

경영학과 학생을 위한 GPT 활용

2025. 1

정용진, 광운대학교 (yjjeong@kw.ac.kr)

GPT

대상: 전남대 경영학과 재학생

기간: 3일, 총 20시간

목적: ChatGPT를 활용한 경영 데이터 분석, 비즈니스 의사결정 지원 및 활용
방법 학습

1일차: ChatGPT 이론 및 기본 활용

시간: 7시간

목표: ChatGPT 모델에 대한 이해와 기본 활용 능력 습득

- 1교시 (1시간): ChatGPT 개요 및 기초
- ChatGPT 모델의 개념, 구조, 핵심 원리 이해
- 언어 생성 메커니즘 및 ChatGPT의 경영학 적용 가능성 탐구
- 2교시 (2시간): ChatGPT 모델 훈련 방법론
- ChatGPT의 데이터셋 구성 및 훈련 방식 이해
- 입력/출력 형식 및 대화 흐름 관리 기술
- 3교시 (2시간): ChatGPT 프로그램 설치 및 환경 설정
- ChatGPT 프로그램 실습 환경 설정
- 간단한 텍스트 생성 및 분석 실습
- 4교시 (2시간): 대화 생성 및 분석 실습
- 경영학 사례 기반 대화 데이터 생성 실습
- 텍스트 기반 데이터 분석 실습

2일차: ChatGPT 심화 활용

시간: 7시간

목표: 경영 데이터 분석 및 의사결정 지원을 위한 ChatGPT 심화 기술 학습

- 1교시 (2시간): ChatGPT의 경영학 응용
- 경영학 사례 기반 ChatGPT 활용 실습
- 시장 데이터 분석 및 트렌드 예측 실습
- 2교시 (2시간): 언어 모델링 실습 (어린왕자 데이터셋 활용)
- 문학 데이터 분석 및 모델링 기술
- 데이터셋을 활용한 자연어 처리 기초 학습
- 3교시 (3시간): 경영 데이터셋 기반 언어 모델링
- 경영학 데이터셋 활용 방법 학습
- 재무 및 마케팅 데이터 분석 사례 실습

GPT

3일차: ChatGPT를 활용한 경영 솔루션 개발

[실습 코드]

시간: 6시간

목표: ChatGPT를 활용한 경영 솔루션 개발 및 실무 적용

- 1교시 (2시간): 대규모 언어 모델링 응용
- 대규모 데이터셋 구성 및 전처리 실습
- 비즈니스 문제 해결을 위한 언어 생성 및 분석
- 2교시 (2시간): ChatGPT를 활용한 실무 프로젝트
- 경영학 문제 해결 사례 기반 실습
- ChatGPT를 통한 비즈니스 아이디어 제안
- 3교시 (2시간): 성과 발표 및 토론
- 팀별 프로젝트 결과 발표
- ChatGPT 활용 방안에 대한 피드백 및 토론

교육 종료 후 기대 효과

1. ChatGPT의 이론 및 기술적 메커니즘에 대한 이해 향상
2. 경영학 데이터를 활용한 실질적인 문제 해결 능력 배양
3. ChatGPT를 활용한 데이터 분석 및 비즈니스 의사결정 지원 기술 습득

1_prompt_engineering_exercise_with_assignment.ipynb

2_gpt_intro.ipynb

3_gpt_direct_or_api.ipynb

7_RAG_examples_with_assignment.ipynb

5_huggingface_transformers.ipynb

4_gpt_api_rev1.ipynb

6_generate_a_novel.ipynb

7_RAG_examples_with_assignment.ipynb

8_making_excel_using_gpt.ipynb

9_making_ppt_using_gpt.ipynb

10_stock_price_prediction_with_assignment.ipynb

What is GPT?

- GPT의 정의:
 - 텍스트 데이터를 학습하여 자연어를 생성하고 이해할 수 있는 인공지능 모델
- GPT의 주요 기능:
 - 질문에 답변
 - 텍스트 생성 (예 이메일 작성, 보고서 작성)
 - 번역
 - 요약
 - 데이터 분석 보조, 등
- 실생활 예시:
 - 고객 지원 자동화: 고객이 제품 관련 질문을 하면 GPT가 즉시 답변
 - 콘텐츠 제작: 블로그 포스트 초안 작성
 - 개인 비서: 일정 관리 및 이메일 초안 작성.
- 경영학 관점 활용 사례:
 - 의사 결정 지원: 데이터를 분석해 보고서 초안 작성
 - 마케팅: 고객 리뷰 분석 및 소비자 트렌드 예측
 - 운영 효율화: 내부 문서 관리 및 자동화된 답변 시스템

Transformer

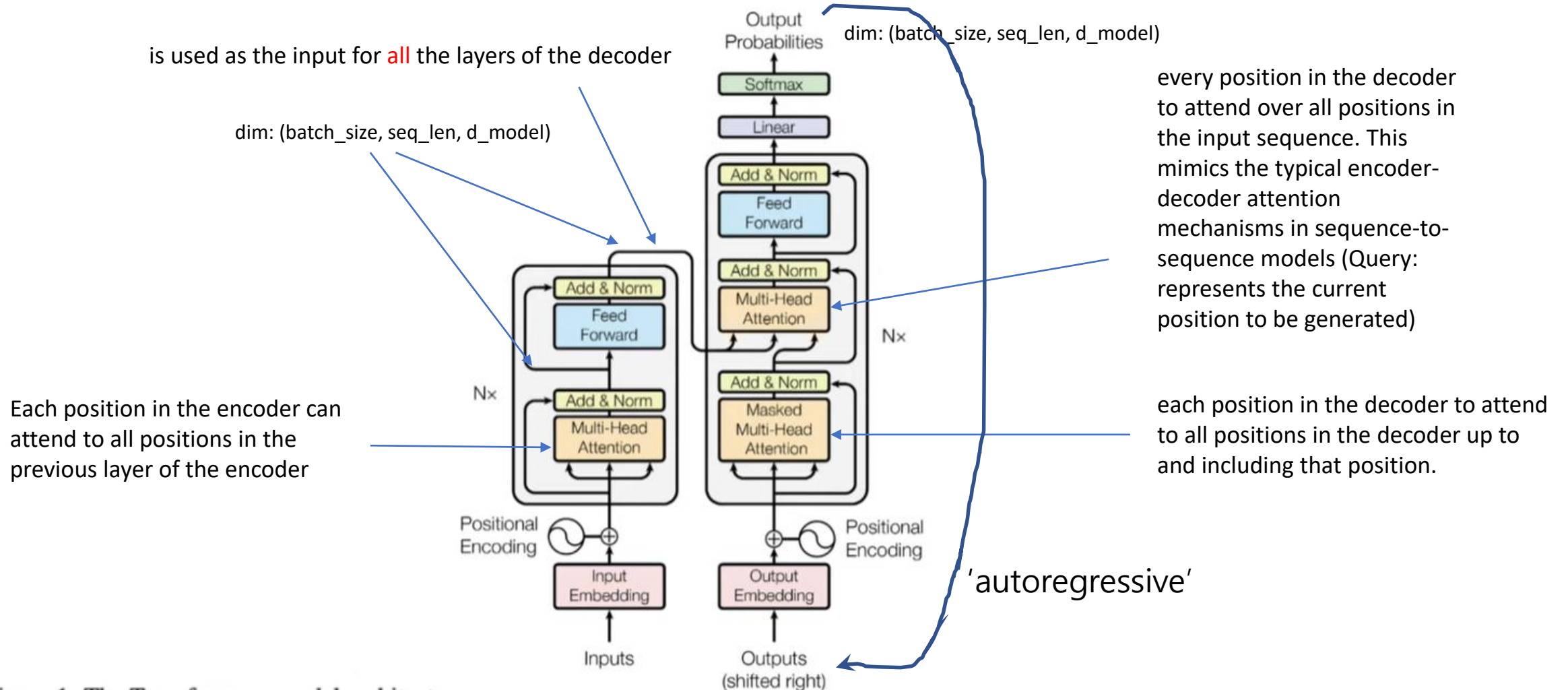


Figure 1: The Transformer - model architecture.

