Chapter 7

순전파 및 역전파

학습 목표

- **순전파의 개념 이해** : 신경망에서 순전파가 어떻게 이루어지는지 설명하고, 입력 데이터가 출력으로 변환되는 과정을 단계별로 이해할 수 있다.
- **역전파의 원리 이해** : 역전파(backpropagation)의 기본 원리와 기울기 계산 과정을 이해하고, 손실 함수의 기울기를 통해 네트워크 파라미터를 업데이트하는 방법을 설명할 수 있다.
- **순전파와 역전파 연결 이해** : 순전파와 역전파가 신경망 학습 과정에서 어떻게 상호작용하는지를 이해하고, 두 과정의 역할을 명확히 구분할 수 있다.
- **파이토치에서의 구현** : 파이토치를 사용하여 순전파와 역전파 과정이 포함된 신경망 모델을 구현하고, 이를 통해 학습 과정을 시뮬레이션할 수 있다.

순전파(Forward Propagation) & 역전파(Backward Propagation)

■ 순전파(Forward Propagation)

순전파는 **입력 데이터를 신경망의 각 계층을 거쳐 출력 값으로 변환**하는 과정입니다.

1. **목적**:
    - 모델이 입력 데이터를 받아 예측 결과를 생성하는 과정입니다.
    - 손실(loss)을 계산하기 위해 예측 값을 생성합니다.
2. **작동 방식**:
    - 입력 데이터가 첫 번째 계층(입력 계층)에서 시작해 네트워크의 각 계층(은닉 계층 및 출력 계층)을 통과합니다.
    - 각 계층에서 `가중치(weight)와 편향(bias)`이 적용되고, 활성화 함수(activation function)가 적용되어 출력값을 계산합니다.
    - 마지막 출력 계층에서 결과값(예: 분류 확률)이 도출됩니다.
3. **결과**:
    - 네트워크의 최종 출력 값과 실제 값 간의 오차(손실)를 계산합니다.
    - 손실 함수(loss function)가 사용됩니다(예: MSE, Cross-Entropy 등).

■ 역전파(Backward Propagation)

역전파는 **손실을 기반으로 네트워크의 가중치와 편향을 업데이트하기 위해 기울기(gradient)를 계산하는 과정**입니다.

1. **목적**:
    - 손실 함수의 값을 최소화하기 위해 가중치와 편향을 조정합니다.
    - 네트워크가 학습하도록 합니다.
2. **작동 방식**:
    - 손실 함수로부터 시작해 네트워크의 마지막 계층에서 첫 번째 계층(입력 계층) 방향으로 거꾸로 진행됩니다.
    - 각 계층에서 `기울기(gradient)`를 계산하여 가중치와 편향의 변화량(업데이트 방향)을 결정합니다.
    - 체인 룰(chain rule)을 사용해 각 계층의 기울기를 계산합니다.
3. **가중치 업데이트**:
    - 경사하강법(Gradient Descent) 등을 사용해 가중치를 업데이트합니다.
4. **결과**:
    - 네트워크의 가중치와 편향이 업데이트되며, 모델의 성능이 점진적으로 개선됩니다.

7.1 순전파(Forward Propagation)

순전파

- 입력 데이터가 네트워크에 들어가면 각 층에서 순차적으로 연산이 이루어집니다.
- 각 뉴런은 입력값에 가중치를 곱하고 그 결과를 활성화 함수에 통과시켜 출력을 생성합니다.
- 이 과정은 네트워크의 각 층을 차례로 지나면서 최종 출력값을 얻는 과정입니다.
- 파이토치에서 순전파는 forward() 메서드를 사용하여 구현됩니다.

- **순전파 단계**
    - 입력층 : 네트워크에 입력 데이터가 들어옵니다.
    - 은닉층 : 입력 데이터는 가중치(weight)와 편향(bias)을 곱한 후 활성화 함수(activation function)를 거쳐 은닉층의 출력으로 변환됩니다.
    - 출력층 : 은닉층의 출력값은 다시 가중치와 편향을 거쳐 최종 출력값을 생성합니다.
    - 손실 계산 : 네트워크의 출력값과 실제 정답값을 비교하여 손실(loss)을 계산합니다.

순전파 구현

In [1]:
```
import torch
import torch.nn as nn

# 1. 신경망 모델 정의
class SimpleNN(nn.Module):   # nn.Module을 상속하여 신경망 모델을 정의
    def __init__(self):      # __init__ 는 신경망의 레이어를 정의
        super(SimpleNN, self).__init__()
        # 입력층 1, 은닉층 10, 출력층 1
        self.fc1 = nn.Linear(1, 10)  # 입력층에서 은닉층
        self.fc2 = nn.Linear(10, 1)  # 은닉층에서 출력층
    # 순전파 구현
```

```
    def forward(self, x):
        # 첫 번째 층에 대해 ReLU 활성화 함수 적용
        x = torch.relu(self.fc1(x))
        # 두 번째 층 (출력층)
        x = self.fc2(x)
        return x

# 2. 모델 초기화
model = SimpleNN()

# 3. 임의의 입력 데이터 생성 (배치 크기 3, 입력 크기 1)
input_data = torch.tensor([[1.0], [2.0], [3.0]])

# 4. 순전파 실행
# model(input_data)를 호출하여 입력 데이터를 모델에 통과시켜 출력값을 계산
output_data = model(input_data)

# 5. 출력 확인
print("입력 데이터:\n", input_data)
print("모델 출력:\n", output_data)
```

```
입력 데이터:
 tensor([[1.],
        [2.],
        [3.]])
모델 출력:
 tensor([[-0.0325],
        [-0.1132],
        [-0.2249]], grad_fn=<AddmmBackward0>)
```

