AI Programming

데이터 마이닝 / 생성형 AI

06. 딥러닝 기초

first coding

딥러닝 개요

딥러닝의 정의와 활용 분야

머신러닝과 딥러닝의 차이

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

딥러닝의 정의와 활용 분야

- 딥러닝

  - "사람의 뇌가 정보를 처리하고 학습하는 방식을 모방한
    인공 신경망(Artificial Neural Network)을 기반으로 한 머신러닝의 하위 분야"
  - 딥러닝의 핵심은 데이터를 바탕으로 복잡한 패턴을 학습하고 문제를 해결하는 것
  - 계층적인 데이터 학습
    - 딥러닝은 여러 층(레이어)으로 구성된 신경망을 사용
    - 각 신경망 층은 데이터에서 점점 더 높은 수준의 추상적인 특징을 학습할 수 있음

      예 : 이미지 데이터 → 첫 번째 층 : 엣지(선),
      두 번째 층 : 간단한 패턴(모양),
      마지막 층 : 복잡한 패턴(얼굴, 사물 등).

딥러닝의 정의와 활용 분야

- 딥러닝
  - 딥러닝의 주요 특징
    - 비정형 데이터 처리에 강함
      - 딥러닝은 텍스트, 이미지, 음성 같은 구조화되지 않은 데이터를 처리하는 데 매우 강력함
      - 이러한 데이터를 처리할 때 사람이 별도로 특징을 추출할 필요 없이 모델이 데이터를 통해 직접 학습
    - 대규모 데이터와 연산 요구
      - 딥러닝은 대량의 데이터와 강력한 계산 자원(GPU/TPU)가 필요
      - 데이터를 많이 사용할수록 더 정교한 모델을 학습할 수 있음
    - 자동화된 학습:
      - 딥러닝은 데이터를 기반으로 스스로 학습 규칙을 생성
      - 이 규칙을 통해 새로운 데이터에 대해 예측이나 분류를 수행

딥러닝의 정의와 활용 분야

- 딥러닝

  - 딥러닝의 활용 분야
    - 컴퓨터 비전 : 이미지 인식, 객체 탐지, 자율 주행.
    - 자연어 처리 (NLP) : 번역, 텍스트 요약, 챗봇.
    - 음성 처리 : 음성 인식, 음성 합성.
    - 의료 : 질병 진단, 의료 영상 분석.
    - 추천 시스템 : 사용자 맞춤 추천 (예: 넷플릭스, 유튜브).
    - 금융 : 사기 탐지, 금융 시장 예측.

머신러닝과 딥러닝의 차이

- **머신러닝과 딥러닝**

  - 머신러닝
    - 컴퓨터가 명시적인 프로그래밍 없이 데이터를 학습하여 예측이나 분류를 수행할 수 있게 하는 기술
    - 데이터에서 특징(feature)을 추출하고, 이 특징을 바탕으로 패턴을 학습하여 결과를 예측
    - 머신러닝 알고리즘은 종종 사람이 수작업으로 데이터를 전처리하고 중요한 특징을 선정해야 함

  - 딥러닝
    - 머신러닝의 하위 분야로, 인공 신경망(Artificial Neural Network)을 기반으로 한 기술
    - 데이터를 학습하는 과정에서 특징을 자동으로 추출
    - 계층적인 학습(여러 레이어)을 통해 복잡한 패턴을 이해하고 학습

머신러닝과 딥러닝의 차이

- **특징 엔지니어링**

  - 머신러닝
    - 데이터를 학습하기 전에 사람이 주요 특징(feature)을 선정하고, 이를 모델에 제공해야 함
      예: 이미지에서 색상, 모서리, 텍스처 같은 특징을 사람이 추출.
    - 알고리즘은 제공된 특징을 기반으로 학습.

  - 딥러닝
    - 자동 특징 학습(Auto Feature Learning)이 가능
    - 모델이 원시 데이터(raw data) 를 입력 받아, 데이터에서 특징을 자동으로 추출하고 학습
      예: 이미지 데이터를 입력하면 모델이 자동으로 엣지, 패턴, 사물을 학습.

머신러닝과 딥러닝의 차이

- **데이터 요구량**

  - 머신러닝
    - 상대적으로 적은 양의 데이터로도 학습이 가능
    - 작은 데이터셋에서도 좋은 성능을 낼 수 있도록 설계된 알고리즘이 많음

  - 딥러닝
    - 매우 큰 데이터셋이 필요
    - 딥러닝 모델은 계층이 깊고 복잡하기 때문에, 충분히 많은 데이터를 사용해야 높은 성능을 보장
      예: 자율주행 차량의 딥러닝 모델은 수백만 개의 이미지와 동영상을 학습

머신러닝과 딥러닝의 차이

- **연산 자원**

  - 머신러닝
    - 일반적인 CPU에서도 작동 가능하며, 연산 자원 요구가 상대적으로 적음
    - 모델이 단순하고 학습 과정이 비교적 빠름

  - 딥러닝
    - GPU(Graphics Processing Unit)나 TPU(Tensor Processing Unit) 같은 강력한 연산 자원이 필요
    - 모델의 계층이 많아 학습 과정이 시간이 많이 걸리고, 계산 복잡도가 높음

인공 신경망(Artificial Neural Network, ANN)

- **강인공지능과 약인공지능**

  - 인공지능의 분류
    - 약인공지능
      - 한분야를 집중해 인공지능을 실현
      - 음성인식, 얼굴인식 등 (음악 추천, 챗봇, 스마트폰의 생체인식)
    - 강 인공지능
      - 인간이 하는 행동 수준 또는 그 이상의 행동
      - 터미네이터 스카이넷, 프로메테우스 데이빗 등

first coding

인공신경망

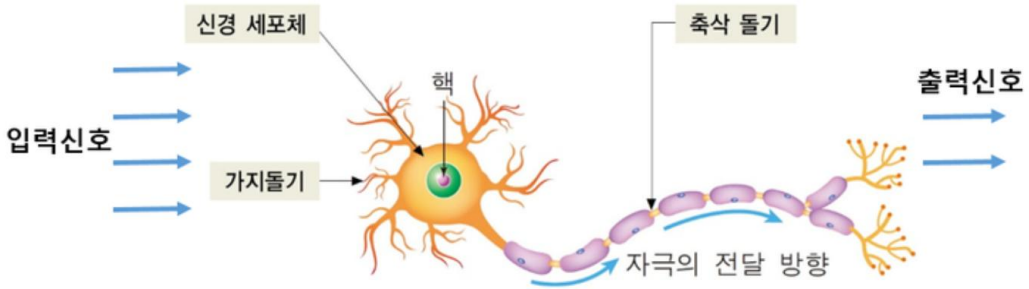
인공 신경망(Artificial Neural Network, ANN)

- **인공신경망**

  - 인공신경망(Artificial Neural Network, ANN)
    - 컴퓨터가 사람의 뇌를 본떠서 만든 알고리즘
    - 뇌에는 뉴런(Neuron)이라는 작은 세포들이 서로 연결되어 정보를 주고받으며 생각하고 결정
    - 인공신경망도 이와 비슷하게 여러 노드(Node, 가상의 뉴런)가 연결되어 정보를 처리

인공 신경망(Artificial Neural Network, ANN)

- 인공신경망 (Artificial Neural Network)



  - 뉴런의 구조
    - 신경세포체(Cell Body 또는 Soma)
      - 뉴런의 중심 부분으로, 핵과 다양한 세포 소기관이 포함
      - 세포의 생명 유지와 대사 활동을 담당하며, 단백질 합성 등 중요한 기능을 수행
    - 가지돌기(Dendrite)
      - 신경세포체에서 나뭇가지처럼 뻗어 나온 짧은 돌기들로, 다른 뉴런으로부터 신호를 받아들이는 역할
      - 여러 개의 가지돌기를 통해 동시에 다양한 신호를 수신
    - 축삭돌기(Axon)
      - 신경세포체에서 길게 뻗어 나온 하나의 돌기로, 전기 신호를 먼 거리까지 전달
      - 축삭돌기의 끝부분은 여러 갈래로 나뉘어 신경 말단(Axon Terminal)을 형성
      - 여기서 신경전달물질을 분비하여 다른 뉴런이나 근육 세포와 정보를 주고받음

인공 신경망(Artificial Neural Network, ANN)

