

# 텍스트 마이닝의 이론과 실제

\*\* 수업용이며 배포 금지<sup>1</sup>입니다.

## Table of Contents

- 텍스트 마이닝의 이해
- 텍스트 마이닝 방법론
  - 도구 및 원리의 이해
- 텍스트 마이닝의 문제점
- 문제 해결을 위한 방안
  - 기존의 방법
  - 딥러닝에 의한 방법

## 텍스트 마이닝 (Text Mining)이란?

- Wikipedia: the process of **deriving high-quality information from text**.
- High-quality information is typically derived through the devising of **patterns and trends** through means such as **statistical pattern learning**.
- The overarching goal is, essentially, to **turn text into data for analysis**, via application of natural language processing (NLP) and analytical methods.

# 텍스트 마이닝 (Text Mining)이란?

- Turn unstructured text into structured data,
  - (일정한 길이 (sparse or dense) 의 vector)로 변환
- and use the structured data in various tasks such as
  - text classification, clustering, sentiment analysis, document summarization, translation, prediction, etc.
  - (변환된 vector)에 (머신러닝 (딥러닝) 기법)을 적용



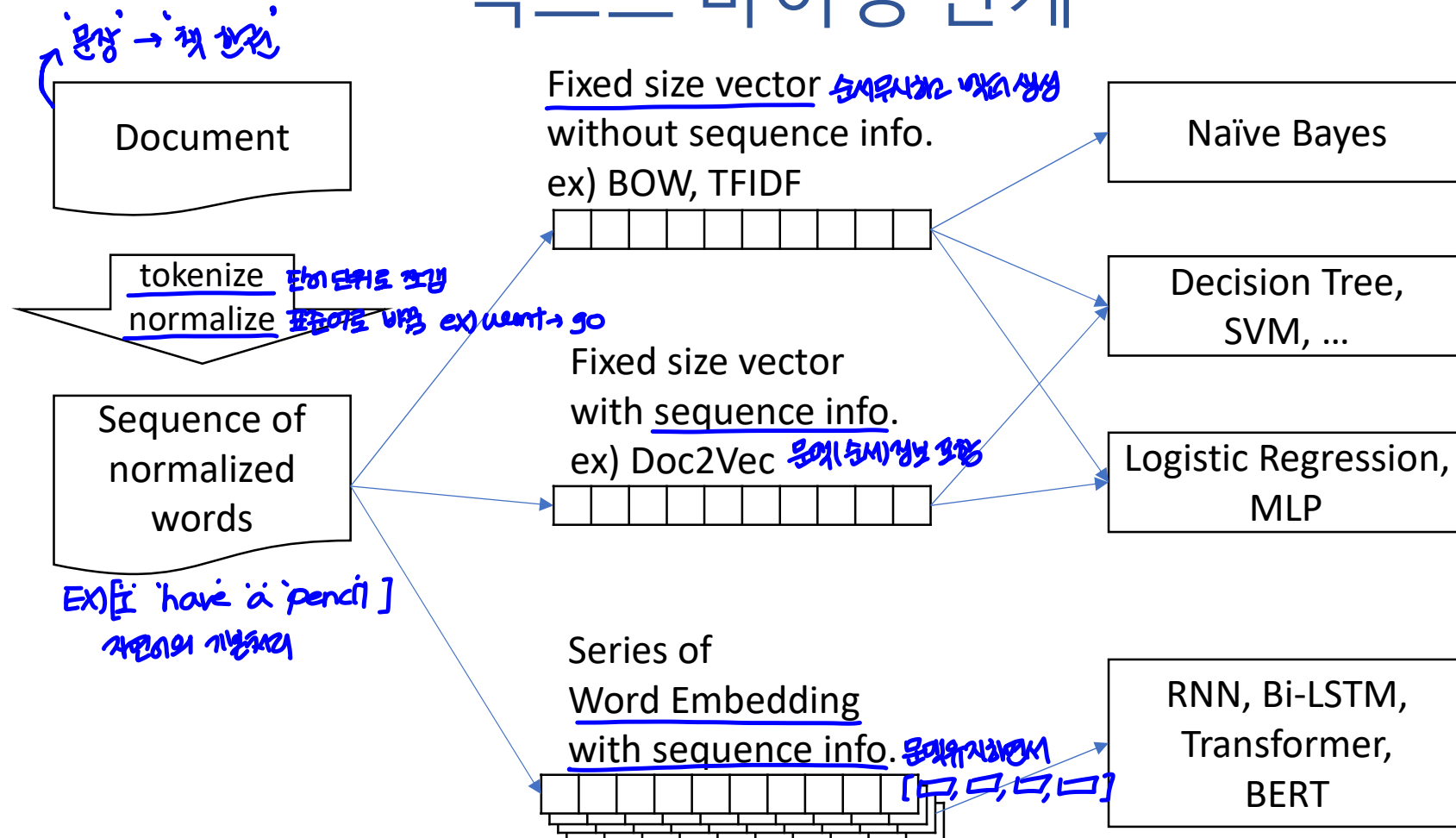
## 텍스트 마이닝의 이해를 위한 기본요구지식

- 자연어 처리
  - 강의자료에서 정리
- 통계학 & 선형대수
  - 조건부 확률, 벡터, 선형결합 ...
- 머신러닝
  - 회귀분석의 개념
  - 머신러닝의 다양한 기법(나이브 베이즈,
- 딥러닝
  - 딥러닝의 개념
  - 딥러닝의 다양한 기법(CNN, RNN, ...)


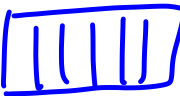
## 텍스트 마이닝 방법

- NLP(Natural Language Processing) 기본도구
  - Tokenize, stemming, lemmatize
  - Chunking
  - BOW, TFIDF – **sparse representation**
- 머신러닝(딥러닝)
  - Naïve Bayes, Logistic egression, Decision tree, SVM
  - Embedding(Word2Vec, Doc2Vec) – **dense representation**
  - RNN(LSTM), Attention, Transformer

# 텍스트 마이닝 단계



## 텍스트 마이닝 적용분야

- ① • Document classification
    - Sentiment analysis, classification
    - Document generation *생성*
    - ② Q&A ③ summarization ④ translation *각각 생성하는 것!*
    - Keyword extraction *중요한 정보 추출*
      - tagging/annotation *태깅*
    - Topic modeling *NER* 
      - LSA(Latent Semantic Analysis), LDA(Latent Dirichlet Allocation)
-   
*도출하고 하나의 공간에 표현 가능*



# 텍스트 마이닝 도구 - 파이썬 <sup>↑</sup> R

같이 보면 됨

- **NLTK**
  - 가장 많이 알려진 NLP 라이브러리 (영어)
- **Scikit Learn**
  - 머신러닝 라이브러리
  - 기본적인 NLP, 다양한 텍스트 마이닝 관련 도구 지원
- **Gensim**
  - **Word2Vec**으로 유명
  - **sklearn**과 마찬가지로 다양한 텍스트 관련 도구 지원
- **Keras**
  - **RNN, seq2seq** 등 딥러닝 위주의 라이브러리 제공
- **PyTorch** ↑ **강제학습**

## 텍스트 마이닝 기본 도구 NLP

- 목적: document, sentence 등을 (sparse) (vector)로 변환
- **Tokenize** 단어들의 sequence (정확하게, 문서의 의미)
  - 대상이 되는 문서/문장을 최소 단위로 쪼갬
- **Text normalization**
  - 최소 단위를 표준화
- **POS-tagging**
  - 최소 의미단위로 나누어진 대상에 대해 품사를 부착
- **Chunking**
  - POS-tagging의 결과를 명사구, 형용사구, 분사구 등과 같은 말모임으로 다시 합치는 과정 → 의미 이해를 위해서!
- BOW, TFIDF
  - tokenized 결과를 이용하여 문서를 vector로 표현

## Tokenize

- Tokenize

- Document를 Sentence의 집합으로 분리
- Sentence를 Word의 집합으로 분리
- 의미 없는 문자 등을 걸러 냄

- 영어 vs. 한글

- 영어는 공백(space) 기준으로 비교적 쉽게 tokenize 가능
- 한글은 구조상 형태소(morpheme) 분석이 필요
  - 복합명사, 조사, 어미 등을 분리해내는 작업이 필요
  - 영어에 비해 어렵고 정확도 낮음

EX)

go → went  
↙

## Text Normalization

- 동일한 의미의 단어가 다른 형태를 갖는 것을 보완
  - 다른 형태의 단어들을 통일시켜 표준단어로 변환
- **Stemming** (어간 추출)
  - 단수 - 복수, 현재형 - 미래형 등 단어의 다양한 변형을 하나로 통일
  - 의미가 아닌 규칙에 의한 변환
  - 영어의 경우, Porter stemmer, Lancaster stemmer 등이 유명
- **Lemmatization** (표제어 추출)
  - 사전을 이용하여 단어의 원형을 추출
  - 품사(part-of-speech)를 고려
  - 영어의 경우, 유명한 어휘 데이터베이스인 WordNet을 이용한 WordNet lemmatizer가 많이 쓰임

## Stemming 예제 – Porter Stemmer

- 원문
  - "This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things"
- tokenize 결과
  - 'This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate', 'copy', ',', 'complete', 'in', 'all', 'things'
- stemming 결과
  - 'thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi', ',', 'complet', 'in', 'all', 'thing'
- 문제점
  - this → thi, was → wa, accurate → accur 등 이상한 단어로 변환
  - 이는 사전이 아닌 알고리즘(규칙)에 의해 변환하기 때문

## Lemmatization 예제 - WordNetLemmatizer

- 원문

- "This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things"

- tokenize 결과

- 'This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', '"s"', 'chest', ',', 'but', 'an', 'accurate', 'copy', ',', 'complete', 'in', 'all', 'things'

필수

- 결과

- 'This', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', '"s"', 'chest', ',', 'but', 'an', 'accurate', 'copy', ',', 'complete', 'in', 'all', 'thing'
- was는 품사를 모르기 때문에 형태적으로 접근
- things는 사전에 근거하여 thing으로 변환
- 나머지 단어는 사전의 원형을 유지

## POS-tagging

- 토큰화와 정규화 작업을 통해 나누어진 형태소(의미를 가지는 최소단위)에 대해 품사를 결정하여 할당하는 작업
- 동일한 단어라도 문맥에 따라 의미가 달라지므로 품사를 알기 위해서는 문맥을 파악해야 함
- Text-to-speech(텍스트를 읽어주는 시스템)에서도 각 단어에 대해 올바른 발음을 하기 위해 품사 태깅을 이용
- POS Tagging은 형태소 분석으로 번역되기도 하는데, 형태소 분석은 주어진 텍스트(원시말뭉치)를 형태소 단위로 나누는 작업을 포함하므로 앞의 토큰화, 정규화 작업에 품사 태깅을 포함한 것으로 보는 것이 타당


## 한글 형태소 분석기 예제

- 한글 형태소 분석기 도구 - 파이썬
  - <http://konlpy.org/ko/v0.4.3/>
- 토큰화, 정규화, POS-tagging이 모두 이루어짐
- 예시 *→ 한문자씩 새겨서 볼 필요가 있음*
  - 대상문장: 주택 문제의 경우 제 나이가 아직 젊으니까 가능성이 많지요.

주택	주택/NNG
문제의	문제/NNG+의/JKG
경우	경우/NNG
제	저/NP+의/JKG
나이가	나이/NNG+가/JKS
아직	아직/MAG
젊으니까	젊/VN+으니까/EC
가능성이	가능/XR+성/XSN+이/JKS
많지요	많/VN+지요/EC



## Chunking

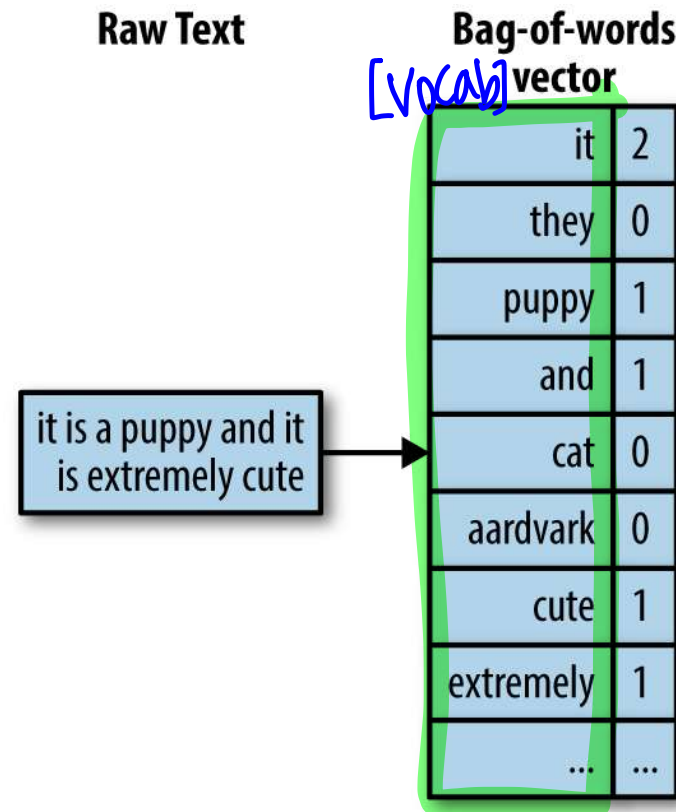
- Chunk는 언어학적으로 말모임을 뜻하며, 명사구, 형용사구, 분사구 등과 같이 주어와 동사가 없는 두 단어 이상의 집합인 구(phrase)를 의미
- Chunking은 주어진 텍스트에서 이와 같은 chunk를 찾는 과정
- 즉, 형태소 분석의 결과인 각 형태소들을 서로 겹치지 않으면서 의미가 있는 구로 묶어나가는 과정임
- 텍스트로부터  Information Extraction(정보추출)을 하기 위한 전단계로 보거나 혹은 Information Extraction에 포함되기도 함

## 개체명 인식(Named Entity Recognition, NER)

- 개체명(Named Entity)은(기관, (단체), (사람) (날짜) 등과 같이 특정 정보에 해당하는 명사구를 의미
- 따라서 NER은 텍스트로부터 뭔가 의미 있는 정보를 추출하기 위한 방법으로 사용
- 예를 들어 "James is working at Disney in London"이라는 문장이 있을 때, (James는 PERSON), (Disney는 ORGANIZATION), (London은 GPE(지리적인 장소)임을 알아내는 작업.
- 관계 인식 (Relation Extraction)
  - NER에 의해 추출된 개체명들을 대상으로 그들 간의 관계를 추출하는 작업
  - 특정 건물이 특정 장소에 위치하는 관계와 같은 지식을 텍스트로부터 추출할 때 사용
  - 위 NER 예에서 in을 이용해 (Disney)는 (London)에 위치한다는 관계를 추출  
이 관계도 추출 가능!