$\text{dim: (batch_size, seq_len, d_model)}$ # typically, $d_model = d_emb$

GPT and BERT

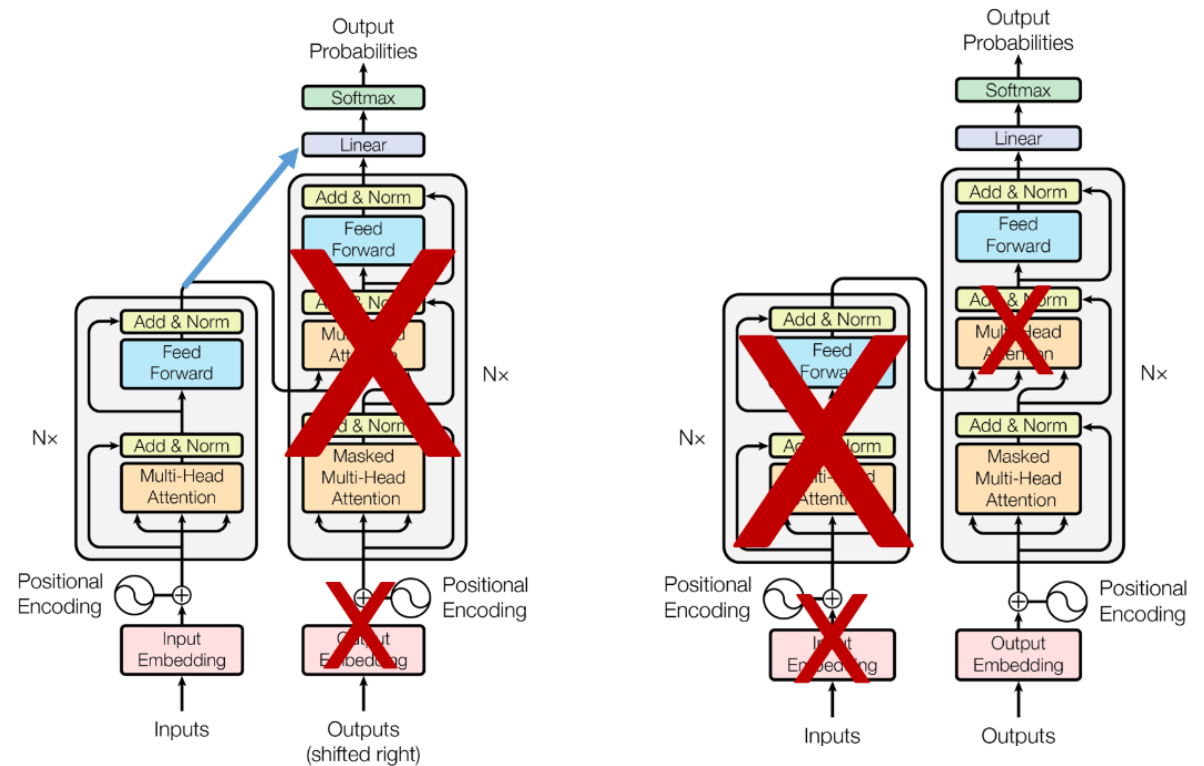
- GPT:
 - Language model (predicts the next word)
 - Unidirectional
- BERT(Bidirectional Encoder Representations from Transformers)
 - Masked Language Model
 - Predicts the blanks in the middle of sentences (bidirectional)

GPT

어제 카페 갔었어 거기 사람 많더라

BERT

어제 카페 갔었어 사람 많더라



GPT and BERT 학습

- GPT:

- Autoregressive(자기회귀) 방식으로 왼쪽
→ 오른쪽 방향으로 단어를 예측하며 학습
- 다음 단어 예측
- 문자생성 (자연어 생성)
- (ex) "The cat sat on the mat."

입력	모델의 예측
The	cat
The cat	sat
The cat sat	on
The cat sat on	the
The cat sat on the	mat

- BERT:

- Masked Language Model (MLM) 방식을 사용하며, 문장 내 특정 단어를 [MASK] 처리하고 양방향(Bidirectional)으로 문맥을 이해
- 마스킹 단어 예측
- 문장이해 (분류, 검색, 요약 등)

입력	예측 대상
The cat [MASK] on the mat.	sat
The [MASK] sat on the mat.	cat
The cat sat on the [MASK].	mat

트랜스포머(Transformer)

- Transformer 구조:
 - 입력 텍스트 처리: 단어를 숫자로 변환해 이해 (예: "Hello" → 숫자 배열)
 - 어텐션(Attention) 메커니즘: 단어 간의 관계를 이해
 - 예를 들어, "The bank"라는 문장에서 "bank"가 물리적 은행인지 강변(bank)인지 문맥을 통해 구별 가능.
 - 출력 생성: 입력 데이터를 기반으로 가장 적합한 단어를 예측해 텍스트를 생성
- 사전 학습 (Pre-training)과 미세 조정 (Fine-tuning):
 - 사전 학습: 인터넷의 방대한 데이터를 읽고 언어 패턴을 학습
 - 미세 조정: 특정 작업(예: 금융 데이터 분석)에 맞게 모델을 추가 학습
- GPT 의 구조
 - 토큰화 (Tokenization): 문장을 작은 단위(단어 또는 글자)로 나눈다.
 - 임베딩 (Word Embedding + Position Embedding): 각 단어를 숫자 (벡터)로 변환
 - 어텐션 레이어(Attention Layer): 문맥의 중요도를 계산해 더 적합한 텍스트 생성
 - 출력 레이어(Output Layer): 최종적으로 텍스트를 생성

GPT 동작 방식

- Transformer 구조:
 - 입력 텍스트 처리: 단어를 숫자로 변환해 이해 (예: "Hello" → 숫자 배열)
 - 어텐션(Attention) 메커니즘: 단어 간의 관계를 이해
 - 예를 들어, "The bank"라는 문장에서 "bank"가 물리적 은행인지 강변(bank)인지 문맥을 통해 구별 가능.
 - 출력 생성: 입력 데이터를 기반으로 가장 적합한 단어를 예측해 텍스트를 생성
- 사전 학습 (Pre-training)과 미세 조정 (Fine-tuning):
 - 사전 학습: 인터넷의 방대한 데이터를 읽고 언어 패턴을 학습
 - 미세 조정: 특정 작업(예: 금융 데이터 분석)에 맞게 모델을 추가 학습
- GPT 의 구조
 - 토큰화 (Tokenization): 문장을 작은 단위(단어 또는 글자)로 나눈다.
 - 임베딩 (Word Embedding + Position Embedding): 각 단어를 숫자 (벡터)로 변환
 - 어텐션 레이어(Attention Layer): 문맥의 중요도를 계산해 더 적합한 텍스트 생성
 - 출력 레이어(Output Layer): 최종적으로 텍스트를 생성

