02 이론1 토큰화와 LSTM

02

토큰화와 LSTM

오늘 학습을 통해 우리는

- 자연어처리 모델링을 위한 텍스트 데이터 전처리 기법에 대해 알아봅니다.

- 텍스트를 수치로 바꾸기 위한 시도 중, 토큰화와 벡터화에 어떤 기법들이 있는지 탐구해봅니다.

- 자연어처리 모델의 기본이 되는 RNN, LSTM, GRU 셀에 대해 알아봅니다.

목차

토큰화와 LSTM

01

텍스트 데이터 전처리

텍스트 데이터의 특징

- 비정형성
  - 텍스트 데이터의 형태를 표현하기 위한 정형화된 필드나 칼럼이 없음
  - 이는 텍스트 데이터를 분석하거나 처리하기 위한 전처리 과정이 필수적이라는 것을 의미
- 고차원성
  - 텍스트 데이터는 일반적으로 고차원의 데이터
  - 이는 각각의 단어나 구문이 개별적인 차원을 형성하기 때문

텍스트 데이터의 특징

- 시퀀스성
  - 텍스트 데이터는 관점에 따라 시계열 데이터로 볼 수 있음
  - 단어나 문장들은 특정 순서에 따라 배열
  - 이 순서는 데이터의 의미에 중요한 역할을 반영함

- 이러한 특성들은 텍스트 데이터가 복잡한 구조를 가지며, 이를 처리하기 위해서는 특화된 기법과 알고리즘이 필요함을 보여줌

전처리 과정

- 텍스트 데이터 전처리는 원시 텍스트(Raw text) 데이터를 자연어 처리 작업에 적합한 형태로 변환하는 과정
- 전처리 과정은 다음의 주요 단계를 포함:
  - 정규화(Text regularization)
  - 불용어(Stopwords) 제거
  - 어간 & 표제어 추출

정규화(Text Regularization)

- 정규화는 일반적으로 전처리 과정의 첫 단계로 수행되며, 텍스트 데이터를 표준화된 형태로 변환하여 후속 처리 작업을 간소화
- 소문자 변환
  - 모든 텍스트를 소문자로 변환하여 모델이 동일한 단어를 다르게 인식하는 것을 방지
- 문장 부호 및 특수 문자 제거
  - 대부분의 경우, 문장 부호와 특수 문자는 의미 분석에 큰 도움을 주지 않음
  - 이들을 제거함으로써 텍스트 데이터의 크기를 줄이고, 텍스트 분석을 단순화할 수 있음

정규화(Text Regularization)

- 정규 표현식을 이용한 정제:
  - 텍스트 데이터에는 URL, 이메일 주소, 날짜, 전화번호 등 다양한 양식의 정보가 포함됨
  - 이러한 요소들은 종종 텍스트 분석에 방해가 될 수 있으므로, 정규 표현식을 사용하여 이들을 제거하거나 표준화된 형태로 변환함
- 유니코드 정규화
  - 유니코드 문자는 여러 가지 형태로 표현될 수 있음
  - 이로 인해 동일한 문자가 서로 다른 코드 포인트를 가지는 문제가 발생할 수 있음

불용어 제거

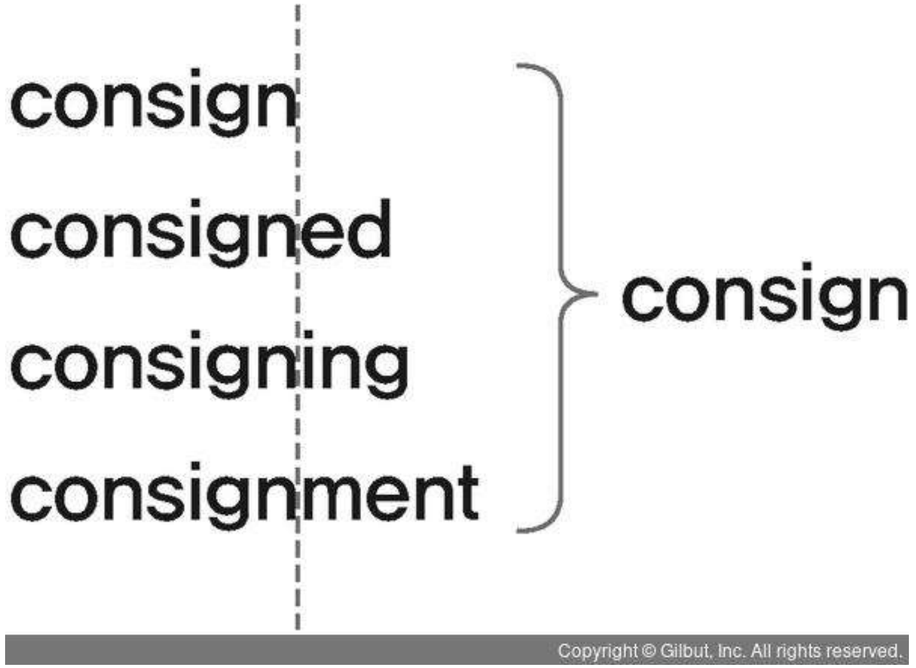
- 불용어(Stopwords)
  - 텍스트 내에서 빈번하게 등장하지만 실질적인 의미를 가지지 않는 단어를 의미
  - 예를 들어 영어에서 'the', 'a', 'is', 'in', 'at', 'which' 등은 불용어로 간주됨
  - 한글의 경우, 조사(은/는, 이/가, 을/를), 어미(-겠으나, -ㄹ지언정, -고서라도)가 해당
  - 이 외에도 언어마다, 사용되는 분야마다 불용어의 범위와 대상은 서로 다르므로 사용자가 이를 지정해주는 것이 최선

불용어 제거의 이유

- 효율성 향상:
  - 불용어는 일반적으로 텍스트 내에서 높은 빈도로 등장하며, 이로 인해 데이터의 차원 수를 불필요하게 높임
  - 이를 제거하면 텍스트 데이터의 크기를 줄일 수 있으며, 저장 공간과 처리 시간 절약
- 중요한 정보 강조:
  - 불용어를 제거함으로써, 데이터 내에서 중요한 정보를 더욱 강조할 수 있음
  - 이는 텍스트 분석이나 정보 검색과 같은 작업에서 특히 중요

어간 추출과 표제어 추출

- 어간 추출과 표제어 추출은 텍스트 데이터의 복잡성을 줄이기 위한 기법
- 텍스트 데이터 내의 단어를 그들의 기본 형태로 축소하는 역할을 함



출처 : https://thebook.io/080263/0482/

어간 추출과 표제어 추출

- 어간 추출:
  - 단어를 어간(단어의 기본 형태)으로 축소하는 과정
  - 이 과정은 단어의 접미사나 접두사를 제거하는 규칙 기반 방법을 사용
  - 예를 들어, 'jumps', 'jumping', 'jumped' 모두 'jump'로 축소될 수 있음
  - 한국어의 경우 용언(동사, 형용사)이 어간 + 어미 구조로 구성됨
    - 먹으므로, 먹고, 먹니, 먹지, 먹으며, 먹어서 → 먹다

어간 추출과 표제어 추출

- 표제어 추출
  - 단어를 그것의 표제어(사전 형식)으로 축소하는 과정
  - 어간 추출보다 복잡한 과정이며, 단어의 형태학적 분석을 통해 이루어짐
  - 예를 들어, 'is', 'am', 'are' 모두 'be'로 축소될 수 있음

어간 추출과 표제어 추출

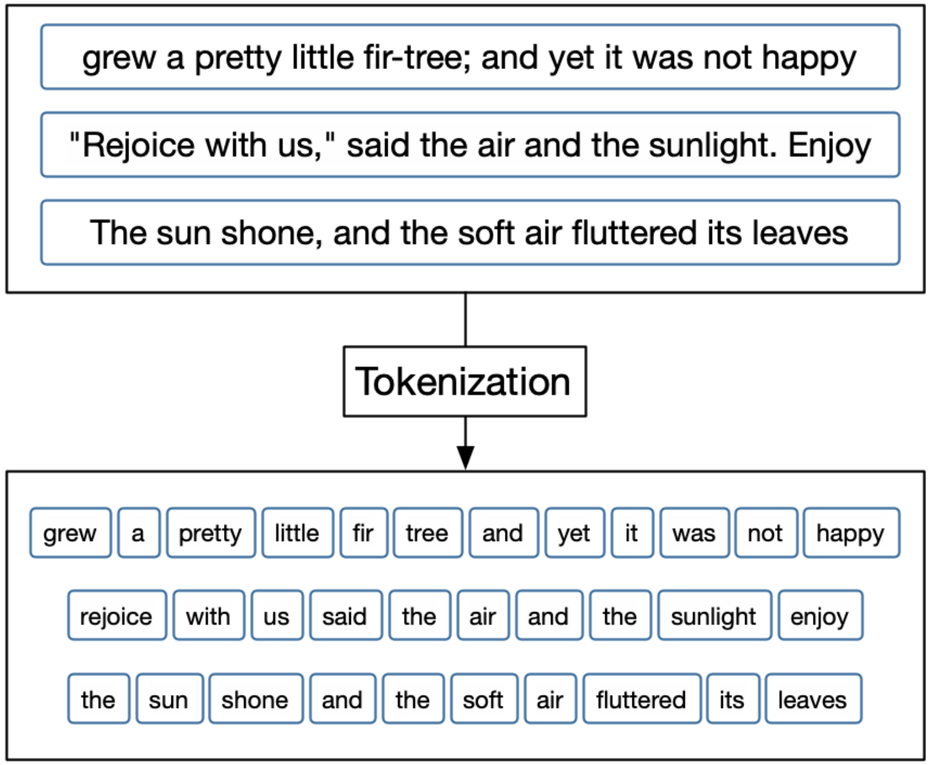
- 이러한 기법들은 텍스트 데이터의 차원 수를 줄이고, 단어의 다양한 변형이 같은 의미를 가진 것으로 인식되도록 함
- 어간 추출은 상대적으로 품질이 낮으나 간단하고 빠른 장점이 있고,
- 표제어 추출은 보다 정확하고 품질이 높은 결과를 제공함

02

토큰화

토큰화(Tokenization)란?

