Sentence Embeddings Using Korean Corpora

NPLM, Word2Vec, FastText, LSA, GloVe

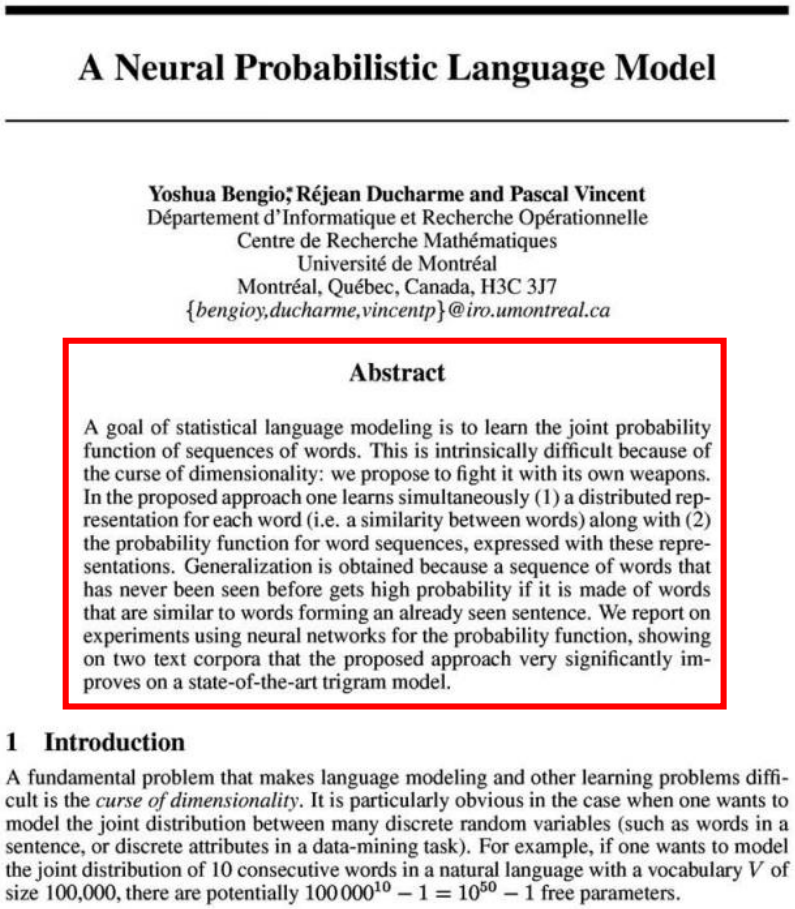
Presented by | MyungHoon-Jin

github.com/jinmang2

2020.02.25

4.1 NPLM: Neural Probabilistic Language Model

A Neural Probabilistic Language Model – Bengio et al., 2003



A Neural Probabilistic Language Model

Yoshua Bengio; Réjean Ducharme and Pascal Vincent
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal
Montréal, Québec, Canada, H3C 3J7
{*bengioy,ducharme,vincentp*}*@iro.umontreal.ca*

Abstract

A goal of statistical language modeling is to learn the joint probability function of sequences of words. This is intrinsically difficult because of the curse of dimensionality: we propose to fight it with its own weapons. In the proposed approach one learns simultaneously (1) a distributed representation for each word (i.e. a similarity between words) along with (2) the probability function for word sequences, expressed with these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar to words forming an already seen sentence. We report on experiments using neural networks for the probability function, showing on two text corpora that the proposed approach very significantly improves on a state-of-the-art trigram model.

1 Introduction

A fundamental problem that makes language modeling and other learning problems difficult is the *curse of dimensionality*. It is particularly obvious in the case when one wants to model the joint distribution between many discrete random variables (such as words in a sentence, or discrete attributes in a data-mining task). For example, if one wants to model the joint distribution of 10 consecutive words in a natural language with a vocabulary V of size 100,000, there are potentially $100\,000^{10} - 1 = 10^{50} - 1$ free parameters.

- 통계기반 전통적인 언어 모델의 한계를 극복하려 시도
- Objective: Sequence of words 결합 확률 함수 학습
- Curse of Dimensionality, 기존 언어 모델의 문제점?
  1) **학습 데이터에 존재하지 않는 n-gram이 포함된 문장이 나타날 확률을 0으로 부여**
  2) **문장의 장기 의존성을 포착하기 어려움**
  3) **단어/문장 간 유사도를 계산할 수 없다**
- 각 단어에 대한 분산표현(의미 유사도)와 문장의 결합 확률 함수를 동시에(Simultaneously) 학습
- SOTA tri-gram model을 업그레이드시킴. 우린최고

4.1 NPLM: Neural Probabilistic Language Model

1) 학습 데이터에 존재하지 않는 n-gram이 포함된 문장이 나타날 확률을 0으로 부여

$$P(w_1^T) = \prod_{t-1}^{T} P(w_t|w_1^{t-1}) \quad w_i^j = (w_i,\ w_{i+1}, \cdots, w_{j-1},\ w_j)$$

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-n+1}^{t-1})$$

2) 문장의 장기 의존성(long-term dependency)을 포착하기 어려움

As ***n*** *gets bigger,*

word sequence *increases exponentially*

$s.t\ occurence\ probability = 0$

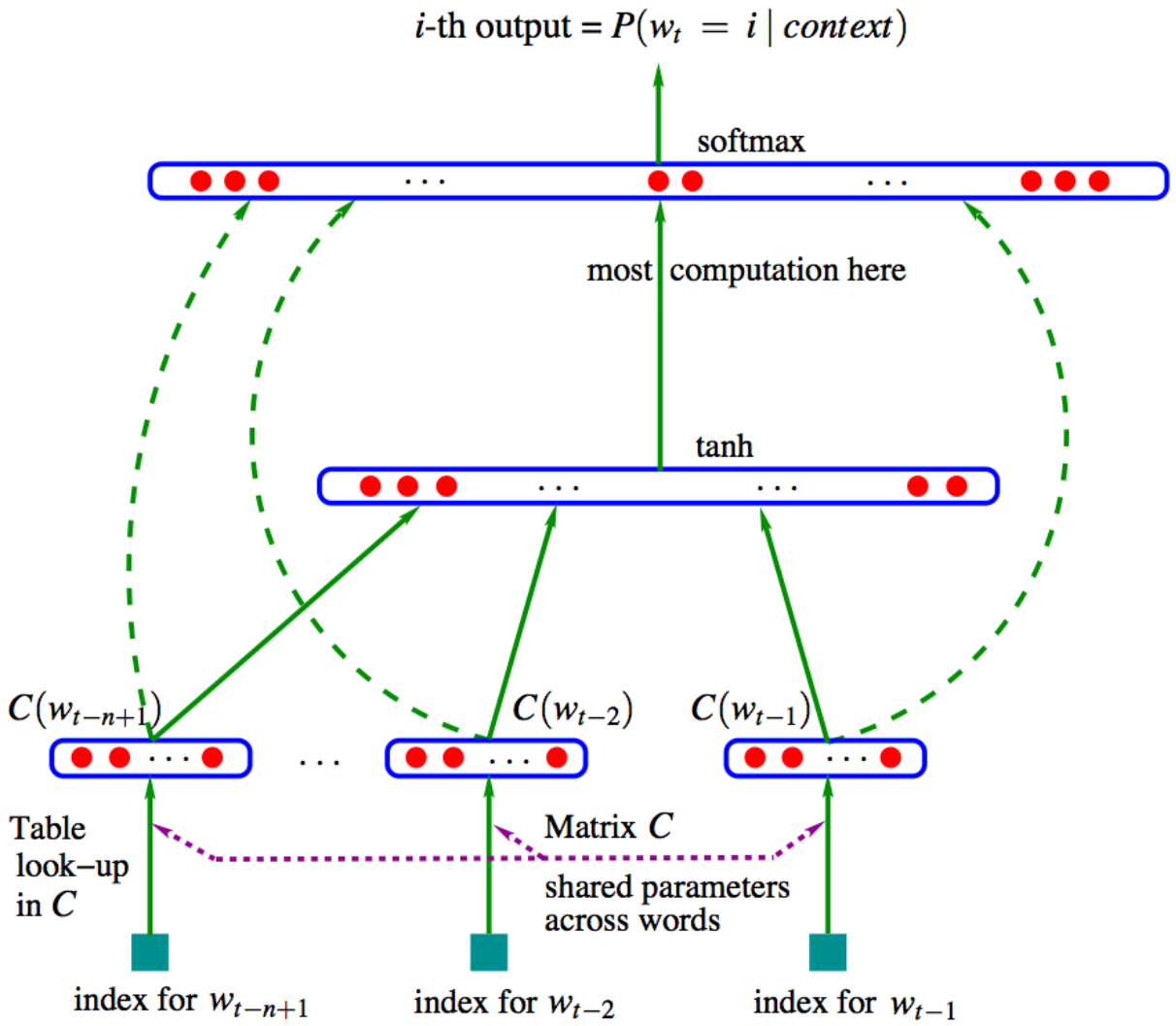
3) 단어/문장 간의 유사도(Similarity)를 계산할 수 없다.