- 인공신경망

  - 퍼셉트론(Perceptron)
    - 인공 신경망의 가장 기본적인 구성 요소
    - 여러 입력을 받아 하나의 출력을 생성하는 단순한 모델
    - 1957년 프랭크 로젠블랫(Frank Rosenblatt)에 의해 개발

인공 신경망(Artificial Neural Network, ANN)

- **인공신경망**
  - 퍼셉트론의 구조
    - 입력(Input)
      - 퍼셉트론은 여러 개의 입력 값을 받아들임 (예를 들어, $x1$,$x2$,$x3$ 같은 데이터가 들어옴)
      - 입력은 우리가 처리하고 싶은 정보(예: 이미지의 픽셀 값, 숫자 데이터 등)를 의미
    - 가중치(Weights)
      - 각 입력에는 가중치라는 숫자가 곱해짐 (예를 들어, $w1$,$w2$,$w3$ 는 입력 $x1$,$x2$,$x3$에 곱해지는 값)
      - 가중치는 입력 값이 얼마나 중요한지를 나타내는 역할을 함
    - 편향(Bias)
      - 편향은 계산에 더해지는 추가 값 → 퍼셉트론이 더 유연하게 작동하도록 함
        예를 들어, 출력 값이 항상 0이 되는 것을 막아주는 역할

인공 신경망(Artificial Neural Network, ANN)

- 인공신경망
  - 퍼셉트론의 구조
    - 합산기(Sum)
      - 각 입력에 가중치를 곱한 후 모두 더하고, 편향 값을 추가

        $$S=(x1\times w1)+(x2\times w2)+(x3\times w3)+b$$

        S 값은 퍼셉트론이 다음 단계로 넘어가도록 만드는 기준
    - 활성화 함수(Activation Function)
      - 합산된 값 S가 특정 임계값을 넘는지 확인
        - » S>임계값 : 출력은 1 S≤임계값 : 출력은 0
      - 이 과정은 스위치를 켜고 끄는 것처럼 동작
    - 출력(Output) : 최종적으로 0 또는 1의 값을 출력
      - 퍼셉트론이 "이 데이터는 어떤 분류에 속한다"라고 판단한 결과

인공 신경망(Artificial Neural Network, ANN)

- 인공신경망
  - 퍼셉트론의 동작 원리
    - 입력 신호 수집
      - 입력층에서 여러 개의 입력 신호를 받음
    - 가중합 계산
      - 각 입력 신호에 해당 가중치를 곱한 후, 모든 값을 합산하고 편향을 더하여 총합 S를 계산

$$S=(x1\times w1)+(x2\times w2)+\cdots+(xd\times wd)+b$$

    - 활성화 함수 적용
      - 계산된 총합 S를 활성화 함수에 입력하여 출력 값을 결정
    - 퍼셉트론에서는 총합이 임계값을 넘으면 1, 그렇지 않으면 0을 출력

$$y=\begin{cases}1 & \text{if } S>\text{임계값}\\ 0 & \text{if } S\leq\text{임계값}\end{cases}$$

인공 신경망(Artificial Neural Network, ANN)

- **인공신경망**
  - 퍼셉트론의 학습
    - 퍼셉트론은 주어진 데이터에 대해 가중치와 편향을 조정하여 원하는 출력이 나오도록 학습
    - 학습 과정
      1. 초기화

         가중치와 편향을 임의의 값으로 설정
      2. 예측 및 오차 계산

         입력 데이터를 통해 예측 값을 계산하고, 실제 값과의 차이(오차)를 구함
      3. 가중치 및 편향 업데이트

         오차를 기반으로 가중치와 편향을 조정하여 모델이 점차 정확한 예측을 할 수 있도록 함
      4. 반복

         오차가 최소화될 때까지 또는 정해진 횟수만큼 위 과정을 반복

인공 신경망(Artificial Neural Network, ANN)

- 인공신경망
  - 퍼셉트론의 학습 : 주어진 조건(입력값)을 기반으로 특정 사람이 커피를 좋아하는지(결과값)를 판단
    - 입력값 (Input) : $x1, x2, x3$: 각각 쓴맛, 카페인, 향에 대한 선호도
    - 가중치 (Weights) : 각각의 입력값이 얼마나 중요한지 결정하는 값
      - $w1=0.5$: 쓴맛의 중요도
      - $w2=0.7$: 카페인의 중요도
      - $w3=0.6$: 향의 중요도
    - 편향 (Bias) : 기본값으로 $b=2$를 설정
    - 활성화 함수 (Activation Function) :
      입력값의 가중합이 임계값(Threshold) 6.0을 넘으면 커피를 좋아한다고 판단
      - 합산값 $S > 6.0$: 커피를 좋아한다 → 출력 1
      - 합산값 $S \leq 6.0$: 커피를 좋아하지 않는다 → 출력 0

인공 신경망(Artificial Neural Network, ANN)

- **인공신경망**
  - 퍼셉트론의 학습 : 주어진 조건(입력값)을 기반으로 특정 사람이 커피를 좋아하는지(결과값)를 판단
    - 입력값
      - $x1=2$ (쓴맛을 별로 안 좋아함) ,(8)
      - $x2=3$ (카페인을 별로 안 좋아함) ,(7)
      - $x3=1$ (향을 별로 안 좋아함) ,(6)
    - 계산 과정
      - 가중합 계산 $S=(x1\times w1)+(x2\times w2)+(x3\times w3)+b$

        $S=(2\times 0.5)+(3\times 0.7)+(1\times 0.6)+2.0$

        $S=1.0+2.1+0.6+2.0=5.7$
      - 활성화 함수 판단

        $S=5.7 \leq 6.0$, 따라서 출력은 1 → "커피를 좋아하지 않는다."

인공 신경망(Artificial Neural Network, ANN)

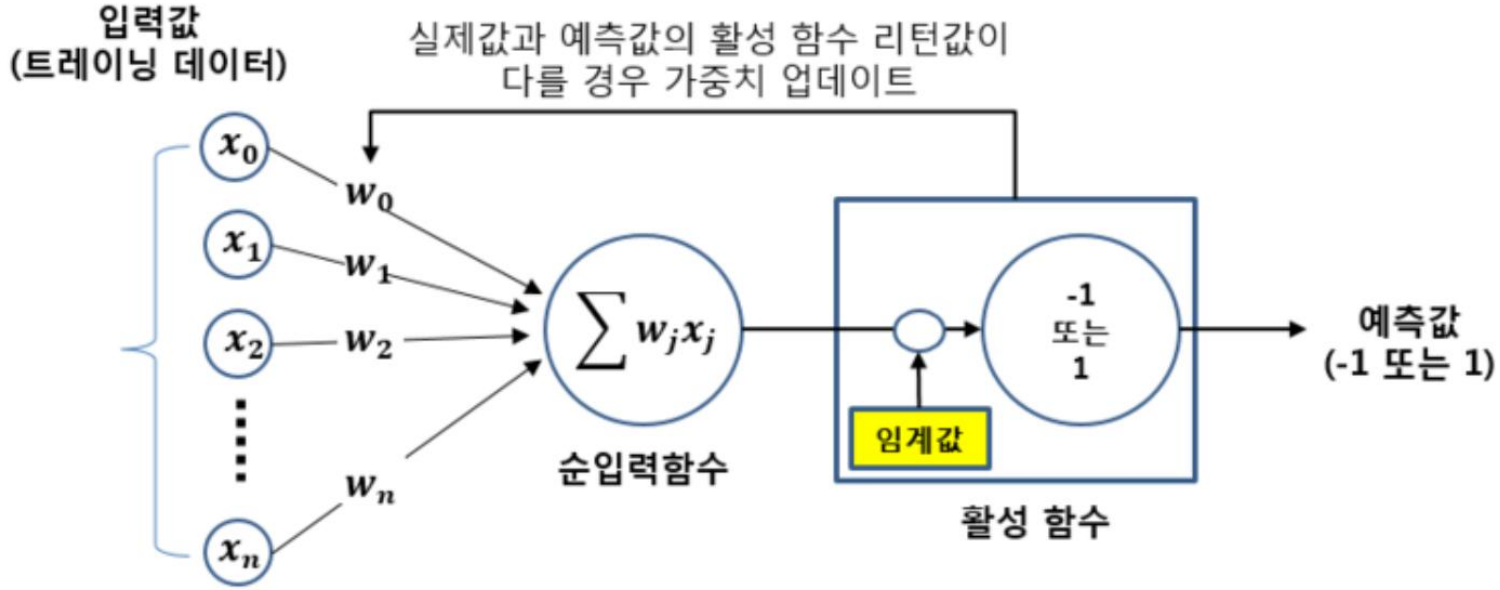
- 인공신경망

  - 퍼셉트론의 한계
    - 퍼셉트론은 간단한 "선형 분리" 문제(데이터를 직선으로 나눌 수 있는 문제)는 잘 해결
    - "XOR 문제"처럼 복잡한 문제를 해결하지 못함
    - 이를 극복하기 위해 다층 퍼셉트론(MLP)과 같은 더 복잡한 구조가 개발

