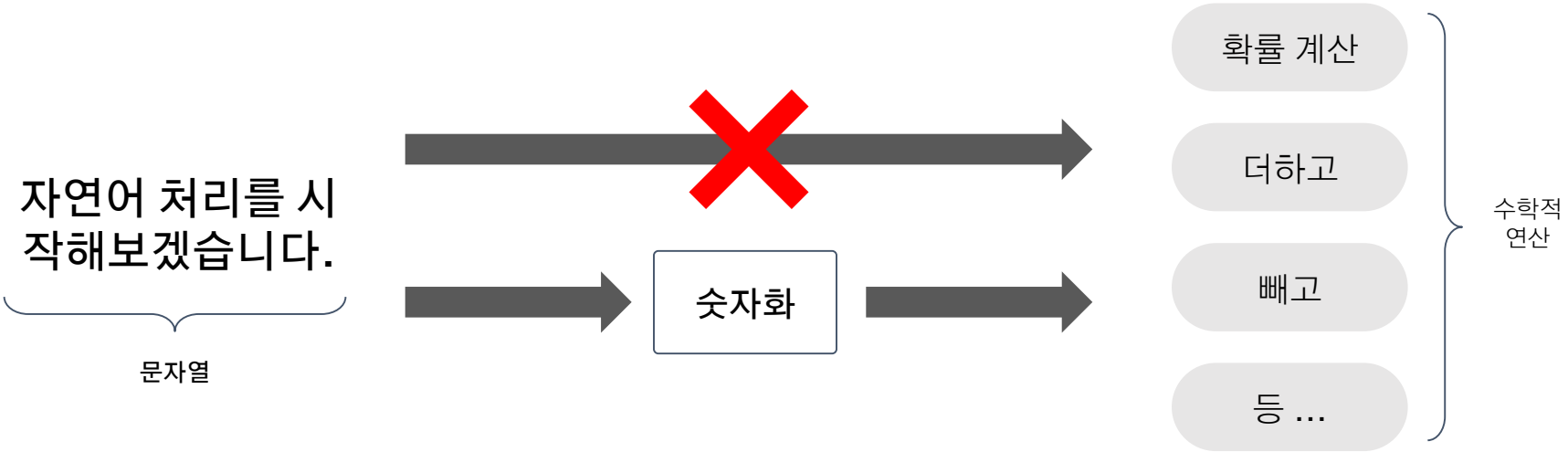
단어 임베딩 (Word Embedding)

자연어처리 텍스트마이닝

단어의 표현이 필요한 이유

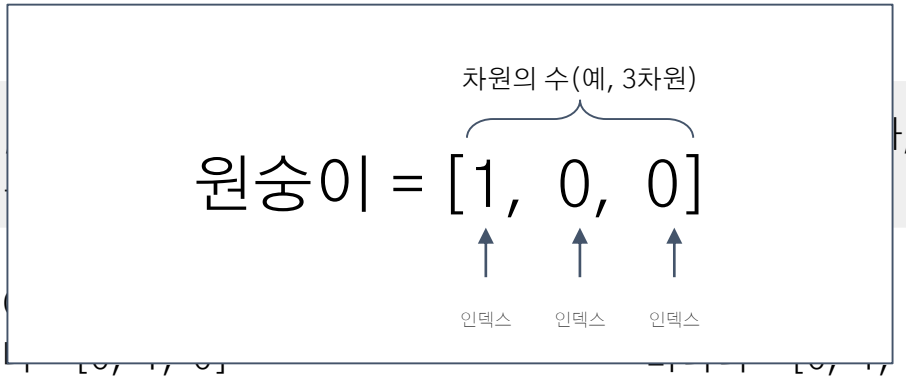


원핫-인코딩
(One-Hot-Encoding)

단어의 표현(Word Representation)

원핫-인코딩(One-Hot-Encoding)

원핫-인코딩은 단어(word)를 숫자로 표현하고자 할 때 적용할 수 있는 간단한 방법론



원숭이 = [1, 0, 0]
바나나 = [0, 1, 0]
사과 = [0, 0, 1]

원숭이 = [1, 0, 0, 0]
바나나 = [0, 1, 0, 0]
사과 = [0, 0, 1, 0]
코끼리 = [0, 0, 0, 1]

원핫-인코딩(One-Hot-Encoding) 한계점

1) 차원 크기의 문제

원숭이, 바나나, 사과를
표현할 때

원숭이 = [1, 0, 0]
바나나 = [0, 1, 0]
사과 = [0, 0, 1]

단어의 수만큼 차원이 필요함

단어수가 많아진다면?

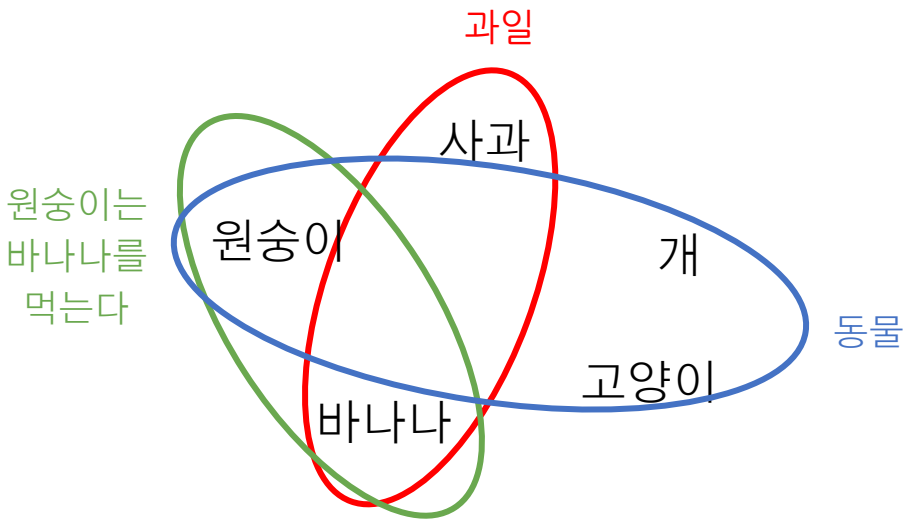
2017년 표준국어대사전에 등재된 단어 수 약 50만개
=> 50만개의 차원이 필요

원숭이 = [1, 0, 0, 0, 0, 0, 0, 0, 0, ... 0, 0, 0, 0, 0]

50만 차원 벡터

원핫-인코딩(One-Hot-Encoding) 한계점

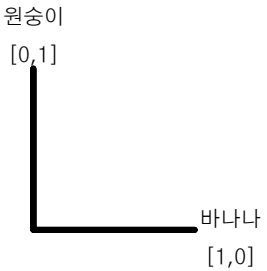
2) 의미를 담지 못하는 문제



원핫-인코딩(One-Hot-Encoding) 한계점

2) 의미를 담지 못하는 문제

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$



$$\text{similarity} = \frac{(0 \times 1) + (1 \times 0)}{(1^2 + 0^2) \times (0^2 + 1^2)} = 0$$

원핫-인코딩(One-Hot-Encoding) 한계점

2) 의미를 담지 못하는 문제

원숭이, 바나나, 사과, 개, 고양이를 표현할 때

원숭이 = [1, 0, 0, 0, 0]
바나나 = [0, 1, 0, 0, 0]
사과 = [0, 0, 1, 0, 0]
개 = [0, 0, 0, 1, 0]
고양이 = [0, 0, 0, 0, 1]

- “원숭이, 사과” 코사인 유사도 : 0
- “원숭이, 바나나” 코사인 유사도 : 0
- “개, 고양이” 코사인 유사도 : 0

=> 원핫 벡터간 코사인 유사도는 모두 0

=> 따라서 의미를 분간 하기 어려움

단어 임베딩
(Word Embedding)

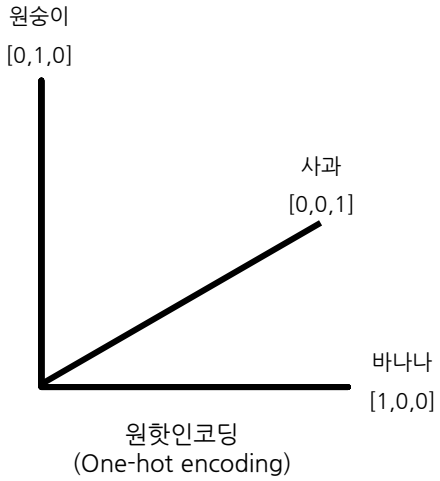
원핫-인코딩(One-Hot-Encoding) 한계점

벡터로 표현한 단어 차원이 너무 큼	연산이 낭비되어 모델 학습에 불리하게 적용
단어 의미를 담지 못함	분석을 효과적으로 수행할 수 없음

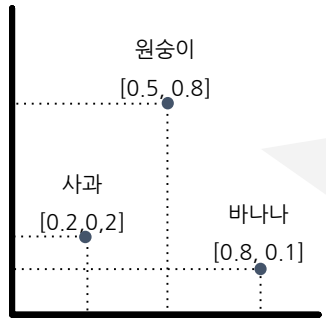
단어 임베딩(Word embedding)

단어 임베딩은 단어의 의미를 간직하는 밀집 벡터(Dense Vector)로 표현하는 방법

원숭이, 바나나, 사과를 표현할 때



원핫인코딩
(One-hot encoding)



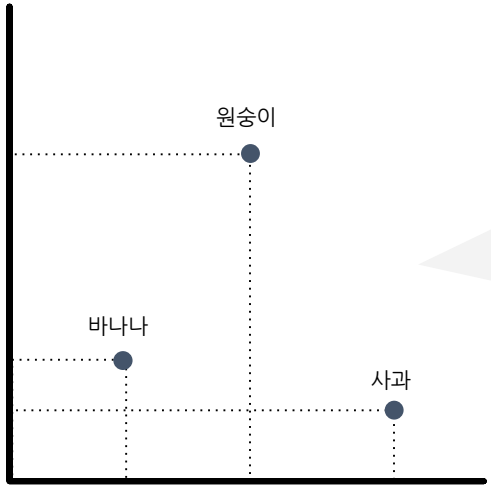
밀집 벡터
(Dense Vector)

- 벡터가 공간에 꽉차 있음
- 새로운 단어 추가시 차원을 추가할 필요가 없음
  => 차원을 줄일 수 있음
  => 추후 분류나 예측 모델을 학습할 때 연산을 줄일 수 있는 이점을 갖음

단어 임베딩 (Word Embedding)의 한계

벡터로 표현한 단어 차원이 너무 큼

밀집 벡터(Dense vector)로 해결



사과 벡터는 어디에 표현되는 것이 맞을까요?

