Introduction to Deep Learning for Natural Language Processing

3. Word Embedding

앞으로 배우게 될 내용

- Text preprocessing for NLP & Language Model
- Basic Model & Vectorization
- **Word Embedding (Word2Vec, FastText, GloVe)**
- Text Classification (using RNN & CNN)
- Chatbot with Deep Learning
- Sequence to Sequence
- Attention Mechanism
- Transformer & BERT

교재 : https://wikidocs.net/book/2155

Language Model (복습)

Language Model

- Language Model, 줄여서 LM이란 단어 시퀀스에 확률을 할당하는 모델.
- 단어 시퀀스에 확률을 할당하는 이유? LM을 통해 더 그럴듯한 문장을 선택할 수 있으니까.

아래에서 P는 확률(Probability)을 나타낸다.

기계 번역(Machine Translation)
- **P(나는 버스를 탔다)** > P(나는 버스를 태운다)

음성 인식(Spell Correction)
- P(나는 메롱을 먹는다) < **P(나는 메론을 먹는다)**

오타 교정(Spell Correction)
- **P(선생님이 달려갔다)** > P(선생님이 잘려갔다)

<검색 엔진에서의 언어 모델이 동작하는 예 : 다음 단어 예측>

Language Model

- Languae Model은 어떤 문장이 가장 그럴듯한지를 판단한다.
- Languae Model은 주어진 단어 시퀀스 다음에 어떤 단어가 등장해야 자연스러운지를 판단한다.

1. 전체 단어 시퀀스의 확률

- $P(W) = P(w_1, w_2, w_3, w_4, w_{5,\dots,} w_n)$ = n개의 단어가 동시에 등장할 확률

2. 이전 단어들이 주어졌을 때 다음 단어의 등장 확률

다섯번째 단어의 등장 확률은 다음과 같이 표현할 수 있다.

$$P(w_5|w_1, w_2, w_3, w_4)$$

일반화하면 전체 단어 시퀀스의 확률은 아래와 같이 정의할 수 있다.

- $P(W) = P(w_1, w_2, w_3, w_4, w_{5,\dots,} w_n) = \prod_{i=1}^{n} P(w_n|w_1, \dots, w_{n-1})$

Language Model

- Languae Model은 어떤 문장이 가장 그럴듯한지를 판단한다.
- Languae Model은 주어진 단어 시퀀스 다음에 어떤 단어가 등장해야 자연스러운지를 판단한다.

예문으로 보는 단어 시퀀스 확률

$$P(W) = P(w_1, w_2, w_3, w_4, w_{5,\dots,} w_n)$$
$$= \prod_{i=1}^{n} P(w_n|w_1, \dots, w_{n-1})$$

위 수식에 따라서 P("its water is so transparent") =

P(its) × P(water|its) × P(is|its water)

× P(so|its water is) × P(transparent|its water is so)

Statistical Language Model

- **통계적 방법** : 훈련 데이터에서 단순 카운트하고 나누는 것은 어떨까?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

통계적 언어 모델의 접근 방법 :

Its water is so transparent that 다음에 the가 나올 확률은 전체 훈련 데이터에서
Its water is so transparent that the의 등장 횟수를 카운트한 것을 분자로
Its water is so transparent that의 등장 횟수를 카운트한 것을 분모로 한다.

Statistical Language Model

- **통계적 방법** : 훈련 데이터에서 단순 카운트하고 나누는 것은 어떨까?

$$P(\text{the} \mid \text{its water is so transparent that}) = \frac{Count(\text{its water is so transparent that the})}{Count(\text{its water is so transparent that})}$$

훈련 데이터가 정-말 방대하지 않은 이상 제대로 카운트할 수 있는 경우가 거의 없다.

통계적 언어 모델의 접근 방법 :

Its water is so transparent that 다음에 the가 나올 확률은 전체 훈련 데이터에서
Its water is so transparent that the의 등장 횟수를 카운트한 것을 분자로
Its water is so transparent that의 등장 횟수를 카운트한 것을 분모로 한다.

Statistical Language Model

훈련 데이터가 정-말 방대하지 않은 이상 제대로 카운트할 수 있는 경우가 거의 없다.

방대한 데이터의 집합체인 구글 검색 엔진에서조차 '그 물은 투명했다' 라는 간단한 문장이 포함된 결과가 나오지 않는다.

언어라는 것은 너무나 가능한 경우의 수가 많다.

단순 카운트 방법으로 언어를 모델링하기에는 그만큼의 방대한 훈련 데이터를 가지기 어렵다.

N-gram Language Model

- n-gram은 n개의 연속적인 단어 나열을 의미하며 n은 사용자가 정하는 값이다.
- 이전 단어들의 주어졌을 때 다음 단어의 등장 확률의 추정을 앞의 n-1개의 단어에만 의존한다.

' An adorable little boy is spreading ' 다음에 나올 단어를 예측하고자 하며 n이 4일 때 (예시)

~~An adorable little~~ boy is spreading ?
무시됨!
n-1개의 단어

$$P(w|\text{boy is spreading}) = \frac{\text{count(boy is spreading } w)}{\text{count(boy is spreading)}}$$

통계적 언어 모델

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_n|w_1, \ldots, w_{n-1})$$

N-gram 언어 모델 (if n=4)

$$P(w_1, w_2, \ldots, w_n) \approx \prod_{i=1}^{n} P(w_n|w_{n-3}, w_{n-2}, w_{n-1})$$

Bigram Language Model (n=2)

- Bigram Language Model로 생성한 문장은 다음 단어 생성이 오직 이전 단어에만 의존한다.

Bigram Language Model이 생성한 문장

texaco, rose, one, in, this, issue, is, pursuing, growth, in,
a, boiler, house, said, mr., gurria, mexico, 's, motion,
control, proposal, without, permission, from, five, hundred,
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

Bigram Language Model (n=2)

- n=2이면 Bigram Language Model이라고 하며 오직 이전 단어 하나만 고려하여 카운트하여 확률 추정.
- n이 작으면 얼토당토 않는 문장이 되버리지만 n이 크면 카운트하기가 어려워짐. 적절한 n 선정이 중요.

1. 다음 단어 등장 확률 추정

$$P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$$

3. 전체 단어 시퀀스의 확률 추정

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_n \mid w_{n-1})$$

2. 확률 추정은 훈련 데이터에서 카운트에 기반함.

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Bigram Language Model Example

다음과 같이 3개의 문장을 가진 훈련 데이터가 있다고 하자.
<s>와 </s>는 special token으로 <s>는 문장의 시작, </s>는 문장의 끝을 의미한다.

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

계산식

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$P(\text{I} \mid \text{<s>}) = ?$ $P(\text{Sam} \mid \text{<s>}) = ?$

Bigram Language Model Example

다음과 같이 3개의 문장을 가진 훈련 데이터가 있다고 하자.
<s>와 </s>는 special token으로 <s>는 문장의 시작, </s>는 문장의 끝을 의미한다.

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

계산식

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$P(\text{I} \mid \text{<s>}) = \frac{2}{3} = .67$ $P(\text{Sam} \mid \text{<s>}) = \frac{1}{3} = .33$

Bigram Language Model Example

다음과 같이 3개의 문장을 가진 훈련 데이터가 있다고 하자.
<s>와 </s>는 special token으로 <s>는 문장의 시작, </s>는 문장의 끝을 의미한다.

<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>

계산식

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P(\texttt{I} \mid \texttt{<s>}) = \frac{2}{3} = .67 \qquad P(\texttt{Sam} \mid \texttt{<s>}) = \frac{1}{3} = .33 \qquad P(\texttt{am} \mid \texttt{I}) = \frac{2}{3} = .67$$

$$P(\texttt{</s>} \mid \texttt{Sam}) = \frac{1}{2} = 0.5 \qquad P(\texttt{Sam} \mid \texttt{am}) = \frac{1}{2} = .5 \qquad P(\texttt{do} \mid \texttt{I}) = \frac{1}{3} = .33$$

Sparsity Problem

- N-gram 언어 모델은 앞의 단어를 n-1개만 고려하므로 n이 너무 작으면 **장기 의존성 문제 발생.**
- n이 커지면 훈련 데이터 내에서 카운트 자체를 하지 못해서 제대로 된 모델링이 되지 않음.
- **카운트 자체를 하지 못할 경우 분자 또는 분모가 0이 되는 상황 발생.**

$$P(\text{is} \mid \text{An adorable little boy}) = \frac{\text{count}(\text{An adorable little boy is})}{\text{count}(\text{An adorable little boy})}$$

카운트가 0이면?

카운트가 0이면?

충분한 데이터를 관측하지 못하여
언어를 정확히 모델링하지 못하는 문제를
희소 문제(sparsity problem)라고 한다.

Sparsity Problem

- N-gram 언어 모델은 앞의 단어를 n-1개만 고려하므로 n이 너무 작으면 **장기 의존성 문제 발생.**
- n이 커지면 훈련 데이터 내에서 카운트 자체를 하지 못해서 제대로 된 모델링이 되지 않음.
- **카운트 자체를 하지 못할 경우 분자 또는 분모가 0이 되는 상황 발생.**

카운트가 0이면?

$$P(\text{is}|\text{An adorable little boy}) = \frac{\text{count}(\text{An adorable little boy is})}{\text{count}(\text{An adorable little boy })}$$

카운트가 0이면?

충분한 데이터를 관측하지 못하여
언어를 정확히 모델링하지 못하는 문제를
희소 문제(sparsity problem)라고 한다.

인공 신경망 언어모델(NNLM)의 시대로.

Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룬다.
- 인공 신경망 언어 모델은 **희소 문제(sparsity problem)**를 Word Embedding으로 해결하였다.

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있다.

귀여운 ≈ 사랑스러운 **강아지 ≈ 고양이** **좋아해 ≈ 사랑해**

Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룬다.
- 인공 신경망 언어 모델은 **희소 문제(sparsity problem)**를 Word Embedding으로 해결하였다

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있다.

귀여운 ≈ 사랑스러운 **강아지 ≈ 고양이** **좋아해 ≈ 사랑해**

이로부터 훈련 데이터에 없던 시퀀스라도 생성이 가능한다.

Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룬다.
- 인공 신경망 언어 모델은 **희소 문제(sparsity problem)**를 Word Embedding으로 해결하였다

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있다.

귀여운 ≈ 사랑스러운 **강아지 ≈ 고양이** **좋아해 ≈ 사랑해**

이로부터 훈련 데이터에 없던 시퀀스라도 생성이 가능한다.

훈련 데이터 : 귀여운 강아지 좋아해 ⟶

Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룬다.
- 인공 신경망 언어 모델은 **희소 문제(sparsity problem)**를 Word Embedding으로 해결하였다

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있다.

귀여운 ≈ 사랑스러운 **강아지 ≈ 고양이** **좋아해 ≈ 사랑해**

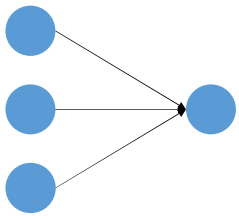
이로부터 훈련 데이터에 없던 시퀀스라도 생성이 가능한다.

훈련 데이터 : 귀여운 강아지 좋아해 → **사랑스러운 고양이 사랑해** **(훈련 데이터에 없던 시퀀스)**

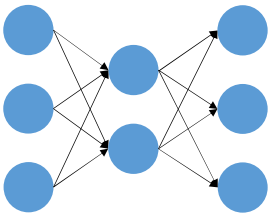
Neural Network Language Model

다양한 신경망 용어들

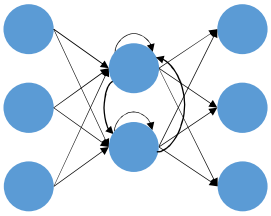
- 피드 포워드 신경망(Feed Forward Neural Network)이란 입력층에서 출력층 방향으로 향하는 신경망.
- 피드 포워드 신경망 중 가장 대표적이고 간단한 신경망은 다층 퍼셉트론(Multilayer Perceptron).



단층 퍼셉트론



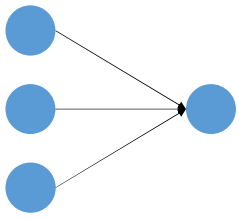
다층 퍼셉트론(Multilayer Perceptron, MLP)
피드 포워드 신경망
(Feed Forward Neural Network, FFNN)



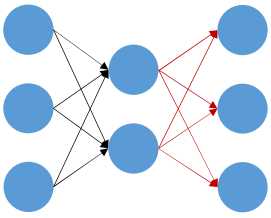
순환 신경망
(Recurrent Neural Network, RNN)

다양한 신경망 용어들

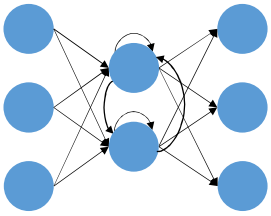
- 피드 포워드 신경망(Feed Forward Neural Network)이란 입력층에서 출력층 방향으로 향하는 신경망.
- 피드 포워드 신경망 중 가장 대표적이고 간단한 신경망은 다층 퍼셉트론(Multilayer Perceptron).



