PyTorch

Framework para DL

Pesquisa e Indústria

Destaques: tensor, GPU, autograd

PyTorch

Ecossistema

torchvision (imagens)

torchrec (recomendações)

torchao (eficiência)

torchft (fine-tuning)

executorch (edge)

PyTorch/XLA (TPUs)

```
device = torch.device("cuda")
```

selecionar o dispositivo
para processamento

```
model = nn.Sequential(
    nn.Linear(n_in, n_hidden),
    nn.ReLU(),
    nn.Dropout(dropout),
    nn.Linear(n_hidden, n_out)
)
```

Especificação da Arquitetura da Rede Neural

Backpropagation

```
optimizer.zero_grad()      # limpa gradientes anteriores
loss.backward()            # autograd calcula derivadas
optimizer.step()           # aplica atualização nos pesos
```

```python
# Função de treino minimalista
def train(X, y, n_hidden=8, lr=0.1, epochs=100, verbose=False):
    # Tensores
    X_t = torch.tensor(X, dtype=torch.float32, device=device)
    y_t = torch.tensor(y.reshape(-1,1), dtype=torch.float32, device=device)

    model = make_mlp(X_t.shape[1], n_hidden, n_out=1)

    criterion = nn.BCEWithLogitsLoss()             # Loss binária
    optimizer = torch.optim.SGD(model.parameters(), lr=lr)  # SGD puro

    losses = []
    for ep in range(epochs):
        optimizer.zero_grad()
        logits = model(X_t)
        loss = criterion(logits, y_t)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())

        if verbose and (ep % max(1, epochs//10) == 0):
            print(f"época {ep:4d} | loss={loss.item():.6f}")

    return model, losses
```

Função de treinamento

```python
# MLP simples em PyTorch (1 camada oculta)
def make_mlp(n_in, n_hidden, n_out=1, dropout=0.0):
    return nn.Sequential(
        nn.Linear(n_in, n_hidden),
        nn.ReLU(),
        nn.Dropout(dropout),
        nn.Linear(n_hidden, n_out)  # saída = logit (sem sigmoid aqui)
    ).to(device)
```