어로 주변 단어들을 예측하는 방법.

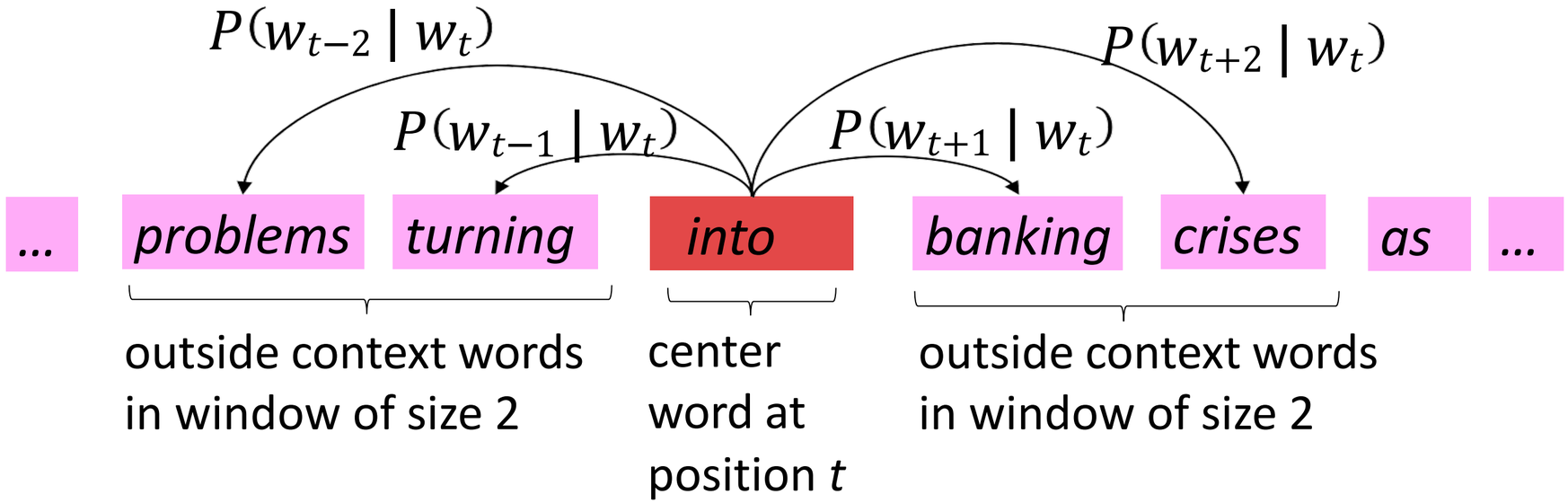
CBOW(Continuous Bag of Words): 주변에 있는 단어들을 가지고, 중간에 있는 단어들을 예측하는 방법

Ref: Starford cs224n

Word2Vec Overview (Skipgram)

- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors

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

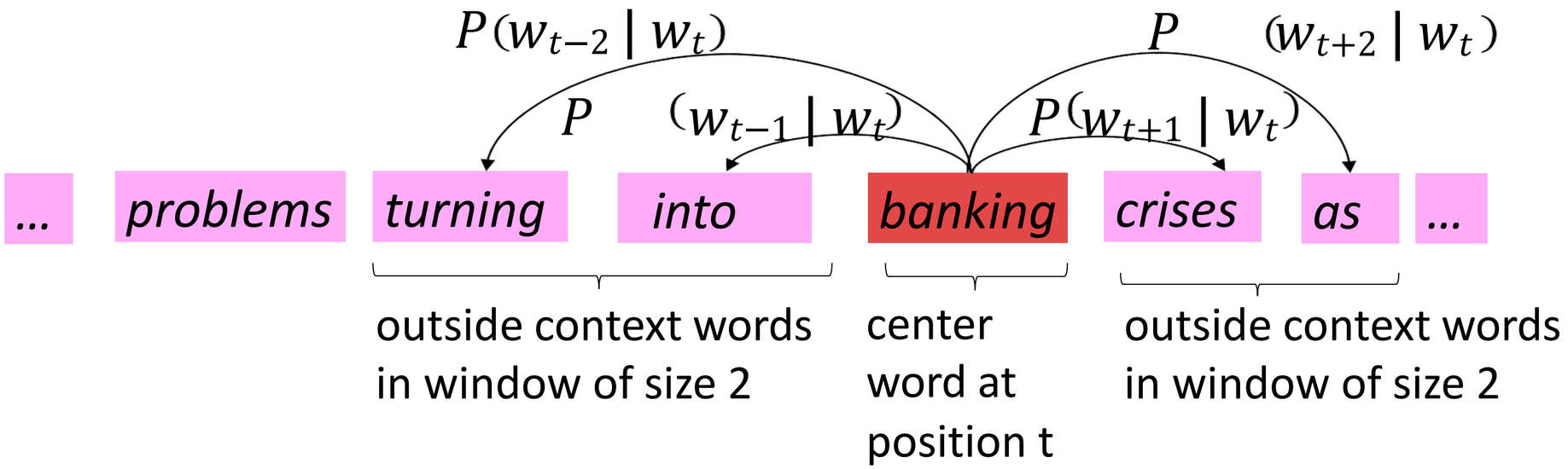


Ref: Starford cs224n

Word2Vec Overview (Skipgram)

