[544] PyTorch Basics

Tyler Caraza-Harter

Learning Objectives

- deploy JupyterLab with PyTorch inside a Docker container
- compare different numeric types in terms of space requirements, range, and precision
- perform calculations on PyTorch tensors
- formulate models as functions that multiply input data by parameters

Outline

PyTorch Overview

Numeric Types

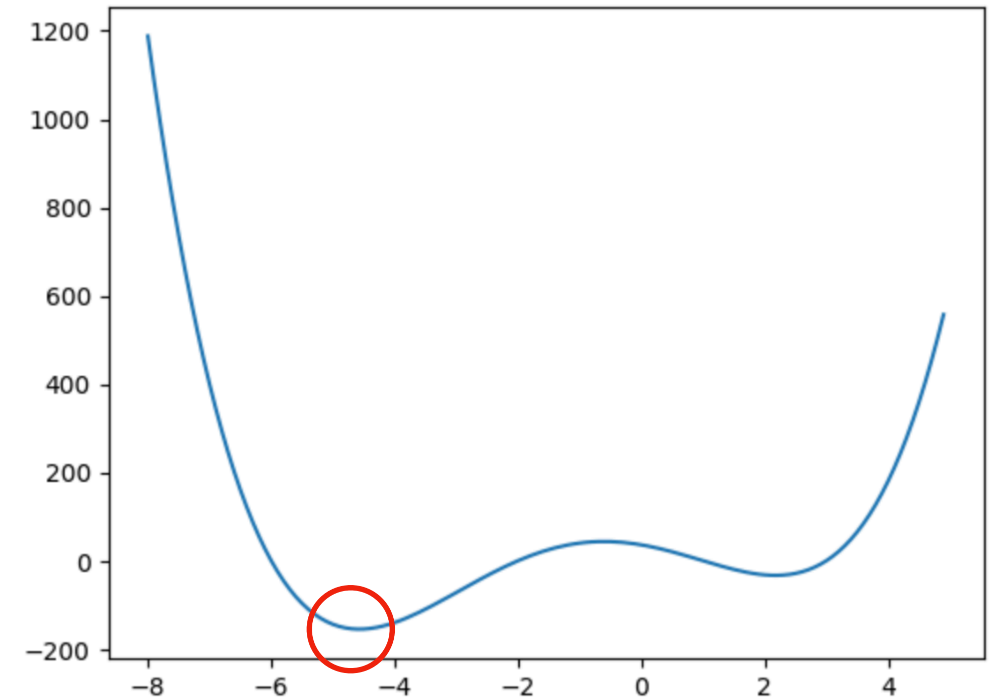
Coding Demos

- numeric types
- calculations: element wise, sigmoid, matrix multiplication, linear models
- optimization
- troubleshooting

PyTorch Uses

1 Floating point operations

- scientific computing, machine learning
- matrices, linear algebra
- seamless: on CPU or GPU
- distributed computing



2 Optimization

- y = f(x)
- which x makes y smallest? (or largest?)

3 Machine learning:

- what parameters yield best performance metrics for some data?
- simple example:
  $y = b * x + c$ what b and c parameters give the best fit?
- deep learning
  $y = sigmoid(sigmoid(data\ @\ matrix1 + bias1)\ @\ matrix2 + bias2)$

Setup

See snippets:

https://tinyurl.com/4pn9db3n

Outline

PyTorch Overview

Numeric Types

Coding Demos

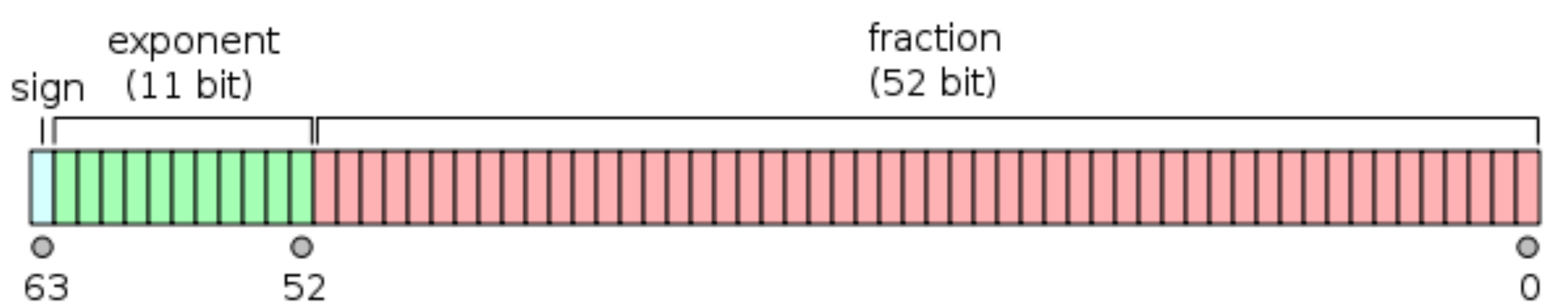
- numeric types
- calculations: element wise, sigmoid, matrix multiplication, linear models
- optimization
- troubleshooting

Python Numeric Types (Built In)

https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Python Types

- ints
  - ➡ no maximum/minimum size (Python is unusual in this way)
  - ➡ bigger/smaller values => more bits necessary
- floats
  - ➡ usually 64 bits ("double precision"; 32 bits would "single precision")
  - ➡ like exponential notation (1.23×10^2), but in binary instead of decimal
  - ➡ min/max size. Inf, -Inf, NaN have special bit combinations



https://en.wikipedia.org/wiki/Double-precision_floating-point_format

- complex
  - ➡ real and imaginary represented as two floats
  - ➡ not covered in 544

Other Numeric Types

Common numeric types that (a) CPUs can directly manipulate and (b) PyTorch supports

- integers: uint8, int8, int16, int32, int64
- floats: float16, float32, float64
- names specify bits, float vs. int, and signed ("u" => unsigned)
- dtype (data type)

```
import torch
x = torch.tensor(3.14, dtype=torch.float16)
```

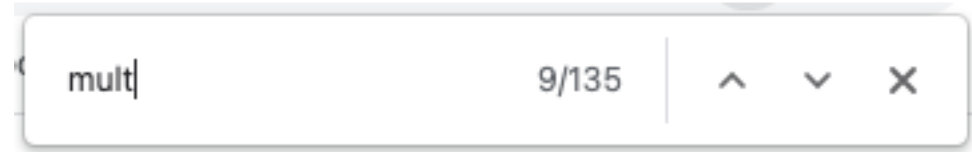
PyTorch float16

Python float

```
print(x.element_size())  # 2 bytes (instead of 8)
```

Tradeoffs: precision, range, memory usage

Hardware Support



...	...
MULPD	Multiply Packed Double Precision Floating-Point Values
MULPS	Multiply Packed Single Precision Floating-Point Values
MULSD	Multiply Scalar Double Precision Floating-Point Value
MULSS	Multiply Scalar Single Precision Floating-Point Values
...	...

https://www.felixcloutier.com/x86/

Hypothetical Scenario: all the ints in your dataset fit nicely in 3 bytes. Should you come up with a new integer byte representation?

Pro: utilize memory more efficiently based on your use case

Con: your CPU won't have instructions for working with this new type. Solutions:

- perform the multiplication in software instead of hardware (slow!)
- keep the data in your 3-byte format, but convert to a regular 4-byte it on an as-needed basis to do calculations (slow!)

Common to have one form for computation, another for storage, messages, etc.

Outline

PyTorch Overview

Numeric Types

Coding Demos

- numeric types
- tensors
- calculations: element wise, sigmoid, matrix multiplication
- linear models