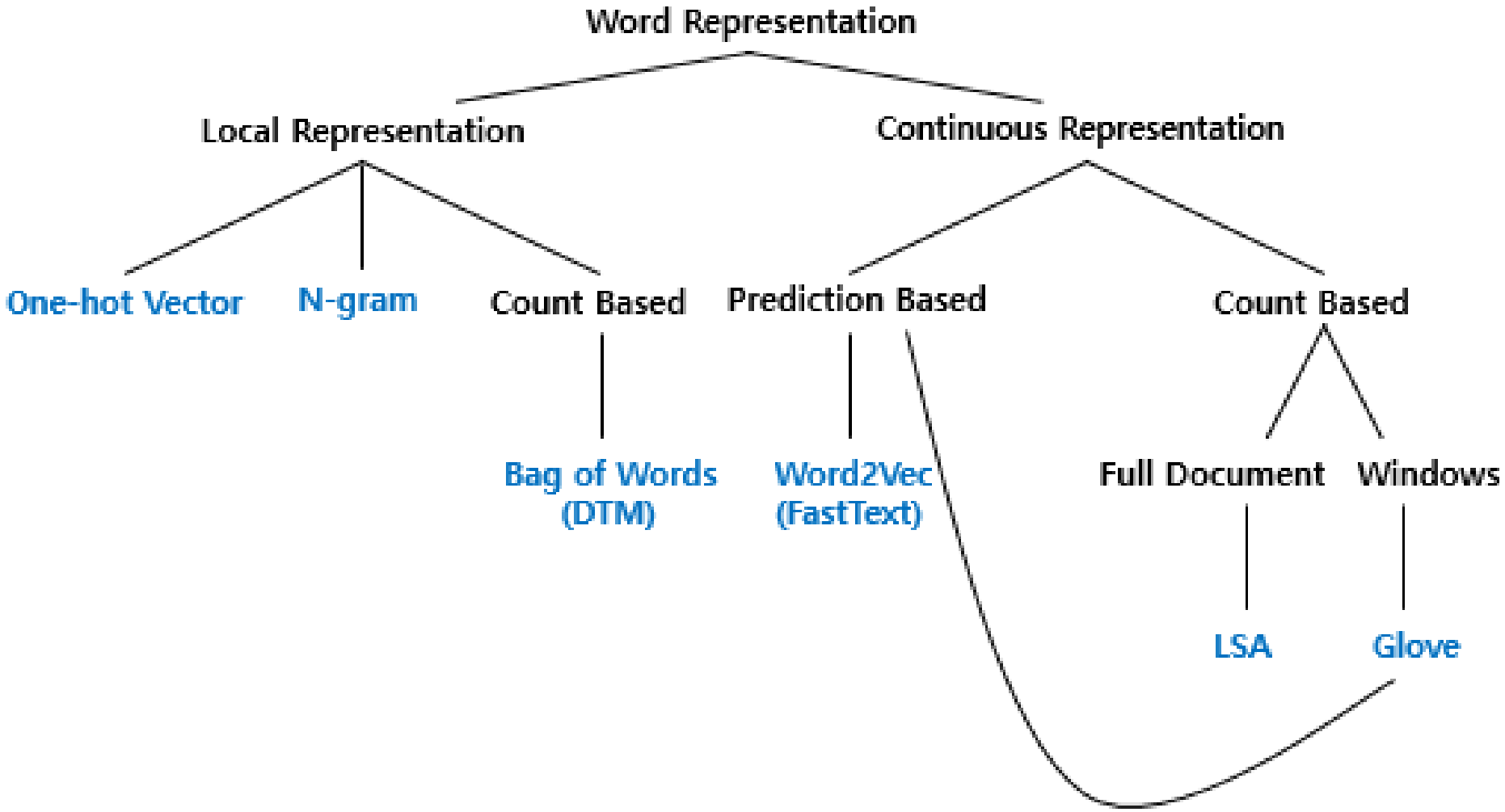
Lecture 6: 문서 표현과 확률적 언어 모형(Probabilistic Language Model)

