

# Word2Vec

---

이주형

juhyungee1025@gmail.com

NLP team

2024/03/19



# Contents

---

## 1. 기본 개념

## 2. Word2Vec

- I. 배경
- II. Previous works
- III. CBOW
- IV. Skip-gram
- V. CBOW vs. Skip-gram
- VI. NNLM vs. Word2Vec
- VII. Why Word2Vec?

## 3. Evaluation & Limitation

# 기본 개념

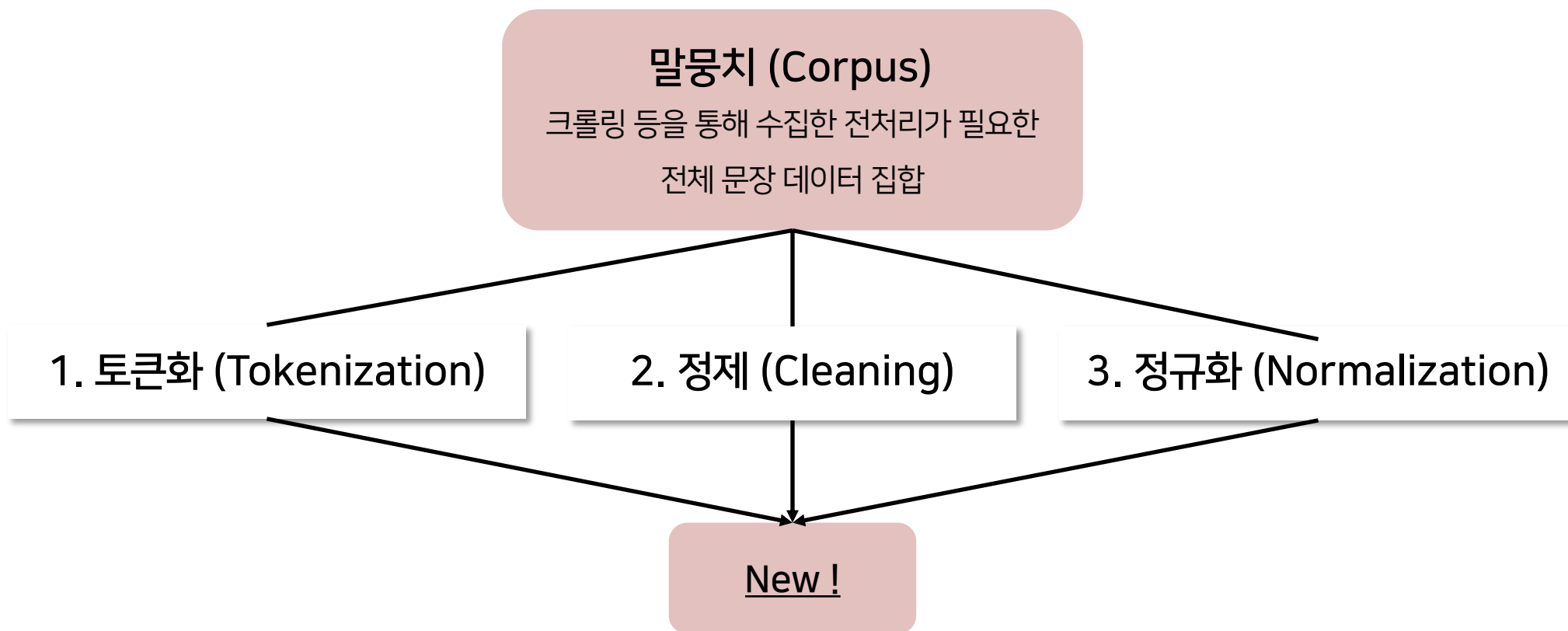
---



# 자연어 전처리

## 자연어 전처리

: 분석이나 모델링을 수행하기 전에 텍스트 데이터를 정제하고 정규화하는 과정



# 자연어 전처리

## 토큰화 (Tokenization)

: 토큰 (의미를 가지는 최소 단위) 단위로 코퍼스를 나누는 작업

Tokenize on  
rules

Let	's	tokenize	!	Is	n't	this	easy	?
-----	----	----------	---	----	-----	------	------	---

Tokenize on  
punctuation

Let	'	s	tokenize	!	Isn	'	t	this	easy	?
-----	---	---	----------	---	-----	---	---	------	------	---

Tokenize on  
white spaces

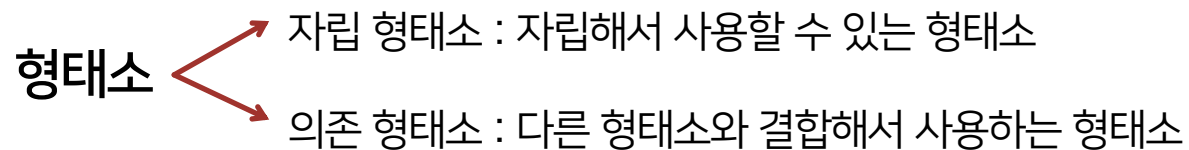
Let's	tokenize!	Isn't	this	easy?
-------	-----------	-------	------	-------

**Let's tokenize! Isn't this easy?**

단어, 구분자, 문장 등 다양한 방법으로 토큰화 가능 → 목적에 따라 적절한 토큰 선택 필요

# 자연어 전처리

## 한국어 토큰화



eg)  
나는 밥을 먹는다.  $\xrightarrow{\text{토큰화}}$  띄어쓰기 단위 : [나는, 밥을, 먹는다]  
형태소 단위 : [나, 는, 밥, 을, 먹-, -는-, -다]

eg) 밥을, 밥과 / 먹었다, 먹는다

한국어는 조사, 어미 등이 붙어서 의미가 달라지는 경우가 많음 → 형태소 단위 구분 필요

# 자연어 전처리

---

## 정제 (Cleaning) & 정규화 (Normalization)

### 정제 (Cleaning)

코퍼스에서 불필요한 데이터를 제거하는 작업

eg) 불용어 제거, 등장빈도가 낮은 단어 제거 등

### 정규화 (Normalization)

표현 방법이 다른 단어를 합쳐 같은 단어로 만들어주는 작업

eg) Potato, potato → 대소문자 통일 (일반적으로 소문자로 통일)

# 임베딩 (Embedding)

## 희소표현

- : 벡터(행렬)의 값이 대부분 0으로 표현
- : 서로 다른 단어들의 집합 (단어 집합) 의 크기 = 벡터의 차원
- : 표현하고 싶은 단어의 인덱스에 1의 값을 부여, 다른 인덱스에는 0을 부여

단어 집합  
[Seoul, Dubai, Paris, Tokyo]

One-Hot Encoding

One-Hot Vector

Seoul = [1, 0, 0, 0]

Dubai = [0, 1, 0, 0]

Paris = [0, 0, 1, 0]

Tokyo = [0, 0, 0, 1]

한계점

1) 단어가 늘어날수록 벡터의 차원이 증가, 2) 단어의 유사도 표현 불가, 3) 공간적 낭비 심함



# 임베딩 (Embedding)

## 밀집표현

: 서로 다른 단어들의 집합 (단어 집합) 의 크기  $\neq$  벡터의 차원

: 사용자가 벡터 크기 설정  $\rightarrow$  데이터 값이 실수로 표현

희소표현

0
1
0
...
0

워드 임베딩



= 임베딩 벡터

밀집표현

0.4
0.7
-0.1
...
0.2

단어를 밀집표현으로 표현하는 방법 = 워드 임베딩 / 워드 임베딩의 결과 벡터 = 임베딩 벡터 = 밀집표현  
각 단어가 벡터로 표현이 됨으로써 연산이 가능하게 됨