The cat is walking in the bedroom

A dog was running in a room

- 통계적 언어 모델은 이전 문장이 주어질 경우 다음 단어가 나올 조건부 확률로 쓸 수 있음
- 자연어의 통계 모델을 구축할 때, 단어 순서의 이점 및 단어 순서에서 시간적으로 더 가까운 단어가 종속적임
- N-Gram 모델로 다음 단어의 조건부 확률 테이블 구축
- 문제는, 학습 코퍼스에 존재하지 않는 n-gram 문장이 나타나면 확률 값을 0으로 부여
- 때문에 n을 키우면 키울수록 등장 확률이 0인 word sequence가 많아져 장기의존성을 포착하기 어려움
- 추가적으로 one-hot vector의 단점으로 유사도 계산이 불가능 (Distance가 전부 똑같음)

NPLM (Direct) Architecture



Proposed Model's Idea

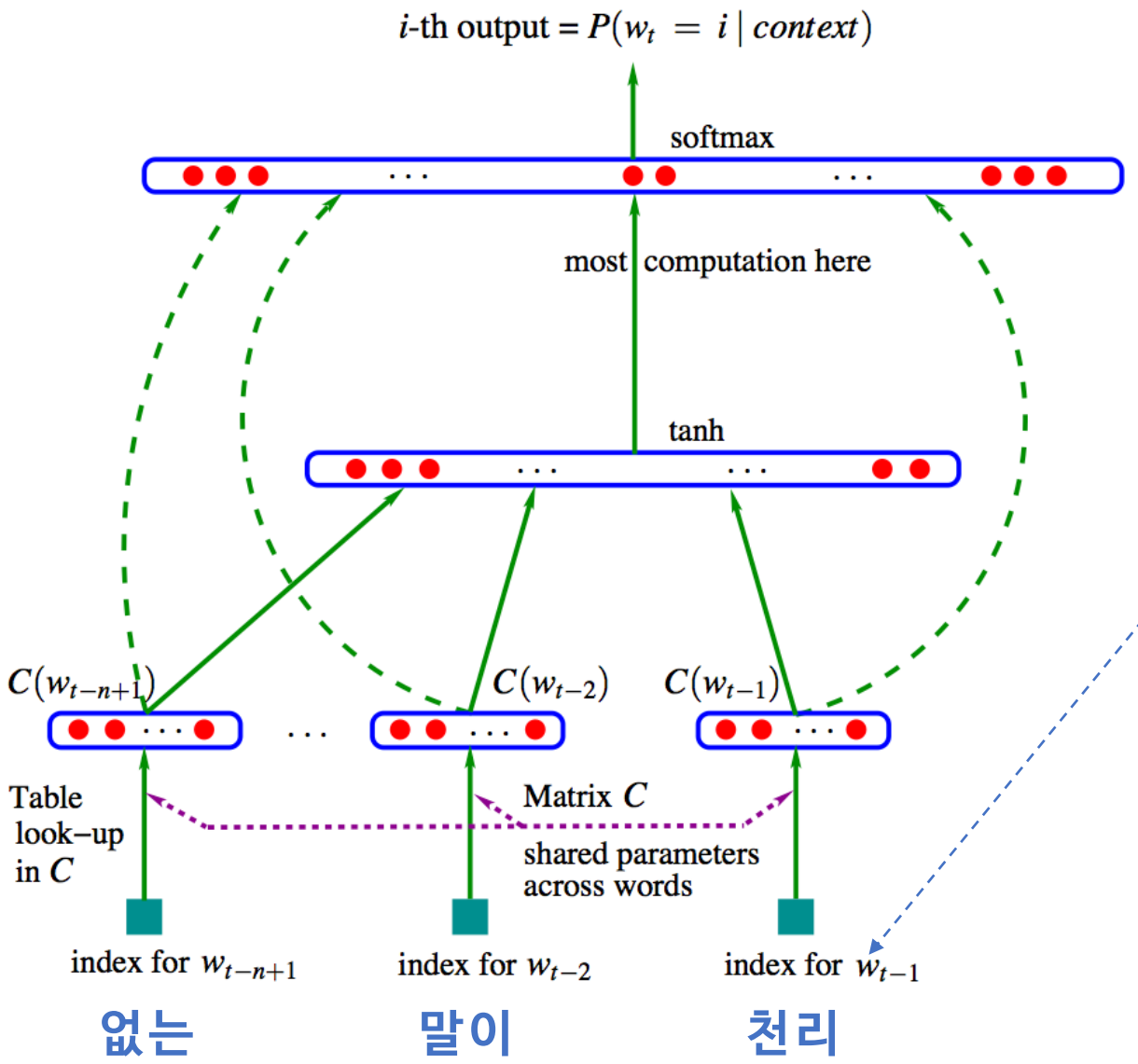
- 단어 사전의 단어들을 feature vector (a real-valued vector in $\mathbb{R}^m$)로 분산 표현하여 단어들 사이의 유사성을 만든다.
- 이러한 word sequenc의 feature vector를 결합 확률 함수(joint probability function)으로 표현
- 위 함수의 parameter와 word feature vector를 동시에 (simultaneously) 학습
- Number of feature vector(m)은 단어 사전의 size보다 작게 설정 (실험적으로 m=30, 60, 90으로 설정)
- 확률 함수는 이전 단어가 주어졌을 때 다음 단어가 나올 조건부 확률의 곱으로 표현 (NN을 적용)
- 이 함수에 가중치 감소 패널티를 추가하여 학습 데이터 또는 정규화된 criterion의 log-likelihood를 최대화하기 위해 반복적으로 조정할 수 있는 매개변수가 존재
- 각 feature vector는 사전 지식으로 초기화 가능

Relation to Previous Work (~2003)

- NN으로 고차원 이진 분산 표현을 실시하는 아이디어는 (1)에서 소개
- (2) Hinton 교수의 연구가 성공적으로 적용된 사례도 존재
- (3) Neural Network를 Language Model에 적용시키려 한 것이 완전 새로운 것은 아님
- 위의 논문과 대조적으로 본 연구에서는 (4) 문장에서 단어의 역할을 배우기보다는 단어 시퀀스 분포의 통계 모델을 배우는데 집중
- 제안한 모델에서는 discrete random or deterministic variable의 유사성을 특징화시키지 않고 distributed feature vector, continuous real-vector for each word를 사용
- 본 논문의 중요한 부분은, 자연어 텍스트에서 word sequence의 확률 분포를 간단히 표현하는데 도움이 되는 단어의 표현을 찾는 것!

Reference.

- (1) Modeling high-dimensional discrete data with multi-layer neural networks – Yoshua Bengio and Samy Bengio
- (2) Extracting distributed representations of concepts and relations from positive and negative propositions – A.Paccanaro and G.E Hinton.
- (3) Natural language processing with modular neural networks and distributed lexicon – R.Miikkulainen and M.G.Dyer
- (4) Sequential neural text compression – Jurgen Schmidhuber



NPLM의 학습 (1)

- $n = 4$
- Text = [‘발’, ‘없는’, ‘말이’, ‘천리’, ‘간다’]
- [발, 없는, 말이] → [천리]
- [없는, 말이, 천리] → [간다]
- 학습시킬 단어의 수가 1,200개라 가정 ($V = 1200$)
- 해당 단어들을 one-hot vector화. 예를 들어,
- 없는 = $[0 \quad 0 \quad \cdots \quad 0 \quad 1 \quad 0] \in \mathbb{R}^{1200}$
- 말이 = $[0 \quad 1 \quad \cdots \quad 0 \quad 0 \quad 0] \in \mathbb{R}^{1200}$
- 천리 = $[0 \quad 0 \quad \cdots \quad 0 \quad 0 \quad 1] \in \mathbb{R}^{1200}$라고 가정,
- C 행렬로 V의 각 원소를 $C(i) \in \mathbb{R}^m$로 매핑

$$C \in \mathbb{R}^{|V| \times m}$$

없는 $[0 \quad 0 \quad \cdots \quad 0 \quad 1 \quad 0]$

말이 $[0 \quad 1 \quad \cdots \quad 0 \quad 0 \quad 0]$

천리 $[0 \quad 0 \quad \cdots \quad 0 \quad 0 \quad 1]$

$1 \times V$

$\times$

$\begin{bmatrix} 11 & 18 & 25 \\ 10 & 12 & 19 \\ \vdots & \vdots & \vdots \\ 23 & 5 & 7 \\ 17 & 24 & 1 \end{bmatrix}$