Word Representation



단어의 표현 방법

1. **국소 표현(Local Representation) 방법**
2. **분산 표현(Distributed Representation) 방법**

국소 표현(Local Representation) 방법

- 해당 단어 그 자체만 보고, 특정값을 매핑하여 단어를 표현하는 방법
- ex) puppy(강아지), cute(귀여운), lovely(사랑스러운)라는 단어가 있을 때
    - 각 단어에 1번, 2번, 3번 등과 같은 숫자를 맵핑(mapping)하여 부여한다.
      → 국소 표현 방법
- 국소 표현 방법은 단어의 의미, 뉘앙스를 표현할 수 없다.
- 국소 표현 방법을 **이산 표현(Discrete Representation)**이라고도 한다.

분산 표현(Distributed Representation) 방법

- 그 단어를 표현하고자 주변을 참고하여 단어를 표현하는 방법
- ex) puppy(강아지), cute(귀여운), lovely(사랑스러운)라는 단어가 있을 때
    - puppy(강아지)라는 단어 근처에는 주로 cute(귀여운), lovely(사랑스러운)이라는 단어가 자주 등장
    - "puppy라는 단어는 cute, lovely한 느낌이다"로 단어를 정의한다.
- 분산 표현 방법은 단어의 뉘앙스를 표현할 수 있다.
- 분산 표현 방법을 **연속 표현(Continuous Representation)**이라고도 한다.

카운트 기반의 문서 표현 및 활용

카운트 기반의 문서 표현

- 문서를 단어의 나열로 보는 것이 아니라 단어의 통계로 보려는 시각
- “어떤 단어가 몇 번 쓰였는가?” 이것 만으로도 문서에 대한 짐작이 가능

BOW

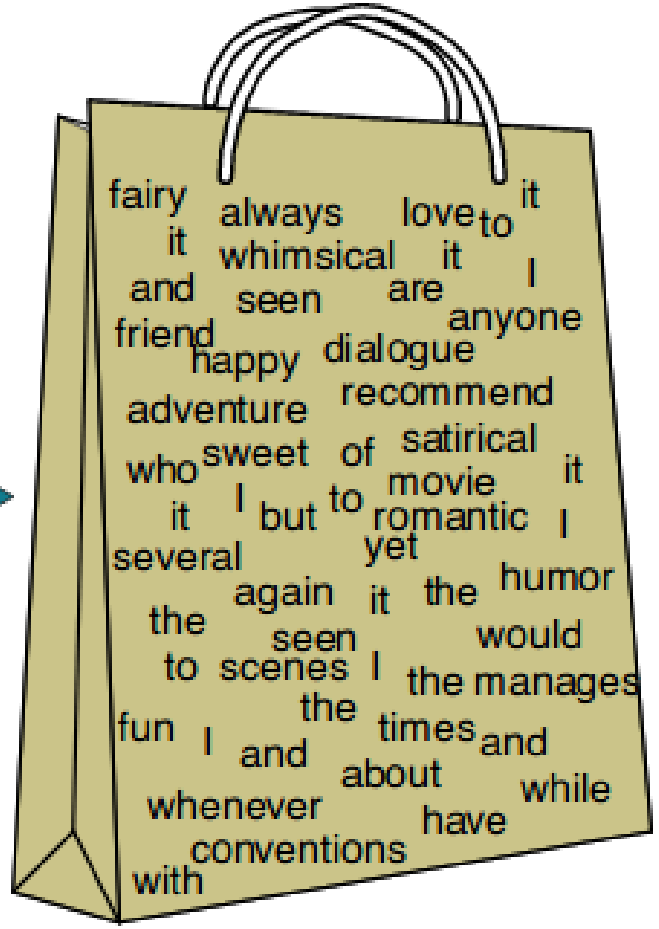
- 문서를 단어들의 sequence가 아닌 bag으로 표현
- tokenize된 결과를 이용하여 문서를 각 단어의 유무 혹은 개수로 이루어진 vector로 변환

텍스트와 특성 (feature)