GPT 훈련 방식

1. 일반 문장 기반 학습 (Standard Text Training):

- 특징:
 - 인터넷 문서, 책, 기사 등의 비대화형 텍스트를 사용해 학습
 - 문장 간의 논리적 흐름과 언어적 패턴을 학습
 - GPT가 배경지식과 자연스러운 문장 구성을 배우는 데 사용
- 목적:
 - 일반 상식과 배경 지식 습득
 - 문장 구성 능력 및 일관성 있는 글쓰기 학습
- 학습 데이터:
 - Common Crawl: 대규모 웹 크롤링 데이터로, 다양한 웹사이트에서 수집 (다양한 주제, 언어)
 - Wikipedia: 방대한 지식 기반의 백과사전 (정확하고 신뢰성)
 - BooksCorpus: 다양한 장르의 책으로 구성
 - OpenWebText: Reddit에서 추천받은 웹 링크의 콘텐츠를 크롤링
 - 기타 데이터셋: 뉴스 기사, 기술 블로그, FAQ 문서 등

GPT 훈련 방식

2. 대화형 학습 (Conversational Training)

- 특징:
 - 질문-응답(Q&A), 챗봇, 상담 등의 대화형 데이터를 사용해 학습
 - 문맥(Context)을 유지하며 자연스러운 대화 흐름을 학습
 - GPT가 사용자와 대화할 때 활용되는 방식 학습 방식
- 목적:
 - 자연스러운 대화와 맥락을 반영한 응답 생성
 - 질문-답변(Q&A), 상담, 챗봇 서비스 등 대화형 AI 서비스 구현
- 학습 데이터 (추정):
 - 공개된 대화 데이터셋 (Public Datasets): Reddit 대화, Stack Exchange Q&A, Wikipedia 토론 페이지, 영화 대사 스크립트, 영화 및 TV 자막 데이터 (구어체 표현)
 - Q&A 및 고객 서비스 데이터 (QA and Support Data)
 - OpenAI가 직접 제작한 데이터 (Human Feedback Data)
 - 커뮤니티 기반 데이터 (Community-driven Data): GitHub, Quora, 포럼, 기술 블로그
- 데이터 수집 및 정제
 - 데이터 필터링 및 정제: 유해하거나 편향된 데이터 제거
 - 구조화 및 전처리: 대화형 데이터는 역할(Role)과 내용(Content)으로 구조화
 - 강화 학습 적용 (RLHF: Reinforcement Learning from Human Feedback): 사람이 직접 모델의 응답을 평가하고, 피드백을 반영

GPT 특징

- GTP 의 장점

- 대규모 데이터 기반 학습:

- 방대한 데이터로 학습하여 다양한 주제에 대해 자연스러운 대화를 가능하게 함.
 - 예시: "GPT를 사용하여 회계 용어의 정의를 빠르게 검색
 - 학생 질문: "감가상각이란 무엇인가요?"
 - GPT 답변: "감가상각은 자산의 가치를 사용 기간에 걸쳐 점차 감소시키는 회계 기법입니다."

- 업무 자동화 및 효율화

- 반복적인 업무를 줄이고 창의적인 작업에 더 많은 시간을 할애할 수 있음
 - 예시: "GPT를 활용한 보고서 요약 및 문서 생성"
 - 원문 입력: 한 문서의 내용
 - GPT 출력: 요약된 주요 항목과 결론

- 사용자 친화적 도구

- 코딩 기술 없이도 누구나 사용할 수 있는 쉬운 인터페이스 제공
 - 예시: Excel 데이터를 분석하거나 PowerPoint 슬라이드 자동 생성

GPT 특징

- 외부 툴과의 Interaction

- GPT 는 외부 툴과의 상호작용(External Tool Interaction)을 통해 자체 한계를 보완하고, 보다 정확하고 신뢰할 수 있는 정보를 제공할 수 있다.
- 이는 단순한 정보 검색 뿐 아니라, 복잡한 계산이나 논리적 추론에도 활용된다.
- (예시):
 - 실시간 주식 가격 검색 - 외부 웹 브라우저를 통해 실시간 주가 정보를 조회하여 제공
 - 최신 뉴스 검색 - 뉴스 사이트에서 최신 기사를 검색해 제공
- 작동방식:
 - 1. 질문 입력: "오늘 삼성전자 주가가 얼마야?"
 - 2. 웹 브라우저 호출: GPT가 실시간 웹 브라우저를 사용해 정보를 검색
 - 3. 응답 제공: 최신 데이터를 반영한 정확한 정보를 사용자에게 제공
- 대표적인 외부 도구
 - Python 환경 (NumPy, SymPy, SciPy 등): 복잡한 수학 계산
 - Z3 Solver: 논리 퍼즐이나 조건 만족 문제 해결
 - Prolog: 논리 기반 추론
- 추가 플러그인 (Plugin) 사용하여 확장 가능 (-> 기본 제공으로 전환 중)

GPT 특징

- 기본적으로 지원하는 외부 상호작용 기능

기능	활용 예시
웹 브라우징	최신 뉴스, 날씨, 실시간 트렌드 검색
Python 실행	수학 계산, 데이터 분석, 알고리즘 문제 해결
파일 업로드 및 처리	엑셀, CSV, PDF 파일 분석 및 요약
데이터 시각화	그래프, 차트 생성
논리적 추론 보완	퍼즐, 최적화 문제 해결 (Z3 Solver, SymPy 등 활용)

- 플러그인 없이도 가능한 외부 연동 활용 예시
 - 데이터 분석 및 시각화
 - 엑셀 파일을 업로드하면 데이터를 분석하고 시각화하여 인사이트 제공.
 - 실시간 정보 검색
 - "오늘 삼성전자 주가는 얼마인가요?" → 웹 검색을 통해 최신 정보 제공
 - 복잡한 수학적 계산
 - "1000 팩토리얼 값을 계산해줘" → Python 계산 기능을 활용해 정확한 결과 제공.
 - 논리 퍼즐 및 최적화 문제 해결
 - Cross River Puzzle과 같은 퍼즐 문제를 Python 코드나 Z3 Solver로 해결.

GPT 특징

- 외부 툴과의 Interaction (예시)

- 예시 1: 실시간 정보 검색 (웹 브라우저 연동)

- 질문: 오늘 삼성전자 주가가 얼마야?
- GPT 응답: 삼성전자의 주가는 2025년 1월 9일에 56,100원으로 마감되었습니다. [알파스퀘어\(AlphaSquare\)](#) 이는 전일 대비 1,200원 (2.09%) 하락한 수치입니다. 최근 삼성전자는 10조 원 규모의 자사주 매입 계획을 발표하였으며, [Chosun](#) 이는 투자자들의 불안 심리를 완화시키는 데 긍정적인 역할을 할 것으로 예상됩니다. 그러나 반도체 업황의 둔화와 경쟁력 약화에 대한 우려가 지속되고 있어, 주가의 본격적인 상승 전환에는 시간이 더 필요할 것이라는 분석도 있습니다.

