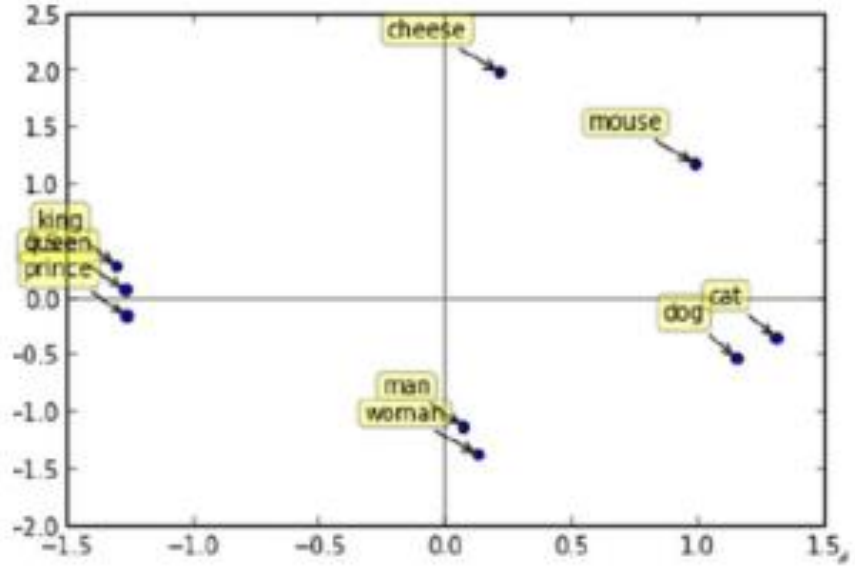
마무리

1. 텍스트 마이닝 최근 트렌드

- **word2vec**

- 단어(혹은 단어 쌍)를 벡터공간에 표현
- 유사한 의미나 관계의 단어는 유사한 벡터로 표현됨
- Input text를 받아 학습하는 지도학습(Supervised Learning) 방식
- 기존 텍스트 마이닝에 적용되던 머신 러닝 기법에 비해 성능이 좋음