      → 여러 개의 은닉층을 통해 복잡한 패턴을 학습

인공 신경망(Artificial Neural Network, ANN)

- 인공신경망
  - 퍼셉트론의 구조



인공 신경망(Artificial Neural Network, ANN)

- **인공 신경망(Artificial Neural Network, ANN)의 구성 요소**

1) 뉴런(Neuron)
   - 인공 신경망의 기본 단위로, 생물학적 뉴런을 모방한 구조
   - 각 뉴런은 입력 신호를 받아 계산을 수행하고, 결과를 다음 층으로 전달
   - 뉴런의 동작
     - 입력 신호와 가중치를 곱한 값을 합산하고, 편향(bias)을 더함
     - 결과를 활성화 함수(Activation Function)에 통과시켜 최종 출력값을 생성

2) 가중치(Weight)
   - 뉴런 간 연결의 강도를 나타내는 값
   - 각 입력 데이터(feature)는 고유한 가중치를 가지며, 학습 과정에서 이 값이 조정됨
   - 가중치는 신경망이 학습하여 패턴을 인식하는 데 중요한 역할을 함

인공 신경망(Artificial Neural Network, ANN)

- **인공 신경망(Artificial Neural Network, ANN)의 구성 요소**

3) 편향(Bias)

- 가중치와 입력값의 합에 더해지는 상수 값으로, 뉴런의 출력값을 조정하는 역할
- 모델이 특정한 방향으로 데이터를 더 잘 학습하도록 지원

4) 층(Layer)

- 신경망은 여러 층으로 구성되며, 각각의 층은 뉴런의 집합
  - **입력층(Input Layer)** : 데이터를 입력 받는 층
  - **은닉층(Hidden Layer)** : 입력 데이터를 처리하고 학습하는 층
  - **출력층(Output Layer)** : 최종 결과를 출력하는 층.

인공 신경망(Artificial Neural Network, ANN)

- 인공 신경망(Artificial Neural Network, ANN)의 구성 요소

5) 활성화 함수(Activation Function)

- 뉴런의 출력을 비선형(non-linear)으로 변환하여 복잡한 패턴을 학습할 수 있게 함
- 대표적인 활성화 함수
  - **ReLU**: $\max(0, x)$
  - **Sigmoid**: $\frac{1}{1+e^{-x}}$
  - **Tanh**: $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
  - **Softmax**: 출력값을 확률로 변환 (다중 클래스 분류에 사용).

인공 신경망(Artificial Neural Network, ANN)

- **인공 신경망(Artificial Neural Network, ANN)의 구조**

  – 신경망은 입력층, 은닉층, 출력층의 계층적 구조

1) 입력층(Input Layer)

   - 데이터를 입력받는 층으로, 입력 데이터의 각 특징(feature)은 하나의 뉴런에 대응

     예 : 이미지 데이터 (28x28 픽셀) → 입력층 뉴런 수: 784개.

인공 신경망(Artificial Neural Network, ANN)

- 인공 신경망(Artificial Neural Network, ANN)의 구조

2) 은닉층(Hidden Layer)
   - 신경망의 핵심 부분으로, 데이터의 패턴을 학습
   - 각 은닉층의 뉴런은 이전 층의 모든 뉴런과 연결되며, 이를 완전 연결(Fully Connected)이라고 함
   - 은닉층의 개수와 뉴런 수는 문제의 복잡도에 따라 결정
   - 다층 구조의 은닉층을 가진 신경망을 심층 신경망(Deep Neural Network, DNN)이라고 함

3) 출력층(Output Layer)
   - 최종 결과를 제공하는 층으로, 문제 유형에 따라 뉴런 수와 활성화 함수가 달라짐
     - **회귀 문제 : 출력 뉴런 1개 (선형 활성화 함수)**
     - **이진 분류 문제 : 출 력 뉴런 1개 (Sigmoid 활성화 함수).**
     - **다중 분류 문제 : 출력 뉴런 = 클래스 수 (Softmax 활성화 함수).**

인공 신경망(Artificial Neural Network, ANN)

- **신경망의 동작 원리**

  – 신경망은 입력 데이터를 받아 **순전파(Forward Propagation)**와 **역전파(Backpropagation)** 과정을 통해 학습

1) 순전파 (Forward Propagation)

- 데이터를 입력층에서 시작해 은닉층을 거쳐 출력층까지 전달하며 계산을 수행하는 과정
- 각 뉴런은 다음 과정을 거치며 작업
  ① 입력값과 가중치를 곱하고 합산
  ② 편향 값을 더함
  ③ 활성화 함수로 변환
  ④ 결과를 다음 층으로 전달.

인공 신경망(Artificial Neural Network, ANN)

- **신경망의 동작 원리**

2) 손실 함수 (Loss Function)

- 출력층에서 예측값과 실제값의 차이를 계산하는 함수입니다
- 대표적인 손실 함수
  - MSE (Mean Squared Error): 회귀 문제
  - Cross-Entropy Loss: 분류 문제

3) 역전파 (Backpropagation)

- 학습 과정에서 손실을 최소화하기 위해 가중치와 편향 값을 조정하는 과정
- 경사하강법(Gradient Descent) 알고리즘을 사용하여 가중치를 업데이트

first
coding

딥러닝의 구조

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- 딥러닝 모델의 기본 구성

  - 기본적으로 입력층(Input Layer), 은닉층(Hidden Layer), 출력층(Output Layer)으로 구성
    - 이 구성 요소들은 인공 신경망(Artificial Neural Network)의 구조를 이루는 핵심
    - 각각의 층은 데이터를 처리하고 학습을 진행하는 역할을 함

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- **딥러닝 모델의 기본 구성**
  - 입력층 (Input Layer)
    - 역할
      - 입력층은 데이터가 모델로 들어가는 첫 번째 단계
      - 모델이 학습해야 할 데이터를 받아들이는 역할
      - 입력 데이터의 각 특징(feature)이 입력층의 뉴런(Neuron) 하나에 연결
    - 특징
      - 입력층의 뉴런 수는 데이터의 특징(feature) 개수와 동일
        - » 예: 이미지 데이터 (28x28 픽셀의 흑백 이미지)
          입력 데이터는 총 784개의 픽셀 값(특징) → 입력층의 뉴런 수는 784개
        - » 예: CSV 데이터 (5개의 열) → 입력층의 뉴런 수는 5개
      - 입력층은 계산하지 않고 데이터를 단순히 전달

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- **딥러닝 모델의 기본 구성**
  - 은닉층 (Hidden Layer)
    - 역할
      - 은닉층은 데이터의 패턴을 학습하는 역할을 함. 신경망의 연산과 학습은 대부분 은닉층에서 이루어짐
      - 입력 데이터를 처리하여 더 높은 수준의 추상적 정보를 추출
    - 특징
      - 은닉층의 각 뉴런은 입력 뉴런(또는 이전 층의 뉴런)과 연결되어 있으며, 가중치(Weight)와 편향(Bias) 값을 사용하여 데이터를 변환
      - 각 연결에는 가중치가 부여되고, 뉴런은 가중치와 입력값을 곱한 후 이를 더한 결과를 활성화 함수(Activation Function)에 전달

$$z = \sum(\text{입력값} \times \text{가중치}) + \text{편향}$$

$$\text{출력} = \text{활성화함수}(z)$$

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- 딥러닝 모델의 기본 구성

  - 은닉층 (Hidden Layer)
    - 활성화 함수(Activation Function)
      - 활성화 함수는 은닉층의 출력을 비선형(non-linear)으로 변환하여 복잡한 패턴을 학습할 수 있게 함
      - 대표적인 활성화 함수
        - **ReLU**: $\max(0, x)$
        - **Sigmoid**: $\frac{1}{1+e^{-x}}$
        - **Tanh**: $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
        - **Softmax**: 출력값을 확률로 변환 (다중 클래스 분류에 사용).

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- **딥러닝 모델의 기본 구성**

  - 은닉층 (Hidden Layer)
    - 은닉층의 개수와 뉴런 수
      - 층 수와 뉴런 수는 모델의 복잡도에 영향을 미침
        - » 많은 층 → 더 복잡한 패턴 학습 가능 (하지만 과적합 위험)
        - » 뉴런 수가 적으면 학습이 불충분할 수 있음
      - 일반적으로 하이퍼파라미터 튜닝을 통해 최적의 층 수와 뉴런 수를 결정

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- 딥러닝 모델의 기본 구성

  - 출력층 (Output Layer)
    - 출력층은 최종 결과를 제공하는 역할을 하고 은닉층에서 학습된 정보를 바탕으로 예측값을 출력
    - 출력층의 뉴런 수는 모델이 해결하려는 문제의 종류에 따라 다름
      - **회귀 문제 (Regression)**
        - » 출력값은 연속적인 숫자 (예: 집값 예측). → 출력층 뉴런 수: 1개.
      - **이진 분류 문제 (Binary Classification)**
        - » 출력값은 0 또는 1 (예: 이메일 스팸 여부). → 출력층 뉴런 수: 1개 (Sigmoid 함수 사용).
      - **다중 분류 문제 (Multiclass Classification)**
        - » 출력값은 여러 클래스 중 하나 (예: 고양이/개/새 분류).
        - → 출력층 뉴런 수: 클래스 개수 (Softmax 함수 사용).

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- 딥러닝 모델의 기본 동작 과정

① **데이터 입력**

입력층에서 데이터가 전달

② **패턴 학습**

은닉층에서 활성화 함수를 사용해 입력 데이터의 패턴을 학습

③ **결과 출력**

출력층에서 학습된 정보를 바탕으로 예측값을 계산

④ **손실 계산**

예측값과 실제값 간의 차이를 손실 함수(Loss Function)를 사용해 계산

⑤ **가중치 업데이트**

역전파(Backpropagation)와 경사하강법(Gradient Descent)을 사용하여 가중치와 편향을 조정

딥러닝 모델의 기본 구성 요소 (입력층, 은닉층, 출력층)

- 딥러닝 모델의 기본 구성

구성 요소	역할	뉴런 수 결정 기준
입력층	데이터를 받아 모델로 전달	입력 데이터의 특징(feature) 개수
은닉층	패턴을 학습하고 데이터를 가공	모델의 복잡도 및 학습 목표에 따라 조정
출력층	최종 결과를 예측하여 출력	문제의 유형 (회귀, 이진 분류, 다중 분류)

first coding

딥러닝 기본 수학

활성화 함수(Activation Functions): ReLU, Sigmoid, Softmax

손실 함수(Loss Functions): MSE, Cross-Entropy

역전파(Backpropagation) 개념

활성화 함수(Activation Functions): ReLU, Sigmoid, Softmax

- **활성화 함수**
  - 활성화 함수는 인공 신경망에서 입력 신호를 처리하고 출력 신호를 결정하는 역할

    → 입력 값을 기반으로 출력 값을 비선형적으로 변환하여 신경망이 복잡한 패턴과 관계를 학습
  - 활성화 함수의 역할
    - **비선형성 추가** : 활성화 함수는 신경망에 비선형성을 도입
      - 신경망의 각 층에서 입력과 가중치의 선형 조합만 계산하면 전체 네트워크는 선형 함수가 되어, 복잡한 문제를 해결하지 못함
      - 활성화 함수는 비선형 변환을 통해 신경망이 복잡한 패턴과 관계를 학습할 수 있음
    - **출력 범위 제한**
      - 활성화 함수는 출력 값을 특정 범위로 제한하여 학습 안정성을 높이고,
      - 극단적인 값이 네트워크 전체에 영향을 주는 것을 방지
    - **특정 특징 강조 :** 활성화 함수는 특정 입력 신호를 강화하거나 억제하여 중요한 특징을 강조

활성화 함수(Activation Functions): ReLU, Sigmoid, Softmax

- **활성화 함수**
  - ReLU (Rectified Linear Unit)
    - 수식 : $f(x)=\max(0,x)$
    - 특징
      - 입력 값이 양수일 때 그대로 출력하고, 음수일 때는 0으로 출력
      - 계산이 간단하고, 큰 네트워크에서도 잘 작동
      - **대부분의 은닉층에서 기본적으로 사용**
    - 장점
      - 계산이 빠르고 학습 속도가 빠름
      - 기울기 소실(Vanishing Gradient) 문제를 줄여 줌
    - 단점
      - 입력 값이 항상 0 이하로 유지되면 뉴런이 비활성화되어 학습하지 못하는 문제가 생김

활성화 함수(Activation Functions): ReLU, Sigmoid, Softmax

- **활성화 함수**
  - Sigmoid
    - 수식 : $f(x) = \frac{1}{1+e^{-x}}$
    - 특징
      - 입력 값을 0과 1 사이로 변환 → 출력 값이 확률처럼 보이기 때문에 **이진 분류 문제**에서 자주 사용
      - 출력층에서 확률 값이 필요한 이진 분류 문제에 사용
    - 장점
      - 출력 값이 제한되어 있어 안정적임
    - 단점
      - 기울기 소실 문제가 발생할 수 있음
      - 입력 값이 너무 크거나 작으면 출력의 변화가 거의 없어 학습이 어려워 짐
      - 계산이 상대적으로 느림

활성화 함수(Activation Functions): ReLU, Sigmoid, Softmax

- **활성화 함수**
  - Softmax
    - 수식 : $$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$ 여기서 xi 는 출력의 각 원소이고, n은 클래스의 개수
    - 특징
      - 출력 값을 확률로 변환하여, 모든 출력 값의 합이 1이 되도록 만듦
      - **다중 클래스 분류 문제**에서 사용
    - 장점
      - 출력 값이 확률 분포를 가지므로 직관적
    - 단점
      - 계산이 비교적 복잡하며, 클래스 수가 많아지면 느려질 수 있음

손실 함수(Loss Functions)

- 손실 함수(Loss Functions)

  - 손실 함수는 모델이 예측한 값과 실제 값 간의 차이를 측정하는 함수
  - 신경망이 학습할 때 손실 값을 최소화하는 방향으로 가중치가 조정 됨

    → 즉, 손실 함수는 학습의 방향과 성능을 평가하는 기준임

  - 손실함수의 기능
    - **성능 평가** : 모델이 얼마나 잘 작동하고 있는지 알려줌
    - **학습 방향 제공** : 손실 값을 기준으로 경사 하강법(Gradient Descent)이 실행되어 모델의 가중치를 조정

      Adam (효율적인 경사 하강법 알고리즘)
    - **문제 유형에 따라 손실 함수 선택** : 회귀와 분류 문제에 적합한 손실 함수가 다름

손실 함수(Loss Functions)

- 손실 함수(Loss Functions)

  - MSE (Mean Squared Error)
    - MSE는 예측 값과 실제 값의 차이를 제곱하여 평균을 구한 값
      - yi : 실제 값
      - $y\hat{}i$: 모델의 예측 값
      - n : 데이터의 개수

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

    - 간단하고 계산이 쉽고 예측 오류의 크기에 민감해 큰 오류에 더 큰 패널티를 줌
    - 오류가 큰 데이터에 민감합니다(이상치의 영향)

손실 함수(Loss Functions)

- 손실 함수(Loss Functions)

  - Cross-Entropy Loss

    - Cross-Entropy는 분류 문제에서 모델이 출력한 확률 분포와 실제 레이블의 차이를 측정

      - yi : 실제 레이블 (0 또는 1)
      - $y\hat{}i$: 모델이 예측한 확률 값
      - n : 데이터의 개수

$$Loss = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

    - 확률 값(0~1)을 기반으로 계산하므로 분류 문제에 적합
    - 모델이 잘못된 확률을 출력했을 때 큰 패널티를 줌
    - 예측 값이 0에 가까울 때(예 : $y\hat{}$=0.0001) 손실 값이 매우 커질 수 있음

손실 함수(Loss Functions)

- 손실 함수(Loss Functions)

  - MSE 와 Cross-Entropy Loss 비교

특성	MSE	Cross-Entropy
사용되는 문제	회귀 문제	분류 문제
출력 값	실수 값 (예: 100, 0.5 등)	확률 값 (0~1 사이)
오류 처리 방식	큰 오류에 민감 (제곱 계산)	잘못된 확률에 큰 패널티
장점	단순하고 계산이 빠름	분류 문제에서 성능이 뛰어남
단점	이상치에 민감	계산이 복잡하고 확률 값에 의존

역전파(Backpropagation)

- 역전파(Backpropagation)

  – 역전파는 신경망의 학습 과정에서 오류를 기반으로 가중치를 조정하는 알고리즘

  → 이 알고리즘은 신경망이 입력 데이터를 출력으로 변환하는 과정에서 발생한 오류(Error) 를
  각 층으로 역방향으로 전파(Backward Propagation)하여,
  각 가중치를 조정하는 데 사용

역전파(Backpropagation)

- 역전파(Backpropagation)

  - 역전파 과정의 세부 단계

    ① **순방향 전달 (Forward Propagation)**

    - 입력 데이터를 각 층에 전달하여 출력을 계산
    - 각 뉴런에서 활성화 함수를 적용하여 비선형성을 추가
    - 마지막 층에서 출력 값을 생성

    ② **손실 계산**

    - 손실 함수(Loss Function)를 사용하여 예측 값과 실제 값 간의 손실을 계산
      예를 들어, MSE를 사용하여 계산

역전파(Backpropagation)

- 역전파(Backpropagation)

  - 역전파 과정의 세부 단계

    ③ **오류 역전파 (Backward Propagation)**

    - 손실 함수의 출력 값을 기준으로, 출력층에서 시작해 입력층으로 이동하며
      각 가중치에 대한 기울기를 계산
    - 이 과정은 체인 룰(Chain Rule)을 사용하여 이루어짐
      - » 체인 룰 : **복잡한 함수의 미분을 계산할 때, 각 단계별 미분 값을 곱하는 방법**
        예를 들어, 손실 함수 L이 활성화 함수 f와 가중치 w에 의해 결정된다면

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial w}$$

    - 각 뉴런의 출력에 대한 손실의 변화량(기울기)을 계산하고, 이를 다음 층으로 전달

- 역전파(Backpropagation)

  - 역전파 과정의 세부 단계

    ④ **가중치 업데이트**

      - 경사 하강법(Gradient Descent)을 사용하여 가중치를 조정

$$w_{\text{new}} = w_{\text{old}} - \eta \cdot \frac{\partial L}{\partial w}$$

- w: 가중치
- η: 학습률(Learning Rate)
- $\frac{\partial L}{\partial w}$: 손실 함수의 가중치에 대한 기울기

first coding

딥러닝 프레임워크

딥러닝 프레임워크(Keras, TensorFlow)의 이해

간단한 신경망 모델 구성 및 훈련

딥러닝 프레임워크(Keras, TensorFlow)의 이해

- 딥러닝 프레임워크

  - 딥러닝 모델을 쉽고 효율적으로 개발할 수 있도록 도와주는 도구
    - 복잡한 수학적 연산이나 하드웨어 자원의 효율적인 사용을 프레임워크가 처리해 주기 때문에, 연구자나 개발자는 모델 설계와 데이터 분석에 집중

  - 가장 널리 사용되는 프레임워크 중 하나가 TensorFlow
  - Keras는 TensorFlow의 고수준 API로 제공

딥러닝 프레임워크(Keras, TensorFlow)의 이해

- 딥러닝 프레임워크

  - TensorFlow
    - 구글에서 개발한 오픈소스 딥러닝 라이브러리로, 다양한 플랫폼에서 실행 가능
    - 복잡한 수학적 연산(예: 행렬 연산, 미분 계산 등)을 자동으로 처리
    - 딥러닝뿐만 아니라 머신러닝, 강화학습에도 활용될 수 있음
    - 특징
      - **유연성** : 저수준의 연산도 제어 가능. 필요에 따라 매우 복잡한 모델 설계 가능.
      - **멀티플랫폼 지원** : CPU, GPU, TPU에서 실행 가능
      - **확장성** : 대규모 분산 학습을 지원.

딥러닝 프레임워크(Keras, TensorFlow)의 이해

- 딥러닝 프레임워크

  - Keras
    - Keras는 TensorFlow 위에서 작동하는 고수준의 딥러닝 API
    - 초보자도 쉽게 사용할 수 있도록 설계되었음
    - 복잡한 코드 없이 직관적으로 모델을 설계하고 학습시킬 수 있음
    - 다양한 딥러닝 구조(CNN, RNN 등)를 지원하며, 필요에 따라 커스터마이징
    - 특징
      - **간결함** : 코드가 간단하고 직관적
      - **빠른 프로토타이핑** : 모델을 빠르게 설계하고 실험 가능
      - **유연성** : 다양한 백엔드 엔진(TensorFlow, Theano, CNTK) 지원

딥러닝 프레임워크(Keras, TensorFlow)의 이해

- 딥러닝 프레임워크

  - TensorFlow와 Keras의 관계
    - TensorFlow는 엔진, Keras는 운전대처럼 생각할 수 있음
    - Keras를 사용하면 TensorFlow의 복잡한 저수준 작업을 몰라도 쉽게 모델을 설계하고 훈련
    - 최신 TensorFlow(2.0 이상)에서는 Keras가 기본 내장되어 있어 설치와 사용이 더 간단함

간단한 신경망 모델 구성 및 훈련

- 간단한 신경망 모델을 구성하고 훈련하는 예제

1. 필요한 라이브러리 로드

```
# 1. 필요한 라이브러리 임포트
# TensorFlow와 Keras의 기능을 사용하기 위해 필요한 모듈을 불러옵니다.
import numpy as np
from tensorflow.keras.models import Sequential       # 순차적 모델을 생성하기 위한 모듈
from tensorflow.keras.layers import Dense            # 밀집층(fully connected layer)을 추가하기 위한 모듈
from sklearn.model_selection import train_test_split # 데이터를 학습/테스트 세트로 나누기 위한 모듈
from sklearn.datasets import make_classification     # 예제 데이터셋 생성 모듈
```

간단한 신경망 모델 구성 및 훈련

- 간단한 신경망 모델을 구성하고 훈련하는 예제

2. 데이터 생성 및 전처리

```
# 2. 데이터 생성 및 전처리
# 예제용으로 가상 데이터셋을 생성합니다.
X, y = make_classification(
    n_samples=1000,   # 데이터 샘플 수
    n_features=20,    # 특징(특성) 수
    n_classes=2,      # 클래스 수 (이진 분류)
    random_state=42   # 랜덤 시드 고정 (결과 재현 가능성 보장)
)

# 데이터를 학습용(train)과 테스트용(test)으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

간단한 신경망 모델 구성 및 훈련

- 간단한 신경망 모델을 구성하고 훈련하는 예제

3. 모델 생성

```
# 3. 모델 생성
# Sequential()은 모델을 레이어 순서대로 구성할 수 있도록 해줍니다.
model = Sequential([
    Dense(16, activation='relu', input_shape=(X_train.shape[1],)),  # 첫 번째 은닉층
    Dense(8, activation='relu'),                                    # 두 번째 은닉층
    Dense(1, activation='sigmoid')                                  # 출력층 (이진 분류에서 사용)
])
# 여기서 `input_shape`는 입력 데이터의 차원을 지정합니다.
# `activation`은 각 층에서 사용할 활성화 함수입니다.
# - relu: 0보다 큰 값은 그대로 두고, 작은 값은 0으로 설정 (비선형성 제공)
# - sigmoid: 출력값을 0과 1 사이로 제한 (이진 분류에서 확률 계산에 유용)
```

간단한 신경망 모델 구성 및 훈련

- 간단한 신경망 모델을 구성하고 훈련하는 예제

4. 모델 컴파일

```
# 4. 모델 컴파일
# 모델 학습 전에 학습 방법(optimizer), 손실 함수(loss), 평가 지표(metrics)를 정의합니다.
model.compile(
    optimizer='adam',              # 학습 속도를 조정하며 손실 함수의 최솟값을 찾습니다.
    loss='binary_crossentropy',    # 이진 분류에서 자주 사용되는 손실 함수
    metrics=['accuracy']           # 학습 성과를 평가할 지표
)
```

간단한 신경망 모델 구성 및 훈련

- 간단한 신경망 모델을 구성하고 훈련하는 예제

5. 모델 훈련

```
# 5. 모델 훈련
# fit() 메서드는 모델을 학습시키는 역할을 합니다.
history = model.fit(
    X_train, y_train,              # 학습용 데이터와 레이블
    validation_split=0.2,          # 검증 데이터 비율 (학습 데이터의 20%)
    epochs=10,                     # 학습 반복 횟수
    batch_size=32,                 # 한 번의 학습에서 사용하는 데이터 샘플 수
    verbose=1                      # 학습 진행 상태를 출력
)
```

간단한 신경망 모델 구성 및 훈련

- 간단한 신경망 모델을 구성하고 훈련하는 예제

6. 모델 평가 및 예측

```
# 6. 모델 평가
# 테스트 데이터로 학습된 모델을 평가합니다.
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"테스트 손실: {test_loss:.4f}, 테스트 정확도: {test_accuracy:.4f}")

# 7. 모델 예측
# 학습된 모델을 사용해 새로운 데이터 예측
predictions = model.predict(X_test[:5])  # 테스트 데이터 중 5개의 샘플 예측
print("예측 결과:", predictions)

# 여기서 predictions의 출력은 확률 값입니다.
# 일반적으로 0.5 이상은 클래스 1, 미만은 클래스 0으로 간주합니다.
```

간단한 신경망 모델 구성 및 훈련

- **간단한 신경망 모델을 구성하고 훈련하는 예제**

① 모델의 구성

- 모델은 입력층 → 은닉층 → 출력층으로 구성
- 활성화 함수는 모델이 데이터를 비선형적으로 표현

② 학습 흐름

- 데이터 준비 → 모델 설계 → 모델 컴파일 → 훈련 → 평가 → 예측

③ Keras의 장점

- 코드가 직관적이고 간단하여 초보자도 쉽게 딥러닝 모델을 설계하고 학습시킬 수 있음

first coding

딥러닝 모델 실습 - 이진 분류

이진 분류 문제 해결

데이터셋 준비 및 모델 훈련

이진 분류 문제 해결

- **이진 분류**

  – 이진 분류(Binary Classification)는 주어진 데이터를 두 개의 클래스로 구분하는 문제를 해결하는 과정
  예를 들어
  이메일 스팸 필터링(스팸 또는 정상), 의료 진단(양성 또는 음성), 고객 행동 예측(구매 또는 비구매) 등

1. 문제 정의
   - 주어진 입력 데이터(특성, Features)를 기반으로 해당 데이터가 두 개의 클래스 중 어디에 속하는지 예측
   - 출력 → 클래스 0 또는 클래스 1의 확률.

이진 분류 문제 해결

- **이진 분류**

2. 데이터 준비
    - 데이터 수집
        - 문제에 맞는 데이터셋을 확보. 예시: 의료 데이터, 고객 행동 데이터, 텍스트 데이터 등
    - 데이터 전처리
        - 특성 선택 및 정규화
            - » 숫자 데이터를 정규화 하거나 범주형 데이터를 인코딩(Label Encoding, One-Hot Encoding)
        - 결측값 처리
            - » 결측값을 제거하거나 보완(Imputation).
        - 데이터 분할
            - » 데이터를 훈련(Train), 검증(Validation), 테스트(Test) 데이터셋으로 나눔
            - » 일반적인 비율: 70%(Train) - 15%(Validation) - 15%(Test).

이진 분류 문제 해결

- **이진 분류**

3. 모델 설계 : 딥러닝 기반 이진 분류 모델은 다음과 같은 구조를 가질 수 있음
    - 입력층 (Input Layer)
        - 데이터의 특성 수에 따라 노드 수 결정.
          예: 데이터가 10개의 특성을 가진다면 입력층의 노드 수는 10개
    - 은닉층 (Hidden Layers)
        - Fully Connected Layer 사용.
        - 활성화 함수(Activation Function) : **ReLU(Rectified Linear Unit)를 주로 사용**
        - **과적합 방지를 위해 Dropout Layer 추가 가능**
    - 출력층 (Output Layer)노드 수
        - 1 (이진 분류는 하나의 확률값 출력)
        - 활성화 함수 : **Sigmoid (출력값을 0~1 사이의 확률로 변환)**

이진 분류 문제 해결

- 이진 분류

4. 모델 컴파일
    - 손실 함수 : Binary Cross-Entropy
    - 최적화 함수 : Adam (또는 SGD)
    - 평가지표 : 정확도(Accuracy), F1-Score, Precision, Recall 등.

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

이진 분류 문제 해결

- 이진 분류

5. 모델 훈련
   - 훈련 데이터를 사용하여 모델을 학습
   - 동시에 검증 데이터를 사용하여 모델의 성능을 평가하고 과적합을 방지

```
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=20,
                    batch_size=32)
```

     - 검증 데이터(Validation Set)
       - » 훈련 중 모델 성능을 측정하는 데 사용
       - » 테스트 데이터와는 분리되어야 함.

이진 분류 문제 해결

- 이진 분류

5. 모델 훈련
   - 훈련 데이터를 사용하여 모델을 학습
   - 동시에 검증 데이터를 사용하여 모델의 성능을 평가하고 과적합을 방지

```
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=20,
                    batch_size=32)
```

   – 검증 데이터(Validation Set)
      » 훈련 중 모델 성능을 측정하는 데 사용
      » 테스트 데이터와는 분리되어야 함.

이진 분류 문제 해결

- 이진 분류

6. 과적합 방지

- Early Stopping
  - 훈련 중 검증 성능이 더 이상 향상되지 않으면 조기에 학습을 멈춤
  - 과적합을 방지하는 데 효과적

```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3)
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=32,
                    callbacks=[early_stopping])
```

이진 분류 문제 해결

- 이진 분류

6. 과적합 방지
    - Dropout
        - 학습 중 일부 노드를 무작위로 제외하여 과적합을 방지.

```
from tensorflow.keras.layers import Dropout
model.add(Dropout(0.5))
```

이진 분류 문제 해결

- 이진 분류

7. 모델 평가
    - 훈련 후 테스트 데이터를 사용해 모델 성능을 평가

```
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")
```

이진 분류 문제 해결

- 이진 분류

8. 예측
    - 새로운 데이터를 입력하여 예측값을 출력

```
predictions = model.predict(new_data)
predicted_classes = (predictions > 0.5).astype(int)
```

이진 분류 문제 해결

- 이진 분류

8. 예측
    - 새로운 데이터를 입력하여 예측값을 출력