- 텍스트 마이닝을 위해서는 대상이 되는 텍스트를 우리가 다룰 수 있는 수치 형태로 변환
- 특성(feature)
  - 대상이 갖고 있는 정보 중 우리가 관심이 있는 것을 표현
  - ex) 사람의 특성: 이름, 나이, 키, 몸무게 등
- 텍스트의 특성
  - 텍스트를 이해할 수 있는 특성은 무엇인가?
- 카운트 기반의 문서표현
  - 텍스트의 특성을 단어(토큰)로 표현
  - 텍스트에서 '쓰였다'/'안쓰였다'로 구분하기 위해 True/False로 표현
  - 혹은 그 단어가 텍스트에서 사용된 횟수를 특성 값으로 사용

Bag of Words (BoW)

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
…	…

Bag of Words (BoW)

- Vector Space Model (one-hot vector)
  - 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
  - 단어의 유/무 대신 단어가 문서에 나타난 횟수로 표현
  - count가 weight로 작용

문서 단어 행렬 (DTM)

- 문서 단어 행렬(Document-Term Maxtrix, DTM)은 다수의 문서 데이터(=Corpus)에서 등장한 모든 단어의 출현 빈도수(frequency)를 행렬로 표현한 것
- DTM은 다수의 문서 데이터에 대한 Bag of Words(BoW)를 행렬로 표현한 것
- DTM은 국소 표현(Local Representation) 또는 이산 표현(Discrete Representation)의 일종으로 카운트 기반의 단어 표현방법

DTM 한계점

희소 행렬 표현(Sparse Matrix Representation)

- DTM은 문서 개수와 문서 내 모든 단어 집합의 크기만큼의 차원을 갖는다
- 문서 데이터가(=row 개수) 방대해질수록 단어 집합의 크기(=column 개수)는 기하급수적으로 증가하기 때문에, DTM은 수만, 수십만 이상의 차원을 가질 수 있다.
- 차원의 개수가 커진 만큼 계산의 복잡도는 증가하기 때문에 리소스 낭비를 유발
- DTM은 대부분의 column이 0인 행렬 표현방식임에도 불구하고, 단어 집합의 크기만큼 차원을 갖기 때문에 저장공간의 낭비를 유발

단순 빈도수 기반 단어 표현

- DTM은 문서 데이터에서 단어의 빈도수만 고려하기 때문에 중요 단어와 불필요한 단어를 구분하기 어렵다는 한계 존재

TF-IDF (Term Frequency - Inverse Document Frequency)

TF

Frequency of a word within the document

IDF

Frequency of a word across the documents

TF-IDF (Term Frequency - Inverse Document Frequency)

- 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

TF-IDF (Term Frequency - Inverse Document Frequency)

- 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(전체문서수)= 10으로 가정하면, fast의 idf는 $\log(10/(1+2)) = 0.52$
  - fly의 idf는 $\log(10/(1+1)) = 0.70$
- TF-IDF: $tf(d,t) * idf(t)$

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

TF-IDF의 변형

Variants of term frequency (tf) weight

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

TF-IDF의 변형

Variants of inverse document frequency (idf) weight

weighting scheme	idf weight ($n_t = \left\vert\{d \in D : t \in d\}\right\vert$)
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	$\log \frac{N-n_t}{n_t}$

언어 모델의 개념

- 언어 모델(Language Model)은 문장이 얼마나 자연스러운지 확률적으로 계산함으로써 문장 내 특정 위치에 출현하기 적합한 단어를 확률적으로 예측하는 모델
- 언어 모델은 문장 내 앞서 등장한 단어를 기반으로 뒤에 어떤 단어가 등장해야 문장이 자연스러운지 판단하는 도구

언어 모델의 종류

- 통계학적 언어 모델
- 인공신경망 기반의 언어 모델

최근에는 BERT, GPT-3와 같은 인공신경망 기반의 언어 모델의 성능이 뛰어나 대부분의 자연어처리 문제에서는 인공신경망 기반의 언어 모델을 사용