- Sliding Window with the window size 2

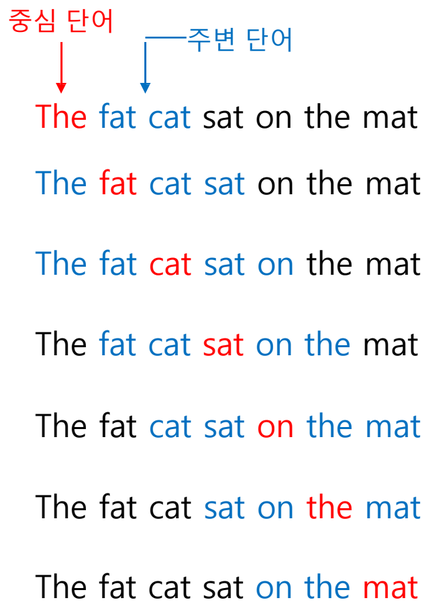
Example windows and process for computing $P\left(w_{t+j} \mid w_t\right)$



Ref: Starford cs224n

Word2Vec Overview (Skipgram)

- An example of training Data with the window size 2



window size=2. 좌, 우 단어 2개씩을 참고.

중심 단어	주변 단어
[1, 0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0]	[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]

Ref: Starford cs224n

Word2Vec Objective Function

- Word2vec is trained based on Negative Log Likelihood (NLL)

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

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

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:

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

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Ref: Starford cs224n

Word2Vec Objective Function

- Let's minimize our objective function!

  - We want to minimize the objective function:

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

  - **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*:

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

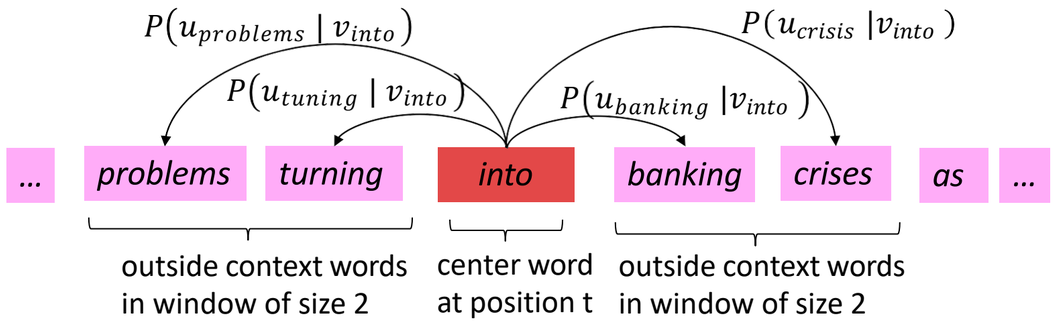
Ref: Starford cs224n

Word2Vec Objective Function

- Let's minimize our objective function!

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $P(problems \mid into\,; u_{problems}, v_{into}, \theta)$

All words vectors θ appear in denominator



Ref: Starford cs224n

Word2Vec Objective Function

- Let's minimize our objective function!

② Exponentiation makes anything positive

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

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

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!

Ref: Starford cs224n

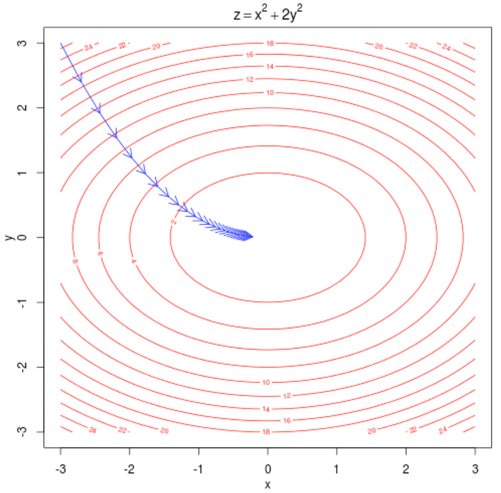
Word2Vec Objective Function

- Let's optimize our objective function!

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

- Recall: θ represents **all** the model parameter 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!

Ref: Starford cs224n

Word2Vec Structure (the Skipgram model)

- Let's Train the Skipgram word embedding

(a) 문서 D로부터 사용된 단어를 모두 추출하여

ont-hot encoding w_t를 만든다.

(b) 최종적으로 우리가 학습할 word embedding 파라메터 V

(그림 V, Center word vector)를 랜덤 초기화함,

(c) scoring을 위한 파라메터 **U**

(아래그림 빨간네모 3개짜리, Context word vector) 를 랜덤 초기화함.