7.2 역전파(Backward Propagation)

- 역전파는 순전파 후 계산된 손실을 기반으로 가중치와 편향을 업데이트하는 과정입니다.
- **경사 하강법(Gradient Descent)** 을 이용하여 수행됩니다.

- 역전파 단계
- 출력층에서 오차 계산: 출력층에서 네트워크의 예측값과 실제값 간의 차이를 계산합니다. 손실함수를 통해 오차를 구합니다.
- 오차의 미분값 계산: 오차를 이용해 가중치를 조정할 수 있도록 오차를 역으로 전파합니다.
  - 출력층에서부터 각 층으로 오차의 기울기를 계산합니다.
  - 체인 룰(Chain Rule)을 사용하여 각 층의 가중치가 오차에 미친 영향을 계산합니다.
  - 체인 룰은 복합 함수의 미분을 계산하는 방법으로, 신경망에서 각 층의 미분값을 곱하여 오차를 전파하는 데 사용됩니다.
- 가중치와 편향 업데이트: 계산된 기울기를 사용하여 가중치와 편향을 업데이트합니다.

역전파 구현

- 2개의 클래스를 가진 이진 분류 문제
- 손실함수는 CrossEntropyLoss

적용 실습

In [2]:
```
import torch
import torch.nn as nn
import torch.optim as optim
```

In [4]:
```
# 데이터 준비 X: 입력, y: 목표 출력
X = torch.tensor([[0.0, 0.0],
                  [0.0, 1.0],
                  [1.0, 0.0],
                  [1.0, 1.0]], dtype=torch.float32)

y = torch.tensor([0, 1, 1, 0], dtype=torch.long)  # XOR 문제 (레이블 0과 1)
```

In [5]:
```
# 신경망 클래스 정의
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(2, 2)  # 입력층 → 은닉층 (2개 뉴런)
        self.fc2 = nn.Linear(2, 2)  # 은닉층 → 출력층 (2개 클래스)

    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))  # 은닉층에서 시그모이드 활성화 함수
        x = self.fc2(x)  # 출력층은 CrossEntropyLoss 사용
        return x

# 모델 초기화
model = SimpleNN()
```

In [6]:
```
# 손실 함수: CrossEntropyLoss (소프트맥스 포함)
criterion = nn.CrossEntropyLoss()

# 옵티마이저: 경사 하강법 (SGD)
optimizer = optim.SGD(model.parameters(), lr=0.1)

# 학습 과정 (순전파, 역전파, 가중치 업데이트)
epochs = 10000
for epoch in range(epochs): # 1000번 에폭 반복
    # 순전파: 입력 데이터를 모델에 통과시켜 예측값을 얻음
    output = model(X)

    # 손실 계산
    loss = criterion(output, y)

    # 역전파
    optimizer.zero_grad()  # 기울기 초기화
    loss.backward()  # 기울기 계산
    optimizer.step()  # 가중치 업데이트

    # 1000번마다 손실 출력
    if epoch % 1000 == 0:
        print(f'Epoch {epoch}, Loss: {loss.item()}')
```

```
Epoch 0, Loss: 0.8996731638908386
Epoch 1000, Loss: 0.6721575260162354
Epoch 2000, Loss: 0.5699892044067383
Epoch 3000, Loss: 0.4408780336380005
Epoch 4000, Loss: 0.16908827424049377
Epoch 5000, Loss: 0.07245902717113495
Epoch 6000, Loss: 0.04325035959482193
Epoch 7000, Loss: 0.030279584228992462
Epoch 8000, Loss: 0.023122671991586685
Epoch 9000, Loss: 0.018631787970662117
```

In [7]:
```python
# 학습한 모델에서 최종 예측
with torch.no_grad(): # 기울기 계산 비활성화
    output = model(X)
    predicted = torch.argmax(output, dim=1) # 샘플에 대해 가장 높은 확률을 가진 클래스
    print("last prediction:", predicted.numpy())  # x에 대한 레이블 y값 예측
```

```
last prediction: [0 1 1 0]
```

1. 신경망 정의 (SimpleNN)

- fc1은 첫 번째 선형층(입력층 → 은닉층)입니다.
- fc2는 두 번째 선형층(은닉층 → 출력층)입니다.
- forward 함수는 순전파를 정의합니다. 입력을 은닉층에 통과시키고, 활성화 함수로 시그모이드를 적용한 후 출력층으로 전달합니다.

2. 모델 초기화 : 모델을 인스턴스화하고, 손실 함수와 최적화 방법을 설정합니다.