> **"비행기를 타기 위해 공항을 가는데 차가 너무 막혀서 결국 비행기를 [???]"**

- 사람은 이 문장에서 '비행기를' 단어 뒤에 이어질 물음표(???)에 들어갈 단어로 '놓쳤다'를 쉽게 예측
- 수많은 단어와 문장을 듣고 쓰고 말하며 언어능력을 학습해왔기 때문에, 확률적으로 '놓쳤다'라는 단어가 가장 적합하다고 판단 가능
- 해당 문장을 기계(=언어 모델)에게 보여주고 물음표에 올 적합한 단어를 예측하라고 하면?
- 기계 역시 사람과 같은 프로세스로 동작
  - 먼저, 언어 모델은 텍스트 기반의 수많은 문장을 통해 어떤 단어가 어떤 어순으로 쓰인 것이 가장 자연스러운 문장인지 학습
  - 이를 통해 언어 모델은 문장이 주어졌을 때 앞뒤에 주어진(=어순) 단어 조합을 기반으로 가장 높은 확률로 등장할 만한 단어를 출력

확률적 언어 모형(Probabilistic Language Model)

수학에서 시퀀스(Sequence)는 순서를 고려하여 나열된 여러 객체들의 묶음을 의미한다는 점에서, 문장은 어순을 고려하여 여러 단어로 이루어진 단어 시퀀스(Word Sequence)라고도 함

m 개의 단어로 구성된 단어 시퀀스 W를 확률적으로 표현하면

$$P(W) = P(w_1, w_2, \ldots, w_m)$$

단어 시퀀스 자체의 확률은 모든 단어 $(w_1, w_2, \ldots, w_m)$ 예측이 완료되었을 때, 즉 문장이 완성되었을 때 알 수 있다.

단어 시퀀스의 확률은 각 단어의 확률과 단어들의 조건부 확률을 이용하여 다음과 같이 계산

$$P(w_1, w_2, \ldots, w_m) = P(w_1, w_2, \ldots, w_{m-1}) \cdot P(w_m | w_1, w_2, \ldots, w_{m-1})$$
$$= P(w_1, w_2, \ldots, w_{m-2}) \cdot P(w_{m-1} | w_1, w_2, \ldots, w_{m-2}) \cdot P(w_m | w_1, w_2, \ldots, w_{m-1})$$
$$= P(w_1) \cdot P(w_2 | w_1) \cdot P(w_3 | w_1, w_2) P(w_4 | w_1, w_2, w_3) \cdots P(w_m | w_1, w_2, \ldots, w_{m-1})$$

위 식을 곱연산(product operator)을 활용해 표현하면

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

여기에서 $P(w_m | w_1, w_2, \ldots, w_{m-1})$ 은 지금까지 $w_1, w_2, \ldots, w_{m-1}$라는 단어 열이 나왔을 때, 그 다음 단어로 w_m이 나올 조건부 확률을 말한다. 여기에서 지금까지 나온 단어를 **문맥(context)** 정보라고 한다.

언어 모델 활용 예시

기계 번역(Machine Translation)

- 아래와 같이 번역기를 돌린 2가지 문장이 있다고 가정했을 때, 좌측 문장이 우측 문장보다는 자연스러운 문장
- 언어 모델은 좌측 문장이 우측 문장보다 높은 확률을 갖는다고 판단

> ("나는 지하철을 탔다") > ("나는 지하철을 태웠다")

오타 교정(Spell Correction)

- 키보드 자판의 위치로 인해 아래와 같이 '알아갔다'를 '랄아갔다'로 오타가 발생
- 오타가 발생한 우측 문장보다는 좌측 문장이 자연스러운 문장
- 언어 모델은 좌측 문장의 확률이 우측 문장보다 높다고 판단

> ("자연어처리 방법론을 알아갔다") > ("자연어처리 방법론을 랄아갔다")

언어 모델 활용 예시

음성 인식(Speech Recognition)

- "신라"는 소리 내어 발음하면 [실라]
- 언어 모델은 이처럼 실제 텍스트와 발음 간의 차이를 교정해 주는 데 활용 가능

> ("신라의 달밤") > ("실라의 달밤")