=> 단어를 벡터로 표현하는 명확한 방법이 존재하지 않음

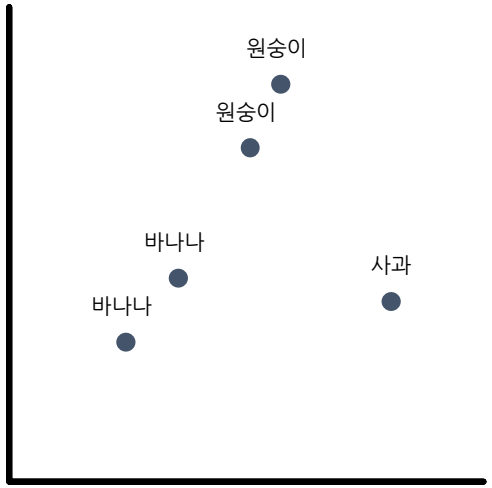
단어 의미를 담지 못함

?

밀집 벡터를 만드는 방법

분포 가설이란,

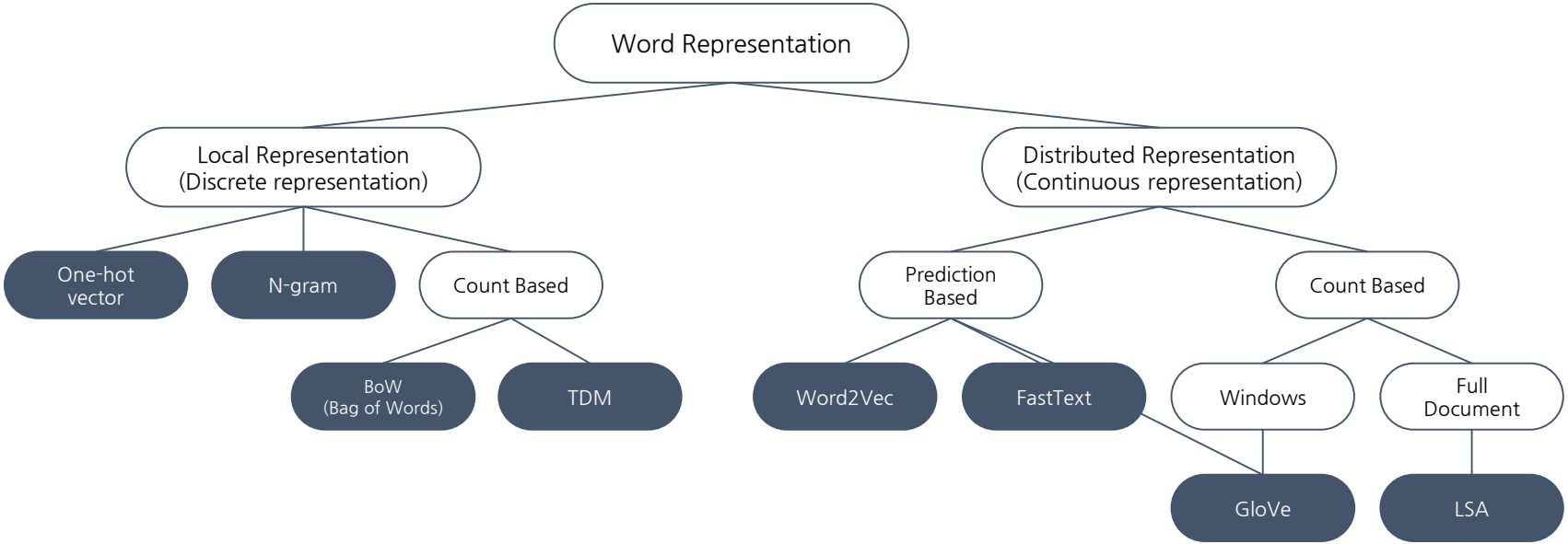
'같은 문맥에서 등장하는 단어는 유사한 의미를 지닌다'



1) 임의의 위치에 벡터 생성

2) 같은 문맥이 등장하는 단어를 더 가까이 표현

Word Representation



- Local representation (Discrete representation) : 해당 단어 그 자체만 보고 값을 매핑하여 표현
- Distributed representation (Continuous representation) : 단어를 표현하기 위해 주변을 참

NPLM
(A Neural Probabilistic Language Model)

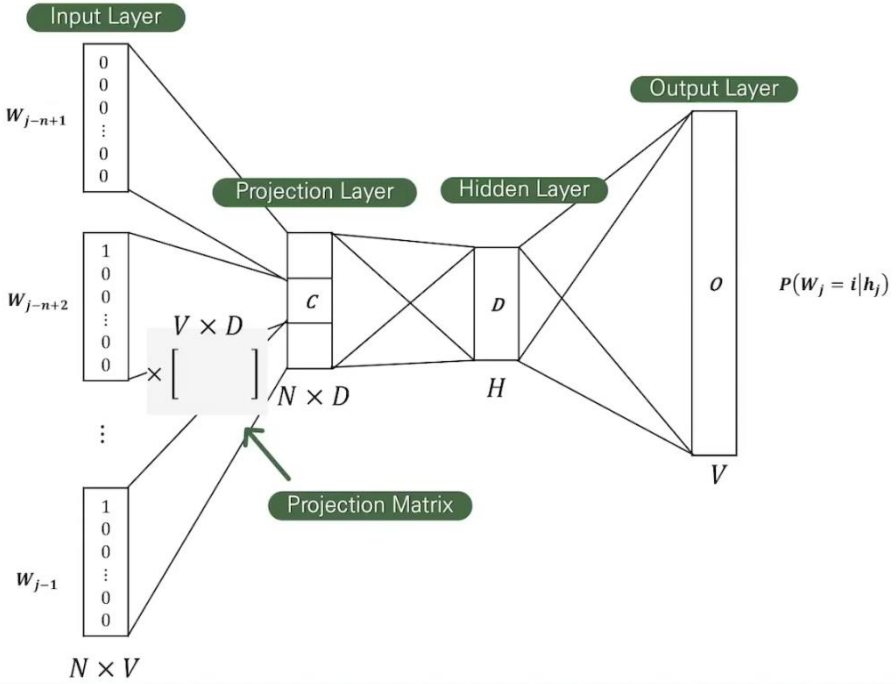
https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

NPLM

- 통계 기반의 전통적인 모델의 한계를 극복 (n-gram 모델)
- 기존 언어 모델의 문제점
  - 학습 데이터에 존재하지 않는 n-gram이 포함된 문장이 나타날 확률 값이 0, 이를 위해 back-off나 smoothing이 제안 되었으나 한계
  - 문장의 장기 의존성을 포착하기 어렵다. n-gram을 5 이상 하기 어려움, n이 커질 수록 등장 확률이 0인 단어 다수 존재
  - 단어/문장 간 유사도를 계산 할 수 없다
- 한국어 임베딩 p115

NNLM(Feedforward Neural Net Language Model)

- 단어의 벡터화라는 개념 도입
- 단어 시퀀스가 주어졌을 때 다음 단어가 무엇인지 맞추는 과정



NPLM

- **한계**
  - 몇 개의 단어를 볼 건지에 대한 파라미터 N이 고정되어 있고, 정해주어야 한다.
  - 이전의 단어들에 대해서만 신경쓸 수 있고, 현재 보고 있는 단어 앞에 있는 단어들을 고려하지 못한다
  - 가장 치명적인 단점으로, 느리다. (N=10, D=500, H=500 인경우 O(50억))

Word2Vec

(Efficient Estimation of Word Representations in Vector Space)

https://arxiv.org/abs/1301.3781

한국어 임베딩 p121

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

- 단어의 연속적 벡터 표현의 2가지 모델을 제안
- 단어 유사성으로 이 벡터 표현의 질을 측정
- 더 적은 비용으로 높은 정확도(accuracy)를 개선

Introduction

Many current NLP systems and techniques treat words as atomic units – there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons – simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. … With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. Probably the most successful concept is to use distributed representations of words . For example, neural network based language models significantly outperform N-gram models

