PyTorch Cheatsheet

Tensors and Basic Operations

PyTorch is an open-source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing.

- Import: `import torch`
- Creation:
  - `torch.tensor([1, 2, 3])`: From list.
  - `torch.zeros(3, 2)`, `torch.ones(2, 2)`.
  - `torch.rand(3, 3)`: Random values in [0, 1).
  - `torch.randn(3, 3)`: Standard normal distribution.
- Device Management (GPU):
  - `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`
  - `tensor = tensor.to(device)`: Move tensor to GPU/CPU.

Deep Learning Workflow

Model Definition (nn.Module)

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(10, 50)
        self.relu = nn.ReLU()
        self.layer2 = nn.Linear(50, 1)

    def forward(self, x):
        x = self.layer1(x)
        x = self.relu(x)
        x = self.layer2(x)
        return x

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

Loss Functions and Optimizers

```
import torch.optim as optim

criterion = nn.MSELoss() # Regression
criterion = nn.CrossEntropyLoss() # Classification

optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Training Loop

```
for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad() # Reset gradients

    outputs = model(inputs)
    loss = criterion(outputs, labels)

    loss.backward() # Compute gradients
    optimizer.step() # Update weights
```

Data Handling (Dataset & DataLoader)

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, x, y):
        self.x = torch.tensor(x, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.float32)
    def __len__(self): return len(self.x)
    def __getitem__(self, idx): return self.x[idx], self.y[idx]

loader = DataLoader(dataset, batch_size=32, shuffle=True)
```

Evaluation and Saving

- Evaluation Mode: `model.eval()`.
- Disable Gradients: `with torch.no_grad(): ...` (saves memory during inference).
- Save/Load:
  - `torch.save(model.state_dict(), "model.pth")`
  - `model.load_state_dict(torch.load("model.pth"))`

Common Layers (torch.nn)

- Linear: `nn.Linear(in, out)`.
- Convolutional: `nn.Conv2d(in_ch, out_ch, kernel)`.
- Pooling: `nn.MaxPool2d(2)`.
- Activation: `nn.ReLU()`, `nn.Sigmoid()`, `nn.Tanh()`, `nn.Softmax(dim=1)`.
- Dropout: `nn.Dropout(p=0.5)`.
- Batch Normalization: `nn.BatchNorm2d(num_features)`.

Pro Tips

Autograd

PyTorch automatically tracks operations on tensors with `requires_grad=True` to compute gradients for backpropagation.

JIT (Just-In-Time) Compilation

Convert models to TorchScript for production deployment outside Python.

- `traced_model = torch.jit.trace(model, example_input)`
- `script_model = torch.jit.script(model)`

Profiling

Use `torch.profiler` to identify bottlenecks in your training and inference code.