Chapter 1

딥러닝 및 머신러닝 이론

> ## 학습 목표

- AI(인공지능), ML(머신러닝), DL(딥러닝)의 개념과 차이점을 이해한다.
- 지도학습(Supervised Learning), 비지도학습(Unsupervised Learning), 강화학습(Reinforcement Learning)의 차이를 이해한다.
- 신경망(Neural Network)의 기본 작동 원리(입력, 가중치, 편향, 활성화 함수, 역전파 등)와 학습 내용을 직접 실습, 구현할 수 있다.

1.1 AI-ML-DL의 구조

■ 인공지능(AI, Artificial Intelligence)

> 정의

- 인간의 지능을 모방하는 컴퓨터 시스템, 학습, 추론, 문제 해결, 언어 이해, 인지 등의 능력을 포함
- 목표: 인간처럼 사고하고 학습하는 기계 구현

■ 머신러닝(ML, Machine Learning)

AI의 하위 분야, 데이터를 학습하여 예측하거나 결정을 내리는 알고리즘을 개발하는 것을 의미

1. 지도학습 (Supervised Learning)

> 정의

- **레이블이 있는 데이터**를 학습하여 **입력과 출력 간의 관계**를 파악.

> 예시

- **입력**: 키와 몸무게 데이터 (특징).
- **출력**: 성별 (레이블).
- **알고리즘**:
  - 의사결정 트리 (Decision Tree)
  - 서포트 벡터 머신 (SVM)
  - 합성곱 신경망 (CNN)

> 과정

1. **학습 단계**

- 모델이 입력 데이터를 학습하여 레이블(정답)을 예측할 수 있는 능력을 만듦.

2. **테스트 단계**
   - 학습된 모델을 사용해 새로운 데이터에 대해 결과를 예측.
   - 모델의 성능을 평가.

---

2. 비지도학습 (Unsupervised Learning)

정의

- **레이블이 없는 데이터**를 분석하여 **패턴 인식, 군집화, 차원 축소** 등의 작업 수행.

주요 응용 분야

- 데이터 군집화 (Clustering).
- 차원 축소 (Dimensionality Reduction).

---

3. 강화학습 (Reinforcement Learning)

정의

- **에이전트(Agent)가** 환경(Environment)과 상호작용하며, **보상(Rewards)을** 통해 학습.

목표

- 최적 행동 방식을 학습하여 **장기적으로 최대 보상**을 얻는 전략 수립.

예시

- 게임 AI (예: 알파고).
- 자율 주행 자동차.