단층 퍼셉트론



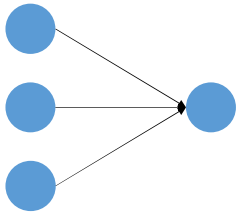
다층 퍼셉트론(Multilayer Perceptron, MLP)
피드 포워드 신경망
(Feed Forward Neural Network, FFNN)



순환 신경망
(Recurrent Neural Network, RNN)

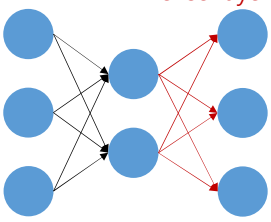
다양한 신경망 용어들

- 피드 포워드 신경망(Feed Forward Neural Network)이란 입력층에서 출력층 방향으로 향하는 신경망.
- 피드 포워드 신경망 중 가장 대표적이고 간단한 신경망은 다층 퍼셉트론(Multilayer Perceptron).

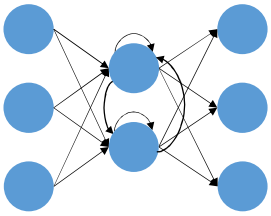


단층 퍼셉트론

Fully-connected layer 또는
Dense layer



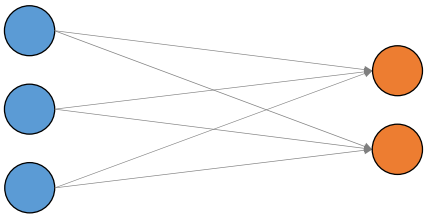
다층 퍼셉트론(Multilayer Perceptron, MLP)
피드 포워드 신경망
(Feed Forward Neural Network, FFNN)



순환 신경망
(Recurrent Neural Network, RNN)

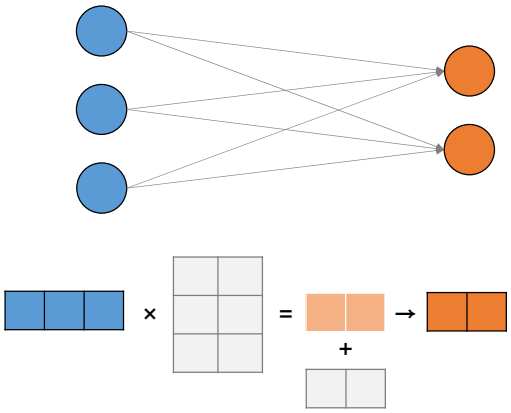
Feed Forward Neural Network

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



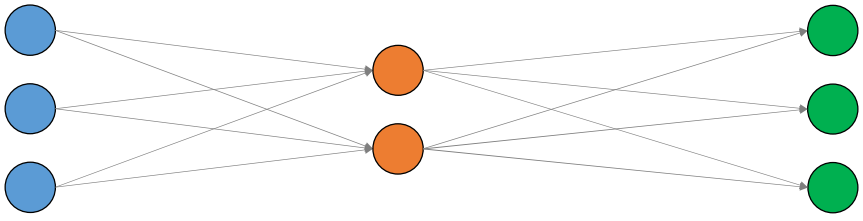
Feed Forward Neural Network

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



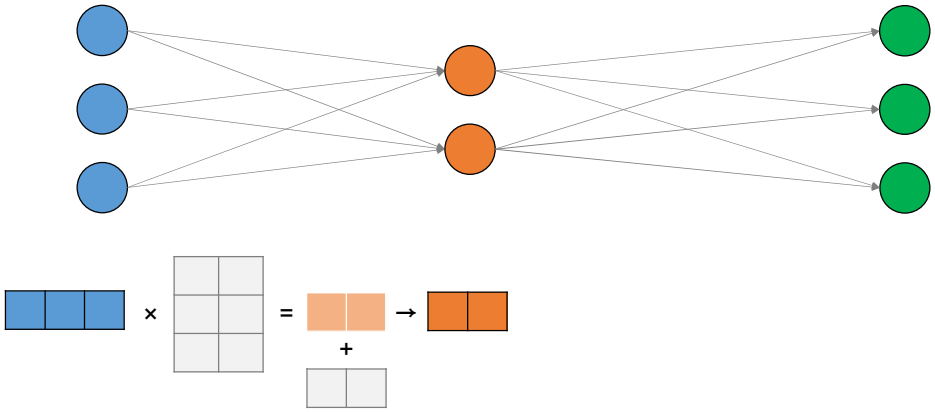
Feed Forward Neural Network

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



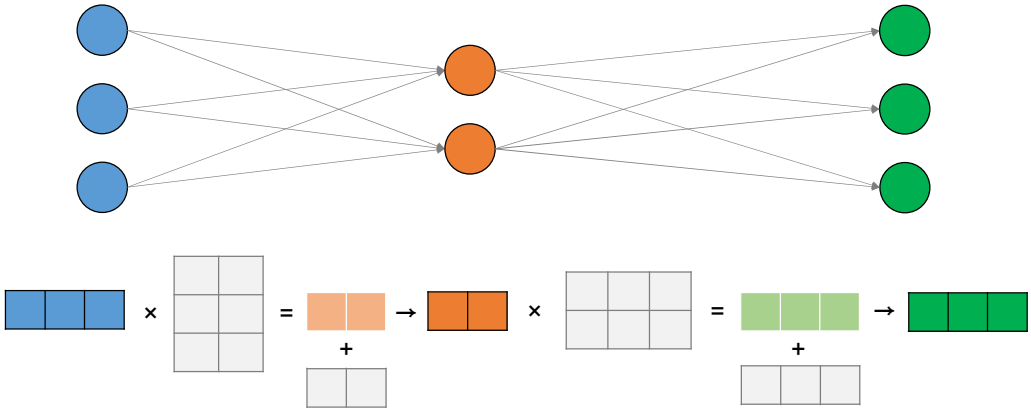
Feed Forward Neural Network

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



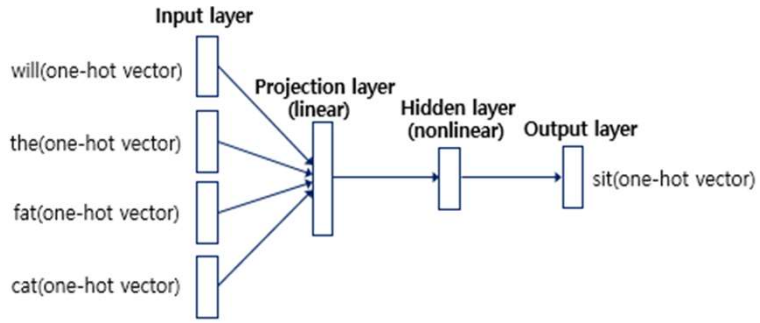
Feed Forward Neural Network

신경망을 행렬과 벡터 연산으로 이해할 수 있다면, 신경망을 이해하기가 매우 용이하다.
다음 신경망의 파라미터의 개수는?



NNLM(Neural Network Language Model)

피드 포워드 신경망으로 구현한 Language Model.
Language Model이므로 이전 단어들로부터 다음 단어를 예측한다.



NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

↓ 원-핫 인코딩

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

원-핫 인코딩

sit을 예측한다고 하면 n은 4이므로 will, the, fat, cat으로부터 sit을 예측한다.

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

원-핫 인코딩

sit을 예측한다고 하면 n은 4이므로 will, the, fat, cat으로부터 sit을 예측한다.

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

입력

will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]

NNLM(Neural Network Language Model)

NNLM은 n개의 이전 단어들로부터 n+1 단어를 예측하는 모델.
신경망의 입력은 원-핫 인코딩을 사용하여 얻은 원-핫 벡터로 한다.
다음과 같은 예문을 훈련한다고 해보자.

예문 : "what will the fat cat sit on"

원-핫 인코딩

sit을 예측한다고 하면 n은 4이므로 will, the, fat, cat으로부터 sit을 예측한다.

what = [1, 0, 0, 0, 0, 0, 0]
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]
sit = [0, 0, 0, 0, 0, 1, 0]
on = [0, 0, 0, 0, 0, 0, 1]

입력

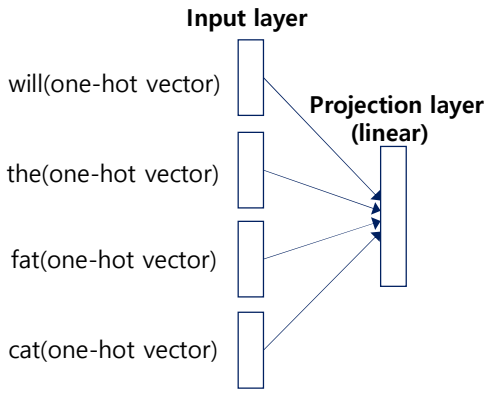
will = [0, 1, 0, 0, 0, 0, 0]
the = [0, 0, 1, 0, 0, 0, 0]
fat = [0, 0, 0, 1, 0, 0, 0]
cat = [0, 0, 0, 0, 1, 0, 0]

예측 대상(출력)

sit = [0, 0, 0, 0, 0, 1, 0]

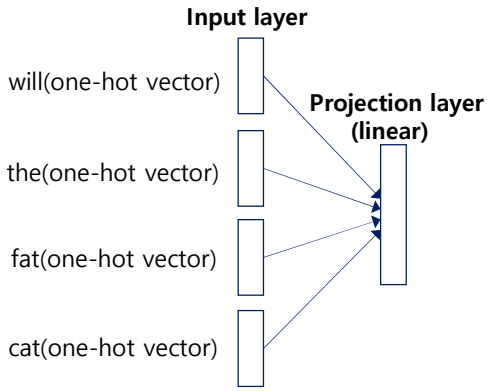
Projection layer

입력층(Input layer)와 투사층(Projection layer) 사이의 연산을 생각해보자.
각각의 원-핫 벡터는 가중치 행렬과 곱해져서 투사층을 형성할 것이다.



Projection layer

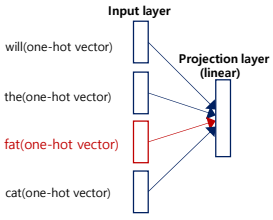
입력층(Input layer)와 투사층(Projection layer) 사이의 연산을 생각해보자.
각각의 원-핫 벡터는 가중치 행렬과 곱해져서 투사층을 형성할 것이다.



투사층의 크기를 M이라고 한다면?

Projection layer

- V차원의 원-핫 벡터는 V x M 크기의 행렬과 곱해져 M차원의 벡터를 얻는다.
- i번째 인덱스에 1의 값을 가지는 원-핫 벡터와 가중치 W 행렬의 곱은 W행렬의 i번째 행을 그대로 읽어오는 것과(lookup) 동일하다.
- 이 연산을 **lookup table**이라 한다.



x_{fat} × $W_{V\times M}$ = e_{fat}

0	0	0	1	0	0	0	×

0.5	2.1	1.9	1.5	0.8
0.8	1.2	2.8	1.8	2.1
0.1	0.8	1.2	0.9	0.7
2.1	1.8	1.5	1.7	2.7

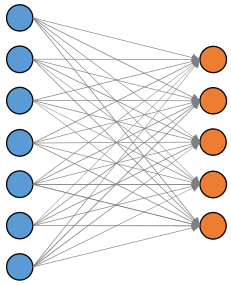
=

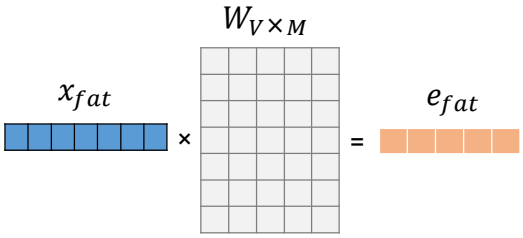
2.1	1.8	1.5	1.7	2.7

lookup table

Projection Layer

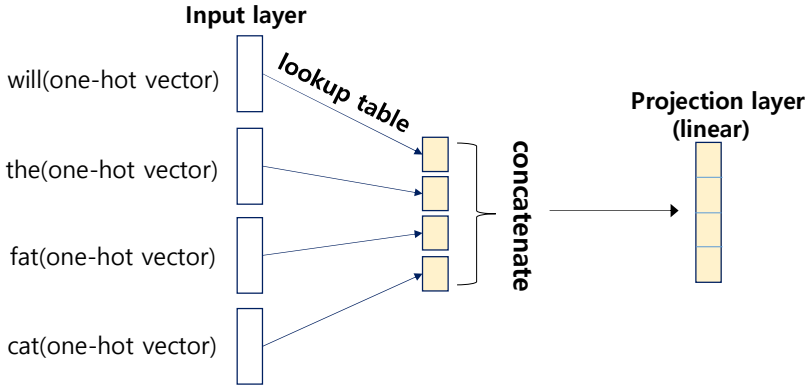
- Projection Layer는 신경망 그림으로 해석한다면 다음과 같다.
- 단, Weight Matrix는 존재하지만 Bias는 사용하지 않는다.
- 활성화 함수도 사용하지 않는다.