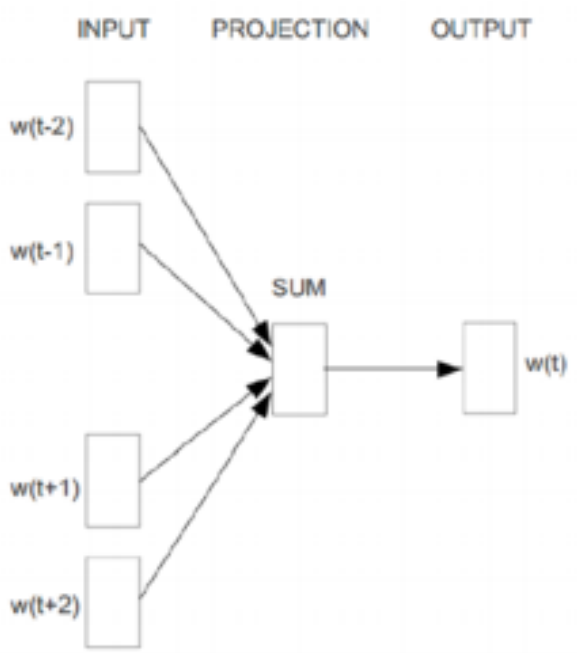


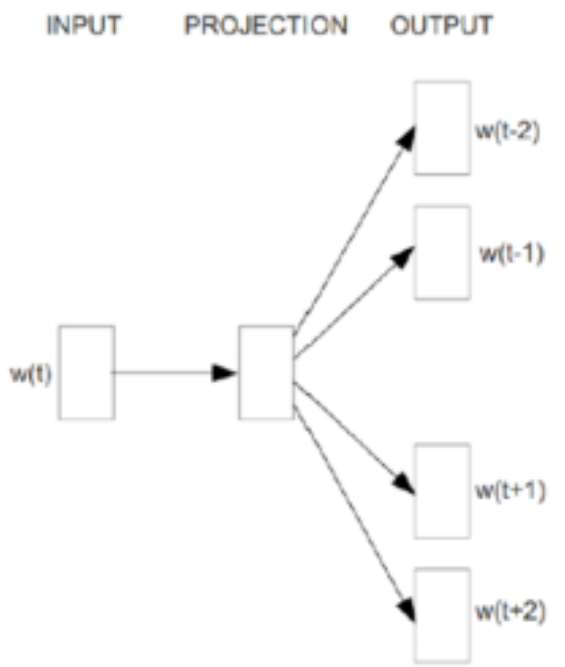
마무리

1. 텍스트 마이닝 최근 트렌드

- **word2vec**

- CBOW (Continuous Bag Of Words) vs Skip-gram





Word2Vec 목차

1. Word Embedding

의미 : 문자로 이루어진 단어를 숫자로 변환하는 것

Why 필요? 컴퓨터가 인식하게 하기 위해

컴퓨터는 숫자만 인식할 수 있기 때문에

한글이나 그림 등 컴퓨터가 다루는 모든 것들은 반드시 숫자로 변환되어야 함

또한, 기존의 단어 자체를 유니코드 등으로 처리하는 방식으로는

의미 추론이 불가능

Ex) king – queen : 남자, 여자라는 속성만 다를 뿐 유사한 단어. 하지만 기존 방식으로는 이걸 인지하지 못했음. '왕'이라는 단어와 가까운 단어가 무엇인지 알고 싶기 때문에 단어를 숫자로 변환할 필요가 있었음

1. Word Embedding

[초기 모델]

- NNLM (Neural Network Language Model)
- RNNLM (Recurrent NNLM)

[최근]

- **Word2Vec**
  1) **CBOW**
  2) **Skip-Gram**

- word embedding 프로세스
  : 신경망, 차원감소, 확률적 모델, 문맥상 표현 등의 여러 가지 처리 과정들이 포함

- NNLM은 단어 벡터화 학습에 좋은 방법이나 굉장히 많은 단어 데이터를 학습시켜야 하는 관계로 성능에 문제를 안고 있다. 이후 NNLM의 이러한 문제점을 보완하여 빠른 학습이 가능하게 된 Word2Vec이 등장하게 된다.

2. Word2Vec

: 단어를 숫자로 변환해서 다음에 올 단어를 예측하는 모델

- 다음 단어를 예측하기 위해 Word Embedding을 구현해야 하는데
  이 때 **1) CBOW** or **2) Skip-Gram** 알고리즘을 사용

