2024 생성형 AI 01

언어 모형과 트랜스포머

과목 개요

- 딥러닝을 이용한 자연어 생성 기술을 다룬다
- Python을 이용하여 자연어 생성을 할 수 있다
- 프롬프트 공학을 이용하여 생성형 AI를 활용할 수 있다
- AI 애플리케이션을 개발할 수 있다

- 평가방법:
  - 퀴즈 40%
  - 기말고사 40%
  - 출석 10%
  - 수업참여도 10%

주차별 계획

주차	내용	주차	내용
1	언어 모형과 트랜스포머	9	증강
2	자연어 생성	10	프롬프트 공학
3	문장 분류	11	멀티 모달
4	토큰 분류	12	자율 에이전트
5	임베딩	13	AI 애플리케이션 개발
6	강화학습	14	동향과 전망
7	강화학습과 LLM	15	기말고사
8	효율적 LLM		

언어 모형

언어 모형 language models

- 문장의 확률을 계산하기 위한 모형

$$P(x_1, x_2, \cdots, x_n)$$

- 예) 밥을 먹었다 vs. 밥을 마셨다

확률의 연쇄규칙

- 결합 확률(joint probability): x_1과 x_2이 동시에 발생할 확률

$$P(x_1, x_2)$$

  - 예: 한 문장이 "밥"과 "먹다"로 구성될 확률

- 조건부 확률(conditional probability): x_1이 주어졌을 때, x_2가 발생할 확률

$$P(x_2|x_1)$$

  - 예: "밥"이라는 단어로 시작한 문장에서, 그 다음에 "먹다"가 나올 확률

- 연쇄 규칙(chain rule): 결합확률은 조건부 확률의 곱으로 분해할 수 있음

$$P(x_1, x_2) = P(x_1)P(x_2|x_1)$$

  - 예: "밥"이라는 단어로 시작할 확률 × 그 다음에 "먹다"가 나올 확률

인과적 언어 모형 causal language models

- 결합 확률 대신 조건부 확률 형태의 언어 모형

$$P(x_n|x_1, x_2, \cdots x_{n-1})$$

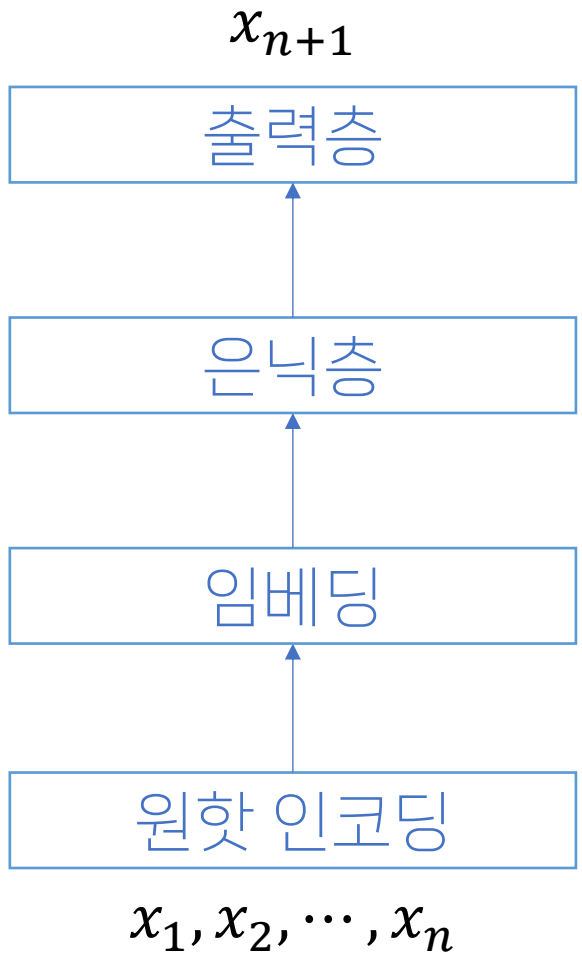
- 일반적으로 "언어 모형"이라고 하면, 이것을 가리킴
- 인공신경망 등의 모형으로 구현하기 쉬움
- 문장에 이어질 단어를 예측 → 단어를 순서대로 생성할 수 있음

n-gram 언어 모형

- 언어 모형을 만드는 가장 간단한 방법
- 텍스트에서 최대 n개까지의 단어 조합의 빈도를 세서 각 조합의 확률을 구함
- 단점:
  - n 단어 이상의 맥락은 고려할 수 없음
  - n이 커질 수록 조합이 폭발적으로 증가하여 많은 저장 공간이 필요(storage problem)
  - 텍스트가 충분히 많지 않으면, 대부분의 조합은 관찰되지 않아 확률이 0으로 추정될 수 있음(sparsity problem)

신경망 언어 모형 Neural Network Language Model

- n개의 단어를 입력으로 받음 $x_1, x_2, \cdots, x_n$
- 각 단어 x_t를 원핫 인코딩
- 단어를 임베딩(embedding)
- 앞먹임 신경망
- 다음에 나올 x_{n+1} 번째 단어를 예측



n-gram vs. NNLM

- storage problem:
  - n-gram은 어휘의 종류가 w개이면 w^n개의 조합을 저장
  - NNLM은 임베딩의 크기가 e, 은닉층의 크기가 h일 경우 $we + eh + hw$의 파라미터만 필요

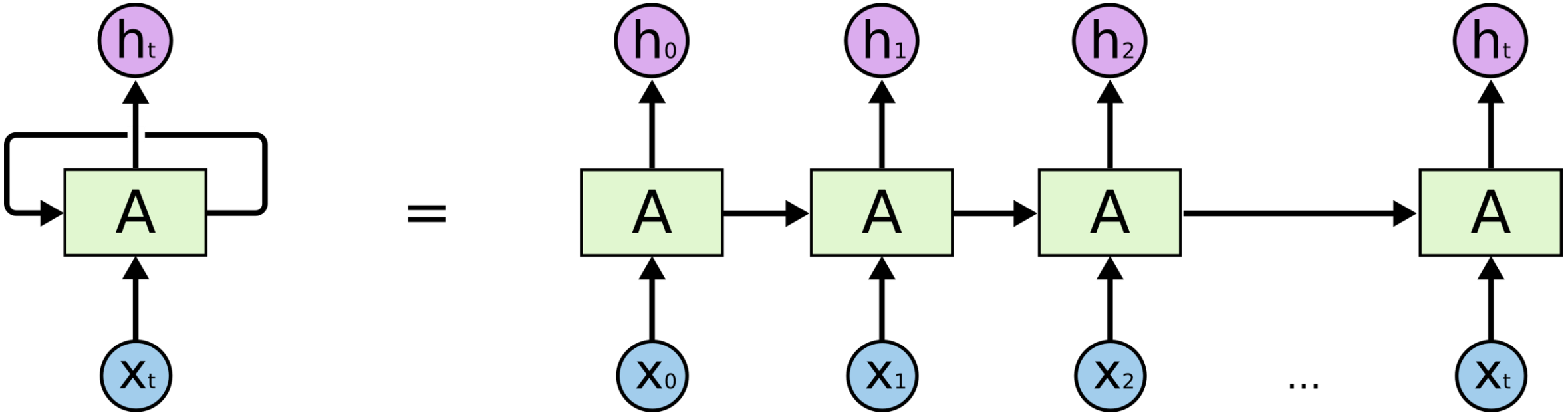
- sparsity problem:
  - NNLM에서는 비슷한 단어는 비슷한 임베딩을 갖게됨
  - 말뭉치에 "떡을 먹다"라는 문장이 없어도 "밥을 먹다", "빵을 먹다" 등의 사례를 통해 "떡을 먹다"에 높은 확률을 주도록 학습할 수 있음

- NNLM의 한계
  - n-gram과 마찬가지로 n개의 단어까지만 반영됨
  - 단어의 위치에 따라 가중치가 달라짐
  - n을 키우면 모형이 커짐

→ 순환신경망 언어 모형으로 극복

순환신경망 recurrent neural network

- 자연어 처리에서 흔히 사용하는 구조
- 이전의 입력이 이후의 입력에 처리에 영향
- 먼 거리에 떨어진 입력 간에 정보를 전달하려면 여러 단계를 거쳐야 함
- 사라지는 경사(vanishing gradient) 문제