\*  **BOW (Bag of Words)**  
 vector      '백지로 변환'      '단어가 없는 단어가 있다면 0으로 표시한다.'

- Vector Space Model
  - 문서를 bag of words 로 표현
  - 단어가 쓰여진 순서는 무시
  - 모든 문서에 한번 이상 나타난 단어들에 대해 유(1)/무(0)로 문서를 표현
- count vector *단어가 몇번 나왔는지!*
  - 단어의 유/무 대신 단어가 문서에 나타난 횟수로 표현
  - count가 weight로 작용



## TFIDF(Term Frequency - Inverse Document Frequency)

- count vector의 문제점  $\ominus$ 
  - 많은 문서에 공통적으로 나타난 단어는 중요성이 떨어지는 단어일 가능성이 높음 ex) the, a, ...
- TFIDF
  - 단어의 count를 단어가 나타난 문서의 수로 나눠서 자주 등장하지 않는 단어의 weight를 올림
  - $tf(d, t)$ : 문서 d에 단어 t가 나타난 횟수, count vector와 동일, 로그스케일 등 다양한 변형이 있음
  - $df(t)$ : 전체 문서 중에서 단어 t를 포함하는 문서의 수
  - $idf(t)$ :  $df(t)$ 의 역수를 바로 쓸 수도 있으나, 여러가지 이유로 로그스케일과 스무딩을 적용한 공식을 사용,  
 $\log(n/(1+df(t)))$ , n은 전체 문서 수  
df가 0인 경우가 발생할 수 있음  
→  $1/df(t)$ 와 같음

## TFIDF 계산 예제

- tokenize 결과 예제

- d1: fast, furious, shoot, shoot
- d2: fast, fast, fly, furious

- count vector:  $tf(d, t)$

word	fast	fly	furious	shoot
d1	1	0	1	2
d2	2	1	1	0

- idf:  $\log(n/(1+df(t)))$

- fast, furious는 두 개의 문서에, fly, shoot는 하나의 문서에 출현
- 따라서 fast, furious는 중요도가 fly, shoot보다 떨어져야 함
- $n(\text{전체문서수}) = 10$ 으로 가정하면, fast의 idf는  $\log(10/(1+2)) = 0.52$
- fly의 idf는  $\log(10/(1+1)) = 0.70$

- TFIDF:  $tf(d, t) * idf(t)$

- 가중치 변화를 확인

word	fast	fly	furious	shoot
d1	$1*0.52$	$1*0.70$	$0*0.52$	$0*0.70$
d2	$2*0.52$	$0*0.70$	$1*0.52$	$1*0.70$

## TFIDF의 변형

- 다양한 변형이 있음

Variants of term frequency (tf) weight

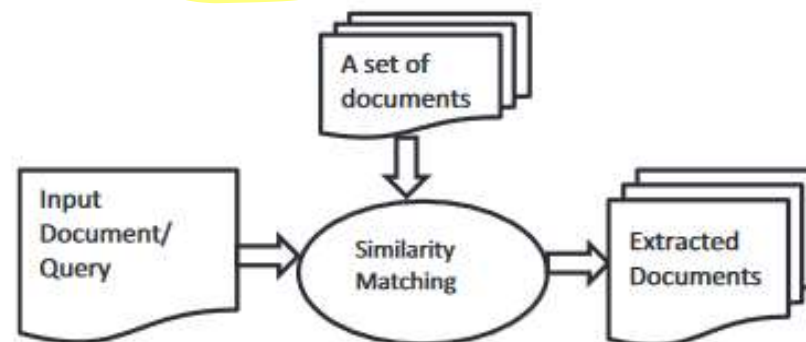
weighting scheme	tf weight
binary	0, 1
raw count	$f_{t,d}$
term frequency	$f_{t,d} / \sum_{t' \in d} f_{t',d}$
log normalization	$\log(1 + f_{t,d})$
double normalization 0.5	$0.5 + 0.5 \cdot \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$
double normalization K	$K + (1 - K) \frac{f_{t,d}}{\max_{t' \in d} f_{t',d}}$

Variants of inverse document frequency (idf) weight

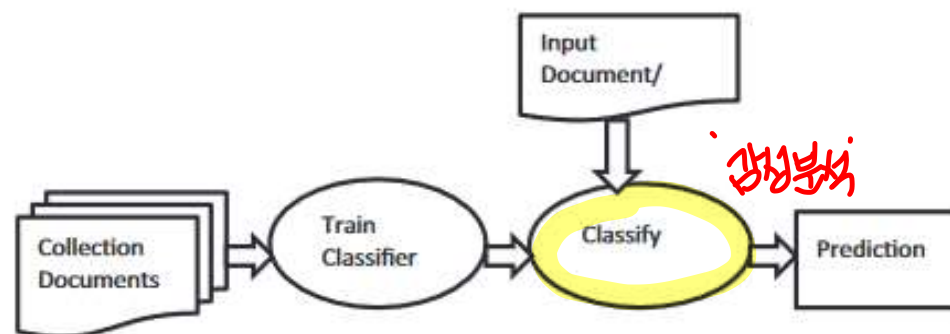
weighting scheme	idf weight ( $n_t =  \{d \in D : t \in d\} $ )
unary	1
inverse document frequency	$\log \frac{N}{n_t} = -\log \frac{n_t}{N}$
inverse document frequency smooth	$\log \left( \frac{N}{1 + n_t} \right) + 1$
inverse document frequency max	$\log \left( \frac{\max_{t' \in d} n_{t'}}{1 + n_t} \right)$
probabilistic inverse document frequency	$\log \frac{N - n_t}{n_t}$

(크기 불변?) [Fixed length vector]   
 [크기 일정한 벡터로]   
 [한편으로는 크기가 너무 큼.]

**BOW**의 활용



**Figure 2** Retrieving matching documents (Weiss et al., 2010).



**Figure 3** Classification and prediction.

Khan, Khairullah, et al. "Mining opinion components from unstructured reviews: A review." Journal of King Saud University-Computer and Information Sciences 26.3 (2014): 258-275.

## TFIDF를 이용한 유사도 계산

- TFIDF Matching score
  - TFIDF vector의 내적을 이용

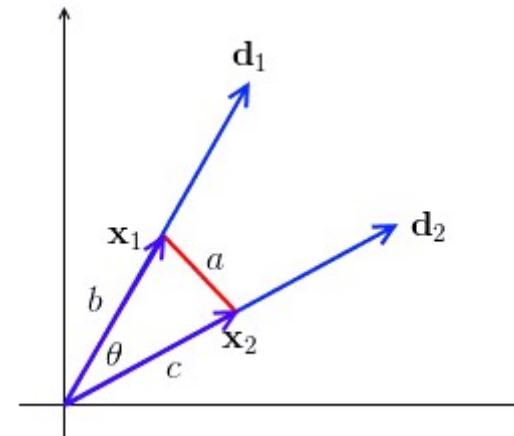
Document ID	Cat	Dog	Mouse	Fish	Horse	Cow	Matching Scores
d1	0.397	0.397	0.000	0.475	0.000	0.000	1.268
d2	0.352	0.301	0.680	0.000	0.000	0.000	0.653
d3	0.301	0.363	0.000	0.000	0.669	0.741	0.664
d4	0.376	0.352	0.636	0.558	0.000	0.000	1.286
d5	0.301	0.301	0.000	0.426	0.544	0.544	1.028

- Cosine Similarity ?
  - vector의 방향에 대한 유사도

두개의 방향성만!

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \|\vec{b}\| \cos \theta$$

$$\cos \theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$





## Text Classification with BOW/TFIDF

- Naïve Bayes
- Logistic regression
  - Ridge regression
  - Lasso regression
- Decision tree
  - Random Forest

\* 단어가 너무 많잖아? oh my god!

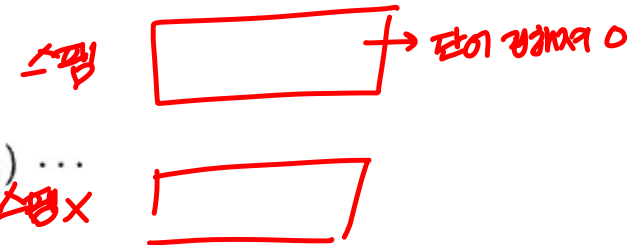
## Naïve Bayes

- Wikipedia: a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies (count vector) as the features.
- $(x_1, \dots, x_n)$  의 단어집합으로 이루어진 문서가 분류  $c_k$ 에 속할 확률

$$\begin{aligned}
 p(C_k | x_1, \dots, x_n) &\propto p(C_k, x_1, \dots, x_n) \\
 &\approx p(C_k) p(x_1 | C_k) p(x_2 | C_k) p(x_3 | C_k) \dots \\
 &= p(C_k) \prod_{i=1}^n p(x_i | C_k), \quad 100\% / 10 \text{ 스팸}
 \end{aligned}$$

(Handwritten notes: 이항의 정리, 스팸일 확률, 스팸이냐 아니냐, 스팸이냐 아니냐 이 양자 다 양자할 때도)

- 위 확률이 가장 큰  $c_k$ 로 분류



## Naïve Bayes 예제

- 아래와 같은 상황에서 “fun, furious, fast” 문서는 어디로 분류가 될까?

movie	단어(Word)	분류
1	fun, couple, love, love	Comedy
2	fast, furious, shoot	Action
3	Couple, fly, fast, fun, fun	Comedy
4	Furious, shoot, shoot, fun	Action
5	Fly, fast, shoot, love	Action

- 구하고자 하는 확률
  - 문서가 comedy일 확률:  $p(\text{comedy} | \text{fun, furious, fast})$
  - (fun, furious, fast)를 words라고 줄여서 쓴다면,
  - $p(\text{comedy} | \text{words})$ ,  $p(\text{action} | \text{words})$ 을 구해서 확률이 더 큰 쪽으로 결정!

출처: <http://bcho.tistory.com/m/1010>

읽어보기!

## 풀이 과정

- $p(\text{comedy} | \text{words}) = p(\text{words} | \text{comedy}) * p(\text{comedy}) / p(\text{words})$
- $p(\text{action} | \text{words}) = p(\text{words} | \text{action}) * p(\text{action}) / p(\text{words})$
- 대소비교만 하면 되므로 아래와 같이 수정
  - $p(\text{comedy} | \text{words}) \sim p(\text{words} | \text{comedy}) * p(\text{comedy})$
  - $p(\text{action} | \text{words}) \sim p(\text{words} | \text{action}) * p(\text{action})$
- 각 단어의 빈도수
  - $\text{Count}(\text{fast}, \text{comedy}) = 1$  (코메디 중, fast 라는 단어가 나오는 횟수)
  - $\text{Count}(\text{furious}, \text{comedy}) = 0$ ,  $\text{Count}(\text{fun}, \text{comedy}) = 3$
  - $\text{Count}(\text{fast}, \text{action}) = 2$ ,  $\text{Count}(\text{furious}, \text{action}) = 2$
  - $\text{Count}(\text{fun}, \text{action}) = 1$
- 단어와 상관 없는 확률:  $p(\text{comedy}) = 2/5$ ,  $p(\text{action}) = 3/5$
- $p(\text{words} | \text{comedy}) = p(\text{fast} | \text{comedy}) * p(\text{furious} | \text{comedy}) * p(\text{fun} | \text{comedy}) = 1/9 * 0/9 * 3/9$
- $p(\text{comedy} | \text{words}) = 0$
- $p(\text{words} | \text{action}) = p(\text{fast} | \text{action}) * p(\text{furious} | \text{action}) * p(\text{fun} | \text{action}) = 2/11 * 2/11 * 1/11$
- $p(\text{action} | \text{words}) = 3/5 * 4/1331$

## Logistic Regression

- 분류를 위한 회귀분석
  - 종속 변수와 독립 변수간의 관계를 구체적인 함수로 나타내어 향후 예측 모델에 사용
  - 종속 변수가 범주형 데이터를 대상으로 하며, 입력 데이터가 주어졌을 때 해당 데이터의 결과가 특정 분류로 나뉘기 때문에 일종의 분류 (classification) 기법에 해당
- 텍스트 마이닝에서의 문제
  - 추정해야 할 계수가 vector의 크기(단어의 수)만큼 존재하므로, 과적합이 발생하기 쉽고 많은 데이터 셋이 필요
  - 그럼에도 불구하고 잘 작동하는 편
  - 정규화(regulation)을 이용해 과적합 해결 노력

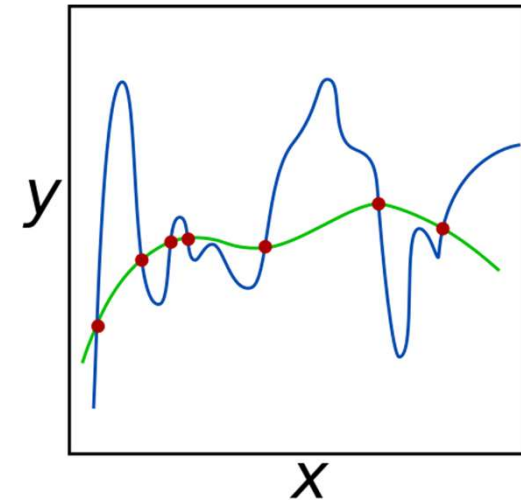
사용된 단어 10000 (특정문단)

최소값 요구하는 데이터 1000/보다 작을  
최소

데이터가 필요!