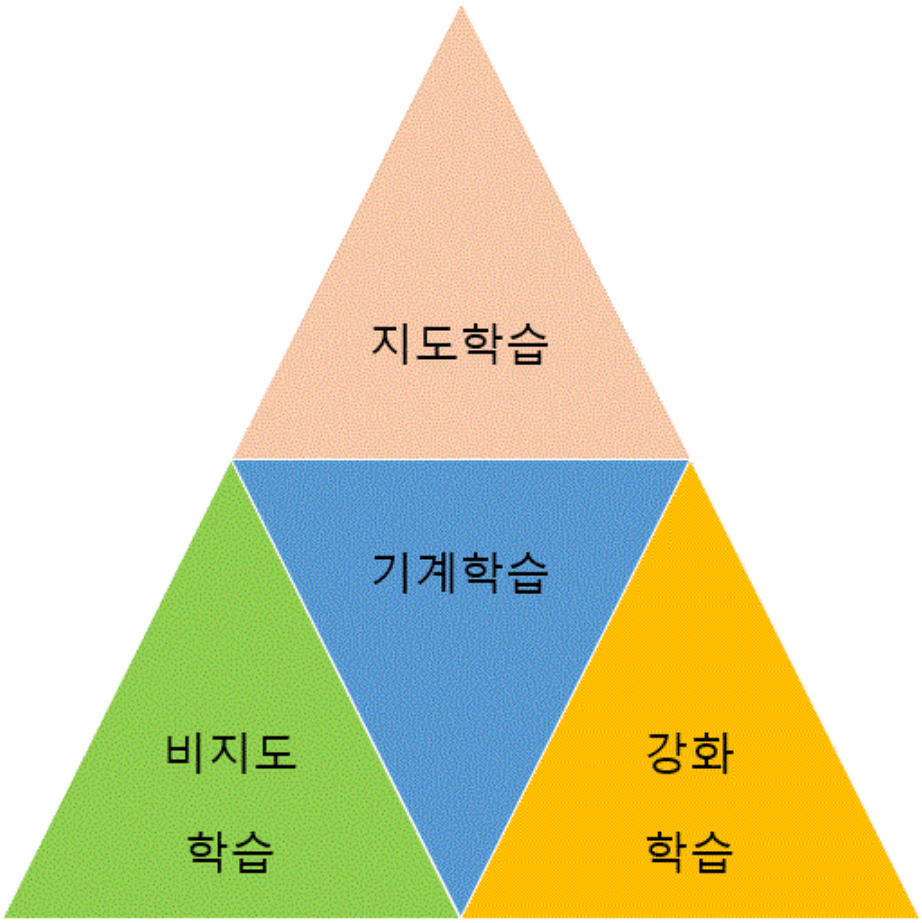


그림 1.1 머신러닝 학습

- Scikit-Learn 코드:
  - 빠르고 간단하며 직관적임.
  - 초보자에게 추천.
  - 복잡한 신경망을 필요로 하지 않는 간단한 지도학습 문제에 적합.
- PyTorch 코드:
  - 딥러닝을 배우거나, 더 복잡한 신경망 모델을 정의하고 학습할 때 유용.
  - 학습 데이터셋의 형태를 명확히 이해하며, 앞으로 GPU 환경에서도 쉽게 확장 가능.

[파이썬과 사이킷런활용 예제]

- 기본적인 머신러닝 프로세스를 잘 보여주며, 파이썬과 사이킷런을 이용한 데이터 분석, 모델 학습 및 평가의 패턴을 익힐 수 있는 좋은 예입니다.
- 다양한 모델과 데이터셋에 대해 이와 유사한 과정을 반복하며 실습하면 머신러닝의 기초를 더 확고히 할 수 있습니다.

In [ ]:
```
from sklearn.model_selection import train_test_split
# 데이터를 학습용과 테스트용으로 나누는 함수
from sklearn.datasets import load_iris # 아이리스 데이터셋을 불러오는 함수
from sklearn.linear_model import LogisticRegression
# 로지스틱 회귀 모델을 사용하는 클래스
from sklearn.metrics import accuracy_score # 모델의 정확도를 측정하는 함수
```

```
# 데이터 로드
data = load_iris()  # Iris 꽃 데이터셋 사용
# 아이리스 데이터셋은 150개의 샘플과 4개의 특성(꽃받침 길이, 꽃받침 폭, 꽃잎 길이, 꽃잎
X = data.data  # 입력 데이터
y = data.target  # 출력 데이터(클래스 라벨)
# X는 독립 변수(특성, 입력 데이터)이며, y는 종속 변수(클래스 레이블)입니다.

# 데이터 분할 (학습용/테스트용)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=
# train_test_split 함수를 사용하여 데이터를 학습용(80%)과 테스트용(20%)으로 분할# rando
# 같은 값으로 지정하면 매 실행 시 동일한 분할

# 로지스틱 회귀 모델 초기화 및 학습
model = LogisticRegression(max_iter=200)  # 최대 반복 횟수를 지정 (기본값 100)
model.fit(X_train, y_train)
# LogisticRegression(max_iter=200)는 로지스틱 회귀 모델을 초기화
# 반복 횟수를 200으로 설정하여 학습의 수렴 속도를 높입니다.
# fit() 메서드를 통해 학습용 데이터를 모델에 학습시킵니다. 이 과정을 통해 모델은 데이터

# 예측
predictions = model.predict(X_test)
# predict() 메서드를 통해 테스트용 데이터에 대한 예측 수행
# 학습된 모델을 사용하여 X_test의 각 샘플이 어떤 클래스에 속하는지를 예측

# 정확도 평가
accuracy = accuracy_score(y_test, predictions)
print(f"모델 정확도: {accuracy * 100:.2f}%")
# accuracy_score() 함수를 사용하여 예측값과 실제값(y_test)을 비교하여 모델의 정확도를 계
# 출력시, 정확도는 백분율 형식으로 포맷팅하여 표시합니다.
```

```
모델 정확도: 100.00%
```

