PyTorch 치트시트

PyTorch 기본

임포트

PyTorch를 사용하기 위해 필요한 기본 라이브러리를 임포트합니다.

```
import torch
from torch import nn

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

텐서 (Tensors)

PyTorch의 핵심 데이터 구조인 텐서를 생성하는 방법입니다.

```
// 스칼라 텐서 생성
let scalar = torch.tensor(7)

// 무작위 텐서 생성 (3x4 크기)
let random_tensor = torch.rand(size=(3,
4))

// 텐서 곱셈
let tensor1 = torch.rand(size=(3, 4))
let tensor2 = torch.rand(size=(3, 4))
let result = tensor1 * tensor2
```

장치 설정 (Device Agnostic Code)

코드가 CPU, GPU, MPS 등 다양한 환경에서 동작하도록 설정합니다.

```
// 사용 가능한 장치 확인 및 설정
if torch.cuda.is_available():
    device = "cuda"  // NVIDIA GPU
elif torch.backends.mps.is_available():
    device = "mps"   // Apple GPU
else:
    device = "cpu"

print(f"Using device: {device}")

// 텐서를 해당 장치로 이동
let x = torch.tensor([1, 2, 3])
x = x.to(device)
print(x.device)
```

랜덤 시드 고정

실험의 재현성을 위해 랜덤 시드를 고정합니다.

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

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

// random_tensor_A와 random_tensor_B는 동
일
```

신경망 (Neural Networks) (torch.nn)

레이어 (Layers)

신경망을 구성하는 기본 단위인 레이어입니다.

Linear Layers

```
// 선형 레이어
let linear_layer = nn.Linear(
  in_features: 10, out_features: 10
)
```

Convolutional Layers (CNN)

```
// 1D Convolution (텍스트)
let conv1d = nn.Conv1d(
  in_channels: 1, out_channels: 10,
kernel_size: 3
)
// 2D Convolution (이미지)
let conv2d = nn.Conv2d(
  in_channels: 3, out_channels: 10,
kernel_size: 3
)
// 3D Convolution (비디오)
let conv3d = nn.Conv3d(
  in_channels: 3, out_channels: 10,
kernel_size: 3
)
```

Transformer Layers

```
// Transformer 모델
let transformer_model = nn.Transformer()

// Transformer 인코더 레이어
let transformer_encoder =
nn.TransformerEncoderLayer(
  d_model: 768, nhead: 12
)
```

Recurrent Layers (RNN)

```
// LSTM 레이어
let lstm_stack = nn.LSTM(
  input_size: 10, hidden_size: 10,
num_layers: 3
)
// GRU 레이어
let gru_stack = nn.GRU(
  input_size: 10, hidden_size: 10,
num_layers: 3
)
```

활성화 함수 (Activation Functions)

신경망에 비선형성을 추가하는 함수입니다.

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

손실 함수 (Loss Functions)

모델의 예측이 실제 값과 얼마나 다른지를 측정합니다.

```
// 회귀 문제용 (Regression)
let l1_loss = nn.L1Loss()       // MAE
let mse_loss = nn.MSELoss()     // MSE

// 이진 분류용 (Binary Classification)
let bce_loss = nn.BCEWithLogitsLoss()

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

옵티마이저 (Optimizers)

손실 함수 값을 최소화하도록 모델의 가중치를 업데이트합니다.

```
// 모델 파라미터 정의
let model = nn.Linear(1, 1)
let params = model.parameters()

// SGD 옵티마이저
let sgd_optimizer = torch.optim.SGD(
  params: params, lr: 0.01
)

// Adam 옵티마이저
let adam_optimizer = torch.optim.Adam(
  params: params, lr: 0.001
)
```

일반적인 학습 과정 (Typical Training Loop)

1. 모델 정의

nn.Module을 상속받아 커스텀 모델을 정의합니다.

```
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):
        x = torch.relu(self.layer1(x))
        return self.layer2(x)

let model = MyModel().to(device)
```

2. 손실 함수 및 옵티마이저 설정

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

3. 학습 루프 (Training Loop)

```
let epochs = 100
for epoch in range(epochs):
    // 훈련 모드
    model.train()
    for X_batch, y_batch in
train_dataloader:
        X_batch, y_batch =
X_batch.to(device), y_batch.to(device)

        // 1. Forward pass
        y_pred = model(X_batch)

        // 2. Calculate loss
        loss = loss_fn(y_pred, y_batch)

        // 3. Zero gradients
        optimizer.zero_grad()

        // 4. Backward pass
        loss.backward()

        // 5. Step optimizer
        optimizer.step()
```

4. 평가 루프 (Evaluation Loop)

```
// 평가 모드
model.eval()
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"Test Loss: {test_loss}")
```