Pytorch 시작하기

학습 내용

- 파이토치가 무엇일까요?
- 타이타닉 데이터 셋을 활용한 딥러닝 모델 구축

파이토치(pytorch)는 무엇일까요?

- (1) Facebook에서 개발한 오픈 소스 머신 러닝 라이브러리.
- (2) PyTorch는 파이썬을 기반으로 한다.
- (3) 주요 특징
  - 동적 계산 그래프 : PyTorch는 실행 시점에 계산 그래프를 생성한다.
  - 강력한 GPU 지원
  - 파이썬과 친숙한 문법과 사용성 제공
  - PyTorch는 자동으로 기울기(gradient)를 계산하는 기능을 제공한다.
  - 연구자와 실무자 모두에게 인기가 있다.

목차

01. 사전 환경 확인
02. 라이브러리 및 데이터 불러오기
03. 신경망 모델 정의
04. 예측 수행
05. 추가 학습

01. 사전 환경 확인

목차로 이동하기

In [1]:
```
import torch

# PyTorch 버전 확인
print(torch.__version__)

# CUDA 사용 가능 여부 확인
print(torch.cuda.is_available())

# 사용 가능한 GPU 장치 수 확인
print(torch.cuda.device_count())
```

```
2.5.0+cu121
True
1
```

02. 라이브러리 및 데이터 불러오기

목차로 이동하기

In [2]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

In [3]:
```
# 시드를 고정하면 동일한 코드를 실행할 때마다 정확히 같은 결과를 얻을 수 있습니다
# 시드 고정
torch.manual_seed(42)   # PyTorch의 난수 생성기 고정
np.random.seed(42)      # NumPy의 난수 생성기 고정

# 1. 데이터 준비
# 데이터 로드
data = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/m
data.head(3)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.

In [4]:
```
data.isnull().sum()
```

Out[4]:

	0
PassengerId	0
Survived	0
Pclass	0
Name	0
Sex	0
Age	177
SibSp	0
Parch	0
Ticket	0
Fare	0
Cabin	687
Embarked	2

dtype: int64

In [5]:
```
# 필요한 피처 선택
features = ['Pclass', 'Sex', 'Age']
target = 'Survived'

# 성별 인코딩
data['Sex'] = data['Sex'].map({'male': 0, 'female': 1})

# 결측값 처리
imputer = SimpleImputer(strategy='median')
X = imputer.fit_transform(data[features])
y = data[target].values

# Numpy 배열을 다시 Pandas DataFrame으로 변환
X_df = pd.DataFrame(X, columns=features)

# 결측값 확인 (결측값 처리 후)
print("\n결측값 처리 후 데이터:")
print(X_df.isnull().sum())
```

```
결측값 처리 후 데이터:
Pclass    0
Sex       0
Age       0
dtype: int64
```

In [6]:
```
# 스케일링
# 스케일링은 대부분의 머신러닝 및 딥러닝 모델에서 매우 중요한 전처리 단계
# StandardScaler는 각 피처의 평균을 0, 표준편차를 1로 만듭니다.
# 모든 피처를 동일한 스케일로 조정합니다.
# 경사 하강법(Gradient Descent) 기반 알고리즘의 수렴 속도를 개선
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
# 데이터 분할
# 80%, 20%
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=42)

# NumPy to PyTorch Tensor 변환
# X_train이라는 NumPy 배열을 PyTorch의 FloatTensor로 변환.
print("변환 전")
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

X_train = torch.FloatTensor(X_train)
X_test = torch.FloatTensor(X_test)
y_train = torch.FloatTensor(y_train).unsqueeze(1)
y_test = torch.FloatTensor(y_test).unsqueeze(1)

print("변환 후")
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
변환 전
(712, 3) (179, 3) (712,) (179,)
변환 후
torch.Size([712, 3]) torch.Size([179, 3]) torch.Size([712, 1]) torch.Size([179,
1])
```

03. 신경망 모델 정의

목차로 이동하기

- 딥러닝의 이해를 위해 일부 특징(변수)만 지정하였음.
- 이미지를 사용할 때는 지정된 이미지 전체를 입력 데이터로 사용하는 경우가 대부분.

In [7]:
```
# 2. 신경망 모델 정의
model = nn.Sequential(
    nn.Linear(3, 8),
    nn.ReLU(),
    nn.Linear(8, 1),
    nn.Sigmoid()
)