(d) Forward연산: w_t*V = vc (워드임베딩),

vc*U = uc (워드임베딩*스코어링 parm),

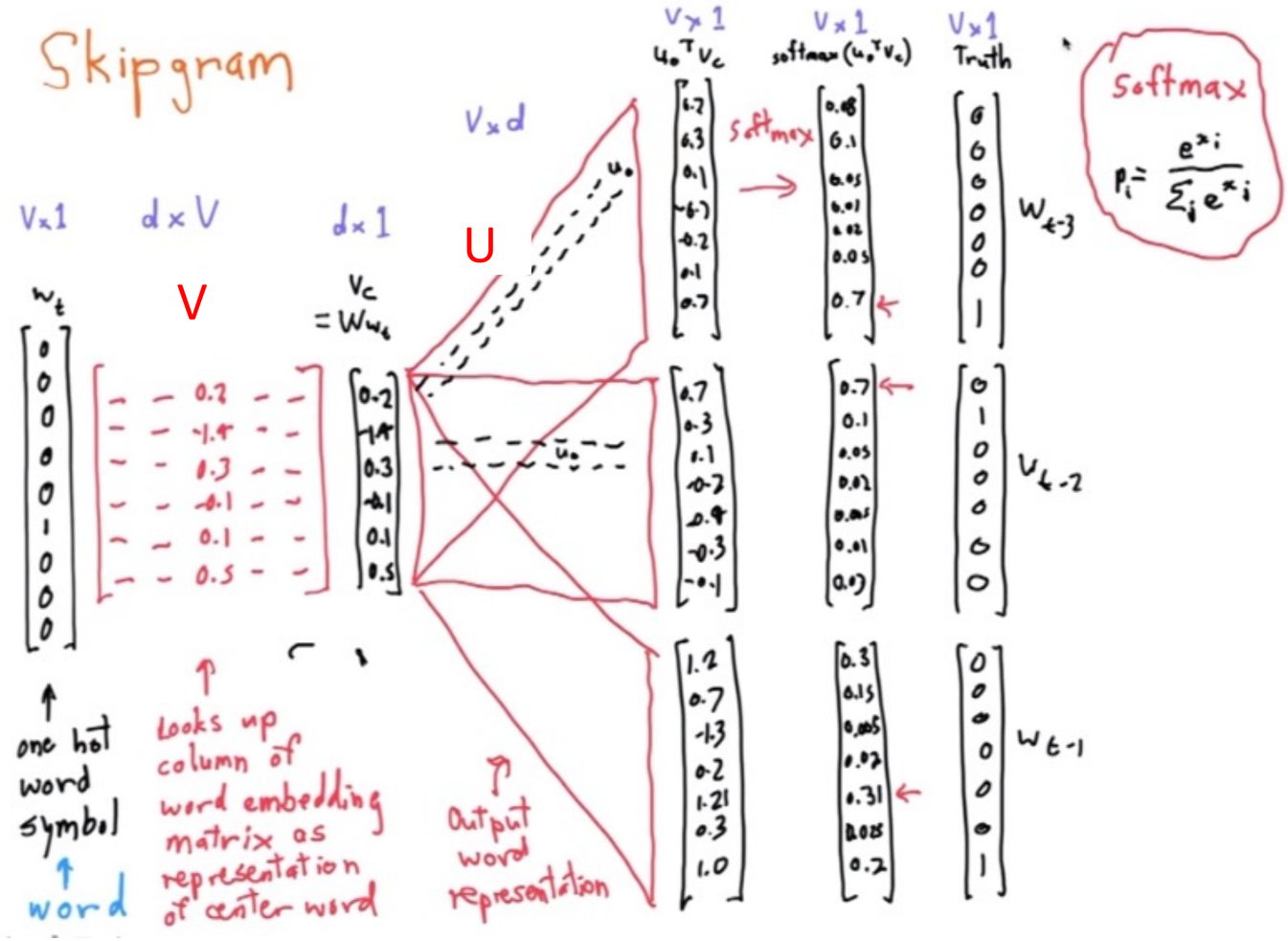
softmax(uc) = y' (*는 벡터곱)

(e) 손실값 구함: **Cross-Entropy**사용

(f) Backward연산: Gradient Decent계산

(g) 파라메터 업데이트: Gradient Decent의 결과에 따라

파라메터 V, U 업데이트.