## Ridge and Lasso Regression

- 릿지 회귀 (Ridge regression)
  - 목적함수에 추정할 계수(parameter)에 대한 **L2 norm**(규제항)을 추가하여 모형의 **과적합을 방지**
- 라쏘 회귀 (Lasso regression)
  - **L1 norm**을 규제항으로 사용함으로써 0에 가까운 계수를 0으로 만들어 **영향을 거의 미치지 않는 단어들을 제외**
  - 남은 단어들로 분류의 이유에 대해 설명이 가능하다는 장점이 있음
  - feature selection의 효과가 있음

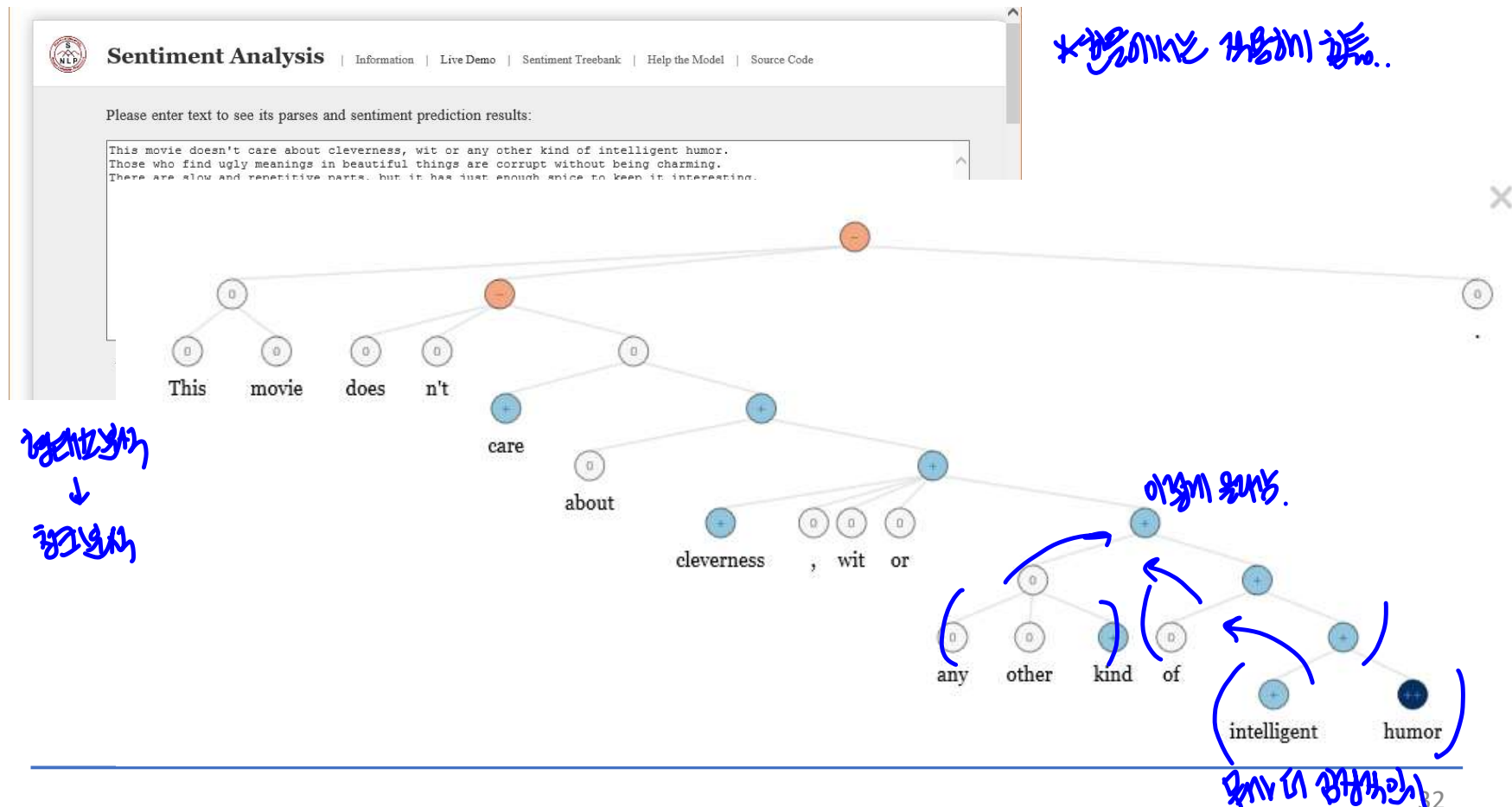


## 문서분류의 활용 - Sentiment Analysis(감성분석)

- 네이버 지식백과
  - 소비자의 감성과 관련된 텍스트 정보를 자동으로 추출하는 텍스트 마이닝(Text Mining) 기술의 한 영역. 문서를 작성한 사람의 감정을 추출해 내는 기술로 문서의 주제보다 어떠한 감정을 가지고 있는가를 판단하여 분석한다.
- Wikipedia – 보다 포괄적
  - **Sentiment analysis** (sometimes known as **opinion mining** or **emotion AI**) refers to the use of [natural language processing](#), [text analysis](#), [computational linguistics](#), and [biometrics](#) to systematically identify, extract, quantify, and study **affective states** and subjective information.

## Demonstration: Sentence-level Sentiment

- <http://nlp.stanford.edu:8080/sentiment/rntnDemo.html>





## 한글 감성분석 예제

- 학습을 위한 데이터 예
  - label이 0이면 부정, 1이면 긍정

리뷰	Label
아 더빙... 진짜 짜증나네요 목소리	0
교도소 이야기구먼.. 솔직히 재미는 없다..	0
너무 재밌었다그래서보는것을추천한다	1

- 학습 방법
  - 리뷰를 BOW로 변환 후 input으로 쓰고, label을 target으로 하여 학습
  - 나이브베이지, 로지스틱 회귀분석, SVM 등 다양한 방법의 사용이 가능
  - 새로운 리뷰에 대해 긍정/부정을 예측

# Curse of Dimensionality

(BOW)

사용된 단어들

↓  
단어 사용된 횟수

↓  
300개...

↓  
문서 전체에 사용된 단어의 수

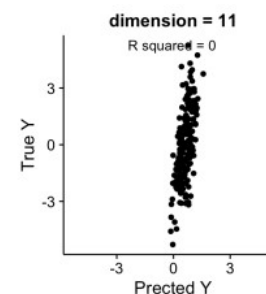
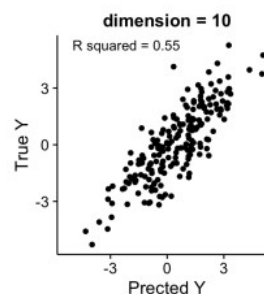
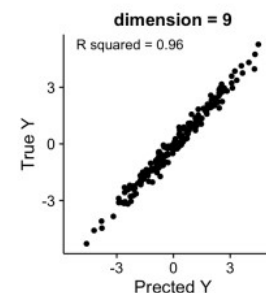
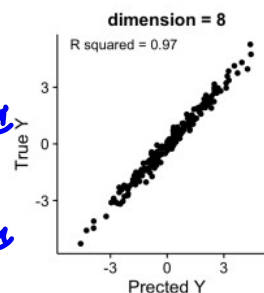
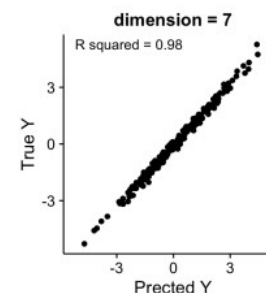
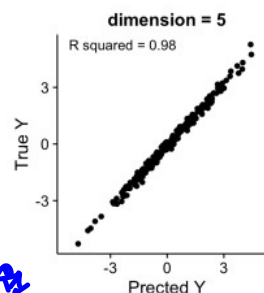
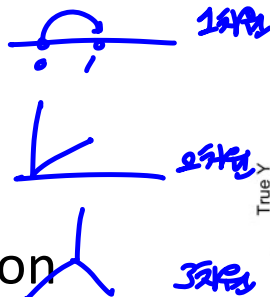
∴ 거의 다 0이므로 해결방법

## 차원의 저주

- Extremely sparse data
- 각 데이터 간의 거리가 너무 멀게 위치

해결방법

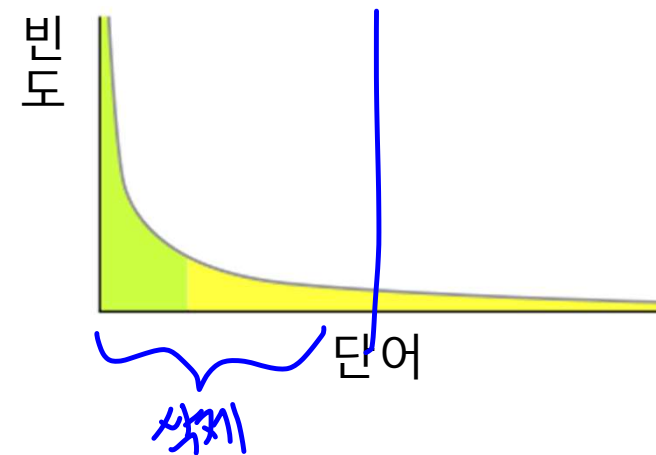
- 더 많은 데이터
- Dimension reduction (차원 축소 방법)
  - feature selection
  - feature extraction



<http://thesciencelife.com/archives/1001>

## 단어 빈도의 불균형

- Zipf's law(멱법칙)
  - 극히 소수의 데이터가 결정적인 영향을 미치게 됨
- 해결방안 *→ 2중의 언어를 쓰이는 것 ..*
  - feature selection
    - 빈도 높은 단어를 삭제
    - 심한 경우 50% 삭제
  - Boolean BOW 사용 *(유익)*
    - 1이상이면 1로 변환
  - log 등의 함수를 이용해 weight를 변경



60W의 가장 큰 문제  
단어가 쓰인 순서정보의 손실

- 통계에 의한 의미 파악 vs. 순서에 의한 의미 파악
- Loss of sequence information
  - 단어들의 순서 – context가 중요
  - 특히 번역과 같은 sequence-to-sequence 문제에서 매우 중요
- 해결방안
  - n-gram
    - 부분적 해결, 주로 classification 문제에서 유용
  - Deep learning
    - RNN, Attention, Transformer, BERT

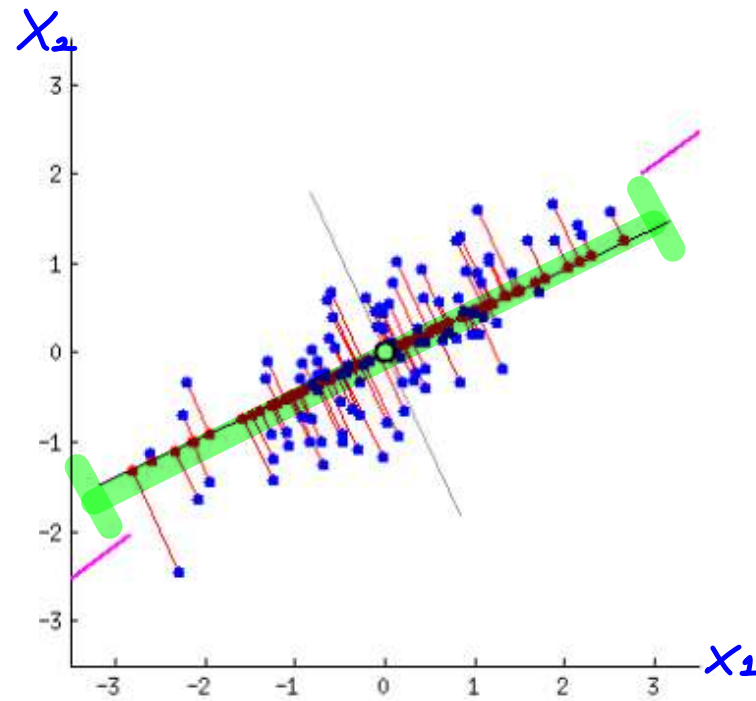
## Dimensionality Reduction (차원 축소)

- 차원의 저주를 해결하기 위한 노력
- Feature selection *한번에 걸러내기 Ex) 100개만*
  - Manual, Regularization(Lasso)
- Feature extraction *만개 → 정보량 → 100개 차원*
  - PCA, LSA(SVD)
- Embedding
  - Word embedding, Document embedding → *0.4차원*
- Deep Learning
  - RBM, Autoencoder

## Feature Extraction

- feature extraction
  - Wiki: feature extraction starts from an initial set of measured data and builds derived values (features) intended to be informative and non-redundant
- PCA
  - 데이터의 분산을 최대한 보존하는 새로운 축을 찾아 변환함으로써 차원을 축소

0.7을 일차원으로 바꿀 수 있다. → 분산 ↑ (3차원 밑에 놓아도 됨)



<https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues>

## 주성분 분석 (Principal Component Analysis)

- 선형결합 (linear combination)
  - $X$  (데이터,  $n \times p$ ):  $n$ (sample의 수),  $p$ (변수의 수)
  - $X_i$ :  $i$ 번째 feature에 대한 크기  $n$ 의 vector
  - $Z$ ( $X$ 의 선형결합으로 이루어진 새로운 변수,  $n \times p$ ) =  $XV^T$
  - $V^T$ : 새로운 축 ( $p \times p$ )  $\rightarrow$  공분산행렬의 고유벡터로 구성
- 공분산행렬
$$\frac{X^T X}{n} = \frac{1}{n} \begin{pmatrix} \text{dot}(X_1, X_1) & \text{dot}(X_1, X_2) & \cdots & \text{dot}(X_1, X_d) \\ \text{dot}(X_2, X_1) & \text{dot}(X_2, X_2) & \cdots & \text{dot}(X_2, X_d) \\ \vdots & \vdots & \ddots & \vdots \\ \text{dot}(X_d, X_1) & \text{dot}(X_d, X_2) & \cdots & \text{dot}(X_d, X_d) \end{pmatrix}$$
- 고유값이 큰 순서대로 고유벡터를 정렬하여 차원 선택
- 선택된 고유벡터와  $X$ 의 선형결합으로 차원 축소

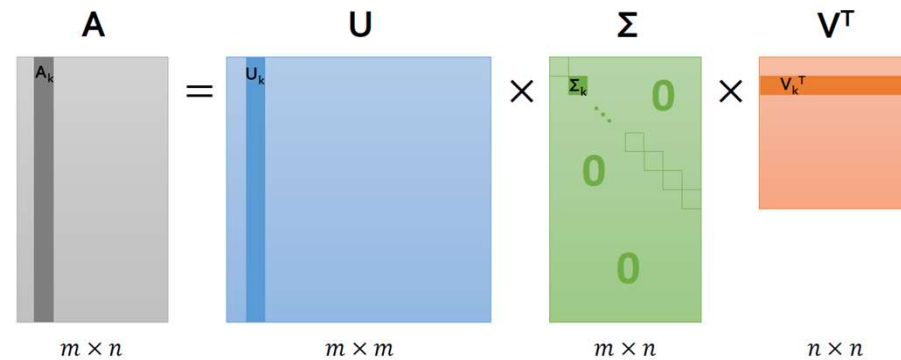
## LSA(Latent Semantic Analysis)

- a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by **producing a set of concepts related to the documents and terms**. LSA assumes that *words that are close in meaning will occur in similar pieces of text* (the distributional hypothesis)
- SVD(Singular Vector Decomposition)
  - 주어진 dtm(document term matrix, A)을  $A=U\Sigma V^T$ 의 형태로 분해
  - document 수:m, term 수: n
  - U, V는 각각 열벡터가 singular vector로, 서로 직교하는 성질을 가짐.  $U^T U = I, V^T V = I$
  - $\Sigma$ 는 대각행렬로 제공하면 PCA에서의 고유값이 됨