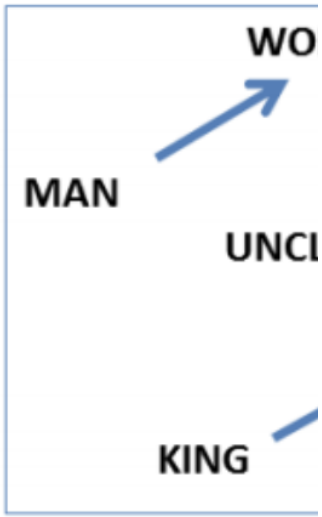
- 단어를 원자 단위(=원핫 인코딩)로 보기 때문에 단어간 유사성에 대한 고려가 없음
- 많은 양(huge) 데이터를 활용한 단순 모델이 적은데이터 복잡한 모델을 적용한 것보다 성능이 좋다
- 기술발전으로 많은 양 데이터를 복잡한 모델로 학습시키는 것이 가능해짐. => 이 경우 단어의 분산 표형(distributed representation)을 사용

Goals of the Paper

The main goal of this paper is to introduce techniques that can be used for learning high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary … We use recently proposed techniques for measuring the quality of the resulting vector representations, with the expectation that not only will similar words tend to be close to each other, but that words can have multiple degrees of similarity … it was shown for example that vector("King") − vector("Man") + vector("Woman") results in a vector that is closest to the vector representation of the word Queen … we try to maximize accuracy of these vector operations by developing new model architectures that preserve the linear regularities among words. We design a new comprehensive test set for measuring both syntactic and semantic regularities, and show that many such regularities can be learned with high accuracy. Moreover, we discuss how training time and accuracy depends on the dimensionality of the word vectors and on the amount of the training data.

- 수십억 단어로 질 좋은(high-quality) 단어 벡터를 학습하는 방법
- 유사단어 간에는 거리가 가까운 경향이 있고, 단어는 다양한 유사도를 가진다

의미 보존



King - Man + Woman

=

Queen

Model architecture

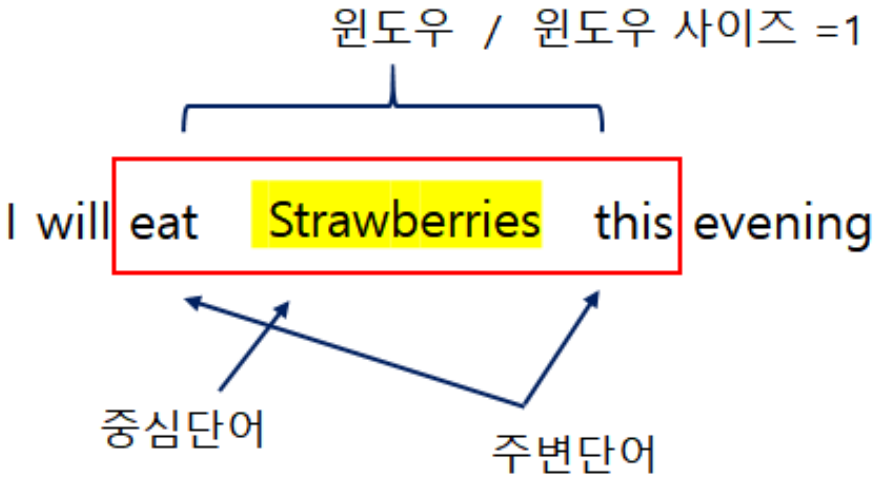
In this paper, we focus on distributed representations of words learned by neural networks, as it was previously shown that they perform significantly better than LSA for preserving linear regularities among words [20, 31]; LDA moreover becomes computationally very expensive on large data sets…For all the following models, the training complexity is proportional to

$$O = E \times T \times Q$$

where E is number of the training epochs, T is the number of the words in the training set and Q is defined further for each model architecture. Common choice is E = 3 – 50 and T up to one billion. All models are trained using stochastic gradient descent and backpropagation

- LSA보다 뛰어난 선형 정규성(Linear regularity)
- LDA는 데이터 양이 많을 수록 많은 연산을 필요로함
- Word2vec의 복잡도는 $O = E \times T \times Q$ (E : Epoch, T : 트레이닝셋 단어갯수, Q : 모델마다 별도 설정)

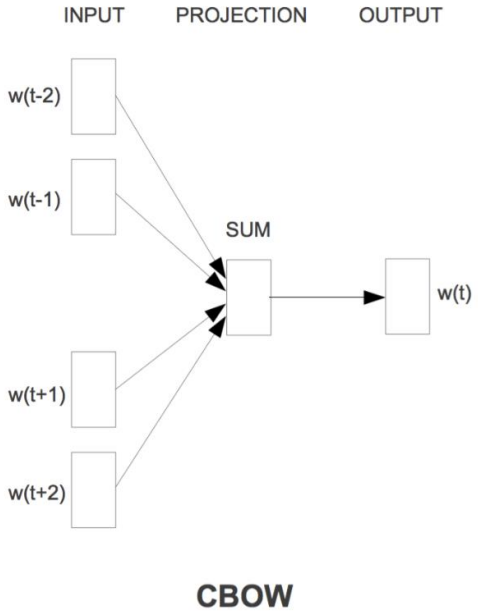
Word2Vec 용어 정리

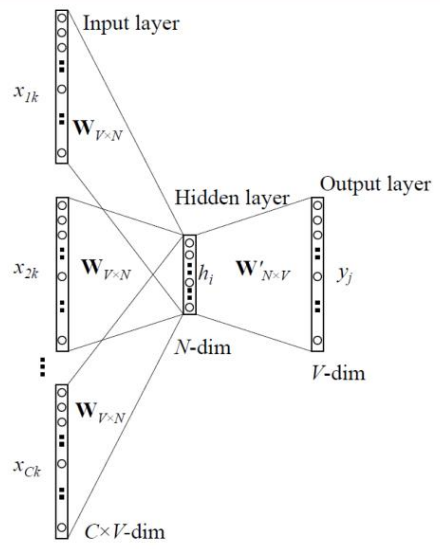


- positive sample : 중심단어(타겟단어)와 주변단어(문맥단어)와의 쌍
- negative sample : 중심단어와 주변에 등장하지 않은 단어와의 쌍

Continuous Bag-of-Words Model

- 주변단어를 사용해 중심단어를 예측



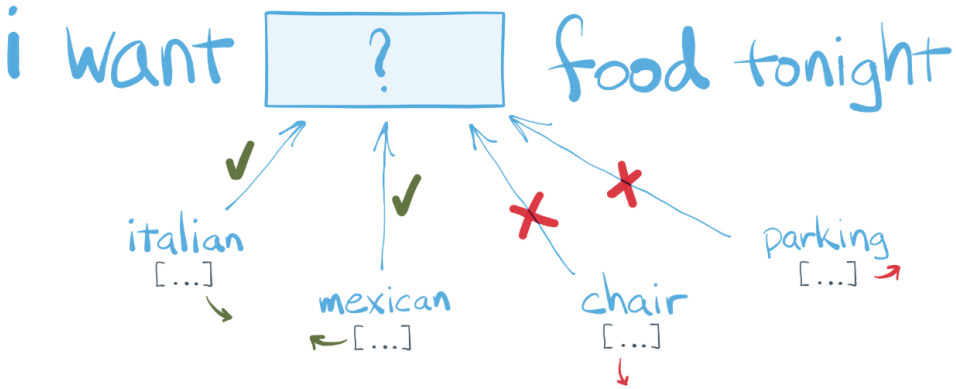


CBOW Architecture

Continuous Bag-of-Words Model

빈칸에 어떤 단어가 들어갈 수 있을까?

Continuous Bag-of-Words Model



빈칸에 들어가기 적합한 단어들과 부적합한 단어들

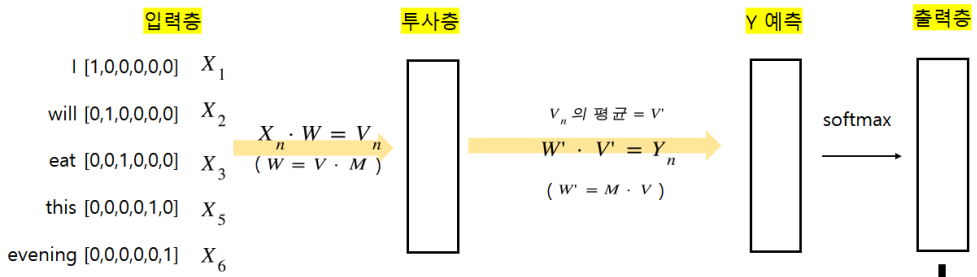
Continuous Bag-of-Words Model

	중심단어	주변단어
I will eat Strawberries this evening.	[1,0,0,0,0,0]	[0,1,0,0,0,0] [0,0,1,0,0,0]
I will eat Strawberries this evening.	[0,1,0,0,0,0]	[1,0,0,0,0,0] [0,0,1,0,0,0] [0,0,0,1,0,0]
I will eat Strawberries this evening.	[0,0,1,0,0,0]	[1,0,0,0,0,0] [0,1,0,0,0,0] [0,0,0,1,0,0] [0,0,0,0,1,0]
I will eat Strawberries this evening.	[0,0,0,1,0,0]	[0,1,0,0,0,0] [0,0,1,0,0,0] [0,0,0,0,1,0] [0,0,0,0,0,1]
I will eat Strawberries this evening.	[0,0,0,0,1,0]	[0,0,1,0,0,0] [0,0,0,1,0,0] [0,0,0,0,0,1]
I will eat Strawberries this evening.	[0,0,0,0,0,1]	[0,0,0,1,0,0] [0,0,0,0,1,0]