- 예시 2: 논리 퍼즐 해결 (Z3 Solver 연동)

- 질문: 1000! (1000 팩토리얼)의 값을 알려줘.
- GPT 응답: 1000! (1000 팩토리얼)의 값은 매우 큰 수로, 다음과 같습니다: 40238726007709377354370243392300398571 ...
00

- 예시 3: 논리 퍼즐 해결 (Z3 Solver 연동)

- 질문: Cross River Puzzle을 풀어줘.
- GPT 응답: 순서: (1) 사람이 염소를 데리고 강을 건넵니다. → 강 이쪽: 늑대, 양배추 | 강 건너편: 염소 (2) 사람만 돌아옵니다. . . . 염소 → 돌아오기 → 늑대 → 염소 데리고 돌아오기 → 양배추 → 돌아오기 → 염소, 모든 것이 안전하게 강을 건넵니다!

- 예시 4: Plugin 을 사용한 외부 상호작용 (기본 제공으로 전환 중)

- "인터넷에서 최신 연구 논문을 찾아줘" → ScholarAI 플러그인
- "내 Google Calendar에 일정 추가해줘" → Zapier 플러그인
- "내 Shopify 재고 현황 알려줘" → Shopify 플러그인

GPT 특징

- GTP 의 단점

- 맥락 오류 및 환각 (Hallucination):

- 존재하지 않는 사실을 자신 있게 제시하거나 답변의 근거가 부정확할 수 있음.
 - 예시: (환각)
 - 학생 질문: 질문: "2023년 삼성의 매출은 얼마였나요?"
 - GPT 답변: "2023년 삼성의 매출은 500억 달러였습니다." (실제 수치와 다를 수 있음).“
 - 예시: (논리적 오류) – 틀린 답을 자신 있게 제시하는 점에서 환각과 유사한 특징.
 - Cross-River Puzzle (조건이 바뀐 경우에도 기존 학습된 정보에 의해 판단), Try it !

- 최신 정보 부족

- GPT는 훈련된 데이터 범위 이후의 최신 정보를 반영하지 못할 수 있음
 - 예시: 2024년 최신 정책이나 시장 변화에 대한 질문에 대해 정확하지 않은 답변 가능

- 윤리적 문제

- 비윤리적이거나 편향된 답변이 생성될 가능성
 - 해결책: GPT를 사용할 때 결과를 비판적으로 검토하고 필요하면 추가 검증

GTP 버전

- GTP 버전 별 발전 방향

- GPT-1 → GPT-2: 모델 크기 및 데이터 증가로 자연어 생성 능력 급상승
- GPT-2 → GPT-3: Few-shot Learning으로 사전 학습만으로 다양한 작업 수행
- GPT-3 → GPT-4: 멀티모달(텍스트+이미지) 이해, 고급 추론 및 창의성 대폭 향상

버전	출시 시기	파라미터 수	주요 특징
GPT-1	2018년	1.17억	최초의 GPT 모델, Unsupervised Pre-training + Fine-tuning 기반 학습
GPT-2	2019년	15억	대규모 데이터 학습, 자연어 생성 성능 급격히 향상, 공개 자연 이슈
GPT-3	2020년	1,750억	Few-shot Learning 강화, API 제공, 실생활 활용 가능성 대폭 증가
GPT-3.5	2022년	약 1,750억	대화형 AI(ChatGPT) 최적화, 코드 작성 능력 향상
GPT-4	2023년	약 1조 이상	멀티모달(Multimodal) 지원(텍스트+이미지), 고도화된 추론 능력
GPT-4 Turbo	2023년	비공개	더 빠르고 저렴한 비용으로 제공, 성능 유지 및 최적화

- Few-shot Learning:

- 예시 기반 학습 (사전 학습을 통해 축적된 지식을 프롬프트(예시 기반 지시문)를 통해 활용)
- Few-shot Learning에서 제공하는 예시들은 모델의 학습(training)에 사용되는 것이 아니라, 실제 예측(inference) 시에 추가적인 입력(Prompt)으로 작동한다.
- 대량의 데이터로 사전 학습(Pre-training)된 모델이, 소량의 예시만으로 새로운 작업을 빠르게 학습하고 수행하는 방법 (모델 파라미터는 조정하지 않음)
- 반면, Fine-tuning이나 대화형 학습은 모델 자체를 업데이트하거나 피드백을 반영해 성능을 개선한다.

GTP 버전

- ChatGPT and GPT3.5

- ChatGPT는 GPT-3.5 또는 GPT-4 모델을 기반으로 만들어진 대화형 서비스.
- GPT-3.5는 OpenAI가 개발한 언어 모델(Language Model)
- 즉, GPT-3.5는 기반 모델이고, ChatGPT는 이 모델을 이용해 대화형 AI 서비스로 최적화된 형태. (GPT-3.5는 강력한 엔진이며, ChatGPT는 이 엔진을 사용해 만든 자동차로 비유가 가능함)

구분	GPT-3.5	ChatGPT
기본 개념	자연어 처리(NLP)를 위한 언어 모델	GPT-3.5 또는 GPT-4 기반의 대화형 서비스
사용 목적	다양한 작업(텍스트 생성, 번역, 요약 등) 수행	자연스러운 대화, 질문 응답, 작업 자동화
최적화 방향	범용 자연어 처리 (NLP 전반)	대화에 최적화 (자연스러운 응답, 유저 친화적)
출시 시기	2022년 (GPT-3.5 공개)	2022년 11월 (ChatGPT 서비스 시작)
기능 차이	모델 자체만 제공 (API 형태)	대화 UI, 맥락 유지, 사용자 피드백 반영
기술적 차별점	Fine-tuning 없음 (범용 모델)	RLHF(인간 피드백 강화학습)로 대화 최적화

GTP 활용

- GTP 의 사용방법
 - 대화형 사용(Chat-based Interaction)
 - 프로그래밍 기반 활용(Programmatic Interaction)

구분	대화형 사용 (Chat-based Interaction)	프로그래밍적 사용 (Programmatic Usage)
사용 방식	웹 또는 앱에서 채팅으로 질문/명령 수행	코드나 API 호출로 GPT를 제어 및 응용
대상 사용자	일반 사용자, 비개발자	개발자, 연구자, 데이터 사이언티스트
대표 도구	ChatGPT, Bing Chat, 웹 기반 챗봇	OpenAI API, Hugging Face Transformers
커스터마이징	불가능 (정해진 기능만 사용)	가능 (Fine-tuning, 파이프라인 구성)
활용 범위	간단한 질문/답변, 글쓰기, 요약	모델 학습, 데이터 처리, 애플리케이션 통합

OpenAI API

- 개요

- OpenAI가 제공하는 클라우드 기반 API로, ChatGPT(GPT-3.5, GPT-4)와 같은 대규모 언어 모델(Large Language Model, LLM)을 간편하게 사용할 수 있도록 제공
- 사용자는 API 호출을 통해 모델을 직접 학습하거나 튜닝할 필요 없이 고성능 AI 모델을 이용할 수 있다.
- API 키 발급 (<https://platform.openai.com/docs/overview>)

```
import openai

openai.api_key = "YOUR_API_KEY"

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "What is the capital of France?"}
    ]
)

print(response['choices'][0]['message']['content'])
```

The capital of France is Paris.

✓ 주요 기능

- 대화형 응답: ChatGPT (GPT-3.5, GPT-4).
- 코드 생성: Codex (프로그래밍 코드 작성).
- 텍스트 임베딩: 텍스트 검색, 추천.
- 이미지 생성: DALL-E.
- 음성 → 텍스트 변환: Whisper.