검색어 추천(Keyword Search Recommendation)

- 네이버나 구글 검색 시자동완성 기능
- 언어 모델을 통해 사용자가 입력한 단어를 기반으로 뒤이어 입력할 만한 단어를 추천

모형의 종류

이 때 조건부 확률을 어떻게 모형화하는냐에 따라

- 유니그램 모형(Unigram Model)
- 바이그램 모형(Bigram Model)
- N그램 모형(N-gram Model)

유니그램 모형

만약 모든 단어의 활용이 완전히 서로 독립이라면 단어 열의 확률은 다음과 같이 각 단어의 확률의 곱이 된다. 이러한 모형을 유니그램 모형이라고 한다.

$$P(w_1, w_2, \ldots, w_m) = \prod_{i=1}^{m} P(w_i)$$

바이그램 모형

만약 단어의 활용이 바로 전 단어에만 의존한다면 단어 열의 확률은 다음과 같다. 이러한 모형을 바이그램 모형 또는 마코프 모형(Markov Model)이라고 한다.

$$P(w_1, w_2, \ldots, w_m) = P(w_1) \prod_{i=2}^{m} P(w_i | w_{i-1})$$

N그램 모형

만약 단어의 활용이 바로 전 $n-1$개의 단어에만 의존한다면 단어 열의 확률은 다음과 같다. 이러한 모형을 N그램 모형이라고 한다.

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

조건부 확률의 추정

빈도를 사용한 각각의 조건부 확률 추정

$$P(w|w_c) = \frac{C((w_c, w))}{C((w_c))}$$

위 식에서 $C(w_c, w)$은 전체 말뭉치에서 (w_c, w)라는 바이그램이 나타나는 횟수이고 $C(w_c)$은 전체 말뭉치에서 (w_c)라는 유니그램(단어)이 나타나는 횟수

바이그램 언어 모형

조건부 확률을 알게 되면 각 문장의 확률을 구할 수 있다.

다음으로 이 토큰열을 N-그램형태로 분해한다.
바이그램 모형에서는 전체 문장의 확률은 다음과 같이 조건부 확률의 곱으로 나타난다.

$$P(\mathrm{SS\ I\ am\ a\ boy\ SE}) = P(\mathrm{I}|\mathrm{SS}) \cdot P(\mathrm{am}|\mathrm{I}) \cdot P(\mathrm{a}|\mathrm{am}) \cdot P(\mathrm{boy}|\mathrm{a}) \cdot P(.|\mathrm{boy}) \cdot P(\mathrm{SE}|.)$$

N-gram 기반 단어 예측

N-gram 언어 모델은 이전 $(n-1)$개의 단어를 바탕으로 n번째 올 단어를 예측

- '오늘 점심 추천 메뉴'에 뒤이어 나올 단어 w를 예측할 때, n이 4인 4-gram을 활용한다고 가정
- 4-gram은 3개의 단어만 고려
- '오늘 점심 추천 메뉴'에서 '오늘'이라는 단어는 고려하지 않고 '점심 추천 메뉴'만 고려
- '오늘 점심 추천 메뉴'에서 뒤이어 나올 단어 가 등장할 확률을 수학적으로 표현하면

$$P(w|\text{점심 추천 메뉴}) = \frac{count(\text{점심 추천 메뉴 } w)}{count(\text{점심 추천 메뉴})}$$

> 전체 문서 데이터에서 '점심 추천 메뉴'이라는 문장이 1,000번, '점심 추천 메뉴' 문장 뒤에 '파스타' 단어가 500번 등장하고, '갈비탕'이 300번 등장했다고 가정

$$P(\text{파스타}|\text{점심 추천 메뉴}) = \frac{count(\text{점심 추천 메뉴 파스타})}{count(\text{점심 추천 메뉴})} = 0.5$$

$$P(\text{갈비탕}|\text{점심 추천 메뉴}) = \frac{count(\text{점심 추천 메뉴 갈비탕})}{count(\text{점심 추천 메뉴})} = 0.3$$