간단한 뉴럴 네트워크를 사용한 이진 분류 프로그램으로 PyTorch를 활용하여 10개의 특징(feature)를 가진 데이터를 학습하고, 이를 기반으로 데이터를 이진 분류(0 또는 1)하는 모델을 훈련하는 예제입니다.

- 데이터를 생성 및 분리.
- 딥러닝 모델 설계.
- 모델 학습(Training) 및 테스트(Test).
- 결과 출력(테스트 손실 및 정확도).

학습 데이터는 가상의 데이터로 생성되며, 현실에서의 "합격 여부", "환자 건강 상태(정상 또는 이상)" 같은 이진 분류 문제를 모방한 예제입니다.

In [19]:
```
# 필요한 라이브러리 불러오기
import torch  # 딥러닝 프레임워크
import torch.nn as nn  # 신경망 설계를 위한 모듈
import torch.optim as optim  # 최적화 함수 (파라미터 업데이트)
from sklearn.datasets import make_classification  # 데이터를 생성하는 모듈
from sklearn.model_selection import train_test_split  # 데이터 분리 도구

# (1) 데이터 생성
# 1000개의 데이터 샘플을 생성하며, 각 데이터는 10개의 특징(feature)을 가짐
# 이진 분류를 수행하기 위해 2개의 클래스로 데이터 생성
X, y = make_classification(n_samples=1000, n_features=10, n_classes=2, random_state=42

# 데이터를 학습용(80%)과 테스트용(20%)으로 나누기
```

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# (2) 데이터를 PyTorch 텐서로 변환
# PyTorch는 내부적으로 tensor 데이터 타입을 사용함
# 학습용 데이터
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)  # 특징 데이터
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)  # 레이블 데이
# 테스트용 데이터
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

# (3) 신경망 모델 정의하기
# 신경망 모델인 "SimpleNN" 클래스 정의
class SimpleNN(nn.Module):  # PyTorch의 nn.Module을 상속받아 정의
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.layer = nn.Linear(10, 1)  # 선형 레이어. 입력 10개 -> 출력 1개

    def forward(self, x):
        # 순전파(forward) 함수. 입력 데이터를 받아 결과를 반환
        return torch.sigmoid(self.layer(x))
    # 활성화 함수로 시그모이드(Sigmoid) 사용

# 모델 객체 생성
model = SimpleNN()

# (4) 손실 함수와 최적화 함수 정의
# 손실 함수: BCELoss - 이진 분류(Binary Classification)를 위한 손실 함수
criterion = nn.BCELoss()
# 최적화 함수: SGD(Stochastic Gradient Descent) - 학습 속도는 0.01
optimizer = optim.SGD(model.parameters(), lr=0.01)

# (5) 모델 학습(Training)
epochs = 100  # 데이터 전체를 100번 학습
for epoch in range(epochs):  # 각 에포크(epoch) 반복
    model.train()  # 모델을 학습 모드로 전환
    optimizer.zero_grad()  # 이전 에포크에서 계산된 기울기(gradient) 초기화
    y_pred = model(X_train_tensor)  # 입력 데이터를 모델에 전달해 예측값 계산
    loss = criterion(y_pred, y_train_tensor)  # 예측값과 실제값을 비교해 손실 계산
    loss.backward()  # 손실에 따른 기울기(gradient) 계산
    optimizer.step()  # 기울기를 사용해 모델 파라미터(가중치, 편향) 업데이트

