

Tham Khảo Nhanh PyTorch

Tensor, autograd, mạng nơ-ron và vòng lặp huấn luyện

Tensor

Tạo Tensor

```
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
```

Hàm Khởi Tạo Tensor

torch.zeros(m, n)	Toàn số 0, kích thước (m, n)
torch.ones(m, n)	Toàn số 1, kích thước (m, n)
torch.randn(m, n)	Phân phối chuẩn ngẫu nhiên
torch.arange(start, end, step)	Các giá trị cách đều nhau
torch.linspace(start, end, steps)	Số lượng điểm cố định
torch.eye(n)	Ma trận đơn vị
torch.empty(m, n)	Bộ nhớ chưa khởi tạo

Tương Tác với NumPy

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

Autograd

Theo Dõi Gradient

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

Tắt Theo Dõi Gradient

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

Kiểm Soát Gradient

x.requires_grad_(True)	Bật theo dõi gradient tại chỗ
x.grad.zero_()	Đặt lại gradient tích lũy
x.detach()	Tensor mới không có lịch sử gradient
x.grad	Truy cập gradient đã lưu

Mạng Nơ-ron

Định Nghĩa Model

```
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)
```

Model Tuần Tự

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

Các Layer Phổ Biến

nn.Linear(in, out)	Layer kết nối đầy đủ
nn.Conv2d(c_in, c_out, k)	Tích chập 2D, kích thước kernel k
nn.BatchNorm2d(n)	Chuẩn hóa theo batch
nn.LSTM(in, hidden)	Layer hồi quy LSTM
nn.Dropout(p)	Dropout với xác suất p
nn.Embedding(vocab, dim)	Bảng tra cứu embedding

Nạp Dữ Liệu

Dataset Tùy Chỉnh

```
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)
```

Dataset Có Sẵn

```
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)
```

Vòng Lặp Huấn Luyện

Vòng Lặp Huấn Luyện Chuẩn

```
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()
```

Đánh Giá

```
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()
```

Danh Sách Kiểm Tra Huấn Luyện

model.train()	Bật dropout / batch norm huấn luyện
model.eval()	Chuyển sang chế độ suy luận
optimizer.zero_grad()	Xóa gradient trước backward
loss.backward()	Tính toán gradient
optimizer.step()	Cập nhật tham số

Optimizer

Optimizer Phổ Biến

```
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)
```

Scheduler Tốc Độ Học

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

So Sánh Optimizer

SGD	Đơn giản, cần chỉnh tham số, tốt với momentum
Adam	LR thích nghi, hội tụ nhanh, mặc định phổ biến
AdamW	Adam với weight decay tách biệt
RMSprop	Thích nghi, tốt cho RNN

Hàm Mất Mát

Hàm Mất Mát Phổ Biến

nn.CrossEntropyLoss()	Phân loại (logits, không cần softmax)
nn.BCEWithLogitsLoss()	Phân loại nhị phân (logits)
nn.MSELoss()	Hồi quy (sai số bình phương trung bình)
nn.L1Loss()	Hồi quy (sai số tuyệt đối trung bình)
nn.NLLLoss()	Log-likelihood âm (sau log_softmax)
nn.HuberLoss()	Hồi quy bền vững (ít nhạy với ngoại lệ)

Cách Dùng

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

Hàm Mất Mát Tùy Chỉnh

```
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()
```

Lưu & Tải

Lưu / Tải State Dict (Khuyến Nghị)

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

Lưu Checkpoint Đầy Đủ

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

Tải Checkpoint

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

GPU

Quản Lý Thiết Bị

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

Tham Khảo Nhanh PyTorch

Tiện Ích GPU

<code>torch.cuda.is_available()</code>	Kiểm tra CUDA có sẵn không
<code>torch.cuda.device_count()</code>	Số lượng GPU
<code>torch.cuda.memory_allocated()</code>	Bộ nhớ GPU đang dùng (bytes)
<code>torch.cuda.empty_cache()</code>	Giải phóng bộ nhớ cache chưa dùng

Multi-GPU

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

Mẫu Phổ Biến

Khởi Tạo Trọng Số

```
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)
```

Cắt Gradient

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

Đóng Băng Layer

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

Tóm Tắt Model

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