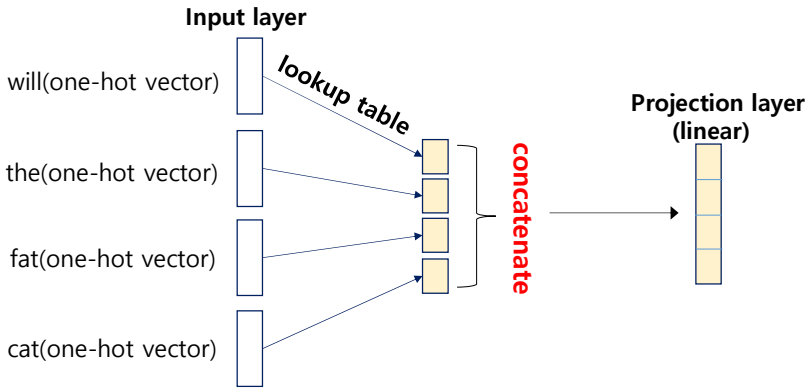
Projection layer

각 원-핫 벡터가 lookup table을 거치면 모두 concatenate된다.
여기서 concatenate란 벡터를 단순히 나열하는 것을 의미한다.
Ex) 5차원 벡터 4개를 concatenate하면 20차원 벡터가 된다.



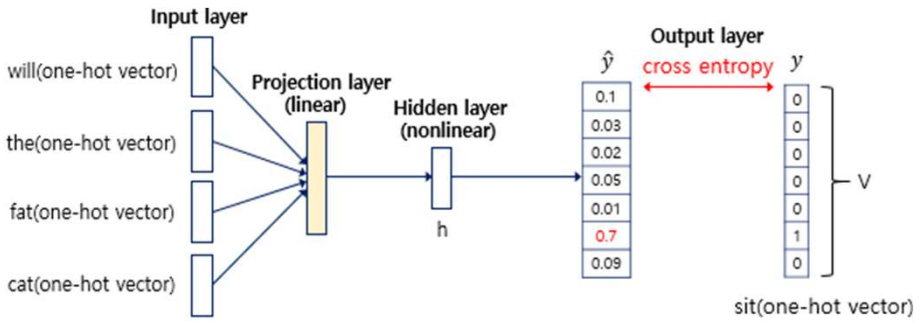
Projection layer

- 벡터의 concatenate는 딥 러닝 연산에서 정보를 모두 사용한다는 의미로 두 개 이상의 정보(벡터)를 함께 사용하고자 할 때 쓰는 연산이다.



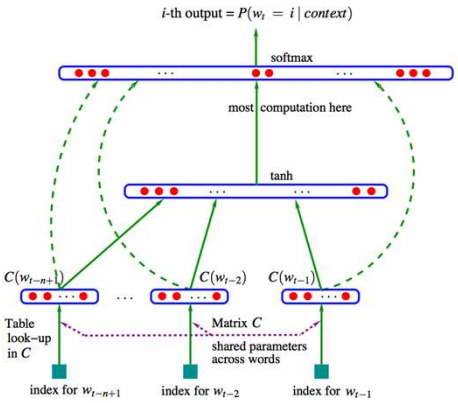
Hidden layer and Output layer

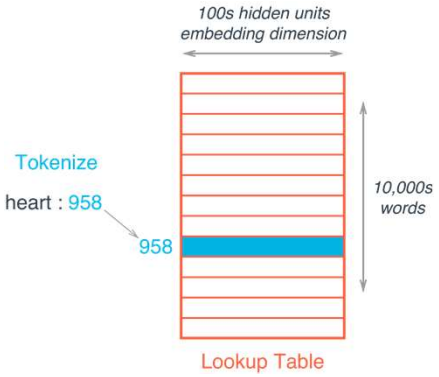
Projection layer 이후, 하나의 Hidden layer를 거치고 Output layer에서 실제값인 다음 단어 원-핫 벡터로부터 loss를 구하고 학습된다.
이 과정에서 Projection layer의 embedding table이 학습된다.



NNLM(Neural Network Language Model)

다른 그림으로 이해해보자.



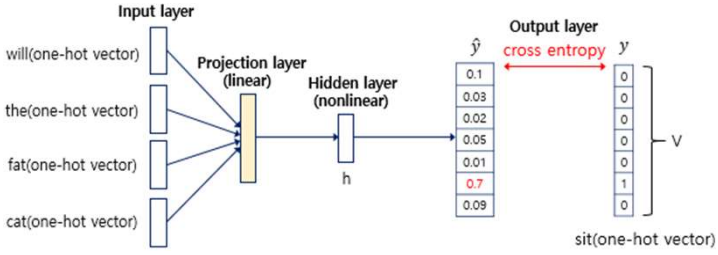


NNLM(Neural Network Language Model)

투사층의 가중치 행렬 W의 각 행은 각 단어와 맵핑되는 밀집 벡터(Dense Vector)이다.
훈련 과정에서 유사한 용도로 사용되는 단어들은 유사한 단어 벡터값을 가지게 된다.

ex) 만약 a cute dog를 학습했다면, a cute cat이 훈련 데이터에 없더라도 a cute cat을 생성 가능.

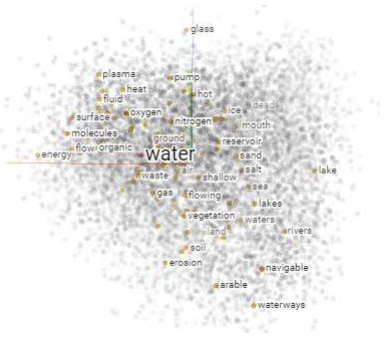
→ 희소 문제(sparsity problem)의 해결

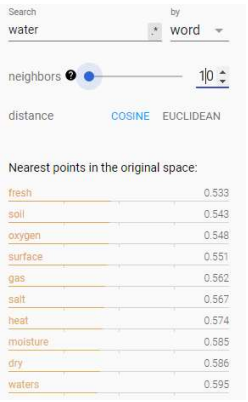


Word Embedding

Word Embedding

- Words or Phrases from the vocabulary are mapped to vectors of real numbers.
- Conceptually it involves a mathematical embedding from a space with many dimensions per word to a continuous vector space with a much lower dimension.

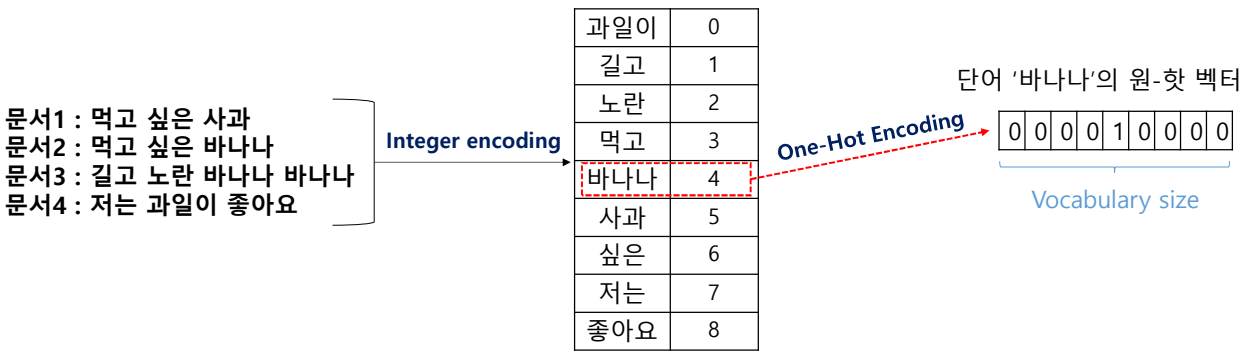




https://projector.tensorflow.org/

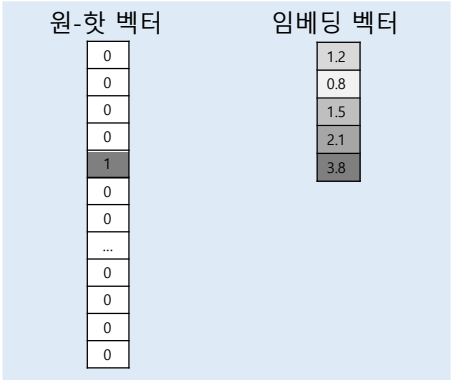
Vectorization : One-Hot Encoding

- **원-핫 인코딩**은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 집합)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.



Vectorization : OHE vs. Word Embedding

- **원-핫 인코딩**으로 표현된 단어들은 단어의 의미를 반영한 유사도를 구할 수 없다.
- **워드 임베딩**을 사용하면 '선생님' 벡터가 '전화기'라는 벡터보다 '교사'라는 벡터와 가깝다는 것을 알 수 있다.



-	원-핫 벡터	임베딩 벡터
차원	고차원(단어 집합의 크기)	저차원
다른 표현	희소 벡터의 일종	밀집 벡터의 일종
표현 방법	수동	훈련 데이터로부터 학습함
값의 타입	1과 0	실수

Word Embedding

- **워드 임베딩은 크게 두 가지로 구분된다.**

1. 랜덤 초기화 임베딩

- NNLM과 마찬가지로 랜덤값을 가지가 오차를 구하는 과정에서 embedding table을 학습.
- NNLM은 이전 단어가 주어졌을 때, 다음 단어를 구하는 학습 과정에서 오차를 줄이면서 학습되었으나 텍스트 분류, 개체명 인식 등 수많은 태스크에서도 오차를 줄이며 학습 가능.
- Task에 맞도록 embedding vector값이 최적화 된다.

2. 사전 훈련된 임베딩(Pre-trained Word Embedding)

- 정해진 특정 알고리즘에 방대한 데이터를 입력으로 학습시킨 후 여러 Task의 입력으로 사용
- 대표적인 알고리즘으로 Word2Vec, FastText, GloVe가 존재한다.

Embedding layer (랜덤 초기화 방법)

케라스, 파이토치 등의 딥 러닝 프레임워크에서는 임베딩 층(embedding layer)을 제공.
초기에 모든 단어의 임베딩 벡터값은 랜덤 초기화 되며, 모델이 역전파하는 과정에서 학습된다.

Text Classification Implemententation

```
model = Sequential()
# 임베딩 벡터의 차원은 256
model.add(Embedding(vocab_size, 256))
model.add(LSTM(256))
model.add(Dense(1, activation='sigmoid'))
```

Projection layer? Embedding layer.

원-핫 벡터들은 단어 집합의 크기인 V를 행, 하이퍼파라미터 M을 열의 크기를 가지는 행렬과 곱한다.
이 행렬 W는 가중치 행렬로 학습 과정에서 학습된다.
~~NNLM에서는 투사층(projection layer)이라고 한다.~~ 이를 임베딩 층(Embedding layer)이라 한다.

x_{fat} × $W_{V\times M}$ = e_{fat}

0	0	0	1	0	0	0

×

0.5	2.1	1.9	1.5	0.8
0.8	1.2	2.8	1.8	2.1
0.1	0.8	1.2	0.9	0.7
2.1	1.8	1.5	1.7	2.7

=

2.1	1.8	1.5	1.7	2.7

lookup table

Projection layer? Embedding layer.

실제 신경망을 구현할 때는 입력을 원-핫 벡터가 아니라 정수로 사용한다는 점에서 차이가 있다.
사실 굳이 원-핫 벡터를 입력으로 사용할 필요가 없이 정수 인코딩 된 상태의 정수를
입력으로 사용하여 룩업 테이블을 하더라도 lookup table 자체는 동일하기 때문이다.

$W_{V\times M}$ = e_{fat}

x_{fat}
3번 단어

0	0.5	2.1	1.9	1.5	0.8
1	0.8	1.2	2.8	1.8	2.1
2	0.1	0.8	1.2	0.9	0.7
3	2.1	1.8	1.5	1.7	2.7
...					