    # 10 에포크마다 손실 값을 출력
    if epoch % 10 == 0:
        print(f"에포크 {epoch} | 손실: {loss.item():.4f}")  # 현재 에포크와 손실 값 출

# (6) 모델 테스트(Test)
model.eval()  # 모델을 평가 모드로 전환
with torch.no_grad():  # 평가에서는 기울기를 계산하지 않음
    y_test_pred = model(X_test_tensor)  # 테스트 데이터를 사용해 예측값 계산
    y_test_pred = (y_test_pred > 0.5).float()  # 0.5를 기준으로 0 또는 1로 분류
    # 정확도 계산
    accuracy = (y_test_pred.eq(y_test_tensor).sum().item()) / y_test_tensor.shape[0]
    print(f"테스트 정확도: {accuracy * 100:.2f}%")  # 테스트 정확도 출력
```

```
에포크 0 | 손실: 0.5895
에포크 10 | 손실: 0.5726
에포크 20 | 손실: 0.5573
에포크 30 | 손실: 0.5433
에포크 40 | 손실: 0.5305
에포크 50 | 손실: 0.5188
에포크 60 | 손실: 0.5080
에포크 70 | 손실: 0.4981
에포크 80 | 손실: 0.4889
에포크 90 | 손실: 0.4804
테스트 정확도: 82.00%
```

딥러닝(DL, Deep Learning)

> 정의 : ML 중에서 최적화된 기술, 인공신경망 기반 학습 기법

- 핵심 구조: 다층 신경망(Neural Network)
  - 대규모 데이터, 특징 추출, 높은 성능, 비정형 데이터 처리
  - 딥러닝 응용 사례: 이미지 인식(ex. 고양이와 개 구분), 자연어 처리(ex. 번역 모델, 챗봇), 생성 모델(ex. GAN을 활용한 이미지 생성)
  - 인공신경망은 각 뉴런의 입력 신호 x에 신호의 가중치(weight) w를 곱한 값으로 신호가 들어오며, 임계값으로 편향 b를 더하여 뉴런의 신호 출력 여부를 결정합니다.
  - 뉴런간의 전달된 신호들의 값을 모두 합하고 활성화 함수(activation function)를 통해 값이 0보다 크면 1, 0보다 작으면 0을 출력한다.

DNN(Deep Neural Network)의 기본 구조

DNN은 크게 입력층(input layer), 은닉층(hidden layer), 출력층(output layer)으로 구성됩니다.

이들 층은 다음과 같은 방식으로 연결됩니다:

- 입력층 (Input Layer):
  - 모델에 전달되는 데이터가 첫 번째 층으로 입력됩니다.
  - 각 입력 노드는 데이터의 특성(feature)을 나타냅니다.
- 은닉층 (Hidden Layer):
  - 입력 데이터는 하나 이상의 은닉층을 통해 처리됩니다.
  - 은닉층은 가중치(weights)와 편향(bias)을 이용해 입력 데이터를 변형하고, 활성화 함수를 통해 비선형성을 추가하여 더 복잡한 패턴을 학습합니다.
  - DNN에서 중요한 점은 여러 개의 은닉층을 사용하여 데이터의 고차원적인 표현을 학습하는 것입니다.
- 출력층 (Output Layer):