# 3. 모델 학습
# 손실 함수와 옵티마이저(최적화) 정의

# BCE Loss : 모델의 예측 확률과 실제 레이블 간의 차이를 측정
criterion = nn.BCELoss()  # 이진 분류 손실 함수

# 최적화 함수 : 딥러닝에서 가장 널리 사용되는 최적화 알고리즘 중의 하나.
# 모멘텀(Momentum)과 RMSprop의 장점을 결합
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

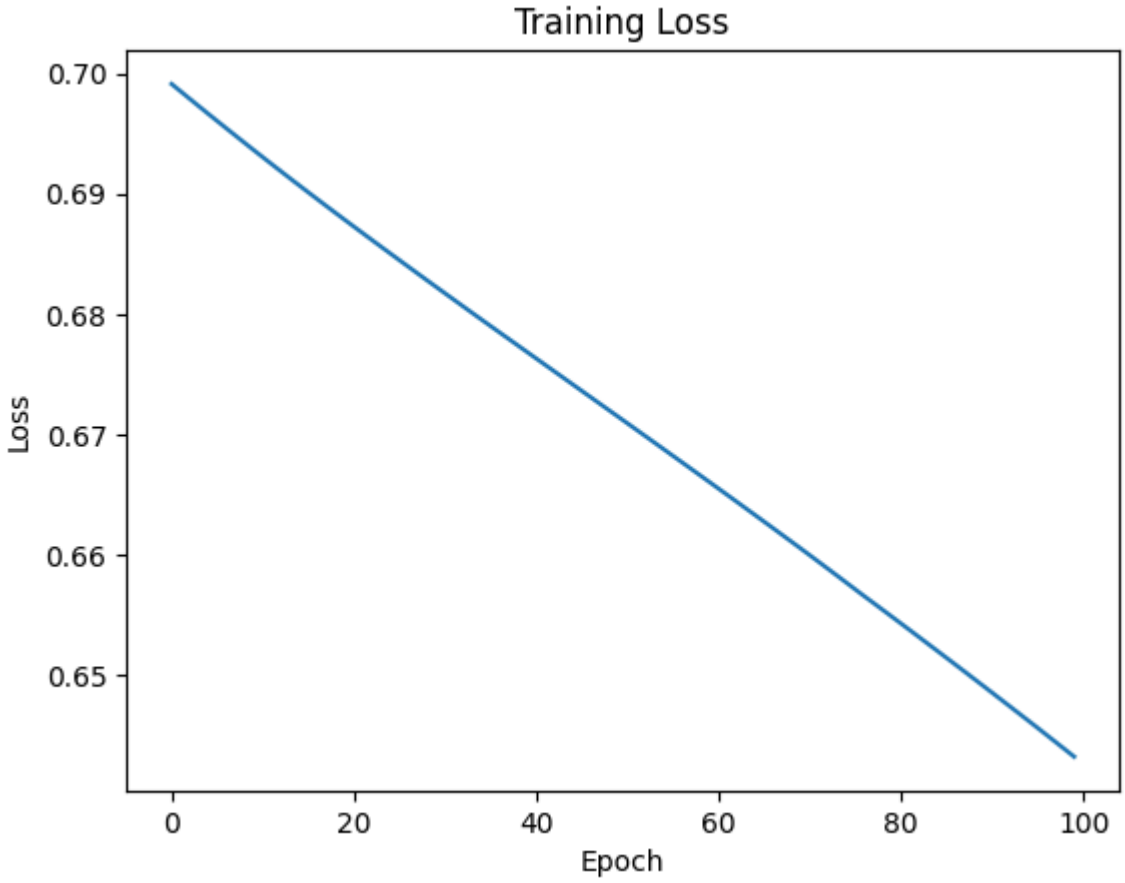
In [8]:
```
# 학습 진행
# 학습 과정 모니터링을 위한 코드
losses = []

epochs = 100     # 전체 데이터셋을 100번 반복 학습
for epoch in range(epochs):
```

```
    # 순전파
    # 입력 데이터를 모델에 통과시켜 예측값 생성
    outputs = model(X_train)

    # 예측값과 실제 레이블 간 차이(손실) 계산
    loss = criterion(outputs, y_train)

    # 손실 기록
    losses.append(loss.item())

    # 역전파
    # 이전 반복의 기울기 초기화 (매우 중요!)
    optimizer.zero_grad()    # 누적된 기울기를 0으로 초기화

    # 손실(loss)을 기반으로 각 가중치의 기울기 계산
    # 자동미분을 통한 기울기 계산
    loss.backward()

    # 가중치 업데이트
    # 옵티마이저가 계산된 기울기를 사용해 가중치 조정
    optimizer.step()

    # 20번마다 손실 출력
    if (epoch + 1) % 20 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [20/100], Loss: 0.6878
Epoch [40/100], Loss: 0.6768
Epoch [60/100], Loss: 0.6660
Epoch [80/100], Loss: 0.6548
Epoch [100/100], Loss: 0.6432
```

