Introduction to Jax

Lecture 18 for 14-763/18-763
Guannan Qu

Nov 6, 2024

Why Jax?

TensorFlow v1 was the early dominant player but difficult to use

TensorFlow v2 was released to improve ease-of-use but many compatibility issues

Jax was released

2015

2017

2019

2019-2021

2022

PyTorch was released popular due to ease of use

TensorFlow was gradually surpassed by PyTorch, especially in research

Why Jax?

(My guess on why google developed Jax)
Google wanted to remain a relevant player in the basic tools for AI research

- Didn't want to be constrained by the legacy TensorFlow framework
- Didn't want to create a product that completely mirrored PyTorch

Therefore, Jax is neither TensorFlow nor PyTorch…

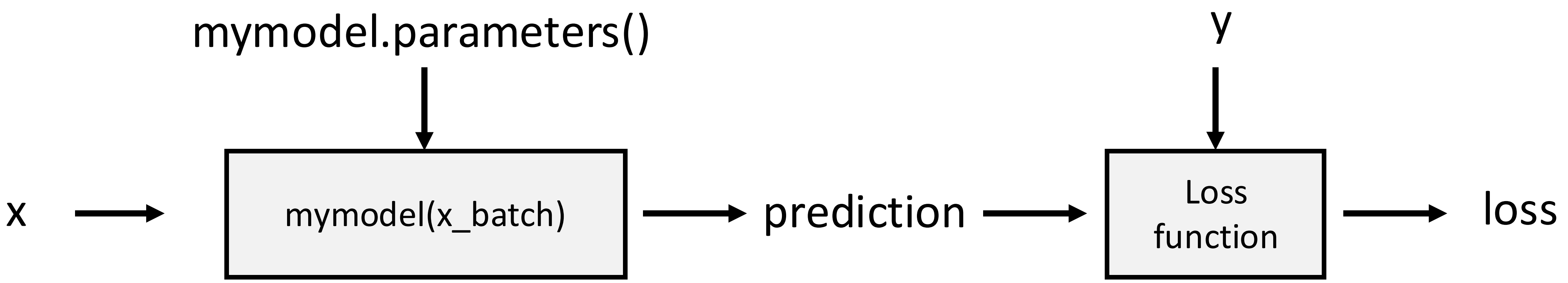
Jax is more like a numerical computing framework (numpy-like) that is tailored to ML

- Usage of Jax very similar to numpy

- Jax supports autograd and parallel computing, which one would argue is essentially the core functionality an ML framework needs

What is Jax?

One would argue PyTorch is essentially a bunch of algebra computations on Tensors, most of which numpy can do (maybe less conveniently)



There are two key things that PyTorch can do but numpy cannot

- Backward (autograd, i.e. automatic differentiation)
- Parallel computing on GPUs

Jax is numpy-like library that supports autograd and parallel computing

…and other functionalities to accelerate development and computing

Basics of Jax: jax.numpy

```python
import jax.numpy as jnp
import jax
```

```python
x = jnp.array([1.,2.,3.])
y = jnp.ones((3))
print(f"x = {x}, y = {y},  dotproduct = {jnp.dot(x,y)}")
```

jax.numpy is modeled after numpy and has very similar usages!

Features of Jax

- jit: Just-in-time compilation, enables very fast and parallelized numerical computation over many types of hardware
- automatic-differentiation: enables gradient computation