N-gram의 한계점

정확도 문제

- N-gram 언어 모델은 문장 내 전체 단어를 고려하는 언어 모델과 비교했을 때 정확도가 낮음
- N-gram 언어 모델은 이전에 N개의 단어만 고려하기 때문에, 앞 문장의 맥락을 놓치는 경우 발생
- 이는 문장의 앞부분과 뒷부분의 문맥이 매끄럽지 못하게 만들 수 있음
- 예를 들어, N-gram 기반 챗봇을 만들었다고 가정
  - '오늘 점심 메뉴 추천' 다음 이어질 단어를 예측할 때 trigram을 사용하면 '오늘'이라는 단어는 고려하지 않습니다.
  - 만약, 사용자가 '오늘'이 아닌 '내일'의 점심 메뉴를 물어봐도 챗봇은 '오늘' 점심 추천 메뉴와 같은 결과만 출력
- N-gram은 문장 앞부분의 중요한 키워드나 맥락을 놓치는 경우 발생

희소(Sparsity) 문제

- 전체 문서(=Corpus)에서 문장 내 전체 단어를 고려하는 언어 모델과 비교했을 때, N-gram 언어 모델은 일부 단어만 고려하기 때문에 전체 문서에서 특정 단어의 출현 확률을 높일 수 있다는 장점 존재
- 그럼에도 불구하고, 전체 문서에서 N개의 단어를 연속적으로 갖는 문장 자체 개수가 드물다는 희소 문제 존재

상충(Trade-off) 문제

- N-gram 언어 모델은 N값 설정 시 상충(Trade-off) 문제가 존재
- N의 값이 커도 문제이고, 작아도 문제
- N은 최대 5를 넘지 않는 선에서 활용하는 것을 권장
- N을 무작정 크게 설정하면 전체 문서에서 해당 N-gram 문장을 카운팅 할 확률이 낮아지므로 희소 문제 심각
- N이 클수록 언어 모델의 사이즈 증가로 연산 비용도 증가
- N의 값이 너무 작으면 전체 데이터에서 문장 카운팅은 잘 될 수는 있어도, 실질적으로 매끄러운 문장이 만들어지긴 어려움

> N의 값에 따른 성능 비교 연구
> 해당 연구는 월스트리트 저널(Wall Street Journal)에서 3,800만 개의 단어를 기반으로 언어 모델을 학습하고, 1,800만 개의 테스트 데이터를 활용한 테스트를 통해 Perplexity(PPL) 값을 비교

	Unigram	**Bigram**	**Trigram**
Perplexity	962	170	109

언어 모델의 평가 방법(Evaluation metric) - 외부 평가(extrinsic evaluation)

외부 평가(extrinsic evaluation)

- 두 개의 모델 A, B가 있을 때 이 모델의 성능은 어떻게 비교할 수 있을까?
  - 두 개의 모델을 오타 교정, 기계 번역 등의 평가에 투입해볼 수 있음
  - 그리고 두 모델이 해당 업무의 성능을 누가 더 잘했는 지를 비교하면 된다.
- 그런데 두 모델의 성능을 비교하고자, 일일히 모델들에 대해서 실제 작업을 시켜보고 정확도를 비교하는 작업은 공수가 너무 많이 드는 작업이다.
- 만약 비교해야 하는 모델이 두 개가 아니라 그 이상의 수라면 시간은 비교해야 하는 모델의 수만큼 배로 늘어날 수 있다.
- 이러한 평가를 외부 평가(extrinsic evaluation)라고 한다.

언어 모델의 평가 방법(Evaluation metric) - 내부 평가(Intrinsic evaluation)

펄플렉서티(perplexity)

- 이러한 외부 평가보다는 어쩌면 조금은 부정확할 수는 있어도 테스트 데이터에 대해서 빠르게 식으로 계산되는 더 간단한 평가 방법이 있다.
- 바로 모델 내에서 자신의 성능을 수치화하여 결과를 내놓는 `내부 평가(Intrinsic evaluation)`에 해당되는 `펄플렉서티(perplexity)`이다.