  - 모델의 최종 예측값을 계산하는 층입니다.

- 분류 문제에서는 보통 소프트맥스(Softmax) 함수로 확률값을 출력하고, 회귀 문제에서는 선형(Linear) 출력을 제공합니다.

알고리즘

a. 합성곱 신경망(Convolutional Neural Network, CNN)은 이미지 데이터 처리에 사용되는 신경망입니다.

b. 순환 신경망(Recurrent Neural Network, RNN)은 순차 데이터(시계열 데이터) 처리를 위해 설계된 신경망입니다.

c. 트랜스포머(Transformers)는 자연어 처리에서 순차적으로 처리하지 않고 모든 입력 데이터를 동시에 처리할 수 있도록 설계된 모델입니다.

d. 생성적 적대 신경망(Generative Adversarial Network,GAN)은 생성자와 판별자를 상호 적대적으로 학습시켜 데이터를 생성하는 모델입니다.

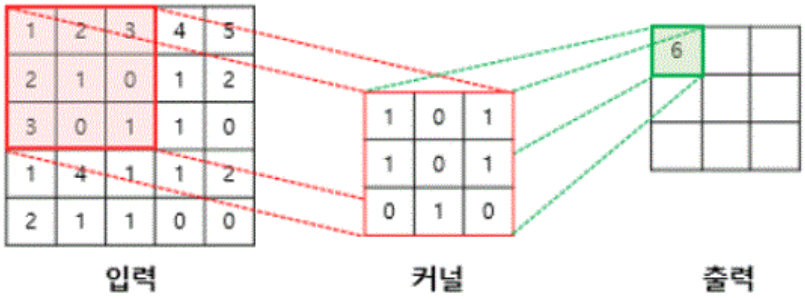
그림 1.2 AI-ML-DL 계층적 구조

합성곱 연산

CNN 합성곱 연산은 필터를 가중치로 사용하고 입력데이터 전체에 반복적으로 적용하여 변수의 수를 줄입니다.

필터를 적용하여 특징맵(feature map)을 추출하는 컨볼루션 레이어(convolution layer), 특징의 수를 줄이는 풀링 레이어(pooling layer),

모든 노드들이 연결되어 있는 완전연결 레이어(fully connected layer)로 구성된다.



$$(1\times1) + (2\times0) + (3\times1) + (2\times1) + (1\times0) + (0\times1) + (3\times0) + (0\times1) + (1\times0) = 6$$

그림 1.3 컨볼루션 연산

호환성

파이토치는 TensorFlow, Keras와 같은 다른 딥러닝 프레임워크와 비교하여 더 직관적이고 파이썬 친화적인 코드 작성이 가능합니다. 또한, PyTorch와 TensorFlow는 모델을 서로 변환할 수 있는 도구도 제공됩니다.

1.2 신경망의 기본 작동 원리

- 신경망의 핵심 작동 원리(입력 → 가중치 → 출력 → 손실 함수 → 역전파) 학습.
- 신경망 구조는 데이터를 입력하는 입력층(Input Layer),
- 입력 데이터를 분석하고 패턴을 학습하는 은닉층(Hidden Layer),
- 결과를 반환하는 출력층(Output Layer)으로 구성되어 있습니다.

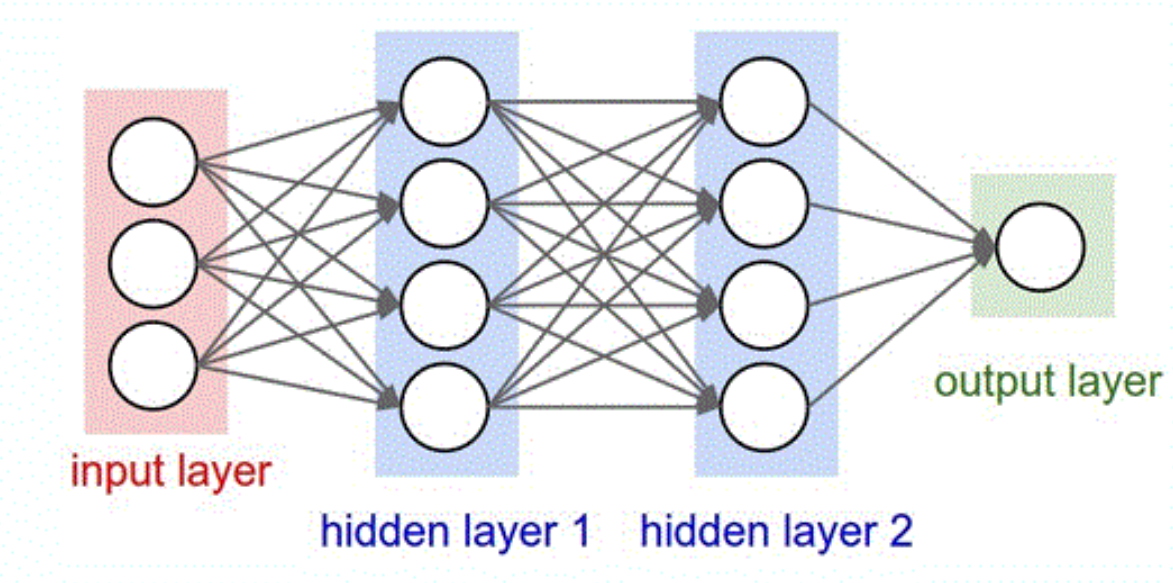


그림 1.4 신경망 구조(은닉층 2개)

- 입력 데이터 → 가중치 계산 (y = wx + b).
- 활성화 함수 (ex. Sigmoid, ReLU 등) 사용으로 비선형성을 추가.

- 출력 결과를 손실 함수(Loss Function)로 평가 후 역전파 학습.

코드 예제

PyTorch를 통해 신경망의 기본 논리 구현 알아보기

- 라이브러리 불러오기

In [20]:
```
import torch
import torch.nn as nn
import torch.optim as optim
```

- 입력 데이터와 레이블

In [21]:
```
import torch
import torch.nn as nn
import torch.optim as optim

