Referensi Cepat PyTorch

Tensor, autograd, neural network, dan training

Tensor

Membuat Tensor

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

Konstruktor Tensor

`torch.zeros(m, n)`	Semua nol, shape (m, n)
`torch.ones(m, n)`	Semua satu, shape (m, n)
`torch.randn(m, n)`	Acak distribusi normal standar
`torch.arange(start, end, step)`	Nilai berspasi merata
`torch.linspace(start, end, steps)`	Jumlah titik tetap
`torch.eye(n)`	Matriks identitas
`torch.empty(m, n)`	Memori tidak diinisialisasi

Interop NumPy

```
t = torch.from_numpy(np_array)
arr = tensor.numpy()  # berbagi memori
t = torch.as_tensor(np_array)
```

Autograd

Melacak Gradien

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

Menonaktifkan Pelacakan Gradien

```
with torch.no_grad():
    pred = model(x)  # hanya inferensi
x_det = x.detach()  # lepaskan dari graph
```

Kontrol Gradien

`x.requires_grad_(True)`	Aktifkan pelacakan gradien di tempat
`x.grad.zero_()`	Reset gradien yang terakumulasi
`x.detach()`	Tensor baru tanpa riwayat gradien
`x.grad`	Akses gradien yang tersimpan

Neural Network

Definisikan 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 Sequential

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

Layer Umum

`nn.Linear(in, out)`	Layer fully connected
`nn.Conv2d(c_in, c_out, k)`	Konvolusi 2D, kernel size k
`nn.BatchNorm2d(n)`	Batch normalization
`nn.LSTM(in, hidden)`	Layer recurrent LSTM
`nn.Dropout(p)`	Dropout dengan probabilitas p
`nn.Embedding(vocab, dim)`	Tabel pencarian embedding

Memuat Data

Dataset Kustom

```
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 Bawaan

```
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)
```

Loop Training

Loop Training Standar

```
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()
```

Evaluasi

```
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()
```

Checklist Training

`model.train()`	Aktifkan dropout / batch norm training
`model.eval()`	Beralih ke mode inferensi
`optimizer.zero_grad()`	Bersihkan gradien sebelum backward
`loss.backward()`	Hitung gradien
`optimizer.step()`	Perbarui parameter

Optimizer

Optimizer Umum

```
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)
```

Learning Rate Scheduler

```
sched = optim.lr_scheduler.StepLR(
    opt, step_size=10, gamma=0.1)
# dalam loop: sched.step() setelah tiap epoch
```

Perbandingan Optimizer

SGD	Sederhana, perlu tuning, bagus dengan momentum
Adam	LR adaptif, konvergensi cepat, default
AdamW	Adam dengan weight decay terpisah
RMSprop	Adaptif, bagus untuk RNN

Fungsi Loss

Fungsi Loss Umum

`nn.CrossEntropyLoss()`	Klasifikasi (logits, tanpa softmax)
`nn.BCEWithLogitsLoss()`	Klasifikasi biner (logits)
`nn.MSELoss()`	Regresi (mean squared error)
`nn.L1Loss()`	Regresi (mean absolute error)
`nn.NLLLoss()`	Negative log-likelihood (setelah log_softmax)
`nn.HuberLoss()`	Regresi robust (kurang sensitif outlier)

Penggunaan

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

Loss Kustom

```
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()
```

Simpan & Muat

Simpan / Muat State Dict (Disarankan)

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

Simpan Checkpoint Lengkap

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

Muat Checkpoint

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

GPU

Manajemen Device

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

Referensi Cepat PyTorch

Utilitas GPU

`torch.cuda.is_available()`	Periksa apakah CUDA tersedia
`torch.cuda.device_count()`	Jumlah GPU
`torch.cuda.memory_allocated()`	Penggunaan memori GPU saat ini (byte)
`torch.cuda.empty_cache()`	Bebaskan memori cache yang tidak digunakan

Multi-GPU

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

Pola Umum

Inisialisasi Bobot

```
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)
```

Gradient Clipping

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

Bekukan Layer

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

Ringkasan Model

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