- 텍스트를 의미 있는 단위인 '토큰'으로 분리하는 과정



토큰화(Tokenization)란?

- 토큰화의 기대 효과
  - 텍스트 분석 용이성
    - 토큰화는 텍스트를 더 작은 부분으로 분리하여 분석을 용이하게 함
    - 분리된 텍스트는 단어 빈도 분석, 감정 분석, 텍스트 분류 등 다양한 자연어 처리 작업에 활용됨
  - 텍스트의 이해 보조
    - 토큰화는 텍스트의 구조를 이해하는데 도움을 줌
    - 예를 들어, 문장 토큰화는 텍스트를 개별 문장으로 분리하여 문장 간의 관계를 파악하는 데 도움이 됨

토큰화의 필요성

- 기계 학습 모델에 이해 가능한 형식으로 데이터 제공:
  - 대부분의 기계 학습 알고리즘은 숫자를 입력으로 사용
  - 토큰화는 텍스트를 이러한 알고리즘에 입력 가능한 형식, 즉 토큰으로 변환함
- 데이터의 차원 축소:
  - 텍스트를 토큰으로 하여, 기계 학습 모델이 처리해야 하는 데이터의 양을 축소
  - 이는 모델의 복잡성을 줄이고, 학습 속도를 높임

토큰화의 필요성

- 텍스트 데이터의 표준화:
  - 토큰화 과정에서는 일반적으로 정규화, 불용어 제거, 어간 추출 등의 작업도 함께 이루어짐
  - 이러한 작업들은 텍스트 데이터를 표준화하고, 데이터 내의 불필요한 변동성을 제거하는 데 도움이 됨

토큰화 방법(1) 단어 토큰화

- 공백, 탭, 쉼표, 마침표, 콜론 등의 문자를 구분자로 사용하여 텍스트를 분리
- 장점
  - 간단하고 빠른 방법
    - 단어 토큰화는 텍스트를 분리하는 가장 간단하고 직관적인 방법
    - 이용자가 고민없이 매우 빠르고 효율적으로 텍스트를 토큰화할 수 있음
  - 범용성이 높음
    - 단어는 텍스트의 기본 구성 요소이기 때문에, 텍스트 분석, 텍스트 분류, 감정 분석 등 다양한 작업에 일반적으로 사용될 수 있음

토큰화 방법(1) 단어 토큰화

- 단점
  - Out-of-Vocabulary (OOV) 문제: 단어 토큰화를 할 때, 모든 단어를 사전에 등록하거나 처리하는 것이 불가능함
  - 단어의 의미 표현 부족
    - 단어 토큰화는 문장을 단어 단위로 쪼개기 때문에, 맥락을 무시하며 단어 자체의 의미를 완전히 포착하기 어려움
    - 특히 한국어나 일본어와 같은 교착어에서는 단어 자체만으로는 정보가 부족해짐

토큰화 방법(1) 단어 토큰화

- 단점
  - 다의어 처리: 동음이의어, 다의어 등을 일반화할 위험이 존재
  - 특수 문자 처리: 특수 문자나 이모티콘 등은 단어 토큰화 과정에서 적절히 처리하기 어려움

- 이러한 단점을 해결하기 위해 다른 토큰화 기법이나 전처리 방법을 함께 활용함
  - Ex) Subword 토큰화 등

토큰화 방법(2) 형태소 토큰화

- 문장을 토큰화할 때, 형태소 단위로 토큰화하여 의미의 최소단위로 분할하는 방법
- 형태소: 최소의 의미를 가지는 가장 작은 단위
- 교착어(한국어, 터키어, 일본어 등)에서 주로 나타나는 단어의 구조

출처 : https://heung-bae-lee.github.io/2020/01/19/NLP_02/

토큰화 방법(2) 형태소 토큰화

- 높은 정보 밀도: 단어가 아닌 형태소로 토큰화하면 더 많은 정보를 얻을 수 있음
- 복합어 처리:
  - 복합어는 하나 이상의 단어로 구성되며 종종 새로운 의미를 가짐
  - 형태소 토큰화는 이런 복합어를 그 구성 요소로 분리함으로써 이를 처리할 수 있음
- 고유명사 및 신조어 처리
  - 이들은 종종 단어로는 인식되지 않지만, 형태소로는 인식될 수 있음

토큰화 방법(2) 형태소 토큰화

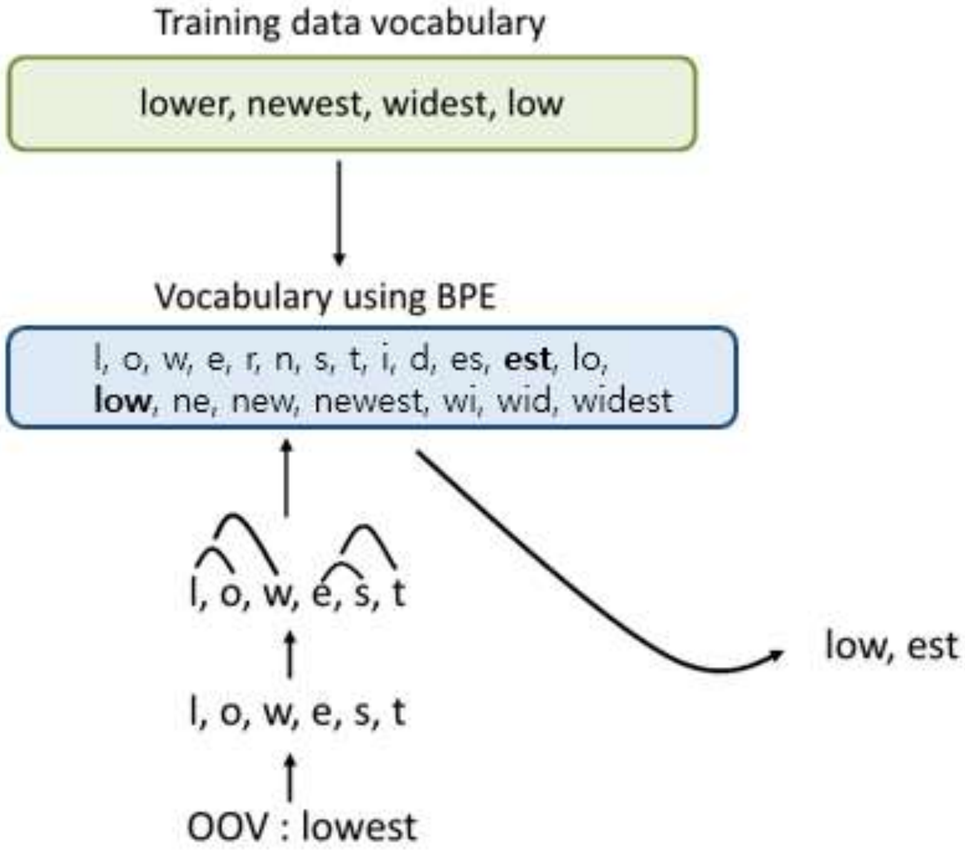
- 그러나 형태소 토큰화는 보통 복잡하고 시간이 많이 걸림
- 특히, 형태소 분석기가 없는 언어에서는 형태소 토큰화가 어려울 수 있음
  - 한국어의 경우 형태소 분석기가 존재는 하지만, 월등한 성능을 지녔다고 보기는 어려움

토큰화 방법(3) Subword 토큰화

- 단어를 더 작은 의미인 Subword로 쪼개는 방법
- 언어의 다양성과 다의어 문제를 해결하고
- Out-of-Vocabulary(OOV) 문제를 완화하는 데 도움이 됨
- 굴절어 뿐만 아니라 특히 교착어나 띄어쓰기가 없는 언어 등에서 효과적
- 대표적으로 BPE(Byte Pair Encoding)와 SentencePiece가 이에 속함

토큰화 방법(3) Subword 토큰화

- BPE(Byte Pair Encoding)
  - 데이터 압축 알고리즘에서 아이디어를 가져와 자연어 처리에 적용한 방법
  - 텍스트를 빈도수가 높은 Subword 쌍(pair)들로 반복적으로 합치고, 쪼개는 과정을 반복하여 최종적으로 사전을 만드는 방법



출처 : https://wikidocs.net/22592

토큰화 방법(3) Subword 토큰화

- SentencePiece
  - Google에서 제공하는 오픈 소스 라이브러리
  - 언어에 관계없이 Subword 토큰화 가능
  - 데이터를 기반으로 단어 빈도 수 계산 및 BPE 기반 학습 모듈 지원

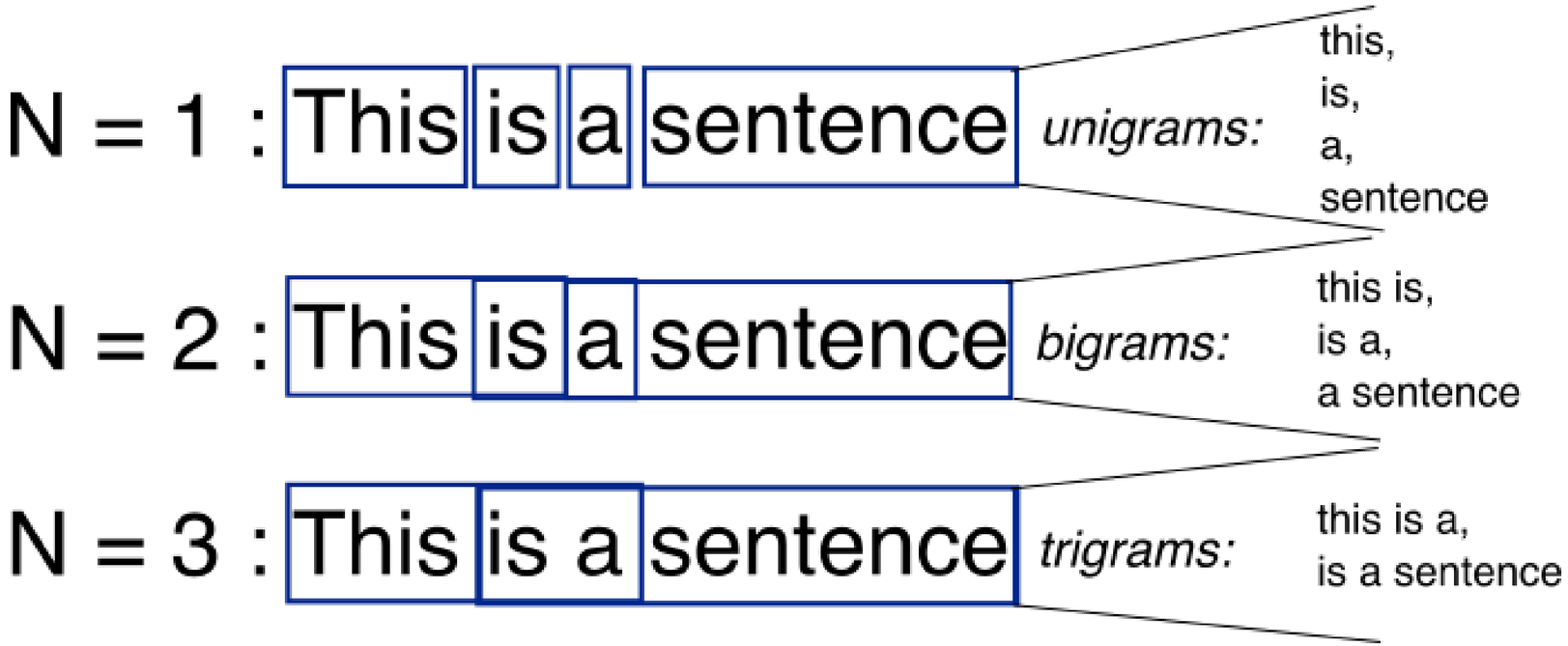
출처 : https://github.com/google/sentencepiece