inputs = torch.tensor([[1.0], [2.0], [3.0], [4.0]])
targets = torch.tensor([[2.0], [4.0], [6.0], [8.0]])
```

- 단순 선형 모델 정의 (y = Wx + b)

In [22]:
```
import torch
import torch.nn as nn
import torch.optim as optim

model = nn.Linear(1, 1)  # 입력 1차원, 출력 1차원
```

- 손실 함수와 옵티마이저 정의

In [14]:
```
import torch
import torch.nn as nn
import torch.optim as optim

criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
```

- 학습 루프

In [15]:
```
import torch  # PyTorch 라이브러리
import torch.nn as nn  # 신경망 설계를 위한 모듈
import torch.optim as optim  # 최적화를 위한 모듈

# 100번 학습 반복(Epoch)
for epoch in range(100):

    # 1. 모델을 사용하여 입력값(inputs)에 대한 출력값(outputs)을 계산
    #    즉, 현재 학습된 가중치와 편향을 바탕으로 예측을 수행
    outputs = model(inputs)
```

```
    # 2. 출력값(outputs)과 실제 정답(targets)을 비교해 손실(loss) 계산
    loss = criterion(outputs, targets)

    # 3. 기울기를 초기화 (이전 에포크의 기울기를 지움)
    optimizer.zero_grad()

    # 4. 손실(loss)에 기반한 기울기를 계산
    #    각 가중치와 편향(weight, bias)이 손실을 줄이기 위해 얼마나 조정되어야 하는지 저
    loss.backward()

    # 5. 계산된 기울기를 사용해 모델의 파라미터(가중치와 편향)를 업데이트
    optimizer.step()

# 최종 학습이 완료된 후, 학습된 가중치(weight)와 편향(bias)를 출력
print(f'학습된 가중치: {model.weight.item()}, 편향: {model.bias.item()}')
```

```
학습된 가중치: 1.6801958084106445, 편향: 0.9402633309364319
```

- 이 코드는 단순한 선형 회귀 로직(y = Wx + b)을 구현한 것으로, 딥러닝 신경망의 핵심 작업을 간략히 보여주고 있습니다.
- 모델이 "데이터로부터 가중치(W)와 편향(b)을 학습" 하는 과정입니다.
- loss = criterion(outputs, targets) # 손실 계산
- loss.backward() # 자동 미분
- optimizer.step() # 파라미터 업데이트

※ 이 코드의 핵심은:

- 모델이 학습: 데이터를 보고 손실을 최소화하기 위해 자신의 파라미터를 지속적으로 조정.
- 가중치와 편향 업데이트: 손실 감소를 위해 학습된 파라미터 출력.

※ 위 주석을 참고해 한 줄씩 분석하면서 학습하면 딥러닝 학습 과정의 기본 흐름을 이해할 수 있습니다.

1.3 데이터셋(dataset)

딥러닝에서 사용되는 데이터셋은 주로 문제의 유형에 따라 크게 구분할 수 있습니다. 주요 데이터셋의 구분은 다음과 같습니다:

1. **이미지 데이터셋**

- **분류(Classification)**: 주어진 이미지를 여러 클래스 중 하나로 분류하는 문제입니다.
  - 예시:
    - **MNIST** (Modified National Institute of Standards and Technology): 손으로 쓴 숫자 이미지 (0-9)
      - MNIST는 28x28 픽셀의 손글씨 숫자 이미지 약 70,000장을 포함하고 있으며, 이 중 약 60,000장은 훈련용, 10,000장은 테스트용으로 사용됩니다. 이 데이터셋은 기계 학습 및 딥러닝 연구의 표준 benchmark로 자리잡고 있습니다.
    - **CIFAR-10 / CIFAR-100** (Canadian Institute For Advanced Research): 10개/100개 클래스의 색상 이미지

        - **CIFAR-10** 구성: 10개의 클래스 (비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 무).
            - 이미지 수: 총 60,000개의 32x32 픽셀 컬러 이미지. (훈련 세트: 50,000개 이미지 / 테스트 세트: 10,000개 이미지.)
            - 특징: 비교적 단순한 분류 작업으로, 다양한 이미지 인식 알고리즘을 테스트하는 데 적합합니다.
        - **CIFAR-100** 구성: 100개의 클래스. 각 클래스는 60개 이미지로 이루어져 있으며, 20개의 상위 레이블로 분류됩니다 (예: 자전거와 같은 항목들은 '탈 것'라는 상위 레이블에 속합니다).
            - 이미지 수: 전체 60,000개의 32x32 픽셀 컬러 이미지.(훈련 세트: 50,000개 이미지 / 테스트 세트: 10,000개 이미지.)
            - 특징: CIFAR-10보다 더 많은 클래스와 더 세분화된 분류 문제를 제공하여, 보다 복잡한 모델을 평가할 수 있습니다.
    - **ImageNet**: 1000개 이상의 객체를 포함한 대형 이미지 데이터셋
- **객체 검출(Object Detection)**: 이미지에서 객체의 위치와 클래스를 예측하는 문제입니다.
    - 예시:
        - **COCO**: 객체 검출, 키포인트 인식, 이미지 캡셔닝 등 다양한 태스크를 지원하는 데이터셋