[특징1] 추론(예측) 가능

변환된 벡터가 단순한 수학적 존재 이상의 복잡한 개념 표현을 넘어 추론까지도 쉽게 구현할 수 있다

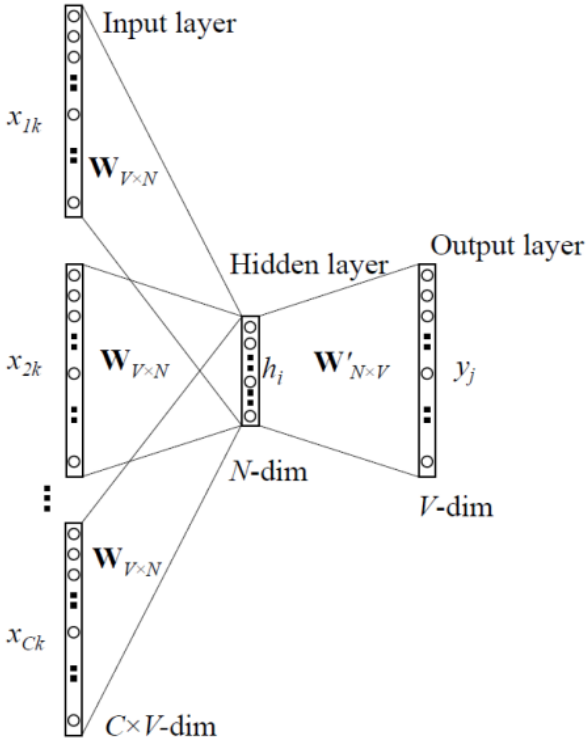
▷ 자연어 처리(NLP)의 엄청난 발전

[특징2] 계산량의 획기적인 감소

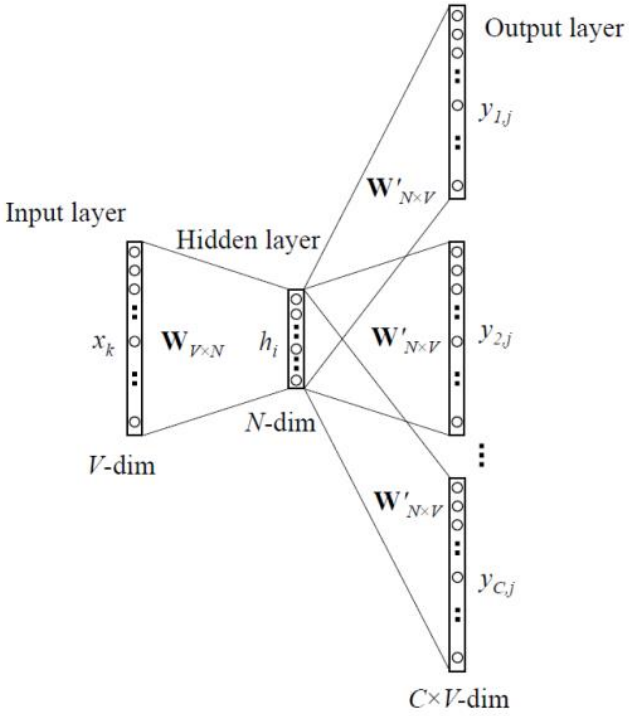
기존 Neural Net 기반 학습방법에 비해 크게 달라진 것은 아니지만,
계산량을 엄청나게 줄여서 기존의 방법에 비해 몇 배 이상 **빠른 학습을 가능**하게 함

▷ 가장 인기있는 Word Embedding 모델 등극

2. Word2Vec



CBOW embedding



Skip-Gram embedding

3. CBOW & Skip-Gram 기본 개념

- **Single-word context :** '하나'의 단어로부터 다음에 오는 단어를 예측
- **Multi-word context : '**여러 개**'**의 단어를 나열, 그에 기반하여 단어를 추정

- **Context (컨텍스트) :** 특정 단어 주변에 오는 단어들의 집합
  '계산이 이루어지는 단어들'을 의미
  ex) the cat sits on the 에서 sits 양쪽 2개 단어 포함 총 5개의 단어가 컨텍스트가 되는 것 (목표 단어 양쪽 2개 단어만 허용한 경우, 허용 단어 개수는 임의로 지정)

1) CBOW (Continuous Bag-of-words) 모델

컨텍스트로부터 찾고자 하는 목표 단어를 예측하는 모델

ex) the cat sits on the (컨텍스트) → "mat" (목표 단어)

2) Skip-Gram 모델

현재 하나의 단어를 통해 주변 단어들(컨텍스트)를 예측하는 모델

ex) the cat sits of the (컨텍스트) ← "mat" (현재 단어)