3. 훈련 과정

- outputs = model(inputs)로 순전파가 이루어집니다.
- loss = criterion(outputs, targets)로 출력값과 실제 값 간의 손실을 계산합니다.
- loss.backward()로 역전파를 통해 기울기를 계산하고, optimizer.step()으로 가중치를 업데이트합니다.

데이터셋 `load_breast_cancer` 이용한 모델 만들기

- load_breast_cancer 데이터셋은 scikit-learn 라이브러리에서 제공하는 유방암(Breast Cancer) 진단에 관한 데이터셋입니다.

Breast Cancer Wisconsin (Diagnostic) Dataset은 유방암 종양의 양성과 악성을 진단하는 데 사용되는 데이터셋으로, 1995년에 공개되었습니다.

이 데이터셋은 사이킷런(sklearn)의 데이터셋 모듈에서도 제공되며, 머신러닝 연구 및 교육 목적으로 널리 사용됩니다.

1. 데이터셋 소개

- **출처**: UCI Machine Learning Repository
- **목적**: 유방암 종양의 진단(양성 또는 악성) 예측
- **크기**:
  - **샘플 수**: 569개

- **특성 수**: 30개 (수치형)
- **클래스 수**: 2개 (양성 `B: Benign`, 악성 `M: Malignant`)

2. 특성 정보

데이터셋은 유방암 종양 세포의 핵 특징을 기반으로 합니다. 30개의 수치형 특성은 다음과 같은 10가지 측정 지표의 평균값, 표준편차, 최댓값을 포함합니다:

1. **Radius**: 핵의 반지름(거리 중심-경계)
2. **Texture**: 그레이스케일의 표준편차
3. **Perimeter**: 경계 길이
4. **Area**: 핵 면적
5. **Smoothness**: 경계의 매끄러움
6. **Compactness**: 주변 면적 대비 둘레^2
7. **Concavity**: 경계의 오목한 부분의 정도
8. **Concave points**: 경계의 오목한 지점 수
9. **Symmetry**: 대칭성
10. **Fractal dimension**: 경계의 복잡성

각 지표별로:

- `_mean`: 평균값
- `_se`: 표준 오차
- `_worst`: 최댓값
  이렇게 3가지의 값이 제공됩니다. 따라서 총 10 × 3 = 30개의 특성이 생성됩니다.

3. 클래스 정보

- **M (Malignant)**: 악성 (212개 샘플, 약 37%)
- **B (Benign)**: 양성 (357개 샘플, 약 63%)

In [8]:
```
%pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\asus\anaconda3\lib\site-packages (2.
2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\asus\appdata\roaming\python\py
thon312\site-packages (from pandas) (1.26.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\asus\anaconda3\lib\si
te-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\asus\anaconda3\lib\site-package
s (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\asus\anaconda3\lib\site-packa
ges (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\asus\appdata\roaming\python\python3
12\site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [9]:
```
%pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\asus\anaconda3\lib\site-package
s (1.5.1)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: numpy>=1.19.5 in c:\users\asus\appdata\roaming\python\py
thon312\site-packages (from scikit-learn) (1.26.3)
Requirement already satisfied: scipy>=1.6.0 in c:\users\asus\anaconda3\lib\site-package
s (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.2.0 in c:\users\asus\anaconda3\lib\site-packag
es (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\asus\anaconda3\lib\site
-packages (from scikit-learn) (3.5.0)
```

In [10]:
```
%pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\asus\anaconda3\lib\site-packages (0.
13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in c:\users\asus\appdata\roaming\py
thon\python312\site-packages (from seaborn) (1.26.3)
Requirement already satisfied: pandas>=1.2 in c:\users\asus\anaconda3\lib\site-packages
(from seaborn) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in c:\users\asus\anaconda3\lib\s
ite-packages (from seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\asus\anaconda3\lib\site-pac
kages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in c:\users\asus\anaconda3\lib\site-package
s (from matplotlib!=3.6.1,>=3.4->seaborn) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\asus\anaconda3\lib\site-pa
ckages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.51.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\asus\anaconda3\lib\site-pa
ckages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\asus\appdata\roaming\python
\python312\site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in c:\users\asus\anaconda3\lib\site-packages
(from matplotlib!=3.6.1,>=3.4->seaborn) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\asus\anaconda3\lib\site-pac
kages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\asus\anaconda3\lib\site
-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\asus\anaconda3\lib\site-package
s (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\asus\anaconda3\lib\site-packa
ges (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\asus\appdata\roaming\python\python3
12\site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [11]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
```

In [12]:
```
# 1. 데이터셋 로딩
data = load_breast_cancer()
X = data.data
```

```
y = data.target
print(data.DESCR)
```

.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset

Data Set Characteristics:

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:
    - radius (mean of distances from center to points on the perimeter)
    - texture (standard deviation of gray-scale values)
    - perimeter
    - area
    - smoothness (local variation in radius lengths)
    - compactness (perimeter^2 / area - 1.0)
    - concavity (severity of concave portions of the contour)
    - concave points (number of concave portions of the contour)
    - symmetry
    - fractal dimension ("coastline approximation" - 1)

    The mean, standard error, and "worst" or largest (mean of the three
    worst/largest values) of these features were computed for each image,
    resulting in 30 features. For instance, field 0 is Mean Radius, field
    10 is Radius SE, field 20 is Worst Radius.

    - class:
            - WDBC-Malignant
            - WDBC-Benign

:Summary Statistics:

	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2