Compile 101

xxx.c

```
// C source code
int x = 0;
…
```

Compile

xxx.exe

```
binary machine code
0011001001010100…
```

Higher-level code (e.g., Python, C), is ***compiled*** into lower-level code

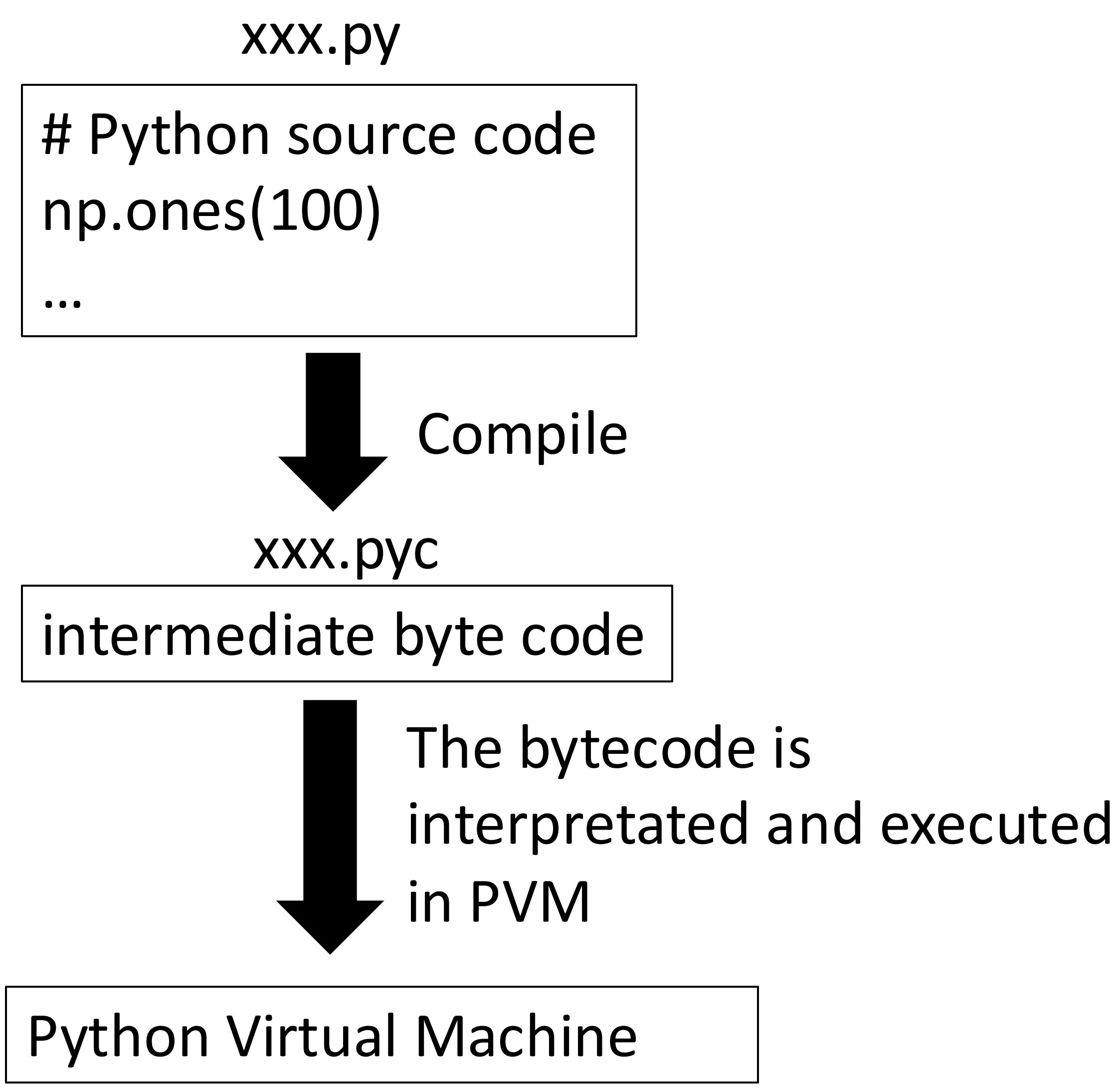
- high-level = human readable
- low-level = not readable, close to hardware

Hardware (x86/64 vs ARM) and OS dependent
Can directly run on your hardware CPU

Compile 101

Higher-level code (e.g., Python, C), is ***compiled*** into lower-level code

- high-level = human readable
- low-level = not readable, close to hardware



Compile 101

Python is slow as the compiled code is NOT directly machine executable. (not as low-level as C)

The Python virtual machine creates overhead

xxx.c

```
// C source code
int x = 0;
…
```

Compile

xxx.exe

binary machine code
0011001001010100…

C is fast as the compiled code is machine executable

xxx.py

```
# Python source code
np.ones(100)
…
```

Compile

xxx.pyc

intermediate byte code

Python Virtual Machine

Compile 101

- To alleviate the slowness of Python, NumPy uses a separate package BLAS/LAPACK written in Fortran for most of its functions (np.dot, np.multiply, …)
- Despite every single line of NumPy code is fast, the Python program as a whole may still be slow (esp. when there is a for loop).

```
for i in range(100):
        x = np.square(x)
```