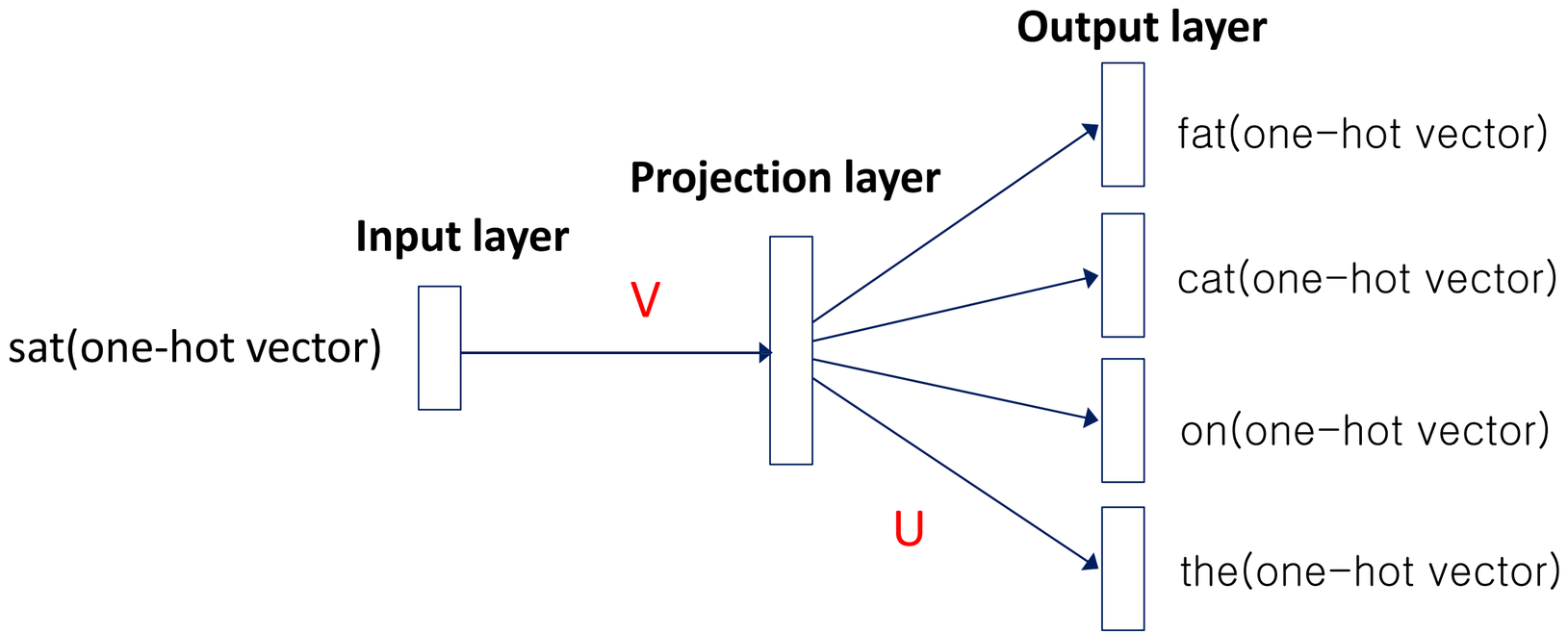


Ref: Starford cs224n

Word2Vec Structure (the Skipgram model)

- A simplified structure of the Skipgram

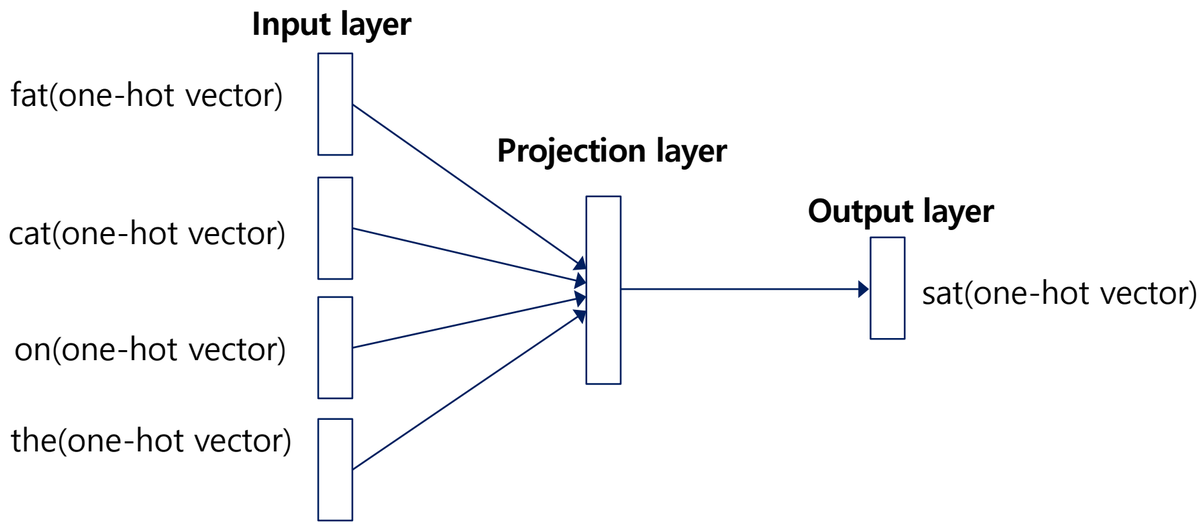
https://blog.naver.com/jujbob/221147997064



Ref: 딥러닝을 이용한 자연어처리 입문

Word2Vec Structure (the CBOW model)