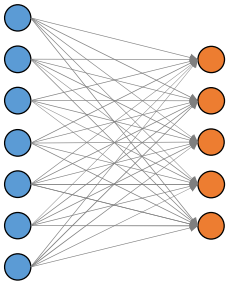
=

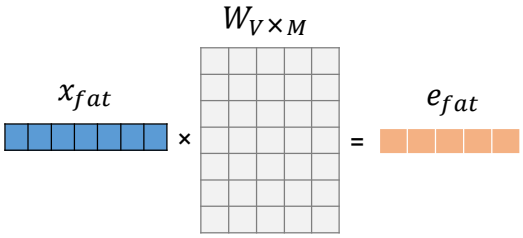
2.1	1.8	1.5	1.7	2.7

lookup table

Projection layer? Embedding layer.

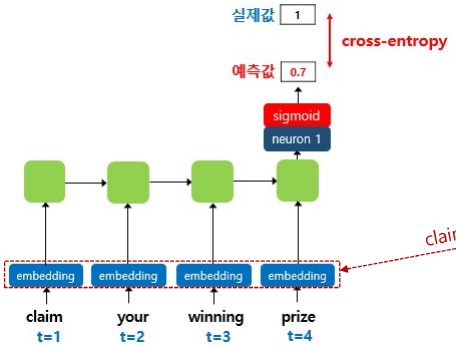
- Embedding Layer는 신경망 그림으로 해석한다면 다음과 같다.
- 단, Weight Matrix는 존재하지만 Bias는 사용하지 않는다.
- 활성화 함수도 사용하지 않는다.
- 다시 말해 학습 가능한 가중치 행렬만 존재한다.

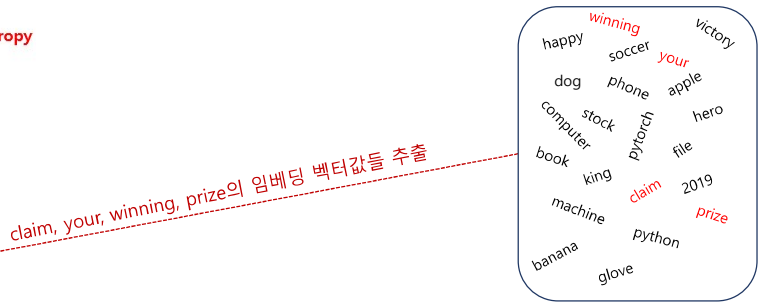




Pre-trained word embedding

- 방대한 양의 텍스트 데이터로 이미 훈련되어져 있는 임베딩 벡터값들을 갖고와서 모델의 입력으로 사용하는 것이 모델의 성능을 높이는 시도.
- 이때 이 임베딩 벡터들은 Word2Vec, FastText, GloVe 등의 알고리즘으로 훈련된 벡터들.



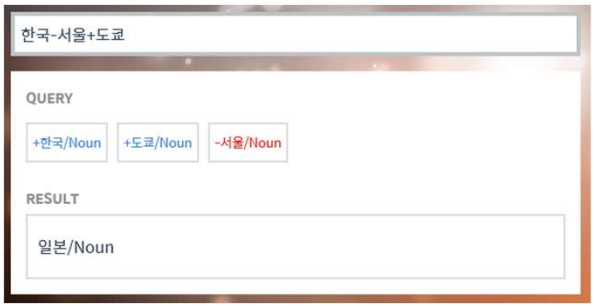


방대한 데이터로 이미 값이 훈련되어져 있는 임베딩 벡터들

Word2Vec

Word2Vec

단어의 의미를 반영한 임베딩 벡터를 만드는 대표적인 방법.
벡터가 된 단어들은 이제 '벡터'이므로 서로 연산 또한 가능하다.

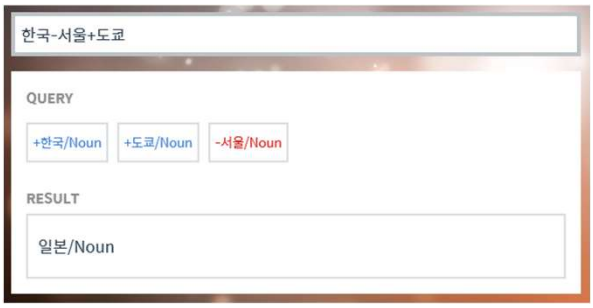


접속해보세요.

http://w.elnn.kr/search/

Word2Vec

단어의 의미를 반영한 임베딩 벡터를 만드는 대표적인 방법.
벡터가 된 단어들은 이제 '벡터'이므로 서로 연산 또한 가능하다.

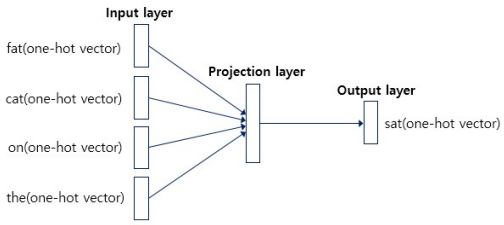


직접 입력해서 결과를 확인해봅시다.

한국 - 서울 + 도쿄 = 일본
박찬호 - 야구 + 축구 = 호나우두

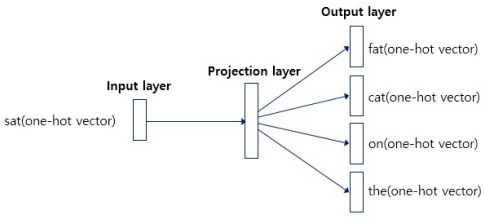
Word2Vec

Word2Vec은 기본적으로 NNLM을 개선한 모델.
이전 단어들로부터 다음 단어를 예측하는 목표는 버리고, 임베딩 그 자체에만 집중.



CBoW

주변 단어들로부터 중심 단어를 예측



Skip-gram

중심 단어들로부터 주변 단어를 예측

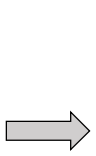
CBoW(Continuous Bag of Words)

주변 단어들로부터 중심 단어를 예측하는 과정에서 임베딩 벡터를 학습.
윈도우 크기를 정해준다면, 주어진 텍스트로부터 훈련 데이터를 자체 구축.

The fat cat sat on the mat

이라는 문장이 있다고 해보자.

윈도우 크기가 2라고 했을 때,
CBoW가 가지는 훈련 데이터셋



window size=2. 좌, 우 단어 2개씩을 참고.

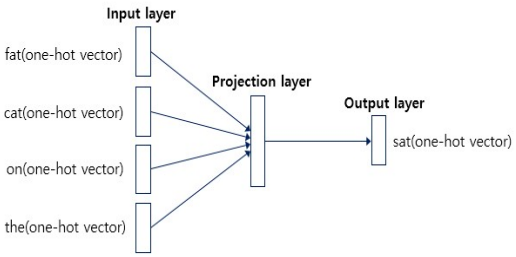
중심 단어 / 주변 단어

The fat cat sat on the mat
The fat cat sat on the mat
The fat cat sat on the mat
The fat cat sat on the mat
The fat cat sat on the mat
The fat cat sat on the mat
The fat cat sat on the mat

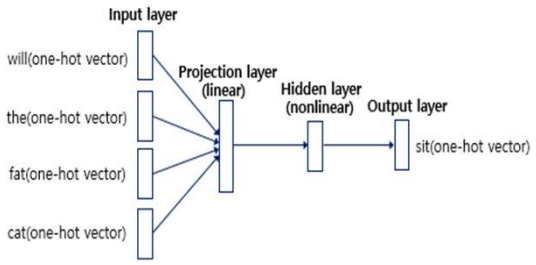
중심 단어	주변 단어
[1, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0,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

CBoW(Continuous Bag of Words)

Word2Vec은 총 3개의 layer로 구성된다. (Deep하지 않은 신경망)
Projection layer가 하는 역할은 Lookup Table.
NNLM과 몇 가지 차이점이 존재하는데 그 중 하는 Hidden layer가 제거되었다는 점이다.



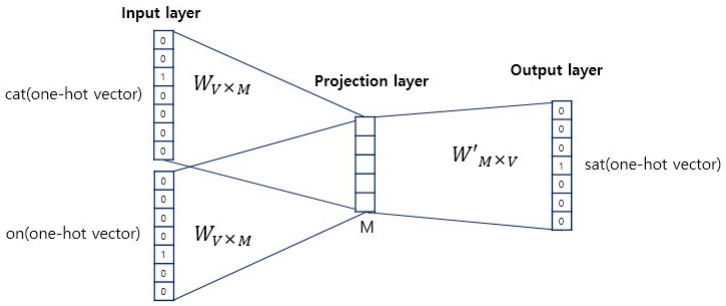
CBoW



NNLM

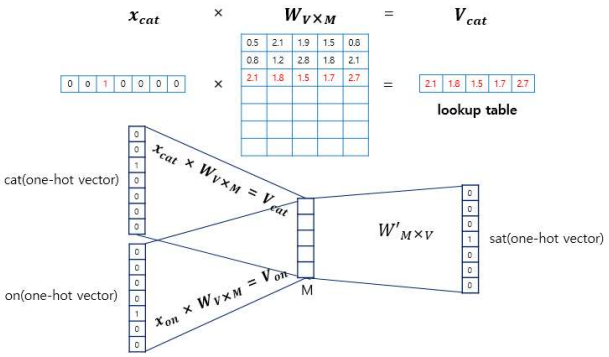
CBoW(Continuous Bag of Words)

Word2Vec은 총 2개의 가중치 행렬을 가진다.
W는 단어 집합의 크기인 V행, 임베딩 행렬의 차원인 M열. (M은 하이퍼파라미터)
W'는 그 반대의 크기. (실제로 W의 전치 행렬은 아님.)



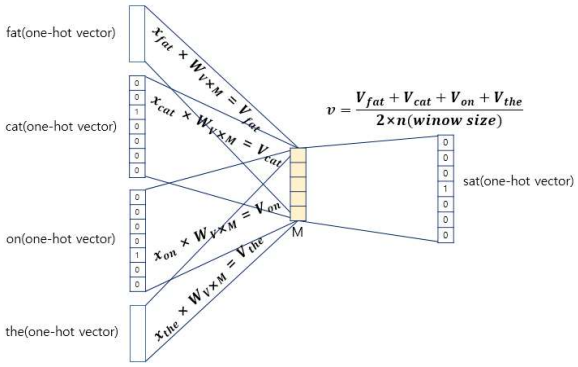
CBoW : Projection layer(lookup table)

Projection layer에서는 입력된 원-핫 벡터와 가중치 행렬 W의 곱.
결과적으로 우리가 목표로 하는 Embedding Vector의 값.



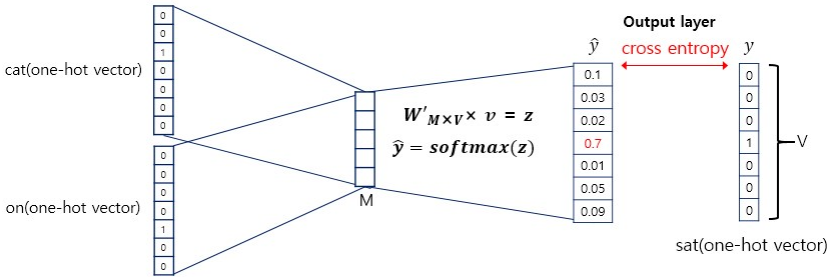
CBoW : Projection layer(averaging vectors)

Projection layer에서의 최종 연산
Projection layer에서 모든 embedding vector들은 평균값을 구하여 M차원의 벡터를 얻음.



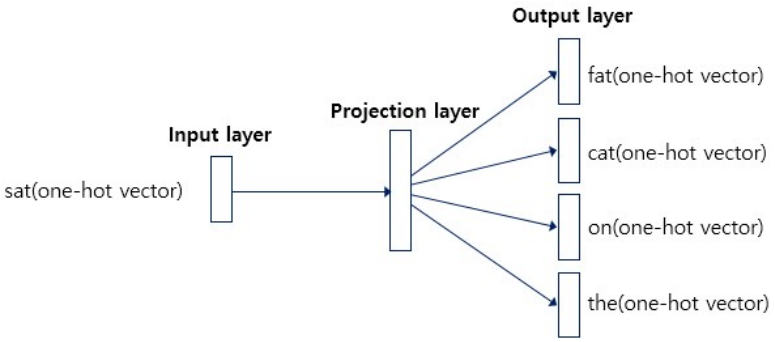
CBoW : Output layer

Projection layer의 M차원의 벡터는 가중치 행렬 W'와 곱하여 소프트맥스 함수를 통과.
이 결과값은 CBoW의 예측값으로 실제 중심 단어의 원-핫 벡터와 loss를 구하고 역전파



Skip-gram

CBoW와는 달리 중심 단어로부터 주변 단어를 예측.