```
area (worst):                          185.2  4254.0
smoothness (worst):                    0.071  0.223
compactness (worst):                   0.027  1.058
concavity (worst):                     0.0    1.252
concave points (worst):                0.0    0.291
symmetry (worst):                      0.156  0.664
fractal dimension (worst):             0.055  0.208
====================================== ====== ======

:Missing Attribute Values: None

:Class Distribution: 212 - Malignant, 357 - Benign

:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L. Mangasarian

:Donor: Nick Street

:Date: November, 1995

This is a copy of UCI ML Breast Cancer Wisconsin (Diagnostic) datasets.
https://goo.gl/U2Uwz2

Features are computed from a digitized image of a fine needle
aspirate (FNA) of a breast mass.  They describe
characteristics of the cell nuclei present in the image.

Separating plane described above was obtained using
Multisurface Method-Tree (MSM-T) [K. P. Bennett, "Decision Tree
Construction Via Linear Programming." Proceedings of the 4th
Midwest Artificial Intelligence and Cognitive Science Society,
pp. 97-101, 1992], a classification method which uses linear
programming to construct a decision tree.  Relevant features
were selected using an exhaustive search in the space of 1-4
features and 1-3 separating planes.

The actual linear program used to obtain the separating plane
in the 3-dimensional space is that described in:
[K. P. Bennett and O. L. Mangasarian: "Robust Linear
Programming Discrimination of Two Linearly Inseparable Sets",
Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server:

ftp ftp.cs.wisc.edu
cd math-prog/cpo-dataset/machine-learn/WDBC/

.. dropdown:: References

  - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction
    for breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on
    Electronic Imaging: Science and Technology, volume 1905, pages 861-870,
    San Jose, CA, 1993.
  - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast cancer diagnosis and
    prognosis via linear programming. Operations Research, 43(4), pages 570-577,
    July-August 1995.
  - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machine learning techniques
    to diagnose breast cancer from fine-needle aspirates. Cancer Letters 77 (1994)
    163-171.
```

사이킷런에서의 데이터셋

- 사이킷런에서 load_breast_cancer를 사용하여 쉽게 접근할 수 있습니다.

In [13]:
```
# 특성과 레이블
X = data.data  # 입력 특성
y = data.target  # 타겟 (0: 악성, 1: 양성)

# 특성 이름과 타겟 이름
print(data.feature_names)
print(data.target_names)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
['malignant' 'benign']
```

In [16]:
```
# 2. 데이터프레임 생성
df = pd.DataFrame(X, columns=data.feature_names)
df['target'] = y
df.tail(3)
```

Out[16]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	n symm
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.

3 rows × 31 columns

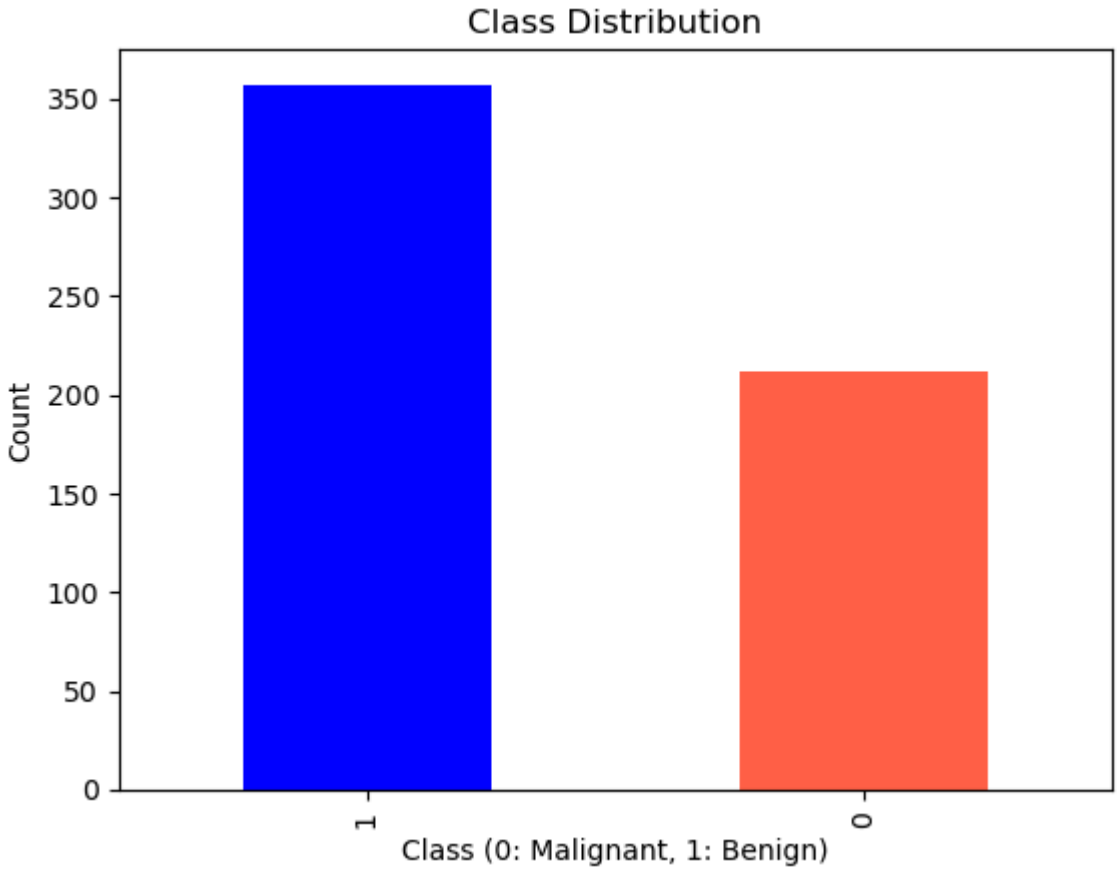
In [17]:
```
# 데이터 크기와 기본 통계