$V \times 3$

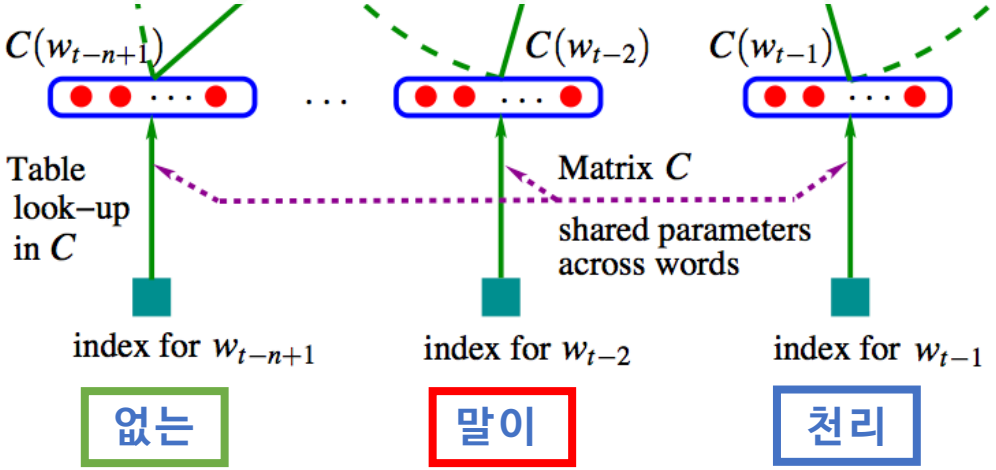
$=$

$[23 \quad 5 \quad 7]$ 없는

$[10 \quad 12 \quad 19]$ 말이

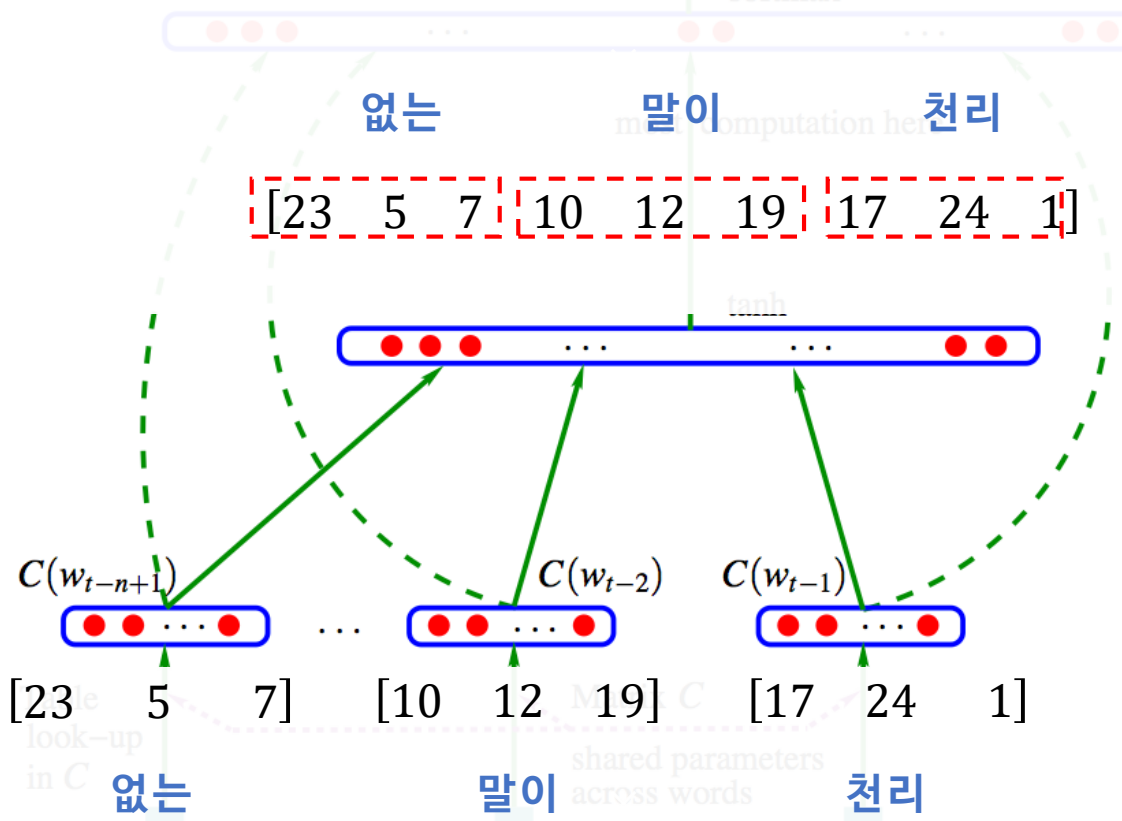
$[17 \quad 24 \quad 1]$ 천리

1×3



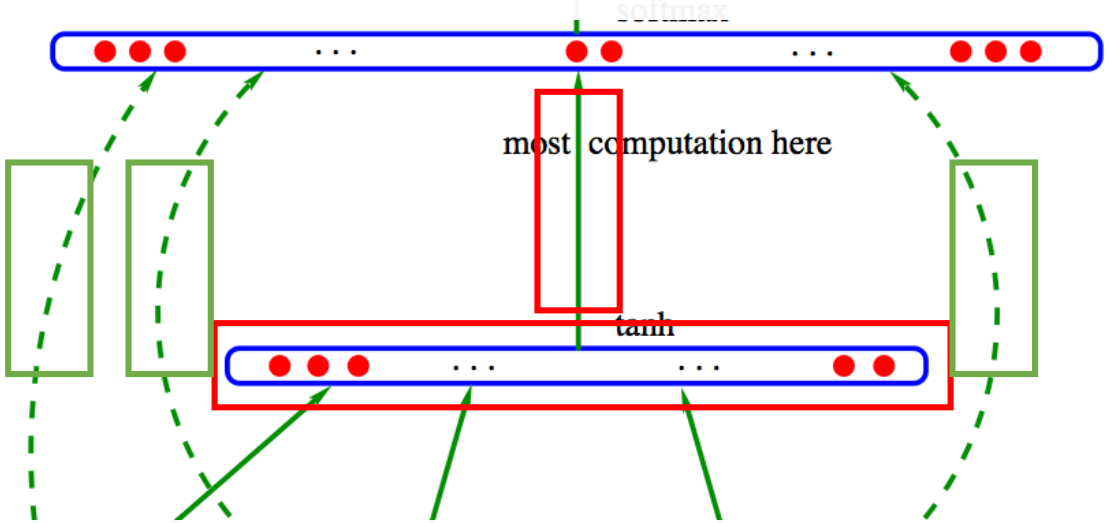
NPLM의 학습 (2)

- C 행렬은 최초에 랜덤으로 초기화 ($m = 3$)
- (prior knowledge로 초기화 가능)
- 없는, 말이, 천리 등의 단어는 one-hot vector
- 즉, 해당 인덱스의 row vector를 C 행렬에서 찾는 것 (look-up)과 동일
- 이를 projection layer라고 표현하기도 하고
- Word compression이라고 말하기도 함
- 각 단어를 C 행렬을 통해 distributed feature vector로 매핑
- → vector 간의 거리를 정의할 수 있음
- → 유사성을 체크할 수 있다!!