```
predictions = model.predict(new_data)
predicted_classes = (predictions > 0.5).astype(int)
```

데이터셋 준비 및 모델 훈련

- **데이터셋**
  - 훈련(Training), 검증(Validation), 테스트(Test) 데이터셋
    → 머신러닝과 딥러닝 모델 학습 과정에서 중요한 역할

1. 훈련 데이터셋 (Training Dataset)
   - 정의모델을 학습시키는 데 사용되는 데이터셋
     → 입력 데이터(X)와 그에 해당하는 실제 레이블(정답, y)을 포함
   - 역할
     - 모델이 데이터의 패턴과 관계를 학습하도록 하고 모델의 가중치와 바이어스를 조정하는 데 사용
     - 학습 과정에서 손실(Loss)을 계산하고 이를 기반으로 최적화 알고리즘이 모델을 업데이트
     - 데이터의 대부분(약 70~80%)을 차지하며, 모델이 충분히 학습할 수 있도록 많은 데이터를 포함

데이터셋 준비 및 모델 훈련

- **데이터셋**

2. 검증 데이터셋 (Validation Dataset)
    - 훈련 중 모델의 성능을 평가하기 위해 사용되는 데이터셋
    - 훈련 데이터와 테스트 데이터와는 별도로 분리된 데이터
    - 역할
        - 모델이 훈련 데이터에만 치우치지 않고 새로운 데이터에도 잘 작동하는지 평가
          → 과적합(Overfitting)을 감지하는 데 사용
        - 하이퍼파라미터(예: 학습률, 은닉층 수, 드롭아웃 비율)를 튜닝할 때 활용
    - 특징
        - 훈련 중에는 검증 데이터를 사용하여 모델의 성능 변화를 실시간으로 모니터링
        - Early Stopping과 같은 기법에서 검증 데이터의 손실이 일정 수준 이상 줄어들지 않으면 학습 중단
        - 일반적으로 전체 데이터의 약 10~15%를 차지

데이터셋 준비 및 모델 훈련

- **데이터셋**

3. 테스트 데이터셋 (Test Dataset)
    - 학습이 완료된 후, 모델의 최종 성능을 평가하기 위해 사용되는 데이터셋
    - 모델이 훈련이나 검증에서 본 적 없는 새로운 데이터로 구성
    - 역할
        - 모델이 실제 환경에서 얼마나 잘 동작할지를 예측
        - 훈련 및 검증 데이터를 사용하지 않았으므로 모델의 일반화 능력을 평가하는 데 적합
        - 최종 평가 결과를 통해 모델의 신뢰성과 성능을 판단
    - 특징
        - 테스트 데이터는 모델이 학습 과정에서 한 번도 접하지 않았기 때문에,
          "새로운 데이터"에 대한 모델의 성능을 측정
          → 과적합이 심한 모델은 테스트 데이터에서 성능이 낮아질 수 있음

데이터셋 준비 및 모델 훈련

- **데이터셋**

4. 데이터 분할 비율
    - 보통 다음과 같은 비율로 데이터를 나눔
        - ✓ 훈련 데이터: 70~80%
        - ✓ 검증 데이터: 10~15%
        - ✓ 테스트 데이터: 10~15%

    - 데이터가 적은 경우 **교차 검증(Cross-Validation)**을 활용해 데이터를 효율적으로 사용할 수도 있음

데이터셋 준비 및 모델 훈련

- **데이터셋**
  - **데이터를 왜 나누는가?**
    1) 과적합(Overfitting) 방지
       - 훈련 데이터만 사용하면 모델이 해당 데이터에만 최적화
         → 새로운 데이터에 대한 예측 성능이 떨어질 가능성이 높음
       - 검증 데이터와 테스트 데이터를 통해 과적합 여부를 확인
    2) 모델 성능 평가
       - 테스트 데이터는 실제 환경에서 모델이 얼마나 잘 작동할지를 평가하기 위한 기준이 됨
    3) 데이터 유출 방지
       - 검증 또는 테스트 데이터가 훈련 데이터에 섞이면
         → 모델은 실제로 학습하지 않은 데이터를 일반화하는 것이 아니라,
         그 데이터를 "암기"할 수 있음

데이터셋 준비 및 모델 훈련

- 데이터셋
  - 실제로 데이터를 나누는 방법

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# 1. 데이터 생성
# 이진 분류를 위한 가상 데이터셋 생성
X, y = make_classification(n_samples=1000, n_features=20,
                           n_informative=15, n_redundant=5,
                           random_state=42)

# 2. 데이터 분할
# (1) 전체 데이터를 훈련 데이터와 임시 데이터로 나눔 (훈련 70%, 나머지 30%)
X_train, X_temp, y_train, y_temp =
         train_test_split(X, y, test_size=0.3, random_state=42)

# (2) 나머지 데이터를 검증 데이터와 테스트 데이터로 나눔 (각각 15%씩)
X_val, X_test, y_val, y_test =
    train_test_split( X_temp, y_temp, test_size=0.5, random_state=42)

# 3. 각 데이터셋 크기 확인
print("훈련 데이터 크기:", X_train.shape, y_train.shape)  # 약 70%의 데이터
print("검증 데이터 크기:", X_val.shape, y_val.shape)      # 약 15%의 데이터
print("테스트 데이터 크기:", X_test.shape, y_test.shape)  # 약 15%의 데이터
```

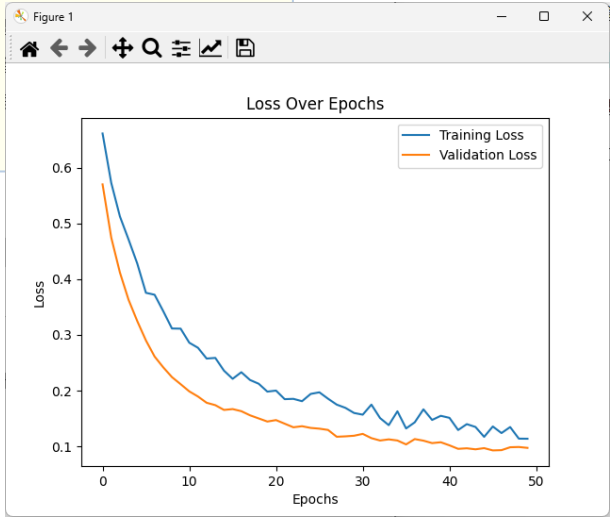
이진 분류 문제 해결

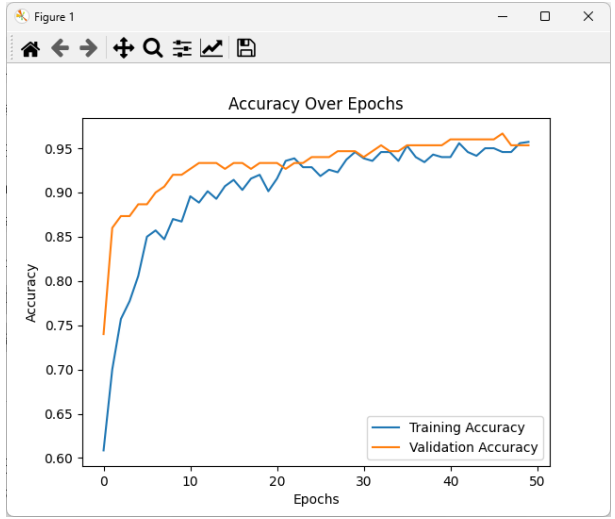
```python
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report

# 1. 데이터 준비
# 예제 데이터 생성 (여기서는 임의로 데이터를 생성합니다. 실제 데이터로 교체 가능)
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
                           n_redundant=5, random_state=42)

# 데이터 분할: 훈련, 검증, 테스트 데이터셋
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3,
random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

# 데이터 정규화 (특성 스케일링)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# 2. 모델 설계
# 입력층, 은닉층, 출력층으로 구성된 이진 분류 모델
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train.shape[1],)),  # 입력층
    tf.keras.layers.Dense(64, activation='relu'),      # 첫 번째 은닉층
    tf.keras.layers.Dropout(0.5),                      # 과적합 방지를 위한
Dropout
    tf.keras.layers.Dense(32, activation='relu'),      # 두 번째 은닉층
    tf.keras.layers.Dense(1, activation='sigmoid')     # 출력층 (Sigmoid 활
성화 함수)
])
```

```python
# 3. 모델 컴파일
# 손실 함수: Binary Cross-Entropy
# 최적화 알고리즘: Adam
# 평가지표: Accuracy
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# 4. Early Stopping 설정
# 검증 데이터 손실이 개선되지 않으면 학습을 조기에 종료
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True)

# 5. 모델 훈련
history = model.fit(X_train, y_train,
                    validation_data=(X_val, y_val),
                    epochs=50,
                    batch_size=32,
                    callbacks=[early_stopping])

# 6. 모델 평가
# 테스트 데이터로 성능 평가
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}, Test Accuracy: {test_accuracy:.4f}")

# 7. 예측
# 새로운 데이터에 대한 예측 수행
predictions = model.predict(X_test)
predicted_classes = (predictions > 0.5).astype(int)  # 0.5를 기준으로 클래
스 결정

# 8. 분류 보고서 출력
# 실제 값과 예측 값을 비교하여 성능 지표 확인
print("\nClassification Report:")
print(classification_report(y_test, predicted_classes))
```