# Word2Vec

---



# Word2Vec : 배경

## 분산표현

: 특정 차원의 벡터로 단어를 매핑하기 위한 parametric한 함수

: “비슷한 위치에 등장하는 단어는 비슷한 의미를 가진다” (=분포가정) 기반

: 분포 가정을 이용하여 텍스트를 학습한 후 단어들의 의미를 다차원 공간에 분산하여 표현

$$W : words \rightarrow \mathbb{R}^n$$

$$W(Seoul) = (0.2, -0.4, 0.7, \dots)$$

$$W(Dubai) = (0.0, 0.6, -0.1, \dots)$$

저차원에 단어의 의미를 여러 차원으로 분산 → 단어 벡터간 유의미한 유사도 계산 가능

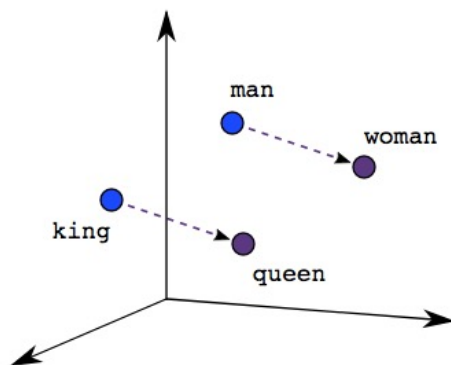
# Word2Vec : 배경

## 분산표현

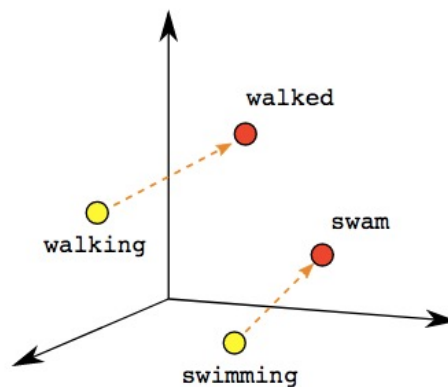
: 특정 차원의 벡터로 단어를 매핑하기 위한 parametric한 함수

: “비슷한 위치에 등장하는 단어는 비슷한 의미를 가진다” (=분포가정) 기반

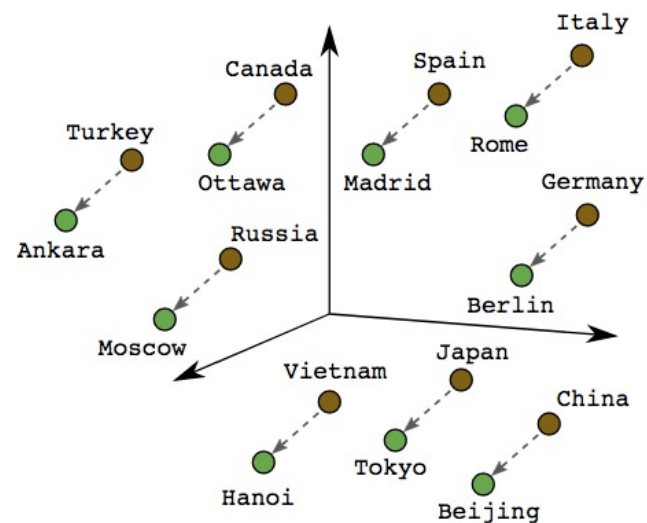
: 분포 가정을 이용하여 텍스트를 학습한 후 단어들의 의미를 다차원 공간에 분산하여 표현



Male-Female



Verb Tense



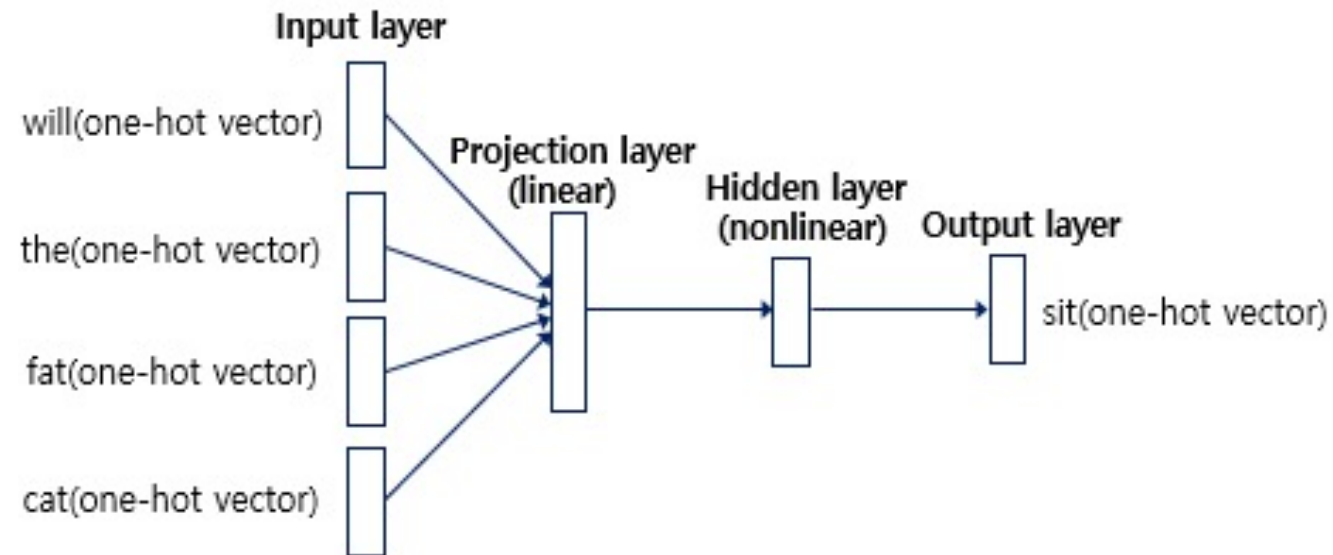
Country-Capital

# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

: 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장

: 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델



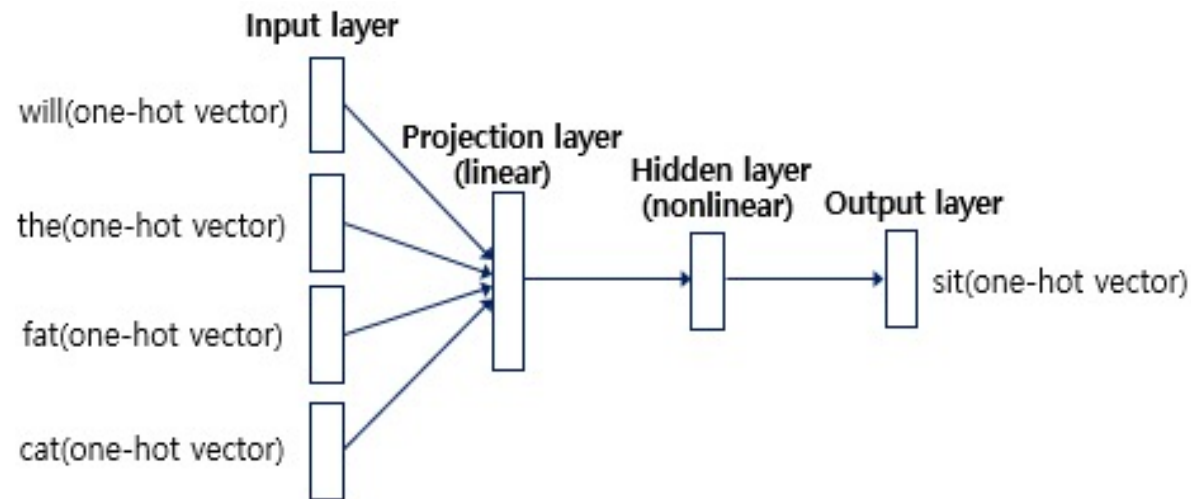
# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

: 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장

: 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

eg) what will the fat cat sit on



$N$  : input으로 들어가는 이전 단어의 개수,  $V$  : vocabulary size  
 $M$  : projection 후 차원,  $H$  : hidden layer size