NPLM의 학습 (3)

- 굵은 녹색선에 집중!
- NPLM은 N-Gram 기반의 모델이기 때문에 n-1개의 단어로 그 다음 단어를 예측해야 함
- 때문에 feature vector x를 아래와 같이 다시 쓸 수 있음
- $\boldsymbol{x} = [\boldsymbol{x_{t-1}}, \boldsymbol{x_{t-2}}, \ldots, \boldsymbol{x_{t-n+1}}]$ (concatenate)
- 각 feature vector(x_i)는 $\mathbb{R}^3$의 vector
- 즉, x는 $\mathbb{R}^3$ feature vector 를 n-1개 만큼 flatten했기 때문에 x는 $\mathbb{R}^{(n-1)m=(4-1)*3=9}$ 차원의 vector
- 이후 학습할 W, H 행렬의 차원이 $(n-1)m$ 인 이유



$$y_{w_t} = b + Wx + U\tanh(d + Hx)$$

NPLM의 학습 (4)

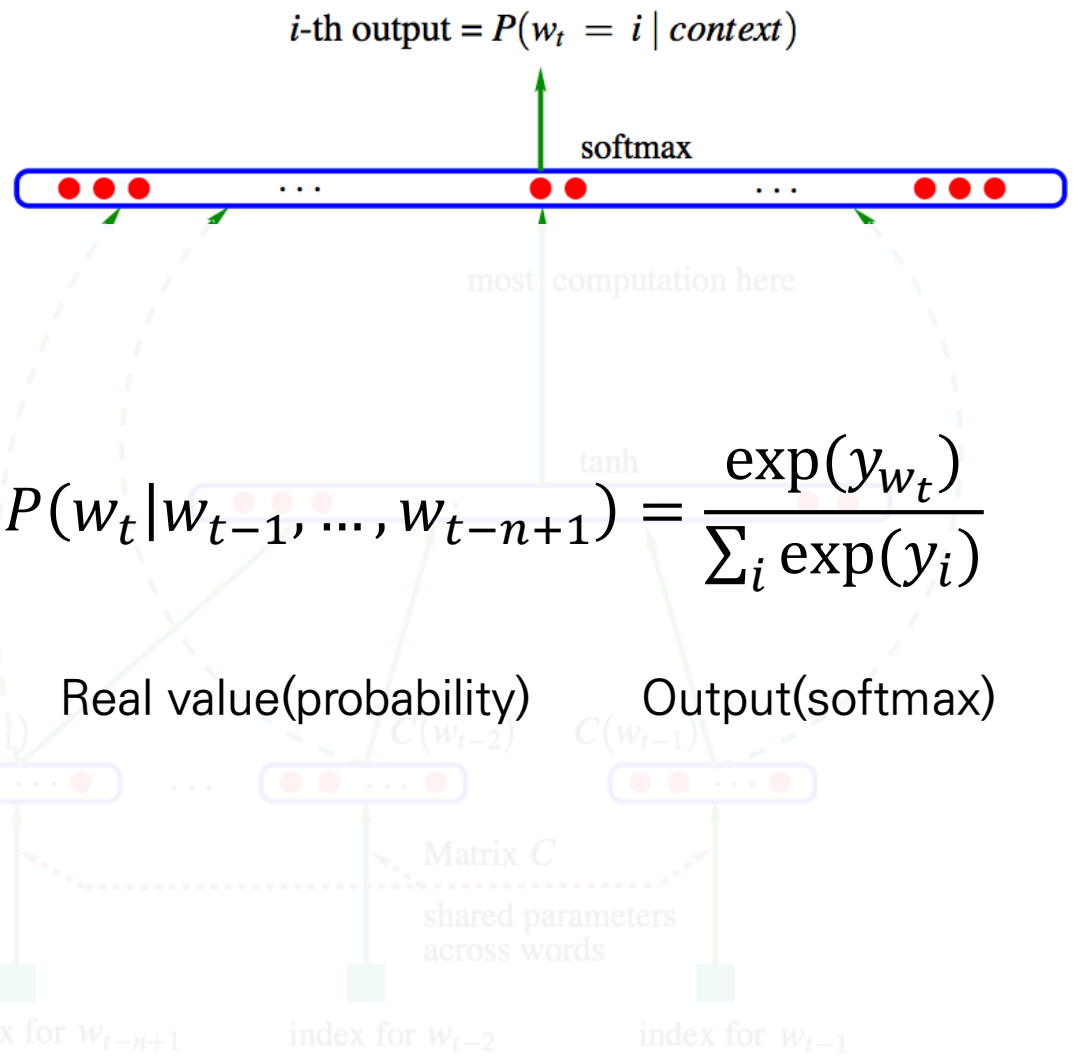
- x 를 h 차원으로 매핑하는 Affine 계산을 실시
- 이를 $tanh$함수로 non-linearity 부여
- $\tanh(d + Hx)$와 x를 bi-linear Affine 계산을 실시
- (다른 torch 구현체에서는 아래처럼 계산하는 경우도 있음!)
- $[b + Wx]$ + $[U\tanh(d + Hx)]$

$$x = [x_{t-1}, x_{t-2}, \ldots, x_{t-n+1}] \in \mathbb{R}^{(n-1)m}$$

$$W \in \mathbb{R}^{|V|\times(n-1)m} \qquad b \in \mathbb{R}^{|V|}$$

$$H \in \mathbb{R}^{h\times(n-1)m} \qquad d \in \mathbb{R}^{h}$$

$$U \in \mathbb{R}^{|V|\times h} \qquad y \in \mathbb{R}^{|V|}$$



$$P(w_t|w_{t-1},\dots,w_{t-n+1}) = \frac{\exp(y_{w_t})}{\sum_i \exp(y_i)}$$

Real value(probability) Output(softmax)

NPLM의 학습 (5)

- (4)에서 계산된 스코어 벡터(y)에 $softmax$를 적용
- 원래 단어의 인덱스와 비교해 loss를 계산하여 역전파하는 방식으로 학습이 진행
- 학습이 완료되면 우리는 행렬 C를 각 단어에 해당하는 m 차원 임베딩으로 사용
- NPLM은 학습할 Parameter가 굉장히 많음
- 이후 제안된 단어 임베딩 기법은 추정 대상 파라미터를 줄이고 그 품질은 높이는 쪽으로 발전해옴

The cat is walking in the bedroom
A dog was running in a room
The cat is running in a room
A dog is walking in a bedroom
The dog was walking in the room

The cat is walking in the bedroom
A dog was running in a room
The cat is running in a room
A dog is walking in a bedroom
The dog was walking in the room

$$C \in \mathbb{R}^{|V| \times m}$$

$$\begin{bmatrix} 11 & 18 & 25 \\ 10 & 12 & 19 \\ \vdots & \vdots & \vdots \\ 23 & 5 & 7 \\ 17 & 24 & 1 \end{bmatrix}$$

NPLM과 의미 정보

- $n = 4$
- walking을 맞추는 과정에서 The, A, cat, dog, is, was 등은 walking이라는 단어와 모종의 관계가 형성
- The, A, cat, dog, is, was 등에 해당하는 C 행렬의 행 벡터들은 walking을 맞추는 과정에서 발생한 학습 손실(train loss)을 최소화하는 그래디언트(gradient)를 받아 동일하게 업데이트됨
- 결과적으로는 The, A, cat, dog, is, was 벡터가 vector space 안에서 같은 방향으로 조금 움직임

4.1 NPLM: Neural Probabilistic Language Model

본문에서 발췌... p118

NPLM은 그 자체로 언어 모델 역할을 수행할 수 있다. 예컨대 학습 데이터에 없는 The mouse is running in a room 이라는 문장의 등장 확률을 예측해야 한다고 가정해보자. 4.1절에서 설명한 기존의 통계 기반 n-gram 모델은 학습 데이터에 한 번도 등장하지 않은 패턴에 대해서는 그 등장 확률을 0으로 부여하는 문제점을 가지고 있다. 하지만 NPLM은 The mouse is running in a room이라는 문장이 말뭉치에 없어도 문맥이 비슷한 다른 문장을 참고해 확률을 부여한다. 결과적으로 NPLM은 The mouse is running in a room의 등장 확률을 그림 4-3의 문장들과 비슷하게 추론하게 되는 것이다.

Question!

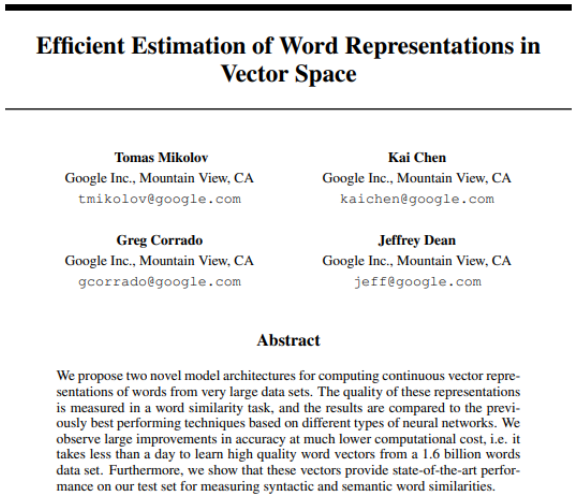
- 가능한가? Vocab에 없는 단어인데 모델에 넣어서 inference 할 수 있는가?
- 〈UNK〉를 vocab에 넣어주나?