토큰화 방법(3) Subword 토큰화

- 단점
  - 단어 토큰화보다 더 많은 연산과 처리가 필요하므로, 처리 시간과 복잡성이 증가
  - 또한, 일부 단어들이 매우 작은 단위로 쪼개지기 때문에 읽기가 어려운 상태가 될 수 있음
  - 이를 해결하기 위해 추가적인 후처리가 필요할 수도 있음

토큰화 방법(4) N-Gram 토큰화

- 텍스트를 연속된 n개의 단어(또는 문자)로 분리하는 방법
- 문맥을 고려한 토큰화 방법으로, n의 값을 조절하여 토큰의 길이를 제어할 수 있음



출처 : https://deepai.org/machine-learning-glossary-and-terms/n-gram

토큰화 방법(4) N-Gram 토큰화

- 장점
  - 문맥 파악:
    - N-gram 토큰화는 토큰 간의 순서를 유지하므로, 텍스트의 문맥을 보다 잘 파악할 수 있음
    - 이는 특히 감성 분석, 문서 분류 등의 작업에서 중요함
  - 미등록 단어 처리
    - N-gram은 특정 단어나 구문이 사전에 등록되지 않아도 처리할 수 있음
    - 이는 신조어, 오탈자, 철자 변형 등을 처리하는 데 유용함

토큰화 방법(4) N-Gram 토큰화

- 단점
  - 주어진 텍스트를 연속된 단위로 쪼개기 때문에, 단어의 의미를 완전히 이해하지 못할 수 있음
  - N-gram 토큰화는 빈도가 높은 단어를 잘 처리하지만, 희귀한 단어는 적절한 분리가 어려울 수 있고, 이로 인해 희귀한 단어가 OOV(out-of-vocabulary)로 처리됨
  - 높은 n 값을 사용하면 생성되는 N-gram 수가 급격히 증가하여 데이터 크기가 늘어남
    - 처리 시간이 증가하고 메모리 사용량이 늘어나는 문제 발생

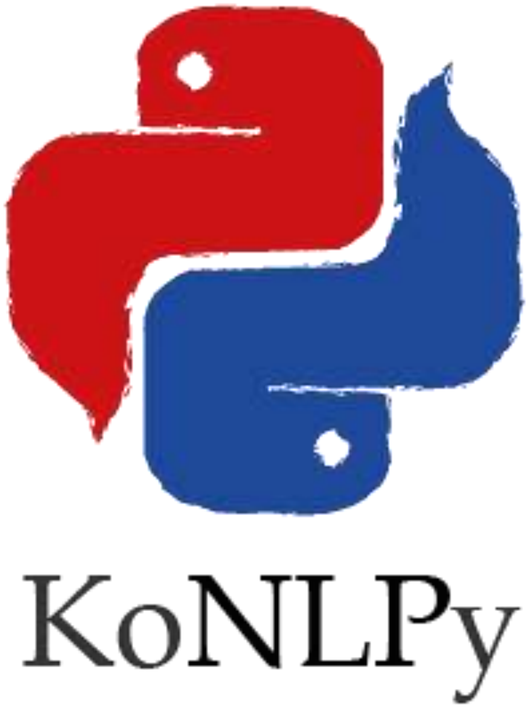
토큰화 라이브러리(1) NLTK(Natural Language Toolkit)

- 자연어처리 작업을 위한 파이썬 라이브러리
- 다음 토큰화 함수를 제공
  - word_tokenize: 공백과 구두점 기반의 단어 토크나이저
  - RegexpTokenizer: 정규표현식 기반 토크나이저
- 그 외에도 품사 태깅, 구문 분석 등을 위한 다양한 NLP API 제공

출처 : https://medium.com/@kelsklane/the-natural-language-toolkit-what-is-it-8ca335dce6a9

토큰화 라이브러리(1) KoNLPy

- 한국어 정보처리를 위한 파이썬 라이브러리
- 한국어의 특성을 고려한 다양한 형태소 분석기 제공
  - Kkma: 국민대에서 개발한 형태소 분석기. 토큰화, 품사 태깅, 구문 분석 등 다양한 기능을 제공
  - Komoran: Kkma에 비해 분석 속도가 빠름
  - Hannanum: KAIST에서 개발한 형태소 분석기
  - Mecab: 일본어 형태소 분석기인 MeCab을 한국어에 맞게 수정하여 배포



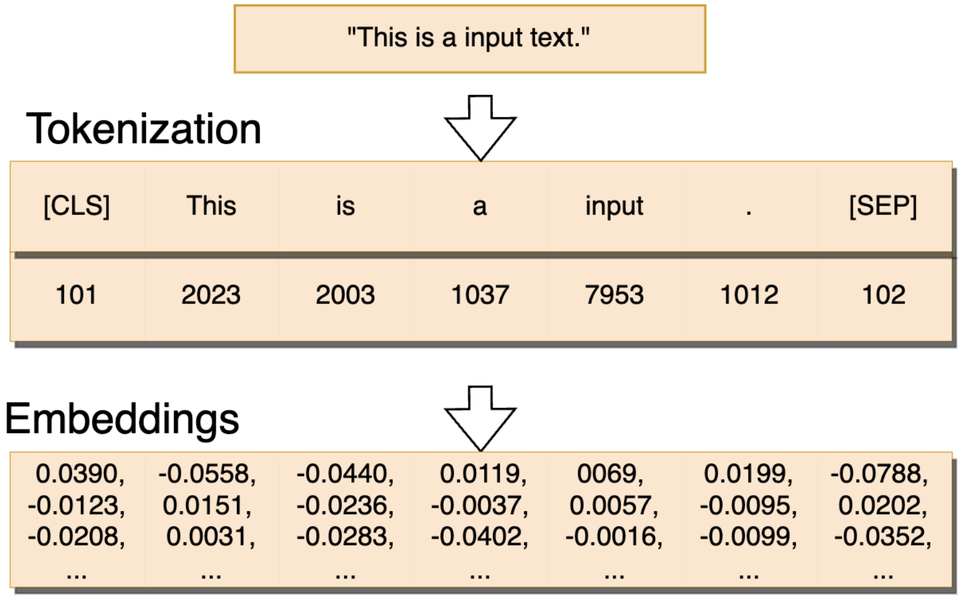
출처 : https://konlpy.org/ko/latest/index.html

03

벡터화

벡터화(Vectorization)란?

- 벡터화의 주요 목적은 텍스트 데이터를 기계 학습 알고리즘이 이해할 수 있는 형태로 변환하는 것
- 단어, 문장, 문서 등 다양한 수준의 텍스트 데이터를 벡터로 표현할 수 있음



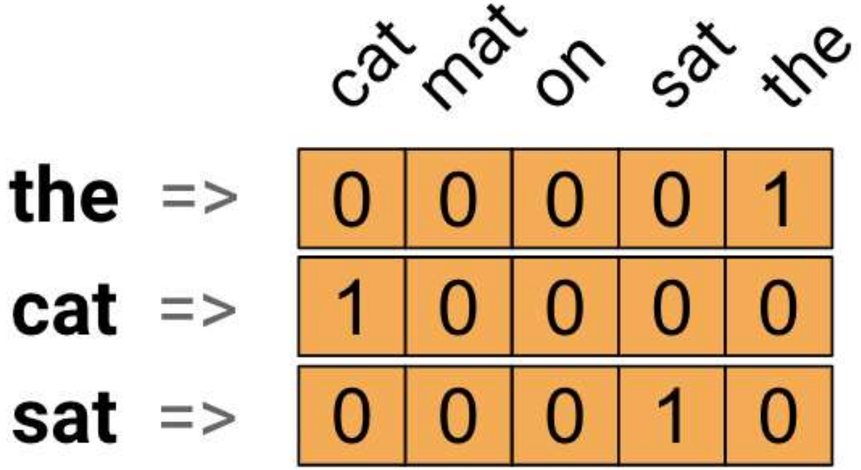
출처 : https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained

벡터화의 기술적 구분

- 크게 '카운트 기반'과 '예측 기반'으로 나뉨
- 카운트 기반의 방법: 텍스트의 단어 빈도를 기반으로 벡터를 생성
  - 대표적인 방법으로는 BoW(Bag of Words)와 TF-IDF(Term Frequency-Inverse Document Frequency)가 있음
- 예측 기반의 방법: 주변 단어를 바탕으로 단어를 예측하는 방식으로 벡터를 생성
  - 대표적인 방법으로는 Word2Vec, GloVe, FastText가 있음.

벡터화 방법(1) One-Hot Encoding

- 특정 단어를 표현하는 벡터에서 그 단어에 해당하는 인덱스만 1이고 나머지는 모두 0인 벡터를 생성하는 방법
- 단어장(Vocabulary)의 크기를 벡터의 차원으로 하며, 각 단어는 단어장에서의 위치(Index)에 따라 하나의 차원에 1의 값을 가짐



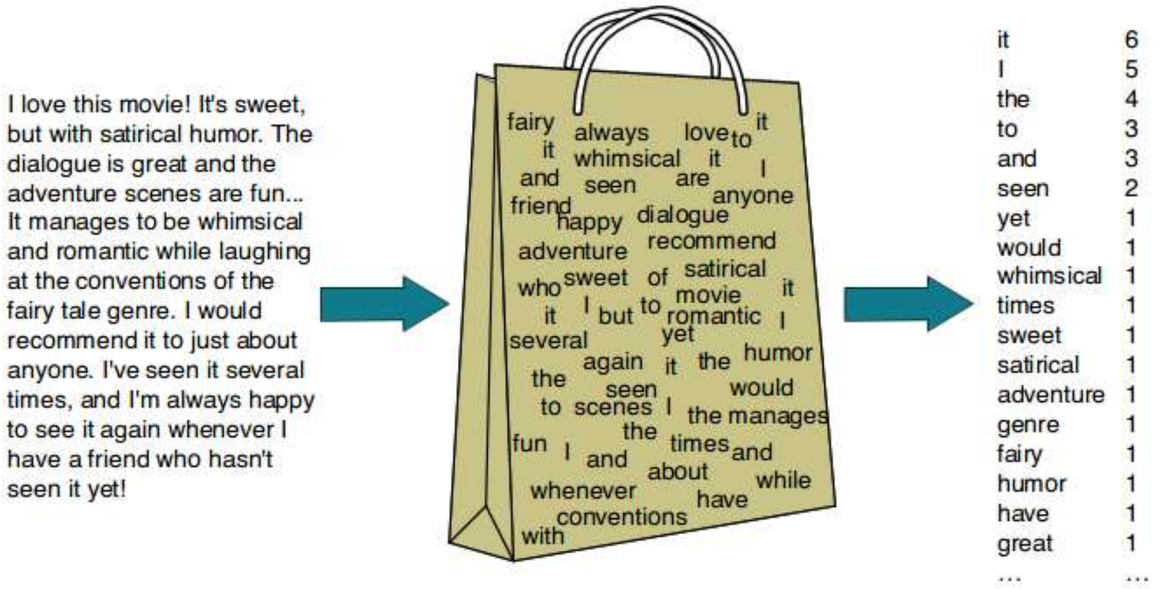
출처 : https://www.tensorflow.org/text/guide/word_embeddings

벡터화 방법(1) One-Hot Encoding

- 구현이 쉽고 이해하기 직관적
- 그러나 세 가지 큰 문제가 발생
  - 차원의 저주: 단어장의 크기가 클 경우 벡터의 차원이 지나치게 커지며 연산에 불리한 구조를 갖게 됨
  - 희소 벡터(Sparse vector) 문제: 대부분의 값이 0으로 표현되기에 연산의 결과에 0이 많이 포함됨
  - 단어 간 유사성을 전혀 표현하지 못함

벡터화 방법(2) BoW(Bag of Words)

- 문서의 내용을 단어의 집합으로 나타내는 방식
  - 단어를 토큰화한 뒤, 중복을 제거하여 단어 사전을 구성
  - 사전을 바탕으로 문장에 사전 속 단어가 등재되었는지를 희소 벡터(Sparse vector)로 표현



추천 : https://koushik1102.medium.com/nlp-bag-of-words-and-tf-idf-explained-fd1f49dce7c4

벡터화 방법(2) BoW(Bag of Words)

- Ex)
  - 문서 1: "I love natural language processing."
  - 문서 2: "Natural language processing is a field of artificial intelligence.”
  - 단어 사전(Vocab): ["I", "love", "natural", "language", "processing", "is", "a", "field", "of", "artificial", "intelligence"]
  - 문서 1 벡터: [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
  - 문서 2 벡터: [0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]

벡터화 방법(3) TF-IDF(Term Frequency-Inverse Document Frequency)

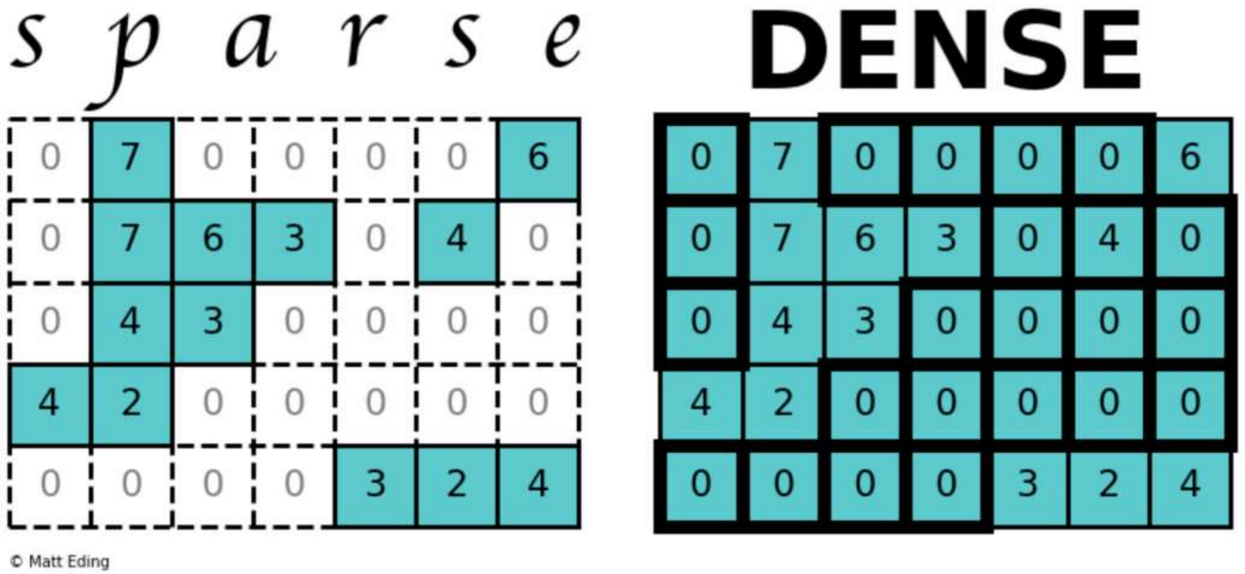
- TF(Term Frequency)
  - 특정 단어가 하나의 문서 내에서 등장하는 빈도
  - 이는 단어가 문서 내에서 얼마나 중요한지를 나타내는 지표
- IDF(Inverse Document Frequency)
  - 특정 단어가 등장하는 문서의 수에 반비례하는 수치
  - 모든 문서에서 자주 등장하는 단어는 중요도가 낮다고 판단
  - 특정 문서에서만 자주 등장하는 단어는 중요도가 높다고 판단

벡터화 방법(3) TF-IDF(Term Frequency-Inverse Document Frequency)

- TF-IDF:
  - TF와 IDF를 곱한 값
  - 이 값이 높을수록 단어의 중요도가 높다고 판단
  - 이 방식을 통해 텍스트를 수치 벡터로 변환하면, 이 벡터는 단어의 빈도와 문서 내에서의 중요도를 동시에 반영하게 됨
  - 텍스트 분류, 정보 검색 등 다양한 자연어 처리 작업에서 널리 사용되며, Sklearn, Gensim 등의 라이브러리에서 쉽게 구현할 수 있음

Word Embedding

- 단어의 의미를 벡터 공간에 표현하는 방법
  - 희소 벡터(Sparse vector)가 아닌 밀집 표현(Dense representation)
  - 이 벡터 공간에서는 위치가 가까운 단어들이 의미적으로 비슷한 경향을 보임



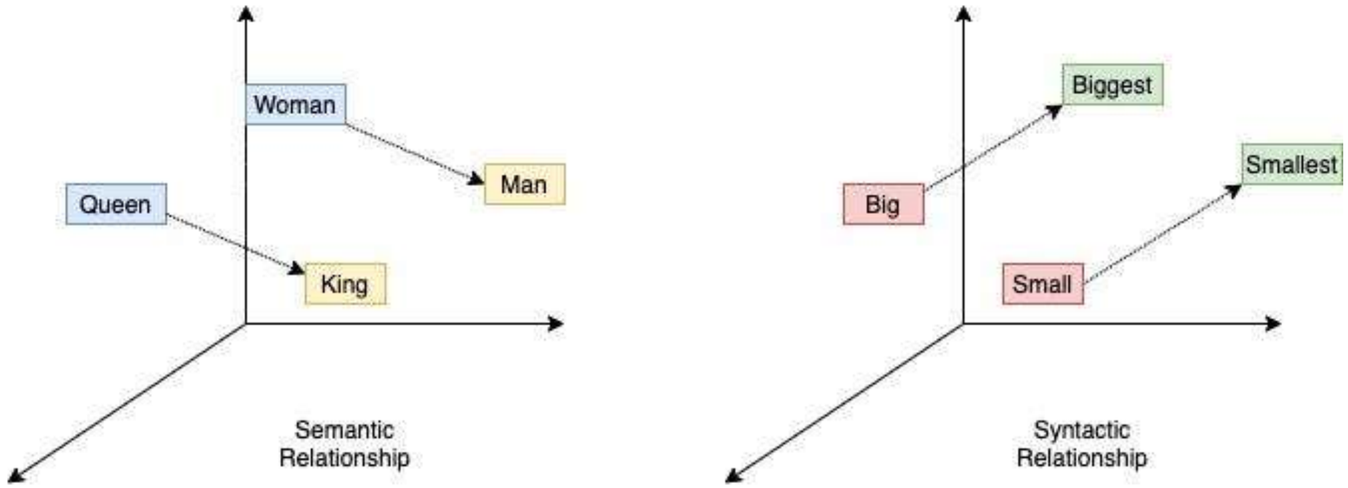
출처 : https://www.quora.com/What-is-the-definition-of-sparse-and-dense-representation-in-machine-learning-Why-are-they-used

Word Embedding

- One-hot encoding 방식이나 카운트 기반 방식에 비해 단어 간의 의미적 관계를 잘 표현할 수 있음
- 더 작은 차원에서 텍스트 데이터를 표현 가능
- 대량의 텍스트 데이터로부터 자동으로 학습할 수 있어, 사람이 수동으로 Feature를 설계하는 것에 비해 더 효과적

벡터화 방법(4) Word2Vec

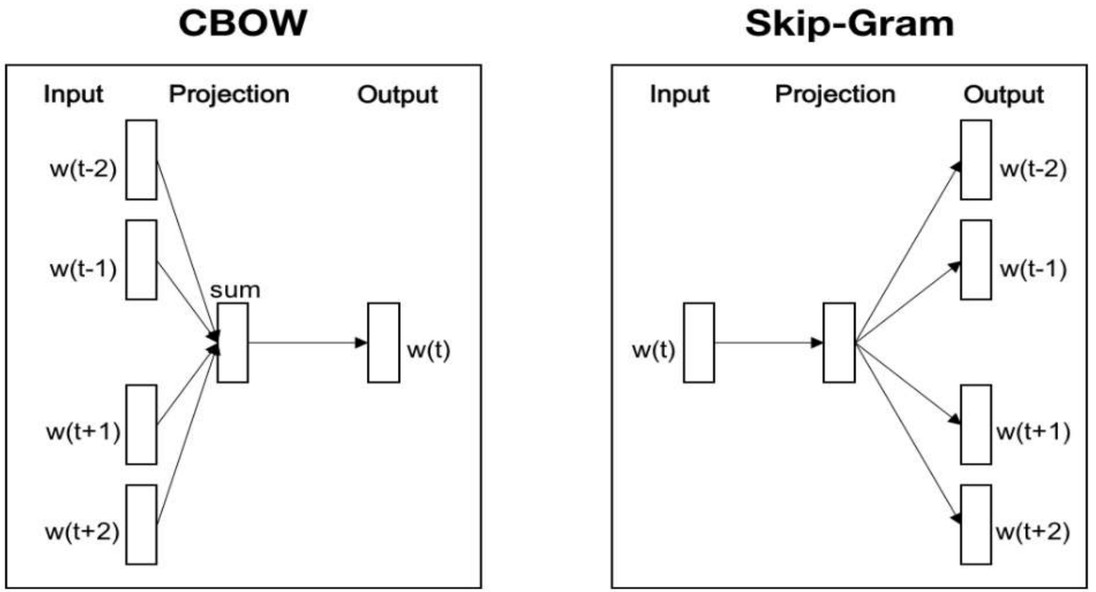
- Word2Vec
  - 단어의 의미를 벡터 공간에 임베딩하는 기술
  - 비슷한 위치에서 등장하는 단어들은 비슷한 의미를 가진다는 분포 가설(Distributional Hypothesis)에 근거



출처 : https://towardsdatascience.com/word2vec-research-paper-explained-205cb7eecc30

벡터화 방법(4) Word2Vec

- 학습 방식에 따라 두 가지 아키텍처가 존재
  - CBOW(Continuous Bag of Words): 주변 단어들을 이용하여 중심 단어를 예측하는 방식.
  - Skip-gram: 중심 단어를 이용하여 주변 단어들을 예측하는 방식.



출처 : https://wooono.tistory.com/244

벡터화 방법(4) Word2Vec

- Word2Vec을 통해 학습된 단어 벡터는 단어 간의 유사성을 잘 표현하며, 벡터 공간에서 의미적으로 유사한 단어는 가까이 위치함
- 또한, 단어 벡터 간의 연산을 통해 의미적인 관계를 파악할 수 있음
  - 예를 들어, "왕 - 남자 + 여자 = 여왕"과 같은 연산이 가능
- Word2Vec는 Gensim, TensorFlow 등의 라이브러리를 통해 쉽게 구현할 수 있음

벡터화 방법(5) FastText

- Word2Vec을 기반으로 하는 벡터화 방법
- FastText는 각 단어를 n-gram의 집합으로 보고 이를 바탕으로 벡터를 학습
  - Word2Vec의 단점 중 하나인 단어 단위 학습만 가능하다는 점을 보완

출처 : https://amitness.com/2020/06/fasttext-embeddings/

벡터화 방법(5) FastText

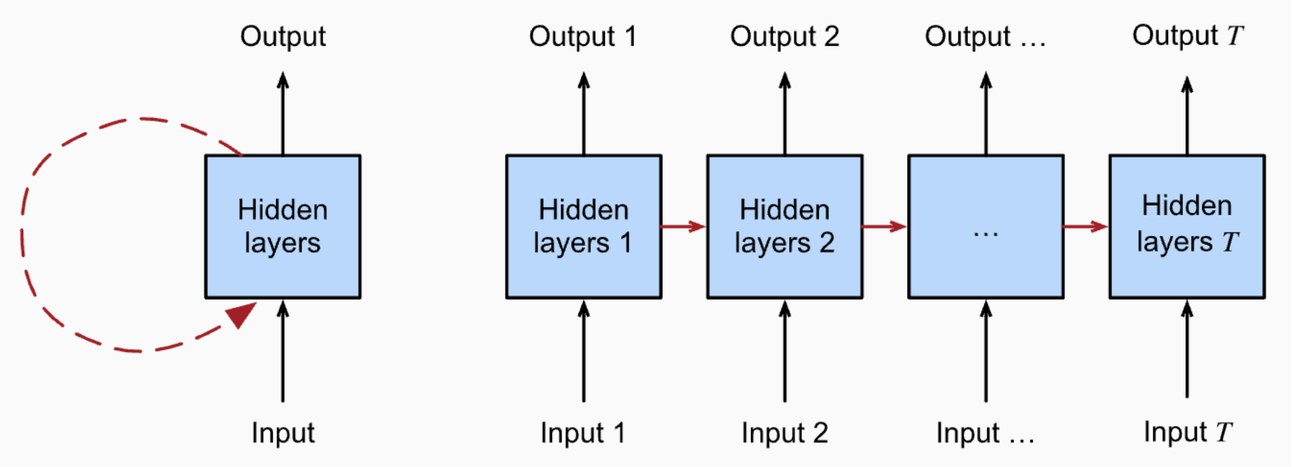
- 단어의 내부 구조 정보를 학습할 수 있어 특히 오타나 단어 형태의 다양성이 많은 언어(교착어)에 유용함
- 단어가 아닌 N-gram 단위로 학습하므로 빠른 학습이 가능
  - 단어들이 Subword들의 조합으로 표현되기 때문에 훨씬 더 적은 수의 Subword에 대한 임베딩을 학습
- Out-Of-Vocabulary(OOV) 문제에 대해서도 잘 대처할 수 있음

04

RNN

RNN(Recurrent Neural Network)

- 시퀀스(Sequential) 데이터를 처리하는 데 특화된 인공 신경망
  - 시퀀스 데이터: 순서가 의미에 영향을 주는 데이터(음성, 텍스트, 시계열 등)
- 자연어를 다룬 초기 딥러닝 모델의 구성 뼈대
  - Transformer 등장 전까지 대부분의 자연어처리 모델을 구성



출처 : https://d2l.ai/chapter_recurrent-neural-networks/index.html

RNN의 특성

- 순차적인 데이터 처리: 시퀀스 데이터에서 강력한 성능을 발휘

- 변동적인 입력/출력 길이:
  - RNN은 시퀀스의 길이에 상관없이 입력과 출력을 처리할 수 있음
  - 문장의 번역, 문장 생성, 감성 분석 등 다양한 NLP 작업에 사용

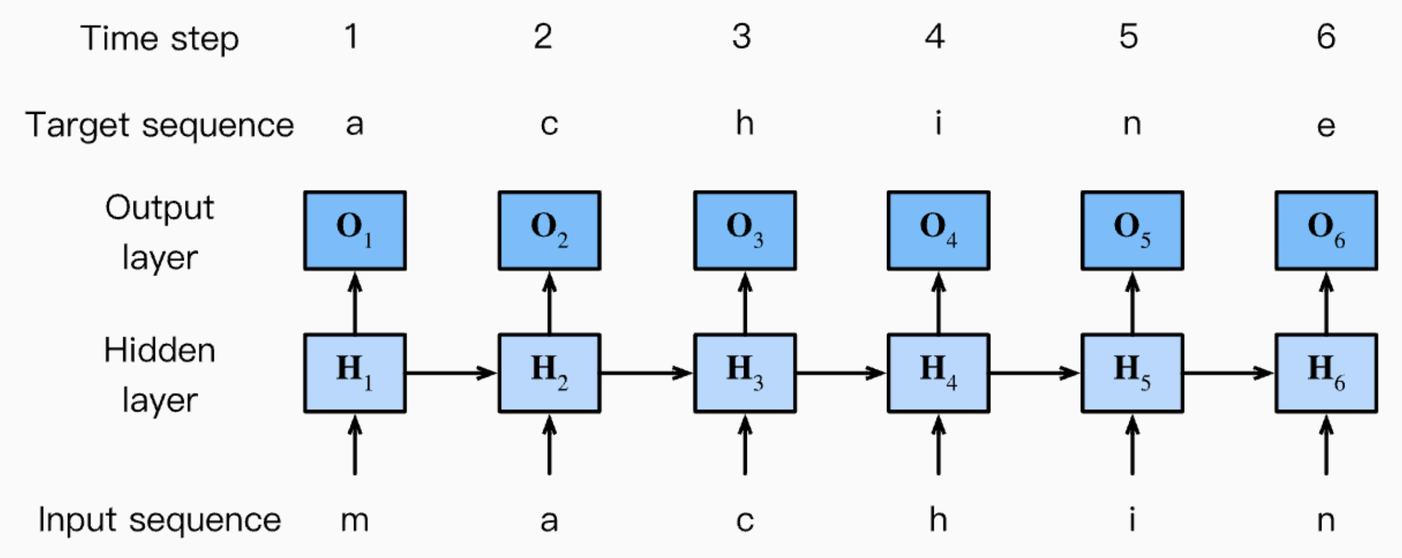
RNN의 특성

- 내부 메모리 사용: RNN은 내부에 상태(state)를 유지하며, 이 상태를 통해 과거의 정보를 기억하고 현재의 출력에 반영

- 시간에 따른 역전파(Backpropagation Through Time, BPTT):
  - RNN은 시간에 걸쳐 역전파를 수행하여 그래디언트를 계산하고 모델 파라미터를 업데이트함
  - 이는 계산 비용이 높은 작업이지만, 이를 통해 시퀀스의 정보를 학습

RNN의 구조

- 입력층(Input Layer):
  - 각 시점(Timestep)에서 시퀀스의 특정 요소를 입력으로 받음
  - 입력 데이터는 일반적으로 워드 임베딩과 같은 벡터 형태



출처 : https://d2l.ai/chapter_recurrent-neural-networks/index.html

RNN의 구조

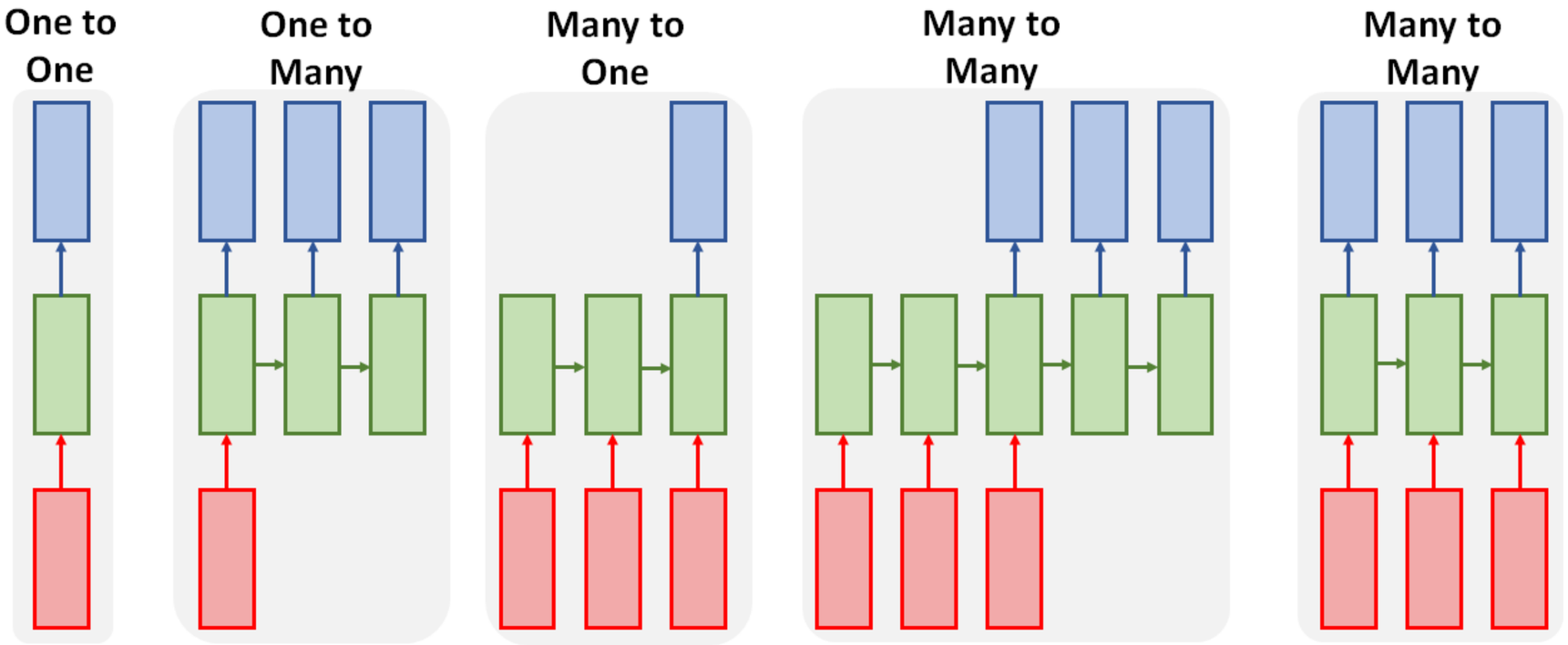
- 은닉층(Hidden Layer):
  - RNN의 핵심은 순환적인 은닉층임
  - 이전 시점의 출력(상태)와 현재 시점의 입력을 받아 처리하고, 그 결과를 다음 시점의 은닉층에 전달
  - 이런 과정을 통해 RNN은 시퀀스의 정보를 메모리에 저장하고 순차적인 패턴을 학습

RNN의 구조

- 출력층(Output Layer):
  - 각 시점에서 은닉층의 출력은 다음 시점의 은닉층 입력으로 사용될 뿐만 아니라, 최종적인 출력을 생성하는 데도 사용됨
  - 출력층에서는 은닉층의 출력을 바탕으로 원하는 작업을 수행

RNN의 다양한 형태

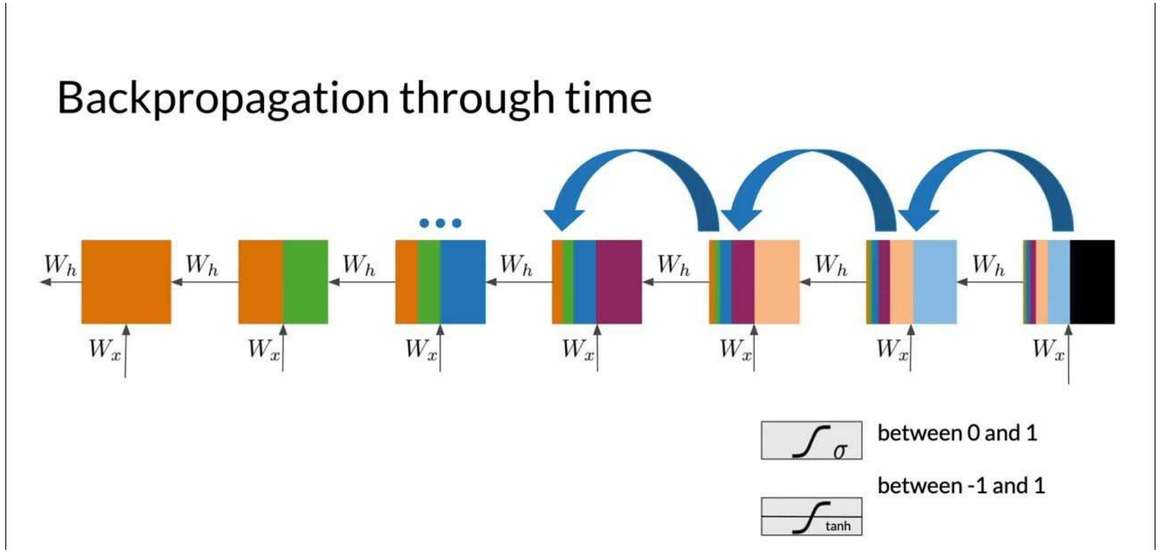
- 모델의 목적에 따라 다향한 형태로 구현할 수 있음



출처 : https://towardsdatascience.com/recurrent-neural-networks-56e1ad215339

RNN의 문제점

- 장기 의존성 문제(Long-Term Dependencies Problem):
  - RNN은 시퀀스의 앞부분에서 일어난 사건을 뒷부분에서 기억하는 데 어려움이 있음
  - 시퀀스가 길어질수록 이 문제가 더욱 심각해짐

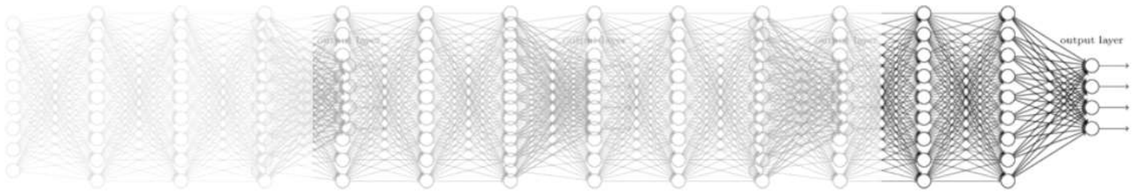


출처 : https://www.youtube.com/watch?v=NgxMUHTJYmU

RNN의 문제점

- 그래디언트 소실과 폭발(Vanishing and Exploding Gradients Problem):
  - RNN에서는 각 시점에서 오차 역전파를 통해 그래디언트를 계산함(BPTT)
  - 하지만, 시점이 많아질수록 그래디언트가 소실되거나 폭발하는 문제가 발생
  - 효과적으로 학습되지 못하고 성능이 저하될 수 있음

Vanishing gradient (NN winter2: 1986-2006)



출처 : https://velog.io/@yunyoseob/Gradient-Vanishing-%EA%B8%B0%EC%9A%B8%EA%B8%B0-%EC%86%8C%EC%8B%A4

RNN의 문제점

- 학습 속도:
  - RNN은 시간에 따른 역전파(Backpropagation Through Time, BPTT)를 사용하여 학습
  - 시퀀스 길이가 길어질수록 계산 비용이 크게 증가함

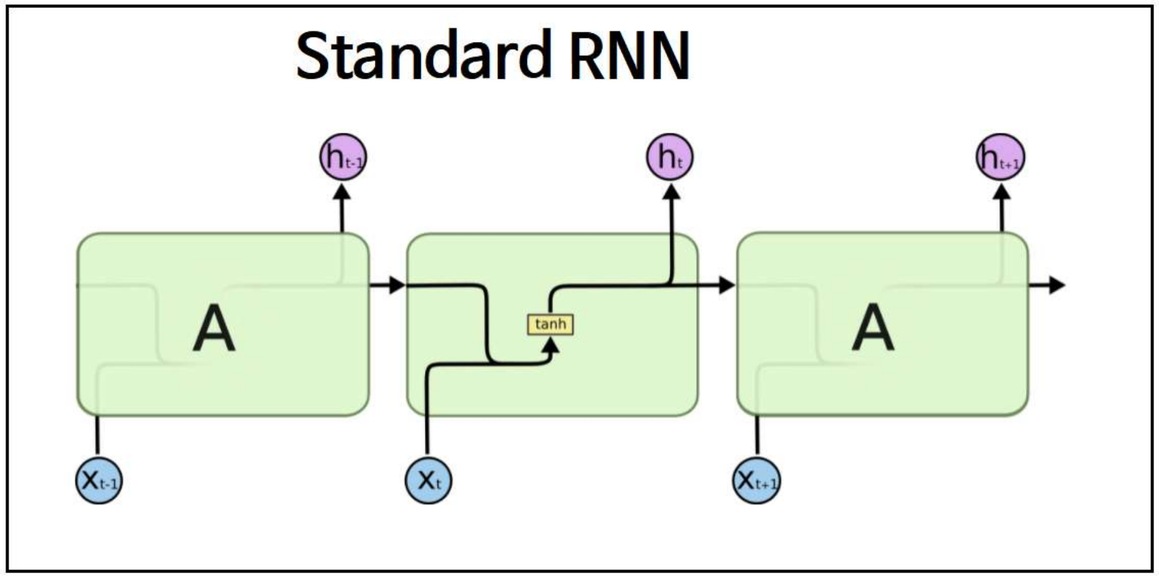
- 병렬 처리 불가능
  - RNN의 순차적인 특성 때문에, 다른 인공 신경망과 달리 RNN은 데이터를 병렬로 처리하는 것이 불가능