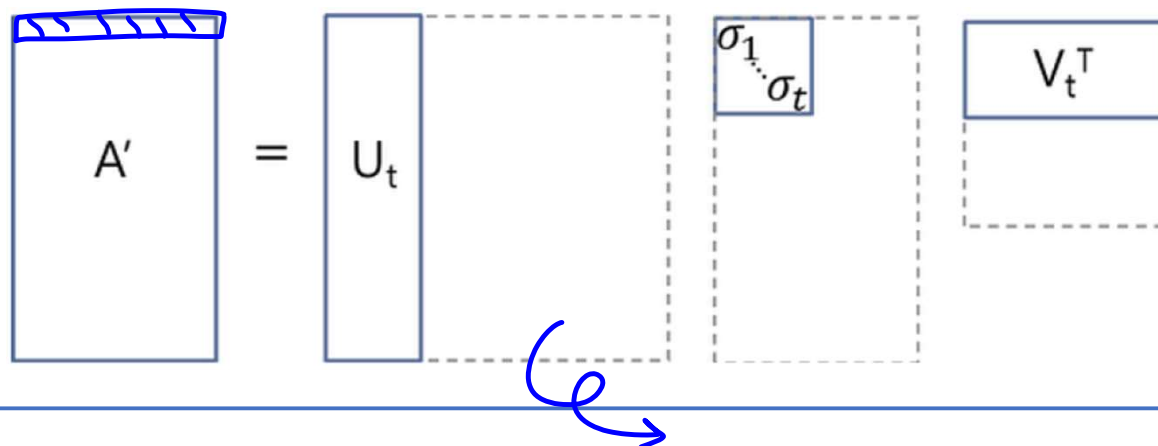
# 특이값 분해 (singular vector decomposition)

- $A = U \Sigma V^T$



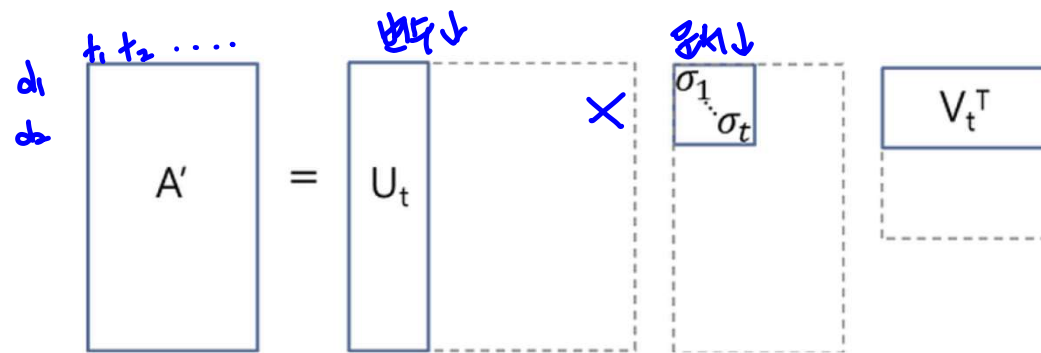
- Truncated SVD

- $\Sigma$  행렬의 대각원소(특이값) 가운데 상위  $t$ 개만 골라냄
- $t$ 는 잠재의미군 중 비중이 큰 것부터 차례대로 골라낸 것
- $A'$ 은 축소된  $U, \Sigma, V$ 로 복원한 값으로  $A$ 에 근사한 값



## 잠재의미분석

- $U_t(m, t)$ ,  $\Sigma(t, t)$ ,  $V_t^T(t, n)$ 
  - $m$ : sample의 수,  $n$ : 변수의 수,  $t$ : 축소한 잠재의미의 차원
  - $V_t$ : 원래  $A$ 의 공분산행렬의 고유벡터로 이루어진 행렬
- $U_t \Sigma(t, t)$ 
  - 각 sample에 대해서  $n \rightarrow t$ 로 변수가 줄어든 data set
  - 각 문서에서 잠재의미 별 비중(?)
- $\Sigma V_t^T(t, n)$ 
  - 각 변수에 대해서 sample의 차원을  $t$ 로 줄임
  - 각 단어에서 잠재의미 별 비중(?)



## 잠재의미분석의 활용

- 문서 간의 유사도

- Count vector나 TFIDF에 cosine similarity를 직접 적용하는 경우, 물리적인 단어로만 유사도를 측정하게 됨
- 잠재의미분석을 활용하면 직접적인 단어가 아니라 의미적으로 유사한(문서에서 함께 많이 등장한) 단어들로 유사도를 측정하는 것이 가능할 것으로 기대

- 단어 간의 유사도

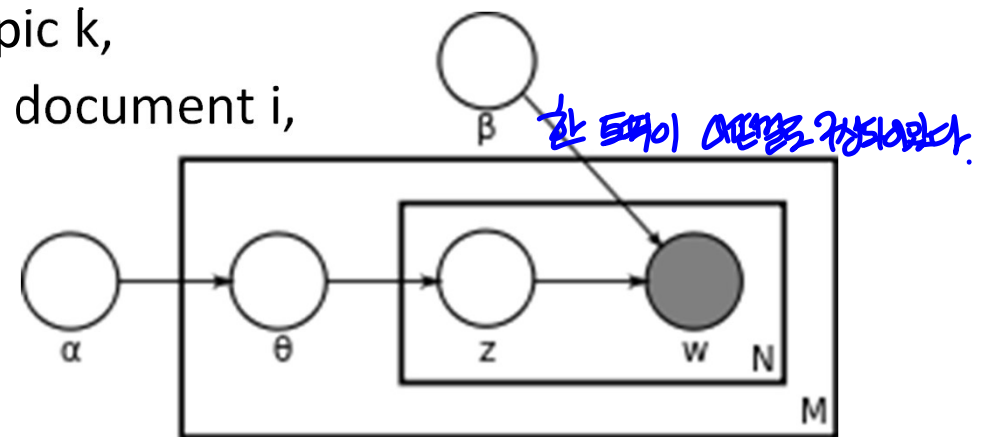
- 마찬가지로 주어진 문서 집합에서 단어들이 어떤 유사도를 가지는 지 볼 수 있음

## Topic Modeling

- Topic models
  - Documents are mixtures of topics 문서가 포함하는 방법 중 하나.
  - A topic is a probability distribution over words 하나의 토픽에 대한 단어
  - A topic model is a generative model for documents 각각의 토픽들이 어떤 단어를 생성 하였을까?
  - Different documents can be produced by picking words from a topic depending on the weight given to the topic
  - Infer the set of topics that were responsible for generating a collection of documents
  - The goal is to find the best set of latent variables that can explain the observed data(observed words in documents)
- Determining the number of topics
  - Too many topics: Uninterpretable topics
  - Too few topics: Very broad topics

## Latent Dirichlet Allocation

- [https://en.wikipedia.org/wiki/Latent\\_Dirichlet\\_allocation](https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation)
  - $\alpha$  is the parameter of the Dirichlet prior on the per-document topic distributions,
  - $\beta$  is the parameter of the Dirichlet prior on the per-topic word distribution,
  - $\theta_i$  is the topic distribution for document  $i$ ,
  - $\varphi_k$  is the word distribution for topic  $k$ ,
  - $z_{ij}$  is the topic for the  $j$ th word in document  $i$ ,
  - $w_{ij}$  is the specific word.



## Topic Modeling의 원리

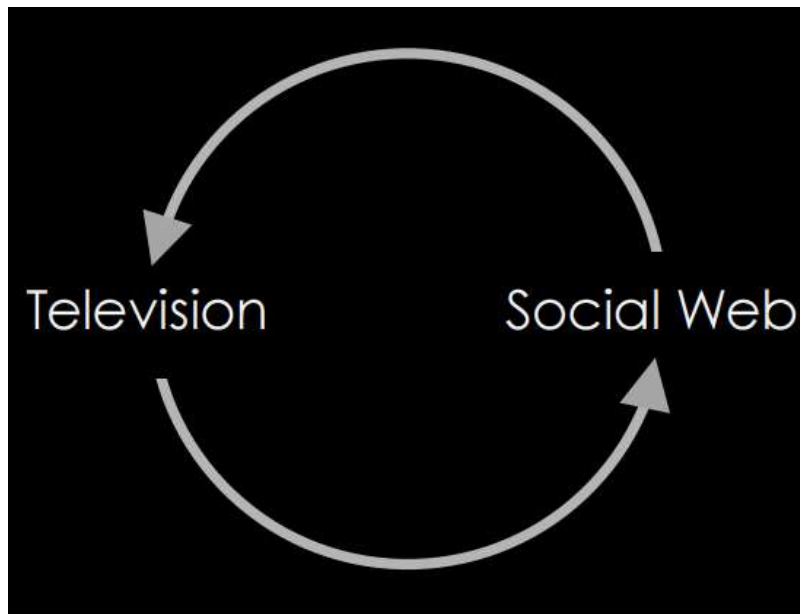
- 토픽은 주제를 의미하는 용어로 사용되며, 각 문서들이 특정한 주제에 속할 확률분포와 주제로부터 특정 단어들이 파생되어 나올 확률분포가 주어졌을 때, 이 두 확률분포를 조합하여 각 문서들에 들어가는 단어들의 확률분포를 계산
- $\theta$ : 문서들이 각 주제들에 속할 확률분포
  - 디리클레분포의 매개변수인 <알파>에 의해 결정
- $N$ : 특정 문서에 속한 단어의 집합
- $M$ : 전체 문서의 집합
- $z$ : 문서 내의 단어들이 주제들에 속할 확률분포
  - $\theta$ 에 의한 다항분포로 선택
- $\beta$ : 각 주제가 특정 단어를 생성할 확률을 나타내는 확률분포
- 결국  $z$ 와  $\beta$ 에 의해 실제 문서들의 단어분포인  $w$ 가 결정
- $w$ 만이 실제로 문서들을 통해 주어진 분포이고 나머지는 모두 잠재변수
- LDA 알고리즘에서는 주어진 문서와 토픽들의 사전확률 분포인  $\alpha$ 와 토픽 내에서 단어의 사전확률분포인  $\beta$ 의 파라미터 값을 활용해 반복적인 시뮬레이션을 통해  $z$ 와  $\theta$ 를 추정

## Topic Modeling 활용사례

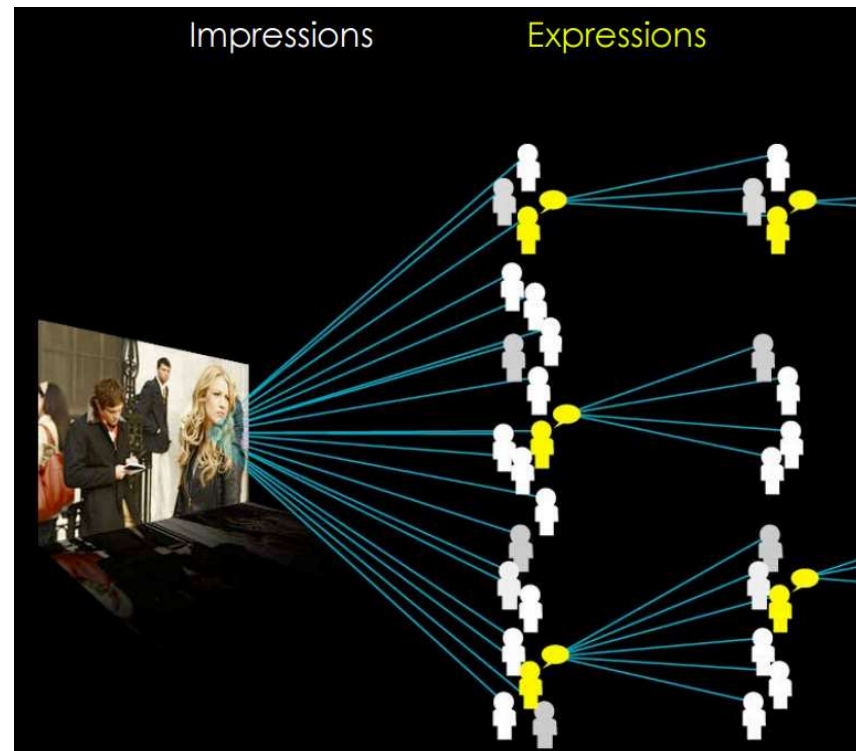
드라마 시청률 변화



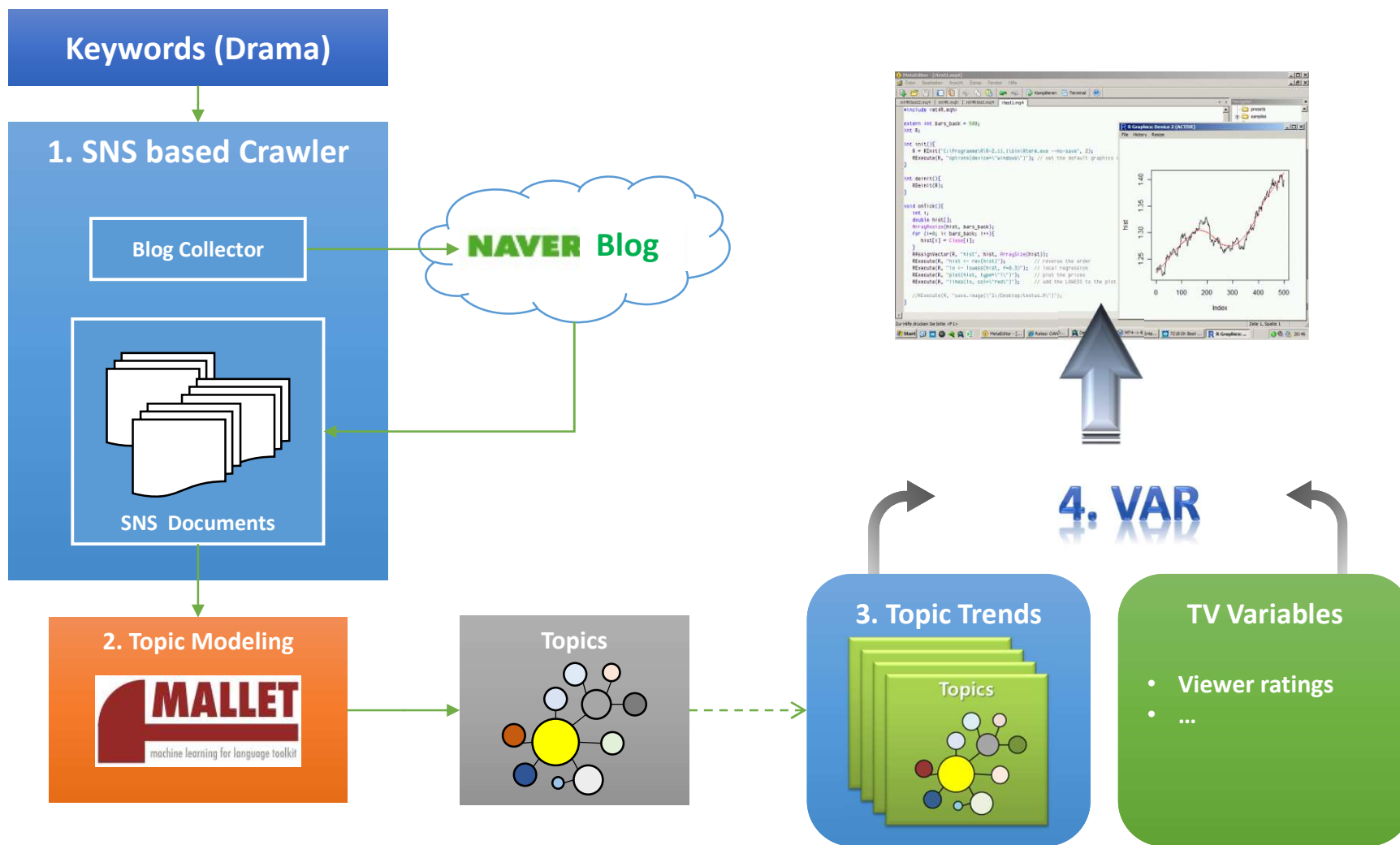
소셜미디어 토픽 변화



Deb Roy (2012)



## Analysis Framework





## Example of Topics – “My Love from the Star”

Topic Title	Topic Num.	Weight	Topic
Food	0	0.13184	Food, price, cheese, strawberry, pizza, ice coffee, sweet potato, cafe ...
Fashion	8	0.12519	Overcoat, price, style, sunglasses, dress, jacket, fashion, skirt, shirt, ...
Story, Roles	1	0.14532	Songee, Dominjun, Semi, Jaekyong, alien, star, epilogue, kiss, supernatural power, ...
Drama – General talk	6	0.18174	Drama, viewer rating, casting, review, actress, The heirs, The thieves, acting ability ...
Daily Life	9	0.20568	You, today, mobile, thought, photo, love, We, human, sister, time, one day, friend, now...