4.2 Word2Vec

Efficient Estimation of Word Representations in Vector Space
– Mikolov et al., 2013a

Distributed Representations of Words and Phrase and their Compositionality
– Mikolov et al., 2013b

Mikolov et al., 2013a



Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

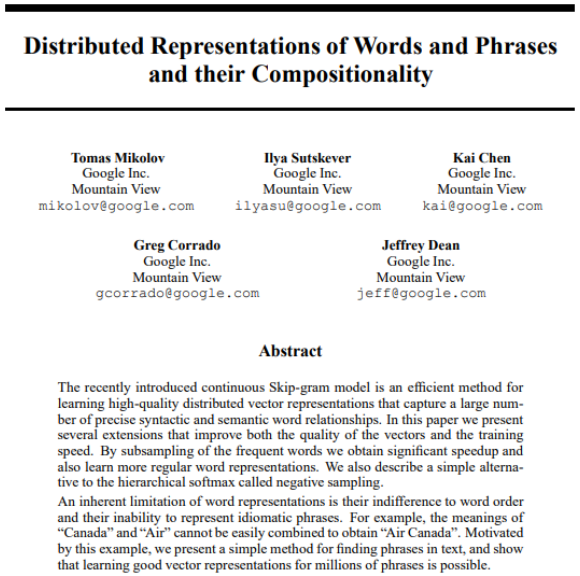
Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

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.

Mikolov et al., 2013b



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada". Motivated by this example, we present a simple method for finding phrases in text, and show that learning good vector representations for millions of phrases is possible.

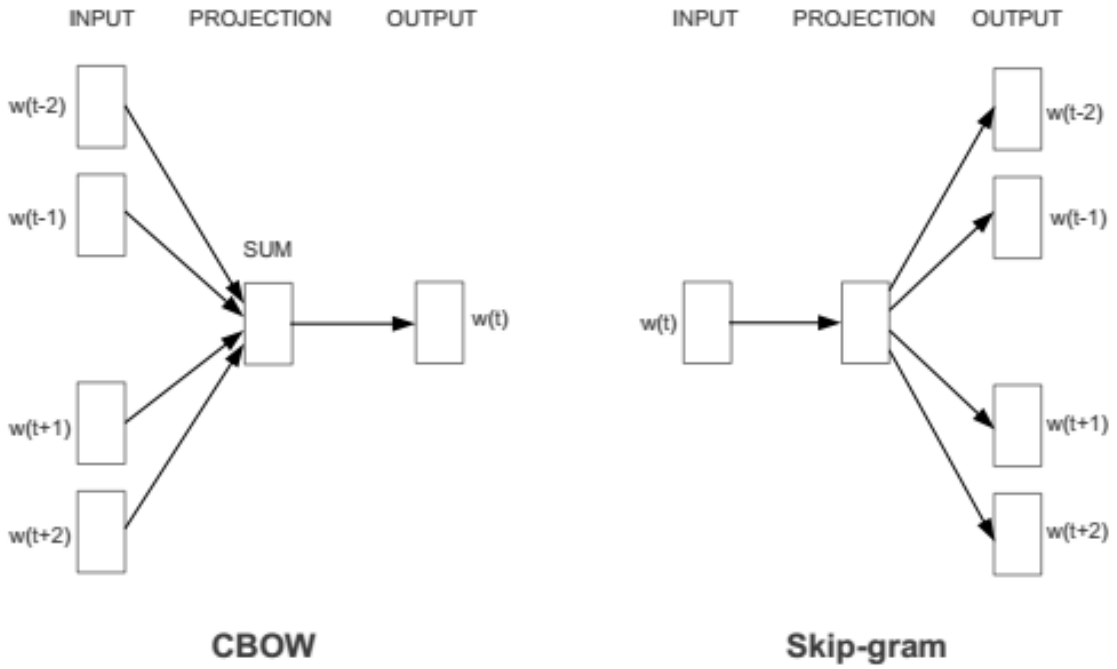
What is Word2Vec?

- 2013년 구글 연구 팀이 발표한 단어 임베딩 기법
- 왼쪽의 두 논문으로 나눠서 발표
- Mikolov et al., (2013a)에서는 Skip-Gram과 CBOW 모델 제시
- Mikolov et al., (2013b)에서는 두 모델을 근간으로 하되 네거티브 샘플링(negative sampling) 등 학습 최적화 기법을 제안

- Efficient Estimation of Word Representations in Vector Space
  https://arxiv.org/abs/1301.3781
- Distributed Representations of Words and Phrase and their Compositionality
  https://arxiv.org/abs/1310.4546

4.2 Word2Vec

Word2Vec models



What is Word2Vec?

- CBOW (Continuous Bag of Words):

  문맥 단어(context word)를 가지고

  타깃 단어(target word)를 예측

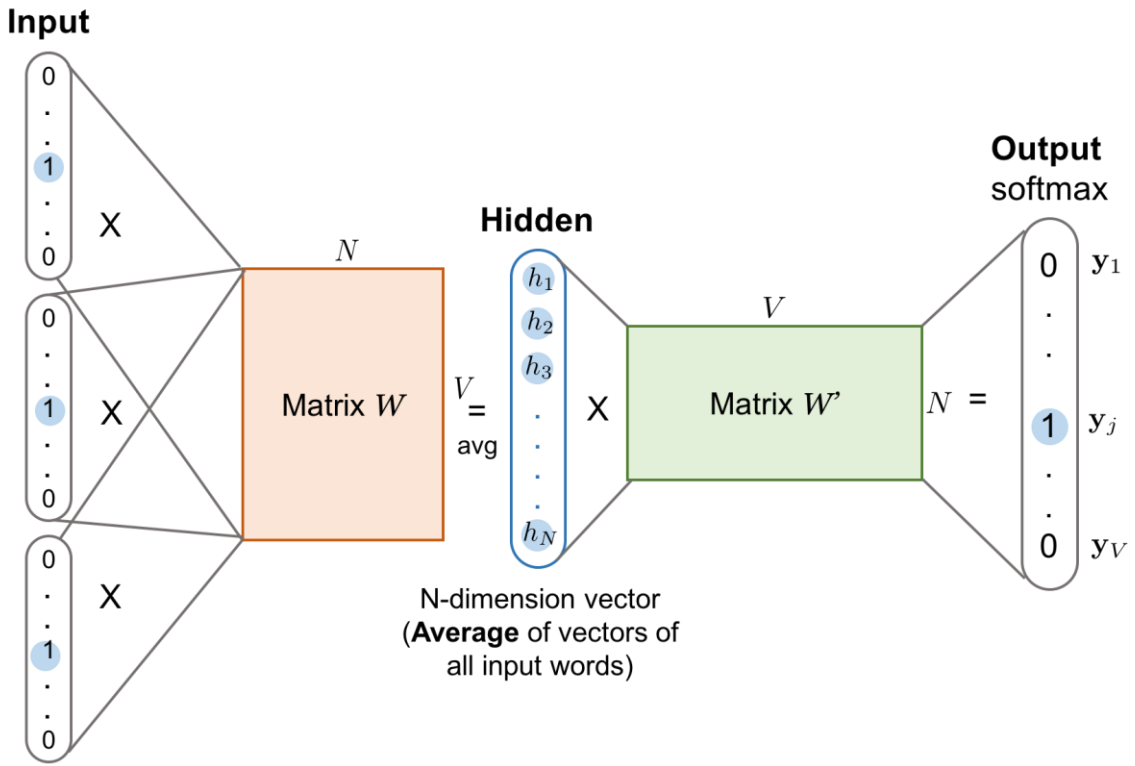
- Skip-gram:

  타깃 단어(target word)를 가지고

  문맥 단어(context word)를 예측

  CBOW보다 같은 말뭉치로도 더 많은 학습 데이터를 확보할 수 있어 임베딩 품질이 더 좋은 경향이 있음

CBOW (Continuous Bag of Words)

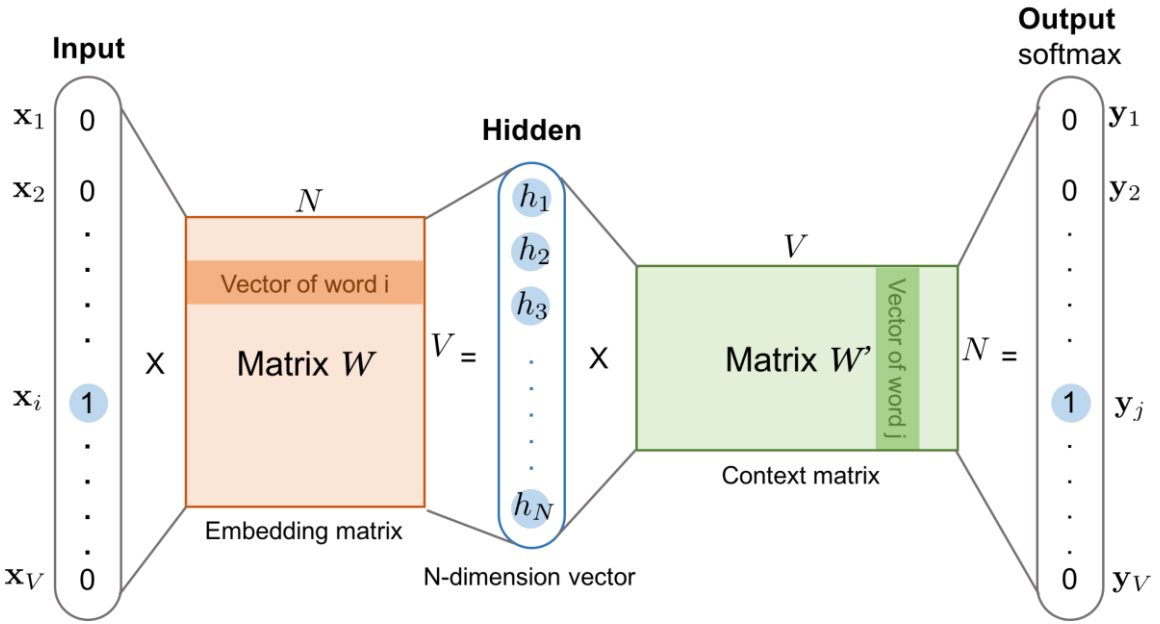


출처:
https://github.com/nlp-kkmas/korean-embedding-study
https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