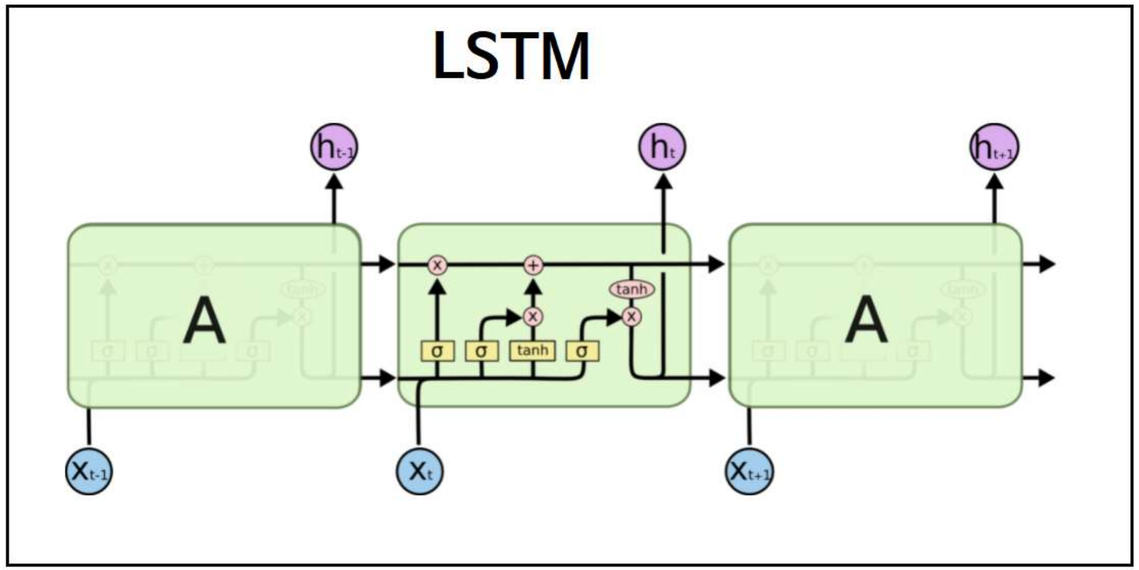
05

LSTM & GRU

LSTM(Long Short-Term Memory)

- RNN의 장기 의존성 문제를 해결하기 위해 설계된 신경망 구조





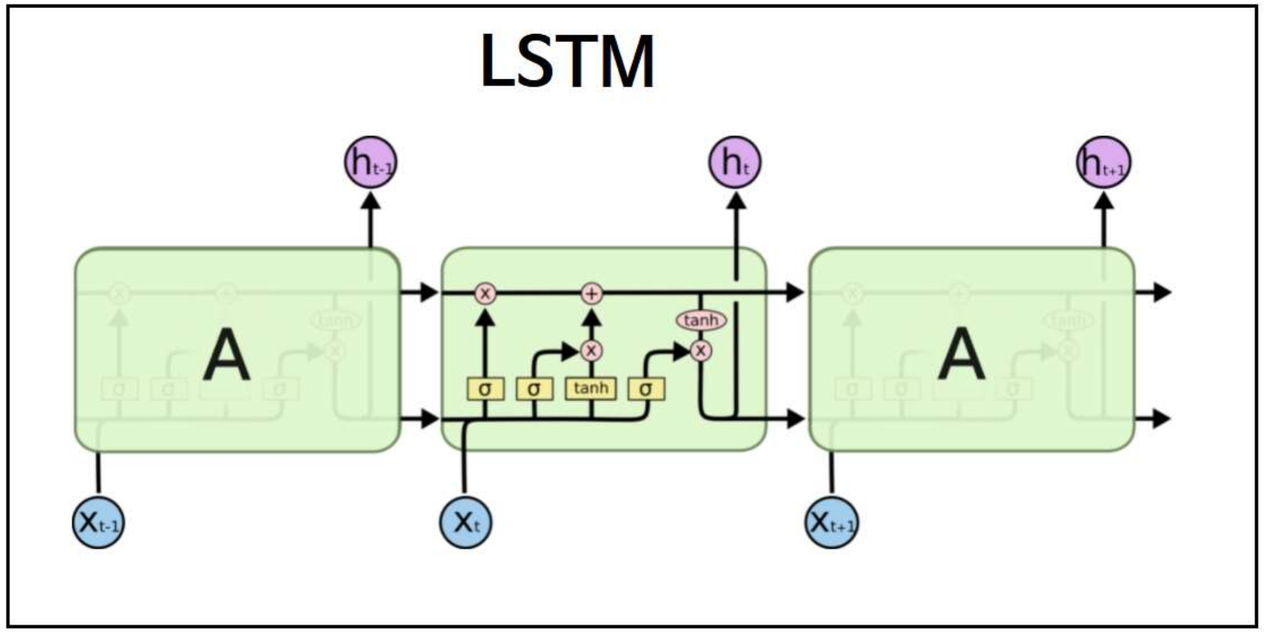
출처 : https://hyen4110.tistory.com/25

LSTM의 특징

- 장기 의존성 문제 해결:
  - LSTM의 Cell state는 필요한 정보를 장기간 유지하는 데 도움을 줌
- 복잡한 구조:
  - 복잡성은 LSTM의 강력한 성능을 보장하지만, 모델을 이해하고 구현하는 데는 상대적으로 높은 수준의 이해가 필요
  - LSTM은 기본 RNN에 비해 복잡한 만큼 학습 속도가 느리고 계산 비용이 더 많이 듦

LSTM의 구조

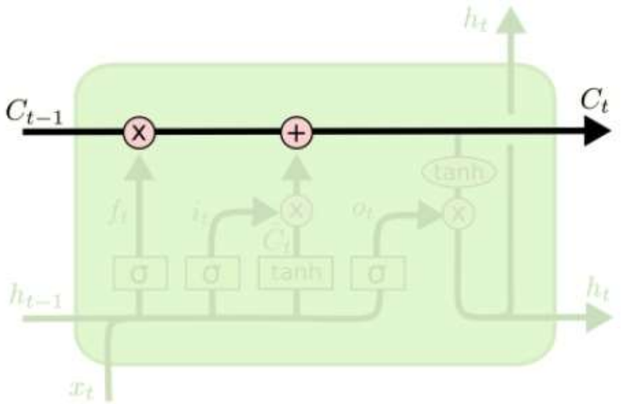
- RNN과 LSTM의 가장 큰 구조적 차이는 Cell state
- Cell state를 업데이트하기 위하여 세 가지 게이트(Gate)를 사용
  - 입력 게이트(Input gate)
  - 망각 게이트(Forget gate)
  - 출력 게이트(Output gate)



출처 : https://hyen4110.tistory.com/25

LSTM의 구조

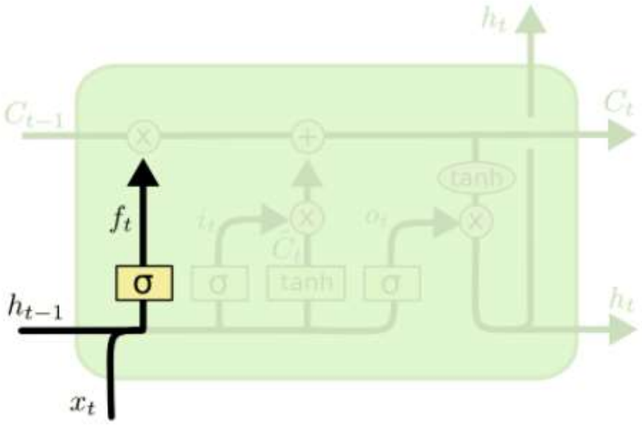
- Cell State
  - 장기적인 의존성을 나타는 정보를 담고 있는 메모리
    - 직렬 구조의 RNN에서 값이 연쇄적으로 연산될 경우, 가장 최근의 정보가 영향력이 강함
    - 장기적인 정보를 보존하기 위한 별도의 구조가 필요
  - 장기적인 정보는 문맥을 관통하므로 이에 포함시키기 위해선 엄격한 입출력의 통제가 필요
    - 세 가지 게이트가 이 역할을 수행



출처 : https://hyen4110.tistory.com/25

LSTM의 구조

- 망각 게이트(Forget Gate):
  - 이전 시점의 은닉 상태를 Cell state에 얼마나 반영할지를 결정
  - 현재 시점의 입력과 이전 시점의 은닉 상태가 Sigmoid 함수를 거치면서 0과 1 사이의 값을 출력
  - 망각할 경우 0, 기억할 경우 1에 가까운 수를 출력

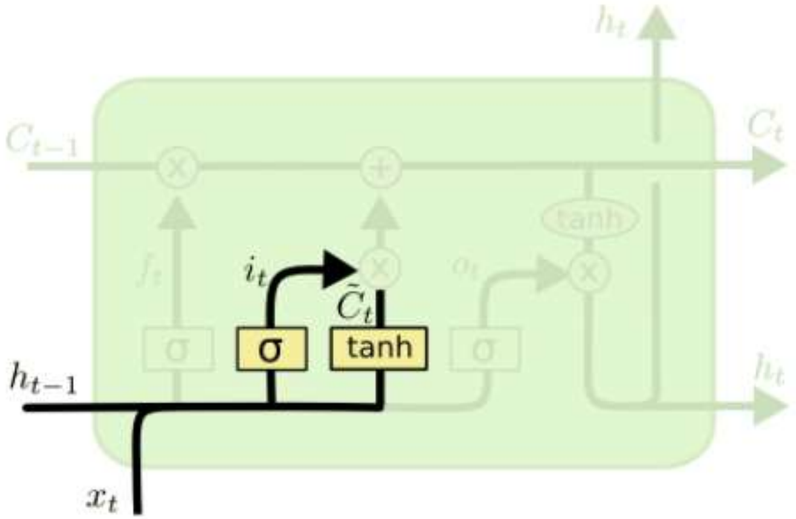


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

출처 : https://hyen4110.tistory.com/25

LSTM의 구조

- 입력 게이트(Input Gate)
  - 새로운 정보를 Cell state에 얼마나 추가할지를 결정
  - 현재 시점의 입력과 이전 시점의 은닉 상태가 Sigmoid 함수를 거치면서 0과 1 사이의 값을 출력



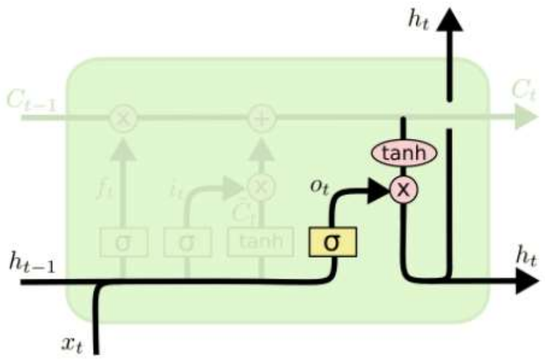
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

출처 : https://hyen4110.tistory.com/25

LSTM의 구조

- 출력 게이트(Output Gate):
  - Cell state에서 어떤 정보를 은닉 상태로 출력할지를 결정
  - 현재 시점의 입력과 이전 시점의 은닉 상태가 Sigmoid 함수를 거치면서 0과 1 사이의 값을 출력
  - 셀 상태는 Tanh 함수를 거쳐 -1과 1 사이의 값을 출력
  - 이 두 값을 곱하여 최종적인 은닉 상태를 얻음



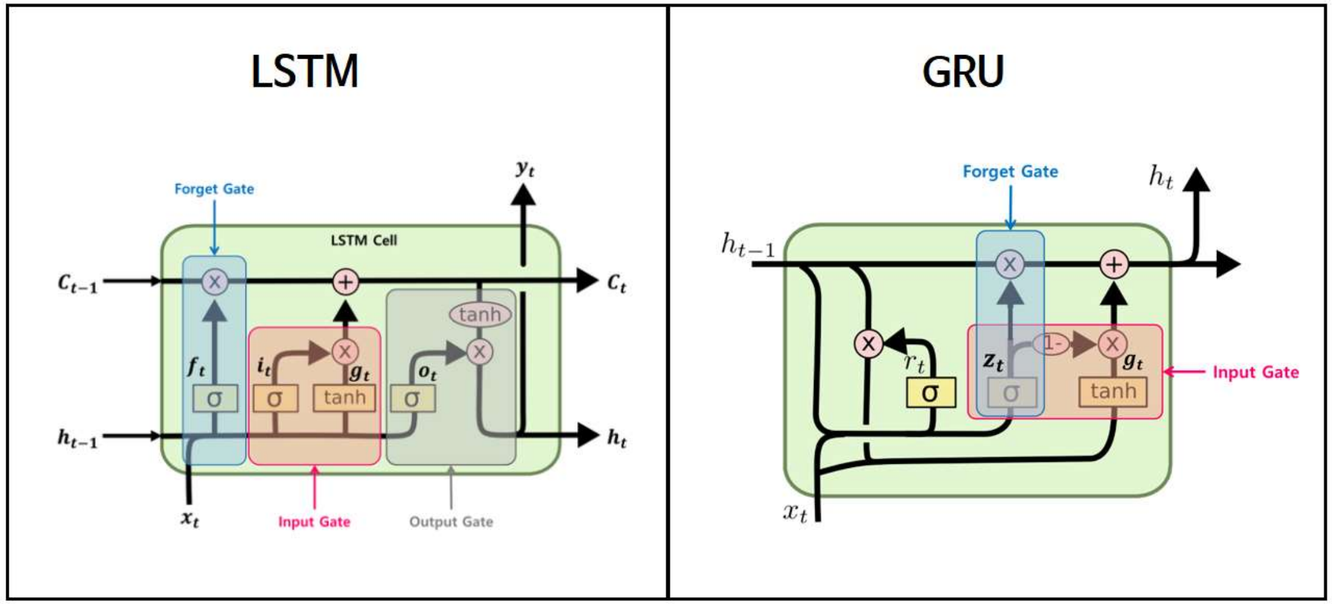
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

출처 : https://hyen4110.tistory.com/25

GRU(Gated Recurrent Unit)

- LSTM의 간소화된 버전
- GRU는 LSTM에 비해 계산 비용이 적고, 학습 속도가 빠름
- 특정 작업에서 LSTM과 동등한 성능을 냄

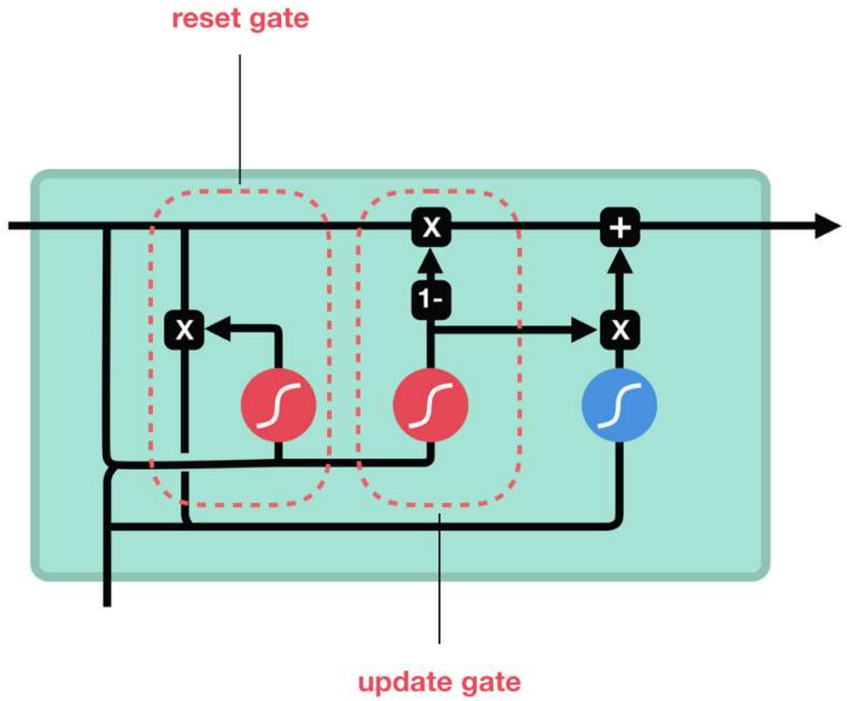


출처 : https://excelsior-cjh.tistory.com/185

GRU의 구조

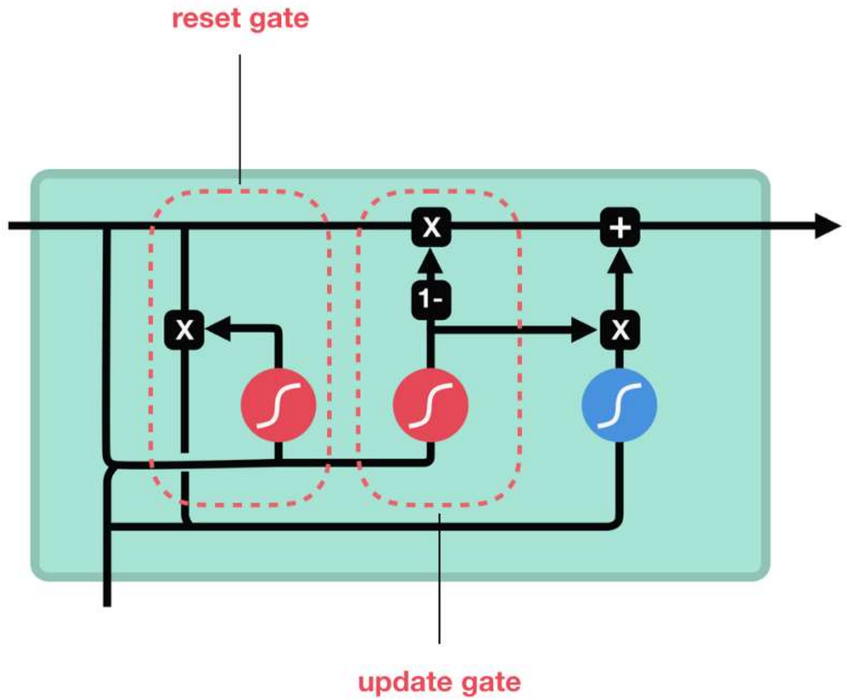
- LSTM과 마찬가지로, Cell state를 통해 장기 기억을 전달 및 보존

- 단, 게이트의 수가 3개에서 2개로 감소
  - 업데이트 게이트(Update Gate)
  - 리셋 게이트(Reset Gate)



GRU의 구조

- 업데이트 게이트(Update Gate):
  - 새로운 은닉 상태를 어떻게 계산할지를 결정
  - 출력이 1에 가까울수록 이전 은닉 상태의 정보가 새로운 은닉 상태에 많이 반영
  - 0에 가까울수록 현재 입력의 정보가 새로운 은닉 상태에 많이 반영됨



GRU의 구조

- 리셋 게이트(Reset Gate):
  - 이전 은닉 상태를 어느 정도 잊을지를 결정
  - 이 게이트의 출력이 0에 가까울수록 이전 은닉 상태의 정보가 많이 망각됨
  - 반면에 출력이 1에 가까울수록 이전 은닉 상태의 정보가 유지됨