Hugging Face Transformers

- 개요

- 오픈소스 라이브러리로, 다양한 NLP 및 생성 AI 모델(GPT, BERT, T5, BLOOM 등)을 로컬 환경이나 클라우드에서 실행할 수 있다.
- 사전 훈련(pretrained)된 모델을 다운로드하거나, Fine-tuning(미세 조정)을 통해 특정 작업에 맞게 학습시킬 수 있다.

```
from transformers import AutoTokenizer, AutoModelForCausalLM

model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

inputs = tokenizer("What is the capital of France?", return_tensors="pt")
outputs = model.generate(**inputs, max_length=20)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

What is the capital of France? The capital of France is Paris.

- ✓ 주요 기능

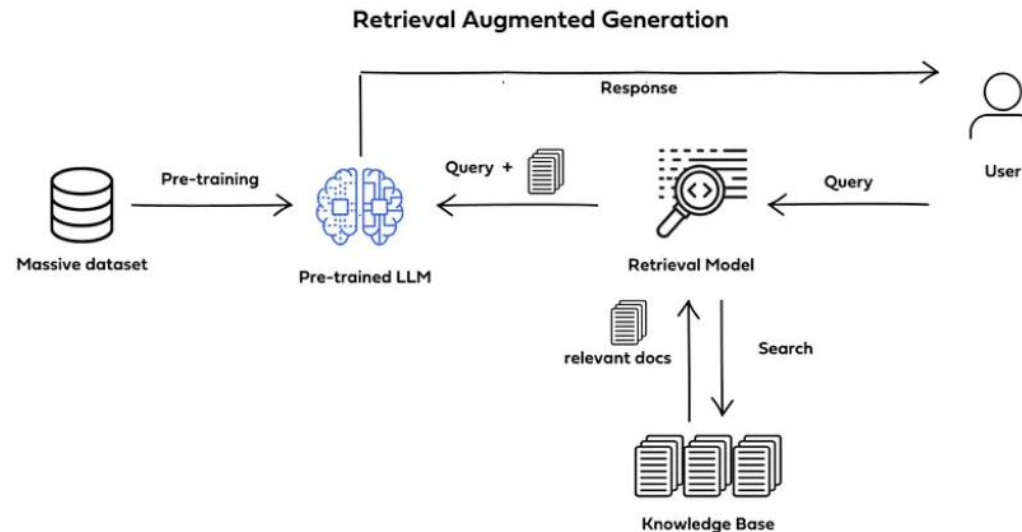
- 다양한 모델 지원: GPT, BERT, T5, DistilBERT, BLOOM, Whisper 등.
- 모델 미세 조정(Fine-tuning): 특정 데이터셋에 맞게 모델을 재학습.
- 로컬/클라우드 실행: 사양에 맞게 CPU, GPU, TPU 사용 가능.
- 다양한 태스크 지원: 텍스트 분류, 번역, 요약, 텍스트 생성, 질문 응답 등.

OpenAI API and Hugging Face Transformers

항목	OpenAI API	Hugging Face Transformers
운영 방식	클라우드 서비스 (API 호출)	로컬 실행 및 클라우드 실행
지원 모델	GPT-3.5, GPT-4, DALL-E, Codex, Whisper	GPT, BERT, T5, BLOOM, Whisper 등 수천 개의 모델
설치 필요성	없음 (API 호출로 사용)	필요 (<code>transformers</code> , <code>torch</code> 설치)
커스터마이징	불가능 (미리 학습된 모델 사용)	가능 (Fine-tuning 및 직접 학습 가능)
성능 최적화	OpenAI 서버에서 최적화된 환경 제공	사용자의 하드웨어 성능에 의존 (GPU/TPU 활용 가능)
비용	유료 (사용량 기반 결제)	무료 (로컬 사용), Hugging Face Inference API는 유료
확장성	대규모 데이터 처리에 최적화 (서버 인프라 활용)	하드웨어 성능에 따라 제한
응답 속도	빠름 (서버 최적화)	로컬 환경에 따라 다름
보안성	민감한 데이터는 외부 서버로 전송됨 (데이터 유출 우려)	로컬에서 직접 실행 가능 (데이터 보안 용이)

RAG(Retrieval-Augmented Generation)

- RAG의 기본 개념
 - Retrieval(검색): 외부 데이터 소스에서 관련 정보를 검색
 - Generation(생성): 검색한 정보를 바탕으로 자연스러운 응답을 생성.
 - (예시) "GPT에게 특정 제품의 최근 리뷰를 기반으로 고객 피드백 요약을 요청하면, GPT는 외부 데이터베이스에서 해당 제품 리뷰를 검색한 후 이를 바탕으로 요약을 생성.



Stock Price Prediction

1. ARIMA (Auto-Regressive Integrated Moving Average) 모델

- 시계열 데이터(Time Series Data)를 예측하기 위해 가장 널리 사용되는 통계적 모델 중 하나
- 원리:
 - 선형 시계열 모델로, 데이터의 자기회귀(AR), 이동평균(MA), 차분(I)을 결합하여 예측
 - 시계열 데이터의 추세(Trend)와 계절성(Seasonality) 없이 자기 상관성을 기반으로 예측.
- 특징:
 - 장점: 수학적으로 해석이 명확하며, 단기 예측에 강하다. 데이터가 정상성(Stationarity)을 가진다면 매우 효과적이다.
 - 단점: 비선형 패턴이나 복잡한 시장 변화를 반영하기 어려움. 장기 예측 시 오차가 누적된다.

➤ 자기회귀(Auto-Regression): 과거 데이터가 현재 값에 미치는 영향력을 설명

$$Y_t = c + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p} + \epsilon_t$$

- Y_t : 시점 t 의 실제 값
- c : 상수항 (bias)
- ϕ_i : 자기회귀 계수(AR coefficient)
- ϵ_t : 백색잡음(White Noise), 평균이 0인 랜덤 오차

➤ 이동평균(Moving Average): 과거의 예측 오차를 현재 값에 반영하여 모델의 정확도를 높인다.

$$Y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

- μ : 평균
- ϵ_t : 백색잡음(white noise)
- θ_i : 이동평균 계수(MA coefficient)

Stock Price Prediction

- 차분(Integrated): 시계열 데이터의 추세(Trend)를 제거하고, 데이터의 정상성(Stationarity)을 확보하기 위해 사용된다.
- ARIMA(p,d,q): AR, I, MA를 결합해 과거 값과 오차를 기반으로 미래 값을 예측한다.

1차 차분(First-order differencing):

$$Y'_t = Y_t - Y_{t-1}$$

d차 차분(differencing):

$$Y_t^{(d)} = (1 - B)^d Y_t$$

- B : 시프트 연산자(Backward Shift Operator), $BY_t = Y_{t-1}$
- d : 차분의 횟수

$$Y'_t = c + \sum_{i=1}^p \phi_i Y'_{t-i} + \epsilon_t + \sum_{j=1}^q \theta_j \epsilon_{t-j}$$

- Y'_t : **차분(d)**이 적용된 시계열 데이터
- ϕ_i : 자기회귀 계수
- θ_j : 이동평균 계수
- ϵ_t : 오차항

➤ ARIMA 모델 동작 절차

- 정상성 확인: 데이터가 정상성(stationary)을 만족하는지 확인 (ADF 검정 사용)
- 차분(I) 적용: 정상성을 만족하지 않는다면, 차분을 적용 (d 결정)
- 모델 차수 선정: p와 q의 값을 결정 (ACF, PACF 그래프 이용)
- 모델 학습: 주어진 데이터로 ARIMA 모델을 학습.
- 예측 및 평가: 미래 값을 예측하고 모델 성능 평가.

Seasonal ARIMA model

- **Seasonal ARIMA**

- Include additional seasonal terms in the ARIMA model

$$\text{ARIMA} \quad \underbrace{(p, d, q)}_{\substack{\uparrow \\ \text{Non-seasonal part} \\ \text{of the model}}} \quad \underbrace{(P, D, Q)_m}_{\substack{\uparrow \\ \text{Seasonal part of} \\ \text{of the model}}} \quad m : \text{number of observations per year}$$

- For example, $\text{ARIMA}(1,1,1)(1,1,1)_4$ model is a quarterly data ($m=4$)

- **ARIMA & SARIMA**

- It aims to find the appropriate values of the parameters (p, d, q) and (P, D, Q, S) in order to minimize the remaining signal, leaving residuals that resemble white noise.
- Model refinement: If the residuals exhibit any patterns or systematic behavior, further adjustments to the model may be needed.

Stock Price Prediction

2. Prophet (by Facebook)

- 원리:
 - 추세(Trend), 계절성(Seasonality), 휴일 효과(Holidays)와 같은 시계열 데이터의 구조적 패턴을 분해하여 예측하는 모델
 - 비정상성, 결측치, 계절성(Seasonality), 이벤트 효과(Holiday Effect) 등을 효과적으로 처리하도록 설계.
 - ARIMA보다 사용이 쉽고, 시계열 데이터의 복잡한 패턴을 잘 포착.
- 특징:
 - 장점: 비정상성 데이터와 결측치에 강하고, 휴일/이벤트 효과 반영 가능하다. 파라미터 설정이 쉬워 비전문가도 쉽게 사용할 수 있다.
 - 단점: 복잡한 시장 패턴(비선형성, 급격한 변화)을 반영하기 어렵고 예측 결과가 평균적인 추세로 수렴.

➤ Prophet 모델의 수학적 구성

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

- $y(t)$: 시점 t 에서의 관측값(Observed Value)
- $g(t)$: **추세(Trend)** 함수 (비선형 성장 모델)
- $s(t)$: **계절성(Seasonality)** 함수 (주기적 패턴)
- $h(t)$: **휴일 효과(Holiday Effect)** (특정 이벤트나 휴일의 영향)
- ϵ_t : **오차항(Noise)** (백색잡음, White Noise)

➤ Prophet 모델 동작 과정

- 데이터 분해(Decomposition): 추세(Trend), 계절성(Seasonality), 이벤트(Holiday Effect)로 분해.
- 모델 학습(Fitting): 각 구성 요소의 최적 계수를 학습. MLE (Maximum Likelihood Estimation) 사용
- 미래 예측: 학습된 구성 요소를 합쳐서 미래 값 예측

Stock Price Prediction

- ARIMA vs Prophet 비교

비교 항목	ARIMA	Prophet
추세(Trend)	직접 차분으로 제거 (비선형 반영 어려움)	선형/비선형 추세 자동 탐지
계절성 (Seasonality)	수동으로 추가해야 함	자동으로 탐지 및 적용
이벤트 효과	직접 모델링 필요	휴일 및 이벤트 효과 자동 반영
사용 편의성	복잡함 (모델 차수 설정 필요)	매우 쉬움 (자동 조정)
복잡한 패턴 반영	제한적 (선형 모델 기반) ↓	비선형 반영 가능 (일부 한계)

Stock Price Prediction

- Prophet 사용 예시

