PyTorch Learning Guide

# Objective

The objective of this guide is to build a strong foundation in PyTorch, starting from beginner-level tensor operations and progressing to advanced topics like model architectures and optimization techniques. By the end, you will be able to implement deep learning models and understand their internals.

# Introduction to PyTorch

PyTorch is an open-source deep learning framework that provides flexibility and ease of use. Compared to TensorFlow, PyTorch emphasizes dynamic computation graphs, making debugging and experimentation easier.  
  
How to Learn: Read the PyTorch 'Get Started' guide and explore introductory blogs or videos.  
How to Do It: Install PyTorch and run a simple script.

import torch  
x = torch.rand(3, 3)  
print(x)

# Tensors and Basic Operations

Tensors are the core data structure in PyTorch. They are similar to NumPy arrays but support GPU acceleration.  
  
Key concepts: tensor attributes (shape, dtype, device), arithmetic, indexing, reshaping, and broadcasting.

x = torch.tensor([[1, 2], [3, 4]])  
y = torch.tensor([[5, 6], [7, 8]])  
print(x + y)  
print(x.view(4))

# Autograd and Computational Graphs

PyTorch provides automatic differentiation using autograd. When requires\_grad=True, all operations on that tensor are tracked.  
  
You can compute gradients using backward().

x = torch.tensor(2.0, requires\_grad=True)  
y = x\*\*2  
z = 3\*y  
z.backward()  
print(x.grad)

# Neural Network Basics

The nn module helps build neural networks. Key components include layers, activation functions, and loss functions.  
  
Example: Multi-Layer Perceptron (MLP).

import torch.nn as nn  
  
class MLP(nn.Module):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.fc1 = nn.Linear(10, 20)  
 self.relu = nn.ReLU()  
 self.fc2 = nn.Linear(20, 1)  
 def forward(self, x):  
 return self.fc2(self.relu(self.fc1(x)))

# Datasets and DataLoaders

PyTorch provides Dataset and DataLoader for handling data efficiently. torchvision provides ready-to-use datasets like MNIST.  
  
Custom datasets can also be created.

from torch.utils.data import DataLoader  
from torchvision import datasets, transforms  
  
train\_data = datasets.MNIST(root='./data', train=True, download=True, transform=transforms.ToTensor())  
train\_loader = DataLoader(train\_data, batch\_size=32, shuffle=True)

# Training Loops and Optimization

Training loops involve forward pass, loss computation, backward pass, and optimizer step.  
  
Popular optimizers include SGD and Adam.

optimizer = torch.optim.SGD(model.parameters(), lr=0.01)  
for data, target in train\_loader:  
 optimizer.zero\_grad()  
 output = model(data)  
 loss = criterion(output, target)  
 loss.backward()  
 optimizer.step()

# Convolutional Neural Networks (CNNs)

CNNs are widely used for image tasks. They use convolutional and pooling layers.  
  
Example: LeNet or ResNet for classification.

class SimpleCNN(nn.Module):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.conv1 = nn.Conv2d(1, 32, 3, 1)  
 self.fc1 = nn.Linear(5408, 10)  
 def forward(self, x):  
 x = self.conv1(x)  
 x = nn.functional.relu(x)  
 x = nn.Flatten()(x)  
 return self.fc1(x)

# Recurrent Neural Networks (RNNs)

RNNs handle sequential data. Variants include LSTMs and GRUs.  
  
Applications: text generation, time series prediction.

rnn = nn.RNN(input\_size=10, hidden\_size=20, num\_layers=2, batch\_first=True)  
input\_seq = torch.randn(5, 3, 10)  
output, hidden = rnn(input\_seq)

# Advanced Architectures

Advanced models include Transformers, GANs, and VAEs. Transfer learning allows fine-tuning pre-trained models like ResNet.

from torchvision import models  
model = models.resnet18(pretrained=True)  
for param in model.parameters():  
 param.requires\_grad = False  
model.fc = nn.Linear(model.fc.in\_features, 10)

# Deployment and Optimization

Deployment options include TorchScript, ONNX, quantization, and distributed training.  
  
You can save and load models easily.

torch.save(model.state\_dict(), 'model.pth')  
model.load\_state\_dict(torch.load('model.pth'))

# Theory of Deep Learning

Important concepts: backpropagation, gradient descent, vanishing/exploding gradients, and evaluation metrics like accuracy, precision, and recall.  
  
These theories form the backbone of deep learning and connect directly to PyTorch implementations.