이진 분류 문제 해결

```python
# 9. 학습 결과 시각화 (Optional)
import matplotlib.pyplot as plt

# 훈련 및 검증 손실 시각화
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# 훈련 및 검증 정확도 시각화
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





first coding

딥러닝 모델 실습 - 다중 분류

다중 분류 문제 해결

Softmax 활성화 함수와 One-Hot Encoding

다중 분류 문제 해결

- **다중 분류 문제**

  - 다중 분류 문제
    - 입력 데이터를 여러 클래스 중 하나로 분류하는 문제

    - 예시
      - 고양이, 개, 새 등 여러 동물 이미지를 특정 클래스로 분류
      - 여러 종류의 꽃을 분류하는 작업
      - 손글씨 숫자 데이터(MNIST)에서 숫자 0부터 9까지를 분류하는 문제
      - 영화 리뷰를 긍정, 중립, 부정으로 분류하는 문제.

다중 분류 문제 해결

- **다중 분류 문제를 해결하기 위한 과정**

1. 데이터 준비

① 데이터 수집 : 사용할 데이터셋을 준비 (데이터는 각 클래스에 해당하는 레이블이 있어야 함)

② 데이터 전처리

- **One-Hot Encoding**

  레이블을 숫자로 인코딩하고, 이를 원-핫 벡터로 변환

  예: 클래스 0, 1, 2가 있다면, 레이블은 [1, 0, 0], [0, 1, 0], [0, 0, 1]로 표현

- **정규화**

  입력 데이터를 정규화 하여 신경망 학습을 안정적으로 처리

  → 이미지의 픽셀 값(0~255)을 [0, 1] 범위로 스케일링.

③ 데이터 분리

- 데이터를 훈련, 검증, 테스트 세트로 나눔 → 일반적으로 70%(훈련), 15%(검증), 15%(테스트)

다중 분류 문제 해결

- **다중 분류 문제를 해결하기 위한 과정**

2. 모델 설계

① 입력층

- 데이터의 차원(예: 이미지 크기)과 동일한 입력을 받는 층.

② 은닉층

- Fully Connected Layer(완전 연결층) 또는 Convolutional Layer(이미지 문제의 경우)를 포함
- 활성화 함수로 ReLU를 사용하여 비선형성을 추가

③ 출력층

- 출력 뉴런의 수는 클래스의 수와 동일

  예를 들어, 클래스가 3개라면 출력층의 뉴런 수는 3
- **Softmax 활성화 함수를 사용**하여 각 클래스에 대한 확률을 계산

  → Softmax는 각 뉴런의 출력을 확률로 변환

다중 분류 문제 해결

- **다중 분류 문제를 해결하기 위한 과정**

3. 모델 훈련

① 손실 함수

- 다중 분류 문제에서는 Categorical Crossentropy 손실 함수를 사용
  - » $Loss = -\sum_i y_i \log(\hat{y}_i)$
  - » 여기서 yi 는 실제 값(원-핫 벡터), $y\hat{}i$ 는 모델의 예측 확률

② 옵티마이저

- Gradient Descent 기반 알고리즘(예: Adam)을 사용하여 가중치를 업데이트

③ 훈련 프로세스

- 모델에 훈련 데이터를 입력하고, 반복적으로 가중치를 업데이트하여 손실을 최소화
- 검증 데이터를 사용하여 과적합(overfitting)을 방지하고 성능을 모니터링

다중 분류 문제 해결

- **다중 분류 문제를 해결하기 위한 과정**

4. 모델 평가
    - **정확도**
        - 테스트 데이터에서 모델이 얼마나 정확하게 예측하는지 평가
    - **혼동 행렬**
        - 각 클래스에 대한 잘못된 예측과 정확한 예측을 시각화
    - **클래스별 평가**
        - 클래스별 정밀도, 재현율, F1 점수를 계산하여 모델의 성능을 평가
5. 모델 평가
    - 학습된 모델을 사용하여 새로운 데이터를 예측
    - **예측은 Softmax 출력을 기반으로 가장 높은 확률 값을 가진 클래스를 선택**

first coding

실습 프로젝트 - 손글씨 숫자 분류(MNIST)

MNIST 데이터셋을 활용한 이미지 분류

전체 딥러닝 프로세스 적용

MNIST 데이터셋을 활용한 이미지 분류

```
# 필요한 라이브러리 가져오기
from tensorflow.keras.models import Sequential  # 순차적 모델 생성
from tensorflow.keras.layers import Dense       # 완전 연결(Dense) 층
from tensorflow.keras.utils import to_categorical  # 원-핫 인코딩
from tensorflow.keras.datasets import mnist     # MNIST 데이터셋 가져오기

# 1. 데이터 로드 및 전처리
# MNIST 데이터셋을 훈련 세트와 테스트 세트로 분리
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 입력 데이터(이미지)를 1차원 벡터로 변환하고 정규화 (0~1 범위로 스케일링)
x_train = x_train.reshape(-1, 784) / 255.0  # 28x28 이미지를 784 길이의 벡터로 변환
x_test = x_test.reshape(-1, 784) / 255.0

# 레이블(출력 값)을 원-핫 인코딩 (예: 3 -> [0, 0, 0, 1, 0, 0, 0, 0, 0, 0])
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# 2. 모델 구성
model = Sequential([
    # 첫 번째 은닉층: 뉴런 128개, 활성화 함수 ReLU, 입력 크기는 784
    Dense(128, activation='relu', input_shape=(784,)),
    # 두 번째 은닉층: 뉴런 64개, 활성화 함수 ReLU
    Dense(64, activation='relu'),
    # 출력층: 뉴런 10개 (클래스 수), 활성화 함수 Softmax
    Dense(10, activation='softmax')
])
```

```
# 3. 모델 컴파일
# 손실 함수: Categorical Crossentropy
# 옵티마이저: Adam (효율적인 경사 하강법 알고리즘)
# 평가 지표: 정확도 (accuracy)
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# 4. 모델 훈련
# 훈련 데이터(x_train, y_train)로 학습, 검증 데이터로 성능 확인
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_split=0.2)

# 5. 모델 평가
# 테스트 데이터(x_test, y_test)로 모델 성능 평가
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f"테스트 정확도: {test_accuracy}")

# 6. 예측
# 테스트 데이터의 첫 번째 샘플을 예측
predictions = model.predict(x_test[:1])
print(f"Softmax 출력: {predictions[0]}")  # 각 클래스에 대한 확률 값
print(f"예측 클래스: {predictions.argmax()}")  # 확률이 가장 높은 클래스
```

MNIST 데이터셋을 활용한 이미지 분류

```
2024-12-04 21:53:19.959601: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-12-04 21:53:23.596202: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1s 0us/step
C:\Users\jin\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2024-12-04 21:53:34.450833: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8599 - loss: 0.4840 - val_accuracy: 0.9594 - val_loss: 0.1386
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9666 - loss: 0.1130 - val_accuracy: 0.9633 - val_loss: 0.1204
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9755 - loss: 0.0792 - val_accuracy: 0.9710 - val_loss: 0.0932
Epoch 4/10
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9824 - loss: 0.0580 - val_accuracy: 0.9695 - val_loss: 0.1032
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9859 - loss: 0.0433 - val_accuracy: 0.9739 - val_loss: 0.0961
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9882 - loss: 0.0332 - val_accuracy: 0.9734 - val_loss: 0.0933
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9914 - loss: 0.0262 - val_accuracy: 0.9748 - val_loss: 0.0996
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9920 - loss: 0.0230 - val_accuracy: 0.9754 - val_loss: 0.1020
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.9939 - loss: 0.0188 - val_accuracy: 0.9777 - val_loss: 0.0984
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9952 - loss: 0.0156 - val_accuracy: 0.9758 - val_loss: 0.1091
313/313 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.9707 - loss: 0.1283
테스트 정확도: 0.9757000207901001
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
Softmax 출력: [3.4918780e-13 6.0301891e-10 1.4999438e-08 1.3343021e-07 2.3343273e-17
 5.8112640e-11 1.6306804e-17 9.9999988e-01 6.5498596e-12 1.4347487e-09]
예측 클래스: 7
```

전체 딥러닝 프로세스 적용

- 전체 딥러닝 프로세스 적용

1. 데이터 로드 및 전처리
    - x_train과 x_test는 입력 데이터(이미지)
    - 데이터를 [0, 1] 범위로 정규화 하여 신경망을 학습
    - 레이블(y_train, y_test)은 원-핫 인코딩하여 다중 분류 작업에 맞게 변환
2. 모델 구성
    - 입력층 → 은닉층 → 출력층으로 이루어진 신경망을 설계
    - 출력층의 Softmax 활성화 함수는 각 클래스에 대한 확률을 계산
3. 모델 컴파일
    - 손실 함수로 Categorical Crossentropy를 사용하여 실제 값과 예측 확률의 차이를 줄이는 방향으로 학습
    - 옵티마이저로 Adam을 사용하여 모델의 가중치를 효율적으로 업데이트

전체 딥러닝 프로세스 적용

- 전체 딥러닝 프로세스 적용

5. 모델 훈련
    - fit 메서드를 사용해 모델을 훈련
    - 훈련 데이터를 기반으로 가중치를 학습하고, 검증 데이터를 통해 과적합을 방지
6. 모델 평가
    - evaluate 메서드를 사용해 테스트 데이터로 모델의 성능(정확도)을 평가
7. 예측
    - 훈련된 모델을 사용해 새로운 데이터의 클래스를 예측
    - Softmax 출력은 각 클래스에 대한 확률 값을 반환하고,
      argmax를 사용해 가장 높은 확률의 클래스를 선택