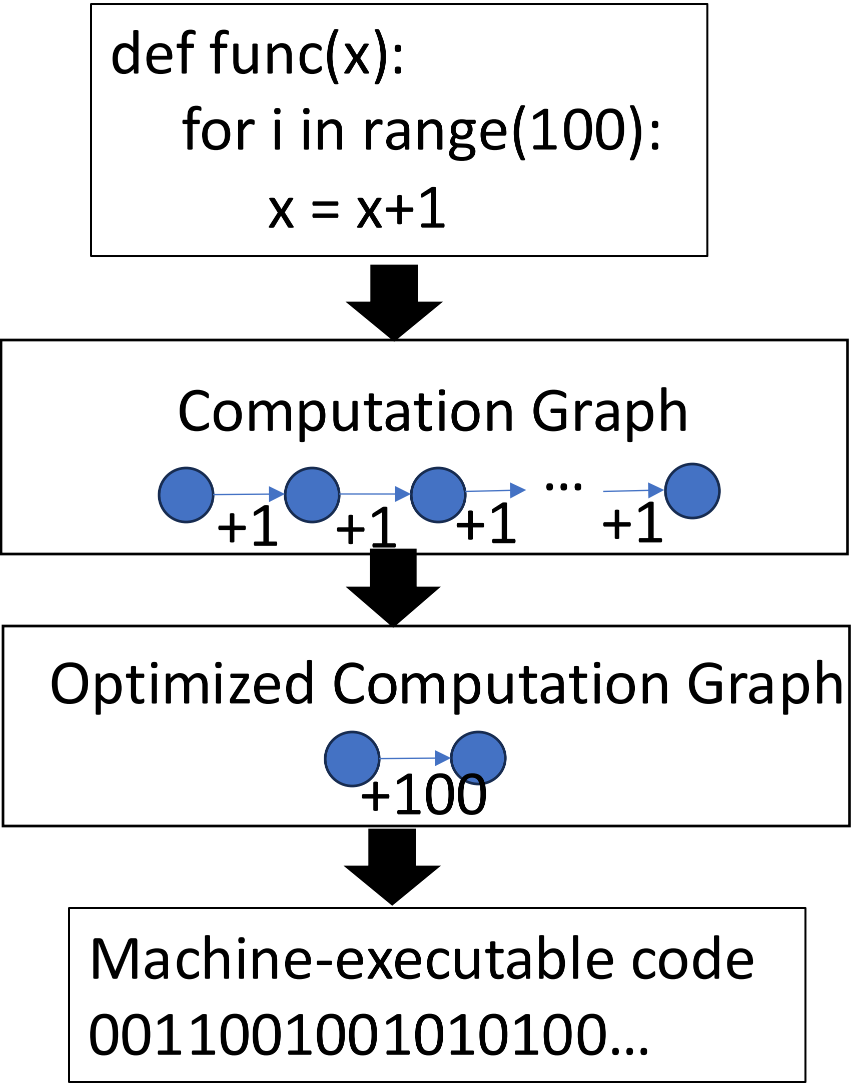
each single line of np code is fast

But overall the code is slow due to the for loop

Just-in-time compilation

- JIT in Jax: Compiling jax.numpy computations (across multiple lines of code) into low-level computation graph optimized and parallelized for specific hardware.
  - Traces multiple lines of jax code and figure out computation graph
  - Hand over computation graph to XLA, which transforms the graph into machine code specialized for your hardware (CPU/GPU), taking advantage of hardware-specific optimizations, such as parallelism

Just-in-time compilation



Jax traces the computation graph during the first time you execute the function

Jax calls XLA* to optimize the computation graph

- Fusion of operations, ignore redundancy
- Device-specific optimizations (parallelize on CPU/GPU/TPU…

… and convert the graph into machine-executable code

*XLA = Accelerated Linear Algebra, another lib developed by Google

Just-in-time compilation

- JIT is called “Just-In-Time” as the compilation happens during execution, not ahead-of-time
- JIT in Jax
  - Can be faster than numpy/torch, especially for complex computations that are repeatedly used
  - Supports many hardware (x86/64 or Apple Arm CPU), GPU (Nvidia/AMD/Apple), Google TPU, more than what PyTorch currently supports

Auto-Grad in Jax

- Auto-Grad in Jax has a very “math” feel and tailored to math/science users
- If you have a function $f(x)$, the gradient is another function $f'(x)$
- e.g. $f(x) = x^2$, then $f'(x) = 2x$

```
def square(x): # a simple square function
    return x*x

grad_square = jax.grad(square) # grad_square

x = jnp.array(3.)
grad_square(3.) # The expected result is 2*x
```

✓ 1.0s

```
Array(6., dtype=float32, weak_type=True)
```

Auto-Grad in Jax

- You can even calculate higher order gradients
- e.g. $f(x) = x^2$, then $f'(x) = 2x$, $f''(x) = 2$

```
def square(x): # a simple square function
    return x*x

grad_square = jax.grad(square) # grad_square

x = jnp.array(3.)
grad_square(3.) # The expected result is 2*x
```

✓ 1.0s

```
Array(6., dtype=float32, weak_type=True)
```

```
second_order_grad_square = jax.grad(grad_square)

second_order_grad_square(x) # this should just ret
```

✓ 0.0s

```
Array(2., dtype=float32, weak_type=True)
```

Example: use Jax to build a neural network

- In PyTorch we used nn.Module to build a neural network
  - nn.linear, nn.relu, …
- Used optimizer.sgd or optimizer.adam for weight updating
- Unfortunately, Jax does not have any of the above

Summary

- Jax is neither tensorflow nor pytorch.
- Jax is numpy + autograd + JIT + other functionalities
- Advantage: Numerical computation-wise, jax is considered faster than numpy/torch thanks to JIT's ability to optimize computing for different hardwares
- Disadvantage:
  - Has all the essential functions for deep learning, but lacks many supporting libraries to build neural networks, manage data, etc…
  - Libraries based on Jax for deep learning are emerging (Google Flax, Trax) but not popular yet

Jax is popular in scientific computing purposes, e.g. for control in robotics.
Not yet popular for deep learning, but in the future, it may.