Continuous Bag-of-Words Model

I will eat [ ] this evening

$$V_n\text{의 평균} = V' = \frac{V(X_2) + V(X_3) + V(X_4) + V(X_5)}{4}$$



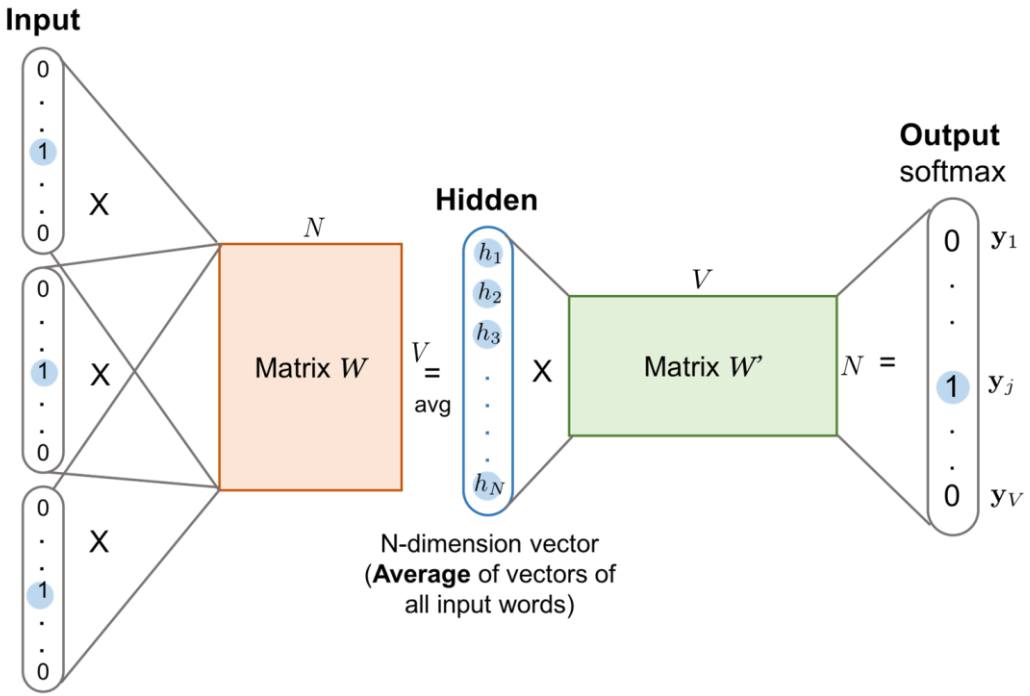
$$will\,(X_2)_{1\times 6} \times W_{6\times 5} = V(X_2)_{1\times 5}$$

will [0,1,0,0,0,0]
eat [0,0,1,0,0,0]
this [0,0,0,0,1,0]
evening [0,0,0,0,0,1]

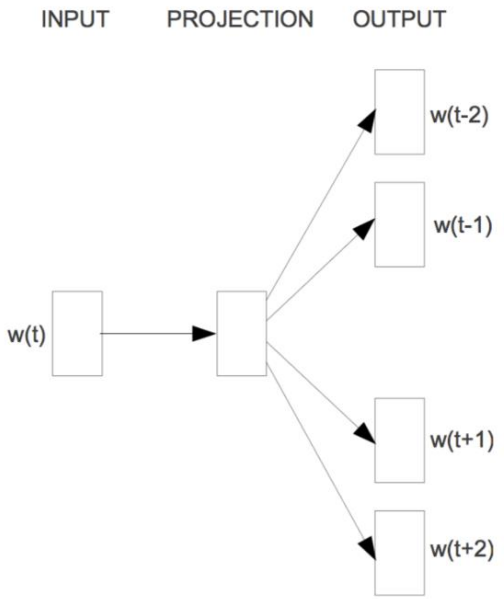
strawberries [0,0,0,1,0,0] X_4

$$W' \times V' = \widehat{Y} \xrightarrow{sof\,tmax} \begin{bmatrix}0.02\\0.07\\0.05\\0.85\\0.02\\0.08\end{bmatrix} \xrightarrow{argmax} \begin{bmatrix}0\\0\\0\\1\\0\\0\end{bmatrix} \xrightarrow{\text{예측결과}} strawberries$$

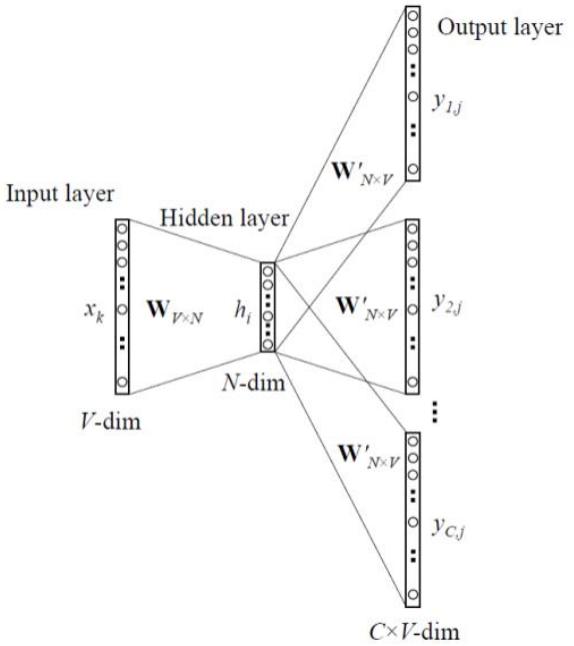
Continuous Bag-of-Words Model



Skip-gram Model

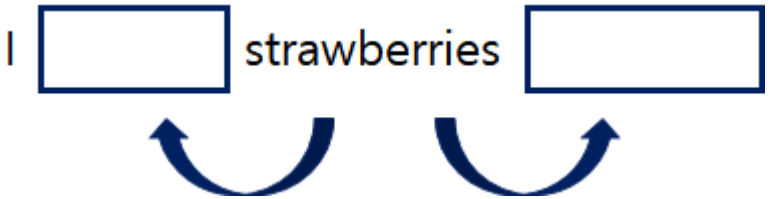


Skip-gram

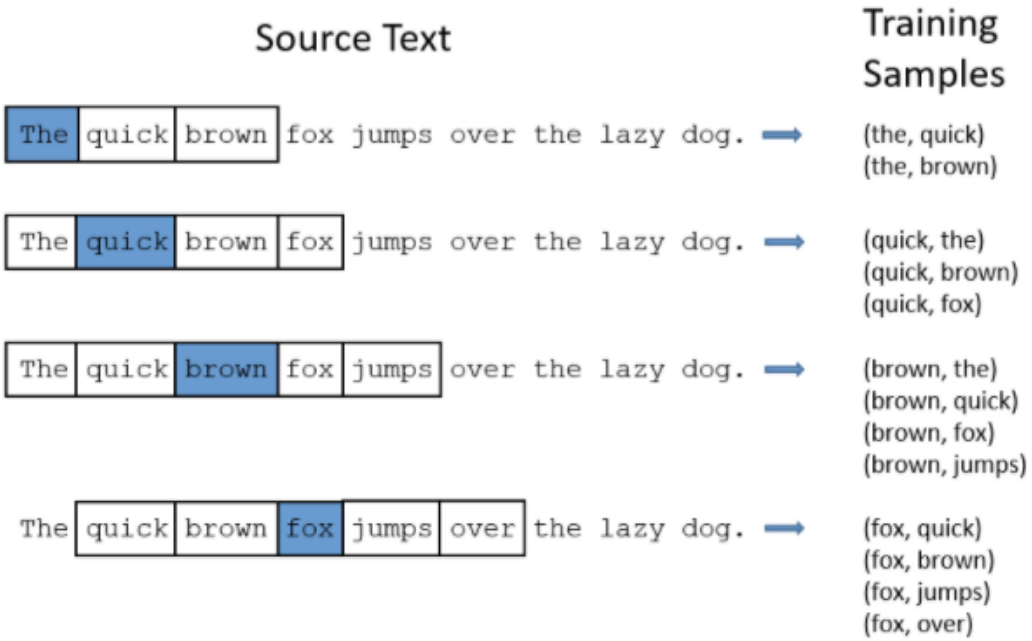


Skip-gram Architecture

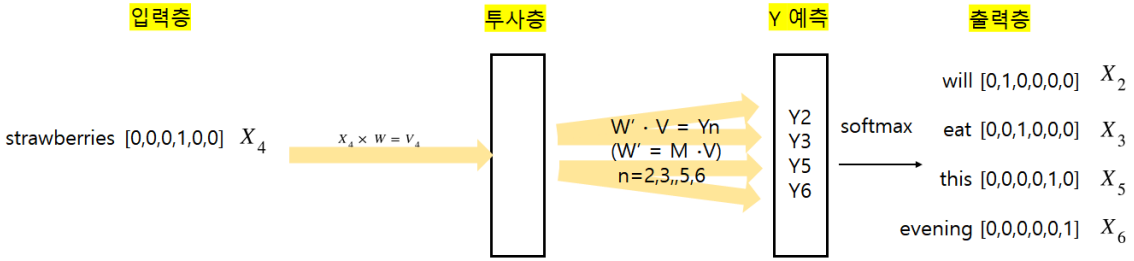
Skip-gram Model



Skip-gram Model



Skip-gram Model



Conclusion

We observed that it is possible to train high quality word vectors using very simple model architectures, compared to the popular neural network models (both feedforward and recurrent). Because of the much lower computational complexity, it is possible to compute very accurate high dimensional word vectors from a much larger data set... To find a word that is similar to small in the same sense as biggest is similar to big, we can simply compute vector X = vector("biggest") −vector("big") + vector("small").... Finally, we found that when we train high dimensional word vectors on a large amount of data, the resulting vectors can be used to answer very subtle semantic relationships between words, such as a city and the country it belongs to, e.g. France is to Paris as Germany is to Berlin. Word vectors with such semantic relationships could be used to improve many existing NLP applications, such as machine translation, information retrieval and question answering systems, and may enable other future applications yet to be invented.