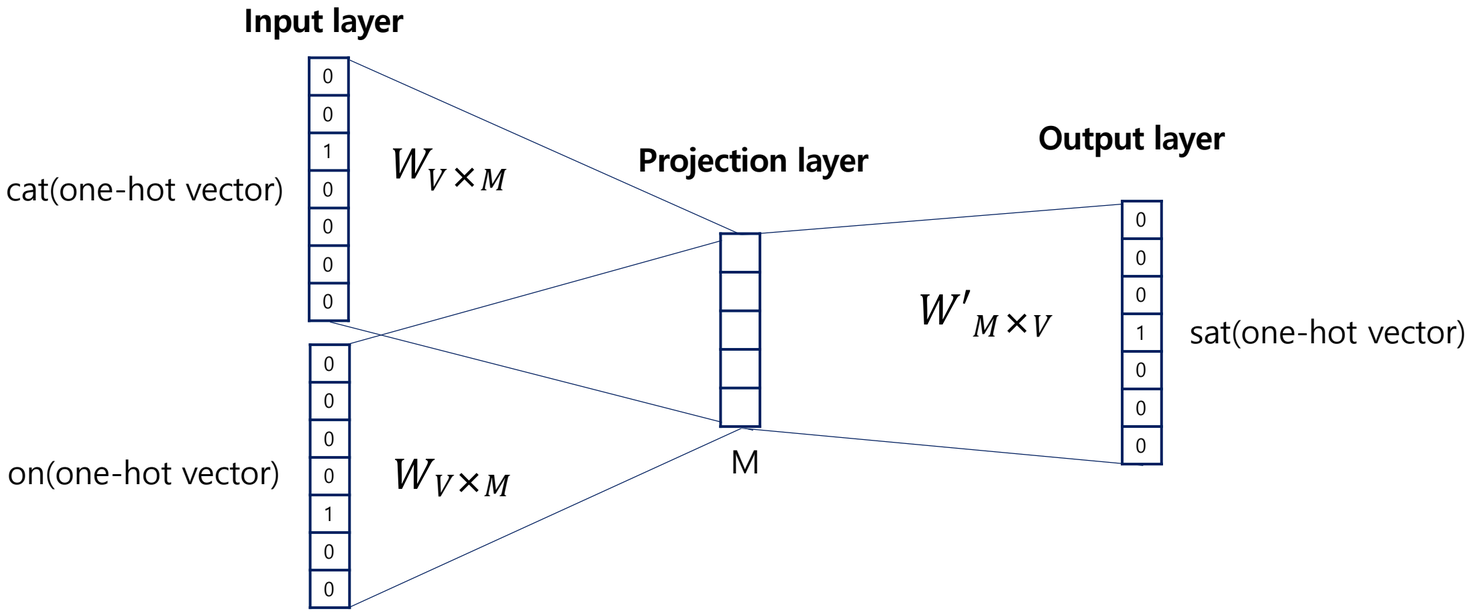
- A simplified structure of the CBOW



Ref: 딥러닝을 이용한 자연어처리 입문

Word2Vec Structure (the CBOW model)