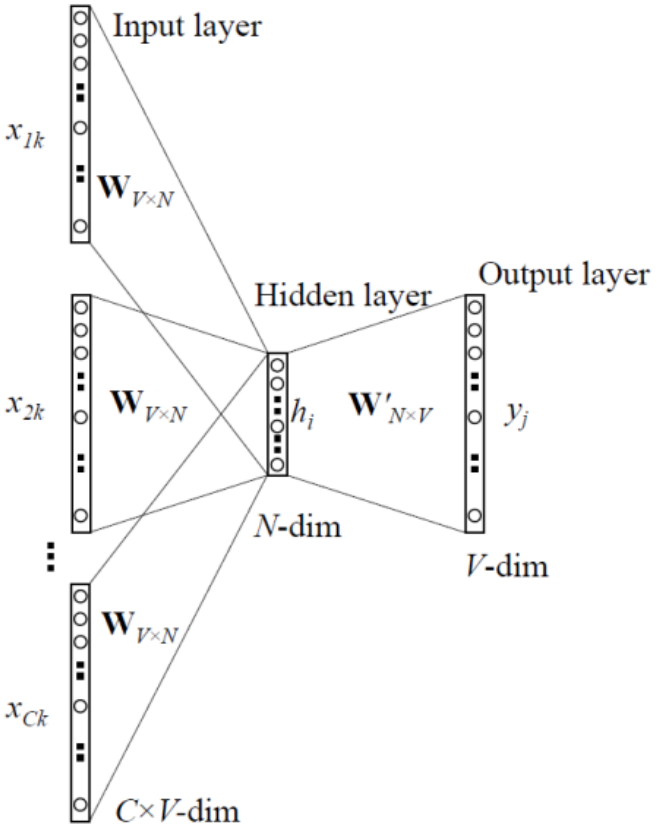
4. CBOW 모델

- **CBOW (Continuous Bag-of-words) 모델**

  컨텍스트로부터 찾고자 하는 목표 단어를 예측하는 모델

  ex) the cat sits on the (컨텍스트) → "mat" (목표 단어)

"아이스크림이 너무 ___ 먹을 수 없었다." 라는 문장에서, 누구나 생략된 ___ 부분의 단어를 추측할 수 있고, 대부분은 옳게 예측한다. CBOW 모델도 마찬가지의 방법을 사용한다.



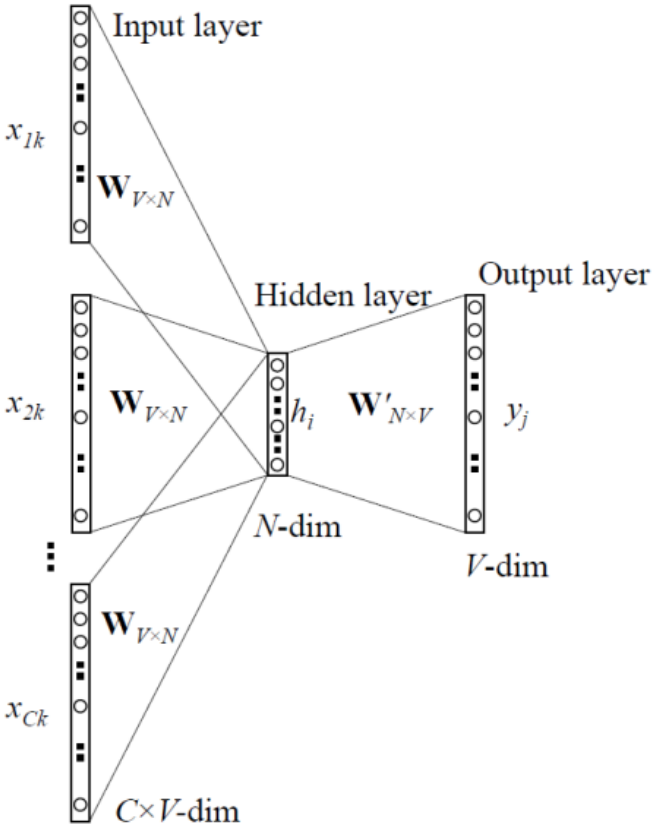
- 주어진 단어에 대해 앞 뒤로 N/2개 씩 총 N개의 단어를 입력으로 사용하여, 주어진 단어를 맞추기 위한 네트워크를 만든다
- 크기가 작은 데이터셋에 적합

4. CBOW 모델

"the quick brown fox jumped over the lazy dog"