Perplexity의 계산

- PPL은 **단어의 수로 정규화(normalization)된 테스트 데이터에 대한 확률의 역수**이다.
- PPL을 최소화한다 $\rightarrow$ 문장의 확률을 최대화한다.
- 문장 W의 길이가 N이라고 했을 때의 PPL은 다음과 같다.

$$PPL(W) = P(w_1, w_2, w_3, \cdots, w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \cdots, w_N)}}$$

- 문장의 확률에 연쇄 법칙(chain rule)을 적용하면 아래와 같다.

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \cdots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(w_i \mid w_1, w_2, w_3, \cdots, w_{i-1})}}$$

- 여기에 n-gram을 적용해볼 수도 있다.
- 예를 들어 bigram 언어 모델의 경우에는 식이 아래와 같다.

$$PPL(W) = N\sqrt{\frac{1}{\prod_{i=1}^{N} P(w_i \mid w_{i-1})}}$$

분기 계수 (Branching factor)

- PPL은 선택할 수 있는 가능한 경우의 수를 의미하는 `분기계수(branching factor)`이다.
- PPL은 이 언어 모델이 특정 시점에서 평균적으로 몇 개의 선택지를 가지고 고민하고 있는 지를 의미한다.
- ex) 언어 모델에 어떤 테스트 데이터를 주고 측정했더니 PPL이 10이 나왔다고 가정
  - 그렇다면 해당 언어 모델은 테스트 데이터에 대해서 다음 단어를 예측하는 모든 시점(time-step)마다 평균적으로 10개의 단어를 가지고 어떤 것이 정답인지 고민하고 있다고 볼 수 있다.
- 같은 테스트 데이터에 대해서 두 언어 모델의 PPL을 각각 계산한 후에 PPL의 값을 비교하면, 두 언어 모델 중 어떤 것이 성능이 좋은 지도 판단이 가능하다.
- 당연히 PPL이 더 낮은 모델의 성능이 더 좋다고 볼 수 있다.

$$PPL(W) = P(w_1, w_2, w_3, \cdots, w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

분기 계수 (Branching factor)

- PPL은 선택할 수 있는 가능한 경우의 수를 의미하는 **분기계수(branching factor)**
- PPL은 이 언어 모델이 특정 시점에서 평균적으로 몇 개의 선택지를 가지고 고민하고 있는 지를 의미
- ex) 언어 모델에 어떤 테스트 데이터를 주고 측정했더니 PPL이 10이 나왔다고 가정
  - 해당 언어 모델은 테스트 데이터에 대해서 다음 단어를 예측하는 모든 시점(time-step)마다 평균적으로 10개의 단어를 가지고 어떤 것이 정답인지 고민하고 있다고 볼 수 있다.
- 같은 테스트 데이터에 대해서 두 언어 모델의 PPL을 각각 계산한 후에 PPL의 값을 비교하면, 두 언어 모델 중 어떤 것이 성능이 좋은 지도 판단이 가능하다.

$$PPL(W) = P(w_1, w_2, w_3, \cdots, w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}^{N}\right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