- **세그멘테이션(Segmentation)**: 이미지를 픽셀 단위로 분할하는 문제입니다.
    - 예시:
        - **PASCAL VOC**: 객체 검출 및 세그멘테이션을 포함한 이미지 데이터셋
        - **Cityscapes**: 도시 환경에서의 픽셀 수준 이미지 분할 데이터셋

그림 1.5 mnist_손글씨

2. **텍스트 데이터셋**

- **자연어 처리(NLP)**: 텍스트를 처리하는 문제로, 텍스트 분류, 번역, 감정 분석, 질의 응답 등 다양한 태스크가 포함됩니다.
    - **IMDB**: 영화 리뷰 감정 분석 데이터셋
    - **SQuAD**: 질문-답변 데이터셋
    - **GLUE**: 여러 자연어 처리 태스크를 위한 벤치마크 데이터셋

3. 시계열 데이터셋

- **시계열 예측(Time Series Forecasting)**: 시간에 따른 데이터 변화를 예측하는 문제입니다.
    - **Kaggle에서 제공하는 시계열 데이터셋**: 주식, 날씨 예측 등 다양한 시계열 데이터

4. 추천 시스템(Recommendation Systems) 데이터셋

- 사용자에게 아이템을 추천하는 문제입니다.
    - **MovieLens**: 영화 추천 데이터셋
    - **Netflix Prize Dataset**: 넷플릭스 영화 추천 데이터셋

딥러닝 예제 맛보기

- FashionMNIST 활용 모델 만들기

In [25]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 1. 데이터 준비
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0

train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True, transf
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True, transf

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)

# 2. 모델 정의 (간단한 신경망)
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 128)
        # 입력: 28x28 크기의 이미지 -> 128 차원
        self.fc2 = nn.Linear(128, 10)
        # 출력: 10개의 클래스 (패션 아이템 종류)

    def forward(self, x):
        x = x.view(-1, 28 * 28)  # 이미지를 1D 벡터로 변환
        x = torch.relu(self.fc1(x))  # 활성화 함수 ReLU
        x = self.fc2(x)              # 출력
        return x
```

```python
# 3. 모델 초기화 및 설정
model = SimpleNN()
criterion = nn.CrossEntropyLoss()  # 다중 클래스 분류를 위한 손실 함수
optimizer = optim.Adam(model.parameters(), lr=0.001)  # Adam 옵티마이저

# 4. 모델 훈련
def train(model, train_loader, criterion, optimizer, epochs=5):
    model.train()
    for epoch in range(epochs):
        running_loss = 0.0     # 에폭마다 손실 초기화
        for data, target in train_loader:
            optimizer.zero_grad()  # 기울기 초기화
            output = model(data)   # 모델 예측
            loss = criterion(output, target)  # 손실 계산
            loss.backward()        # 역전파
            optimizer.step()       # 가중치 업데이트

            running_loss += loss.item()

        print(f'Epoch {epoch+1}, Loss: {running_loss / len(train_loader):.4f}')

# 5. 모델 평가
def test(model, test_loader):
    model.eval()  # 평가 모드로 설정
    correct = 0
    total = 0
    with torch.no_grad():  # 그래디언트 계산을 하지 않음
        for data, target in test_loader:
            output = model(data)
            _, predicted = torch.max(output, 1)
            # 가장 높은 확률을 가진 클래스를 예측
            total += target.size(0)
            correct += (predicted == target).sum().item()  # 맞춘 개수 카운트

    accuracy = 100 * correct / total
    print(f'Test Accuracy: {accuracy:.2f}%')

# 훈련
train(model, train_loader, criterion, optimizer, epochs=5)

# 테스트
test(model, test_loader)
```

```
Epoch 1, Loss: 0.4979
Epoch 2, Loss: 0.3849
Epoch 3, Loss: 0.3475
Epoch 4, Loss: 0.3220
Epoch 5, Loss: 0.3030
Test Accuracy: 86.62%
```