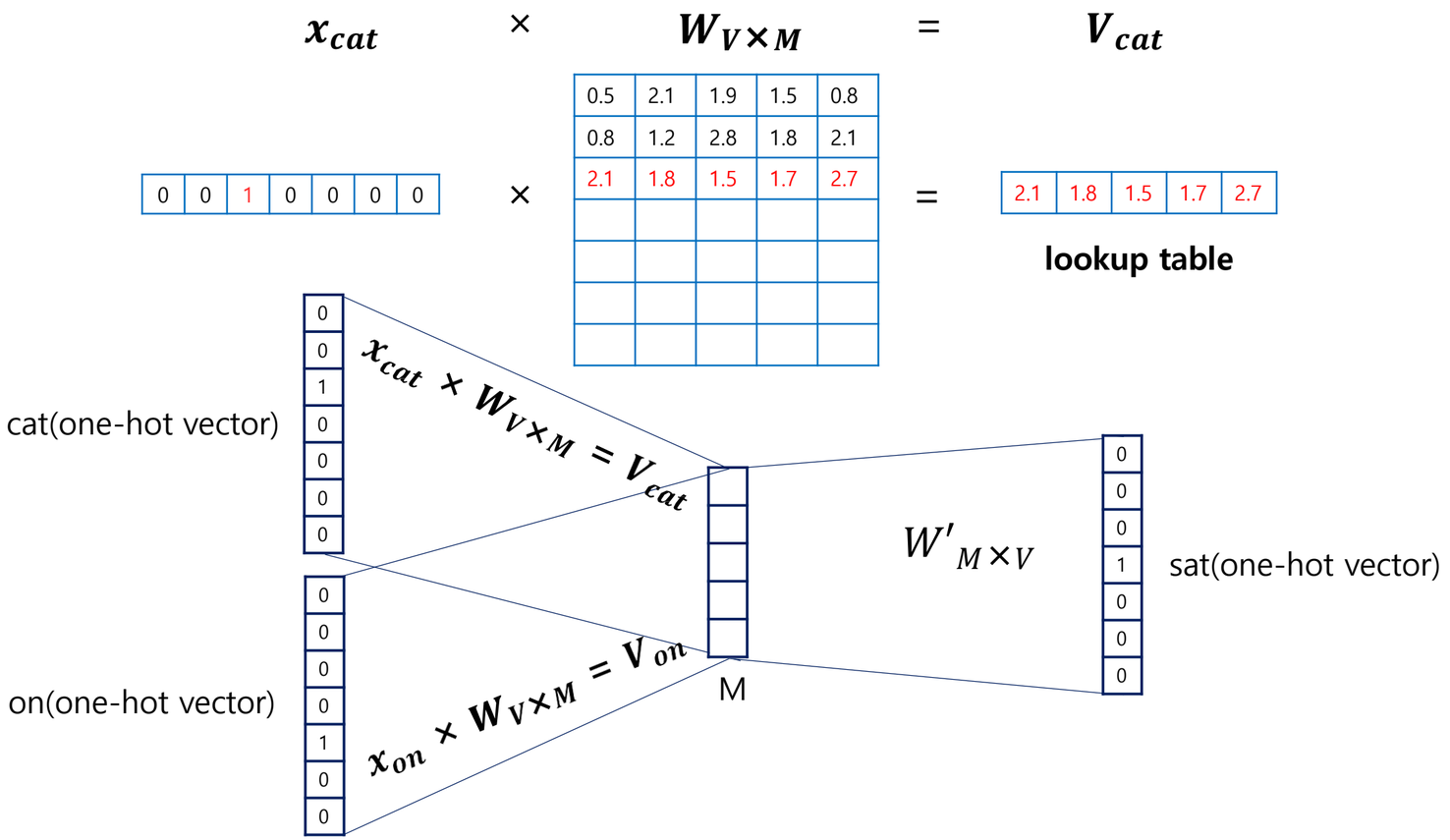
- A simplified structure of the CBOW



Ref: 딥러닝을 이용한 자연어처리 입문

Word2Vec Structure (the CBOW model)

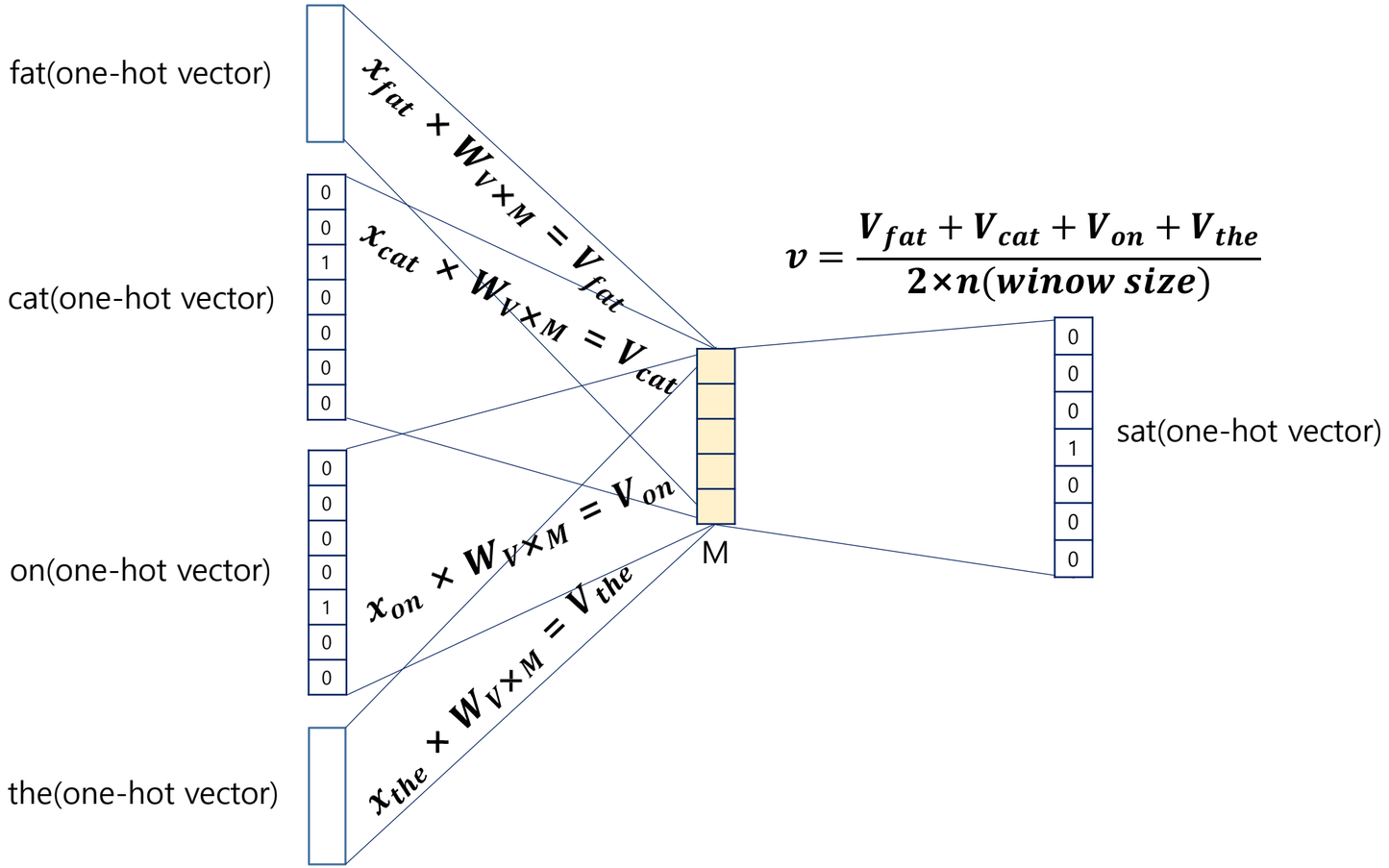
- A simplified structure of the CBOW



Ref: 딥러닝을 이용한 자연어처리 입문

Word2Vec Structure (the CBOW model)

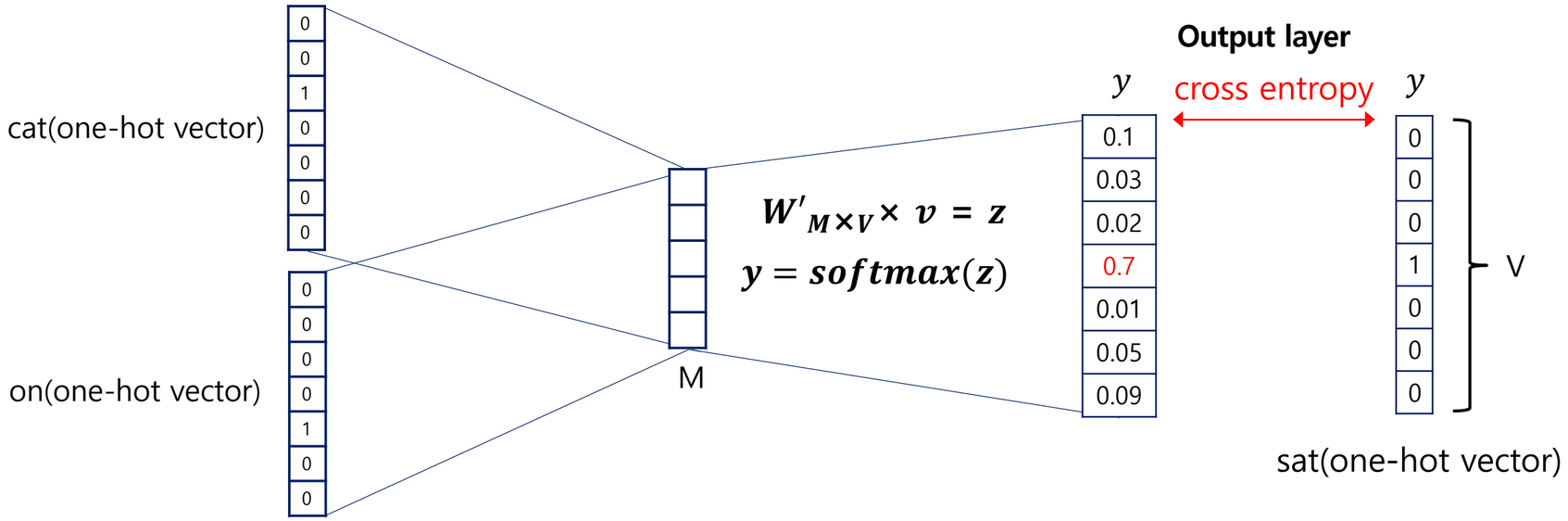
- A simplified structure of the CBOW



Ref: 딥러닝을 이용한 자연어처리 입문

Word2Vec Structure (the CBOW model)

- A simplified structure of the CBOW



Ref: 딥러닝을 이용한 자연어처리 입문

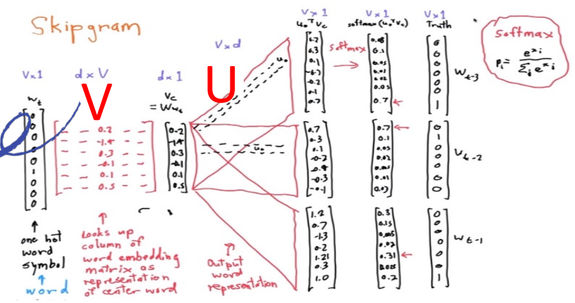
04. Applying Word Embeddings

How to use word embeddings?

- (1) Pretraining on plain texts and then save parameter V as a file
- (2) Load the saved file and use the vector as like tfidf!

(1) 저장
a.emb

(2) 다른 NLP 시스템에 불러와서 사용



	음주운전	사고	가해자	강력	처벌	역주행	사건	집행유예	면허	취소	규정
음주운전	1	0	0	0	0	0	0	0	0	0	0

One-Hot Encoding Vector

*

	Dimension (D)							
음주운전	0.23	0.43	0.32	0.67	0.29	0.56	0.45	0.31
사고	...	...	...	...	...	...	...	...
가해자	...	...	...	...	...	...	...	...
강력	...							...
처벌	...							...
역주행	...							...
사건	...							...
집행유예	...							...
면허	...	...	...	...	...	...	...	...
취소	...	...	...	...	...	...	...	...
규정	...	...	...	...	...	...	...	...

학습 대상

Weight Matrix

=

	Dimension (D)							
음주운전	0.23	0.43	0.32	0.67	0.29	0.56	0.45	0.31

Word Embedding Vector

Ref: 파이토치딥러닝프로젝트

How to train Word2Vec?

- We can use well-coded libraries like Genism or Fasttex.. etc

```
1 from gensim.models import Word2Vec
2
3 embedding_model = Word2Vec(df_drop['token_final'],
4                            sg = 1,
5                            size = 1000,
6                            window = 2,
7                            min_count = 1,
8                            workers = 4)
9
10 print(embedding_model)
11
12 model_result = embedding_model.wv.most_similar("음주운전")
13 print(model_result)
```

```
Word2Vec(vocab=30036, size=1000, alpha=0.025)
[('살인죄', 0.8733304738998413), ('형량', 0.8702058792114258), ('상해',
0.8652387261390686), ('음주', 0.8635096549987793), ('고의',
0.8585751056671143), ('의료사', 0.8552800416946411), ('벌금형',
0.8549568057060242), ('살인', 0.8530592918395996), ('강간',
0.852802038192749), ('전과', 0.849102258682251)]
```

Ref: 파이토치딥러닝프로젝트

How to save the trained Word2Vec?

- We can use well-coded libraries like Genism or Fasttex.. etc

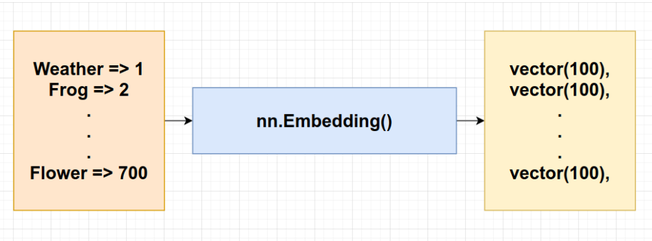
```
from gensim.models import KeyedVectors

embedding_model.wv.save_word2vec_format
('/content/drive/MyDrive/GitHub/MachineLearning/PythorchDeepLearningProject/국민청원 분류하기/petitions_tokens_w2v')
loaded_model = KeyedVectors.load_word2vec_format
('/content/drive/MyDrive/GitHub/MachineLearning/PythorchDeepLearningProject/국민청원 분류하기/petitions_tokens_w2v')

model_result = loaded_model.most_similar("음주운전")
print(model_result)
```

```
[('살인죄', 0.8733304738998413), ('형량', 0.8702058792114258), ('상해',
0.8652387261390686), ('음주', 0.8635096549987793), ('고의',
0.8585751056671143), ('의료사', 0.8552800416946411), ('벌금형',
0.8549568057060242), ('살인', 0.8530592918395996), ('강간',
0.852802038192749), ('전과', 0.849102258682251)]
```

Ref: 파이토치딥러닝프로젝트

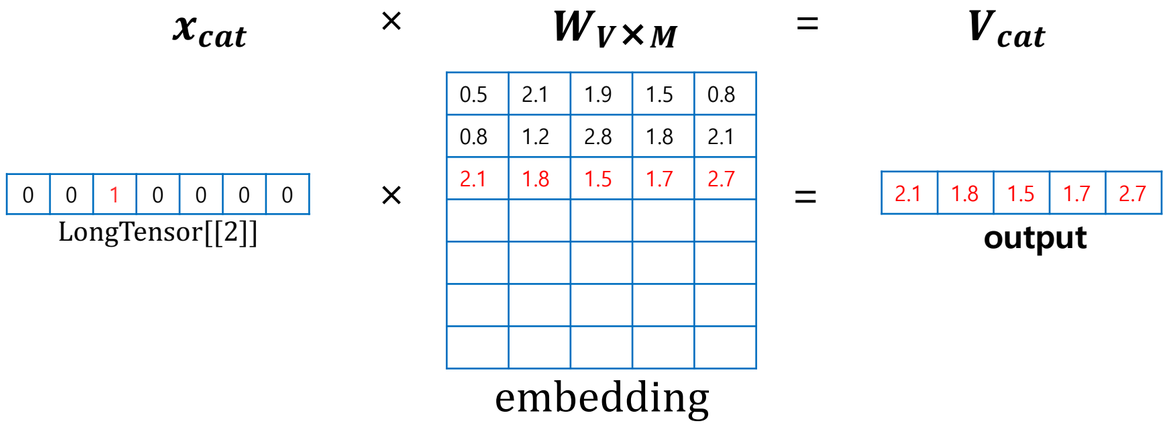
How to load the trained Word2Vec on Pytorch??

- Pytorch 에서는 nn.Embedding 모델을 제공함.



  - nn.Embedding(전체단어수, 단어의차원) 와 같이 사용.
  - emb = nn.Embedding(10, 3)을 하면 10개의 단어에 대해 3차원의 단어 벡터를 랜덤으로 생성
  - emb([0, 2]) 하면 0번째 단어 (3차원 벡터), 2번째 단어(3차원벡터) 벡터들이 반환됨.
  - 해당 벡터를 이용해 NLI모델을 학습 (과제)

```
>>> # an Embedding module containing 10 tensors of size 3
>>> embedding = nn.Embedding(10, 3)
>>> # a batch of 2 samples of 4 indices each
>>> input = torch.LongTensor([[1,2,4,5],[4,3,2,9]])
>>> embedding(input)
tensor([[[-0.0251, -1.6902,  0.7172],
         [-0.6431,  0.0748,  0.6969],
         [ 1.4970,  1.3448, -0.9685],
         [-0.3677, -2.7265, -0.1685]],

        [[ 1.4970,  1.3448, -0.9685],
         [ 0.4362, -0.4004,  0.9400],
         [-0.6431,  0.0748,  0.6969],
         [ 0.9124, -2.3616,  1.1151]]])
```



How to load the trained Word2Vec on Pytorch??

- 심화 실습

  - 네이버 영화리뷰 데이터로 Word2vec을 학습하고 embedding을 파일로저장

  - 저장된 파일을 읽어 랜덤으로 초기화된 nn.Embedding에 넣어 초기화함

  - NLI문제에 도입

05. Deep Contextualized Word Embedding

이건 다음에…

06. ELMO, BERT
이건 다음에…

감사합니다.