Qiyang Hu

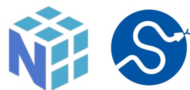
UCLA Office of Advanced Research Computing

Nov 2nd, 2022

About this talk

- The outline
  - Understanding deep learning framework
  - Introduction to PyTorch
  - Four modules in PyTorch (tensor, autograd, optim, nn)
  - Code examples
- My DL talks in this and next quarters
  - Introduction to NN (last Friday)
  - Learning PyTorch (Today)
  - Deep learning, the GBU (Friday)
  - Special NN topics, (conv, gans, transformer, lstm?) (next quarter)

Coding Neural Networks

- From Scratch 

Data Preparation → Neural Network Construction → Training

Neural Network Construction
- Parameter Initialization
- Activation Function
- Forward Pass
- Backward Pass

Training
- Loss Function
- Optimizer
- Iterations (epochs, batches)

Coding Neural Networks

- From PyTorch

Data Preparation → Neural Network Construction → Training

Neural Network Construction
- Parameter Initialization
- Activation Function
- Forward Pass
- Backward Pass

Training
- Loss Function
- Optimizer
- Iterations (epochs, batches)

Qiyang Hu

Coding Neural Networks

- From Keras (TF2.x) and Scikit-Learn

Data Preparation → **Neural Network Construction** (Parameter Initialization, Activation Function, Forward Pass, Backward Pass) → **Training** (Loss Function, Optimizer, Iterations (epochs, batches))

Deep Learning Frameworks

High-level Models

for tasks of Computer Vision (Convnets), NLP (transformer, diffusion models, etc.)

NN Components

Neural Network constructs/layers, activation function, optimizers, loss functions, metrics

Autodiff Engine

Workflows of JVP (forward) and VJP (reverse) to allocate/record the "differential" variables

Multidimensional Array

Data layouts/storages and fundamental matrix-multiplication operations

NN Models

NN Components

Automatic Differentiation

Data Container Primitives

Qiyang Hu

What is PYTORCH

- An open-source Python-based deep learning framework
  - Replacement for Numpy with supporting GPUs, ROCm, TPUs
  - A full set of deep learning libraries
- History
  - Lua-based Torch (2002 - 2011): TH, THC, THNN, THCUNN
  - PyTorch 0.1 (2016): Python-based Torch
  - PyTorch 1.0 (2018): merging Caffe2
  - PyTorch moved to a new, independent PyTorch Foundation (Sept 2022)
  - PyTorch 1.13 (Oct 28, 2022)
- PyTorch as a backend building block
  - Keras-like: PyTorch Lightening, PyTorch Ignite, tensorlayers, fast.ai
  - Advance-models-encapsulated: PyTorch Hubs, HuggingFace
  - For specific domains: FlowTorch, NiftyTorch, Flair, Kornia, Skorch, ELF, Detectron2

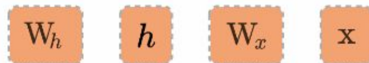
Why PYTORCH

- Simplicity
  - Feels like Numpy
  - Consistent & great APIs
- Flexibility
  - Defining the model
  - Modifying the model
- Dynamic compute graphs
  - Immediate forward execution
  - Tape-based autograd
  - Destroyed immediately after backprop
- Model serialization and quantization
  - JIT, TorchScript, FX
    - Seamlessly switch between Modes, Distributed training, Mobile deployment