Word2Vec: CBOW

1. 문맥 단어들(context words)의 one-hot vector와 행렬 W을 곱하고(look-up) 평균을 냄
2. 앞에서 구한 Hidden layer 벡터 값에 행렬 W'를 곱하여 각 타깃 단어(target word)에 대한 score를 생성
3. $softmax$로 확률 값 계산
4. Loss function으로 Loss 계산 후, 역전파 (backpropagation)

Skip-gram



출처:
https://github.com/nlp-kkmas/korean-embedding-study
https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html

Word2Vec: Skip-gram

1. (target, context) 쌍의 데이터 준비
2. 타깃 단어(target word)를 one-hot vector로 만들고, 행렬 W와 곱하여 embedded vector를 구함
3. Embedded vector를 두 번째 파라미터 행렬 W'와 곱해서 score vector로 만듦
4. 각 score vector를 softmax 함수를 통해 확률 값 계산
5. Loss function으로 Loss 계산 후, 역전파

✓ $positive\ sample: (target\ word, context\ word)$

✓ $negative\ sample: (target\ word, random\ word)$

… 개울가 (에서 속옷 **빨래** 를 하는) 남녀 …

$positive\ sample$

t	c
빨래	에서
빨래	속옷
빨래	를
빨래	하는

$negative\ sample$

t	c
빨래	책상
빨래	안녕
빨래	자동차
빨래	숫자

t	c
빨래	커피
빨래	떡
빨래	사과
빨래	노트북

$window\ size = 2$

Skip-gram 학습 데이터 구축 (1)

- 포지티브 샘플(positive sample): 타깃 단어와 그 주변에 실제로 등장한 문맥 단어 쌍
- 네거티브 샘플(negative sample): 타깃 단어와 그 주변에 등장하지 않은 단어(말뭉치 전체에서 랜덤 추출) 쌍
- Skip-gram 모델은 전체 말뭉치를 단어별로 슬라이딩해 가면서 학습 데이터를 만듦
- 결과적으로 Skip-gram 모델은 같은 말뭉치를 두고도 엄청나게 많은 학습 데이터 쌍을 만들어낼 수 있다.

$$\theta = \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-u_w^T v_c)})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})$$

1 k

The **sigmoid** function
$\sigma(x) = \frac{1}{1+e^{-x}}$
is the 1D version of the softmax and can be used to model a probability

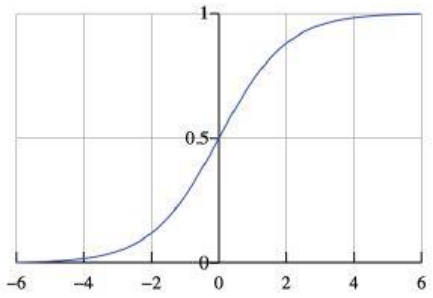


Figure 3: Sigmoid function

출처: https://dalpo0814.tistory.com/6

Skip-gram 학습 데이터 구축 (2)

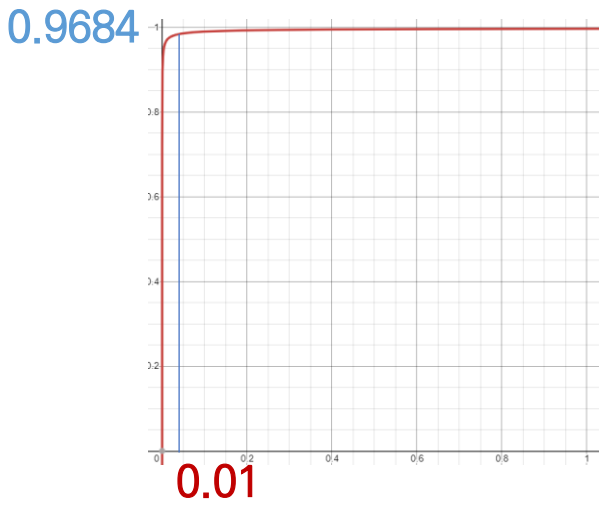
- Mikolov et al., (2013a)에선 타깃 단어가 주어졌을 때 문맥 단어가 무엇일지 맞추는 과정에서 학습
  → softmax 때문에 계산량이 비교적 큰 편
  → 정답 문맥 단어 확률 up, 나머지 단어 확률 down
  → 상당히 비 효율적
- Mikolov et al., (2013b)에서 제안한 skip-gram 모델은 타깃 단어와 문맥 단어 쌍이 주어졌을 때 해당 쌍이 포지티브 샘플(+)인지 네거티브 샘플(-)인지 이진 분류(binary classification)하는 과정에서 학습
- 이렇게 학습하는 기법을 네거티브 샘플링(negative sampling)이라고 부름
- 매 스텝마다 차원 수가 2인 sigmoid를 k+1회 계산
- Mikolov et al., (2013b)에 따르면 작은 데이터에는 k를 5~20, 큰 말뭉치에서는 2~5로 하는 것이 성능이 좋다고 한다.

✓ Negative sampling probability

$$P_{negative}(w_i) = \frac{U(w_i)^{3/4}}{\sum_{j=0}^{n} U(w_j)^{3/4}} \qquad where, U(w_i) = \frac{freq(w_i)}{|V|}$$

✓ Subsampling

$$P_{subsampling}(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \qquad where, t = 10^{-5}$$



Skip-gram 학습 데이터 구축 (3)

- 네거티브 샘플은 말뭉치에 자주 등장하지 않는 희귀한 단어가 네거티브 샘플로 조금 더 잘 뽑힐 수 있게 설계
- 그 수식은 왼쪽의 수식과 같음

- Mikolov et al., (2013b)에서는 이와 별개로 자주 등장하는 단어는 학습에서 제외하는 서브샘플링(subsampling) 기법도 적용
- 고빈도 단어의 경우 등장 횟수만큼 모두 학습시키는 것은 비 효율적

4.2 Word2Vec

✓ t, c가 positive sample(=t 주변에 c가 존재)일 확률

$$P(+|t,c) = \frac{1}{1+\exp(-u_t v_c)} \qquad U, V \in \mathbb{R}^{|V| \times d}$$

✓ t, c가 negative sample(=c를 t와 무관하게 말뭉치 전체에서 랜덤 샘플)일 확률

$$P(-|t,c) = 1 - P(+|t,c) = \frac{\exp(-u_t v_c)}{1+\exp(-u_t v_c)} \propto cosine\ similarity$$

✓ Skip-gram 모델의 로그우도 함수

$$\mathcal{L}(\theta) = \log P\left(+\middle|t_p, c_p\right) + \sum_{i=1}^{k} \log P(-|t_{n_i}, c_{n_i})$$

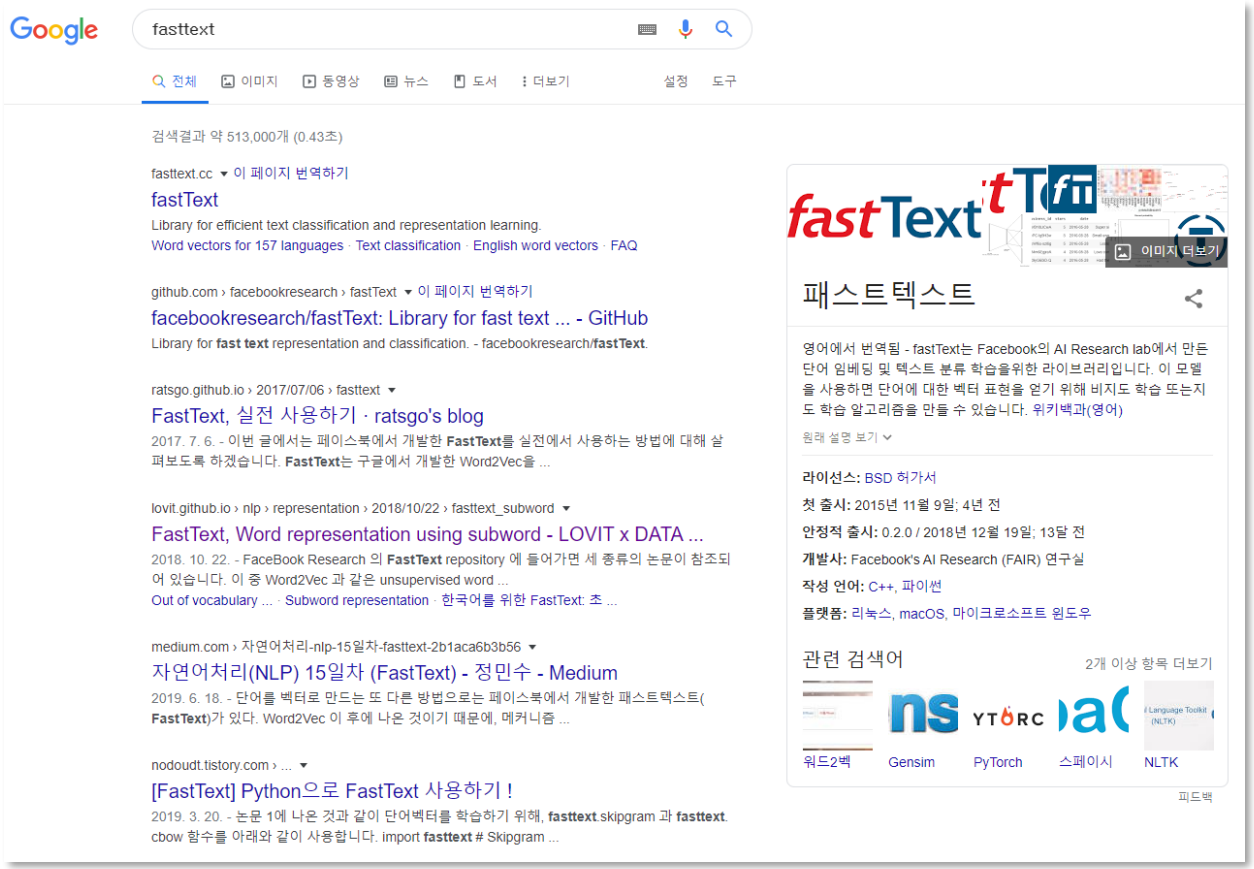
Skip-gram 모델 학습

- Part1: positive sample일 확률 최대화
- Part2: negative sample일 확률 최대화
- U: target word | V: context word 와 대응
- NPLM에 비해 parameter수 대폭 감소
- Part Final: skip-gram의 로그우도 함수(log-likelihood function)
- 모델 학습이 끝나면 U, $U + V^T$, $[U, V^T]$를 활용하여 단어 임베딩으로 사용할 수 있음

4.3 FastText

Enriching Word Vectors with Subword Information – Bonjanowski et al., 2016
: 개별 단어가 아닌 n-gram 단위의 character embedding 방법론

Advances in Pre-Training Distributed Word Representation – Mikolov et al., 2017
: FastText 완성



출처: https://github.com/nlp-kkmas/korean-embedding-study

What is FastText?

- 페이스북에서 개발해 공개한 단어 임베딩 기법
- 각 단어를 문자(Character) 단위 n-gram으로 표현
- 이 밖에는 Word2Vec과 완벽하게 동일

- Enriching Word Vectors with Subword Information
  https://arxiv.org/abs/1607.04606
- Advances in Pre-Training Distributed Word Representation
  https://arxiv.org/abs/1712.09405

- 기존 Word2Vec, GloVe의 문제?
  - 단어의 형태학적 특성을 반영하지 못함
  - 희소한 단어를 임베딩하기 어려움

4.3 FastText

✓ FastText의 단어 벡터 표현

$$u_{시나브로} = z_{<시나} + z_{시나브} + z_{나브로} + z_{브로>} + z_{<시나브로>}$$

$$u_t = \sum_{g \in G_t} z_g$$

✓ FastText 모델에서 t, c가 포지티브 샘플(=t 주변에 c가 존재)일 확률

$$P(+|t,c) = \frac{1}{1+\exp(-u_t v_c)} = \frac{1}{1+\exp(-\sum_{g \in G_t} z_g^T v_c)}$$

✓ FastText 모델에서 t, c가 네거티브 샘플(=c를 t와 무관하게 말뭉치 전체에서 랜덤 샘플)일 확률

$$P(+|t,c) = \frac{1}{1+\exp(-u_t v_c)} = \frac{1}{1+\exp(-\sum_{g \in G_t} z_g^T v_c)} = \frac{\exp(-\sum_{g \in G_t} z_g^T v_c)}{1+\exp(-\sum_{g \in G_t} z_g^T v_c)}$$

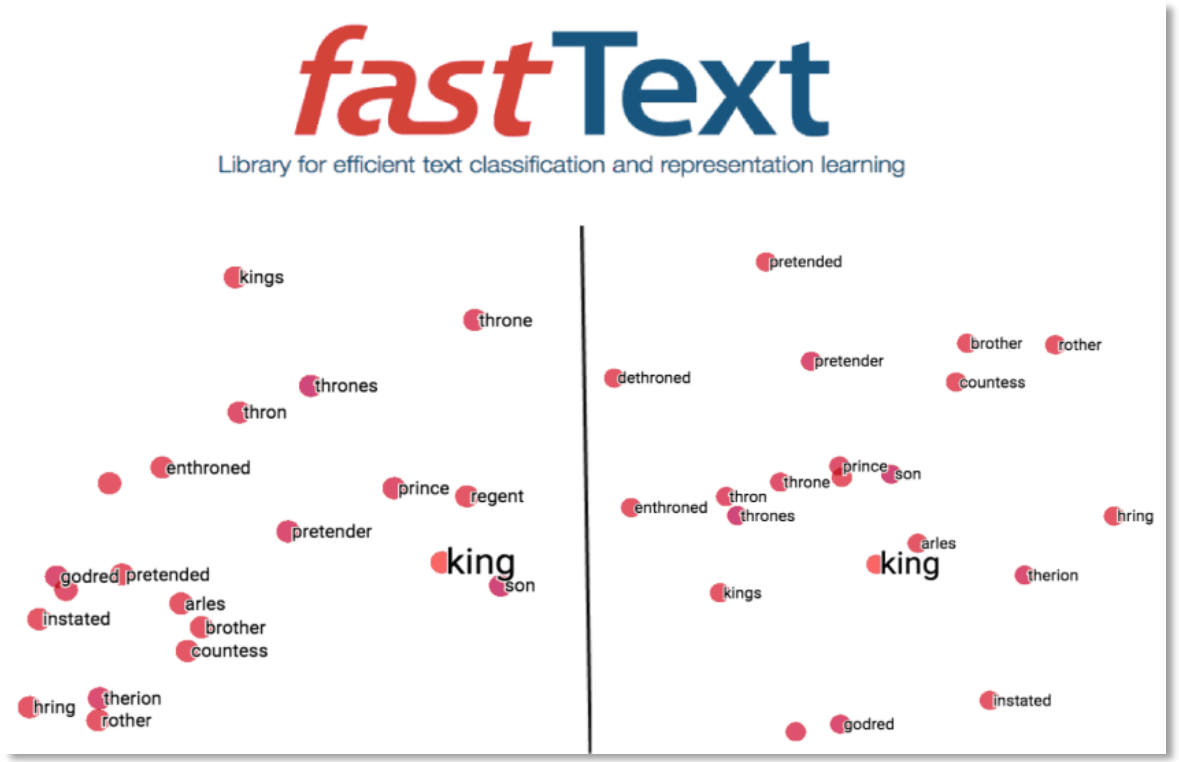
✓ FastText 모델의 로그우도 함수

$$\mathcal{L}(\theta) = \log P\left(+\middle|t_p, c_p\right) + \sum_{i=1}^{k} \log P(-|t_{n_i}, c_{n_i})$$

FastText 모델

- 시나브로라는 단어의 문자 단위 n-gram($n = 3$)
- 〈, 〉는 단어의 경계를 나타내 주기 위해 FastText 모델이 사용하는 특수 기호
- Word2Vec과 동일하게 negative sampling 사용
- Loss function 또한 동일
- 학습의 예시!
- Target: 시나브로 | Positive: 쌓였다
- 〈시나, 시나브, 나브로, 브로〉, 〈시나브로〉를 각각 쌓였다에 해당하는 단어 벡터와의 유사도를 높임
- Target: 시나브로 | Negative: 컴퓨터
- 〈시나, 시나브, 나브로, 브로〉, 〈시나브로〉를 각각 컴퓨터에 해당하는 단어 벡터와의 유사도를 높임

