Sujay Kumar

PyTorch vs JAX

Deep Learning Frameworks

- High-level APIs for model construction, training procedures, data processing and model exports
- Optimized n-dim matrix operations
- Automatic Differentiation
- Hardware Acceleration: GPU/TPU
- Support our favorite programming languages
- Tensorflow, PyTorch, JAX, Caffe, PaddlePaddle, MATLAB

What’s the right framework for me?

- Level of abstraction provided and extendability
- Performance and Efficiency
- Distributed Training: Large Models
- Rapid prototyping and experimentation
- Community and Developer Support

Pytorch vs JAX

PyTorch

- Open-source training framework by Meta
- Dynamic computation graphs
- Focused on ease-of-use and rapid prototyping
- Constructs the autograd graph during runtime

- Open-source training framework by Google
- Functional programming style
- Focused on high-performance numerical computing
- Uses XLA compiler to compile the execution graph beforehand
- JIT compilation

Abstraction: Neural Networks

PyTorch

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

```
class MLP(nn.Module):
  out_dims: int

  @nn.compact
  def __call__(self, x):
    x = x.reshape((x.shape[0], -1))
    x = nn.Dense(128)(x)
    x = nn.relu(x)
    x = nn.Dense(self.out_dims)(x)
    return x

model = MLP(out_dims=10)

x = jnp.empty((4, 28, 28, 1))
variables = model.init(random.key(42), x)
y = model.apply(variables, x)
```

Abstraction: Numpy Ops

```
In [1]: import torch

In [2]: x = torch.linspace(-100, 100, 50)

In [3]: y = torch.sin(x)

In [4]: z = torch.square(y).sum()
```

```
In [1]: import jax.numpy as jnp

In [2]: x = jnp.linspace(-100, 100, 50)

In [3]: y = jnp.sin(x)

In [4]: z = jnp.square(y).sum()
```

Abstraction: Custom CUDA

- Supports writing custom CUDA kernels through CUDA programming model in C++
- Triton: Array-based programming model for GPU kernels in Python

- Supports writing custom CUDA kernels through CUDA programming model in C++
- Pallas: Triton-like that allows writing GPU kernels for both GPU and TPU

Abstraction: tl;dr

- Supports abstractions at all levels
- Open-source libraries:
  - TranformerEngine
  - PyTorch Lightning
  - Triton

- Supports abstractions at all levels
- Open-source libraries
  - TranformerEngine
  - Flax, Haiku
  - Optax
  - Pallas

Performance and Efficiency

- Dynamic computational graphs
- Dynamic CUDA kernel benchmarking
- Flexibility to change graphs during iteration
- torch.compile()
  - Fuse ops
  - CUDAGraph
  - Automatic Triton Kernel codegen

- JIT (Just-In-Time) compilation
- Changing graphs during execution triggers recompilation
- Fuse ops
- CUDAGraphs
- Better memory management
- Low CPU overhead

Performance and Efficiency: tl;dr

- Using open-source training frameworks: efficiency already baked-in
- Building your own training scripts: manually improve efficiency

- Building your own training scripts: provides pretty-good out-of-the-box efficiency and performance
- Need more tweaks? Need to understand OpenXLA

Distributed Training: Large Models

- Model fits on a single GPU?
  - DistributedDataParallel
- Model is in billions
  - FullyShardedDataParallel
- Model is in trillions
  - Tensor Parallel
  - Pipeline Parallel
  - Mixture-of-Experts

- Model fits on a single GPU?
  - DeviceMesh
- Model is in billions
  - DeviceMesh
- Model is in trillions
  - DeviceMesh

Distributed Training: tl;dr

- Constrained in types of parallelism
- Need to manually handle sharding and parallelism

- Can shard the model and data across GPUs/TPUs as you see fit
- Provides most flexibility for training large models
- All sharding, communication and compute handled by OpenXLA compiler

Rapid Prototyping/Experimentation

PyTorch

- Imperative
- Dynamic Graph Computation
- Can put breakpoint anywhere
- Comfortable with Python? You'll feel comfortable with Pytorch

- Functional programming
- Immutability and Pure Functions
- Hard to debug with breakpoints
- Even though there's a Python frontend, is not compatible with python programming philosophy

Community and Developer Support

- Large open-source community
- Driven by Meta for GPU
- Supported by large enterprises such as Nvidia, Microsoft, OpenAI

- Nascent but growing open-source community
- Driven by Google and mainly focusing on TPUs
- Supported by Nvidia

Tl;dr

- Working with big models and require non-standard distributed training techniques? Use JAX
- Would prefer functional programming to reduce bugs and get improved performance? Use JAX
- Primary workloads running on TPUs? Use JAX
- Prefer flexibility and rapid prototypes? Use PyTorch
- Prefer experimenting with the latest research results from academia and industry? Use PyTorch