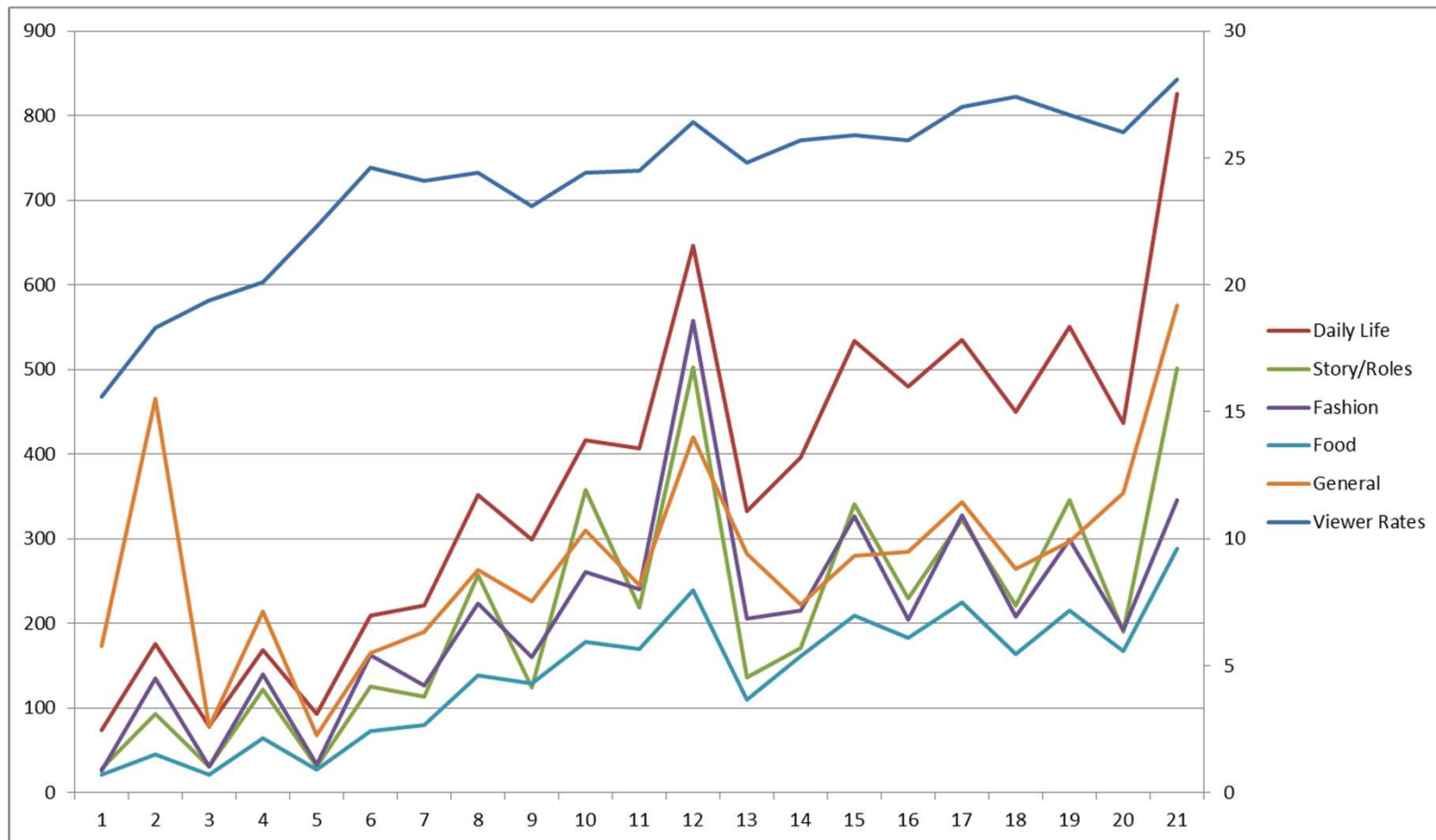
## Topic Trends

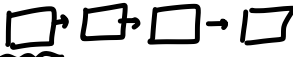
$$\varphi^{(t)} = \frac{\sum_{d=1}^{D_t} \theta^{(d)}}{D_t}$$

$\varphi^{(t)}$ : 날짜 t의 토픽 분포  
 $D_t$ : 날짜 t의 총 문서 (수)  
 $\theta^{(d)}$ : 문서 d의 토픽 분포

- 즉, 당일에 작성된 문서들을 대상으로 하여 각 토픽에 대해 평균값을 계산

## Topic Trends: “My Love from the Star”




Text Mining   
 Text → (fixed length vector) → (비선형) 순서 정보 ~~포함~~

문제의 해결 방안

## Word Embedding

↳ 텍스트내 단어들을 컴퓨터에서 어떻게 다룰까?

- 단어에 대한 vector의 dimension reduction이 목표
- 단어의 표현
  - Term-Document Matrix에서 Document 별 count vector
    - 일반화가 어려움
  - one-hot-encoding: extremely sparse ?
- Word embedding:
  - (one-hot-encoding)으로 표현된 단어를 (dense vector)로 변환
  - 변환된 vector를 이용하여 학습
  - 최종목적에 맞게 학습에 의해 vector가 결정됨 ?
  - 학습목적 관점에서의 단어의 의미를 내포  인공적 의미 부여

## One hot encoding

- 각 단어를 모든 문서에서 사용된 단어들의 수 길이의 벡터로 표현, 심한 경우 길이 30만의 벡터 중에서 하나만 1인 sparse vector가 됨

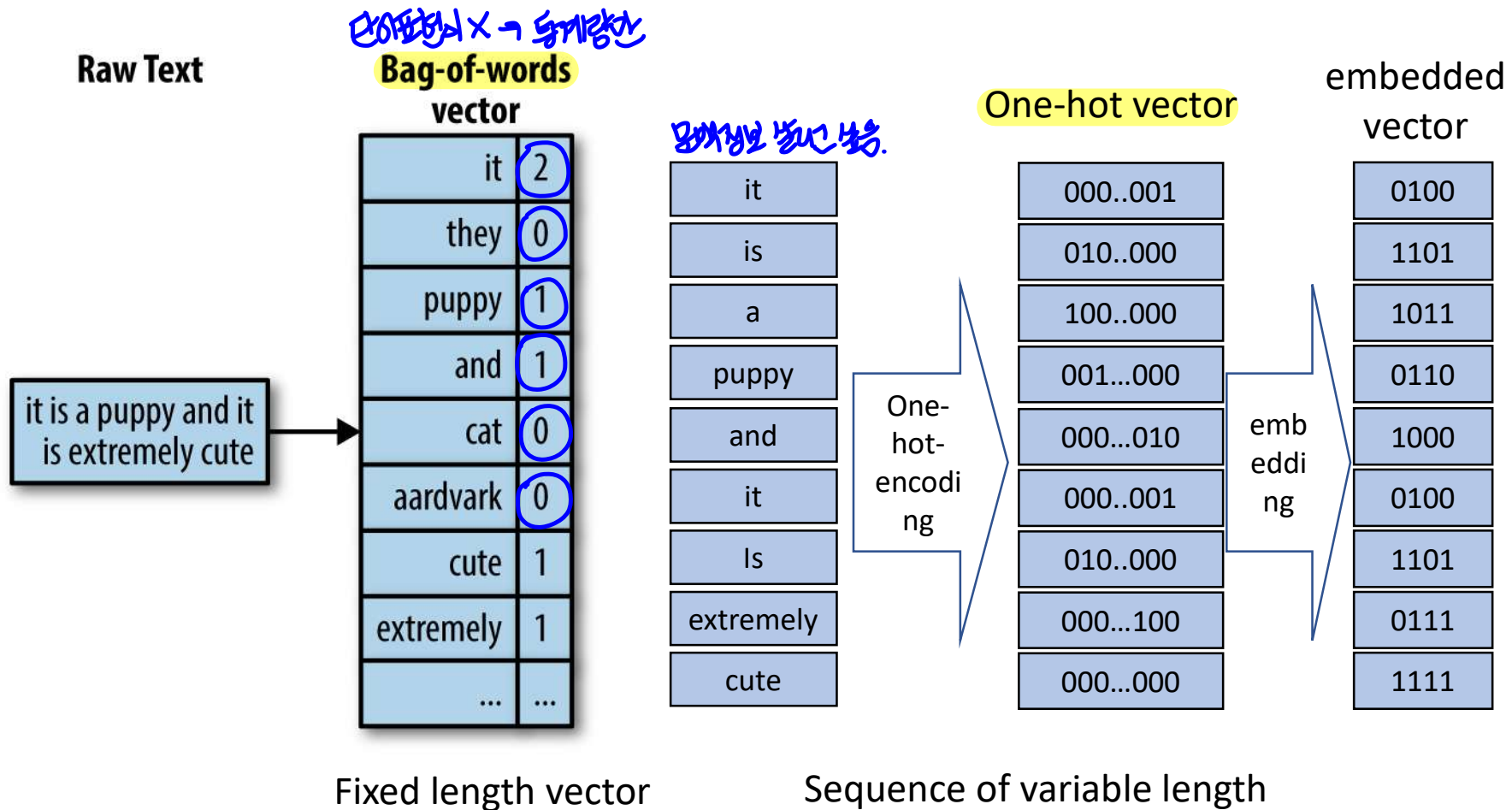
Rome = [1, 0, 0, 0, 0, 0, ..., 0]

Paris = [0, 1, 0, 0, 0, 0, ..., 0]

Italy = [0, 0, 1, 0, 0, 0, ..., 0]

France = [0, 0, 0, 1, 0, 0, ..., 0]

## BOW vs. Word Embedding



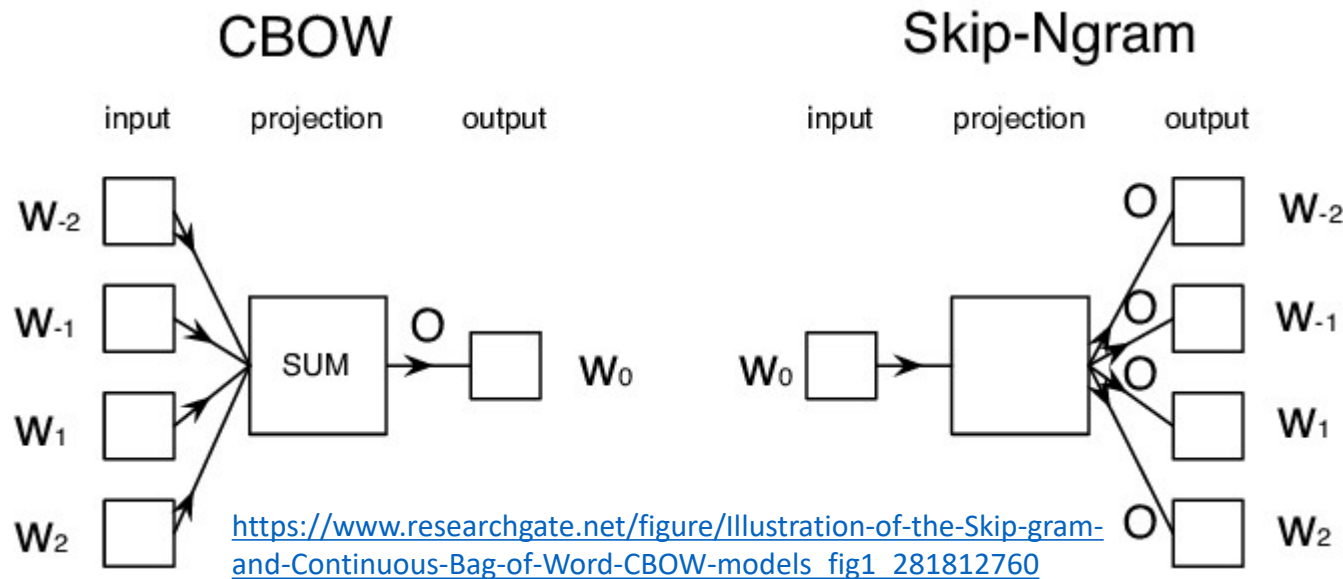
## Word Embedding을 이용한 문서 분류

변경성  
개념이 고정 X

- BOW와는 다른 관점의 문서 표현
  - document: 제한된 maxlen 개의 word sequence (앞이나 뒤를 잘라냄)
  - word: one-hot-vector에서 저차원(reduced\_dim)으로 embedding된 dense vector
  - 즉 document는 (maxlen, reduced\_dim)의 2차원 행렬로 표현
- 단순한 분류모형 (sequence 무시)
  - (maxlen, reduced\_dim) 차원의 document를 maxlen\*reduced\_dim 차원으로 펼쳐서 분류모형에 적용

## Word2Vec

- 문장에 나타난 단어들의 순서를 이용해 word embedding 을 수행
  - CBOW: 주변단어들을 이용해 다음 단어를 예측
  - Skip-gram: 한 단어의 주변단어들을 예측