Skip-gram

윈도우 크기가 2일 때, Skip-gram은 다음과 같이 데이터셋을 구성한다.

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...

input word	target word

Skip-gram

윈도우 크기가 2일 때, Skip-gram은 다음과 같이 데이터셋을 구성한다.

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a

Skip-gram

윈도우 크기가 2일 때, Skip-gram은 다음과 같이 데이터셋을 구성한다.

Thou shalt not make a machine in the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	…

thou	shalt	not	make	a	machine	in	the	…

input word	target word
not	thou
not	shalt
not	make
not	a

Skip-gram

윈도우 크기가 2일 때, Skip-gram은 다음과 같이 데이터셋을 구성한다.

Thou shalt not make a machine n the likeness of a human mind

thou	shalt	not	make	a	machine	in	the	...

thou	shalt	not	make	a	machine	in	the	...

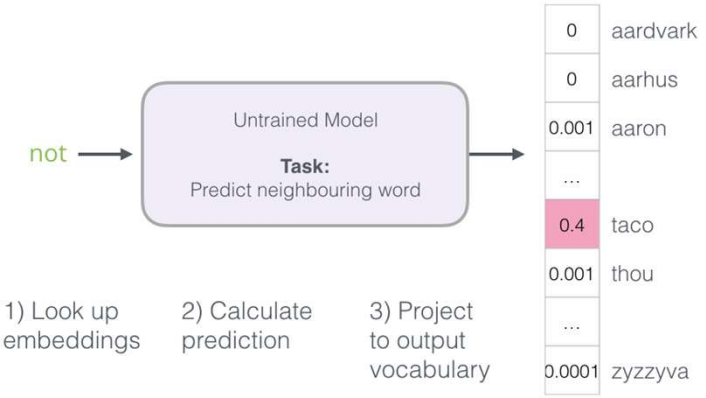
input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

Skip-gram

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine
a	not
a	make
a	machine
a	in
machine	make
machine	a
machine	in
machine	the
in	a
in	machine
in	the
in	likeness

- **Skip-gram은 중심 단어로부터 주변 단어를 예측한다.**

Skip-gram

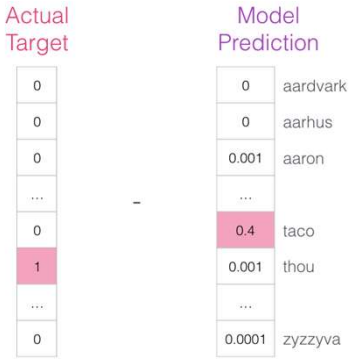


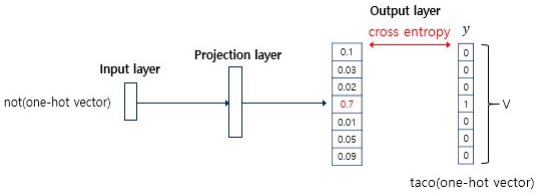
- **Skip-gram은 중심 단어로부터 주변 단어를 예측한다.**

Skip-gram

- **입력층, 투사층, 출력층 3개의 층을 구성된 신경망.**
- **소프트맥스 함수를 지난 예측값(Prediction)과 실제값으로부터 오차(error)를 구한다.**

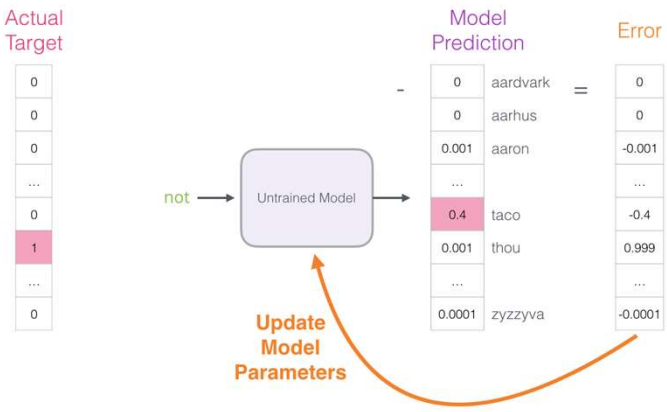
Skip-gram





- 입력층, 투사층, 출력층 3개의 층을 구성된 신경망.
- 소프트맥스 함수를 지난 예측값(Prediction)과 실제값으로부터 오차(error)를 구한다.

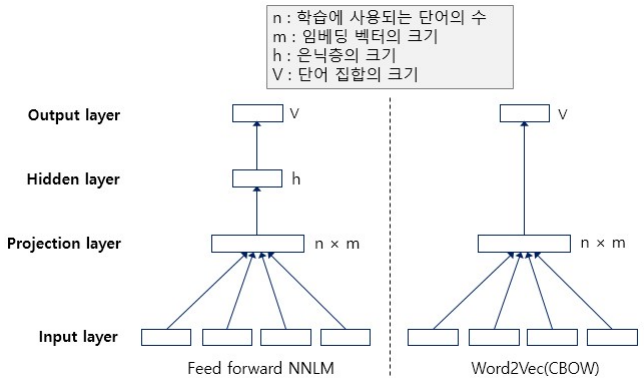
Skip-gram



- 소프트맥스 함수를 지난 예측값(Prediction)과 실제값으로부터 오차(error)를 구하고, 이로부터 embedding table을 update한다.

NNLM vs. CBoW

NNLM Vs. CBoW

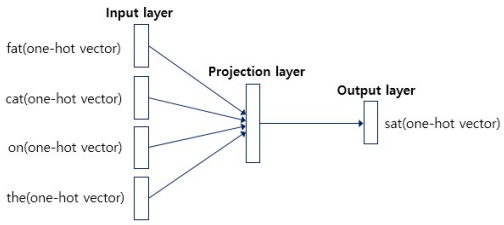


이전 단어들로부터 다음 단어를 예측

주변 단어들로부터 중심 단어를 예측

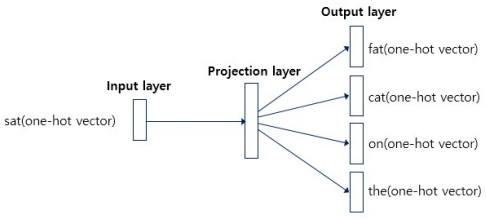
Negative Sampling

실제로는 CBoW나 Skip-gram을 방금 설명한 바와 같이 구현하지는 않음.
그 이유는 아래와 같이 구현한다면 속도가 너무 느리다.



CBoW

주변 단어들로부터 중심 단어를 예측

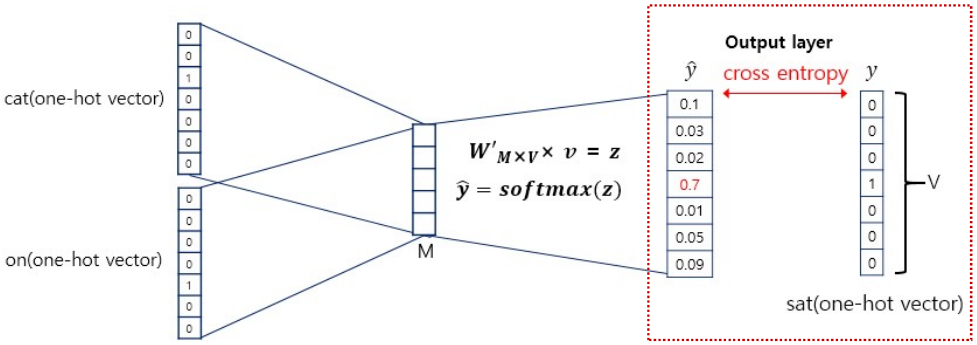


Skip-gram

중심 단어들로부터 주변 단어를 예측

Negative Sampling

실제로는 CBoW나 Skip-gram을 방금 설명한 바와 같이 구현하지는 않음.
그 이유는 아래와 같이 구현한다면 속도가 너무 느리다.
단어 집합의 크기에 대해서 softmax + cross entropy 연산은 너무 Heavy하다.



Skip-Gram with Negative Sampling(SGNS)

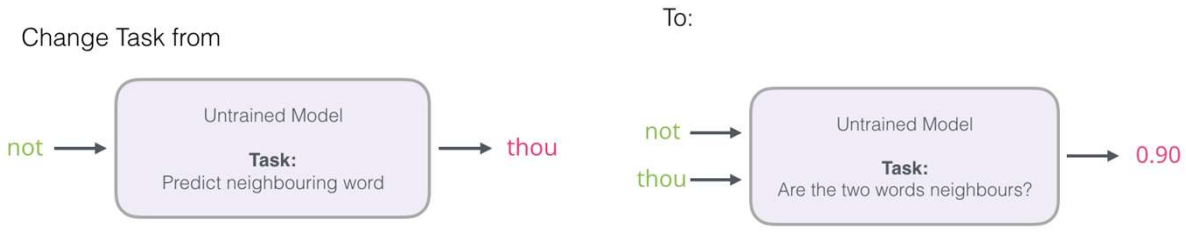
다중 클래스 분류를 이진 분류 문제로 바꾸므로서 연산량을 획기적으로 줄인다.

Change Task from

Skip-gram은 주변 단어로부터 중심 단어를 예측하는 모델이었다.

Skip-Gram with Negative Sampling(SGNS)

다중 클래스 분류를 이진 분류 문제로 바꾸므로서 연산량을 획기적으로 줄인다.



Skip-gram은 주변 단어로부터 중심 단어를 예측하는 모델이었다.
이제 중심 단어와 주변 단어의 내적으로부터 어떤 값을 예측하는 모델로 변경한다.

Skip-Gram with Negative Sampling(SGNS)

우선, 기존의 Skip-gram의 데이터셋부터 변경할 필요가 있다.
중심 단어를 입력, 주변 단어를 레이블로 하는 데이터셋의 형식을 변경해준다.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

Skip-Gram with Negative Sampling(SGNS)

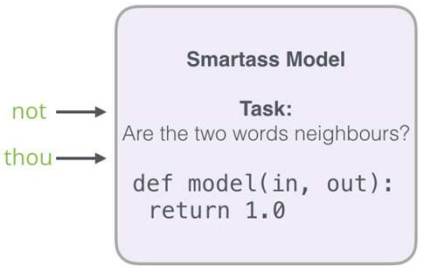
우선, 기존의 Skip-gram의 데이터셋부터 변경할 필요가 있다.
이제 이진 분류 문제를 수행할 것이므로 0 또는 1이라는 레이블이 필요하다.
주변 단어와 중심 단어 데이터셋에 True를 의미하는 레이블 1을 할당해준다.

input word	target word
not	thou
not	shalt
not	make
not	a
make	shalt
make	not
make	a
make	machine

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Skip-Gram with Negative Sampling(SGNS)

이 모델의 목적은 중심 단어와 주변 단어를 입력으로 하였을 때,
실제로 이 두 단어의 관계가 이웃(neighbors)의 관계인지를 예측하는 것이다.



Skip-Gram with Negative Sampling(SGNS)

주변 단어와 중심 단어 데이터셋에 True를 의미하는 레이블 1을 할당해주었다.
실제로 이웃 관계이므로 이들의 레이블은 1을 할당해주는 것이 맞다.

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

Skip-Gram with Negative Sampling(SGNS)

주변 단어와 중심 단어 데이터셋에 True를 의미하는 레이블 1을 할당해주었다.
이제 거짓을 의미하는 샘플들도 추가해주어야 한다. 이를 **Negative Sample**이라고 한다.
Negative Sample은 전체 데이터셋에서 랜덤으로 추가해준다.

input word	output word	target
not	thou	1
not	shalt	1
not	make	1
not	a	1
make	shalt	1
make	not	1
make	a	1
make	machine	1

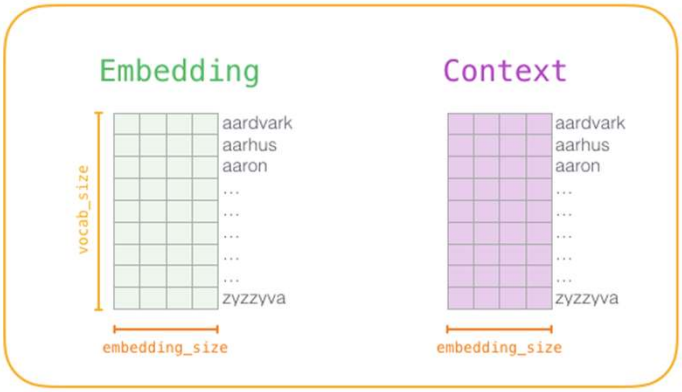
input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	make	1

Pick randomly from vocabulary
(random sampling)

Word	Count	Probability
aardvark		
aarhus		
aaron		
taco		
thou		
zyzzyva		

Skip-Gram with Negative Sampling(SGNS)

이제 모델 구조도 이전과 달라진다.
다음과 같이 두 개의 Embedding table을 준비한다.