```
from prophet import Prophet
import pandas as pd
```

```
# 1. 데이터 불러오기 (Date, y 컬럼 필요)
```

```
df = pd.read_csv('stock_data.csv')
```

```
df.rename(columns={'Date': 'ds', 'Close': 'y'}, inplace=True)
```

```
# 2. 모델 생성 및 학습
```

```
model = Prophet()
```

```
model.fit(df)
```

```
# 3. 미래 데이터프레임 생성
```

```
future = model.make_future_dataframe(periods=30)
```

```
# 4. 예측 수행
```

```
forecast = model.predict(future)
```

```
# 5. 예측 결과 시각화
```

```
model.plot(forecast)
```

```
model.plot_components(forecast)
```

Stock Price Prediction

3. LSTM (Long Short-Term Memory)

- RNN(Recurrent Neural Network) 모델의 한 종류로 장기 의존성(Long-term Dependency) 문제를 해결하기 위해 설계된 딥러닝 기반 시계열 예측 모델.
- 원리:
 - 순환 신경망(RNN)의 한 종류로, 장기 의존성(Long-term dependency) 문제를 해결
 - 과거의 정보를 장기적으로 기억하고, 복잡한 패턴과 비선형 시계열을 학습
- 특징:
 - 장점: 비선형 패턴과 복잡한 시장 변동성을 잘 포착한다.
 - 뉴스, 거래량, 경제 지표 등 다양한 입력을 통합해 예측 가능
 - 단점: 대규모 데이터와 많은 학습 시간이 필요하고 과적합(overfitting) 위험이 있다.
 - 해석이 어려움 (Black-Box)

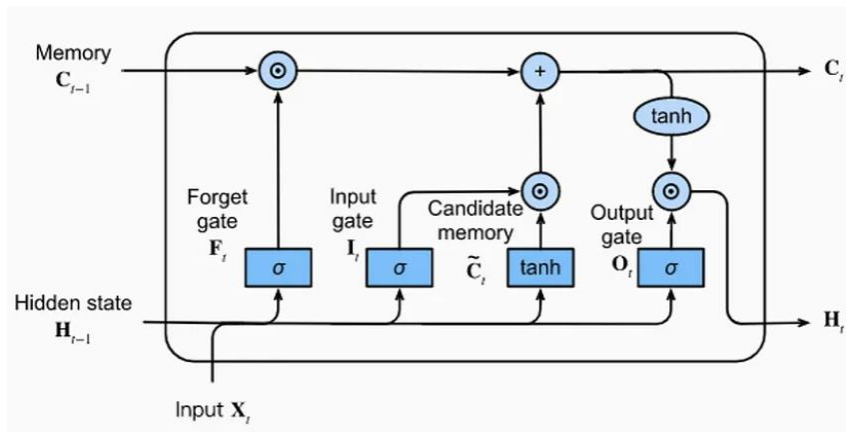


Figure 1: Architecture of a LSTM Unit (Credits: https://d2l.ai/chapter_recurrent-modern/lstm.html)

1. **Forget Gate:** 이전 정보 중 어떤 것을 버릴지 결정

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

2. **Input Gate:** 현재 정보를 얼마나 반영할지 결정

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

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

3. **Cell State 업데이트:** 셀 상태 업데이트

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4. **Output Gate:** 최종 출력 결정

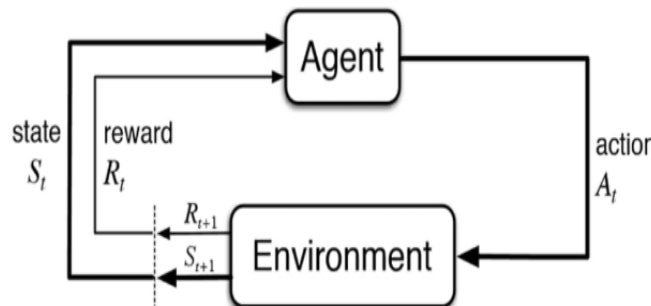
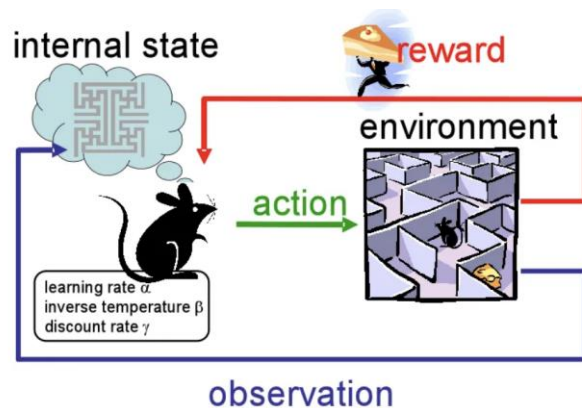
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

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

Stock Price Prediction

4. 강화학습 (Reinforcement Learning)

- 원리:
 - 에이전트(Agent)가 환경(Environment)과 상호작용하며 보상(Reward)을 극대화하도록 학습
 - 주식 시장에서는 에이전트가 매수(Buy), 매도(Sell), 관망(Hold) 중 하나의 행동을 선택하고, 이에 따른 수익 또는 손실로 보상을 받음
- 특징:
 - 장점: 거래 전략(Trading Strategy)을 직접 학습, 시장의 복잡한 상호작용과 비선형성을 반영
 - 단점: 환경 모델링이 매우 중요하며, 잘못된 보상 함수나 상태 설계는 학습 실패로 이어짐. 실제 시장에서는 탐험(Exploration)이 제한됨.



- 상태(State, S): 시장의 현재 상태(주가, 거래량, 기술적 지표 등)
- 행동(Action, A): 에이전트의 결정 (매수, 매도, 관망)
- 보상(Reward, R): 행동에 따른 수익 또는 손실
- 정책(Policy, π): 주어진 상태에서 어떤 행동을 선택할지 결정하는 전략
- 환경(Environment): 주가 변동 데이터 (과거 데이터 시뮬레이션)

Stock Price Prediction (예시)

- RL 사용 예시

- 상태(State): 현재 주가, 보유 현금, 보유 주식 수
- 행동(Action): 매수(Buy), 매도(Sell), 관망(Hold)
- 보상(Reward): 잔고(Balance) 증가 또는 감소

- 환경 구성: StockTradingEnv (주식거래환경)

- 상태(State): 현재 주가, 잔고, 보유 주식 수
- 행동(Action): 0 (관망), 1 (매수), 2 (매도)
- 보상(Reward): 현재 총 자산 가치

- Q-Learning 에이전트

- 탐험(Exploration)과 활용(Exploitation) 전략 적용
- Q-테이블을 사용해 상태-행동 값을 학습하고 업데이트

- 학습 과정

- 여러 에피소드 동안 반복 학습
- 보상 변화 시각화

강화학습 환경 정의