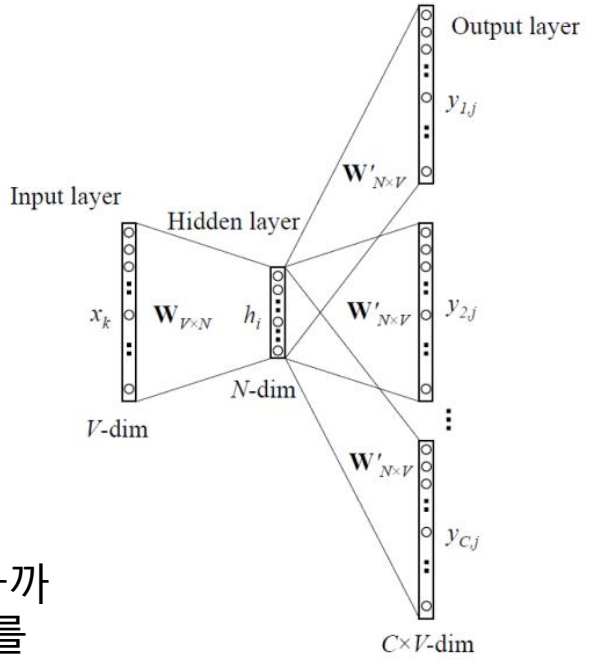
the quick brown (컨텍스트 주어짐) → fox (예측)

각 문맥 단어를 Hidden layer로 투사하는 가중치 행렬 (Wvn) 은 모든 단어(X1k...Xck)에 공통으로 사용



5. Skip-gram 모델

- **Skip-Gram 모델**
  현재 하나의 단어를 통해 주변 단어들(컨텍스트)를 예측하는 모델
  ex) the cat sits of the (컨텍스트) ← "mat" (현재 단어)



일반적으로 입력 단어 주변 k개의 단어를 컨텍스트로 보고 예측 모형을 만듬
- **Window size** : 이 때 k값 (일반적으로 15개 정도 사용)

예측하는 단어들은 현재 단어 주변에서 샘플링하는데, "가까이 있는 단어일수록 현재 단어와 관련이 더 많다"는 원리를 적용하기 위해 **멀리 떨어진 단어를 낮은 확률로 선택**하는 방법을 사용한다.
나머지는 CBOW 모델과 방향만 반대이고, 거의 비슷하다.

- 크기가 큰 데이터셋에 적합
  최근일수록 더욱 많은 데이터를 갖고 있기 때문에 주로 Skip-Gram 모델을 사용

5. Skip-gram 모델

Skip-Gram 모델 구현

the quick brown fox jumped over the lazy dog

1. 컨텍스트 정의
   일반적으로 구문론적(형식적) 컨텍스트를 정의
   일단 컨텍스트를 현재 단어(목표 단어)의 양쪽에 있는 단어들의 윈도우로 정의 해보자

2. Window size 설정 및 데이터셋 도출
   윈도우를 1로 하면 다음과 같은 쌍으로 구성된 데이터셋이 도출됨

   ([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), …

3. 목적 : 'quick' 이라는 현재 단어로 부터 컨텍스트 'the' 와 'brown'을 예측!
   아래와 같은 데이터셋 (단어 관계)을 도출할 수 있어야함