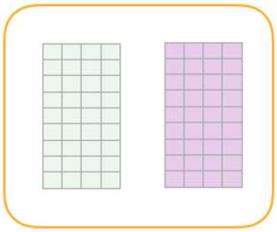
Skip-Gram with Negative Sampling(SGNS)

이제 모델 구조도 이전과 달라진다.
다음과 같이 두 개의 Embedding table을 준비한다.
한 테이블은 중심 단어, 한 테이블은 주변 단어를 위한 테이블이다.

dataset

input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
…	…	…

model

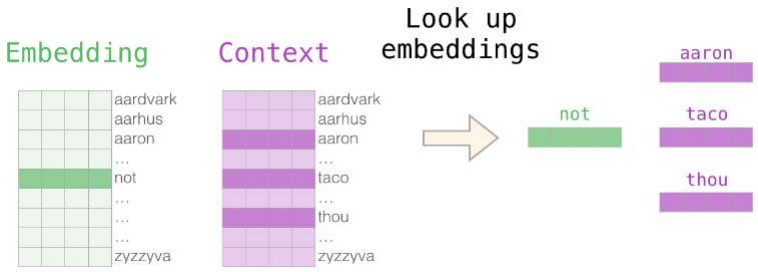


Skip-Gram with Negative Sampling(SGNS)

이제 모델 구조도 이전과 달라진다.
다음과 같이 두 개의 Embedding table을 준비한다.
한 테이블은 중심 단어, 한 테이블은 주변 단어를 위한 테이블이다.

dataset

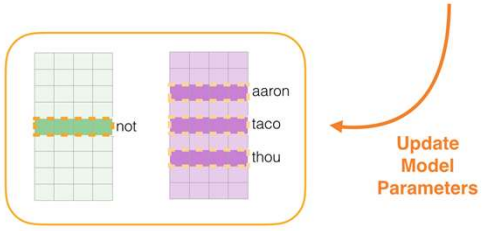
input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...



Skip-Gram with Negative Sampling(SGNS)

중심 단어와 주변 단어의 내적으로부터 실제값인 1 또는 0을 예측하고,
실제값과의 오차(error)를 계산하여, 역전파를 통해 두 개의 테이블을 업데이트한다.

input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



Conclusion

- Embedding vector의 차원을 정하는 것은 사용자의 몫이다.
- CBoW보다는 SGNS(Skipgram with Negative Sampling)이 가장 많이 선호된다.
- 작은 윈도우 크기 (2 ~ 7)을 가질수록, 상호 교환할 수 있을 정도의 높은 유사도를 가진다.
  여기서 상호 교환이 가능하다는 것은 어쩌면 반의어도 포함될 수 있다.
  Ex) 친절한 ↔ 불친절한
- 반면, 커다란 윈도우 크기 (7 ~ 25)는 관련 있는 단어들을 군집하는 효과를 가진다.

Window size : 2

Window size : 7

Conclusion

- 데이터셋을 위한 Negative Sampling의 비율 또한 성능에 영향을 주는 또 다른 결정요소이다.
- 논문에서는 5-20을 최적의 숫자로 정의하고 있다.
- 데이터가 방대하다면 2-5로 충분하다.

Negative samples: 2

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

Skipgram with Negative Sampling

- 실습 코드 : https://wikidocs.net/69141

```
# 데이터 구성

(commited (7837), badar (34572)) -> 0
(whole (217), realize (1036)) -> 1
(reason (149), commited (7837)) -> 1
(letter (705), rediculous (15227)) -> 1
(reputation (5533), midonrnax (47527)) -> 0
```

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, Reshape, Activation, Input
from tensorflow.keras.layers import Dot
from tensorflow.keras.utils import plot_model

embed_size = 100

# 중심 단어를 위한 임베딩 테이블
w_inputs = Input(shape=(1, ), dtype='int32')
word_embedding = Embedding(vocab_size, embed_size)(w_inputs)

# 주변 단어를 위한 임베딩 테이블
c_inputs = Input(shape=(1, ), dtype='int32')
context_embedding  = Embedding(vocab_size, embed_size)(c_inputs)

dot_product = Dot(axes=2)([word_embedding, context_embedding])
dot_product = Reshape((1,), input_shape=(1, 1))(dot_product)
output = Activation('sigmoid')(dot_product)

model = Model(inputs=[w_inputs, c_inputs], outputs=output)
model.summary()
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
w2v.most_similar(positive=['police'])
```

```
[('prohibit', 0.6182408332824707),
 ('provisions', 0.5706381797790527),
 ('cops', 0.565453290939331),
 ('army', 0.563193142414093),
 ('possess', 0.5538119673728943),
 ('armed', 0.5535427331924438),
 ('rkba', 0.5533647537231445),
 ('ksanti', 0.5518242716789246),
 ('courts', 0.5495947599411011),
 ('officers', 0.547795057296752 9)]
```

Skipgram with Negative Sampling

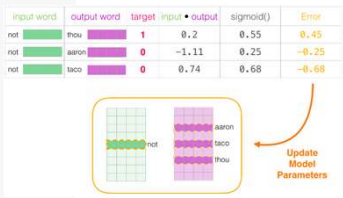
- 실습 코드 : https://wikidocs.net/69141

```
# 데이터 구성

(commited (7837), badar (34572)) -> 0
(whole (217), realize (1036)) -> 1
(reason (149), commited (7837)) -> 1
(letter (705), rediculous (15227)) -> 1
(reputation (5533), midonrnax (47527)) -> 0
```

```
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, Reshape, Activation, Input
from tensorflow.keras.layers import Dot
from tensorflow.keras.utils import plot_model

embed_size = 100

# 중심 단어를 위한 임베딩 테이블
w_inputs = Input(shape=(1, ), dtype='int32')
word_embedding = Embedding(vocab_size, embed_size)(w_inputs)

# 주변 단어를 위한 임베딩 테이블
c_inputs = Input(shape=(1, ), dtype='int32')
context_embedding  = Embedding(vocab_size, embed_size)(c_inputs)

dot_product = Dot(axes=2)([word_embedding, context_embedding])
dot_product = Reshape((1,), input_shape=(1, 1))(dot_product)
output = Activation('sigmoid')(dot_product)

model = Model(inputs=[w_inputs, c_inputs], outputs=output)
model.summary()
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
w2v.most_similar(positive=['police'])
```