# 데이터프레임으로 변환
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# 데이터 개요
print(df.info())
print(df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32  
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
None
       mean radius  mean texture  mean perimeter    mean area  \
count   569.000000    569.000000      569.000000   569.000000   
mean     14.127292     19.289649       91.969033   654.889104   
std       3.524049      4.301036       24.298981   351.914129   
min       6.981000      9.710000       43.790000   143.500000   
25%      11.700000     16.170000       75.170000   420.300000   
50%      13.370000     18.840000       86.240000   551.100000   
75%      15.780000     21.800000      104.100000   782.700000   
max      28.110000     39.280000      188.500000  2501.000000   

       mean smoothness  mean compactness  mean concavity  mean concave points  \
count       569.000000        569.000000      569.000000           569.000000   
mean          0.096360          0.104341        0.088799             0.048919   
std           0.014064          0.052813        0.079720             0.038803   
min           0.052630          0.019380        0.000000             0.000000   
25%           0.086370          0.064920        0.029560             0.020310   
50%           0.095870          0.092630        0.061540             0.033500   
75%           0.105300          0.130400        0.130700             0.074000   
max           0.163400          0.345400        0.426800             0.201200   

       mean symmetry  mean fractal dimension  ...  worst texture  \
```

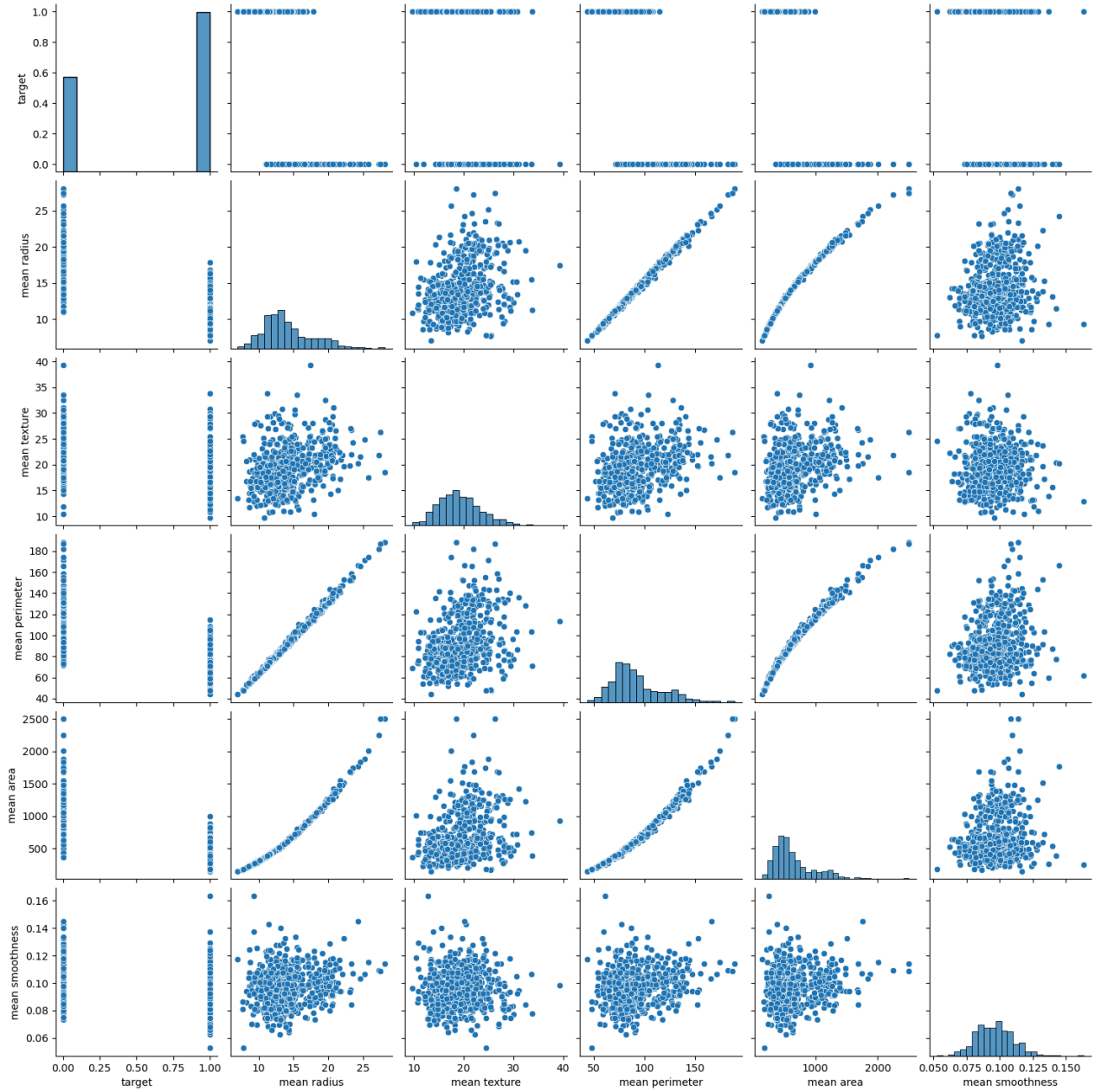
```
count     569.000000              569.000000  ...     569.000000
mean        0.181162                0.062798  ...      25.677223
std         0.027414                0.007060  ...       6.146258
min         0.106000                0.049960  ...      12.020000
25%         0.161900                0.057700  ...      21.080000
50%         0.179200                0.061540  ...      25.410000
75%         0.195700                0.066120  ...      29.720000
max         0.304000                0.097440  ...      49.540000

       worst perimeter   worst area  worst smoothness  worst compactness  \
count       569.000000   569.000000        569.000000         569.000000
mean        107.261213   880.583128          0.132369           0.254265
std          33.602542   569.356993          0.022832           0.157336
min          50.410000   185.200000          0.071170           0.027290
25%          84.110000   515.300000          0.116600           0.147200
50%          97.660000   686.500000          0.131300           0.211900
75%         125.400000  1084.000000          0.146000           0.339100
max         251.200000  4254.000000          0.222600           1.058000

       worst concavity  worst concave points  worst symmetry  \
count       569.000000            569.000000      569.000000
mean          0.272188              0.114606        0.290076
std           0.208624              0.065732        0.061867
min           0.000000              0.000000        0.156500
25%           0.114500              0.064930        0.250400
50%           0.226700              0.099930        0.282200
75%           0.382900              0.161400        0.317900
max           1.252000              0.291000        0.663800

       worst fractal dimension      target
count               569.000000  569.000000
mean                  0.083946    0.627417
std                   0.018061    0.483918
min                   0.055040    0.000000
25%                   0.071460    0.000000
50%                   0.080040    1.000000
75%                   0.092080    1.000000
max                   0.207500    1.000000

[8 rows x 31 columns]
```

In [19]:

```python
# 클래스 분포 확인

import matplotlib.pyplot as plt

# 클래스 분포 시각화
df['target'].value_counts().plot(kind='bar', color=['blue', 'tomato'])
plt.title('Class Distribution')
plt.xlabel('Class (0: Malignant, 1: Benign)')
plt.ylabel('Count')
plt.show()
```



In [20]:
```
# 샘플 몇 개만 확인
sns.pairplot(df[['target'] + list(df.columns[:5])])
plt.show()
```



In [40]:
```python
# 3. 데이터 전처리 (훈련/테스트 데이터 분리 및 정규화)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# 표준화: StandardScaler를 사용하여 데이터 정규화
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# 데이터를 파이토치 텐서로 변환
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)

# 4. 신경망 모델 정의
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(30, 64)  # 30개의 입력 특성
        self.fc2 = nn.Linear(64, 32)  # 중간층
        self.fc3 = nn.Linear(32, 1)   # 출력층
        self.sigmoid = nn.Sigmoid()   # 이진 분류를 위한 시그모이드 활성화 함수
```

```
    def forward(self, x):
        x = torch.relu(self.fc1(x))  # 첫 번째 층: ReLU 활성화
        x = torch.relu(self.fc2(x))  # 두 번째 층: ReLU 활성화
        x = self.fc3(x)              # 출력층
        x = self.sigmoid(x)          # 시그모이드 활성화
        return x

# 5. 모델 초기화
model = SimpleNN()
criterion = nn.BCELoss()  # 이진 분류 문제이므로 BCELoss 사용
optimizer = optim.Adam(model.parameters(), lr=0.01)

# 6. 학습
epochs = 100
for epoch in range(epochs):
    # 순전파
    output = model(X_train_tensor)

    # 손실 계산
    loss = criterion(output, y_train_tensor)

    # 역전파
    optimizer.zero_grad()  # 기울기 초기화
    loss.backward()        # 기울기 계산
    optimizer.step()       # 파라미터 업데이트

    # 10번마다 손실 값 출력
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# 7. 테스트
with torch.no_grad():
    # 모델 예측
    output = model(X_test_tensor)
    predicted = (output > 0.5).float()  # 0.5를 기준으로 이진 분류

