PyTorch 핵심 치트시트

PyTorch 기본 조작

임포트 및 환경 확인

PyTorch 라이브러리를 임포트하고 설치 버전을 확인합니다.

```
import torch
from torch import nn

# 설치된 PyTorch 버전 확인
print(f"PyTorch version: {torch.__version__}")
```

텐서 (Tensors) 생성 및 연산

PyTorch의 핵심 데이터 구조인 텐서는 다차원 배열을 다루는 데 최적화되어 있습니다.

```
# 스칼라 값으로부터 텐서 생성
scalar = torch.tensor(7)

# 특정 크기(3x4)의 무작위 값을 가진 텐서 생성
random_tensor = torch.rand(size=(3, 4))

# 텐서 간의 요소별 곱셈 연산
tensor1 = torch.rand(size=(3, 4))
tensor2 = torch.rand(size=(3, 4))
result = tensor1 * tensor2
```

장치 설정 (Device Agnostic Code)

코드가 CPU, NVIDIA GPU(CUDA), Apple Silicon(MPS) 등 다양한 하드웨어 가속기에서 유연하게 동작하도록 설정합니다.

```
# 현재 환경에서 사용 가능한 최적의 장치를 탐색합니다.
if torch.cuda.is_available():
    device = "cuda"  # NVIDIA GPU 환경
elif torch.backends.mps.is_available():
    device = "mps"   # Apple Silicon 환경
else:
    device = "cpu"   # 가속기가 없는 경우 기본 CPU 사용

print(f"현재 사용 중인 장치: {device}")

# 생성한 텐서를 해당 장치로 이동시켜 연산을 준비합니다.
x = torch.tensor([1, 2, 3])
x = x.to(device)
print(f"텐서 위치: {x.device}")
```

실험 재현성을 위한 시드 고정

매번 실행할 때마다 동일한 결과를 보장하기 위해 랜덤 시드를 설정합니다.

```
torch.manual_seed(42)
random_tensor_A = torch.rand(3, 4)

torch.manual_seed(42)
random_tensor_B = torch.rand(3, 4)

# random_tensor_A와 random_tensor_B는 완벽하게 일치합니다.
```

신경망 구축 (Neural Networks) (torch.nn)

주요 레이어 (Layers)

신경망의 뼈대를 구성하는 핵심 레이어들입니다.

선형 레이어 (Linear Layers)

```
# 입력 특성 10개, 출력 특성 10개인 전결합층 정의
linear_layer = nn.Linear(in_features=10, out_features=10)
```

합성곱 레이어 (Convolutional Layers)

```
# 1D Convolution (주로 텍스트나 시계열 데이터 처리)
conv1d = nn.Conv1d(in_channels=1, out_channels=10, kernel_size=3)

# 2D Convolution (이미지 인식의 핵심 레이어)
conv2d = nn.Conv2d(in_channels=3, out_channels=10, kernel_size=3)

# 3D Convolution (비디오 또는 의료 영상 데이터 처리)
conv3d = nn.Conv3d(in_channels=3, out_channels=10, kernel_size=3)
```

트랜스포머 레이어 (Transformer Layers)

```
# Transformer 모델 전체 정의
transformer_model = nn.Transformer()

# 특정 사양의 Transformer 인코더 레이어 정의
transformer_encoder = nn.TransformerEncoderLayer(d_model=768, nhead=12)
```

순환 레이어 (Recurrent Layers)

```
# LSTM (Long Short-Term Memory) 레이어
lstm_stack = nn.LSTM(input_size=10, hidden_size=10, num_layers=3)

# GRU (Gated Recurrent Unit) 레이어
gru_stack = nn.GRU(input_size=10, hidden_size=10, num_layers=3)
```

활성화 함수 (Activation Functions)

데이터에 비선형성을 부여하여 모델의 표현력을 높입니다.

```
relu = nn.ReLU()
sigmoid = nn.Sigmoid()
softmax = nn.Softmax(dim=1)
```

손실 함수 (Loss Functions)

모델의 예측값과 실제 정답 사이의 오차를 측정합니다.

```
# 회귀 분석용 (Regression)
l1_loss = nn.L1Loss()       # MAE (평균 절대 오차)
mse_loss = nn.MSELoss()     # MSE (평균 제곱 오차)

# 이진 분류용 (Binary Classification)
bce_loss = nn.BCEWithLogitsLoss()

# 다중 클래스 분류용 (Multi-class Classification)
cross_entropy_loss = nn.CrossEntropyLoss()
```

옵티마이저 (Optimizers)

측정된 오차를 바탕으로 모델의 가중치를 업데이트하여 성능을 최적화합니다.

```
# 학습할 모델과 파라미터 정의
model = nn.Linear(1, 1)
params = model.parameters()

# 확률적 경사 하강법 (SGD)
sgd_optimizer = torch.optim.SGD(params=params, lr=0.01)

# 아담(Adam) 옵티마이저 (범용적으로 가장 널리 쓰임)
adam_optimizer = torch.optim.Adam(params=params, lr=0.001)
```

표준적인 모델 학습 워크플로우

1단계: 커스텀 모델 정의

`nn.Module` 클래스를 상속받아 고유한 신경망 구조를 설계합니다.

```
class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(10, 20)
        self.layer2 = nn.Linear(20, 5)

    def forward(self, x):
        # 데이터 흐름(순전파) 정의
        x = torch.relu(self.layer1(x))
        return self.layer2(x)

# 모델 인스턴스 생성 및 장치 할당
model = MyModel().to(device)
```

2단계: 손실 함수 및 옵티마이저 설정

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

3단계: 학습 루프 (Training Loop)

```
epochs = 100
for epoch in range(epochs):
    # 훈련 모드 설정 (Dropout, BatchNorm 활성화)
    model.train()
    for X_batch, y_batch in train_dataloader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        # 1. 순전파 (Forward pass): 예측 수행
        y_pred = model(X_batch)

        # 2. 손실 계산 (Calculate loss)
        loss = loss_fn(y_pred, y_batch)

        # 3. 기울기 초기화 (Zero gradients)
        optimizer.zero_grad()

        # 4. 역전파 (Backward pass): 기울
```

```
기 계산
        loss.backward()

        # 5. 가중치 업데이트 (Step
optimizer)
        optimizer.step()
```

4단계: 평가 루프 (Evaluation Loop)

```
# 평가 모드 설정 (Dropout 등 비활성화)
model.eval()
# 기울기 계산을 생략하여 메모리 아끼고 속도 향
상
with torch.inference_mode():
    for X_test, y_test in
test_dataloader:
        X_test, y_test =
X_test.to(device), y_test.to(device)
        test_pred = model(X_test)
        test_loss = loss_fn(test_pred,
y_test)

    print(f"테스트 손실값:
{test_loss:.4f}")
```