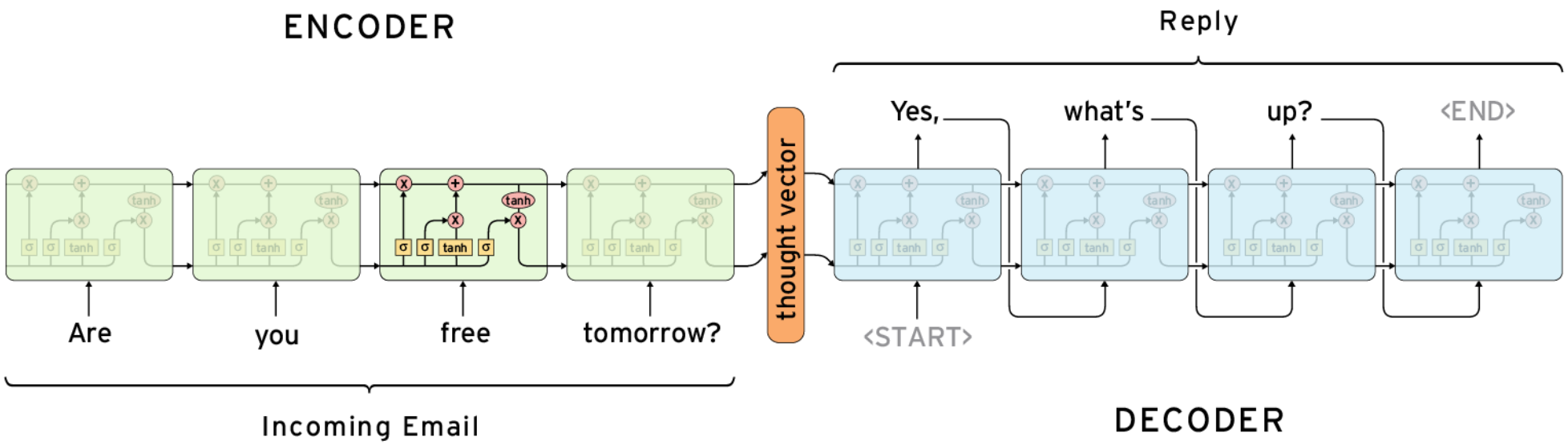


장기 의존성 Long-Term Dependencies

- 멀리 떨어져 있는 단어들 사이의 관계
- 다음에 나올 단어를 예측하는 경우
  - "비가 올 것처럼 하늘이 **흐리다**" → 짧은 맥락만 고려해도 충분
  - "나는 어려서 프랑스에서 자랐다. (중략) 그래서 나는 한국어와 **프랑스어**를 모두 할 수 있다" → 매우 긴 맥락을 고려해야
- RNN은 장기 의존성을 잘 학습하지 못함
- Why? 중간의 처리 단계가 너무 많음

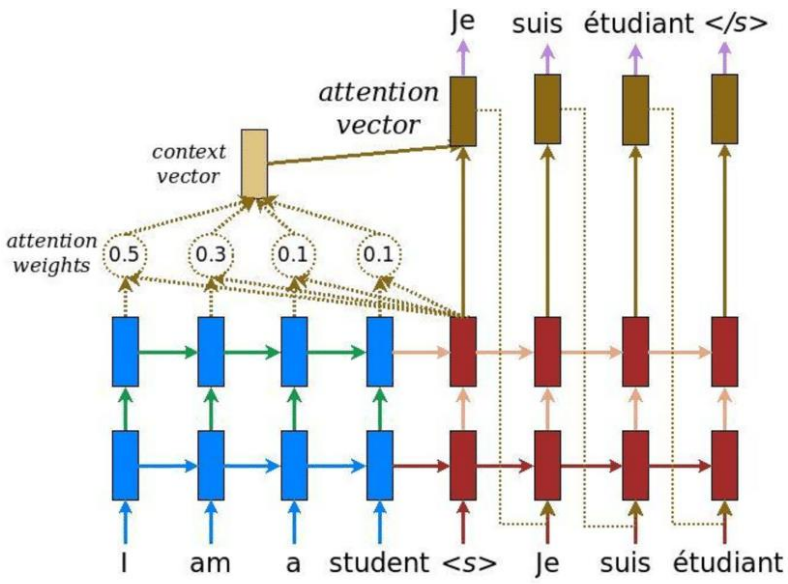
Seq2Seq

- 번역, 요약, 대화 등에 사용하는 신경망 구조
- 입력 문자열을 처리하는 인코더와 출력 문자열을 처리하는 디코더로 구성



주의 메커니즘 attention mechanism

- 번역의 예: I am a student → 나는 학생입니다
- 각 처리 단계에서 필요한 정보는 제한적(영어가 "I"로 시작하면, 한국어는 "나"로 시작)
- 주의 메커니즘의 처리 방식:
  - 이전의 모든 단계의 출력과 현재의 처리 단계에 값을 비교
  - 비슷한 정도에 따라 주의 가중치를 계산
  - 이전 단계의 출력 × 주의 가중치 → 현재 처리 단계에 반영
  - 현재 처리 단계에 필요한 이전 단계의 출력 값이 많이 반영됨

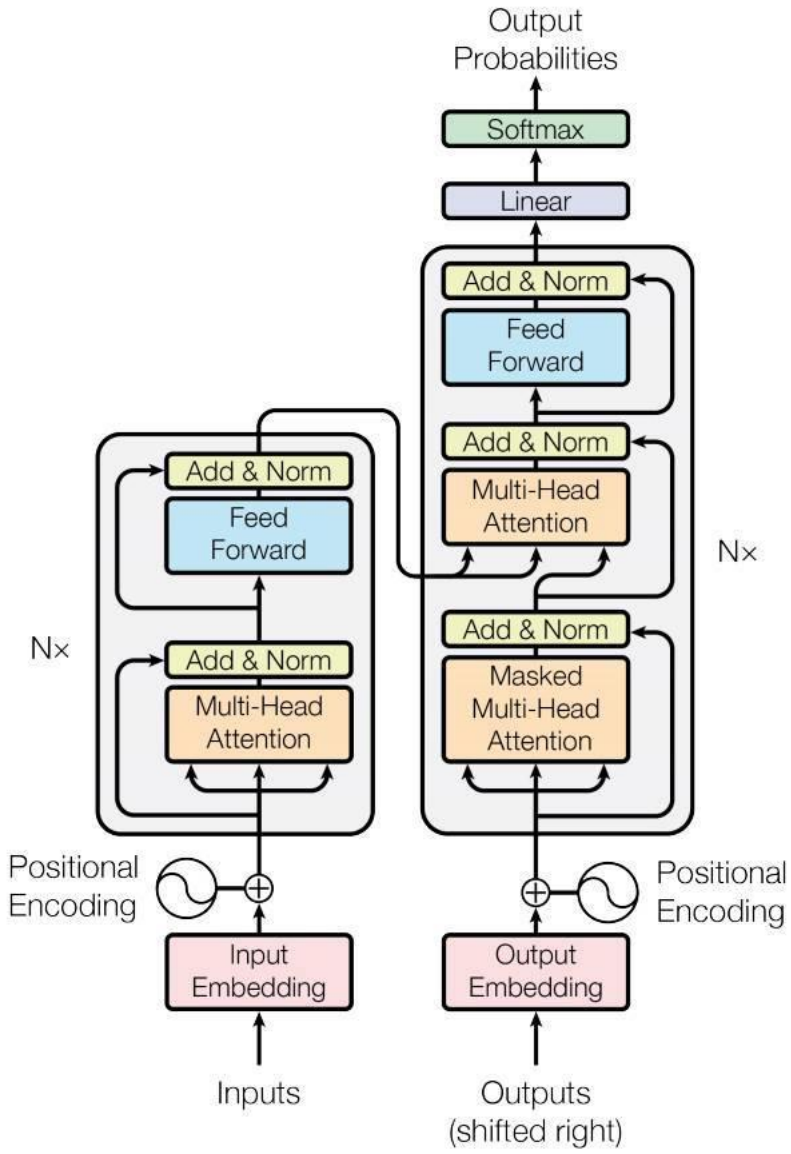


트랜스포머

트랜스포머 Transformer

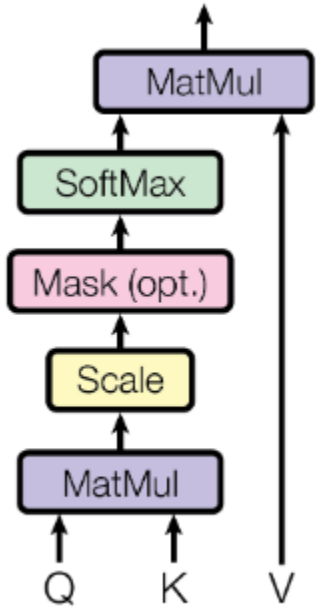
- 주의 메커니즘만을 사용한 Seq2Seq 모형
  - 문장 내에 주의 메커니즘 적용
  - 문장 간에도 주의 메커니즘 적용