   (quick, the), (quick, brown),
   (brown, quick), (brown, fox), …

6. Word2Vec 실습

- Word2Vec : CBOW, Skip-Gram 모델 방식의 word embedding을 구현한 C++ 라이브러리

- 파이썬 : genism 패키지에 Word2Vec 클래스로 구현되어있음

```
In [30]: model.wv['harry']
```

```
Out[30]: array([ 4.9133077e-03, -2.5425579e-03,  4.6561533e-03, -1.2675900e-03,
                -1.2882315e-03, -7.0305570e-04, -5.7528802e-04,  1.6293961e-03,
                 6.7486783e-04, -2.6372571e-03,  5.8776757e-04, -2.4593857e-03,
                 3.6313613e-03,  3.8350918e-03, -1.8897669e-03,  2.3377636e-03,
                -1.0670263e-03, -4.4244207e-03, -1.2251873e-03, -6.5187173e-04,
                 3.3991393e-03,  2.8919363e-03,  1.5546890e-03,  1.5625909e-04,
                 7.3678483e-04, -5.6945422e-04,  2.5736326e-03,  2.1798320e-03,
                -1.4414199e-03, -1.3510181e-03, -4.1584782e-03, -2.2712421e-04,
                -3.6415402e-03,  2.7404006e-03,  2.9884626e-03,  6.1799720e-04,
                -3.5573458e-03, -4.6700244e-03,  4.1038543e-03,  3.1092749e-03,
                -1.5345910e-03,  7.9275994e-04, -8.9970656e-04,  4.3273340e-03,
                -4.9397821e-04,  1.5217594e-03,  4.1830316e-04, -1.9290635e-03,
                -9.2209244e-05,  2.9839799e-04,  2.5808724e-04,  4.3234797e-03,
                 3.6696834e-03, -3.3607972e-03,  1.4446243e-03, -4.9667759e-04,
                 3.8497348e-03, -2.8031552e-03,  2.2935192e-03, -3.2457921e-03,
                -2.2229622e-03,  1.3333579e-03,  3.0017765e-03, -2.8066552e-04,
                 5.9717884e-05,  3.8402050e-03,  4.6476279e-04,  4.6742167e-03,
                -1.1703372e-03, -4.6434286e-03, -1.9046805e-03, -2.8519370e-03,
                 1.2447287e-03, -4.9562980e-03,  3.7071125e-03, -2.8543253e-03,
                -1.4122591e-03, -2.9552144e-03, -1.5373582e-03,  2.2390769e-03,
                 3.6536106e-03,  3.7666881e-03, -8.9312397e-04,  5.5225630e-04,
                -2.4419581e-03,  4.8720338e-03,  2.8404179e-03, -1.0651306e-03,
                -1.7772804e-03,  1.4266926e-04, -4.7169486e-03, -4.6677766e-03,
                 6.3740415e-04, -2.2220563e-03,  8.8301674e-04, -4.2092111e-03,
                -2.6657644e-03, -4.8563234e-03,  5.8039650e-04,  2.8722433e-03],
               dtype=float32)
```

In [40]:

```
## word_vector와 내장함수로 여러가지를 구해볼 수 있다
def word_correlation(word_vector, a, b, c):
    return word_vector.most_similar_cosmul(positive=[a, c], negative=[b])[0][0]


def word_find_most_similar(word_vector, word):
    return word_vector.most_similar(word)[0][0]


def word_odd_one(word_vector, phrase):
    return word_vector.doesnt_match(phrase.split())

def similarity(word_vector,a,b):
    return word_vector.similarity(a,b)

print(word_correlation(model.wv, 'harry', 'voldemort', 'ron'))
print(word_find_most_similar(model.wv, 'ron'))
print(word_odd_one(model.wv,'He had been hugged by a complete stranger'))
print(similarity(model.wv,'harry','ron'))
```

```
fer
train
complete
0.08751295297576796
```

Reference

https://dreamgonfly.github.io/machine/learning,/natural/language/processing/2017/08/16/word2vec_explained.html : 쉽게 쓰여진 word2vec

http://pythonkim.tistory.com/92

https://shuuki4.wordpress.com/2016/01/27/word2vec-%EA%B4%80%EB%A0%A8-%EC%9D%B4%EB%A1%A0-%EC%A0%95%EB%A6%AC/

https://www.nextobe.com/single-post/2017/06/20/Word-Embedding%EC%9D%98-%EC%A7%81%EA%B4%80%EC%A0%81%EC%9D%B8-%EC%9D%B4%ED%95%B4-Count-Vector%EC%97%90%EC%84%9C-Word2Vec%EC%97%90-%EC%9D%B4%EB%A5%B4%EA%B8%B0%EA%B9%8C%EC%A7%80

http://ngio.co.kr/m/5324

https://blog.naver.com/jogakdal/221058303993

https://datascienceschool.net/view-notebook/6927b0906f884a67b0da9310d3a581ee/

https://shuuki4.wordpress.com/2016/01/27/word2vec-%EA%B4%80%EB%A0%A8-%EC%9D%B4%EB%A1%A0-%EC%A0%95%EB%A6%AC/

https://www.nextobe.com/single-post/2017/06/20/Word-Embedding%EC%9D%98-%EC%A7%81%EA%B4%80%EC%A0%81%EC%9D%B8-%EC%9D%B4%ED%95%B4-Count-Vector%EC%97%90%EC%84%9C-Word2Vec%EC%97%90-%EC%9D%B4%EB%A5%B4%EA%B8%B0%EA%B9%8C%EC%A7%80