- 낮은 연산 복잡도
- 높은 정확도

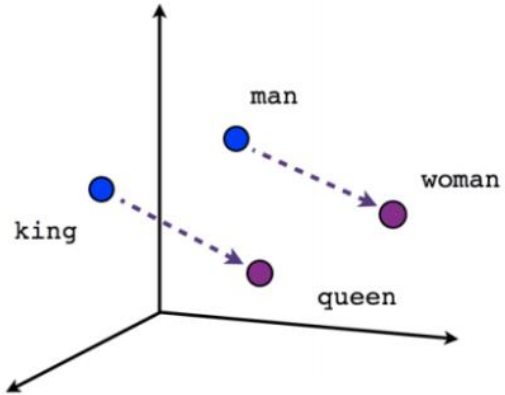
Comparison

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

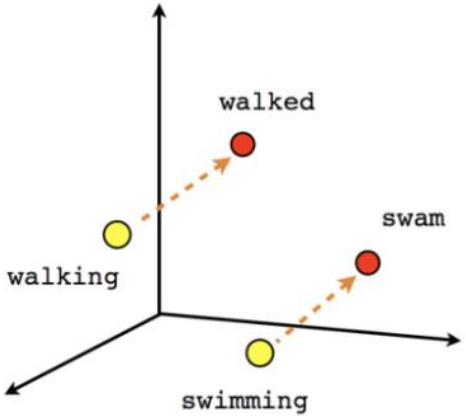
Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

- 8869개 의미적 문항과 10675개의 문법적 문항으로 테스트

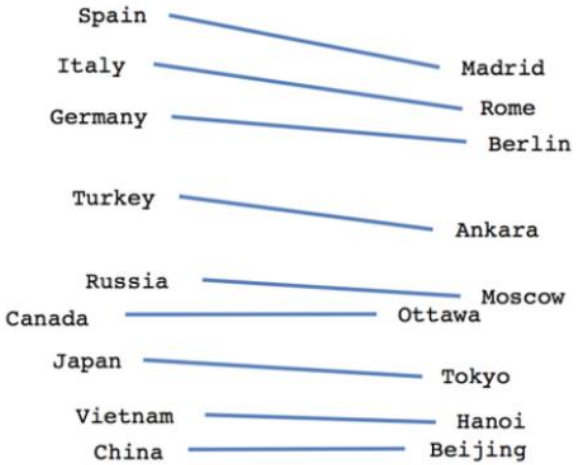
Comparison



Male-Female



Verb tense



Country-Capital

https://tensorflowkorea.gitbooks.io/tensorflow-kr/content/g3doc/tutorials/word2vec/

Comparison

Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Comparison

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Comparison

Table 5: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Comparison

Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Table 7: *Comparison and combination of models on the Microsoft Sentence Completion Challenge.*

Architecture	Accuracy [%]
4-gram [32]	39
Average LSA similarity [32]	49
Log-bilinear model [24]	54.8
RNNLMs [19]	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	**58.9**

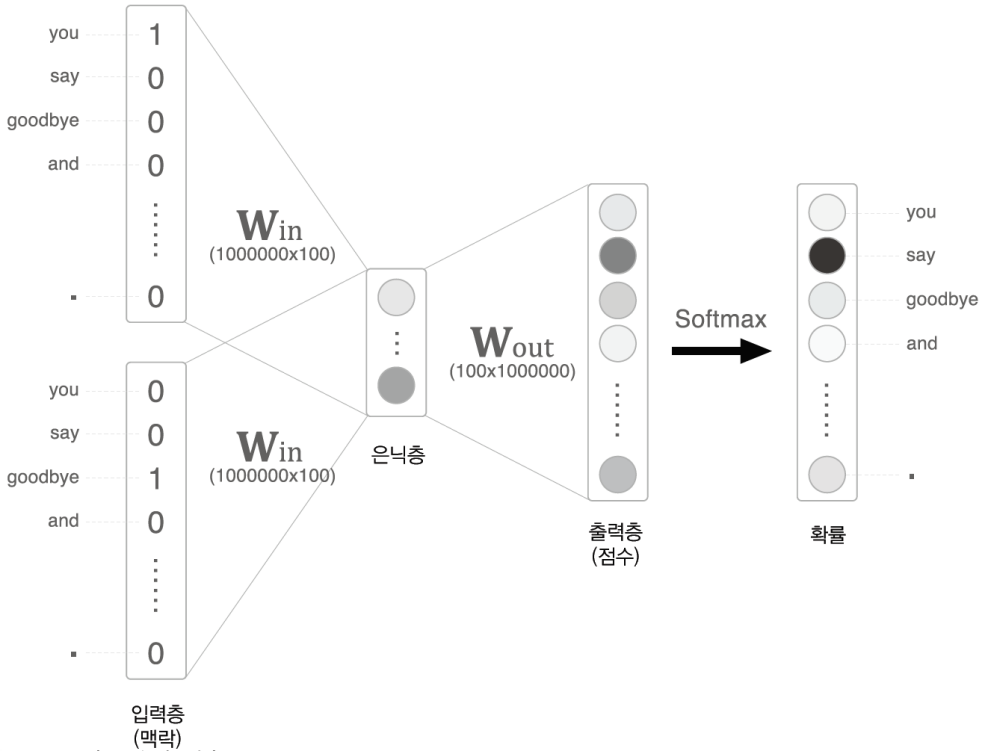
Word2Vec

(Distributed Representations of Words and Phrases and their Compositionality)

https://arxiv.org/abs/1310.4546

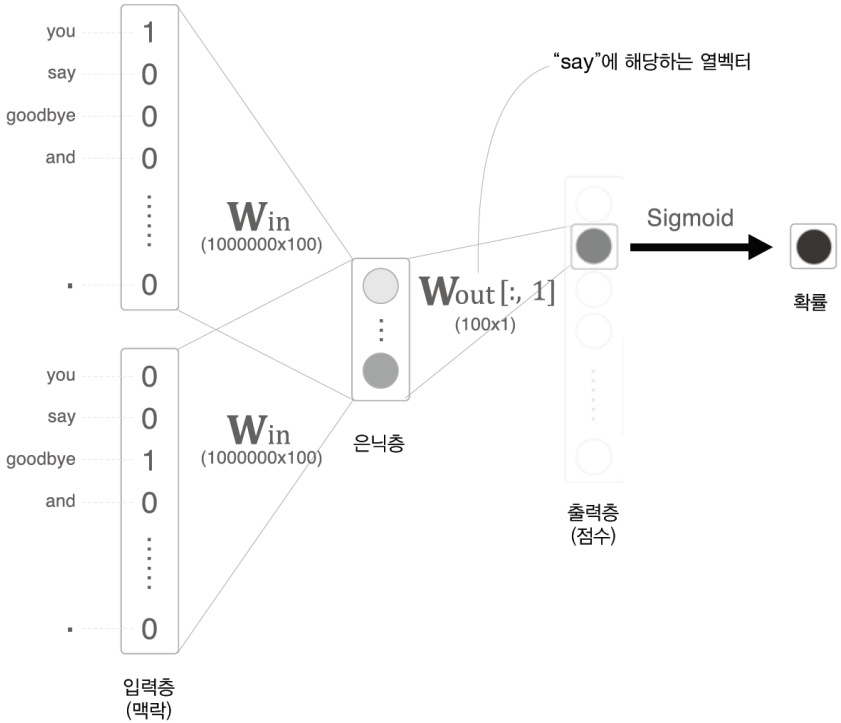
Skip-gram Model with Negative Sampling

- 기존에 skip-gram 이 가졌던 연산량의 문제 극복



Skip-gram Model with Negative Sampling

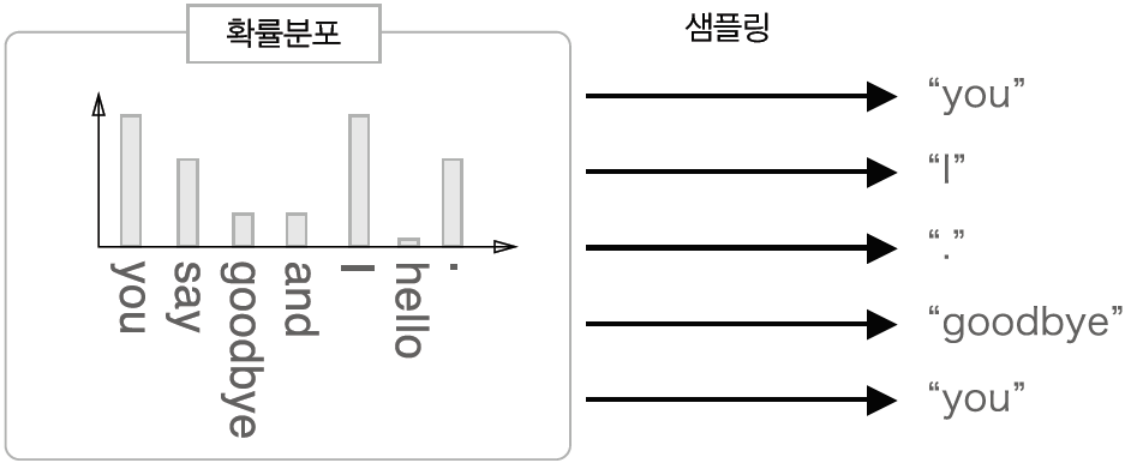
- 전체 단어가 아니라 일부 단어를 뽑아서 계산



Skip-gram Model with Negative Sampling

- 정답지(positive sampling), 오답지(negative sampling) 에서 샘플링
- 무작위 샘플링 보다 말뭉치의 확률 분포를 따라 샘플링 (빈도수가 낮은 단어로 샘플링하면 효과 감소)