```
class StockTradingEnv:
```

```
    def __init__(self, prices):
        self.prices = prices
        self.n_steps = len(prices)
        self.reset()
```

```
    def reset(self):
        self.current_step = 0
        self.balance = 1000 # 초기 자본금
        self.stock_owned = 0
        return self._get_state()
```

```
    def _get_state(self):
        return [self.prices[self.current_step], self.balance, self.stock_owned]
```

```
    def step(self, action):
        price = self.prices[self.current_step]
```

```
        # 행동: 0 = 관망, 1 = 매수, 2 = 매도
```

```
        if action == 1: # 매수
            if self.balance >= price:
                self.stock_owned += 1
                self.balance -= price
```

```
        elif action == 2: # 매도
            if self.stock_owned > 0:
                self.stock_owned -= 1
                self.balance += price
```

```
        self.current_step += 1
        done = self.current_step == self.n_steps - 1
```

```
        # 보상: 현재 총 자산 (현금 + 주식 가치)
        total_asset = self.balance + self.stock_owned * price
        reward = total_asset
```

```
        return self._get_state(), reward, done
```


Stock Price Prediction (예시)

```
# Q-Learning 에이전트
class QLearningAgent:
    def __init__(self, actions, learning_rate=0.1, discount_factor=0.95,
epsilon=1.0, epsilon_decay=0.995):
        self.actions = actions
        self.lr = learning_rate
        self.gamma = discount_factor
        self.epsilon = epsilon
        self.epsilon_decay = epsilon_decay
        self.q_table = {} # Q-테이블 초기화

    def get_state_key(self, state):
        return tuple(np.round(state, 2))

    def choose_action(self, state):
        state_key = self.get_state_key(state)
        if random.uniform(0, 1) < self.epsilon:
            return random.choice(self.actions) # 탐험(Explore)
        else:
            self.q_table.setdefault(state_key, [0] * len(self.actions))
            return np.argmax(self.q_table[state_key]) # 활용(Exploit)

    def learn(self, state, action, reward, next_state):
        state_key = self.get_state_key(state)
        next_state_key = self.get_state_key(next_state)

        self.q_table.setdefault(state_key, [0] * len(self.actions))
        self.q_table.setdefault(next_state_key, [0] * len(self.actions))

        current_q = self.q_table[state_key][action]
        max_next_q = max(self.q_table[next_state_key])

        # Q-값 업데이트
        self.q_table[state_key][action] += self.lr * (reward + self.gamma *
max_next_q - current_q)

        # 탐험률 감소
        self.epsilon *= self.epsilon_decay
```

```
# 주가 시뮬레이션 데이터 생성 (간단한 랜덤 워크)
def generate_stock_data(days=100):
    price = [100] # 시작 주가 100
    for _ in range(days - 1):
        price.append(price[-1] * (1 + np.random.normal(0, 0.01)))
    return price

# 메인 학습 과정
def train_agent(epochs=100):
    stock_prices = generate_stock_data()
    env = StockTradingEnv(stock_prices)
    agent = QLearningAgent(actions=[0, 1, 2]) # 0: Hold, 1: Buy, 2: Sell

    rewards = []

    for episode in range(epochs):
        state = env.reset()
        total_reward = 0

        done = False
        while not done:
            action = agent.choose_action(state)
            next_state, reward, done = env.step(action)
            agent.learn(state, action, reward, next_state)
            state = next_state
            total_reward += reward

        rewards.append(total_reward)
        print(f"Episode {episode + 1}: Total Reward = {total_reward:.2f}")

    # 학습 결과 시각화
    plt.plot(rewards)

# 학습 실행
train_agent()
```

Stock Price Prediction (예시)

- RL 사용 예시

- 상태(State): 현재 주가, 보유 현금, 보유 주식 수
- 행동(Action): 매수(Buy), 매도(Sell), 관망(Hold)
- 보상(Reward): 잔고(Balance) 증가 또는 감소

- 환경 구성: StockTradingEnv (주식거래환경)

- 상태(State): 현재 주가, 잔고, 보유 주식 수
- 행동(Action): 0 (관망), 1 (매수), 2 (매도)
- 보상(Reward): 현재 총 자산 가치

- Q-Learning 에이전트

- 탐험(Exploration)과 활용(Exploitation) 전략 적용
- Q-테이블을 사용해 상태-행동 값을 학습하고 업데이트

- 학습 과정

- 여러 에피소드 동안 반복 학습
- 보상 변화 시각화

- 실제 주가 예측을 위해 과거 데이터 사용 (실제 주식 데이터)

- 주식 시장은 실시간으로 상호작용하며 탐험하기 어렵다. (따라서, 과거 시장 데이터를 시뮬레이션 환경으로 사용해 에이전트가 학습한다)
- yfinance (야후), Alpha Vantage API
- Quandl
- 네이버 금융, KRX, 키움증권, NH 투자증권, 등 (한국)

강화학습 환경 정의

