

PyTorch 빠른 참조

텐서, 자동 미분, 신경망, 학습

텐서

텐서 생성

```
import torch
a = torch.tensor([1, 2, 3])
b = torch.zeros(2, 3)
c = torch.ones(3, 3)
d = torch.randn(2, 4) # normal dist
```

텐서 생성자

torch.zeros(m, n)	모두 0, 형태 (m, n)
torch.ones(m, n)	모두 1, 형태 (m, n)
torch.randn(m, n)	표준 정규 분포 난수
torch.arange(start, end, step)	균등 간격 값
torch.linspace(start, end, steps)	고정 개수의 점
torch.eye(n)	단위 행렬
torch.empty(m, n)	초기화되지 않은 메모리

NumPy 상호 운용

```
t = torch.from_numpy(np_array)
arr = tensor.numpy() # shares memory
t = torch.as_tensor(np_array)
```

자동 미분

그라디언트 추적

```
x = torch.tensor([2.0, 3.0],
                 requires_grad=True)
y = (x ** 2).sum()
y.backward()
print(x.grad) # tensor([4., 6.]
```

그라디언트 추적 비활성화

```
with torch.no_grad():
    pred = model(x) # inference only
    x_det = x.detach() # detach from graph
```

그라디언트 제어

x.requires_grad_(True)	제자리에서 그라드 추적 활성화
x.grad.zero_()	누적 그라디언트 초기화
x.detach()	그라드 이력 없는 새 텐서
x.grad	저장된 그라디언트 접근

신경망

모델 정의

```
import torch.nn as nn
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(784, 128)
        self.fc2 = nn.Linear(128, 10)
    def forward(self, x):
        x = torch.relu(self.fc1(x))
        return self.fc2(x)
```

Sequential 모델

```
model = nn.Sequential(
    nn.Linear(784, 256),
    nn.ReLU(),
    nn.Dropout(0.2),
    nn.Linear(256, 10))
```

주요 레이어

nn.Linear(in, out)	완전 연결 레이어
nn.Conv2d(c_in, c_out, k)	2D 합성곱, 커널 크기 k
nn.BatchNorm2d(n)	배치 정규화
nn.LSTM(in, hidden)	LSTM 순환 레이어
nn.Dropout(p)	확률 p의 드롭아웃
nn.Embedding(vocab, dim)	임베딩 조회 테이블

데이터 로딩

사용자 정의 Dataset

```
from torch.utils.data import Dataset, DataLoader
class MyData(Dataset):
    def __init__(self, X, y):
        self.X, self.y = X, y
    def __len__(self): return len(self.X)
    def __getitem__(self, i):
        return self.X[i], self.y[i]
```

DataLoader

```
loader = DataLoader(dataset, batch_size=32,
                    shuffle=True, num_workers=2)
for batch_x, batch_y in loader:
    output = model(batch_x)
```

내장 데이터셋

```
from torchvision import datasets, transforms
t = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))]
data = datasets.MNIST("data", train=True,
                      download=True, transform=t)
```

학습 루프

표준 학습 루프

```
model.train()
for epoch in range(num_epochs):
    for X, y in train_loader:
        optimizer.zero_grad()
        loss = criterion(model(X), y)
        loss.backward()
        optimizer.step()
```

평가

```
model.eval()
with torch.no_grad():
    correct = 0
    for X, y in test_loader:
        pred = model(X).argmax(dim=1)
        correct += (pred == y).sum().item()
```

학습 체크리스트

model.train()	드롭아웃 / 배치 정규화 학습 모드 활성화
model.eval()	추론 모드로 전환
optimizer.zero_grad()	역전파 전 그라디언트 초기화
loss.backward()	그라디언트 계산
optimizer.step()	파라미터 업데이트

옵티마이저

주요 옵티마이저

```
import torch.optim as optim
opt = optim.SGD(model.parameters(), lr=0.01,
                momentum=0.9)
opt = optim.Adam(model.parameters(), lr=1e-3)
opt = optim.AdamW(model.parameters(), lr=1e-3,
                  weight_decay=0.01)
```

학습률 스케줄러

```
sched = optim.lr_scheduler.StepLR(
    opt, step_size=10, gamma=0.1)
# in loop: sched.step() after each epoch
```

옵티마이저 비교

SGD	단순, 튜닝 필요, 모멘텀과 잘 어울림
Adam	적응형 학습률, 빠른 수렴, 기본 선택
AdamW	분리된 가중치 감소를 가진 Adam
RMSprop	적응형, RNN에 적합

손실 함수

주요 손실 함수

nn.CrossEntropyLoss()	분류 (로짓, softmax 없음)
nn.BCEWithLogitsLoss()	이진 분류 (로짓)
nn.MSELoss()	회귀 (평균 제곱 오차)
nn.L1Loss()	회귀 (평균 절대 오차)
nn.NLLLoss()	음의 로그 가능성도 (log_softmax 이후)
nn.HuberLoss()	강건한 회귀 (이상치에 덜 민감)

사용법

```
criterion = nn.CrossEntropyLoss()
loss = criterion(logits, targets)
# logits: (batch, classes), targets: (batch,)
```

사용자 정의 손실

```
def focal_loss(pred, target, gamma=2.0):
    ce = nn.functional.cross_entropy(
        pred, target, reduction="none")
    pt = torch.exp(-ce)
    return ((1 - pt) ** gamma * ce).mean()
```

저장 및 로딩

상태 디렉터리 저장/로드 (권장)

```
torch.save(model.state_dict(), "model.pt")
model = Net()
model.load_state_dict(
    torch.load("model.pt", weights_only=True))
```

전체 체크포인트 저장

```
torch.save({
    "epoch": epoch,
    "model": model.state_dict(),
    "optimizer": opt.state_dict(),
    "loss": loss}, "checkpoint.pt")
```

체크포인트 로드

```
ckpt = torch.load("checkpoint.pt",
                  weights_only=False)
model.load_state_dict(ckpt["model"])
opt.load_state_dict(ckpt["optimizer"])
```

PyTorch 빠른 참조

GPU

디바이스 관리

```
device = torch.device(
    "cuda" if torch.cuda.is_available()
    else "cpu")
model = model.to(device)
x = x.to(device)
```

GPU 유틸리티

torch.cuda.is_available()	CUDA 사용 가능 여부 확인
torch.cuda.device_count()	GPU 수
torch.cuda.memory_allocated()	현재 GPU 메모리 사용량 (바이트)
torch.cuda.empty_cache()	미사용 캐시 메모리 해제

다중 GPU

```
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(model)
model = model.to(device)
```

일반 패턴

가중치 초기화

```
def init_weights(m):
    if isinstance(m, nn.Linear):
        nn.init.xavier_uniform_(m.weight)
        m.bias.data.fill_(0.01)
model.apply(init_weights)
```

그래디언트 클리핑

```
torch.nn.utils.clip_grad_norm_(
    model.parameters(), max_norm=1.0)
```

레이어 동결

```
for param in model.fc1.parameters():
    param.requires_grad = False
```

모델 요약

```
total = sum(p.numel()
            for p in model.parameters())
trainable = sum(p.numel()
                for p in model.parameters()
                if p.requires_grad)
```