### Input Layer

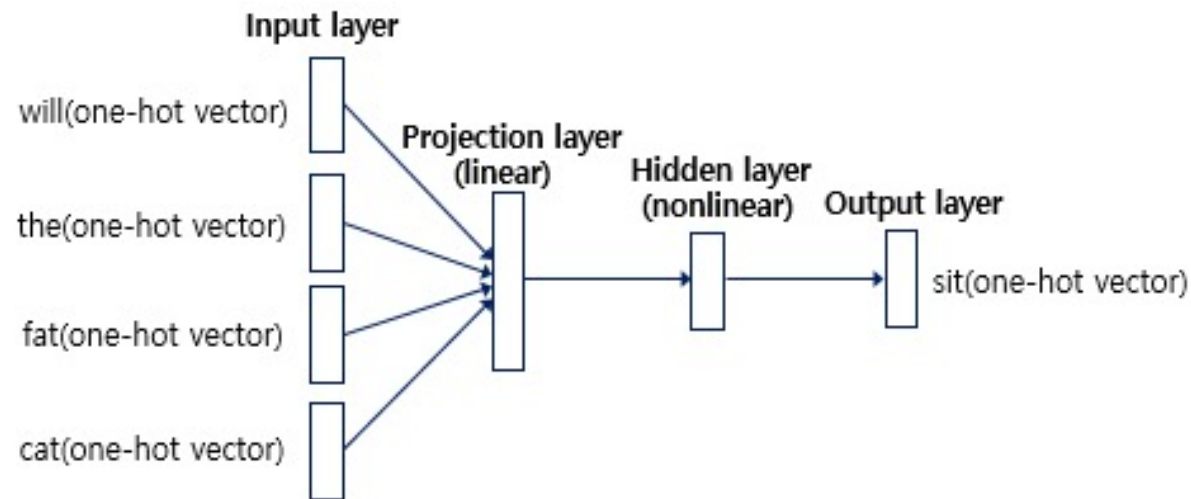
- 현재 보고 있는 것의  $N$ 개의 이전 단어들이 one hot encoding이 되어 들어감
- 이때  $N$ 은 사람이 직접 지정
- $N * V$  차원

# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

- : 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장
- : 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

eg) what will the fat cat sit on



### Projection Layer

- 가중치 행렬( $V \times M$ )와 곱셈

$N$  : input으로 들어가는 이전 단어의 개수,  $V$  : vocabulary size  
 $M$  : projection 후 차원,  $H$  : hidden layer size

# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

- : 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장
- : 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

eg) what will the fat cat sit on

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

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

- 가중치 행렬( $V \times M$ )와 곱셈
- Lookup table 반환 (임베딩 벡터)

N : input으로 들어가는 이전 단어의 개수, V : vocabulary size  
M : projection 후 차원, H : hidden layer size



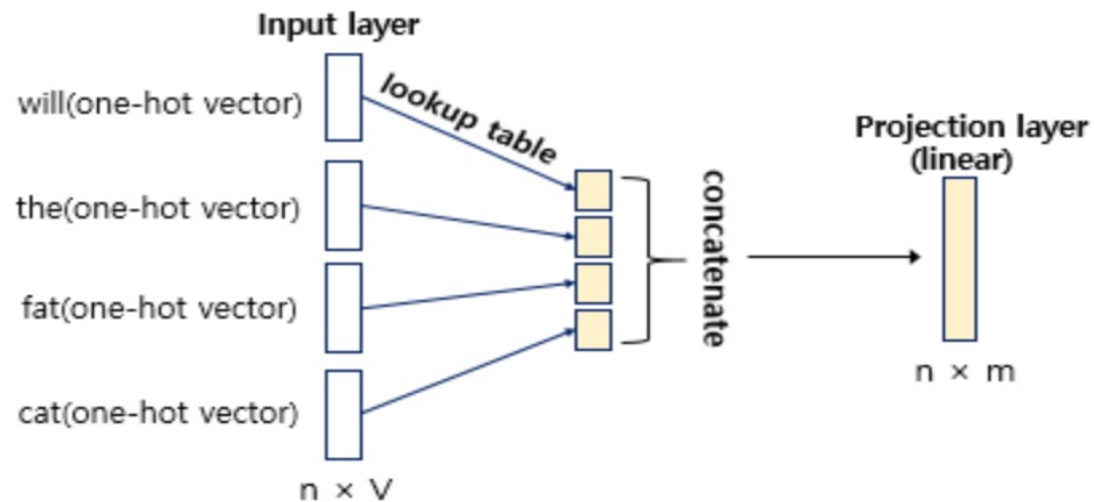
# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

: 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장

: 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

eg) what will the fat cat sit on



### Projection Layer

- 가중치 행렬( $V \times M$ )와 곱셈
- Lookup table 반환 (임베딩 벡터)
- $N \times M$  차원
- 활성화 함수 존재 X

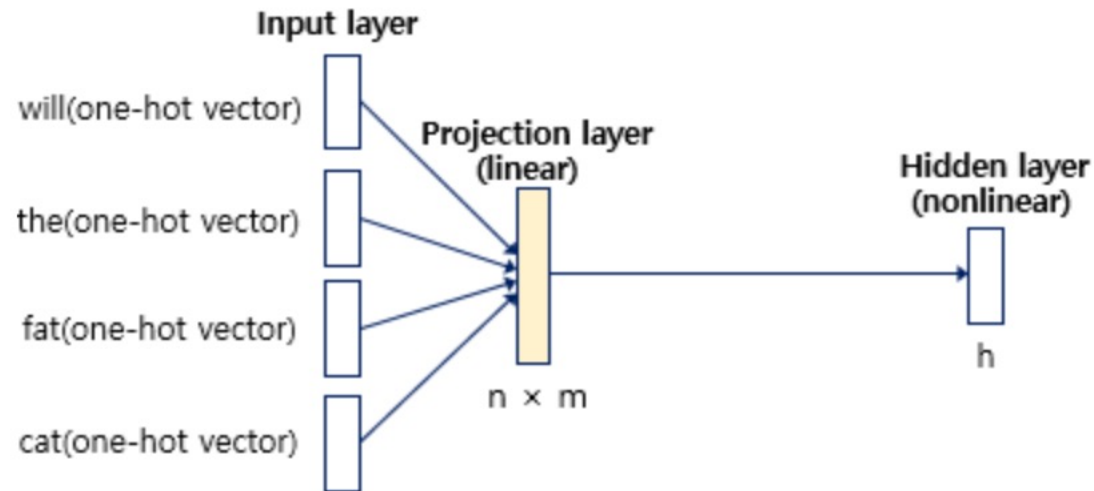
$N$  : input으로 들어가는 이전 단어의 개수,  $V$  : vocabulary size  
 $M$  : projection 후 차원,  $H$  : hidden layer size

# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

- : 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장
- : 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

eg) what will the fat cat sit on



### Hidden Layer

- 투영층의 값이 들어가서 단어들의 확률 분포 계산에 사용
- V차원의 출력 반환

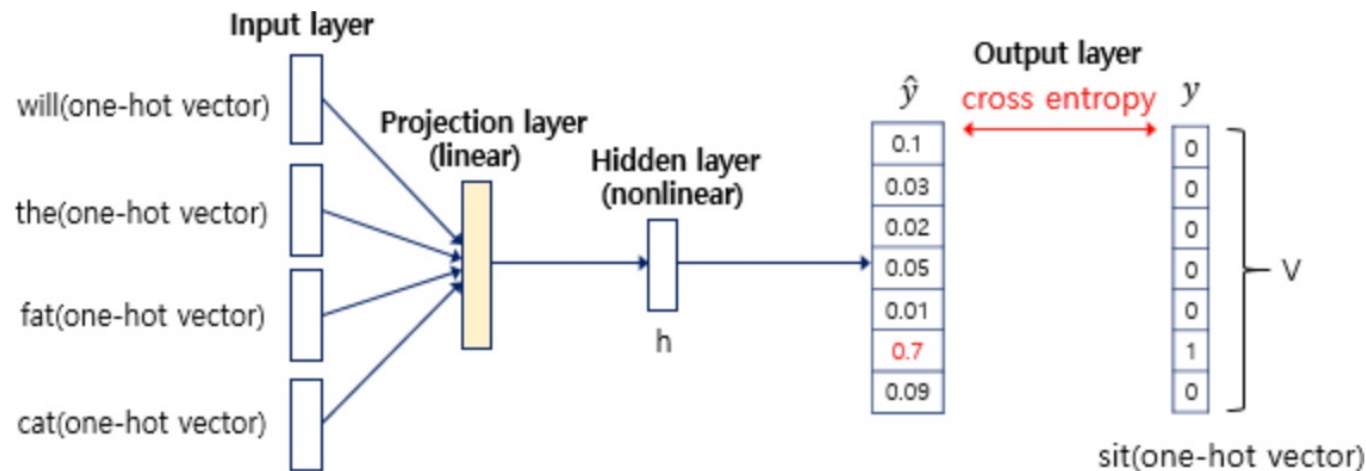
N : input으로 들어가는 이전 단어의 개수, V : vocabulary size  
M : projection 후 차원, H : hidden layer size

# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

- : 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장
- : 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

eg) what will the fat cat sit on



### Output Layer

- 예측하는 정답에 해당하는 단어의 one-hot 벡터와 은닉층의 결과값 사이의 오차를 구함
- error 구한 후 역전파를 이용해 update

$N$  : input으로 들어가는 이전 단어의 개수,  $V$  : vocabulary size  
 $M$  : projection 후 차원,  $H$  : hidden layer size

# Word2Vec : Previous Works

## NNLM (Feed-Forward Neural Net Language Model)

: 분산표현을 이용하여 차원의 저주 문제를 해결하기 위해 등장

: 순차적으로 단어표현이 등장시 다음 단어를 예측하는 모델

### 한계

1. History로 볼 단어의 개수를 고정해주어야 함
2. History만 가지고 예측을 진행하여 미래시점의 단어들을 고려하지 않음
3. 모델 복잡도 :

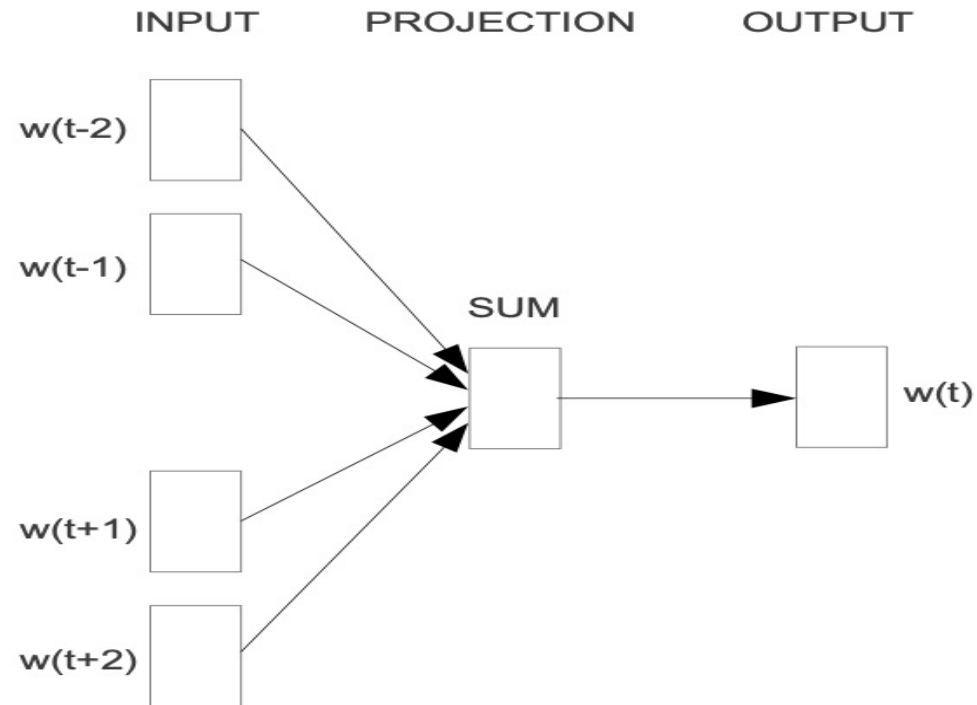
$$Q = N \times D + N \times D \times H + H \times V$$

# Word2Vec : CBOW

## CBOW (Continuous Bag of words model)

: 주변에 있는 단어들로 중간에 있는 단어를 예측하는 방법

: 예측 단어 = 중심단어, 예측 단어 주위의 단어 = 주변단어



# Word2Vec : CBOW

## CBOW (Continuous Bag of words model)

: 주변에 있는 단어들로 중간에 있는 단어를 예측하는 방법

: 예측 단어 = 중심단어, 예측 단어 주위의 단어 = 주변단어

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]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0]	[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]

중심 단어를 예측하기 위해서 주변 단어를 몇 개 볼지의 범위 : window  
window 위치를 변경해가며 training sample 생성 : sliding window