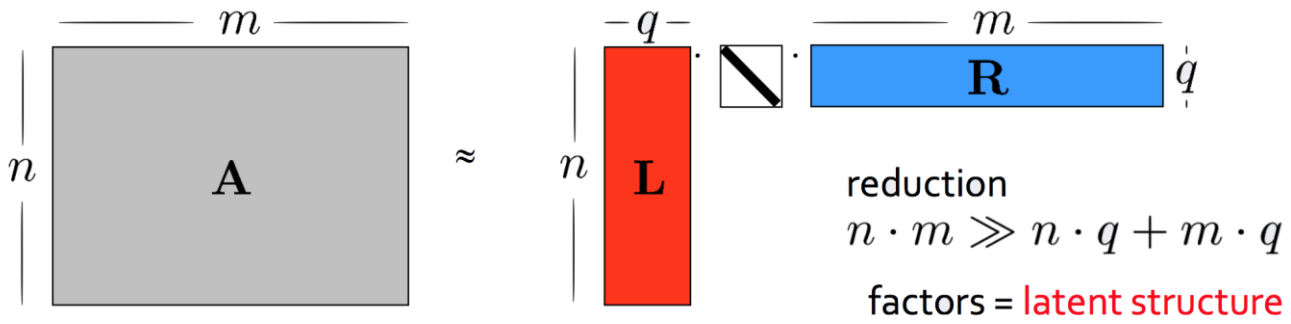
4.3 FastText



FastText 특징

- 조사나 어미가 발달한 한국어에 좋은 성능을 낼 수 있음
- 오타나 미등록 단어(unknown word)에 강건
- 문자 단위 n-gram을 쓰기 때문에 한글과 궁합이 좋음
- 단어 단위뿐만 아니라 한글을 자소 단위로 분해하여 사용 가능하다.
- 미등록 단어(unknown word)에 대한 임베딩을 추정할 수 있다.

4.4 LSA: Latent Semantic Analysis



What is LSA?

- 단어-문서 행렬, TF-IDF 행렬, 단어-문맥 행렬 같은 커다란 행렬에 차원 축소 방법의 일종인 특이값 분해를 수행해 데이터의 차원 수를 줄여 계산 효율성을 키우는 한편, 행간에 숨어있는 잠재 의미를 이끌어내기 위한 방법론
- 단어-문서 행렬, 단어-문맥 행렬 등에 특이값 분해를 시행한 뒤 그 결과로 도출되는 행 벡터들을 단어 임베딩으로 사용할 수 있다.
- 행렬 분해(Matrix Factorization) 기법
- 본 절에서는 PMI행렬에 대해 LSA를 적용하는 것을 살펴봄

출처: https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/05/25/plsa/

4.4 LSA: Latent Semantic Analysis

✓ 점별 상호 정보량(PMI)

$$PMI(A,B) = \log \frac{P(A,B)}{P(A) \times P(B)}$$

✓ 양의 점별 상호 정보량(PPMI)

$$PPMI(A,B) = \max(PMI(A,B), 0)$$

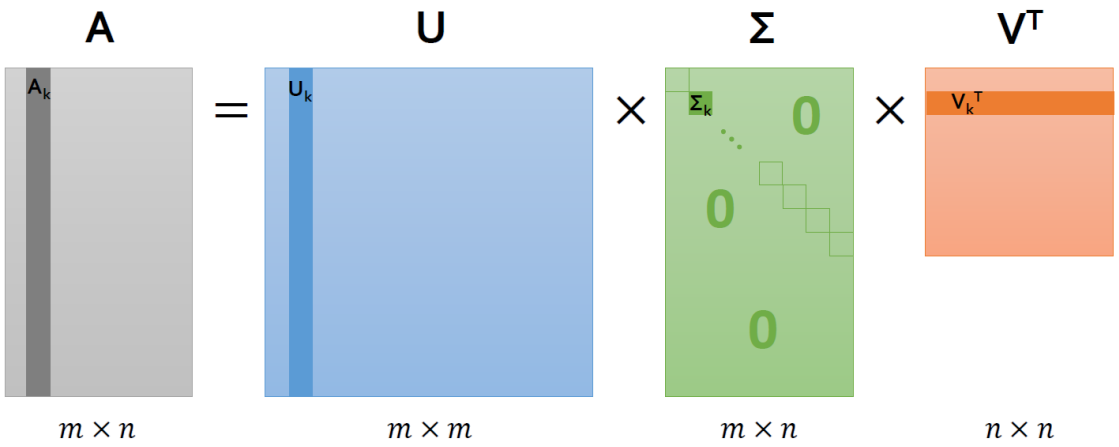
✓ Shifted PMI

$$SPMI(A,B) = PMI(A,B) - \log k \quad s.t\ k > 0$$

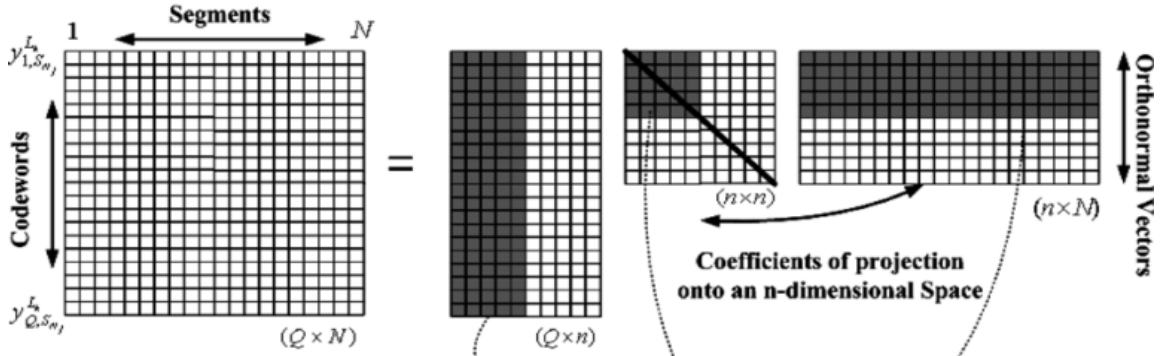
PPMI 행렬

- PMI(Pointwise Mutual Information): 두 확률변수 사이의 상관성을 계량화한 지표, 두 단어의 등장이 독립을 가정했을 때 대비 얼마나 자주 같이 등장하는지를 수치화한 것
- PMI는 A,B 두 단어가 동시에 등장할 확률이 두 단어가 독립일 때보다 작은 경우에 음수
- A, B가 한번도 같이 등장하지 않으면 log0이 됨
- 이를 위해 자연어 처리 분야세서는 PMI 대신 양의 점별 상호 정보량(PPMI, Positive Pointwise Mutual Information)을 사용
- Shifted PMI(SPMI)는 PMI에서 log(k)를 뺀 값
- Shifted PMI는 Word2Vec과 깊은 연관이 있다는 논문이 발표되기도 함

4.4 LSA: Latent Semantic Analysis



$$\textbf{SVD}: \quad M_{Q \times N}^{LIM} = T_{Q \times n} \times S_{n \times n} \times (D_{N \times n})^T$$



$$\textbf{Truncated SVD}: \quad \hat{M}_{Q \times N}^{LIM} = T_{Q \times \rho} \times S_{\rho \times \rho} \times (D_{N \times \rho})^T$$

Matrix Factorization to LSA

- 특이값 분해(Singular Value Decomposition)은 mxn 크기의 임의의 사각행렬 A를 왼쪽 상단 그림과 같이 분해하는 것
- Truncated SVD는 특이값 가운데 가장 큰 d개만 가지고, 해당 특이값에 대응하는 특이 벡터(singular vector)들로 원래 행렬 A를 근사
- m개 단어, n개 문서로 이루어진 단어-문서 행렬에 truncated SVD를 적용했다고 가정
- T는 단어 임베딩, V는 문서 임베딩에 대응
- n, m개 문서와 단어로 표현됐던 벡터들이 d차원만으로 커버 가능
- 각종 연구에 따르면 LSA를 적용하면 단어와 문맥 간의 내재적인 의미(Latent/Hidden meaning)을 효과적으로 보존할 수 있게 돼 결과적으로 문서 간 유사도 측정 등 모델의 성능 향상에 도움을 줄 수 있다고 보고
- 입력 데이터의 noise, sparsity를 줄이는 것도 가능

4.4 LSA: Latent Semantic Analysis

4.5 GloVe

4.5 GloVe