  ➡ 말뭉치에서 자주 등장하는 단어가 선택될 확률이 높음



Skip-gram Model with Negative Sampling

- 윈도우 내 등장하지 않은 단어 wi 가 negative sample 로 뽑힐 확률
- 출현 확률이 낮은 단어를 아주 버리지 않기 위해

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(w_j)^{\frac{3}{4}}}$$

Skip-gram Model with Sub Sampling

- 너무 자주 등장하는 단어는 학습에서 제외 하자(e.g. the, 은/는)
- $f(w_i)$ 는 w_i 의 빈도, $t = 10^{-5}$
- if $f(w_i) = 0.01$ => $P(w_i) = 0.9684$ 즉, 100번의 학습 기회 가운데 96번은 제외

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Doc2Vec

https://arxiv.org/abs/1301.3781

Pharagraph2Vec

https://arxiv.org/abs/1301.3781

ngram2Vec

https://arxiv.org/abs/1301.3781

hashtag2Vec

https://arxiv.org/abs/1301.3781

LDA2Vec

https://arxiv.org/abs/1301.3781

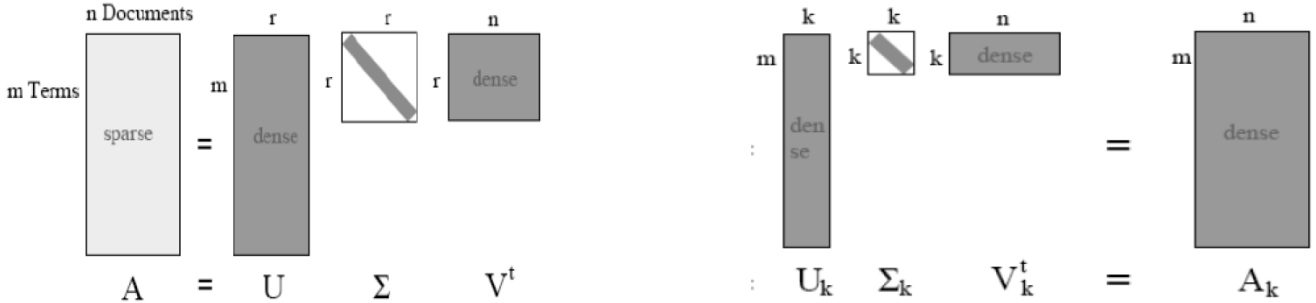
GloVe

단어의 표현(Word Representation)

한국어 임베딩 p145

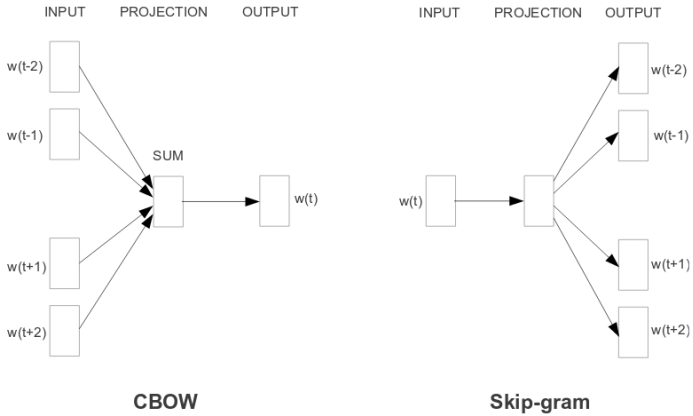
기존 임베딩의 문제점 (1)

- Global Matrix Factorization (예. LSA)
  - 단어-문서 또는 단어-단어 행렬을 분해(예. ED, SVD)하여 저차원 공간에 단어 분산 표현(Distributed Representation)
  - 단어에 대한 **<u>전체적인 통계정보를 활용</u>**한다는 점이 강점
  - **<u>단어 유추 문제에 좋지 않은 성능</u>**을 보임



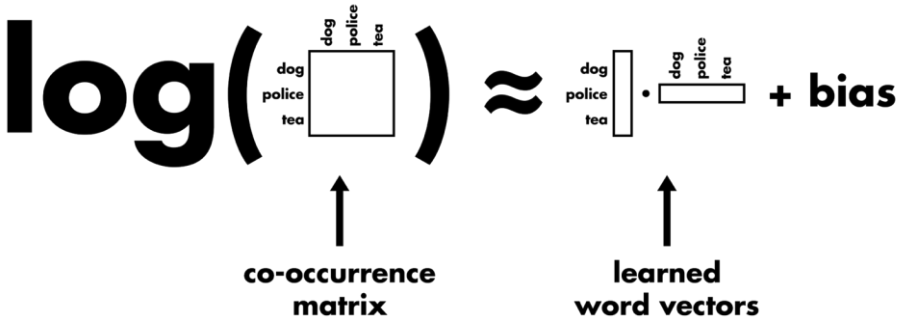
기존 임베딩의 문제점 (2)

- Shallow Window-Based Method (예. Word2Vec)
  - 지역적인 문맥(local context) 정보를 한정적으로 사용하여 단어를 vector로 표현
  - **단어 유추 문제에서는 비교적 좋은 성능**을 보임
  - 학습 데이터(corpus)에서 관찰되는 **단어사용 통계정보를 활용하지 않는 다**는 점에서 한정적
  - 지역적 문맥에 대한 학습은 가능, 학습 데이터(corpus)에서 관찰되는 서로 다른 두 단어의 동시발생 횟수(co-occurrence)에 기반한 학습은 할 수 없음



GloVe (GloVe: Global Vectors for Word Representation)

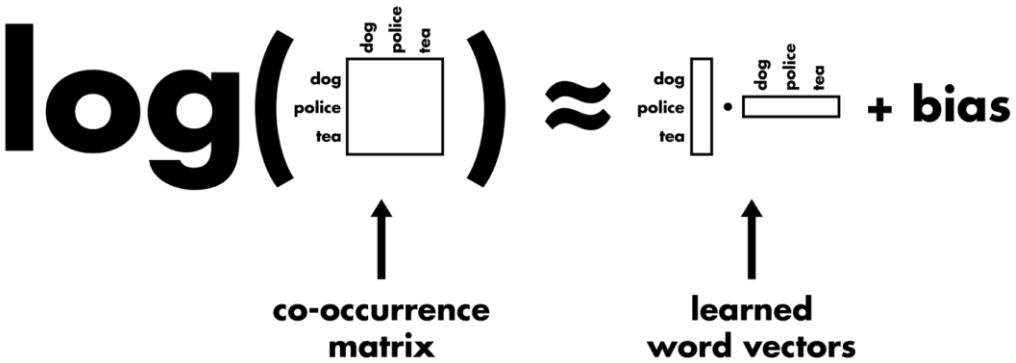
- GloVe는 2013년 구글에서 개발한 Word2Vec의 단점을 보완
- **“임베딩된 단어 벡터간 유추문제에 좋은 성능을 보이면서(word2vec의 장점) 말뭉치 전체의 통계 정보를 반영(LSA의 장점)”**이 GloVe핵심 목표
- 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의
  (their dot product equals the logarithm of the words’ probability of co-occurrence)
- https://nlp.stanford.edu/projects/glove/



[그림]. https://towardsdatascience.com/emnlp-what-is-glove-part-v-fa888272c290

GloVe의 목적함수

- 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의
  (their dot product equals the logarithm of the words' probability of co-occurrence)
- 특정 단어 k가 주어졌을 때 임베딩된 두 단어벡터의 내적이 두 단어의 동시등장확률 간 비율이 되도록 임베딩
  - solid라는 단어가 주어졌을 때 ice와 steam 벡터 사이의 내적값이 8.9가 되도록 (코사인유사도가 커지도록)
  - gas가 주어졌을 때 ice와 steam 벡터 사이의 내적값이 0.0085가 되도록 (코사인유사도가 작아지도록)



단어-문맥 행렬(Term-Context matrix)

- 단어-문맥 간의 동시등장(co-occurrence) 행렬
- 문맥은 사용자가 설정한 window의 크기로 결정
- 문맥 내 등장하는 단어의 빈도를 표기