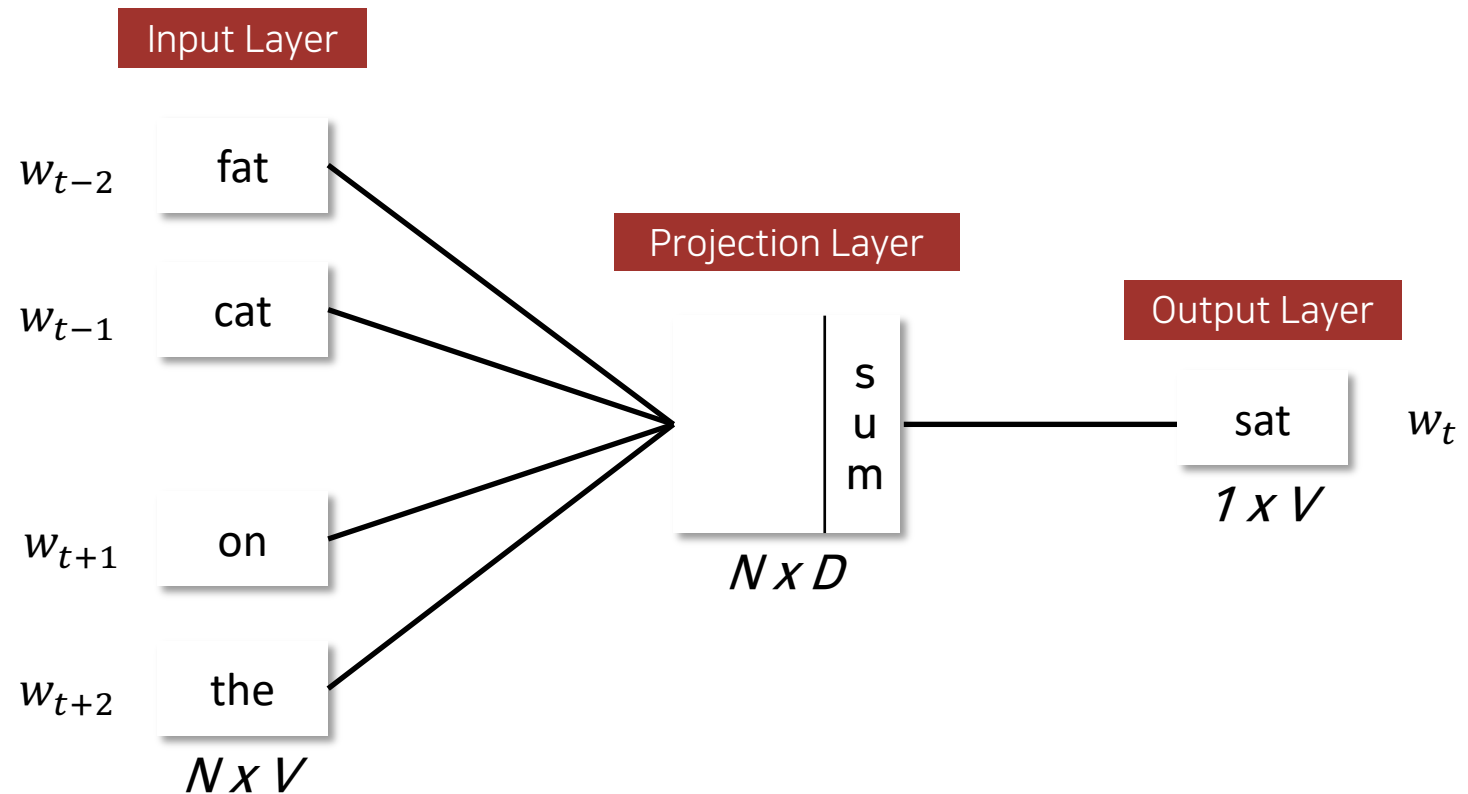
# Word2Vec : CBOW

## CBOW (Continuous Bag of words model)

: 주변에 있는 단어들로 중간에 있는 단어를 예측하는 방법

: 예측 단어 = 중심단어, 예측 단어 주위의 단어 = 주변단어

The fat cat **sat** on the mat



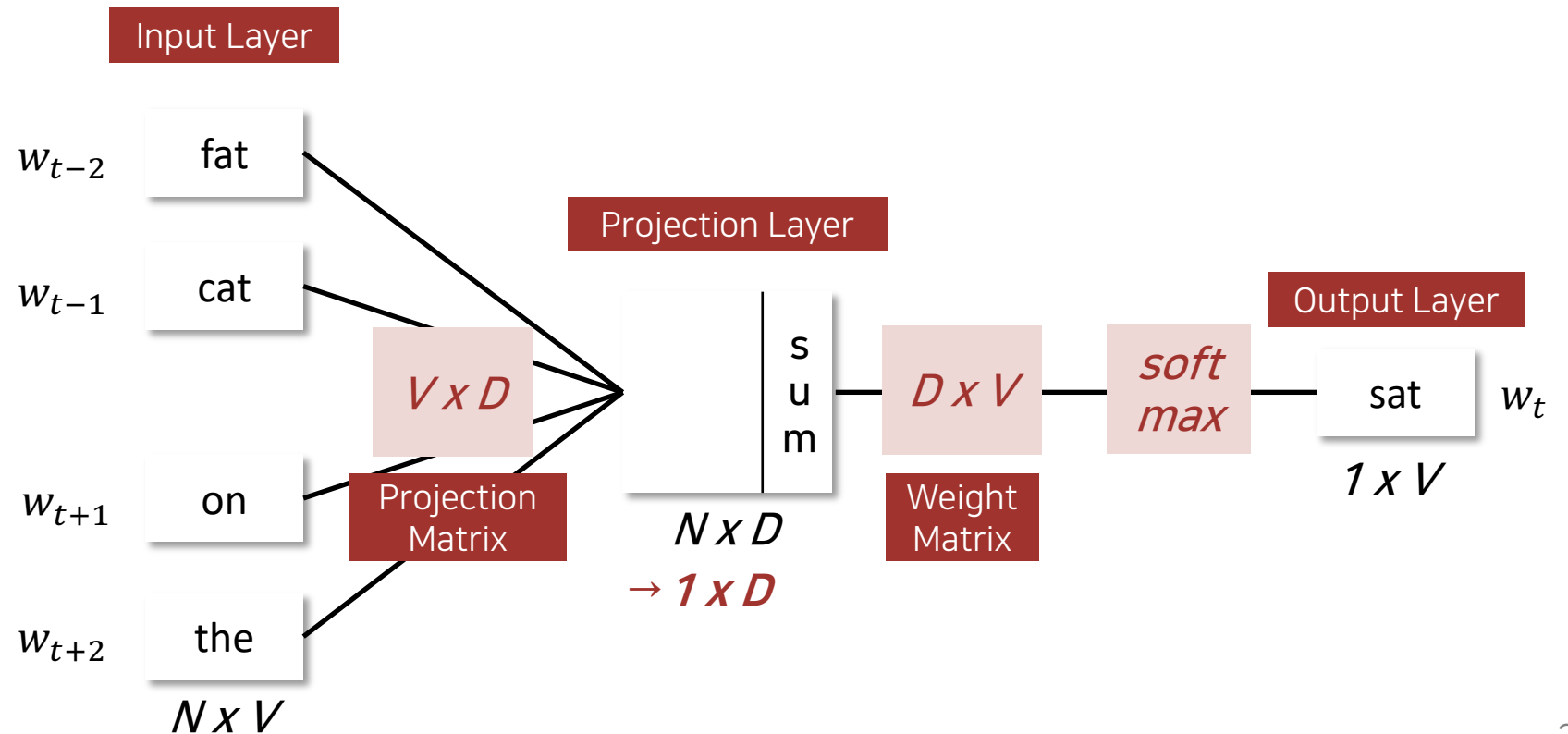
# Word2Vec : CBOW

## CBOW (Continuous Bag of words model)

: 주변에 있는 단어들로 중간에 있는 단어를 예측하는 방법

: 예측 단어 = 중심단어, 예측 단어 주위의 단어 = 주변단어

The fat cat **sat** on the mat



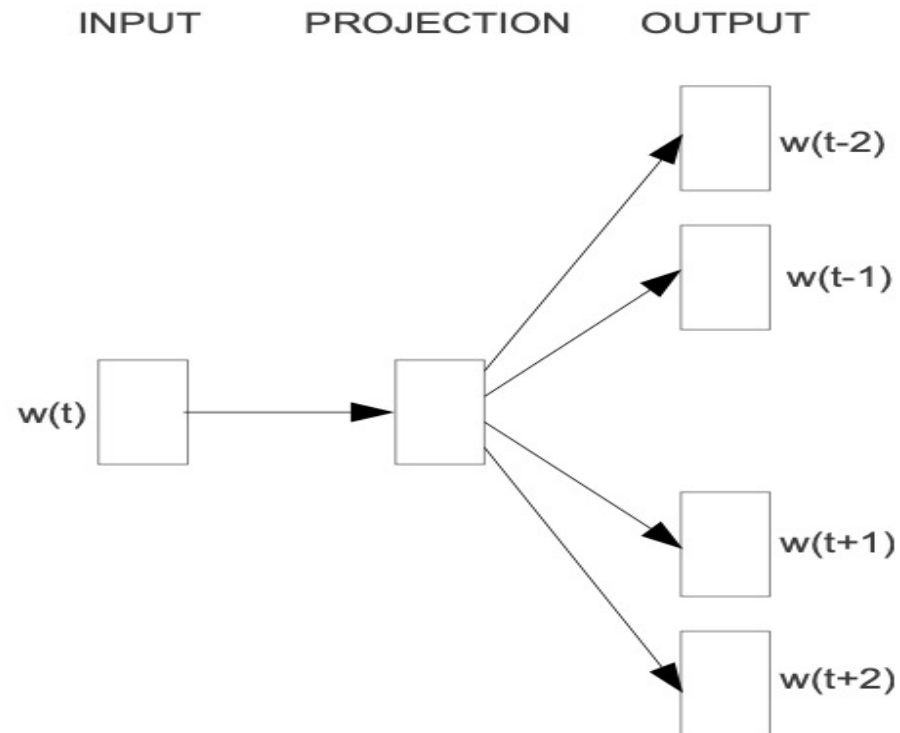


# Word2Vec : Skip-gram

## Skip-gram (Continuous Skip gram model)

: 중심에 있는 단어들로 주변에 있는 단어를 예측하는 방법

: 중심 단어 기준으로 양옆  $c$ 개의 단어가 어떤 단어가 나올 때 최적의 확률을 가지는지 예측

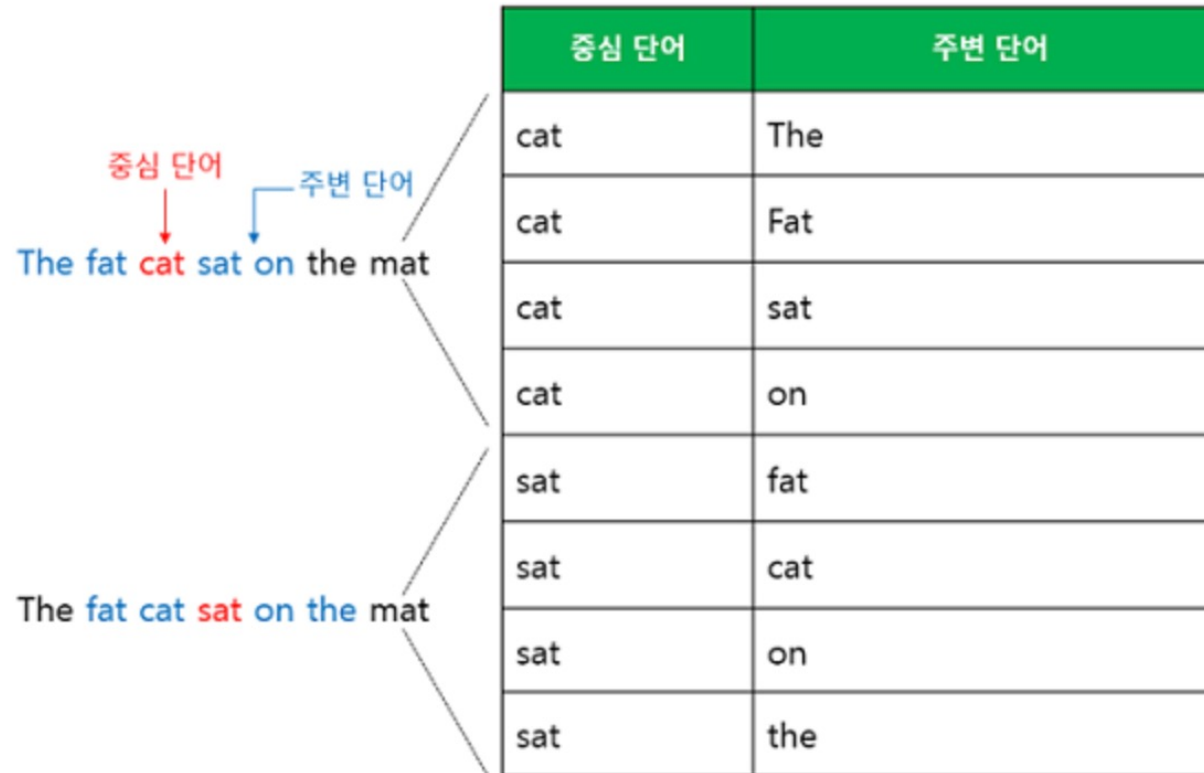


# Word2Vec : Skip-gram

## Skip-gram (Continuous Skip gram model)

: 중심에 있는 단어들로 주변에 있는 단어를 예측하는 방법

: 중심 단어 기준으로 양옆  $c$ 개의 단어가 어떤 단어가 나올 때 최적의 확률을 가지는지 예측



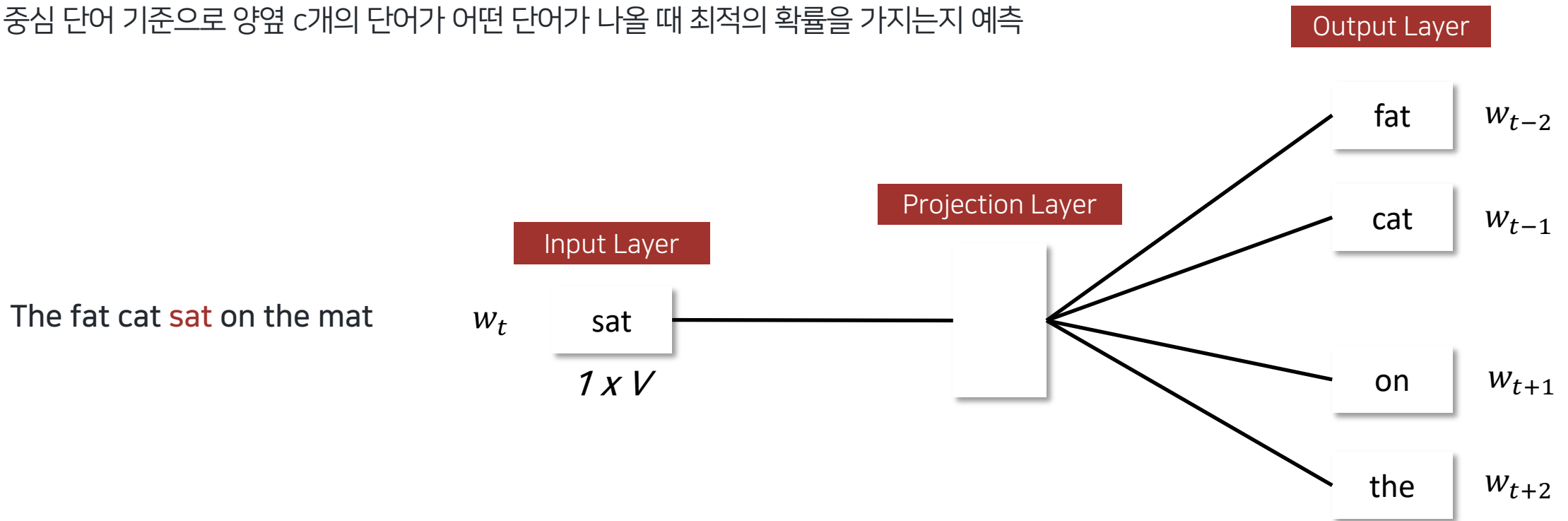
중심 단어	주변 단어
cat	The
cat	Fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the