- 구성요소:
  - Softmax
  - Multi-Headed Attention
  - Positional Encoding
  - Residual Block
  - Layer Normalization



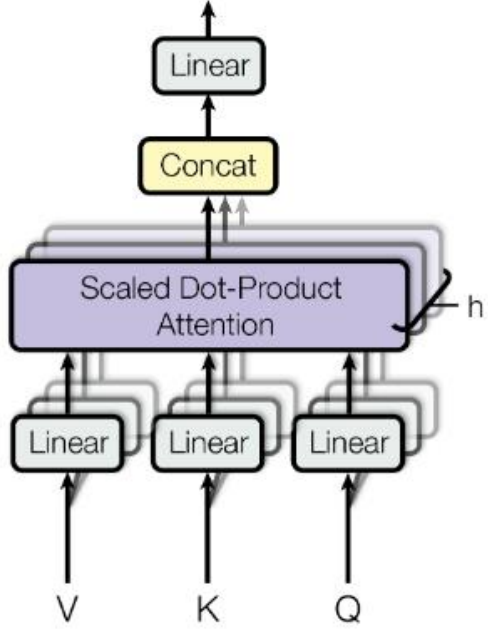
Scaled Dot-Product Attention $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

- 하나의 입력값을 Query, Key, Value 세 가지 값으로 변환
- Query와 비슷한 Key를 가지는 Value에 높은 주의를 부여
- QK^T: Q의 모든 행과 K의 모든 행을 점곱
- 점곱이 너무 커지지 않도록 $\sqrt{d_k}$로 나눠줌 (d_k: Q와 K의 차원)
  - 점곱이 너무 커지면 주의가 극단적으로 한쪽으로 몰리고
    경사가 작아져서 학습이 잘 이뤄지지 않음
- 소프트맥스 함수에 통과시키면 0~1 사이의 주의 가중치를 얻게 됨
- 주의 가중치를 V에 곱하여, Q와 비슷한 K의 V를 가장 많이 반영



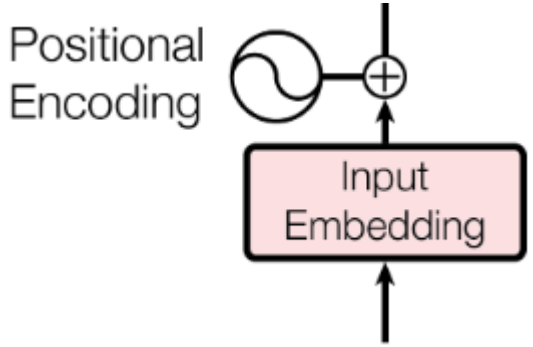
Multi-Headed Attention

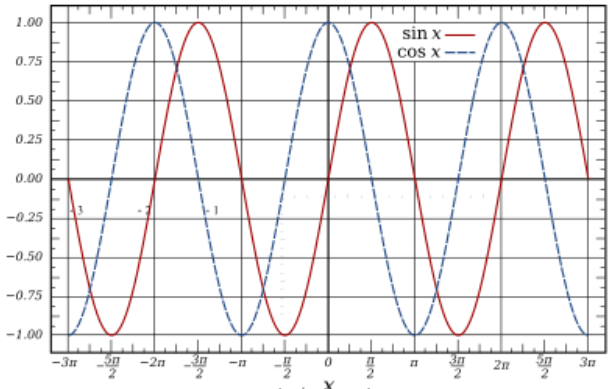
- Q, K, V를 여러 가지로 변환하여 주의 메커니즘 적용
- head = 한 가지 Q, K, V의 결과, 트랜스포머는 head가 8개
- 다양한 위치의 다양한 표상 하위공간(representation subspaces
- 문장에서 여러 가지 방식으로 여러 위치에 주의를 줄 수 있음



위치 인코딩 positional encoding

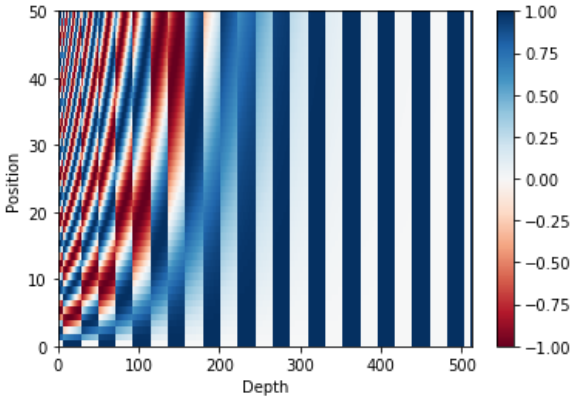
- 순환신경망은 처리 과정 자체가 순서대로 이뤄지므로 단어의 순서를 반영
- 트랜스포머 모형은 순서를 다루는 구조가 없음
- 문장에서 단어의 위치를 인코딩하여 단어 임베딩에 더해줌
- 정현파를 이용하여 만들기도 하고,
  단어 임베딩과 마찬가지로 학습 시킬 수도 있음
- 정현파(sinusoid): sin 함수 또는 cos 함수의 파형





정현파

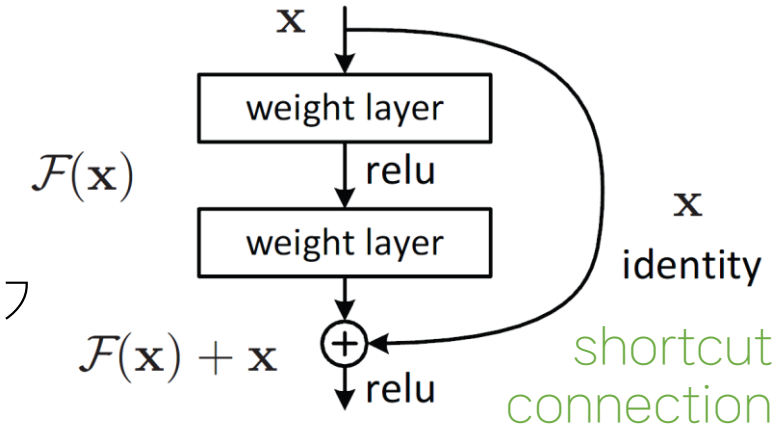
https://en.wikipedia.org/wiki/Sine_wave



정현파를 이용한 위치 인코딩

https://www.tensorflow.org/tutorials/text/transformer

Residual Block



- 신경망이 깊어지면 사라지는/폭발하는 경사와 함께 degradation 문제가
  - 성능이 일정 수준에 도달하면, 오히려 성능이 저하되는 문제
- 신경망의 초반부 레이어들에서 필요한 학습이 충분히 이뤄지면, 후반부의 추가적 레이어들은 불필요
- 해결책: 입력을 그대로 출력 → 비선형 활성화 함수를 사용하므로 어려움
- Residual Block: 레이어의 입력값을 그대로 레이어의 출력값에 더함
  - 추가적인 파라미터 X, 레이어의 가중치가 0이 되면 입력이 그대로 출력
- 일반적인 네트워크는 출력값을 학습 vs.
  Residual Block의 레이어는 입력값과 출력값의 차이(잔차, residual)를 학습
- 사라지는/폭발하는 경사 문제에도 효과

트랜스포머가 잘 작동하는 이유

- CNN은 국소적 관계에 대해, RNN은 순차적 관계에 대해 강한 가정을 가짐
- RNN은 데이터를 순서대로 처리하므로 순서상 가까운 단어가 더 강한 영향
- 그러나 자연어는 반드시 순서대로 처리되는 것은 아님(예: 주술호응)
- 트랜스포머는 순서나 위치에 대한 가정이 없음 → 편향이 적음

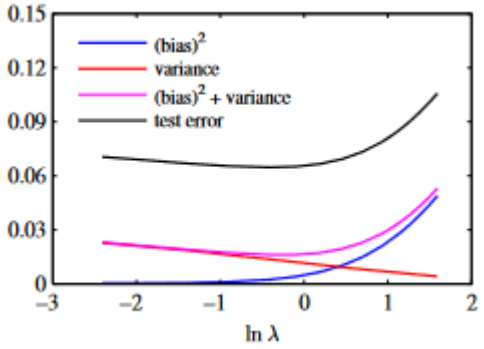
Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

Table 1: Various relational inductive biases in standard deep learning components. See also Section 2.