In [29]:

```python
import matplotlib.pyplot as plt

# 시각화를 위한 함수 정의
def visualize_predictions(model, test_loader, num_images=10):
    """
    모델의 예측 결과를 시각화하고, 예측 라벨과 실제 라벨을 비교합니다.

    :param model: 학습된 PyTorch 모델
    :param test_loader: 테스트 데이터 로더
    :param num_images: 시각화할 이미지의 수
    """
```

```
    # 모델을 평가 모드로 설정
    model.eval()

    # 테스트 데이터에서 배치 하나 가져오기
    data_iter = iter(test_loader)
    images, labels = next(data_iter)

    # 모델로 예측 수행
    outputs = model(images)
    _, predicted = torch.max(outputs, 1)  # 가장 높은 확률을 가진 클래스 선택

    # FashionMNIST 클래스 이름 정의
    classes = [
        'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'
    ]

    # 시각화
    plt.figure(figsize=(15, 4))
    for idx in range(num_images):
        ax = plt.subplot(2, num_images // 2, idx + 1)
        ax.imshow(images[idx].numpy().squeeze(), cmap="gray")  # 이미지를 Grayscale 형
        ax.set_title(f"Pred: {classes[predicted[idx]]}\nTrue: {classes[labels[idx]]}")
        ax.axis('off')
    plt.tight_layout()
    plt.show()


# 시각화 함수 호출
visualize_predictions(model, test_loader)
```