# Word2Vec : Skip-gram

## Skip-gram (Continuous Skip gram model)

: 중심에 있는 단어들로 주변에 있는 단어를 예측하는 방법

: 중심 단어 기준으로 양옆  $c$ 개의 단어가 어떤 단어가 나올 때 최적의 확률을 가지는지 예측



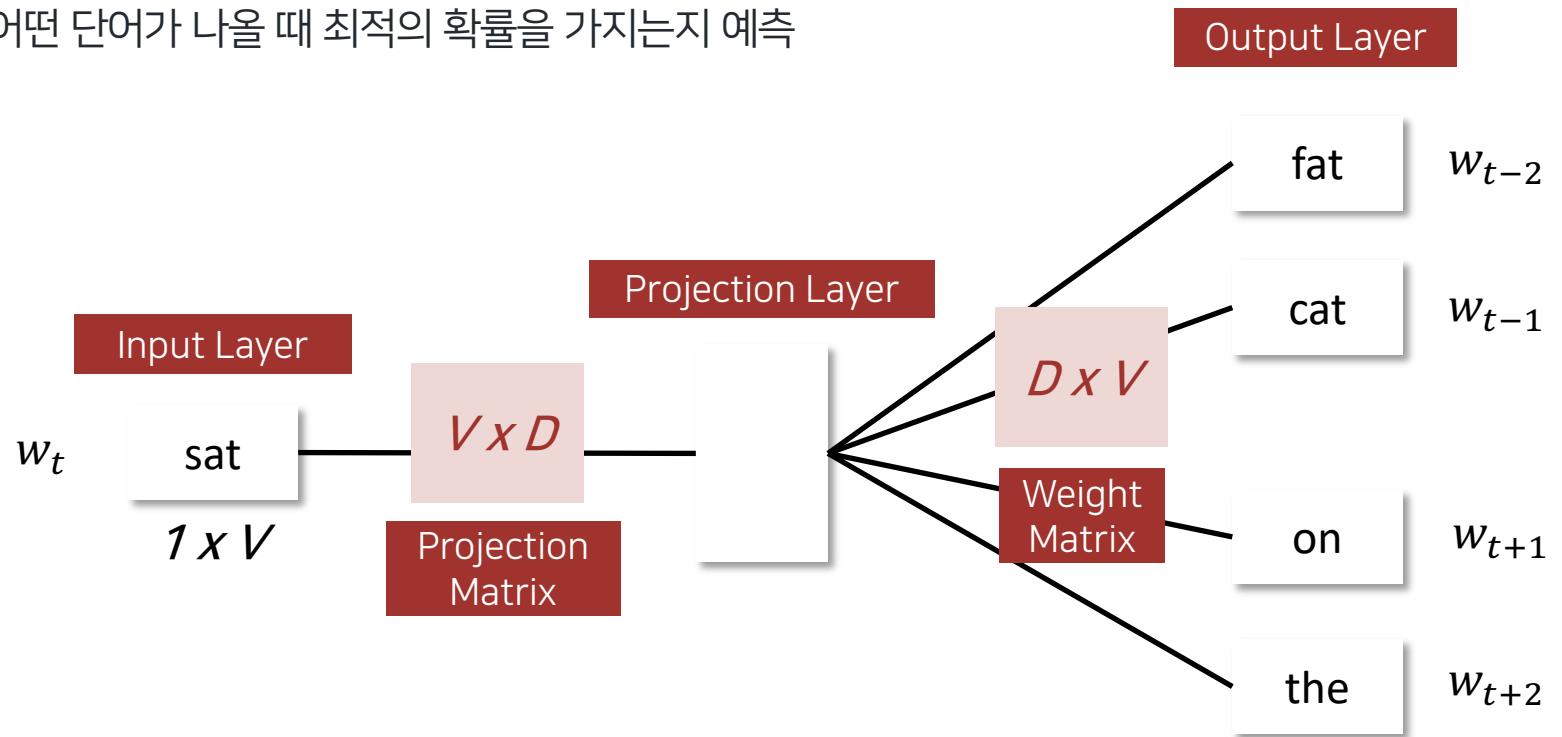
# Word2Vec : Skip-gram

## Skip-gram (Continuous Skip gram model)

: 중심에 있는 단어들로 주변에 있는 단어를 예측하는 방법

: 중심 단어 기준으로 양옆  $c$ 개의 단어가 어떤 단어가 나올 때 최적의 확률을 가지는지 예측

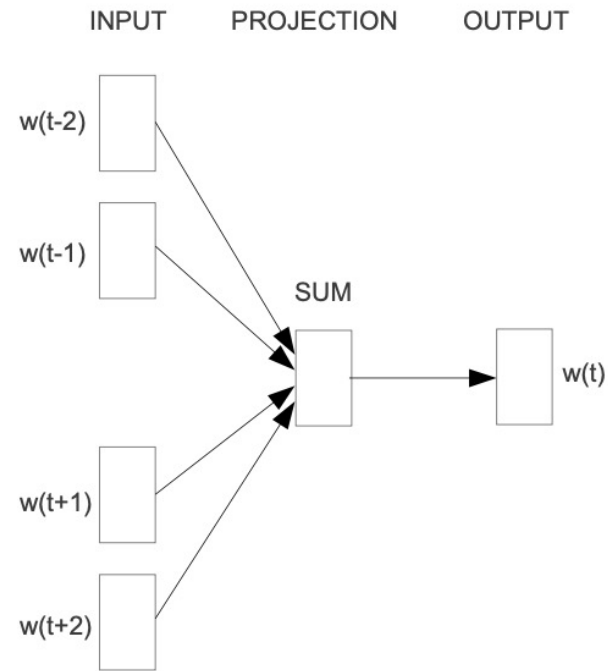
The fat cat **sat** on the mat



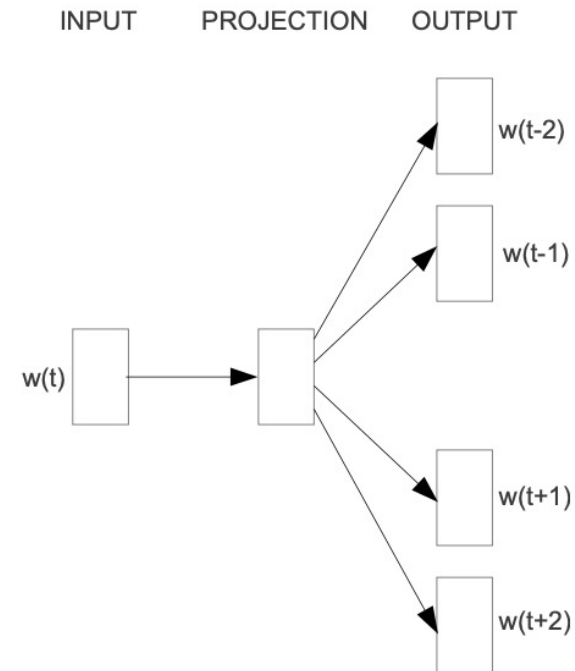
# Word2Vec : CBOW vs Skip-gram

## CBOW vs. Skip-gram

: 일반적으로 CBOW보다 Skip-gram이 성능이 우수하다는 연구 결과



**CBOW**

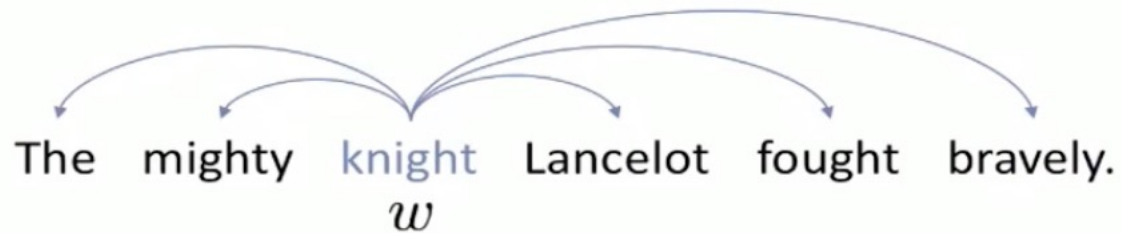


**Skip-gram**

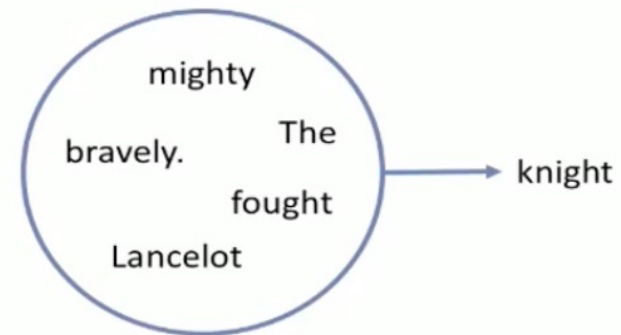
# Word2Vec : CBOW vs Skip-gram

## CBOW vs. Skip-gram

: 일반적으로 CBOW보다 Skip-gram이 성능이 우수하다는 연구 결과

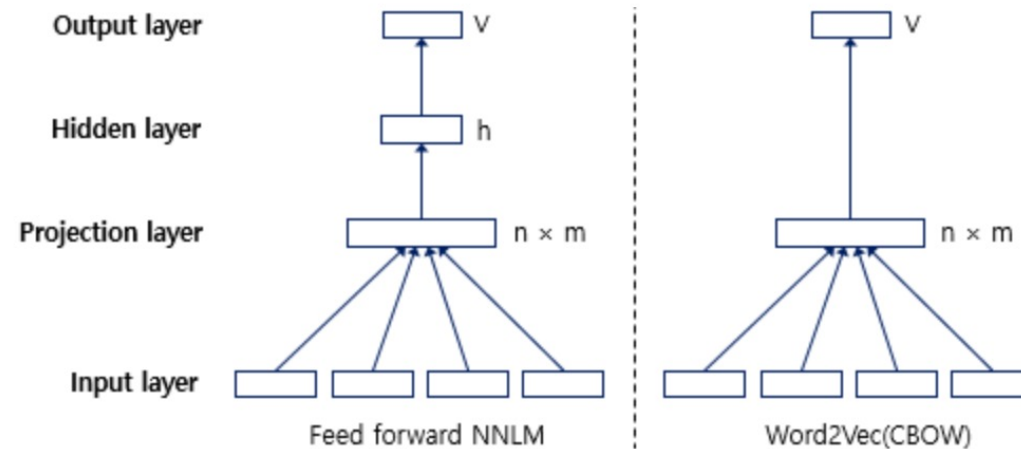


knight → The  
knight → mighty  
knight → Lancelot  
knight → fought  
knight → bravely.



# Word2Vec : NNLM vs. Word2Vec

## NNLM vs. Word2Vec



	NNLM	Word2Vec
예측 목적	다음 단어 예측	워드 임베딩
예측 대상	다음 단어	중심단어
사용하는 단어	과거 단어만 사용	과거, 미래 단어 모두 사용
구조	은닉층 0	은닉층 X

# Why Word2Vec?

## Computational Complexity 개선

$$\text{NNLM} : Q = N \times D + N \times D \times H + H \times V$$

$$\text{CBOW} : Q = N \times D + D \times V$$

$$\text{Skip-gram} : Q = C \times (D + D \times V)$$

V의 곱만큼 complexity가 존재 → 학습 속도에 영향

너무 단어가 많으면 학습 시간이 오래 걸림 but 단어가 많아야 좋은 성능의 모델



# Why Word2Vec?

---

## Computational Complexity 개선

$$\text{NNLM} : Q = N \times D + N \times D \times H + H \times V$$

$$\text{CBOW} : Q = N \times D + D \times V$$

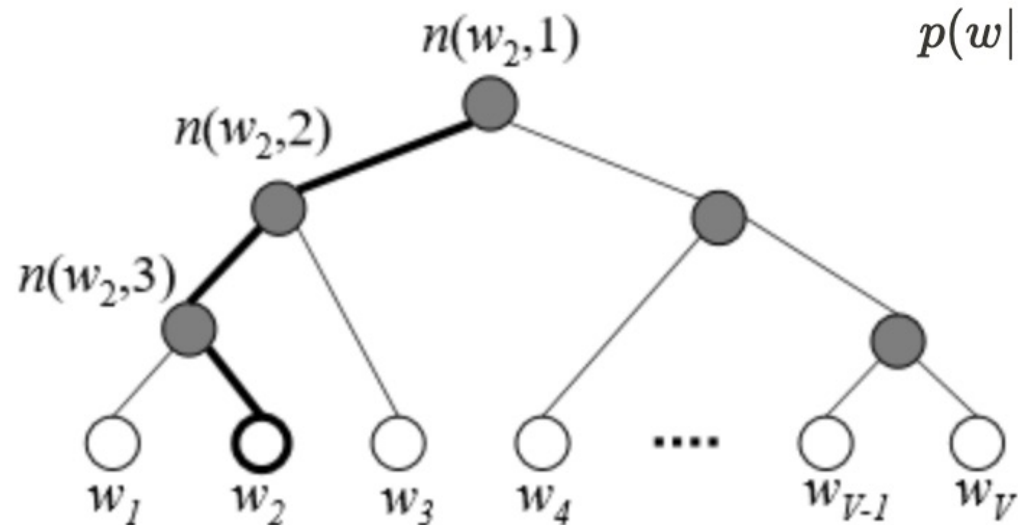
$$\text{Skip-gram} : Q = C \times (D + D \times V)$$