```
[('prohibit', 0.6182408332824707),
 ('provisions', 0.5706381797790527),
 ('cops', 0.565453290939331),
 ('army', 0.563193142414093),
 ('possess', 0.5538119673728943),
 ('armed', 0.5535427331924438),
 ('rkba', 0.5533647537231445),
 ('ksanti', 0.5518242716789246),
 ('courts', 0.5495947599411011),
 ('officers', 0.547795057296752 9)]
```



FastText

FastText

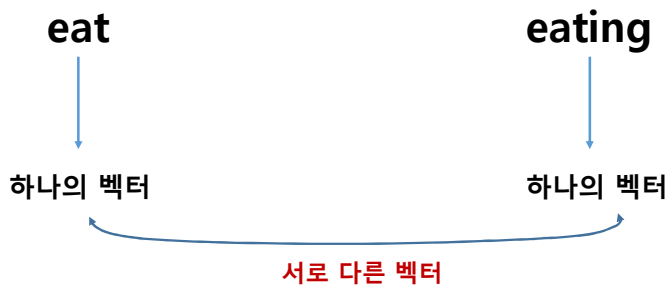
Word2Vec의 개량 알고리즘으로 Subword를 고려한 알고리즘.
Word2Vec은 하나의 단어에 고유한 벡터를 할당하므로 단어의 형태학적 특징을 반영할 수 없다.

eat → 하나의 벡터

eating → 하나의 벡터

FastText

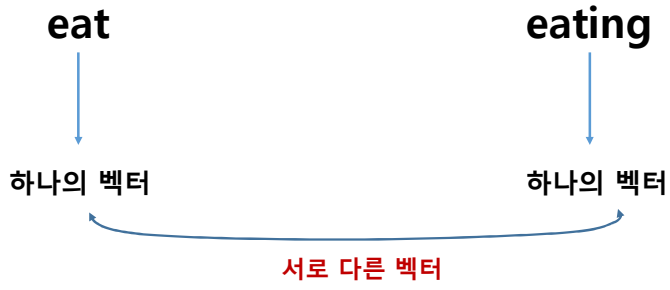
Word2Vec의 개량 알고리즘으로 Subword를 고려한 알고리즘.
Word2Vec은 하나의 단어에 고유한 벡터를 할당하므로 단어의 형태학적 특징을 반영할 수 없다.



FastText

Word2Vec의 개량 알고리즘으로 Subword를 고려한 알고리즘.
Word2Vec은 하나의 단어에 고유한 벡터를 할당하므로 단어의 형태학적 특징을 반영할 수 없다.

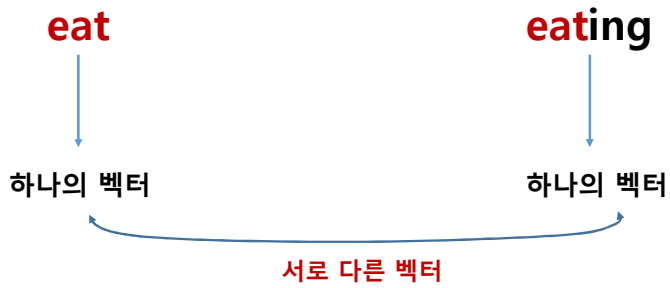
훈련 데이터에서 eat는 충분히 많이 등장해서, 학습이 충분히 잘 되었지만
eating은 잘 등장하지 않아서 제대로 된 임베딩 값을 얻지 못한다고 해보자.



FastText

Word2Vec의 개량 알고리즘으로 Subword를 고려한 알고리즘.
Word2Vec은 하나의 단어에 고유한 벡터를 할당하므로 단어의 형태학적 특징을 반영할 수 없다.

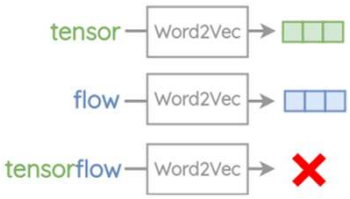
공통적인 내부 단어를 갖고 있는데, 이를 활용할 수는 없을까?



FastText

Word2Vec의 대표적인 문제점 두 가지.

1. OOV(Out-of-Vocabulary) 문제



Word2Vec의 Vocabulary에
"tensor"와 "flow"가 있더라도, "tensorflow"
라는 단어가
Vocabulary에 없다면,
"tensorflow"의 벡터값을 얻을 수 없다.

FastText

Word2Vec의 대표적인 문제점 두 가지.

2. 형태학적 특징을 반영할 수 없다.

Shared radical

eat eats eaten eater eating

다음의 단어들은 eat라는 동일한 어근을 가진다.
하지만 Word2Vec에서는
각 단어는 각 벡터의 값을 가질뿐이다.
이들을 고려할 수는 없을까?

FastText

FastText는 단어를 Character 단위의 n-gram으로 간주한다.
n을 몇으로 하느냐에 따라서 단어가 얼마나 분리되는지가 결정된다.
단어 'eating'을 예를 들어보자.

1. 단어 eating에 시작과 끝을 의미하는 <와 >를 추가한다.

eating → <eating>

FastText

FastText는 단어를 Character 단위의 n-gram으로 간주한다.
n을 몇으로 하느냐에 따라서 단어가 얼마나 분리되는지가 결정된다.
단어 'eating'을 예를 들어보자.

2. n-gram을 기반으로 단어를 분리한다. (Ex) n=3)

FastText

FastText는 단어를 Character 단위의 n-gram으로 간주한다.
n을 몇으로 하느냐에 따라서 단어가 얼마나 분리되는지가 결정된다.
단어 'eating'을 예를 들어보자.

3. 주로 n은 범위로 설정해준다. Ex) n : 3 ~ 6

Word	Length(n)	Character n-grams
eating	3	<ea, eat, ati, tin, ing, ng>
eating	4	<eat, eati, atin, ting, ing>
eating	5	<eati, eatin, ating, ting>
eating	6	<eatin, eating, ating>

FastText

훈련 데이터를 N-gram의 셋으로 구성하였다면, 훈련 방법 자체는 SGNS와 동일하다.
단, Word가 아니라 subwords들이 최종 학습 목표이며
이들의 합을 Word의 vector로 간주한다.

FastText Pre-training

FastText의 훈련 과정을 이해해보자.
여기서는 SGNS(Skip-gram with Negative Sampleing)을 사용한다.

우리의 목표는 중심 단어 eating으로부터 주변 단어 am과 food를 예측하는 것이다.

I am eating food now

FastText Pre-training

앞서 언급하였듯이 단어 eating은
아래와 같이 n-gram들의 합으로 나타낸다.

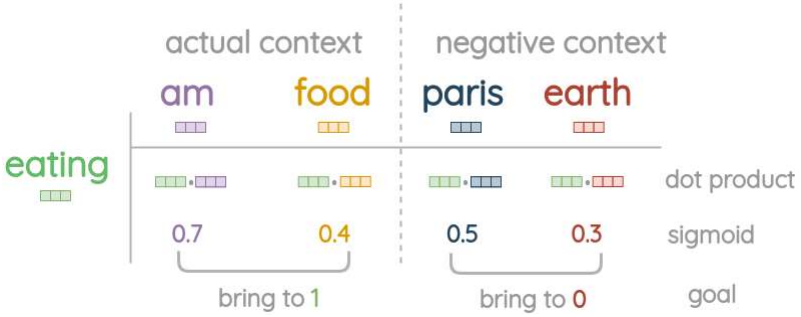
FastText Pre-training

우리가 해야하는 것은 eating으로부터 am과 food를 예측하는 것이다.

Negative Sampling이므로 실제 주변 단어가 아닌 단어들도 필요하다.

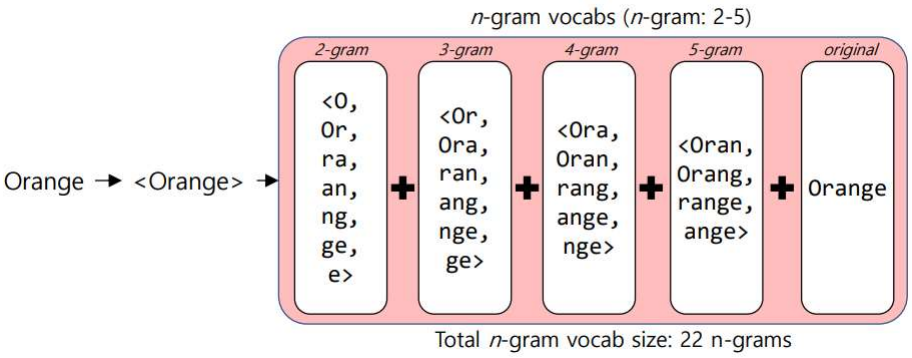
FastText Pre-training

우리가 해야하는 것은 eating으로부터 am과 food를 예측하는 것이다.
eating과 am 그리고 eating과 food의 내적값에 시그모이드 함수를 지난 값은 1이 되도록 학습.
eating과 paris 그리고 eating과 earth의 내적값에 시그모이드 함수를 지난 값은 0이 되도록 학습.



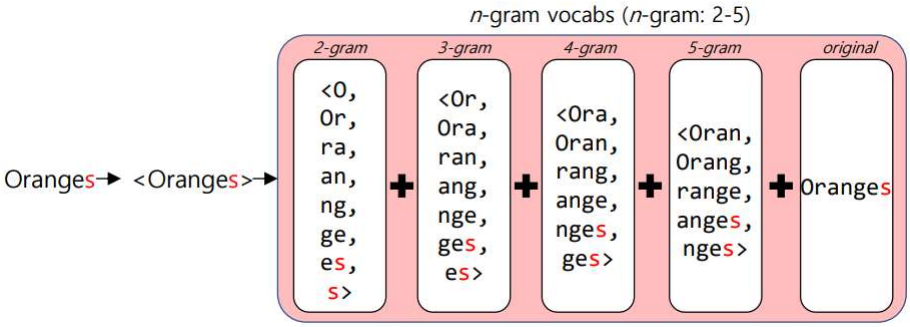
FastText Example

예를 들어 단어 Orange에 대해서 FastText를 학습했다고 해보자. n의 범위는 2-5로 한다.



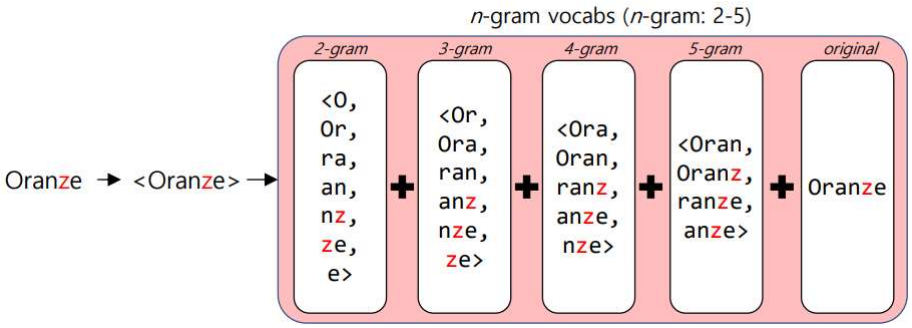
FastText Example

예를 들어 단어 Orange에 대해서 FastText를 학습했다고 해보자. n의 범위는 2-5로 한다.
그 후 Oranges라는 OOV 또는 희귀 단어가 등장했다고 해보자.
Orange의 n-gram 벡터들을 이용하여 Oranges의 벡터값을 얻는다.



FastText Example

예를 들어 단어 Orange에 대해서 FastText를 학습했다고 해보자. n의 범위는 2-5로 한다.
FastText는 오타에도 강건하다.



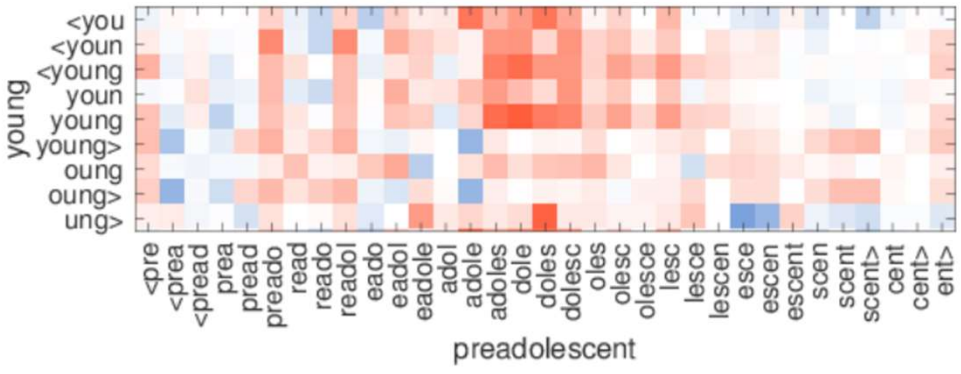
Q & A

Q. 단어에 <, >를 왜 해주는 것인가요?

A) 단어 양끝에 <, >를 해주지 않으면 실제로 독립적인 단어와 특정 단어의 n-gram인 경우를 구분하기 어렵다. 가령, where의 n-gram 중 하나인 her도 존재하지만 독립적인 단어 her 또한 Vocabulary에 존재할 수 있다.
이 때, 독립적인 단어 her는 <her>가 되므로서 where 내의 her와 구분할 수 있다.

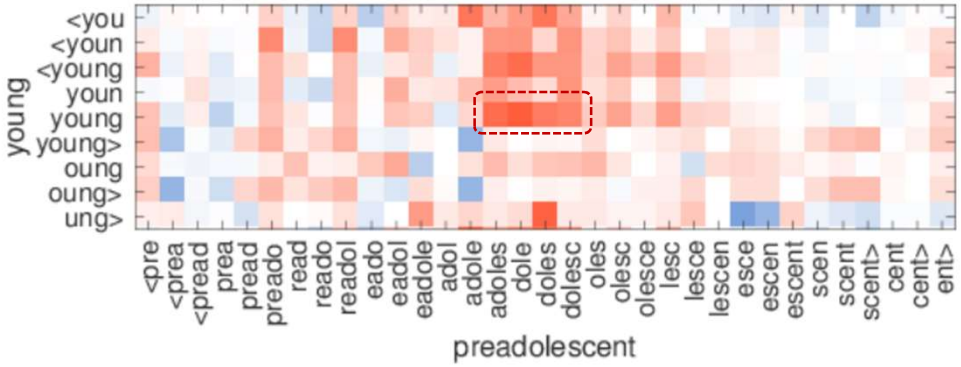
Subword Similarity

충분히 잘 학습된 FastText는 전체 Word가 아니라 Subword들의 유사도를 반영함을 확인할 수 있다.



Subword Similarity

충분히 잘 학습된 FastText는 전체 Word가 아니라 Subword들의 유사도를 반영함을 확인할 수 있다.



한국어에서의 FastText

한국어는 다양한 용언 형태를 가진다.
Word2Vec의 경우 다양한 용언 표현들이 서로 독립된 단어로 표현된다.

동사 원형 : 모르다

모르네, 모르더라, 몰라야, 몰랐다, 모르겠으나, 몰라, 모르겠니, 모르면서 ...

한국어에서의 FastText

한국어는 다양한 용언 형태를 가진다.
Word2Vec의 경우 다양한 용언 표현들이 서로 독립된 단어로 표현된다.

동사 원형 : 모르다

모르네, 모르더라, 몰라야, 몰랐다, 모르겠으나, 몰라, 모르겠니, 모르면서 ...

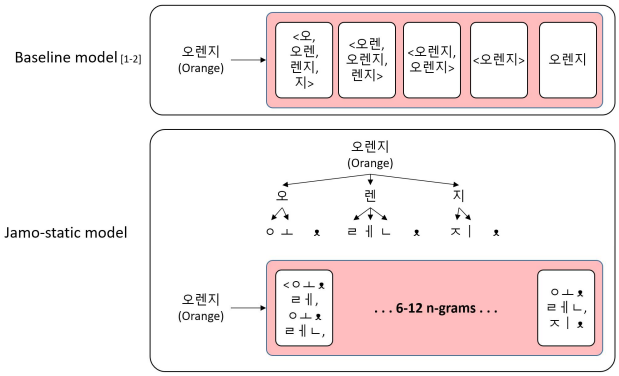
한국어의 경우에는 이를 대응하기 위해 FastText의 n-gram 단위를 자모 단위로 하기도 한다.

Ex) ㅁㅗㄹ

참고 링크 : https://lovit.github.io/nlp/representation/2018/10/22/fasttext_subword/

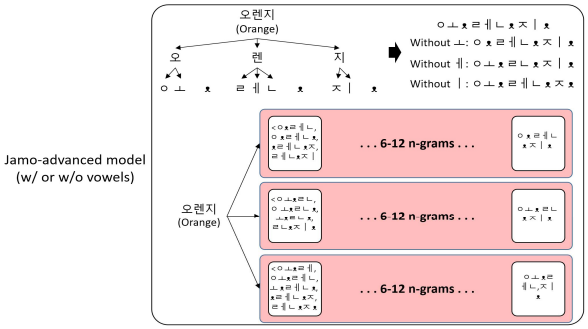
한국어에서의 FastText

한국어의 경우에는 이를 대응하기 위해 FastText의 n-gram 단위를 자모 단위(초성, 중성, 종성)로 하기도 한다.



한국어에서의 FastText

한국어의 경우에는 이를 대응하기 위해 FastText의 n-gram 단위를 자모 단위(초성, 중성, 종성)로 하기도 한다.



링크 : https://colab.research.google.com/drive/1auE0xyFWcYoNH6x2lpsjNWkp9Mwhg96u

Pre-trained Embedding Vs. Embedding layer

사전 훈련된 음절 단위 FastText를 이용해서 네이버 영화 리뷰 분류에 대한 실험 결과

- FastText(자모 단위) : 네이버 영화 리뷰 + 위키피디아 : 86.90%
- FastText(음절 단위) : 네이버 영화 리뷰 + 위키피디아 : 87.07%
- FastText(음절 단위) : 네이버 영화 리뷰 + 위키피디아 + 나무 위키 : 87.15%
- Embedding layer : 네이버 영화 리뷰 : 84.05%

성능

	Accuracy
fastText	87.07%
Keras Embedding	84.05%

생각보다 차이가 나는군요.

성능 차이는 아무래도,

- fastText는 네이버 훈련 데이터 외에 한국어 Wikipedia 데이터도 사용했기 때문에 훨씬 커버리지가 큽니다.
- fastText의 알고리즘 특성상 OOV에 대해서 성능이 더 좋습니다. 네이버 코퍼스가 구어체가 많기 때문에 이 점도 영향을 주었을 것 같고요.

	Accuracy