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Co-occurrence probability

- 단어간 co-occurrence 행렬을 생성
- ice와 steam의 단어가 있음
  - ice가 사용된 문맥에서, 단어 k도 사용되었을 확률
  - steam이 사용된 문맥에서, 단어 k도 사용되었을 확률
  - P(k | ice)/P(k | steam) 상대 비율. 둘 중 상대적으로 더 많이 사용된 곳을 구분
- water와 fashion은 P(k | ice)/P(k | steam) 비율이 1에 가까워 steam과 ice를 구분에 비적합
- solid와 gas의 경우 P(k | ice)/P(k | steam) 값이 1보다 월등히 크거나 작음, 따라서 구분에 적합

Probability and Ratio	*k = solid*	*k = gas*	*k = water*	*k = fashion*
P(k | ice)	1.9 x 10^-4	6.6 x 10^-5	3.0 x 10^-3	1.7 x 10^-4
P(k | steam)	2.2 x 10^-5	7.8 x 10^-4	2.2 x 10^-3	1.8 x 10 ^-5
P(k | ice)/P(k | steam)	8.9	8.5 x 10 ^-2	1.36	0.96

GloVe의 목적함수

- P_{ik}를 P(k | i)로 정의
  - i 번째 단어 주변(윈도우 크기는 사용자 지정)에 k번째 단어가 등장할 조건부확률
  - 빈도수(X_{ik})'를 '$X_i=\Sigma_k X_{ik}$'로 나눠준 값( )입니다. 위 표 기준으로 예를 들면 P(solid | ice)
  - P_{ik}/P_{jk}의 의미 : P(solid | ice)/P(solid $\frac{X_{ik}}{X_i}$ team) = 8.9
  - X : 동시 등장 행렬
  - X_{ij} : 중심 단어 i가 등장 했을때 윈도우 내 주변 단어 j가 등장하는 횟수
  - X_i : $\Sigma_j X_{ij}$: 동시 등장 행렬에서 i행의 값을 모두 더한 값
- d차원 벡터공간에 임베딩된 ice, steam, solid 벡터를 넣으면 8.9를 반환하는 F(목적함수)를 찾는 최적화 문제

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F(w_{ice}, w_{steam}, w_{solid}) = \frac{P_{ice,solid}}{P_{steam,solid}} = \frac{P(solid|ice)}{P(solid|steam)} = \frac{1.9 \times 10^{-4}}{2.2 \times 10^{-5}} = 8.9$$

GloVe의 목적함수

- F 안에 집어넣을 w_i, w_j, w_k 간에 관계를 따져보기 위해 w_i와 w_j를 뺀 벡터에 w_k를 내적후
- 임베딩된 두 단어벡터의 내적은 전체 말뭉치의 동시등장확률이 되도록 하려는 목적이 있었으므로 P_{ik}를 $F(W^T_i W_k)$로 정의.

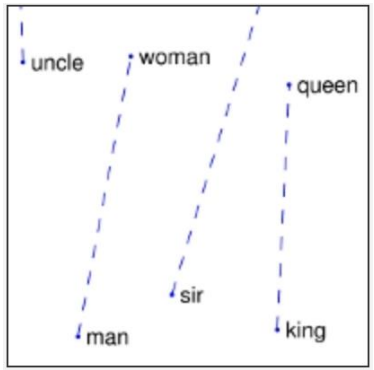
$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Homomorphism

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$F(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$



https://nlp.stanford.edu/projects/glove/

GloVe의 목적함수

- 함수 F의 전제 조건 (Homomorphism 를 만족해야한다)
  - w_i와 w_k를 서로 바꾸어도 같은 값을 반환. w_k는 w_i나 w_j가 될 수 있음
  - 말뭉치 co-occurrence 행렬 X는 대칭행렬(symmetric matrix)이므로 Transpose해도 같은 값을 반환
  - 세번째 조건에 만족하는 함수는 지수함수 => F를 exponential로 치환

$$w_i \longleftrightarrow \tilde{w}_k$$

$$X \longleftrightarrow X^T$$

$$F(X - Y) = \frac{F(X)}{F(Y)}$$

$$e^{x+y} = e^x e^y$$

GloVe의 목적함수

- Homomorphism 만족하는 함수는 지수함수 => F를 exponential로 치환
- w_i와 w_k를 서로 바꾸어도 같은 값을 반환
  - $\log(P_{ik}) = \log(P_{ki})$ 이 성립 해야함
  - $\log(X_{ik}) - \log(X_i) = \log(X_{ki}) - \log(X_k)$ 이 성립 해야함
  - logXi 가 걸림 -> 이를 만족하기 위해 상수항(b_i, b_k)으로 조건을 만족하도록 함 (i행의 값을 모두 더한 값: 스칼라)

$$exp(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{exp(w_i^T \tilde{w}_k)}{exp(w_j^T \tilde{w}_k)}$$

$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$

$$w_i^T \tilde{w}_k = \log X_{ik} - b_i - \tilde{b}_k$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$$

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

log

$$\frac{P_{ik}}{P_{jk}} = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

$$exp(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

GloVe의 목적함수

- $\log(X_{ik})$는 윈도우 내 말뭉치가 전체에서 단어별 등장 빈도를 구한 co-occurrence matrix에 로그를 취한 행렬
- 좌변과 우변의 차이를 최소로 하는 값이 d차원 벡터공간에 임베딩된 단어벡터
- $\log(X_{ik})$를 근사할 수 있는 w_i, w_j, b_i, b_j 를 학습 = 목적함수 J를 최소화하는 w_i, w_j, b_i, b_j 를 학습
- 학습한 임베딩된 단어벡터의 내적이 $\log(X_{ik})$ 근사한다는 의미. 이는 코사인유사도와 비례 관계임

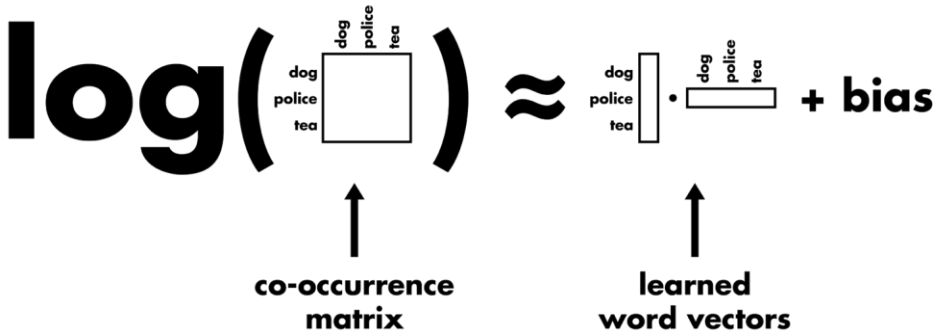
$$exp(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{exp(w_i^T \tilde{w}_k)}{exp(w_j^T \tilde{w}_k)}$$

$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$

$$w_i^T \tilde{w}_k = \log X_{ik} - b_i - \tilde{b}_k$$

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$$

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



Weighted error

- 목적함수에 아래와 같은 모양의 f(X)를 추가했습니다. x_{ij}가 특정 값 이상 빈도가 큰 경우 가중치를 조정.
- 범위 초과($x_{ij} > x_{max}$) 빈도를 가지는 단어들은 error를 그대로 학습에 사용
- 범위 이내($x_{ij} < x_{max}$) 빈도를 가지는 단어들(=infrequent word)은 error의 중요도를 낮춰 학습에 사용

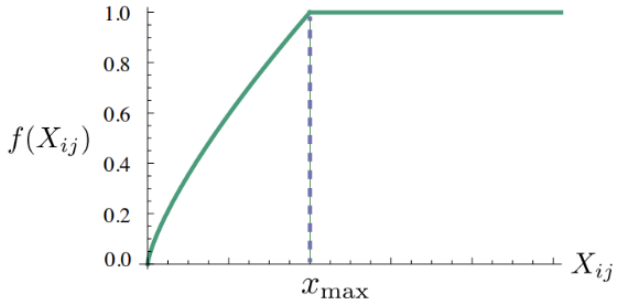


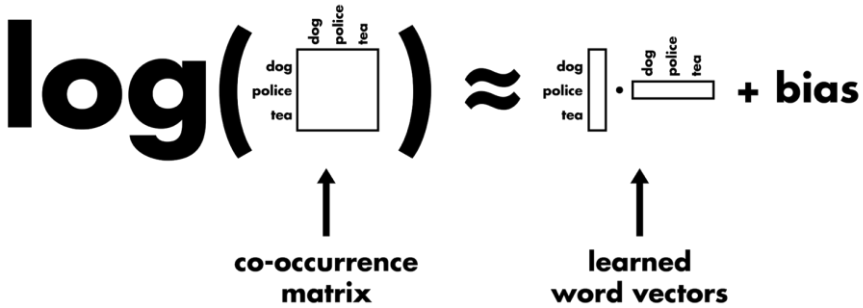
Figure 1: Weighting function f with $\alpha = 3/4$.

Error

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

$$where \quad f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & if \quad x < x_{max} \\ 1 & otherwise \end{cases}$$

GloVe의 결과



$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log X_{ik})^2, \qquad f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}.$$

FastText

단어의 표현(Word Representation)

Before FastText

- BPE (Byte Pair Encoding) - (https://github.com/rsennrich/subword-nmt)
- 서브워드(sub-word) 단위 분절 방법
- 예) "conference" -> 'con', 'f', 'er', 'ence'
- 왜? => OOV 때문에
- 한국어 임베딩 p130

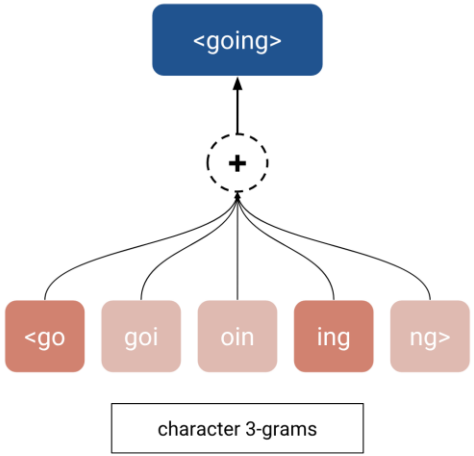
FastText

- Facebook에서 발표한 Word Embedding 기법
- Word2vec나 GloVe의 경우 언어의 형태학적(Morphological)인 특성을 반영하지 못하고, 또 희소한 단어에 대해서는 Embedding이 되지 않음
- FastText에서는 단어를 **Bag-of-Characters**로 보고, 개별 단어가 아닌 n-gram의 characters를 Embedding함 (Skip-gram model 사용)
  - 각 단어는 Embedding된 n-gram의 합으로 표현됨, 그 결과 빠르고 좋은 성능을 보임
- Word2Vec와 FastText의 가장 큰 차이점은 Word2Vec은 단어를 쪼개질 수 없는 단위로 생각한다면, FastText는 하나의 단어 안에도 여러 단어들이 존재하는 것으로 간주(= **Subword**)
- 내부 단어(subword)를 고려하여 학습
  - 내부 단어(subword)를 통해 모르는 단어(OOV)에 대해서도 다른 단어와의 유사도를 계산할 수 있음

OOV (Out Of Vocabulary)

- FastText에서 각 단어를 글자의 n-gram으로 나타냄
- 예를 들어, tri-gram의 경우, apple은 app, ppl, ple로 분리하고 임베딩
- FastText에서 birthplace(출생지)란 단어를 학습하지 않은 상태라고 해보자.
  - 다른 단어 n-gram으로서 birth와 place를 학습한 적이 있다면 birthplace의 임베딩 벡터(Embedding Vector)를 만들어낼 수 있음

〈ap, app, ppl, ple, le〉 # n = 3 이므로 길이가 3
〈apple〉 # 특별 토큰



Rare Word

- Word2Vec 경우 단어 등장 빈도가 높을 수록 정확하게 임베딩 되지만, 희소 단어(rare word) 경우 임베딩 정확도가 높지 않음

- FastText는 희소 단어(rare word)의 경우 문자로 n-gram을 하는 특성학 학습 경우의 수가 많아지므로 Word2Vec와 비교하여 정확도가 높은 경향 (=FastText가 노이즈가 많은 코퍼스에 강점)

- Word2Vec에서는 오타가 섞인 단어는 임베딩이 되지 않음(OOV). FastText는 이 경우도 일정 수준 성능을 보임

한국어 FastText

- 글자 단위 – 글자 단위로 임베딩하는 경우

  예를 들어서 글자(Character) 단위의 임베딩의 경우에 n=3일때 '자연어처리'라는 단어에 대해 n-gram을 만들어보면 다음과 같다.

  <자연, 자연어, 연어처, 어처리, 처리>

- 자모 단위 – 자모 단위(초성, 중성, 종성 단위)로 임베딩하는 경우.

  예를 들어 '자연어처리'라는 단어에 대해서 초성, 중성, 종성을 분리하고, 만약, 종성이 존재하지 않는다면 '_'라는 토큰을 사용한다고 가정한다면 '자연어처리'라는 단어는 아래와 같이 분리가 가능.

  <ㅈ ㅏ, ㅈ ㅏ _, ㅏ _ ㅇ, ... 중략>

- https://fasttext.cc/docs/en/pretrained-vectors.html

임베딩 비교

단어의 표현(Word Representation)

Word Embedding 방법론

임베딩 비교

- Word2Vec
  - Word2Vec는 사용자가 설정한 window내에 등장한 단어의 벡터 요소를 업데이트 하며 단어를 임베딩
  - window 내 등장하지 않는 단어는 멀어지도록 (내적값이 줄어들도록 = 유사도가 작아지도록),
    동시 등장하는 단어는 가까워지도록(내적값이 커지도록 = 유사도가 커지도록) 학습
- Glove
  - GloVe는 2014년 미국 스탠포드대학 연구팀에서 개발한 단어 임베딩 방법론
  - 단어 동시 등장 여부를 사용 (Term-Context matrix, Co-occurrence matrix)
  - GloVe로 임베딩된 단어 벡터끼리의 내적은 동시 등장확률의 로그값과 같습니다.
    (their dot product equals the logarithm of the words' probability of co-occurrence)
- FastText
  - 페이스북이 2016년 발표
  - 단어를 부분단어(subword)의 벡터들로 표현한다는 점을 제외하고는 Word2Vec와 유사
  - 노이즈가 많은 말뭉치에 강함.

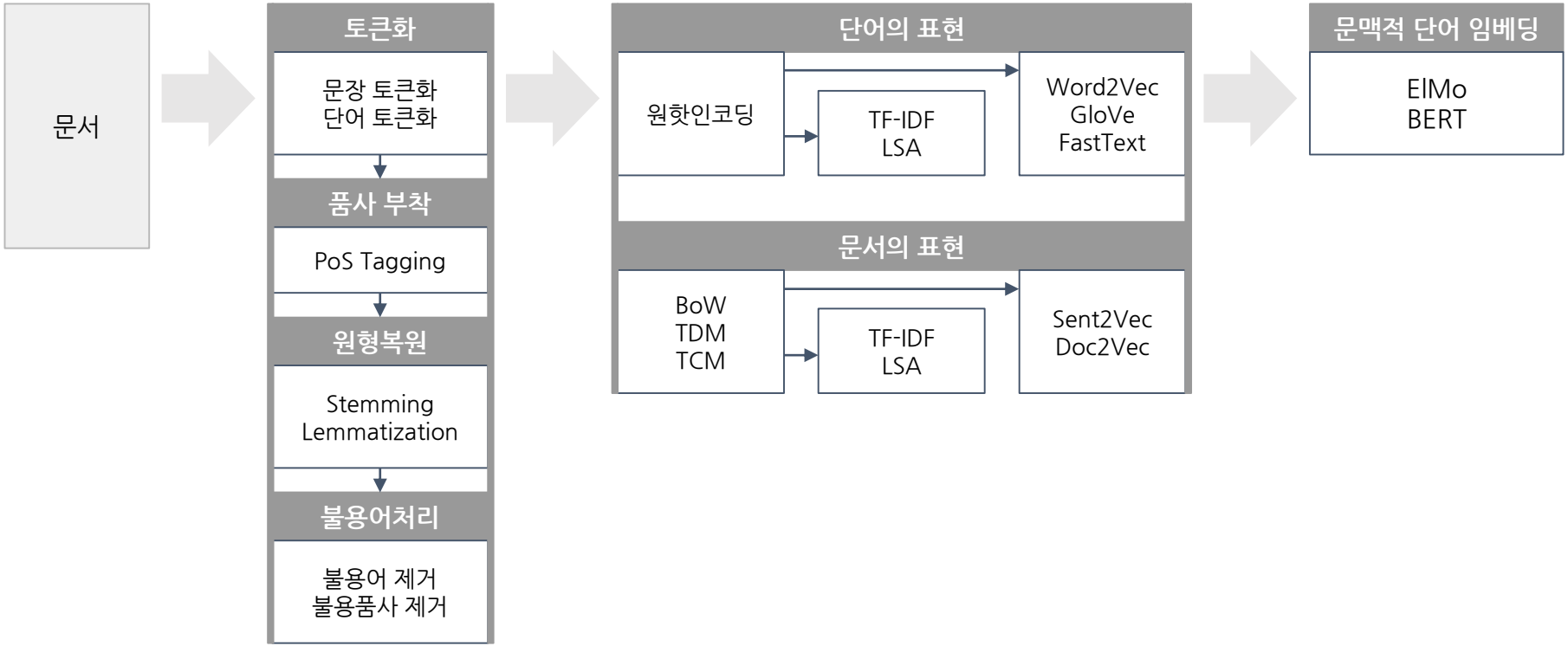
세 방법론의 한계

- 의미상 아무런 관련이 없어 보이는 단어간에도 벡터공간에 가깝게 임베딩(=코사인유사도가 큰) 결과를 보일 수 있음 (예. 소프트웨어, 하드웨어) – **유사 단어라기 보다는 관련성이 있는 단어**

- 임베딩시 단어간 동시 등장 정보를 사용한다는 면에서는 Word2Vec, GloVe, Fast-text 모두 count based 방법과 본질적으로 유사

- 하지만 동시 등장 정보를 벡터로 표현했을때 의미를 보존하고 있다는 면에 있어서, 기존 count based 방법인 TF-IDF, LSA 보다 월등히 개선되었기 때문에 각광을 받음

임베딩 절차



처리 의존도를 고려하여 성능을 높이는 작업을 진행