## Word2Vec의 학습방법

- sliding window를 이용한 학습 set 구성
  - 주어진 주변 단어들을 입력했을 때, target word의 확률이 높아지도록 학습 혹은 그 반대
- embedding vector
  - input이 one-hot vector이므로  $W$ 가 embedding vector의 집합이 됨

$$[0 \ 0 \ 0 \ 1 \ 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

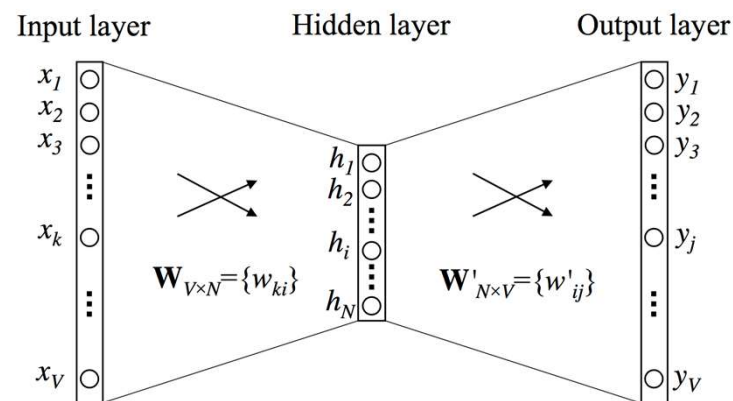
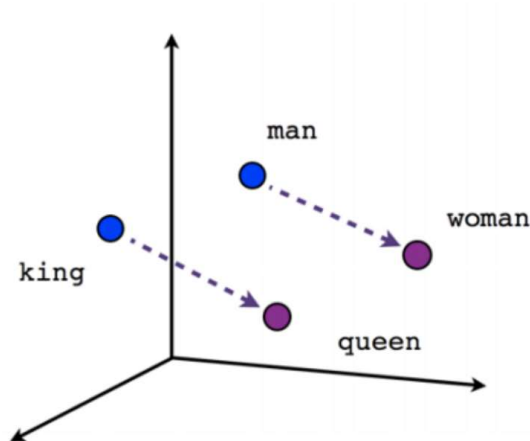


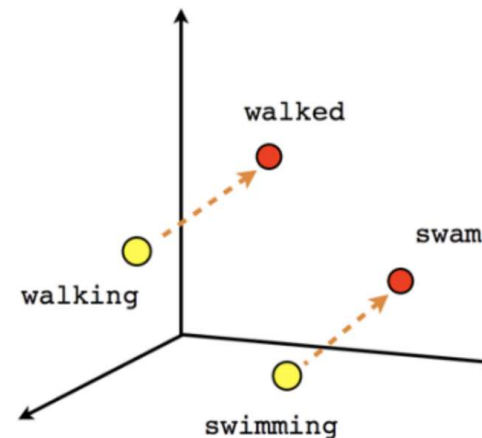
Figure 1: A simple CBOW model with only one word in the context

## Word2Vec의 의미

- 단어의 위치에 기반하여 의미를 내포하는 vector 생성
  - 비슷한 위치에 나타나는 단어들은 비슷한 vector를 가지게 됨
  - 단어 간의 유사성을 이용하여 연산이 가능



Male-Female

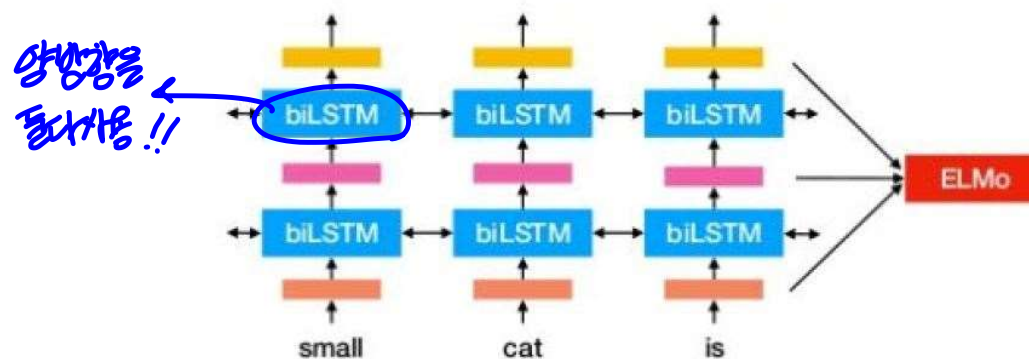


Verb tense

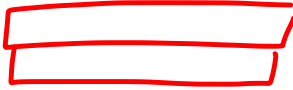
<https://www.tensorflow.org/images/linear-relationships.png>

## ELMo (Embeddings from Language Model)

- 사전 훈련된 언어 모델을 사용하는 워드 임베딩 방법론.
- 이전의 대표적인 임베딩 기법인 (Word2Vec)이나 (GloVe) 등 이 동일한 단어가 문맥에 따라 전혀 다른 의미를 가지는 것을 반영하지 못하는 것에 비해, ELMo는 이러한 문맥을 반영하기 위해 개발된 워드 임베딩 기법.
- 문맥의 파악을 위해 biLSTM으로 학습된 모델을 이용



vector  
 Word2Vec  
 GloVe  
 문맥에 따라 다른 의미를 가지는 단어를 반영하지 못함

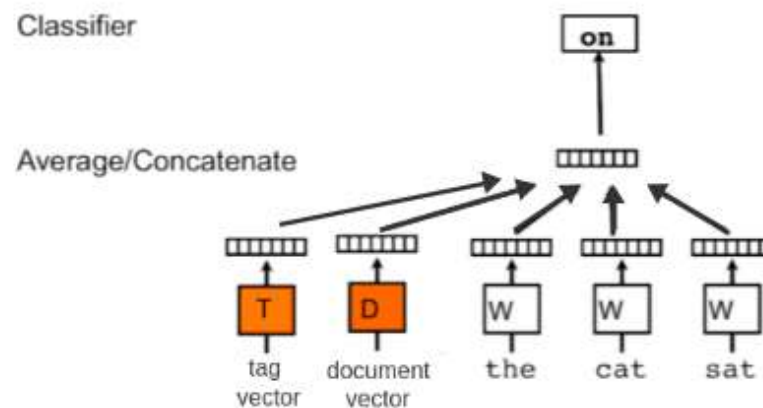


## Transfer Learning

- Transfer Learning
  - Wiki: storing knowledge gained while solving one problem and applying it to a different but related problem
- 텍스트 마이닝에서의 전이학습
  - feature level: 단어에 대한 dense vector(word embedding)를 새로 학습하지 않고 학습된 vector를 그대로 가져다 씀
  - model level: word embedding과 모형 전체를 가져다 학습
  - Word2Vec, Glove, ELMo 등의 사전학습된 word embedding이 전이학습에 많이 사용됨

## Document Embedding

- Word2Vec은 word에 대해 dense vector를 생성하지만, document vector는 여전히 sparse
- Word2Vec 모형에서 주변단어들에 더하여 document의 고유한 vector를 함께 학습함으로써 document에 대한 dense vector를 생성
- 이 dense vector를 이용해 매칭, 분류 등의 작업 수행



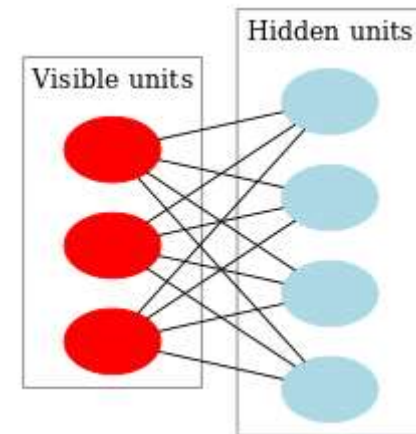
<https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e>

자랑은 안쓰임!

## RBM (Restricted Boltzmann Machine)

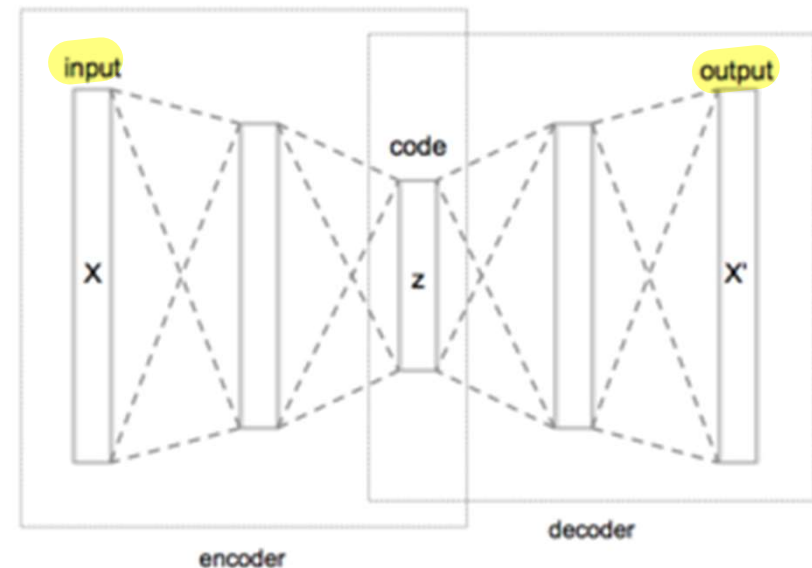
차원의 저주 극복

- 사전학습 목적으로 개발
  - G. Hinton에 의해 제안
  - 차원을 변경하면서 원래의 정보량 유지가 목적
    - 정보량을 물리학의 에너지 함수로 표현
- Deep NN의 vanishing gradient 문제 해결을 위해 제안
  - batch normalization, Dropout, ReLU 등의 기법으로 인해 문제가 해결되면서 지금은 많이 쓰이지 않음
- 사전학습을 통한 차원 축소에 사용 가능



## Autoencoder (다양한 방식에 따라)

- RBM과 유사한 개념
  - encoder로 차원을 축소하고 decoder로 다시 복원했을 때, 원래의  $x$ 와 복원한  $x'$ 이 최대한 동일하도록 학습
- 작동방식은 PCA와 유사
  - 데이터에 내재된 일정한 구조 - 연관성을 추출



## Context(sequence)의 파악

- N-gram *문맥으로 이해*
  - 문맥(context)를 파악하기 위한 전통적 방법
  - bi-gram, tri-gram, ...
  - 대상이 되는 문자열을 하나의 단어 단위가 아닌, 두개 이상의 단위로 잘라서 처리
- 딥러닝 - RNN
  - (문장)을 단어들의(sequence 혹은 series)로 처리
  - 뒷 단어에 대한 hidden node가 앞 단어의 hidden node 값에도 영향을 받도록 함
  - 그 외에도 단어들 간의 관계를 학습할 수 있는 모형을 고안



## N-gram

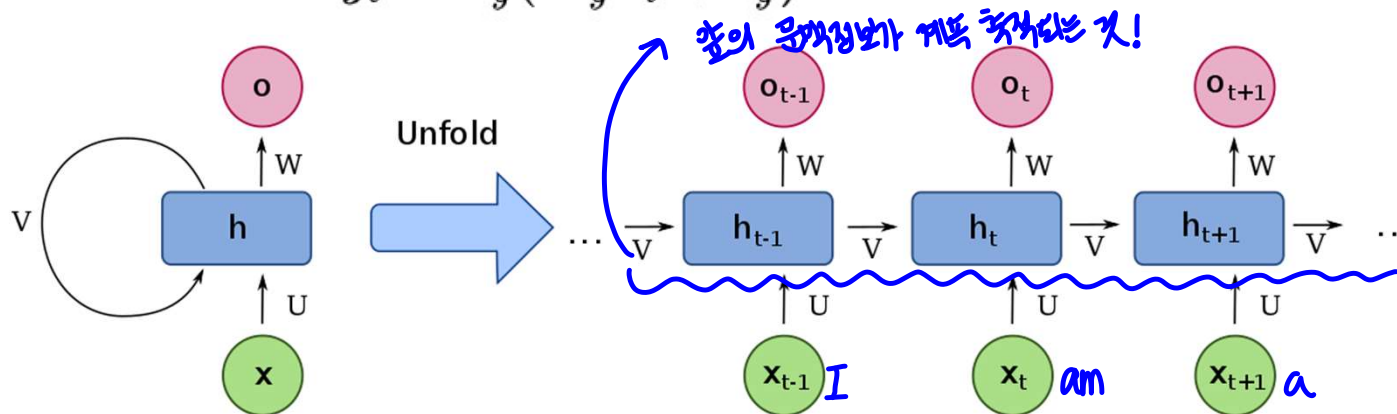
- “The future depends on what we do in the present”
- Unigram
  - The, future, depends, on, what, we, do, in, the, present
- Bi-gram
  - ‘The future’, ‘future depends’, ‘depends on’, ‘on what’, ... *추가 반영되고*
- Tri-gram
  - ‘The future depends’, ‘future depends on’, ‘depends on what’,  
...
- 보통 unigram에 bi-gram, tri-gram을 추가하면서 feature의 수가 증가시켜 사용
  - 문맥 파악에 유리하나, dimension이 더욱 증가

## 딥러닝과 텍스트마이닝 - RNN

- Sequence 정보를 기억할 수 있는 방법은?
  - X는 embedding된 word들을 순서대로 나열한 것
    - ex)  $X_1$ : The,  $X_2$ : future,  $X_3$ : depends,  $X_4$ : on, ...
  - hidden node가 X 뿐만 아니라 이전 hidden node로부터도 입력을 받음

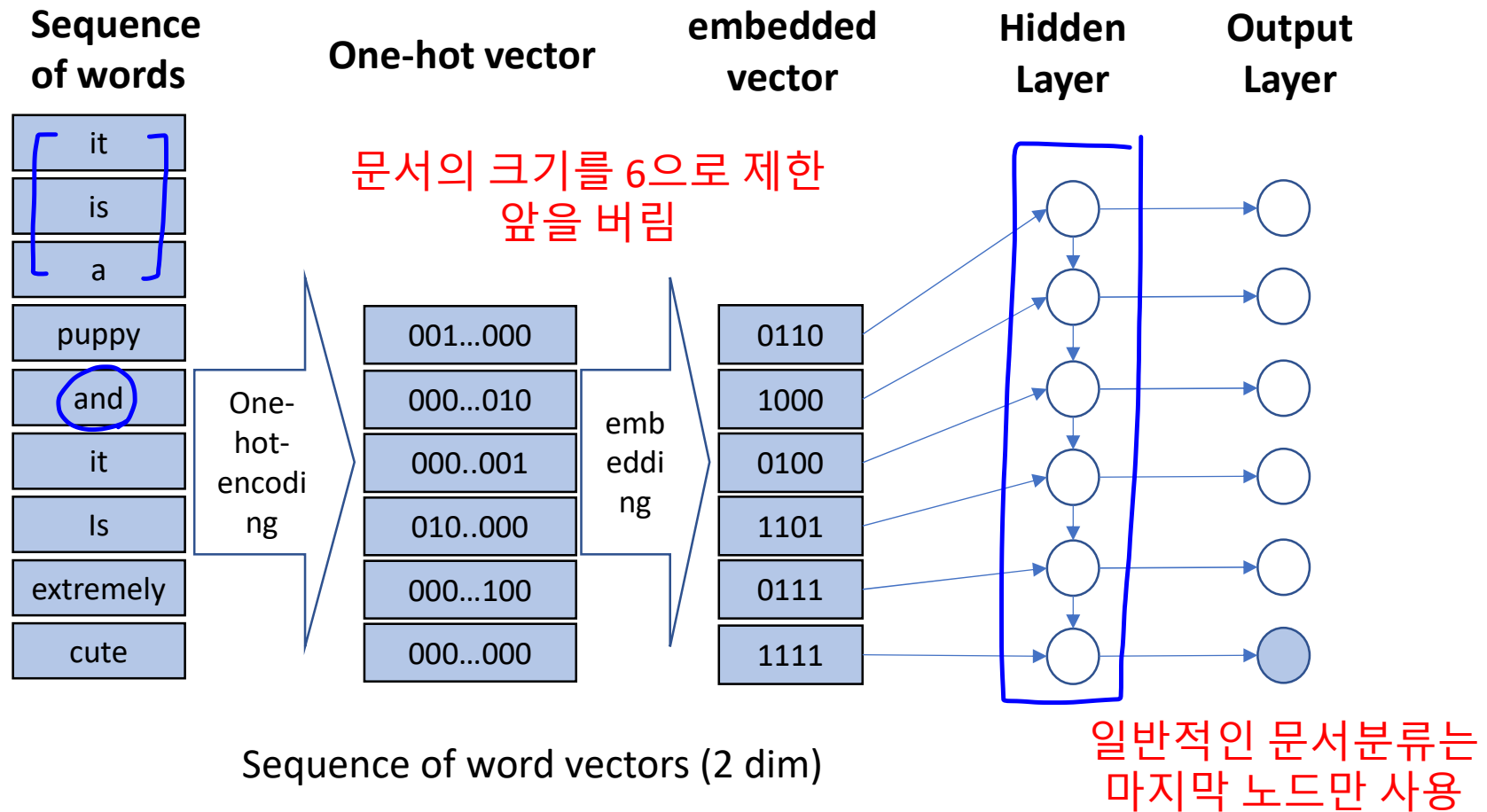
$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$