In [9]:

```
# 손실 곡선 시각화
import matplotlib.pyplot as plt

plt.plot(losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



In [10]:
```
# 4. 모델 평가
model.eval()  # 평가 모드
with torch.no_grad():
    test_outputs = model(X_test)
    predicted = (test_outputs > 0.5).float()
    accuracy = (predicted == y_test).float().mean()
    print(f'Test Accuracy: {accuracy.item():.4f}')
```

```
Test Accuracy: 0.7374
```

실습 과제 1-5

- (1) epoch를 늘려보자. 어떤 결과가 보여지는가?
- (2) 신경망의 은닉층을 늘려보자. 어떤 결과가 보여지는가? 비교 분석
- (3) 신경망의 은닉층의 뉴런를 늘려보자. 어떤 결과가 보여지는가? 비교 분석

도전 실습 과제 1-6

- (4) 딥러닝 성능을 다양한 것을 이용해서 최고 성능으로 만들어보자.

05. 부록 및 추가학습

목차로 이동하기

01 BCE LOSS이해하기

```
    # BCE Loss = -[y * log(p) + (1-y) * log(1-p)]
    # - y: 실제 레이블 (0 or 1)
    # - p: 모델이 예측한 확률
```

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

# 손실 함수 생성
criterion = nn.BCELoss()

# 모델 예측 (Sigmoid 출력)
predictions = torch.tensor([0.7, 0.2, 0.9])
# 실제 레이블
targets = torch.tensor([1.0, 0.0, 1.0])

# 손실 계산
loss = criterion(predictions, targets)
print("Loss:", loss.item())
```

```
Loss: 0.22839303314685822
```

설명

주요 특징

```
0에 가까울수록 좋은 예측
완벽한 예측: 손실 = 0
최악의 예측: 손실 = 매우 큰 값
```

사용 조건

```
마지막 레이어에 Sigmoid 활성화 함수 필요
출력값이 0~1 사이여야 함
```

02 Adam의 추가 이해

- Adam(Adaptive Moment Estimation) 옵티마이저는 딥러닝에서 가장 널리 사용되는 최적화 알고리즘 중 하나

Adam 옵티마이저의 핵심 특징

- 적응형 학습률을 가진 최적화 알고리즘
- 모멘텀(Momentum)과 RMSprop의 장점을 결합
- 대부분의 딥러닝 문제에 잘 작동

Adam의 장점

- 적응형 학습률
- 다양한 문제에 대해 좋은 성능
- 하이퍼파라미터 튜닝이 상대적으로 쉬움
- 희소 그래디언트에 강건함

- 신경망 학습 과정에서 그래디언트(기울기)가 매우 작아지거나 사라지는 현상
- 역전파 과정에서 그래디언트가 연쇄적으로 곱해지면서 매우 작아짐
- 활성화 함수(특히 Sigmoid, Tanh)로 인해 기울기 소실
- 특히 깊은 신경망에서 발생하는 문제

학습률(lr=0.001)의 의미

- 0.001: 일반적으로 사용되는 기본 학습률
- 너무 크면: 발산 위험
- 너무 작으면: 학습 속도 저하

03 훈련 모드와 평가모드의 이해

훈련 모드 (model.train()):

- 드롭아웃 활성화
- 배치 정규화(Batch Normalization) 업데이트 허용
- 그래디언트 계산 활성화

평가 모드 (model.eval()):

- 드롭아웃 비활성화
- 배치 정규화 통계 고정
- 그래디언트 계산 중지

주의할 내용

- 훈련 후 반드시 평가 모드로 전환
- with torch.no_grad(): 메모리 및 계산 효율성
- 예측/추론 시 항상 eval() 모드 사용

평가 모드는 필수이다.

- model.eval(): 모델 평가의 핵심
- 일관된 및 신뢰할 수 있는 예측을 위해 필수
- 훈련과 평가 간 모드 전환 중요

In [ ]: