자연어처리(NLP) 핵심 개념 정리

1. 자연어처리(NLP)의 정의

- 자연어(한국어, 영어 등)의 이해, 생성 및 분석을 다루는 인공지능 연구 분야
- 자연어 이해(NLU): 형태/통사/의미 분석을 통해 자연어를 컴퓨터가 이해할 수 있는 형태로 변환
  - 감정 분석, 개체 독해, 상식 추론, 의미론적 유사도 측정 등
- 자연어 생성(NLG): 컴퓨터가 처리한 결과물을 텍스트나 음성으로 변환
  - 자동 완성, 스토리 생성, 생성형 언어 모델, 캡션 생성 등
- 교집합: 생성형 문서 요약, 생성형 질의응답, E2E 챗봇 등

2. 통계적 언어 모델과 N-gram

- 통계적 언어 모델: 자연어 문장이나 단어 시퀀스에 확률을 할당하는 모델
- N-gram: 연속된 N개 항목(단어, 문자, 음절)을 하나의 단위로 처리
  - Unigram(1-gram): 단일 단어/문자만 고려 (예: "나", "는", "밥")
  - Bigram(2-gram): 연속된 2개 단어/문자 (예: "나는", "는 밥", "밥을")
  - Trigram(3-gram): 연속된 3개 단어/문자 (예: "나는 밥", "는 밥을", "밥을 먹는다")

3. 로그 확률 사용 이유

- 수치적 안정성: 매우 작은 확률값들의 곱셈 시 언더플로우 방지
  - $P(A) \times P(B) \times P(C) \rightarrow \log(P(A)) + \log(P(B)) + \log(P(C))$
- 계산 효율성: 곱셈이 덧셈으로 변환되어 계산 효율 향상
- 확률값 비교: 매우 작은 확률값들의 차이가 로그 스케일에서 더 명확하게 드러남
- 학습 안정성: 딥러닝의 cross-entropy 손실 함수에서 안정적 학습 가능

4. 라플라스 스무딩(Laplace Smoothing)

- 목적: N-gram의 Zero 확률 문제(희소성 문제) 해결
- 방법: 모든 가능한 N-gram 조합에 1을 더해 등장하지 않은 조합에도 최소 확률 부여
- 수식:
  - 기존: $P(w_n \mid w_1, ..., w_{n-1}) = \text{Count}(w_1, ..., w_n) / \text{Count}(w_1, ..., w_{n-1})$
  - 스무딩 적용: $P(w_n \mid w_1, ..., w_{n-1}) = (\text{Count}(w_1, ..., w_n) + 1) / (\text{Count}(w_1, ..., w_{n-1}) + V)$

    - V는 전체 어휘 개수
- 한계: 어휘 수가 클수록 실제 등장 조합의 확률이 과도하게 낮아질 수 있음

5. 마르코프 가정(Markov Assumption)

- 정의: "현재 상태(단어)는 오직 바로 앞의 n-1개 상태(단어)에만 의존한다"는 가정
- 이유:
    - 계산의 단순화: 모든 이전 단어 고려 시 계산량 기하급수적 증가 방지
    - 데이터 희소성 문제 완화: 필요한 데이터양 감소
    - 모델 학습 및 구현 용이
- 한계:
    - 장기 의존성(Long-term dependency) 무시
    - 차원의 저주: n이 커질수록 파라미터가 기하급수적으로 증가

6. 한국어 자연어처리의 난이도

- 교착어적 특성: 조사와 어미가 풍부하여 어근에 다양한 접사 결합
- 형태소의 복잡성: 불규칙 활용, 음운 변화 다양
- 높은 문맥 의존성: 주어나 목적어 생략이 빈번
- 어순의 상대적 자유로움: 조사 사용으로 어순 변경 자유
- 어휘 희소성 문제: 형태소 결합으로 생성 가능한 단어 형태가 매우 다양
- 형태소 분석의 복잡성: 전처리 과정에서 오류 발생 가능성 높음
- OOV(Out-of-Vocabulary) 문제 심각: 조사/어미의 다양한 결합으로 미등장 단어 빈번

7. 임베딩(Embedding)과 BoW(Bag of Words)

- 임베딩: 텍스트 데이터를 고정된 크기의 실수 벡터로 변환하는 방법
    - 고차원→저차원 변환, 의미적 유사성 반영
- BoW 가정:
    - 단어 순서 무시: 문장 내 단어 순서는 중요하지 않음
    - 단어 빈도만 고려: 각 단어의 등장 횟수만 반영
    - 문서 길이 반영 가능: 단어 빈도 벡터의 합으로 간접 반영
- BoW 한계:
    - 단어 순서와 문맥 정보 미반영
    - 의미적 유사 단어 구분 불가
    - 희소하고 고차원 벡터 생성

8. Word2Vec

- 단어를 벡터로 임베딩하는 방법(CBOW, Skip-gram)
- 네거티브 샘플링(negative sampling):
  - 전체 단어집합이 아닌 일부 네거티브 샘플만 사용
  - 중심 단어와 주변 단어 쌍이 실제 문맥에 함께 등장하는지(1) 아닌지(0) 구분하는 이진 분류 문제로 변환
- 효과:
  - 계산 효율성 향상: 전체 softmax 대신 소수 샘플에 대한 sigmoid 계산
  - 학습 속도 향상: 대용량 코퍼스에서도 빠른 학습 가능
  - 임베딩 품질 유지: 적절한 샘플 수 선택 시 전체 softmax와 유사한 품질

9. FastText

- Facebook AI Research에서 개발한 단어 임베딩 및 텍스트 분류 모델
- 특징: 단어 내부의 subword(부분단어, n-gram) 정보 활용
- 장점:
  - 오타 및 희귀 단어 처리에 강함: 문자 단위 n-gram 분해로 OOV 단어도 임베딩 가능
  - 형태소가 중요한 언어(한국어 등)에 유리: 공통 subword 공유로 의미적 유사성 반영
  - 학습 및 추론 속도가 빠름
  - 텍스트 분류에도 활용 가능

10. ELMo(Embeddings from Language Models)

- 문맥(Context)에 따라 단어 임베딩이 달라지는 특징
- 기존 LSTM과의 차이:
  - 딥(Deep) 양방향 LSTM 사용: 여러 층의 BiLSTM으로 깊은 문맥 정보 학습
  - 문맥 기반 임베딩: 문장 내 위치, 주변 단어에 따라 동적 생성
  - 사전학습 언어모델의 각 층 출력을 가중합하여 최종 임베딩 생성
- 필요성:
  - 고정 임베딩(Word2Vec, GloVe)의 한계 극복: 동음이의어, 다의어 구분
  - 문맥에 따라 의미가 달라지는 자연어 특성 반영
  - 다양한 NLP 태스크에서 성능 향상

11. BERT(Bidirectional Encoder Representations from Transformers)

- 2018년 구글 개발, 양방향 트랜스포머 인코더 기반 사전훈련 언어 모델
- 주요 특징:
  - 양방향성: Masked Language Model(MLM) 방식으로 양방향 문맥 학습
  - 사전훈련-미세조정: 대규모 코퍼스로 사전훈련 후 특정 태스크에 미세조정
  - 모델 구조: 트랜스포머 인코더 층 다층 구조(BERT-Base: 12층, BERT-Large: 24층)
- 사전훈련 방식:
  - MLM: 입력 텍스트 일부 토큰(15%) 마스킹 후 예측
  - NSP: 두 문장이 연속적인지 예측하는 이진 분류
- 응용: 텍스트 분류, 개체명 인식, 질의응답, 문장 쌍 분류, 텍스트 요약 등

12. Doc2Vec

- 문서(문장, 문단, 전체 문서) 단위의 임베딩 생성 기법
- Word2Vec과 차이:
  - Word2Vec: 단어 임베딩, 문맥 정보 제한적
  - Doc2Vec: 문서 전체 임베딩, 문맥+문서 전체 의미 반영
- 주요 원리: Word2Vec 구조 확장, 문서 고유 벡터(문서 태그) 추가 학습
  - Distributed Memory(DM) 모델: 문맥 단어 벡터 + 문서 벡터로 다음 단어 예측
  - Distributed Bag of Words(DBOW) 모델: 문서 벡터만으로 문서 내 임의 단어 예측
- 활용: 문서 분류, 유사 문서 검색, 문서 군집화 및 시각화

자연어처리(Natural Language Processing)

자연어처리란?
• 자연어(한국어, 영어 등과 같은 자연적으로 발생한 언어)의 이해, 생성 및 분석을 다루는 인공지능 연구의 한 분야
• 자연어 이해는 형태 분석, 통사 분석, 의미 분석 등을 통하여 자연어를 컴퓨터가 이해할 수 있는 형태로 변환하는 작업
• 자연어 생성은 컴퓨터가 처리한 결과물을 텍스트나 음성 등으로 변화하는 작업

자연어처리는 자연어 이해(NLU)와 자연어 생성(NLG)으로 구분됩니다.

NLU(자연어 이해)

- 감정 분석
- 개체 독해
- 상식 추론
- 의미론적 유사도 측정
- 목적 기반 대화
- 관계 추출
- 의미론적 구문 분석

NLG(자연어 생성)

- 자동 완성
- 스토리 생성
- 생성형 언어 모델
- 데이터 기반 문장 생성
- 캡션 생성

두 영역의 교집합으로는 생성형 문서 요약, 생성형 질의응답, E2E 챗봇 등이 있습니다.

1. 통계적언어모델 이란

통계적 언어 모델(Statistical Language Model)은 자연어 문장이나 단어의 시퀀스에 확률을 할당하는 모델입니다. 이는 특정 맥락에서 단어나 문장이 나타날 확률을 계산하여, 기계번역, 음성 인식, 맞춤법 검사, 문장 생성 등 다양한 자연어처리 작업에 활용됩니다. 단어의 빈도와 순서에 대한 통계적 정보를 학습하여 언어의 패턴을 모델링합니다.

2. N-gram 이란

N-gram은 연속된 N개의 항목(단어, 문자, 음절 등)을 하나의 단위로 처리하는 모델입니다.

- Unigram(1-gram): 단일 단어/문자만 고려 (예: "나", "는", "밥")
- Bigram(2-gram): 연속된 2개 단어/문자 (예: "나는", "는 밥", "밥을")
- Trigram(3-gram): 연속된 3개 단어/문자 (예: "나는 밥", "는 밥을", "밥을 먹는다")

N-gram 모델은 이전 n-1개 단어만을 고려하여 다음 단어의 확률을 예측하는 마르코프 가정(Markov assumption)을 기반으로 합니다. 간단하고 구현이 쉬우면서도 효과적이라는 장점이 있지만, 희소성 문제(sparse data problem)와 문맥 범위 제한이라는 단점도 있습니다.

3. Log처리 하는 이유?

자연어 처리에서 확률 계산 시 로그 함수를 사용하는 주요 이유는 다음과 같습니다:

1. 수치적 안정성(Numerical Stability): 확률값은 매우 작은 수(0~1 사이)이므로, 여러 확률을 곱하면 언더플로우(underflow) 문제가 발생할 수 있습니다. 로그를 취하면 곱셈이 덧셈으로 변환되어 이 문제를 해결할 수 있습니다.
   - $P(A)\times P(B)\times P(C) \rightarrow \log(P(A)) + \log(P(B)) + \log(P(C))$
2. 계산 효율성: 로그를 사용하면 곱셈 연산이 덧셈으로 바뀌어 계산 효율이 향상됩니다.
3. 확률 값의 비교: 매우 작은 확률값들을 비교할 때 로그 스케일에서는 차이가 더 명확하게 드러납니다.
4. 학습 안정성: 딥러닝에서 손실 함수로 자주 사용되는 cross-entropy는 로그 확률을 기반으로 하며, 경사 하강법에서 안정적인 학습을 가능하게 합니다.

로그 확률은 항상 음수값을 가지며(0~1 사이의 확률의 로그이므로), 값이 0에 가까울수록(즉, 로그값이 0에 가까울수록) 더 높은 확률을 의미합니다.

N-gram과 라플라스 스무딩

1. N-gram의 한계: Zero 확률 문제

N-gram 모델은 주어진 (n-1)개의 단어가 등장한 후, 특정 단어가 등장할 확률을 통계적으로 계산합니다. 하지만 훈련 데이터(코퍼스)에 등장하지 않은 단어 조합이 실제 문장에서는 나타날 수 있습니다.
이 경우, 해당 N-gram의 확률은 0이 되어 전체 문장 확률이 0이 되는 문제가 발생합니다.
이러한 현상을 **Zero 확률 문제** 또는 **희소성 문제(Sparse Data Problem)** 라고 합니다.

예시)

- 코퍼스에 "나는 밥을 먹는다"는 있지만, "나는 밥을 마신다"는 없다면,
  P(마신다 | 나는 밥을) = 0

2. 라플라스 스무딩(Laplace Smoothing, Add-one Smoothing)

Zero 확률 문제를 해결하기 위해 **라플라스 스무딩**을 사용합니다.
라플라스 스무딩은 모든 가능한 N-gram 조합에 1을 더해줌으로써, 등장하지 않은 조합에도 최소한의 확률을 부여합니다.

수식

- 기존 확률:
  $P(w_n \mid w_1, ..., w_{n-1}) = Count(w_1, ..., w_n) / Count(w_1, ..., w_{n-1})$
- 라플라스 스무딩 적용:
  $P(w_n \mid w_1, ..., w_{n-1}) = (Count(w_1, ..., w_n) + 1) / (Count(w_1, ..., w_{n-1}) + V)$
  여기서 V는 전체 단어(어휘)의 개수입니다.

예시

코퍼스: "나는 밥을 먹는다", "너는 밥을 먹는다"

- "밥을 먹는다"의 등장 횟수: 2
- "밥을 마신다"의 등장 횟수: 0
- 어휘(V): 5 (나는, 너는, 밥을, 먹는다, 마신다)

라플라스 스무딩 적용 후

- P(먹는다 | 밥을) = (2 + 1) / (2 + 5) = 3/7
- P(마신다 | 밥을) = (0 + 1) / (2 + 5) = 1/7

즉, 등장하지 않은 조합에도 0이 아닌 작은 확률이 부여되어, 실제 문장 생성이나 확률 계산에서 0 확률로 인해 전체 결과가 무의미해지는 것을 방지할 수 있습니다.

3. 라플라스 스무딩의 한계

- 모든 조합에 동일하게 1을 더하기 때문에, 어휘 수(V)가 클수록 실제로 등장한 조합의 확률이 과도하게 낮아질 수 있습니다.
- 실제로는 등장하지 않을 법한 조합에도 확률이 부여됩니다.

이러한 한계를 보완하기 위해 Add-k 스무딩(k를 1보다 작은 값으로 설정)이나 Good-Turing 추정 등 다양한 스무딩 기법이 존재합니다.

통계적 언어 모델에서의 한국어 난이도

한국어의 주요 특징

1. **교착어적 특성**: 한국어는 조사와 어미가 풍부한 교착어로, 어근에 다양한 접사가 결합하여 단어를 형성합니다.
   - 예: '집에서부터' = '집' + '에서' + '부터'
   - 영어와 달리 어순보다 조사가 문법적 기능을 결정
2. **형태소의 복잡성**: 한국어는 형태소 단위의 변형이 매우 다양합니다.
   - 불규칙 활용: '듣다 → 들어요', '짓다 → 지어요'
   - 음운 변화: '꽃이 → 꼬치', '밥을 → 바블'
3. **높은 문맥 의존성**: 한국어는 주어나 목적어가 생략되는 경우가 많아 문맥 의존도가 높습니다.
   - 생략 현상: "나는 밥을 먹었다" → "밥을 먹었다" → "먹었다"
4. **어순의 상대적 자유로움**: SOV(주어-목적어-동사) 기본 구조를 가지지만 조사 사용으로 어순 변경이 비교적 자유롭습니다.

통계적 언어 모델에서의 한국어 난이도

높은 난이도 요소

1. **어휘 희소성(Lexical Sparsity) 문제**
   - 형태소 결합으로 생성 가능한 단어 형태가 매우 많음
   - 예: '먹다'의 활용형 - 먹어요, 먹습니다, 먹었습니다, 먹을게요, 먹었었습니다 등
   - 결과: 코퍼스에서 각 단어 형태의 빈도가 낮아 통계적 모델의 정확도 저하
2. **형태소 분석의 복잡성**
   - 한국어 전처리는 형태소 분석이 필수적이며 이 과정에서 오류 발생 가능성 높음
   - 동음이의어와 중의성 문제: '감(感, 柑)'과 같은 단어의 구분
3. **OOV(Out-of-Vocabulary) 문제의 심각성**
   - 조사와 어미의 다양한 결합으로 학습 데이터에 없는 단어 형태 빈번히 발생
   - n-gram 모델에서 더 심각한 영향을 미침

실증적 근거

1. **언어 모델 성능 비교 연구**

- 동일한 크기의 학습 데이터로 훈련했을 때 영어 대비 한국어 언어 모델의 퍼플렉시티(perplexity)가 일반적으로 더 높음
- Kim et al. (2016)의 연구: 한국어는 영어보다 약 30-40% 높은 퍼플렉시티 보고

2. **토큰화 방식에 따른 성능 차이**
   - 형태소 단위 > 음절 단위 > 단어 단위 (한국어 특성상)
   - 영어는 단어/서브워드 단위가 효과적인 반면, 한국어는 형태소 분석이 필수적
3. **모델 크기 요구사항**
   - 동일한 성능에 도달하기 위해 한국어 모델이 더 많은 매개변수 필요
   - 형태적 복잡성을 학습하기 위한 더 큰 학습 데이터셋 요구

한국어 처리를 위한 접근법

1. **서브워드 토큰화의 활용**
   - BPE(Byte Pair Encoding), WordPiece, SentencePiece 등의 방법으로 OOV 문제 완화
   - 한국어의 형태소적 특성을 반영할 수 있는 토큰화 방식 필요
2. **문맥화된 임베딩 활용**
   - BERT, GPT 등의 사전 학습된 모델이 한국어의 문맥 의존성 처리에 효과적
   - KoBERT, KoGPT 등 한국어에 특화된 모델의 등장
3. **도메인 특화 데이터 활용**
   - 특정 도메인의 한국어 데이터로 추가 학습하여 성능 향상
   - 형태소 분석기의 정확도 향상을 위한 도메인별 사전 구축

결론

한국어는 교착어적 특성, 복잡한 형태소 변화, 높은 문맥 의존성으로 인해 통계적 언어 모델링에서 상대적으로 높은 난이도를 가집니다.
특히 어휘 희소성과 OOV 문제는 모델의 성능을 제한하는 주요 요인입니다.
이러한 도전과제들을 해결하기 위해서는 한국어의 언어학적 특성을 고려한 전처리 기법과 모델링 접근법이 필요합니다.
최근 딥러닝 기반 언어 모델의 발전으로 이러한 어려움이 일부 완화되고 있으나, 여전히 영어 등 다른 언어에 비해 처리가 더 복잡하고 많은 리소스를 요구합니다.

임베딩(Embedding)이란 무엇인가?

임베딩(Embedding)은 자연어 처리(NLP)에서 단어, 문장, 문서 등 텍스트 데이터를 고정된 크기의 실수 벡터로 변환하는 방법입니다. 임베딩을 통해 컴퓨터가 텍스트의 의미적, 문법적 특성을 수치적으로 이해할 수 있게 됩니다.

- **특징**
  - 고차원(One-hot, BoW 등) → 저차원(임베딩 벡터)으로 변환
  - 의미적으로 유사한 단어들은 임베딩 공간에서 가까운 위치에 배치됨
  - 대표적인 임베딩 방법: Word2Vec, GloVe, FastText 등

백오브워즈(Bag of Words, BoW) 가정

BoW는 문서를 단어들의 순서를 무시하고, 단어의 등장 빈도만을 고려하는 대표적인 텍스트 표현 방법입니다.

BoW의 주요 가정

가정 내용	설명
단어 순서 무시	문장 내 단어의 순서는 중요하지 않음. "나는 밥을 먹었다"와 "밥을 나는 먹었다"는 동일하게 처리됨.
단어 빈도만 고려	각 단어가 문서에 몇 번 등장했는지만 반영함. 단어의 위치, 문맥 등은 고려하지 않음.
문서의 길이 반영 가능	단어 빈도 벡터의 합을 통해 문서의 길이(단어 수)도 간접적으로 반영됨.

BoW 예시

아래는 두 개의 문서와 BoW 벡터화 과정을 표로 나타낸 예시입니다.

문서 번호	문장	단어 집합(어휘)	BoW 벡터 (나는, 밥을, 먹었다)
1	나는 밥을 먹었다	나는, 밥을, 먹었다	(1, 1, 1)
2	밥을 나는 먹었다	나는, 밥을, 먹었다	(1, 1, 1)
3	나는 밥을 먹지 않았다	나는, 밥을, 먹었다, 먹지, 않았다	(1, 1, 0, 1, 1)

- 위 표에서 볼 수 있듯이, BoW는 단어의 순서를 무시하고 각 단어가 몇 번 등장했는지만 벡터로 표현합니다.

BoW의 한계

- 단어의 순서와 문맥 정보를 반영하지 못함
- 의미적으로 유사한 단어(예: "먹었다", "식사했다")를 구분하지 못함
- 희소(sparse)하고 고차원 벡터가 생성됨

임베딩과 BoW의 관계

- BoW는 가장 단순한 임베딩(벡터화) 방법 중 하나입니다.
- 최근에는 단어 간 의미적 관계를 반영하는 임베딩(Word2Vec 등)이 BoW의 한계를 극복하기 위해 널리 사용됩니다.

마르코프 가정(Markov Assumption)은 n-gram 언어 모델 등에서 자주 사용되는 확률적 가정입니다. 이 가정은 "현재 상태(단어)는 오직 바로 앞의 n-1개 상태(단어)에만 의존한다"는 내용입니다. 즉, 더 이전의 정보는 무시하고, 최근의 정보만을 사용하여 다음 단어의 확률을 예측합니다.

왜 마르코프 가정을 하는가?

- **계산의 단순화**: 자연어는 매우 긴 문맥을 가질 수 있지만, 모든 이전 단어를 고려하면 계산량이 기하급수적으로 증가합니다. 마르코프 가정을 적용하면, 최근 n-1개 단어만 고려하므로 계산이 훨씬 단순해집니다.
- **데이터 sparsity(희소성) 문제 완화**: 모든 가능한 단어 조합의 확률을 추정하려면 방대한 데이터가 필요합니다. 마르코프 가정을 통해 필요한 데이터의 양을 줄일 수 있습니다.
- **모델 학습 및 구현 용이**: 실제로 n-gram 모델을 구현할 때, 마르코프 가정 덕분에 효율적으로 확률을 계산하고 저장할 수 있습니다.

근거

- **자연어의 지역성(Locality)**: 실제 언어에서도 대부분의 경우, 바로 앞의 몇 개 단어가 다음 단어에 가장 큰 영향을 미칩니다. 예를 들어, "나는 밥을" 다음에는 "먹었다"가 올 확률이 높습니다.
- **경험적 성능**: 마르코프 가정을 적용한 n-gram 모델이 실제로 많은 자연어처리 작업에서 좋은 성능을 보임.

한계

- **장기 의존성(Long-term dependency) 무시**: 문맥이 길어질수록, n-gram 모델은 중요한 정보를 놓칠 수 있습니다. 예를 들어, "비가 오면 길이 미끄럽다"에서 "비가 오면"과 "미끄럽다" 사이에 많은 단어가 끼어 있으면, n-gram 모델은 이 관계를 포착하지 못합니다.
- **차원의 저주(Curse of Dimensionality)**: n이 커질수록 필요한 파라미터(확률표)가 기하급수적으로 늘어나 데이터가 부족해집니다.
- **희소성 문제**: n이 커질수록 실제로 관측되지 않은 n-gram이 많아져 확률 추정이 어려워집니다.

리소스 및 차원의 저주

- n-gram의 n이 커질수록, 필요한 메모리와 계산량이 급격히 증가합니다. 이를 차원의 저주라고 하며, 실제로는 2-gram(빅그램)이나 3-gram(트라이그램) 정도만 실용적으로 사용합니다.
- 차원의 저주를 완화하기 위해 스무딩(smoothing) 기법(예: 라플라스 스무딩 등)을 사용합니다.

요약

마르코프 가정은 자연어처리에서 계산 효율성과 데이터 요구량을 줄이기 위해 도입된 중요한 가정입니다. 하지만 장기 의존성 포착의 한계와 차원의 저주 등 단점도 존재하므로, 최근에는 RNN, LSTM, Transformer 등 더 복잡한 모델이 등장하게 되었습니다.

Word2Vec의 네거티브 샘플링(negative sampling)과 이진분류(binary classification) 효과 설명

Word2Vec는 단어를 벡터로 임베딩하는 대표적인 방법 중 하나로, CBOW(Continuous Bag of Words)와 Skip-gram 두 가지 모델이 있습니다. 특히 Skip-gram 모델은 중심 단어로부터 주변 단어를 예측하는 방식입니다. 하지만 전체 단어 집합(vocabulary)이 매우 클 경우, 모든 단어에 대해 softmax를 계산하는 것은 연산량이 너무 많아 비효율적입니다.

이를 해결하기 위해 **네거티브 샘플링(negative sampling)** 기법이 도입되었습니다.

네거티브 샘플링이란?

- **핵심 아이디어**: 전체 단어 집합이 아닌, 일부(소수)의 '네거티브(음성) 샘플'만을 뽑아 이진 분류 문제로 단순화합니다.
- **방법**:
  - 실제로 주변에 등장한 단어(positive sample)는 1(참)로,
  - 임의로 뽑은 주변에 등장하지 않은 단어(negative sample)는 0(거짓)으로 레이블링합니다.
  - 모델은 중심 단어와 주변 단어 쌍이 실제로 문맥상 함께 등장하는지(1), 아닌지(0)를 구분하는 **이진 분류(binary classification)** 문제를 학습합니다.

이진분류를 통한 효과

- **계산 효율성**: 전체 단어 집합에 대해 softmax를 계산하지 않고, 소수의 네거티브 샘플에 대해서만 sigmoid(이진 분류)를 수행하므로 연산량이 크게 줄어듭니다.
- **학습 속도 향상**: 연산량 감소로 인해 대용량 코퍼스에서도 빠르게 학습할 수 있습니다.
- **임베딩 품질 유지**: 적절한 네거티브 샘플 수를 선택하면, 전체 softmax를 사용한 경우와 비슷한 품질의 임베딩을 얻을 수 있습니다.

수식적 설명

- 중심 단어 w_c와 주변 단어 w_o가 실제로 함께 등장하면 $D = 1$, 임의로 뽑은 단어면 $D = 0$.
- 확률은 sigmoid 함수로 계산:
  - $P(D = 1|w_c, w_o) = \sigma(\mathbf{v}_{w_c} \cdot \mathbf{v}_{w_o})$
  - $P(D = 0|w_c, w_k) = \sigma(-\mathbf{v}_{w_c} \cdot \mathbf{v}_{w_k})$

- 전체 손실 함수는 실제 쌍(positive)와 네거티브 샘플(negative)에 대해 각각 이진 분류 손실을 합산합니다.

요약

- **네거티브 샘플링**은 Word2Vec에서 전체 단어 집합 대신 일부 샘플만을 사용해 이진 분류 문제로 변환함으로써, 계산 효율성과 학습 속도를 크게 높여주는 핵심 기법입니다.
- 이진 분류를 통해 중심 단어와 주변 단어의 관계를 효과적으로 학습할 수 있습니다.

FastText란 무엇인가?

FastText는 Facebook AI Research(FAIR)에서 개발한 단어 임베딩(word embedding) 및 텍스트 분류(text classification) 모델입니다. Word2Vec과 유사하게 단어를 벡터로 변환하지만, **단어 내부의 subword(부분단어, n-gram)** 정보를 활용한다는 점에서 차별점이 있습니다.

FastText의 주요 특징 및 장점

1. **오타 및 희귀 단어 처리에 강함**
   - FastText는 단어를 문자 단위 n-gram(예: 3-gram, 4-gram 등)으로 분해하여 임베딩을 학습합니다.
   - 예를 들어, "시나브로"라는 단어는 ["<시나", "시나브", "나브로", "브로>"]와 같은 subword로 쪼개집니다.
   - 이 방식 덕분에 **오타가 있거나, 학습 데이터에 없는 희귀 단어(Out-of-Vocabulary, OOV)**도 subword 조합을 통해 임베딩 벡터를 생성할 수 있습니다.
   - 예시: "apple"과 "applle"(오타)도 상당히 유사한 subword를 공유하므로, 임베딩 벡터가 비슷하게 나옴.
2. **형태소가 중요한 언어(한국어 등)에 유리**
   - 한국어처럼 어미, 접두사, 접미사 등 다양한 형태소 변화가 많은 언어에서 subword 기반 임베딩이 효과적입니다.
   - 예: "먹다", "먹었다", "먹고", "먹는" 등은 공통된 subword("먹")를 공유하므로 의미적으로 가까운 벡터를 가짐.
3. **학습 및 추론 속도가 빠름**
   - Word2Vec과 유사한 구조이지만, 효율적인 구현 덕분에 대용량 코퍼스에서도 빠르게 학습할 수 있습니다.
4. **텍스트 분류에도 활용 가능**
   - FastText는 단어 임베딩뿐 아니라, 문서 분류, 감정 분석 등 다양한 텍스트 분류 작업에도 효과적으로 사용됩니다.

FastText의 구조적 특징

- **입력**: 문장을 단어로 분리한 뒤, 각 단어를 다시 문자 n-gram(subword)으로 분해
- **임베딩**: 각 subword에 대해 임베딩 벡터를 학습
- **단어 벡터 생성**: 단어의 subword 임베딩 벡터를 모두 합산(또는 평균)하여 최종 단어 벡터로 사용

요약

- FastText는 **subword(부분단어) 정보를 활용**하여 오타, 신조어, 희귀 단어 등에도 강인한 임베딩을 제공합니다.
- 한국어처럼 형태소 변화가 많은 언어에 특히 효과적입니다.
- 빠른 학습 속도와 텍스트 분류 등 다양한 자연어처리 작업에 활용할 수 있는 실용적인 임베딩 기법입니다.

ELMo(Embeddings from Language Models)란 무엇인가?

ELMo는 2018년 발표된 단어 임베딩 기법으로, 기존의 Word2Vec, GloVe 등과 달리 **문맥(Context)에 따라 단어 임베딩이 달라지는** 특징을 가집니다.
즉, 같은 단어라도 문장 내 위치나 주변 단어에 따라 임베딩 벡터가 다르게 생성됩니다.

기존의 단방향/양방향 LSTM과 ELMo의 차이

- **기존 LSTM**
  - 단방향 LSTM: 왼쪽(과거) → 오른쪽(미래) 방향으로만 정보를 전달
  - 양방향 LSTM(BiLSTM): 왼쪽→오른쪽, 오른쪽→왼쪽 두 방향의 LSTM을 따로 학습하여, 두 결과를 단순히 합침(concatenate)
  - **한계**:
    - BiLSTM은 입력 전체를 한 번에 보고, 각 단어의 양방향 정보를 합치지만,
      실제로는 각 방향의 LSTM이 독립적으로 동작함
    - 임베딩이 고정되어 있음(문맥에 따라 변하지 않음)
- **ELMo**
  - **딥(Deep) 양방향 LSTM**을 사용
    - 여러 층의 BiLSTM을 쌓아 더 깊은 문맥 정보를 학습
  - **문맥 기반 임베딩**
    - 각 단어의 임베딩이 문장 내 위치, 주변 단어에 따라 동적으로 생성됨
    - 예: "bank"가 "강가"일 때와 "은행"일 때 임베딩이 다름
  - **특징**
    - 사전학습된 언어모델(대규모 텍스트 코퍼스에서 미리 학습)에서 각 층의 출력을 가중합하여 최종 임베딩으로 사용
    - 다운스트림 태스크(문장 분류, 개체명 인식 등)에 맞게 임베딩을 조정할 수 있음

왜 ELMo가 필요했는가? (왜 달라져야 하는가?)

- **고정 임베딩의 한계**
  - Word2Vec, GloVe 등은 단어별로 하나의 벡터만을 가짐(문맥 무시)
  - 동음이의어, 다의어 구분 불가
    - 예: "나는 은행에 갔다"와 "강가의 은행"에서 "은행"의 의미가 다름에도 같은 벡터
- **문맥에 따라 의미가 달라지는 자연어의 특성**
  - 실제 언어에서는 단어의 의미가 주변 단어(문맥)에 따라 달라짐

    - 문맥을 반영한 임베딩이 필요
- **성능 향상**
    - 문맥 기반 임베딩을 사용하면, 개체명 인식(NER), 문장 분류 등 다양한 자연어처리 태스크에서 성능이 크게 향상됨

요약

- ELMo는 **딥 바이디렉셔널 LSTM**을 활용하여, 각 단어의 임베딩을 문맥에 따라 동적으로 생성하는 기법입니다.
- 기존의 (Bi)LSTM 기반 임베딩과 달리, **문맥 정보를 깊이 있게 반영**하여 자연어처리 성능을 크게 높였습니다.
- 이후 등장한 BERT, GPT 등 트랜스포머 기반 모델의 문맥 임베딩 개념의 시초가 되었습니다.

BERT (Bidirectional Encoder Representations from Transformers)

개요

BERT는 2018년 구글에서 개발한 사전 훈련된 언어 모델로, 양방향 트랜스포머 인코더를 기반으로 합니다. 기존의 단방향 언어 모델과 달리 문맥의 양쪽 방향을 모두 고려하여 단어의 의미를 파악할 수 있습니다.

주요 특징

1. 양방향성(Bidirectional)

- 기존 모델(ELMo, GPT 등)은 한쪽 방향으로만 문맥을 파악
- BERT는 Masked Language Model(MLM) 방식을 통해 양방향 문맥 학습 가능
- 문장 내 임의의 단어를 마스킹하고 이를 예측하는 방식으로 학습

2. 사전 훈련과 미세 조정(Pre-training and Fine-tuning)

- 대규모 코퍼스(위키피디아, BooksCorpus 등)로 사전 훈련
- 특정 태스크에 맞게 미세 조정하여 사용 가능
- 다양한 자연어 처리 태스크에 적용 가능한 범용성

3. 모델 구조

- 트랜스포머 인코더 층을 여러 개 쌓은 구조
- BERT-Base: 12개 층, 768 hidden units, 12 attention heads (110M 파라미터)
- BERT-Large: 24개 층, 1024 hidden units, 16 attention heads (340M 파라미터)

사전 훈련 방식

1. Masked Language Model (MLM)

- 입력 텍스트의 일부 토큰(약 15%)을 무작위로 마스킹
- 마스킹된 토큰을 예측하도록 학습

- 이 중 80%는 [MASK] 토큰으로 대체, 10%는 임의의 다른 단어로 대체, 10%는 원래 단어 유지

2. Next Sentence Prediction (NSP)

- 두 문장이 연속적인지 아닌지 예측하는 이진 분류 작업
- [CLS] 토큰을 사용하여 문장 쌍의 관계 학습
- 50%는 실제 연속된 문장, 50%는 무작위로 선택된 문장으로 구성

입력 표현

- 토큰 임베딩 + 세그먼트 임베딩 + 위치 임베딩의 합
- 토큰 임베딩: WordPiece 토크나이저로 분할된 단어 표현
- 세그먼트 임베딩: 문장 A와 B를 구분
- 위치 임베딩: 단어의 위치 정보 제공

BERT의 응용

- 텍스트 분류(감정 분석, 주제 분류 등)
- 개체명 인식(NER)
- 질의응답 시스템
- 문장 쌍 분류(자연어 추론, 의미 유사도 등)
- 텍스트 요약
- 기계 번역

BERT의 한계

- 생성 작업보다는 이해 작업에 더 적합
- 긴 시퀀스 처리에 제한(일반적으로 512 토큰 제한)
- 계산 비용이 높음
- 사전 훈련 후 지식 업데이트의 어려움

BERT 이후 발전 모델

- RoBERTa: 더 많은 데이터와 최적화된 학습 방식으로 BERT 개선
- DistilBERT: 경량화된 BERT 모델
- ALBERT: 파라미터 공유를 통해 효율성 증가
- ELECTRA: 더 효율적인 사전 훈련 방식 도입

Doc2Vec란 무엇인가?

Doc2Vec은 문서(Document) 단위의 임베딩을 생성하는 대표적인 딥러닝 기반 임베딩 기법입니다.
Word2Vec이 단어를 벡터로 변환하는 것과 달리, Doc2Vec은 **문장, 문단, 전체 문서**와 같은 더 큰 텍스트 단위를 고정 길이의 벡터로 변환합니다. 2014년 Le & Mikolov에 의해 제안되었습니다.

Word2Vec와 Doc2Vec의 차이

구분	Word2Vec	Doc2Vec
임베딩 대상	단어(Word)	문서(문장, 문단, 전체 문서)
입력	단어 시퀀스	문서(문장, 문단 등)
출력	각 단어의 벡터	각 문서의 벡터
활용 예시	유사 단어 찾기, 단어 군집화 등	문서 분류, 유사 문서 검색 등
한계	문맥 정보 제한적	문맥+문서 전체 의미 반영 가능

- **Word2Vec**: 각 단어를 고정된 벡터로 변환. 문장이나 문서의 의미를 파악하려면 여러 단어 벡터의 평균 등 추가적인 처리가 필요.
- **Doc2Vec**: 문서 전체의 의미를 하나의 벡터로 직접 표현. 문서 간 유사도 계산, 분류 등에 바로 활용 가능.

Doc2Vec의 주요 원리

Doc2Vec은 Word2Vec의 구조를 확장하여, **문서 고유의 벡터(문서 태그, document vector)**를 추가로 학습합니다. 대표적으로 두 가지 방식이 있습니다.

1. **Distributed Memory (DM) 모델**
   - 문맥 단어 벡터 + 문서 벡터를 합쳐서 다음 단어를 예측
   - 문서 벡터가 문맥 정보를 기억하는 역할
2. **Distributed Bag of Words (DBOW) 모델**
   - 문서 벡터만으로 문서 내 임의의 단어를 예측
   - Word2Vec의 Skip-gram과 유사

Doc2Vec의 활용 예시

- 문서 분류(뉴스, 이메일, 리뷰 등)
- 유사 문서 검색(질문-답변 매칭, 추천 시스템 등)
- 문서 군집화 및 시각화

요약

- **Word2Vec**: 단어를 벡터로 변환, 문맥 정보는 제한적
- **Doc2Vec**: 문서 전체를 벡터로 변환, 문서 의미를 직접적으로 반영
- Doc2Vec은 Word2Vec의 한계를 극복하여, 문서 단위의 자연어처리 작업에 효과적으로 사용됨