→ Hierarchical Softmax 사용

# Why Word2Vec?

## Hierarchical Softmax

: 모든 단어 V만큼의 확률값을 다 구하지 않고도 특정 확률 추출 가능



$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j+1) = ch(n, w, j)]) \cdot v'_{n(w, j)}{}^T v_{w_I}$$

$n(w, j)$  : 뿌리부터 단어  $w$ 까지의 경로 중  $j$ 번째 마디

$L(w)$  : 경로의 길이

$ch(n)$  : 현재 노드의 왼쪽 자식

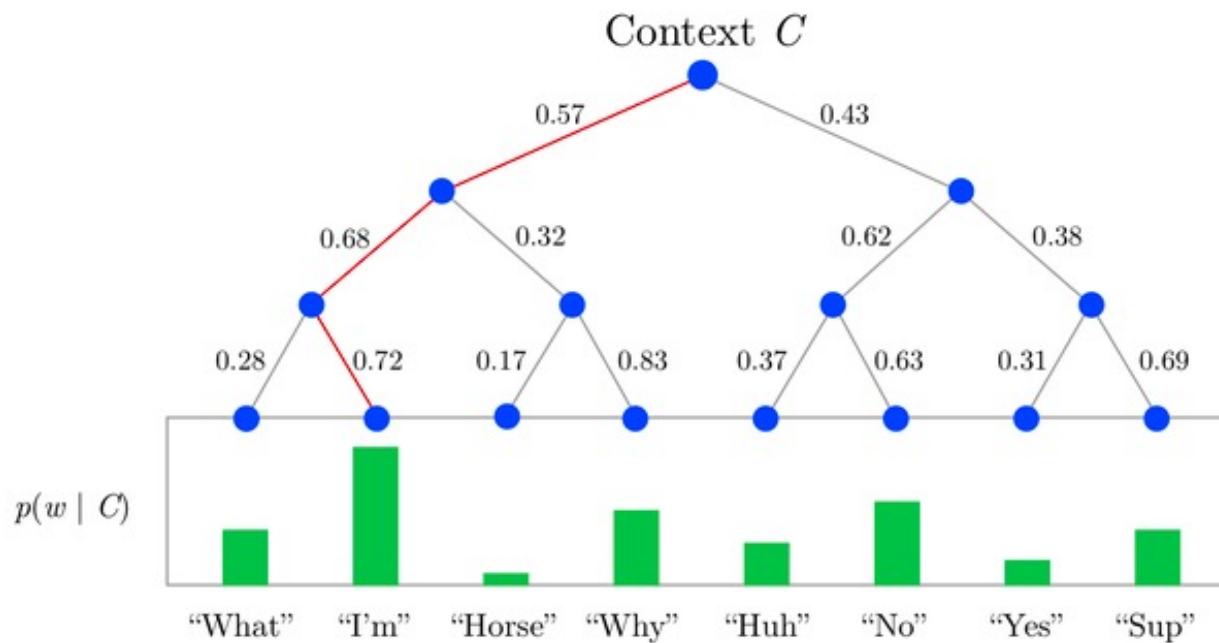
$[x]$  : 만약  $x$ 가 true면 값이 1, 아니면 값이 -1인 함수

$v'_n$  : 내부 마디  $n$ 의 벡터 표현

# Why Word2Vec?

## Hierarchical Softmax

: 모든 단어 V만큼의 확률값을 다 구하지 않고도 특정 확률 추출 가능



context C가 주어졌을 때 I'm이 나올 확률 :  $p(I'm | C) = p(n(I'm, 1), left) \times p(n(I'm, 2), left) \times p(n(I'm, 3), right)$

$$= \sigma(v_{n(w_2,1)}^T v_{w_i}) \cdot \sigma(v_{n(w_2,2)}^T v_{w_i}) \cdot \sigma(-v_{n(w_2,3)}^T v_{w_i})$$

# Why Word2Vec?

## Hierarchical Softmax

: 모든 단어  $V$ 만큼의 확률값을 다 구하지 않고도 특정 확률 추출 가능

$$\text{CBOW} : Q = N \times D + D \times \log_2 V$$

$$\text{Skip-gram} : Q = C \times (D + D \times \log_2 V)$$

계층적 소프트맥스 : 다른 확률을 구할 필요 없이 정답단어의 확률만을 구할 수 있음  
→ 길이  $D$ 에 대한 벡터내적을 이진트리의 깊이만큼만 해도 됨  $\Rightarrow \log_2(V)$

**$\Rightarrow$  연산량 감소 !**

# Evaluation

---



# Evaluation

## Maximization of Accuracy

: 모델 정확도 향상을 위해 단어 벡터의 차원과 train set의 size를 조정

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

특정 지점이 지나면 차원 or training data를 더 추가하는 것 → 성능이 오히려 줄어듦  
⇒ 벡터 차원과 train data를 함께 증가시키는 것이 필요

# Evaluation

## Comparison of Model Architecture 1.

: 모델 정확도 향상을 위해 단어 벡터의 차원과 train set의 size를 조정

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

CBOW : 다른 모델보다 Sementic 부분에 대해 msr test set의 성능 좋음

Skip-gram : word relationship test set의 sementic에서 가장 좋은 성능

# Evaluation

## Comparison of Model Architecture 2.

: 데이터 사이즈, 벡터 차원, 에폭 횟수에 대한 실험 결과

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

같은 데이터 사이즈로 3번 에폭보다 2배 이상의 데이터로 1번 에폭 : 시간 단축, 성능 향상  
training 사이즈를 2배 늘리기보다 벡터의 차원을 2배 늘리는 것이 더 효과적



# Evaluation

## Word Relationships

: 임베딩한 단어 벡터가 의미를 잘 담을까?

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

두 단어에 대한 연산으로 관계 정의 가능 : eg) 파리 - 프랑스 + 이탈리아 = 로마  
나라와 수도의 관계를 단어 벡터가 잘 학습한 것을 확인할 수 있음

# Evaluation & Limitation

---

## Limitation

### 1) 단어의 형태학적 특성을 반영하지 못함

'teach', 'teacher', 'teachers'와 같이 세 단어는 의미적으로 유사한 단어이지만, 각 단어를 개별 단어(unique word)로 처리  
→ 세 단어 모두 벡터 값이 다르게 구성

### 2) 분포 가설을 기반으로 학습 → 단어 빈도 수의 영향을 많이 받음

⇒ 드물게 나타나는 단어를 임베딩하기 어려움

### 3) OOV(Out of Vocabulary)의 처리가 어려움

: 사전에 없는 새로운 단어가 등장시 데이터 전체를 다시 학습시켜야 함

# 참조

---

[https://www.goldenplanet.co.kr/our\\_contents/blog?number=859&pn=1](https://www.goldenplanet.co.kr/our_contents/blog?number=859&pn=1)

<https://wikidocs.net/22660>

<https://mambo-coding->

[note.tistory.com/entry/%EC%9E%90%EC%97%B0%EC%96%B4%EC%B2%98%EB%A6%AC-Word-Embedding-Word2Vec](https://note.tistory.com/entry/%EC%9E%90%EC%97%B0%EC%96%B4%EC%B2%98%EB%A6%AC-Word-Embedding-Word2Vec)

<https://velog.io/@lee9843/Word2Vec%EC%9D%84->

[%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-%EC%9C%84%ED%95%9C-%EB%82%B4%EC%9A%A93-Hierarchical-softmax](https://velog.io/@lee9843/Word2Vec%EC%9D%84-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0-%EC%9C%84%ED%95%9C-%EB%82%B4%EC%9A%A93-Hierarchical-softmax)

<https://youtu.be/s2KePv-OxZM?si=iqrnpUiKw0lScCgJ>

<https://youtu.be/ERibwqs9p38?si=54S3127SDBd2lAtl>



TRAIN AND TEST