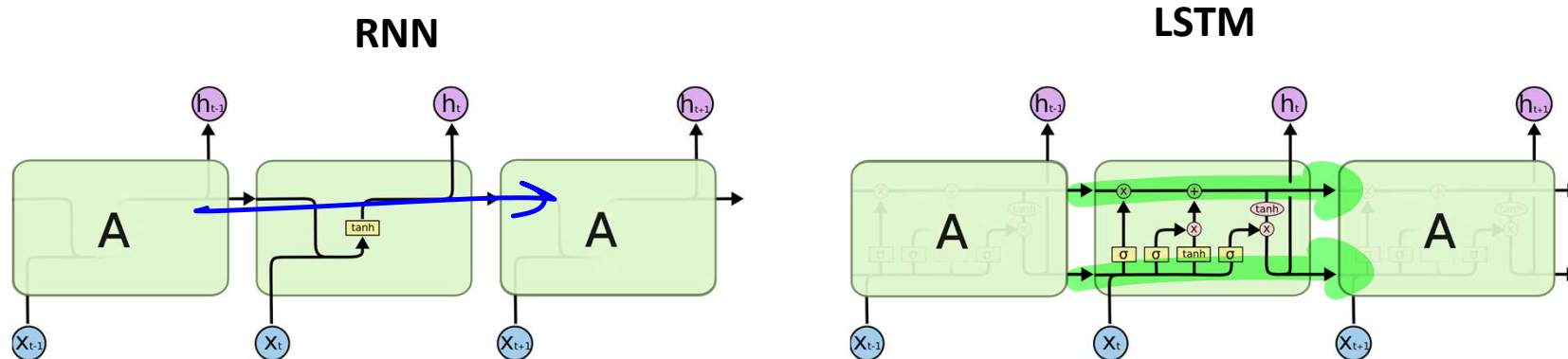
[https://en.wikipedia.org/wiki/Recurrent\\_neural\\_network](https://en.wikipedia.org/wiki/Recurrent_neural_network)

## Text Classification with RNN



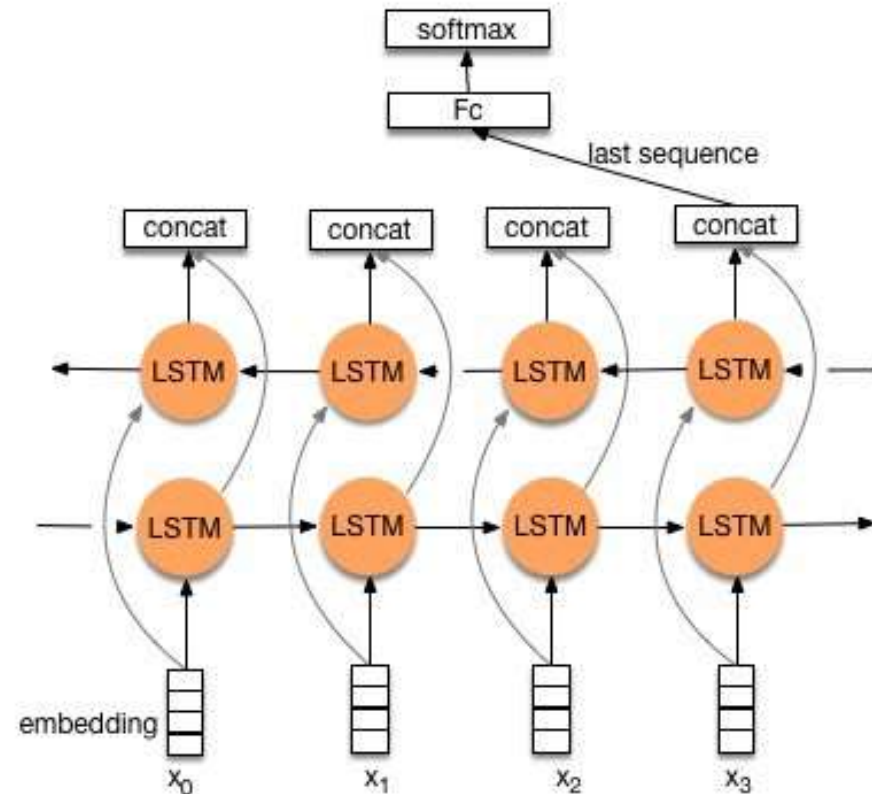
## LSTM (Long Short Term Memory)

- <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- RNN의 문제
  - 문장이 길수록 층이 깊은 형태를 갖게 됨 → 경사가 소실되는 문제 발생 → 앞부분의 단어 정보가 학습되지 않음
- LSTM: 직통 통로를 만들어 RNN의 문제를 해결



## Bi-LSTM

- 단방향 LSTM의 문제
  - 단어 순서가 갖는 문맥 정보가 한 방향으로만 학습된다.
  - 자신의 뒤에 오는 단어에 의해 영향을 받는 경우, 학습이 되지 않음
- Bi-LSTM
  - 양방향으로 LSTM을 구성하여 두 결과를 합침
  - 양방향 순서를 모두 학습



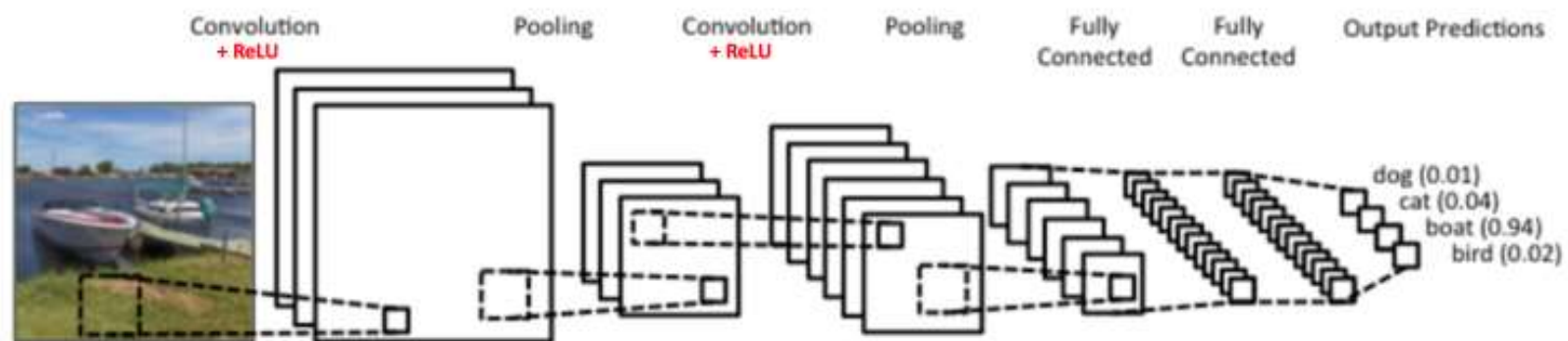
<https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66>

## 합성곱 신경망(Convolutional Neural Networks, CNN)

- CNN은 원래 이미지 처리를 위해 개발된 신경망으로, 현재는 인간의 이미지 인식보다 더 나은 인식 성능을 보이고 있음.
- - 그러나 CNN이 주변 정보를 학습한다는 점을 이용하여 텍스트의 문맥을 학습하여 문서를 분류하는 연구가 처음 있었으며, 의외로 뛰어난 성능을 보이게 되면서 자연어 처리에서의 활용분야가 넓어지게 됨.
- - CNN은 합성곱층(convolution layer)와 풀링층(pooling)으로 구성되며, 합성곱층은 2차원 이미지에서 특정 영역의 특징을 추출하는 역할을 하는데, 이는 연속된 단어들의 특징을 추출하는 것과 유사한 특성이 있음.

## CNN의 구조

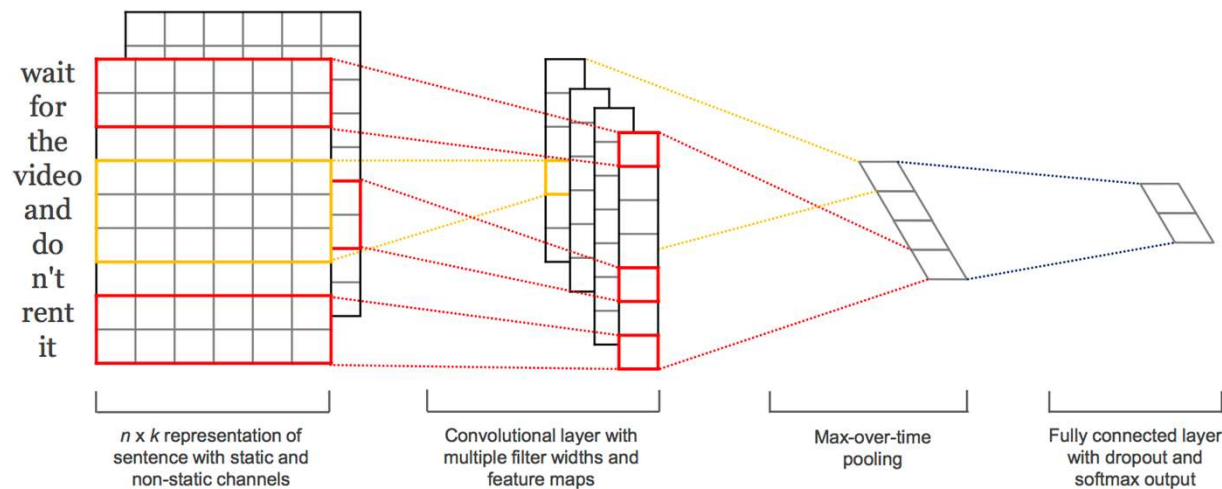
- 아래 그림은 전형적인 CNN의 구조를 보여주며, 합성곱층과 풀링층이 번갈아가면서 이미지의 특징을 단계적으로 추출하고, 마지막에 분류기를 통해 이미지를 판별하는 구조를 보여줌.



출처: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets>

## CNN을 이용한 문서 분류

- 아래 그림은 CNN을 이용한 문서분류 모형을 보여줌. 이미지와 달리 텍스트는 단어들의 1차원 시퀀스로 표현되므로 1D CNN모형을 사용함. 단어 시퀀스에 대해 CNN의 필터는 1차원으로만 적용되고 이렇게 텍스트의 특징을 추출한 결과를 마찬가지로 분류기에 넣어서 문서를 판별.



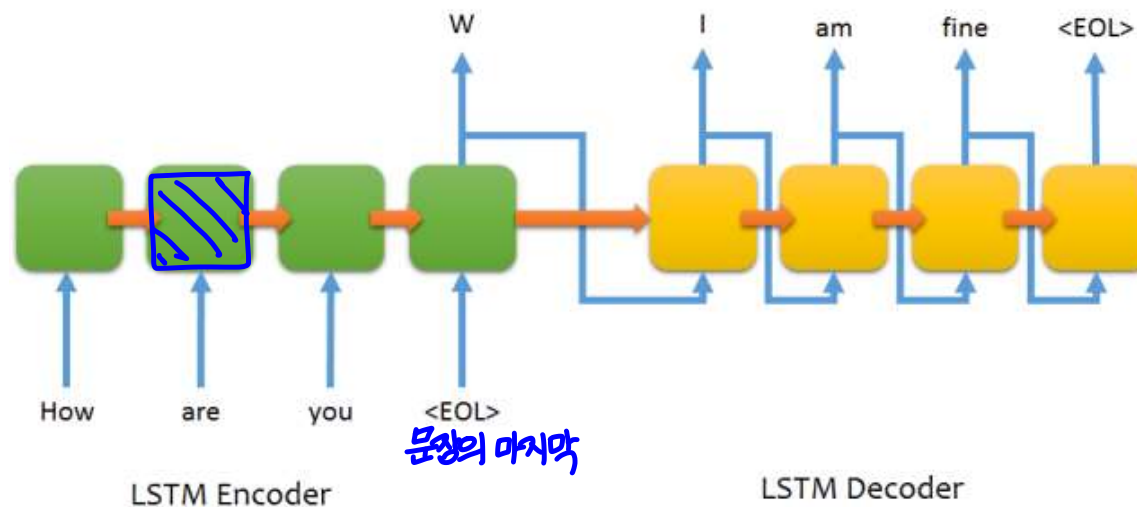
출처: <https://machinelearningmastery.com/best-practices-document-classification-deep-learning/>



## Sequence-to-sequence

- 지금까지는 입력은 sequence, 출력은 하나의 값인 경우가 일반적이었으나, 번역, chat-bot, summarize 등은 출력도 sequence가 되어야 함
- encoder, decoder의 구조를 가짐