    # 정확도 계산
    accuracy = accuracy_score(y_test, predicted.numpy())
    print(f'Accuracy on test set: {accuracy * 100:.2f}%')
```

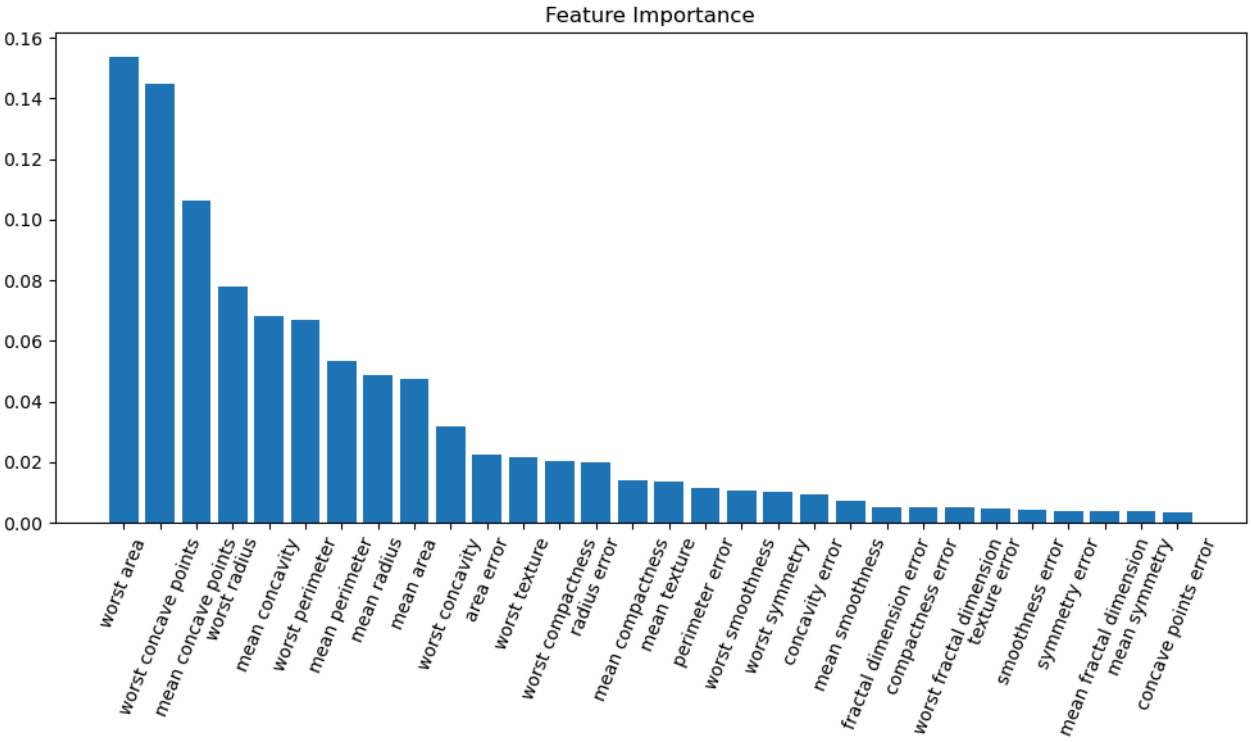
```
Epoch [10/100], Loss: 0.0829
Epoch [20/100], Loss: 0.0552
Epoch [30/100], Loss: 0.0277
Epoch [40/100], Loss: 0.0152
Epoch [50/100], Loss: 0.0082
Epoch [60/100], Loss: 0.0044
Epoch [70/100], Loss: 0.0022
Epoch [80/100], Loss: 0.0009
Epoch [90/100], Loss: 0.0005
Epoch [100/100], Loss: 0.0004
Accuracy on test set: 98.25%
```

In [41]:
```
# 데이터 분할

from sklearn.model_selection import train_test_split

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

In [45]:
```python
# 분류 모델 훈련

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

# 모델 생성 및 학습
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# 예측 및 평가
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9649122807017544
              precision    recall  f1-score   support

           0       0.98      0.93      0.95        43
           1       0.96      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114
```

In [46]:
```python
# 랜덤 포레스트를 사용한 특성 중요도 시각화:

import matplotlib.pyplot as plt
import numpy as np

# 특성 중요도 추출
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

# 시각화
plt.figure(figsize=(10, 6))
plt.title("Feature Importance")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), data.feature_names[indices], rotation=68)
plt.tight_layout()
plt.show()
```



데이터셋의 활용

- 머신러닝 및 딥러닝 기법의 학습과 실험
- 분류 문제 연구 및 알고리즘 비교
- 특성 선택 및 중요도 분석

※ Breast Cancer Wisconsin (Diagnostic) Dataset은 크기가 작아 초급자와 전문가 모두에게 적합하며, 다양한 머신러닝 알고리즘의 성능 비교에 자주 사용됩니다.