```
class StockTradingEnv:
```

```
    def __init__(self, data):
```

```
        self.data = data['Close'].values # 실제 주가 데이터 사용
```

```
        self.n_steps = len(self.data)
```

```
        self.reset()
```

```
    def reset(self):
```

```
        self.current_step = 0
```

```
        self.balance = 10000 # 초기 자본금
```

```
        self.stock_owned = 0
```

```
        return self._get_state()
```

```
    def _get_state(self):
```

```
        return [self.data[self.current_step], self.balance, self.stock_owned]
```

```
    def step(self, action):
```

```
        price = self.data[self.current_step]
```

```
        if action == 1: # 매수
```

```
            if self.balance >= price:
```

```
                self.stock_owned += 1
```

```
                self.balance -= price
```

```
        elif action == 2: # 매도
```

```
            if self.stock_owned > 0:
```

```
                self.stock_owned -= 1
```

```
                self.balance += price
```

```
        self.current_step += 1
```

```
        done = self.current_step == self.n_steps - 1
```

```
        total_asset = self.balance + self.stock_owned * price
```

```
        reward = total_asset
```

```
        return self._get_state(), reward, done
```

Stock Price Prediction (예시)

Q-Learning 에이전트

```
class QLearningAgent:
```

```
    def __init__(self, actions, learning_rate=0.1, discount_factor=0.95,
                  epsilon=1.0, epsilon_decay=0.995):
```

```
        self.actions = actions
```

```
        self.lr = learning_rate
```

```
        self.gamma = discount_factor
```

```
        self.epsilon = epsilon
```

```
        self.epsilon_decay = epsilon_decay
```

```
        self.q_table = {} # Q-테이블 초기화
```

```
    def get_state_key(self, state):
```

```
        return tuple(np.round(state, 2))
```

```
    def choose_action(self, state):
```

```
        state_key = self.get_state_key(state)
```

```
        if random.uniform(0, 1) < self.epsilon:
```

```
            return random.choice(self.actions) # 탐험(Explore)
```

```
        else:
```

```
            self.q_table.setdefault(state_key, [0] * len(self.actions))
```

```
            return np.argmax(self.q_table[state_key]) # 활용(Exploit)
```

```
    def learn(self, state, action, reward, next_state):
```

```
        state_key = self.get_state_key(state)
```

```
        next_state_key = self.get_state_key(next_state)
```

```
        self.q_table.setdefault(state_key, [0] * len(self.actions))
```

```
        self.q_table.setdefault(next_state_key, [0] * len(self.actions))
```

```
        current_q = self.q_table[state_key][action]
```

```
        max_next_q = max(self.q_table[next_state_key])
```

```
        # Q-값 업데이트
```

```
        self.q_table[state_key][action] += self.lr * (reward + self.gamma *
                                                         max_next_q - current_q)
```

```
        # 탐험률 감소
```

```
        self.epsilon *= self.epsilon_decay
```

```
# 주가 시뮬레이션 데이터
```

```
import yfinance as yf
```

```
import pandas as pd
```

```
# 실제 주식 데이터 가져오기 (예: Apple(AAPL))
```

```
def get_real_stock_data(ticker='AAPL', start_date='2020-01-01',
                        end_date='2023-01-01'):
```

```
    stock_data = yf.download(ticker, start=start_date, end=end_date)
```

```
    stock_data.reset_index(inplace=True)
```

```
    return stock_data[['Date', 'Close']]
```

```
# 실제 데이터를 불러옴
```

```
real_data = get_real_stock_data(ticker='AAPL', start_date='2020-01-01',
                                end_date='2023-01-01')
```

```
# 실제 데이터로 강화학습 환경 생성
```

```
env = StockTradingEnv(real_data)
```

```
# Q-Learning Agent 사용
```

```
agent = QLearningAgent(actions=[0, 1, 2]) # 0: Hold, 1: Buy, 2: Sell
```

```
# 학습 반복
```

```
state = env.reset()
```

```
done = False
```

```
while not done:
```

```
    action = agent.choose_action(state)
```

```
    next_state, reward, done = env.step(action)
```

```
    agent.learn(state, action, reward, next_state)
```

```
    state = next_state
```

Stock Price Prediction

- 주가 예측을 위한 강화학습 데이터 생성
 - 행동-보상-정책 학습이라는 강화학습의 구조는 동일
 - 다만, 환경이 정적이고 탐험이 제한적이라는 점에서 일반적인 강화학습과는 차이가 있다.

구분	일반적인 강화학습	주가 예측 강화학습
데이터 수집	에이전트가 환경과 상호작용하여 직접 데이터 생성	과거 데이터를 사용한 시뮬레이션 기반 학습
탐험 (Exploration)	실시간 탐험 가능 (다양한 행동 시도)	과거 데이터 기반으로 제한된 탐험
보상(Reward)	실시간 피드백	시뮬레이션에서의 보상 계산
행동(Action)	자유로운 행동 선택 및 결과 반영	매수, 매도, 관망과 같은 행동은 동일하게 적용
환경 (Environment)	동적 환경 (실시간 변화)	정적 환경 (고정된 과거 데이터)

- 실제 투자 기관에서 사용하는 주가 예측 및 투자 전략
 - 통계적 및 수리적 모델 (Statistical & Quantitative Models): Factor model, ARIMA, GARCH
 - 지도학습 머신러닝 기반 모델:
 - 과거 데이터를 기반으로 주가를 예측
 - Feature 기반 주가 예측: RF, XGBoost, LightGBM
 - 시계열 데이터 기반: LSTM, GRU
 - 비지도학습 머신러닝 기반:
 - 군집(Clustering) 및 이상 탐지(Anomaly Detection)
 - K-Means, DBSCAN, PCA → 데이터 패턴 탐색
 - 강화학습:
 - 트레이딩 전략을 최적화하기 위해 사용
 - 실시간 시장에서의 적용은 제한적.
 - 데이터 기반 투자 (Data-Driven Investment)
 - 대체 데이터(Alternative Data)를 사용
 - 퀀트 트레이딩(Quantitative Trading)

Stock Price Prediction

- 현실적인 주가 예측 모델
 - 실제 투자 기관은 강화학습을 직접적으로 사용하기보다는, 통계 모델과 머신러닝을 기반으로 시장을 예측하고 전략을 선호한다.
 - 강화학습(RL)은 아직도 탐험의 한계와 리스크 관리 문제로 인해 제한적으로 활용되지만, 포트폴리오 최적화나 트레이딩 전략 설계에서 보조적으로 사용되고 있다.
 - 현실적으로는 데이터 기반 전략과 퀀트 트레이딩(?)을 통해 안정적이고 지속 가능한 수익을 추구한다.
- 실제 기관에서 선호하는 방법

구분	실제 활용 방법
전통적 모델	통계 모델(ARIMA, GARCH) → 단기 예측, 리스크 관리
머신러닝 모델	Random Forest, XGBoost → 비선형 패턴 탐지 및 예측
딥러닝 모델	LSTM, GRU → 시계열 예측, 패턴 인식
강화학습(RL)	전략 최적화, 포트폴리오 관리 → 제한적 적용(리스크 통제)
데이터 기반 전략	뉴스, SNS, 대체 데이터 → Alpha 생성