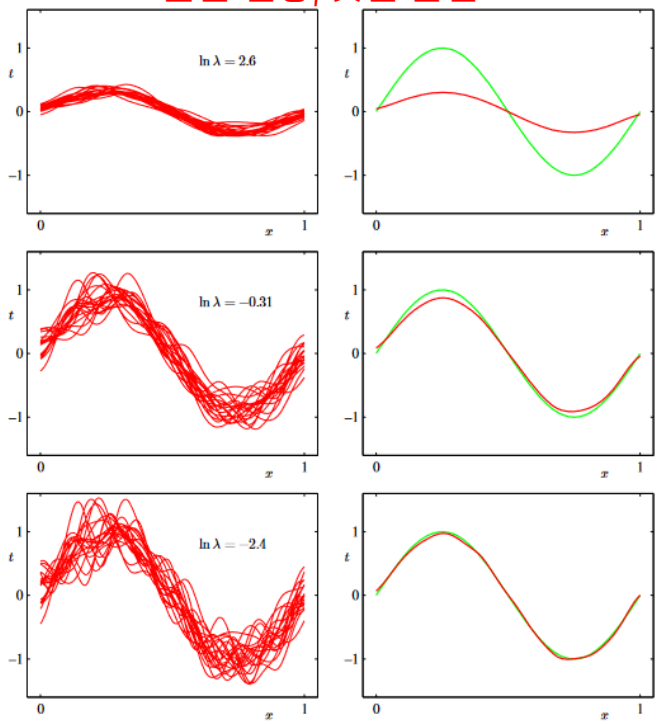
출처: Relational inductive biases, deep learning, and graph networks

귀납 편향 inductive bias

- 모형의 귀납 편향: 데이터의 구조에 대해 모형의 가정
- 예: 선형 모형은 데이터가 직선의 패턴을 가질 것으로 가정
- 머신러닝 모형은 이러한 가정 아래 훈련 데이터의 패턴을 일반화
- 편향-분산 교환(bias-variance trade-off)
- 편향이 강하면 분산이 작아짐
  - 학습 데이터셋에 따라 결과가 크게 달라지지 않음
  - 작은 데이터셋에서 성능이 좋음
- 편향이 약하면 분산이 커짐
  - 학습 데이터셋에 따라 결과가 크게 달라짐
  - 큰 데이터셋에서 성능이 좋음



높은 편향, 낮은 분산



낮은 편향, 높은 분산

전이 학습 Transfer Learning

- 딥러닝 모형이 커질 수록 더 복잡한 함수를 근사할 수 있음 → 성능이 증가
- 큰 모형에는 더 많은 데이터, 더 많은 계산이 필요 → 비용 증가
- 여러 가지 문제는 저수준의 표상들을 공유함(예: 이미지 처리에서 수평선, 수직선 처리 등)
- 하나의 **근원 문제**(source)에 대량의 데이터로 **사전 학습**(pretraining)을 시켜 문제를 해결하는 데 필요한 다양한 표상을 학습
- 새로운 **대상 문제**(target)에 소량의 데이터로 **미세 조정**(fine tuning)하면 적은 데이터로 빠르게 높은 성능을 낼 수 있음
- **기초 모형** foundation model: 컴퓨터 비전이나 자연어 처리처럼 넓은 종류의 분야에 대해 적용할 수 있는 하나의 모형
  - 기초 모형을 대량의 데이터로 사전 학습
  - 사전 학습된 모형을 개별 과업에 맞춰 미세하게 수정해서 사용하는 것

전이 학습의 방식

- 사전 학습된 부분을 포함해서 전체적으로 미세 조정

- 사전학습된 모형의 전반부를 특징 추출기로 사용 + 후반부에 분류기만 추가하고 추가된 부분만 학습

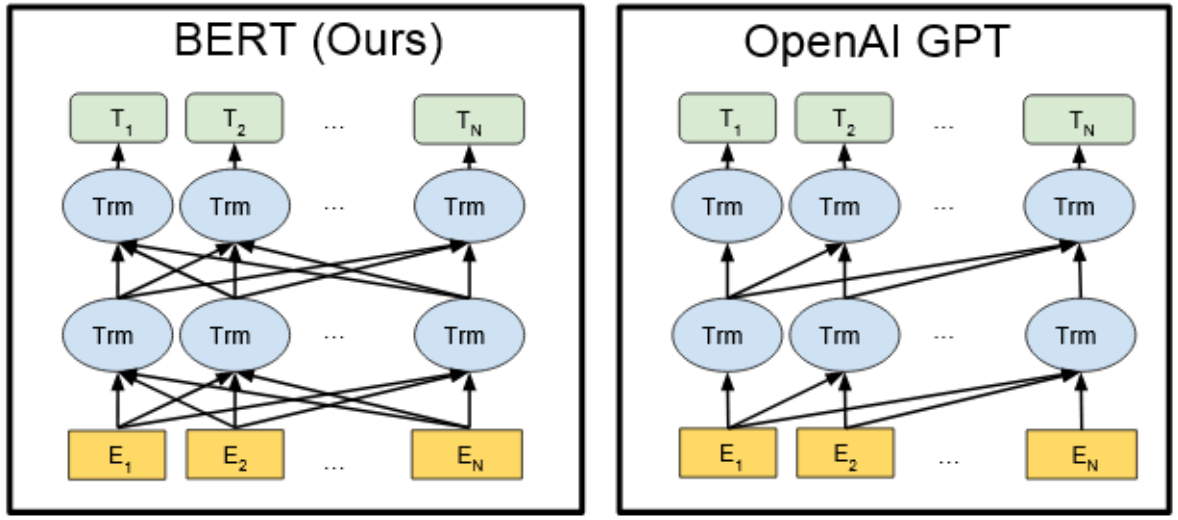
- 사전학습된 모형을 추가 학습 없이 그대로 사용(zero-shot learning)

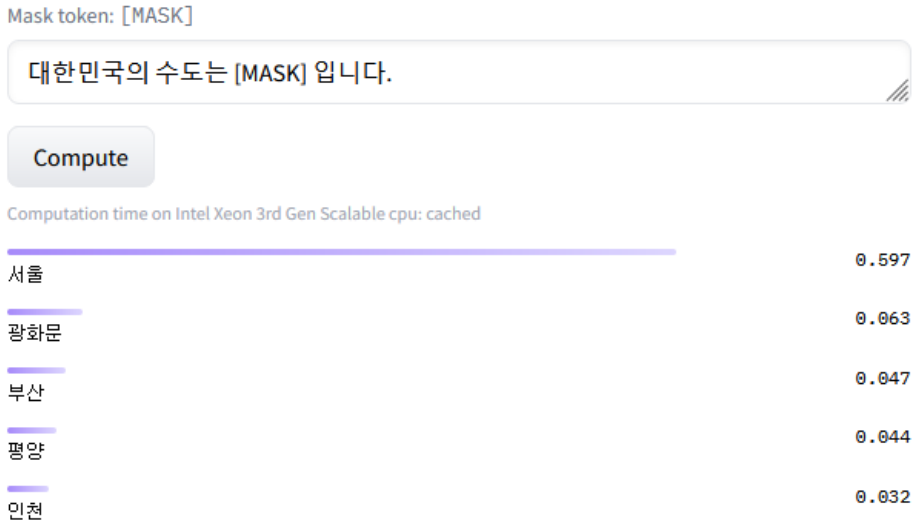
GPT Generative Pre-Training

- Transformer의 디코더를 바탕으로 만든 언어 모형
- 앞에서 나온 단어들을 바탕으로 다음에 나올 단어를 예측
- 2020년 5월 공개된 GPT-3는 약 5천억개의 토큰으로 학습
- 데이터가 소량이거나 심지어 전혀 없어도 전이 학습이 가능
- 예: GPT를 이용한 감성 분석의 zero-shot learning
  - "이 영화는 참 재미있다."라는 문장의 긍/부정을 분류하려는 경우
  - 문장의 끝에 "그래서 나는 이 영화가"라는 표현을 추가
  - 다음에 좋다/싫다가 나올 확률을 예측

BERT Bidirectional Encoder Representations from Transformers

- 양방향 Bi-directional으로 정보를 처리하는 마스크트(masked) 언어 모형
- 문장 내에서 가려진 부분에 나올 단어를 예측
- 전이학습에서 GPT보다 나은 성능, 자연어 생성은 불리
- 다양한 종류의 자연어 처리 과제에 전이 학습을 쉽게 할 수 있도록 설계

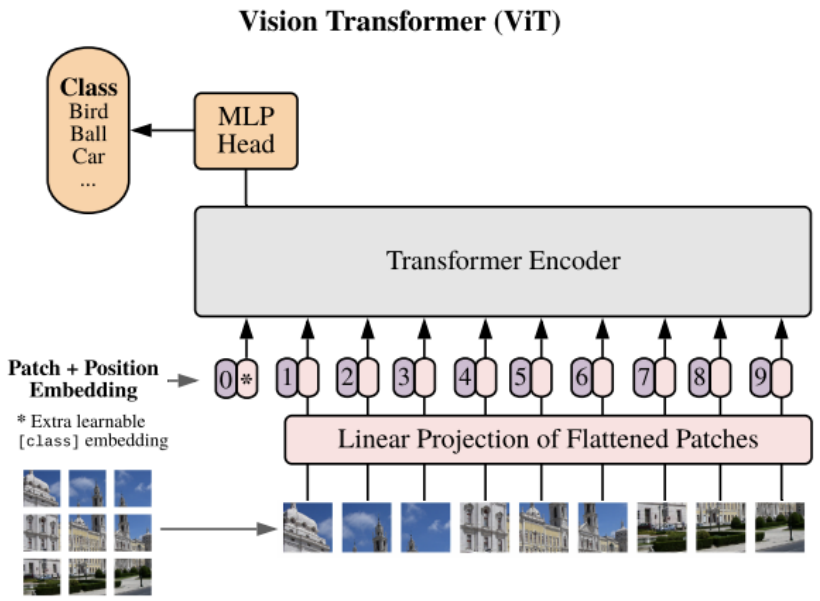


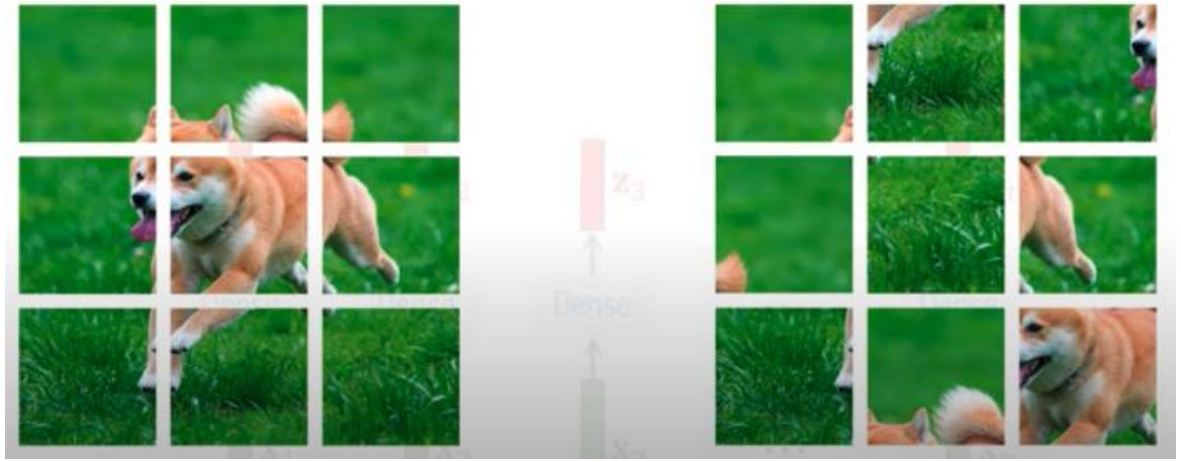


자연어 처리 바깥으로 트랜스포머의 확산

비전 트랜스포머 Vision Transformer

- 트랜스포머를 컴퓨터 비전에 적용한 것
- 이미지를 패치(patch)로 쪼개어, 패치의 시퀀스로 다룸
- 자기 주의(self attention)는 순열 불변(permutation invariant)
  - 입력 순서가 달라져도 출력이 같음
- 위치 인코딩을 함께 입력하여 순서 정보를 추가

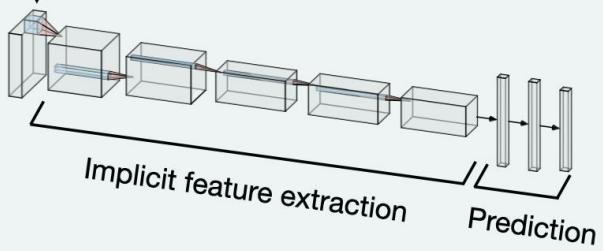




비정형 vs. 정형 데이터

Unstructured data

Deep learning

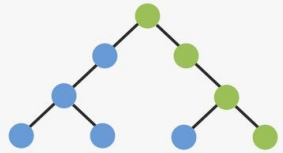


"Manual" feature extraction

Structured data

	Sepal length	Sepal width	Petal length	Petal width
Flower 1	5.1	3.5	1.4	0.3
Flower 2	4.9	3.0	1.4	0.2
Flower 3	5.9	3.0	5.1	1.8
...	...	...	...	...

Conventional machine learning



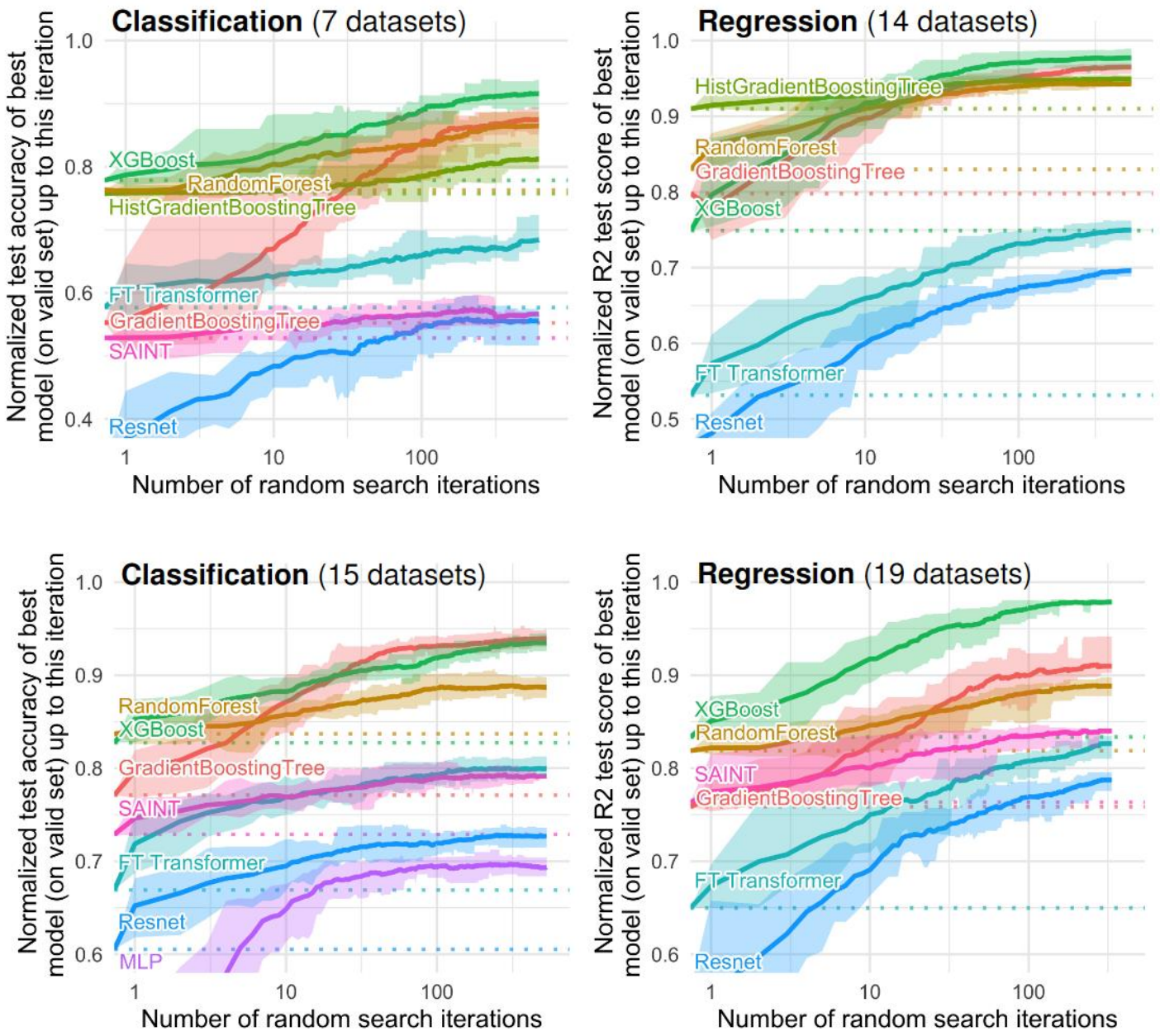
Tabular Data Learning

- 딥러닝은 이미지, 언어, 소리 등 비정형 데이터에서 강점
- 기존의 딥러닝 모형이 이들 데이터에 맞는 귀납 편향을 가지기 때문
- 표 형태의 정형 데이터는 특성이 다름:
  - 이질적인 특징(heterogeneous features)
  - 작은 표본 크기(small sample size)
  - 극단치(extrem values)
- 표 데이터에서는 트리 기반의 모형들이 강세(예: XGBoost 등)
- 표 데이터에 맞춘 다양한 딥러닝 모형이 시도되었으나, 다양한 종류의 데이터에서 트리 기반 모형을 능가하지 못함

원인이 아닌 것

- 하이퍼파라미터 튜닝이 문제가 아님
  - 하이퍼파라미터 튜닝을 해도 모형 간의 성능 격차는 줄어들지 않음
  - 오히려 튜닝을 덜 했을 때 성능 차이가 더 큼

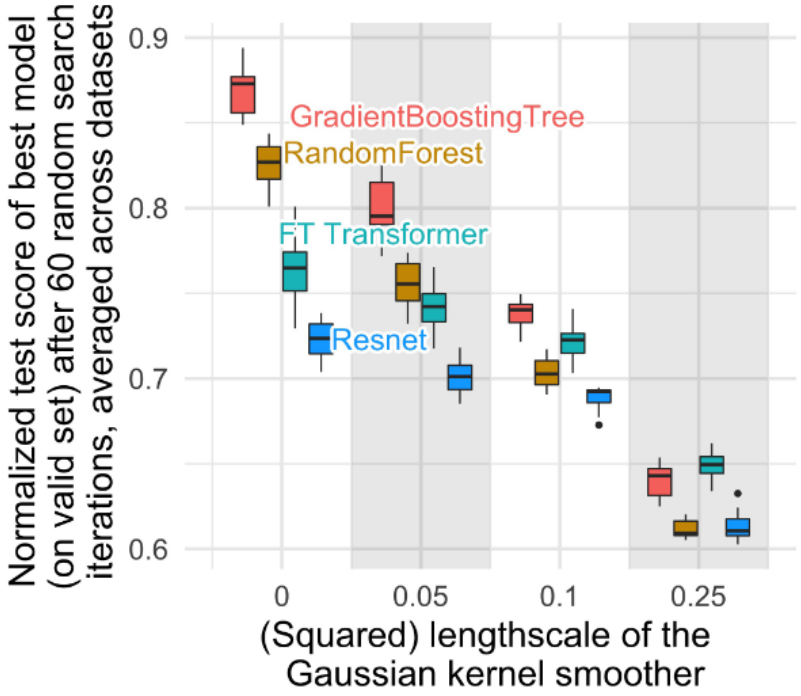
- 범주형 변수가 문제가 아님
  - 연속형 변수만을 사용해도 트리 기반 모형의 성능이 더 높음

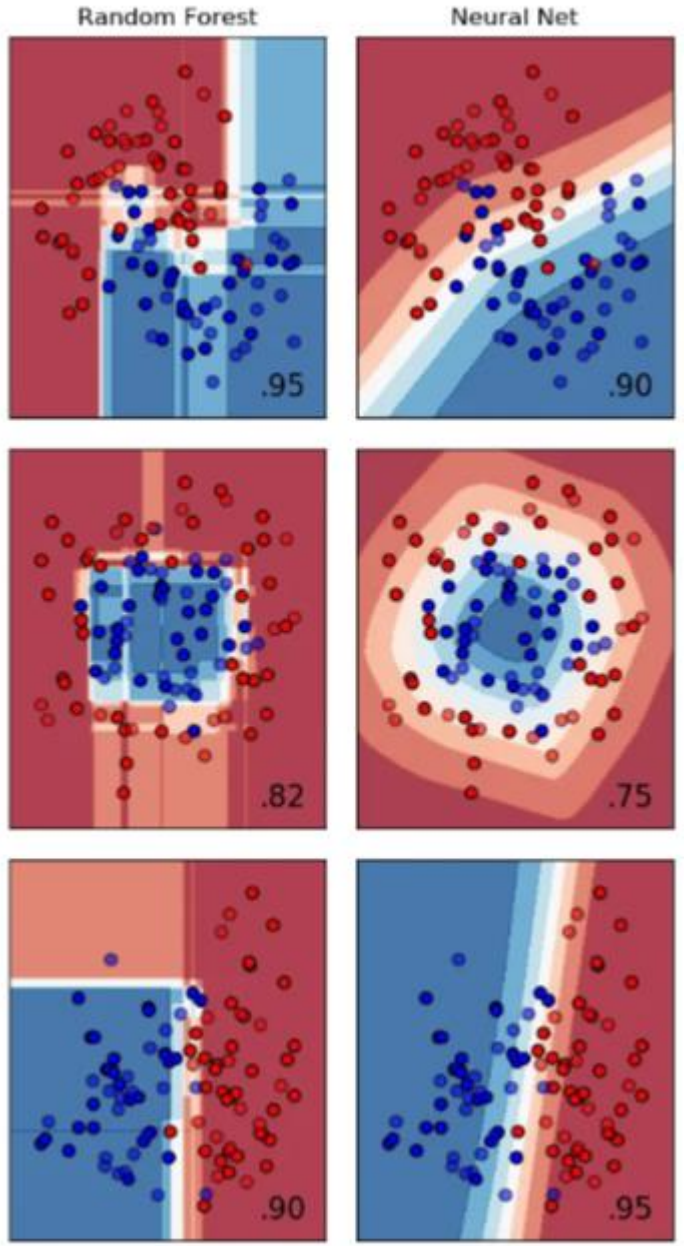


출처: Why do tree-based models still outperform deep learning on tabular data?

원인(1): 매끄러운 함수에 편향

- 신경망 모형은 매끄러운/낮은 주파수의 함수에 편향
- [→] 분류 문제의 결정 경계
- [↓] 데이터의 출력(y)을 평활(smoothing)하면 트리 기반 모형의 성능이 더 많이 떨어짐

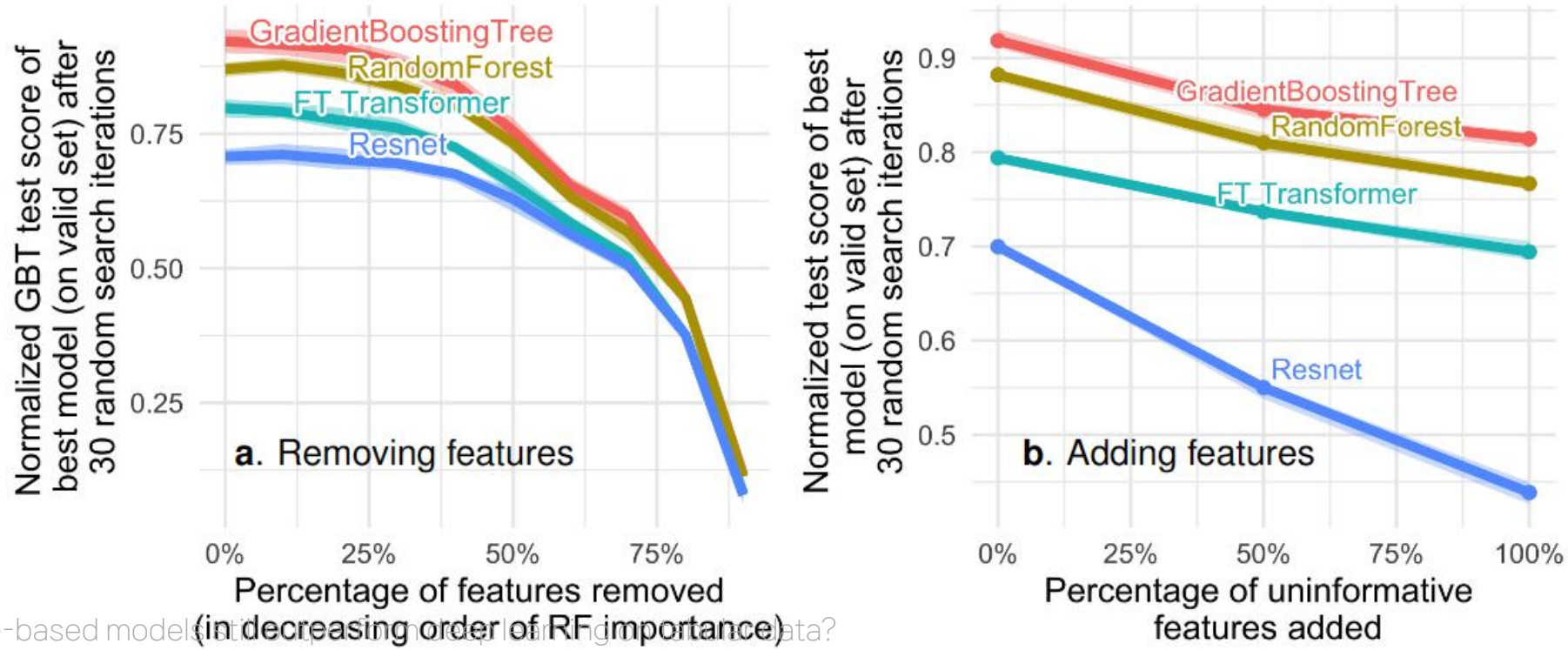




출처: Why do tree-based models still outperform deep learning on tabular data?

원인(2) 정보가 없는 특징에 약함

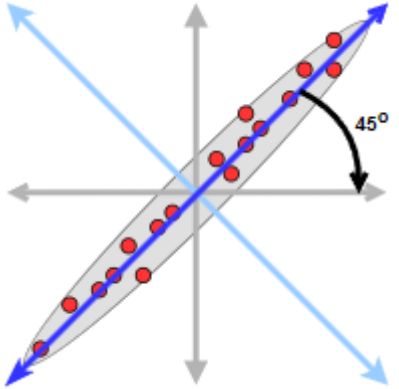
- 표 데이터는 정보가 없는 무정보 특징(uninformative feature)이 많음
- 신경망 모형은 무정보 특징에 약함
- [←] 상대적으로 무정보 특징들을 데이터에서 제거하면 모형 간의 성능 차이가 감소함
- [→] 인위적으로 무정보 특징(무작위 잡음)을 추가하면 일부 신경망 모형(Resnet)의 성능이 더 빨리 하락

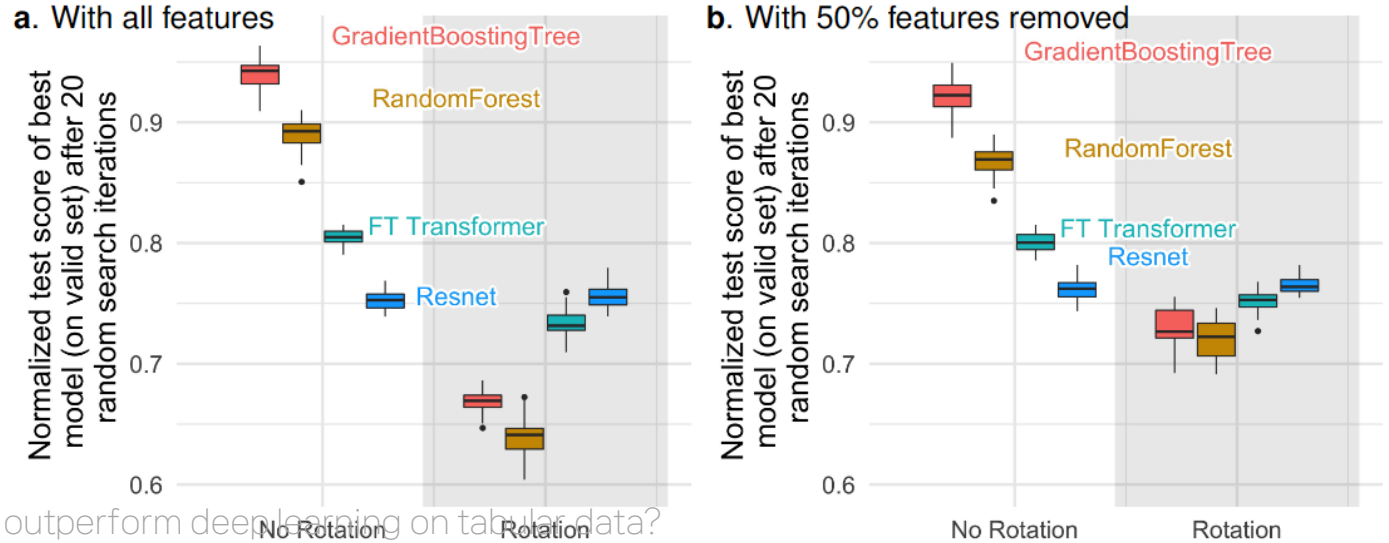


출처: Why do tree-based models still outperform deep learning on tabular data?

원인(3) 회전 불변성

- 훈련 데이터와 테스트 데이터를 회전(rotation)시켰을 때 대부분의 신경망 모형은 영향을 받지 않음 (회전 불변성)
- 회전 불변인 모형은 정보가 없는 특징이 추가될 경우 더 큰 표본이 필요
- 표형태의 데이터는 특징별로 고유한 의미가 있음(예: 키, 몸무게)
- 회전을 시키면 특징들의 정보가 섞임
- [a] 무작위로 회전을 시킬 경우 성능의 순서가 역전됨
- [b] 무정보 특징을 제거하면 회전시켜도 성능 차이가 크지 않음

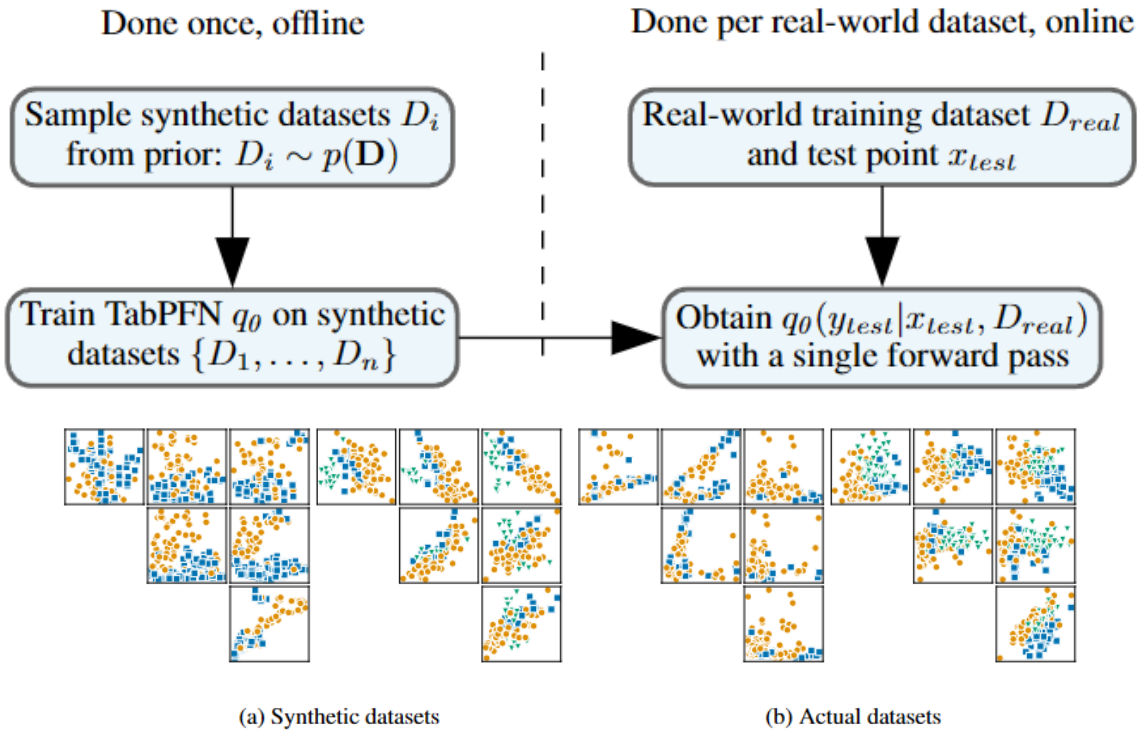


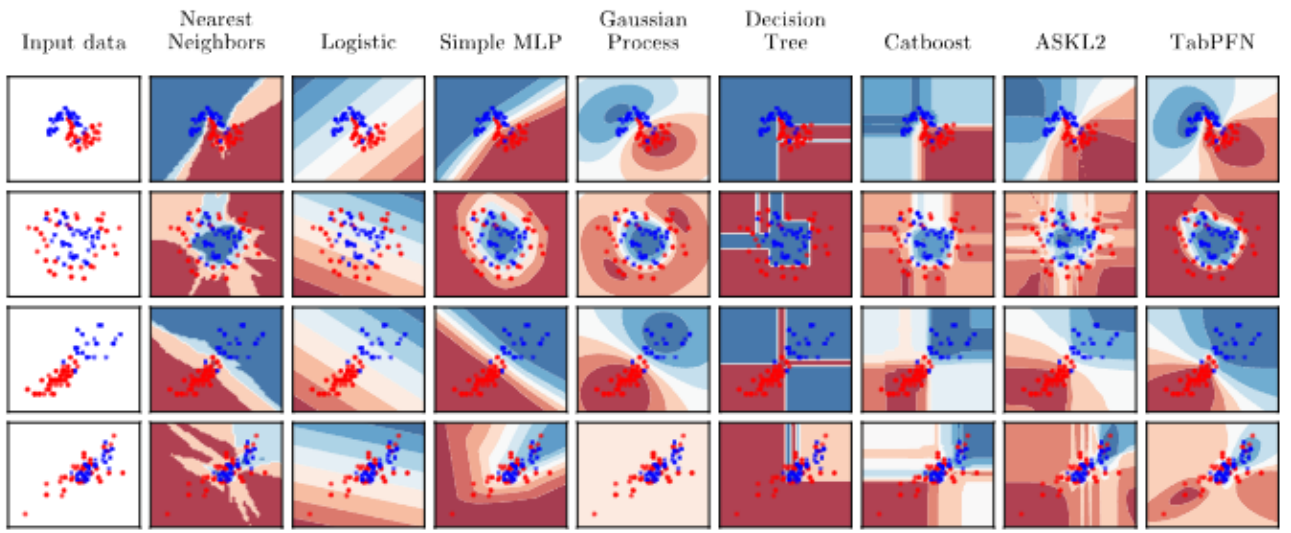


출처: Why do tree-based models still outperform deep learning on tabular data?

TabPFN

- 2022년 발표된 트랜스포머 기반의 표 데이터 모형
- 기존의 머신 러닝은 모형을 특정 데이터에 맞춰 학습
- PFN(Prior-Data Fitted Network)은 사전에 다양한 합성 데이터에 학습
- 단순한 함수를 선호하도록 학습됨

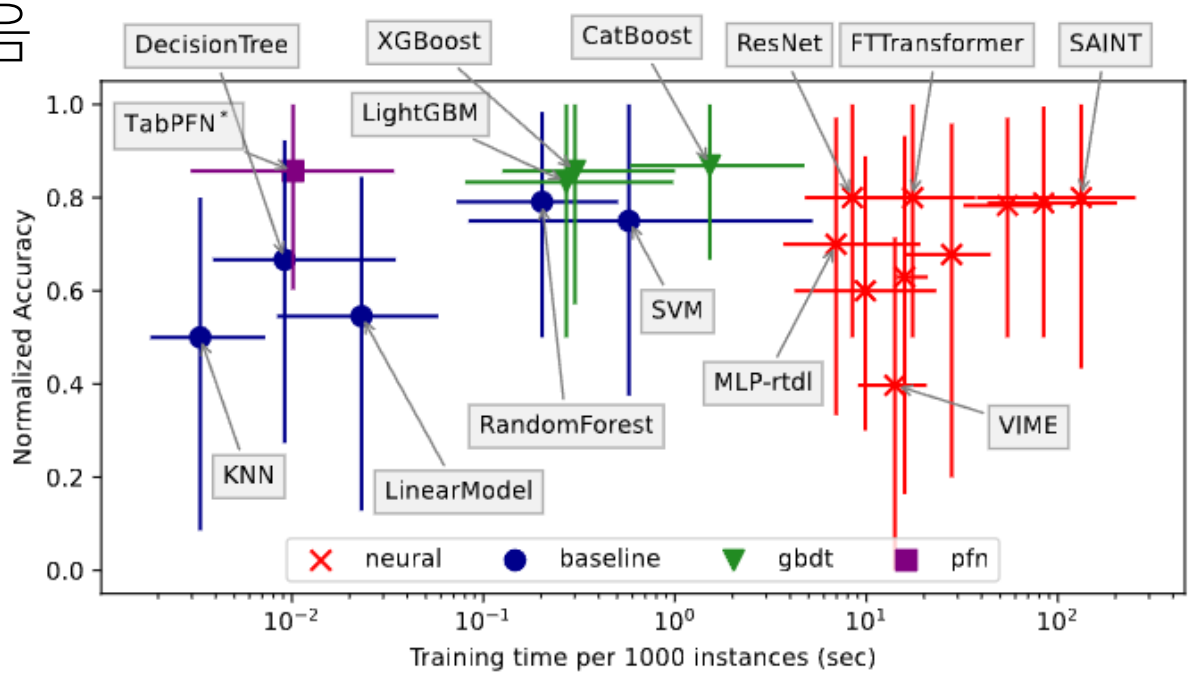




TabPFN의 장단점

- 장점:
  - 매우 짧은 추론 시간
  - 새로운 데이터셋에 대해 학습이 필요 없음
  - 자동 전처리

- 단점:
  - 대규모, 고차원 데이터셋 적용이 제한
  - 범주형 변수 처리할 수 없음



TabPFN 실습

```
# 설치
pip install tabpfn

# 데이터
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=42)
```

TabPFN 실습

```
# 분류
from tabpfn import TabPFNClassifier
classifier = TabPFNClassifier(device='cpu', N_ensemble_configurations=32)
classifier.fit(X_train, y_train)

# 예측 및 평가
from sklearn.metrics import accuracy_score
y_eval, p_eval = classifier.predict(X_test, return_winning_probability=True)
accuracy_score(y_test, y_eval))
```

준단어 토큰화

단어 수준 토큰화 vs. 글자 수준 토큰화

- 미등록 어휘(Out-Of-Vocabulary: OOV) 문제 :
  단어는 열린 집합이므로 새로운 단어가 계속 생김
  → 학습 데이터에 없는 단어는 예측할 수 없음
- 글자는 대체로 닫힌 집합(새로운 글자가 생기는 경우는 거의 없음).
  but, 의미 파악이 어려움
- 한국어의 경우:
  - 현대 한국어 모아쓰기 글자 수: 11,172글자
  - 현대 한국어 풀어쓰기 글자 수:
    - 초성 19글자
    - 중성 21글자
    - 종성 28글자(종성 없음도 1글자로 포함할 경우)

준단어 토큰화 subword tokenization

- 자주 사용되는 글자 조합들을 하나의 토큰(처리 단위)으로 취급(예: t+h → th)
- 글자 수준 토큰화에 비해 좀 더 크고 의미있는 토큰을 사용
- 별도의 프로그래밍이나 데이터 개발을 위한 수작업이 불필요
- 특정 언어의 문법 규칙에 구애받지 않으므로 다양한 언어를 지원할 수 있음
- 단어 경계가 모호한 경우에 도움이 됨('중국요리' vs. '중국_요리')
- 불규칙 변화 등에는 취약 (see → saw, 푸다 → 펐다)

tiktoken

- OpenAI에서 개발한 토크나이저
- 설치:

```
!pip install tiktoken
```

- 사용:

```
import tiktoken

enc = tiktoken.encoding_for_model("gpt-4")

tokens = enc.encode('자연어 처리가 재밌다')

tokens
```

- 합치기

```
enc.decode(tokens[:4])
```

- 비슷한 길이의 문장도 한국어는 영어에 비해 더 많은 토큰으로 나눠지는 경향
  → 한국어 처리시 느린 속도 및 낮은 성능의 원인

What you see

What LLM sees

Words vs Tokens

A word is most likely what you think it is - the most simple form or unit of language as understood by humans. In the sentence, "I like cats", there are three words - "I", "like", and "cats." We can think of words as the primary building blocks of language; the fundamental pieces of language that we are taught from a very young age.

A token is a bit more complex. *Tokenization* is the process of converting pieces of language into bits of data that are usable for a program, and a *tokenizer* is an algorithm or function that performs this process, i.e., takes language and converts it into these usable bits of data. Thus, a token is a unit of text that is intentionally segmented for a large language model to process efficiently. These units can be words or any other subset of language - parts of words, combinations of words, or punctuation.

There are a variety of different tokenizers out there which reflect a variety of trade offs. Well-known tokenizers include NLTK (Natural Language Toolkit), Spacy, BERT tokenizer and Keras. Whether or not to select one of these or a different tokenizer depends upon your specific use case. On average, there are roughly 0.75 words per token, but there can be meaningful differences among tokenizers.