한국어 위키피디아 + 네이버 영화	87.07%
한국어 위키피디아 + 네이버 영화 + 나무위키	87.15%

링크 : https://jins-sw.tistory.com/6?category=858204

GloVe

Window Based Co-occurrence Matrix

- Document Term Matrix와 마찬가지로 고차원을 가지는 행렬 표현 방법.
- Window 크기에 따라서 행렬의 값이 결정된다는 특징을 가지고 있다.

- I like deep learning.
- I like NLP.
- I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

- 다음의 3개의 예시 문장에 대해서 윈도우 크기가 1일 때를 보여준다.
- 일반적으로는 윈도우 크기는 5-10이다.
- 동시 등장 행렬은 기본적으로 대칭 행렬.

Window Based Co-occurrence Matrix

- Document Term Matrix와 마찬가지로 고차원을 가지는 행렬 표현 방법.
- Window 크기에 따라서 행렬의 값이 결정된다는 특징을 가지고 있다.

- I like deep learning.
- I like NLP.
- I enjoy flying.

동시 등장 행렬 그 자체를 모델의 입력으로 사용하면?

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

- 다음의 3개의 예시 문장에 대해서 윈도우 크기가 1일 때를 보여준다.
- 일반적으로는 윈도우 크기는 5-10이다.
- 동시 등장 행렬은 기본적으로 대칭 행렬.

Window Based Co-occurrence Matrix

- Document Term Matrix와 마찬가지로 고차원을 가지는 행렬 표현 방법.
- Window 크기에 따라서 행렬의 값이 결정된다는 특징을 가지고 있다.

- I like deep learning.
- I like NLP.
- I enjoy flying.

동시 등장 행렬 그 자체를 모델의 입력으로 사용하면?

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

- 행과 열이 단어 집합(Vocabulary size)의 크기.
- DTM, OHE와 마찬가지로 저장 공간 낭비.
- 행렬이 매우 희소하다는 특징을 가진다.

Window Based Co-occurrence Matrix

- Document Term Matrix와 마찬가지로 고차원을 가지는 행렬 표현 방법.
- Window 크기에 따라서 행렬의 값이 결정된다는 특징을 가지고 있다.

- I like deep learning.
- I like NLP.
- I enjoy flying.

저차원. 즉, Dense하도록 임베딩하면?

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

GloVe.

GloVe(Global Vectors for Word Representation)

Word2Vec이 전체 코퍼스의 통계 정보를 충분히 활용하지 않는다는 점을 지적.
윈도우 기반의 동시 등장 행렬(Window Based Co-occurrence Matrix)를 사용한다.
Word2vec과 마찬가지로 Dense한 Vector를 학습.
윈도우 기반 동시 등장 행렬로부터 동시 등장 확률을 정의한다.

	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

GloVe(Global Vectors for Word Representation)

Word2Vec이 전체 코퍼스의 통계 정보를 충분히 활용하지 않는다는 점을 지적.
윈도우 기반의 동시 등장 행렬(Window Based Co-occurrence Matrix)를 사용한다.
Word2vec과 마찬가지로 Dense한 Vector를 학습.
윈도우 기반 동시 등장 행렬로부터 동시 등장 확률을 정의한다.

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

GloVe(Global Vectors for Word Representation)

- GloVe는 중심 단어 벡터와 주변 단어 벡터의 내적(dot product)이 동시 등장 확률이 되도록 학습한다.

$$dot\ product(w_i\ \tilde{w}_k) \approx\ P(k \mid i) = P_{ik}$$

GloVe(Global Vectors for Word Representation)

- GloVe는 중심 단어 벡터와 주변 단어 벡터의 내적(dot product)이 동시 등장 확률이 되도록 학습한다.

$$dot\ product(w_i\ \tilde{w}_k) \approx\ P(k \mid i) = P_{ik}$$

- 더 정확히는 다음을 만족하도록 학습한다.

$$dot\ product(w_i\ \tilde{w}_k) \approx\ log\ P(k \mid i) = log\ P_{ik}$$

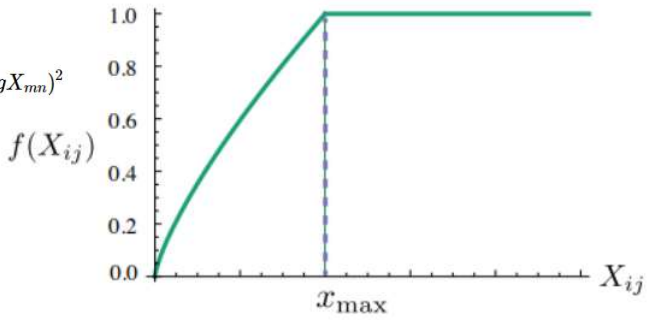
- 두 단어의 내적이 동시 등장 빈도의 로그값의 차이가 최소화 되도록 학습한다.

GloVe(Global Vectors for Word Representation)

- 두 단어의 내적이 동시 등장 빈도의 로그값의 차이가 최소화 되도록 학습한다.

$$Loss\ function = \sum_{m,n=1}^{V} f(X_{mn})(w_m^T \tilde{w}_n + b_m + \tilde{b_n} - logX_{mn})^2$$

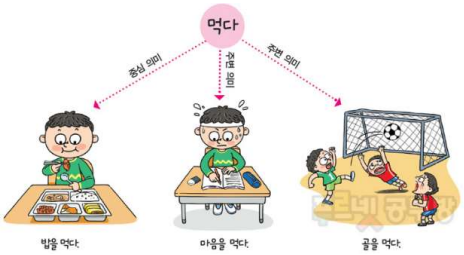
$$f(x) = min(1,\ (x/x_{max})^{3/4})$$

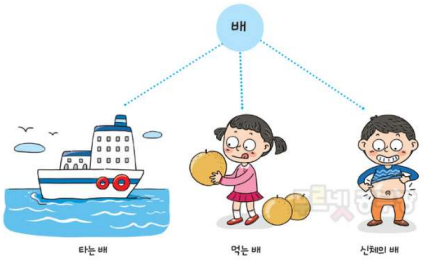


상세 설명 : https://wikidocs.net/22885

Word Embedding의 한계

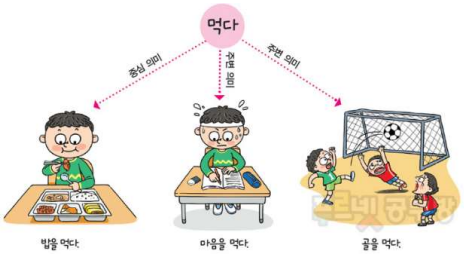
- 동형어, 다의어에 대해서는 제대로 훈련되지 않음.
- 단순히 주변 단어만을 고려하므로 문맥을 고려한 의미를 담고있지는 않음.

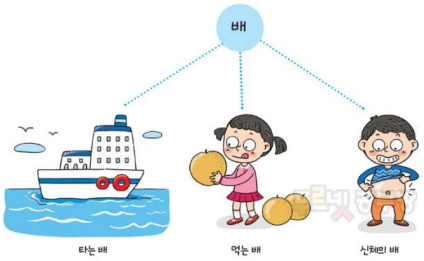




Word Embedding의 한계

- 동형어, 다의어에 대해서는 제대로 훈련되지 않음.
- 단순히 주변 단어만을 고려하므로 문맥을 고려한 의미를 담고있지는 않음.





Contextual Embedding의 등장.
Ex) ELMo, Context2Vec, BERT 등..

도전 과제

- 아직 RNN, LSTM, GRU를 배우지 않았지만 아래의 두 페이지를 참고해서 과제를 진행해볼 것.

- 1. 사전 훈련된 임베딩 사용하기
 : https://wikidocs.net/33793

- 2. IMDB 리뷰 분류하기
 : https://wikidocs.net/24586

2번 링크에 있는 Embedding layer를 1번 링크를 참고하여 Word2Vec 또는 GloVe의 벡터로 바꿔볼 것.
GRU(128)은 아직 배우지 않은 내용이므로 무시하고, 오직 Embedding layer만 바꿔서 훈련한다.

- 힌트 : https://wikidocs.net/86083

과제 하실 때 주의할 점

- Colab에서 GPU를 사용하지 않으면 학습이 느림.
- Colab에서 상단의 런타임 > 런타임 유형 변경 > 하드웨어 가속기 > GPU 선택
  위와 같은 과정을 거치시면 GPU를 사용하여 보다 빠른 딥 러닝 학습이 가능.

- 토의를 위해 Colab에서 상호 링크를 공유하기 위해서는
- 우측 상단에서 공유 > 링크 보기 > 링크가 있는 모든 사용자로 변경 > 링크 복사
  위와 같은 과정을 거치신 후에 복사된 URL을 공유하면 된다.

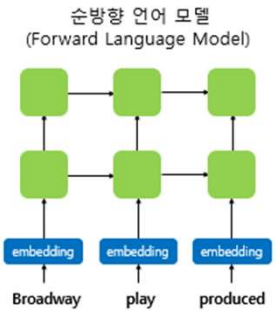
ELMo (예고)

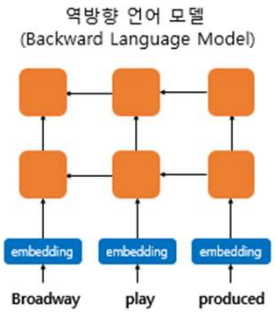
- RNN 이후 학습 예정

엘모(ELMo) – Embeddings from Language Model

- 엘모는 직역하면 '언어 모델로부터 얻는 임베딩'
- 엘모는 BiLM(양방향 언어 모델)을 사용한다.
- 엘모는 순방향과 역방향으로 RNN 언어 모델을 따로 학습시킨다.

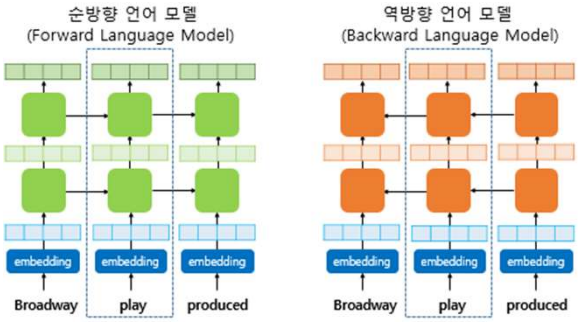
이 케릭터 이름이 엘모.





엘모(ELMo) – Embeddings from Language Model

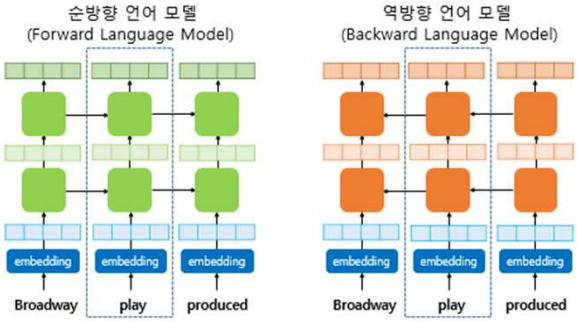
- 엘모는 BiLM(양방향 언어 모델)을 사용한다.
- 엘모는 순방향과 역방향으로 언어 모델을 따로 학습시킨다. 이것이 사전 학습(pre-training) 단계이다.
- 예를 들어 단어 'play'에 대한 임베딩을 얻는다고 가정해보자.



ELMo는 좌측 점선의 사각형 내부의
각 층의 결과값을 재료로 사용한다.

엘모(ELMo) – Embeddings from Language Model

- 예를 들어 단어 'play'에 대한 임베딩을 얻는다고 가정해보자.

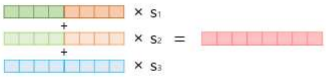


ELMo는 좌측 점선의 사각형 내부의
각 층의 결과값을 재료로 사용한다.

1) 각 층의 출력값을 연결(concatenate)한다.

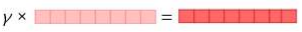
2) 각 층의 출력값 별로 가중치를 준다.

3) 각 층의 출력값을 모두 더한다.



2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

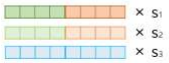
4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.



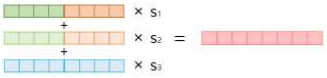
엘모(ELMo) – Embeddings from Language Model

1) 각 층의 출력값을 연결(concatenate)한다.

2) 각 층의 출력값 별로 가중치를 준다.



3) 각 층의 출력값을 모두 더한다.



2)번과 3)번의 단계를 요약하여 가중합(Weighted Sum)을 한다고 할 수 있습니다.

4) 벡터의 크기를 결정하는 스칼라 매개변수를 곱한다.