너는 파이토치 강사야, 내가 파이토치의 순전파, 역전파에 대해서 학습중인데 유방암 관련 소스는 실행해봤고, 다른 소스등으로 조금 더 배워보고 싶어, 추천하는 예제를 파이토치 소스로 주석을 달아서 자세하게 구현해줘.

In [47]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# 1. 데이터 준비
transform = transforms.Compose([
    transforms.ToTensor(),  # 이미지를 텐서로 변환
    transforms.Normalize((0.5,), (0.5,))  # 픽셀값 [-1, 1] 정규화
])

train_dataset = datasets.MNIST(
    root='./data',
    train=True,
    download=True,
    transform=transform
)
```

```python
test_dataset = datasets.MNIST(
    root='./data',
    train=False,
    transform=transform
)

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

# 2. 모델 정의
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),  # 입력층(784) → 은닉층(512)
            nn.ReLU(),              # 비선형 활성화 함수
            nn.Linear(512, 10)      # 은닉층(512) → 출력층(10)
        )

    def forward(self, x):
        """순전파(Forward Propagation) 과정 정의"""
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork()

# 3. 손실 함수와 최적화기
loss_fn = nn.CrossEntropyLoss()  # 분류 문제에 적합한 손실 함수
optimizer = optim.SGD(model.parameters(), lr=0.01)  # 경사 하강법 최적화기

# 4. 학습 루프
def train_loop():
    model.train()  # 학습 모드 설정
    for batch, (images, labels) in enumerate(train_loader):
        # 순전파
        pred = model(images)          # 모델 예측
        loss = loss_fn(pred, labels)  # 손실 계산

        # 역전파
        optimizer.zero_grad()  # 이전 그래디언트 초기화
        loss.backward()        # 자동 미분(역전파) 수행
        optimizer.step()       # 매개변수 업데이트

        if batch % 100 == 0:
            print(f"Batch {batch}: loss={loss.item():.3f}")

# 5. 평가 함수
def test_loop():
    model.eval()  # 평가 모드 설정
    test_loss = 0
    correct = 0

    with torch.no_grad():  # 그래디언트 계산 비활성화
        for images, labels in test_loader:
            pred = model(images)
            test_loss += loss_fn(pred, labels).item()
            correct += (pred.argmax(1) == labels).type(torch.float).sum().item()
```

```
    print(f"\nTest Error:\nAccuracy: {correct/len(test_loader.dataset):.2f}")

# 6. 모델 학습 실행
epochs = 5
for epoch in range(epochs):
    print(f"Epoch {epoch+1}\n-------------------------")
    train_loop()
    test_loop()
print("학습 완료!")

# 7. 예측 결과 시각화
def visualize_predictions():
    model.eval()
    dataiter = iter(test_loader)
    images, labels = next(dataiter)

    with torch.no_grad():
        preds = model(images)

    fig = plt.figure(figsize=(12, 8))
    for idx in range(6):
        ax = fig.add_subplot(2, 3, idx+1)
        ax.imshow(images[idx].squeeze(), cmap='gray')
        ax.set_title(f"Pred: {preds[idx].argmax().item()}, True: {labels[idx]}")
    plt.show()

visualize_predictions()
```

```
Epoch 1
-------------------------
Batch 0: loss=2.255
Batch 100: loss=1.325
Batch 200: loss=0.893
Batch 300: loss=0.574
Batch 400: loss=0.540
Batch 500: loss=0.391
Batch 600: loss=0.405
Batch 700: loss=0.416
Batch 800: loss=0.363
Batch 900: loss=0.276

Test Error:
Accuracy: 0.89
Epoch 2
-------------------------
Batch 0: loss=0.552
Batch 100: loss=0.440
Batch 200: loss=0.509
Batch 300: loss=0.491
Batch 400: loss=0.391
Batch 500: loss=0.439
Batch 600: loss=0.363
Batch 700: loss=0.427
Batch 800: loss=0.466
Batch 900: loss=0.194

Test Error:
Accuracy: 0.91
Epoch 3
-------------------------
Batch 0: loss=0.386
Batch 100: loss=0.358
Batch 200: loss=0.376
Batch 300: loss=0.388
Batch 400: loss=0.265
Batch 500: loss=0.213
Batch 600: loss=0.230
Batch 700: loss=0.340
Batch 800: loss=0.285
Batch 900: loss=0.287

Test Error:
Accuracy: 0.92
Epoch 4
-------------------------
Batch 0: loss=0.184
Batch 100: loss=0.317
Batch 200: loss=0.369
Batch 300: loss=0.297
Batch 400: loss=0.271
Batch 500: loss=0.195
Batch 600: loss=0.172
Batch 700: loss=0.399
Batch 800: loss=0.268
Batch 900: loss=0.210

Test Error:
Accuracy: 0.93
```

```
Epoch 5
-------------------------
Batch 0: loss=0.213
Batch 100: loss=0.349
Batch 200: loss=0.413
Batch 300: loss=0.448
Batch 400: loss=0.551
Batch 500: loss=0.119
Batch 600: loss=0.195
Batch 700: loss=0.350
Batch 800: loss=0.257
Batch 900: loss=0.296

Test Error:
Accuracy: 0.93
학습 완료!
```