기존 언어 모델 Vs. 인공 신경망을 이용한 언어 모델

- 페이스북 AI 연구팀, n-gram 언어 모델과 딥러닝을 이용한 언어 모델에 대해서 PPL로 성능 테스트를 한 표 공개

Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

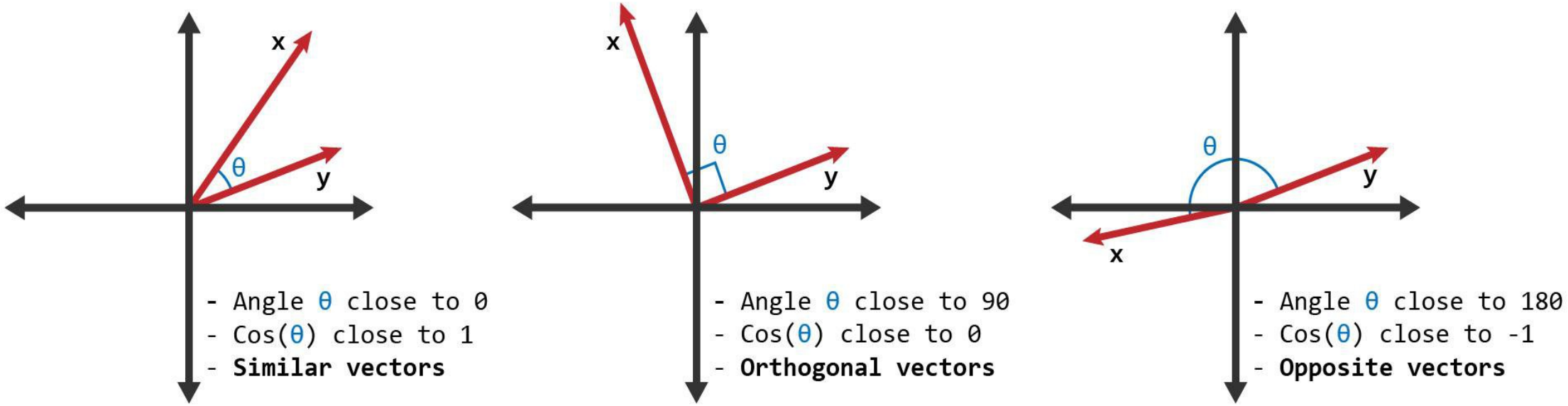
코사인 유사도 (Cosine Similarity)

코사인 유사도의 의미

- 두 벡터 간의 코사인 각도를 이용하여 구할 수 있는 두 벡터의 유사도를 의미
- 두 벡터의 방향이 완전 동일하다 $\rightarrow$ 코사인 유사도 값 = 1
- 두 벡터가 90°의 각을 이룬다 $\rightarrow$ 코사인 유사도 값 = 0
- 두 벡터가 180°의 각을 이룬다(반대의 방향) $\rightarrow$ 코사인 유사도 = -1

코사인 유사도 (Cosine Similarity)

- 코사인 유사도는 -1 이상 1 이하의 값을 가지며 값이 1에 가까울수록 유사도가 높다고 판단
- 직관적으로 이해하면 두 벡터가 가리키는 방향이 얼마나 유사한가를 의미



코사인 유사도 식

- 두 벡터 A, B에 대해서 코사인 유사도는 식으로 표현하면 다음과 같다.

$$similarity = cos(\Theta) = \frac{A \cdot B}{||A||\ ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

- 문서 단어 행렬(DTM)이나 TF-IDF 행렬을 통해서 문서의 유사도를 구하는 경우
  - 문서 단어 행렬(DTM)이나 TF-IDF 행렬이 각각의 특징 벡터 A, B가 된다.

문서 단어 행렬(DTM)에 대한 코사인 유사도 계산

> 문서1 : 저는 사과 좋아요
> 문서2 : 저는 바나나 좋아요
> 문서3 : 저는 바나나 좋아요 저는 바나나 좋아요

- 위의 세 문서에 대해서 문서 단어 행렬(DTM)을 만들면 다음과 같다.

-	바나나	사과	저는	좋아요
문서1	0	1	1	1
문서2	1	0	1	1
문서3	2	0	2	2

```python
from numpy import dot
from numpy.linalg import norm
import numpy as np

def cos_sim(A, B):
  return dot(A, B)/(norm(A)*norm(B))

doc1 = np.array([0, 1, 1, 1])
doc2 = np.array([1, 0, 1, 1])
doc3 = np.array([2, 0, 2, 2])

print("문서1과 문서2의 코사인 유사도 : ", cos_sim(doc1, doc2))
print("문서1과 문서3의 코사인 유사도 : ", cos_sim(doc1, doc3))
print("문서2과 문서3의 코사인 유사도 : ", cos_sim(doc2, doc3))
```

> 문서1과 문서2의 코사인 유사도 : 0.6666666666666667
> 문서1과 문서3의 코사인 유사도 : 0.6666666666666667
> 문서2과 문서3의 코사인 유사도 : 1.0000000000000002