영미문장 → 독일어문장

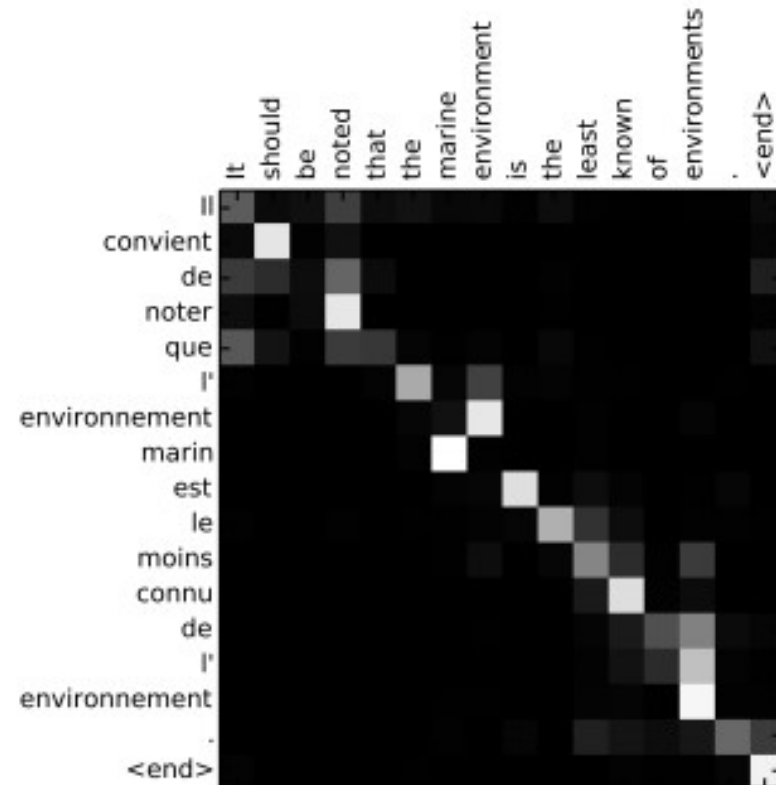
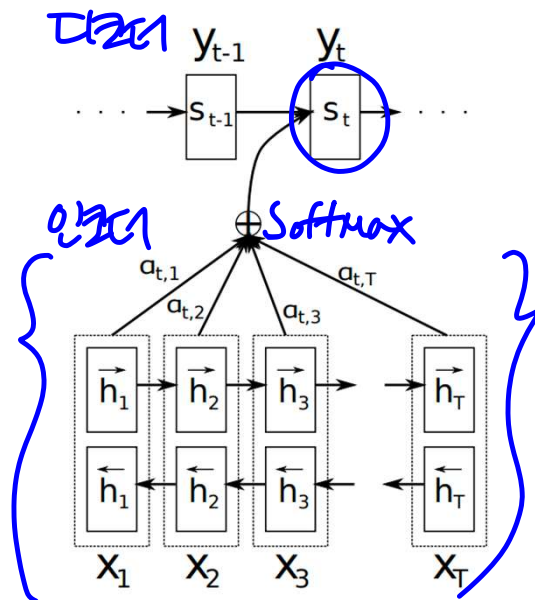


문장의 마지막

<https://github.com/farizrahman4u/seq2seq>

## Attention

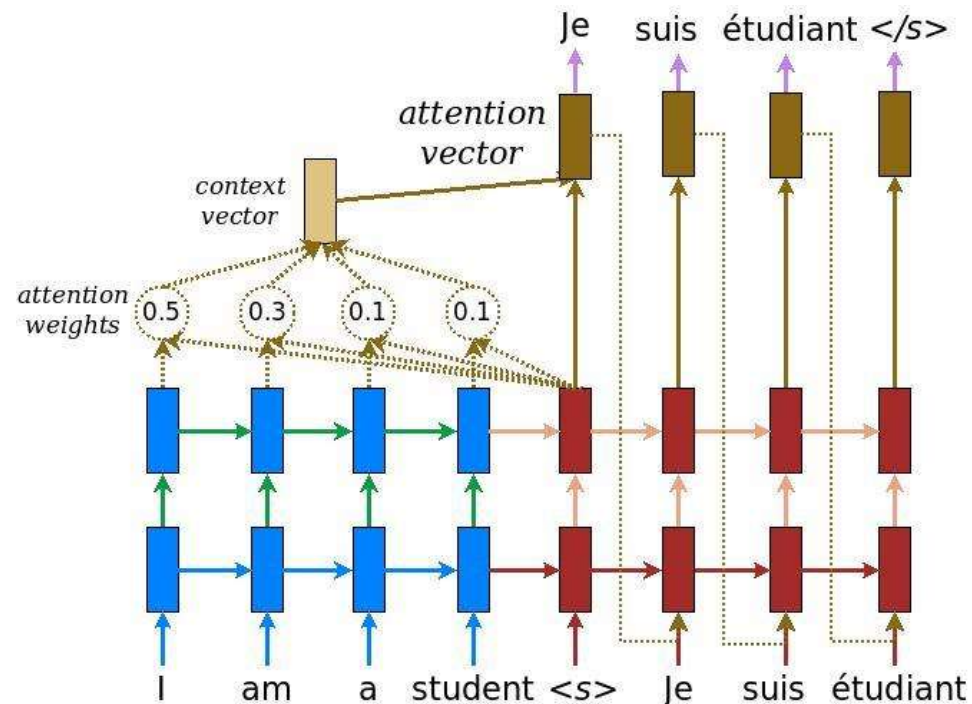
- 출력에 나온 어떤 단어는 입력에 있는 특정 단어들에 민감한 것에 착안
- 입력의 단어들로부터 출력 단어에 직접 링크를 만듦



Taken from "Neural Machine Translation by Jointly Learning to Align and Translate", 2015.

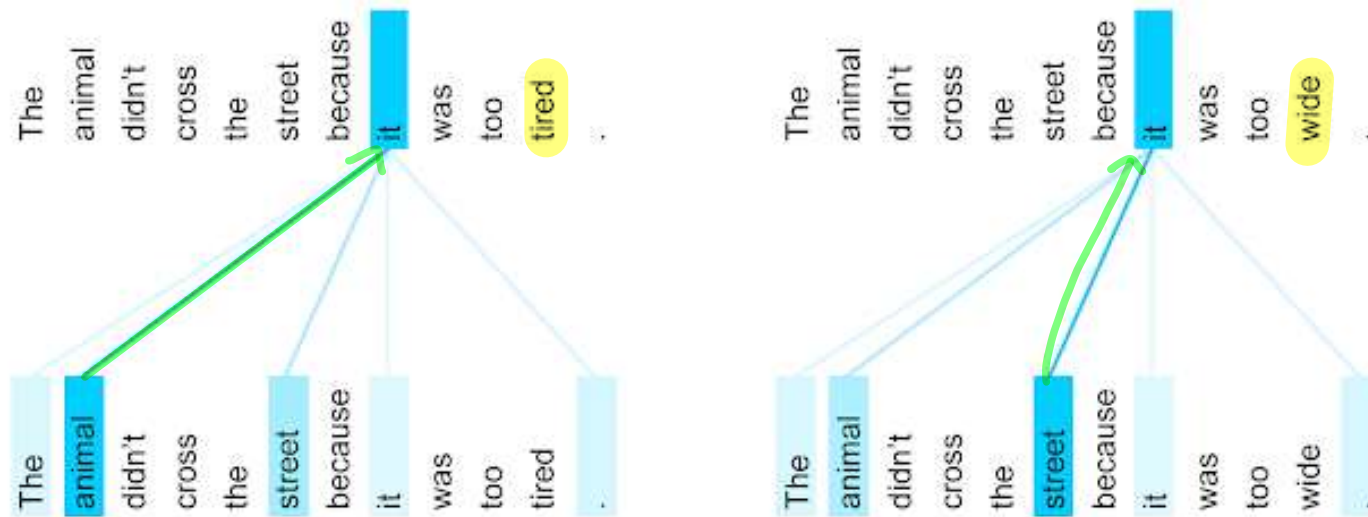
## Attention을 이용한 번역 예

- 아래 그림은 "I am a student"를 번역하는 과정에서, 첫 단어 'Je'가 앞 단어들에 대해 attention으로 연결되어 있으며, 이 단어의 생성에 'I'가 중요한 영향을 미치고 있음을 보여줌.



## Transformer (Self-attention) $RNN \leftrightarrow LSTM$

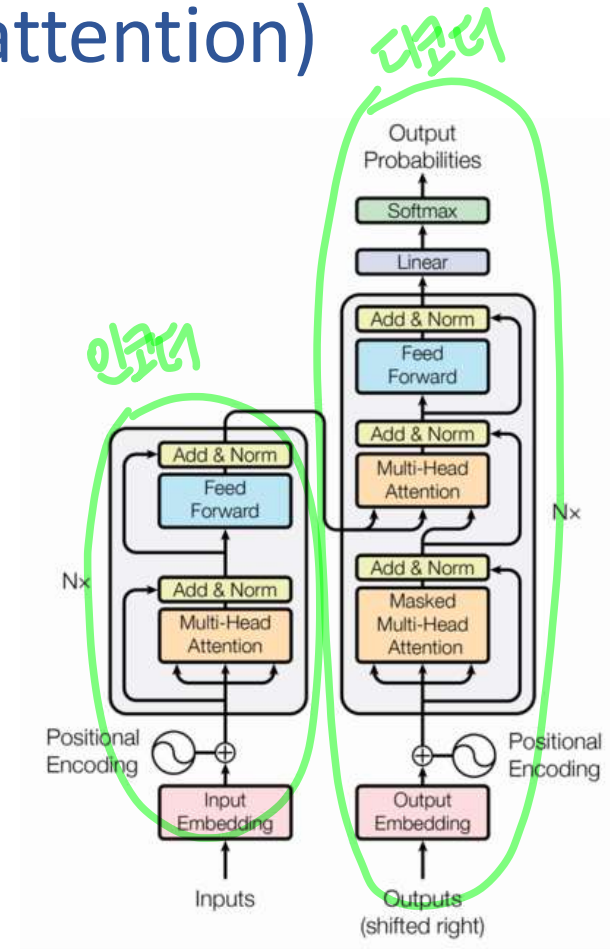
- 입력 단어들끼리도 상호연관성이 있는 것에 착안
  - 즉 입력  $\rightarrow$  출력으로의 attention 외에 입력 단어들 간의 attention, 입력 + 출력  $\rightarrow$  출력으로의 attention을 추가



<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

## Transformer (Self-attention)

- encoder와 decoder가 서로 다른 attention 구조를 사용
  - multi-head attention vs. masked multi-head attention
- RNN이 사라지고 self-attention이 이를 대신

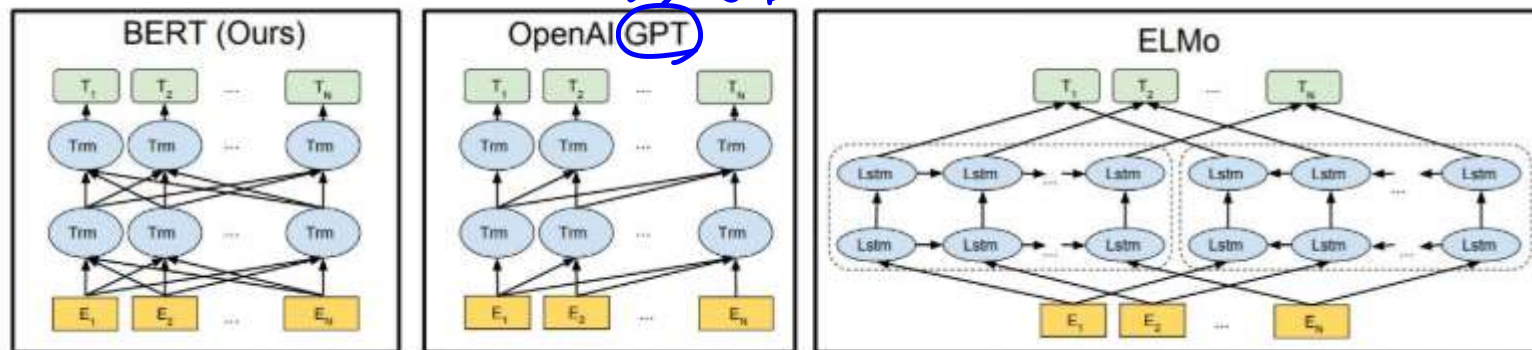


The Transformer Architecture

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.

## BERT (Bidirectional Encoder Representations from Transformer)

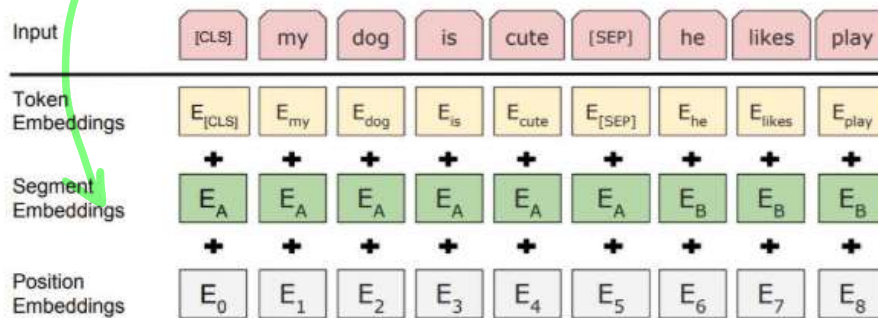
- 양방향 transformer 인코더를 사용
  - transformer에 기반한 OpenAI GPT와의 차이
- transfer learning에서 feature + model을 함께 transfer하고 fine tuning을 통해서 적용하는 방식을 선택
- 거의 모든 분야에서 top score를 기록



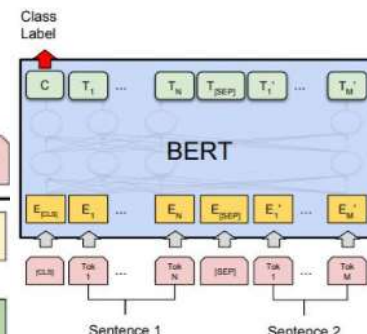
Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

## BERT 참고

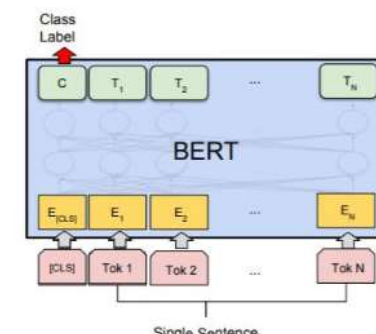
- segment, position embedding을 사용



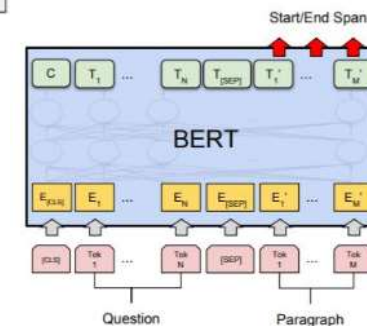
- 다양한 text mining task에 전이학습을 이용해 적용 가능한 구조를 제안



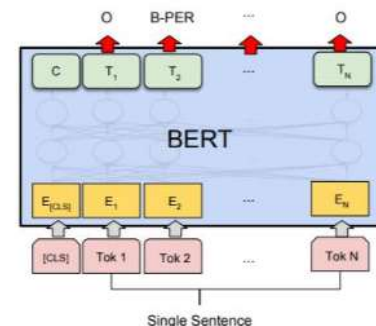
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).