A graph is created on the fly

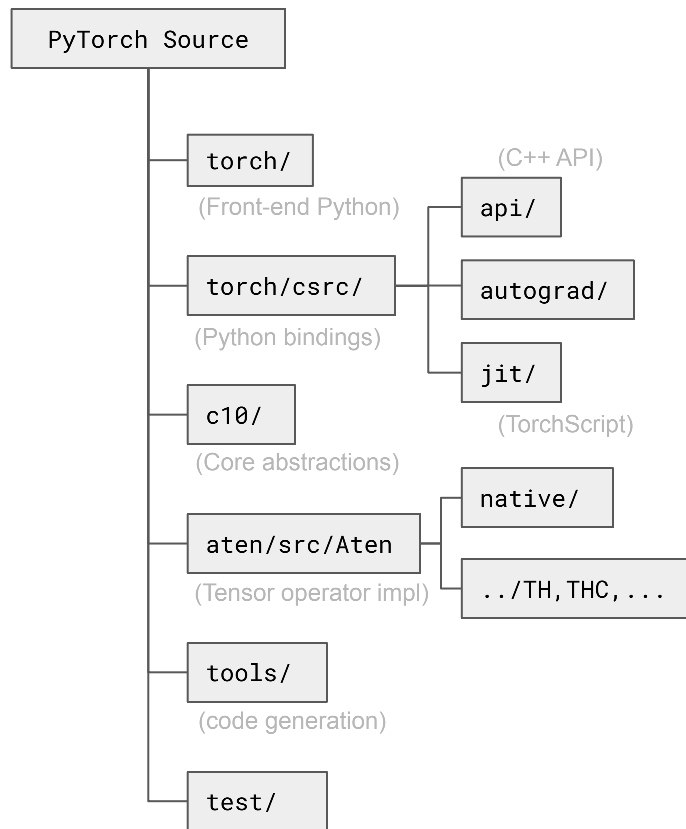
```python
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```



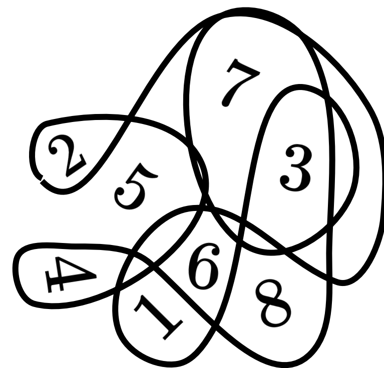
“Py” and “Non-Py” in PyTorch

- PyTorch = Python **+** C/C++ **+** CUDA
  - Python extension objects in C/C++
  - Code base components:
    - The core Torch libraries:
      TH, THC, THNN, THCUNN
    - Vendor libraries:
      CuDNN, NCCL
    - Python Extension libraries
    - Additional 3rd-party libraries:
      NumPy, MKL, LAPACK, DLPack



Tensors as building blocks

$$1 \qquad \begin{bmatrix} 2 \\ 5 \\ 4 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 & 3 \\ 5 & 7 & 9 \\ 4 & 8 & 6 \end{bmatrix} \qquad \begin{bmatrix} 4 & 6 & 9 \\ 1 & 2 & 5 \\ 8 & 7 & 3 \end{bmatrix}$$



Scalar	Vector	Matrix	Tensor	Tensor
0-D	*1*-D	*2*-D	*3*-D	*n*-D
$X = 1$	$X[1] = 5$	$X[2,1] = 8$	$X[0,1,2] = 5$	$X[\underbrace{2,3,\ldots,1}_{N \text{ indices}}] = 6$

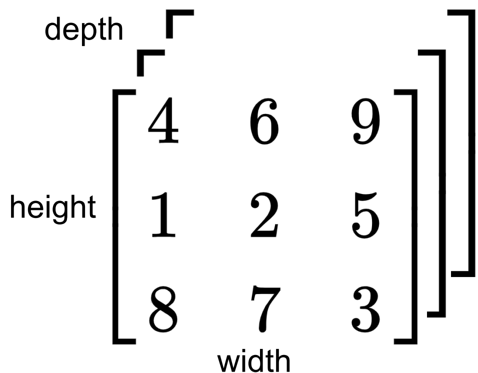
torch.tensor([[[[1.0,1.0],[2.0,2.0]],[[3.0,3.0],[4.0,4.0]]],[[[5.0,5.0],[6.0,6.0]],[[7.0,7.0],[8.0,8.0]]]])

Qiyang Hu

Tensor, Storage and Views

$$M(i,j) = \text{offset} + \text{stride}[0] \cdot i + \text{stride}[1] \cdot j$$

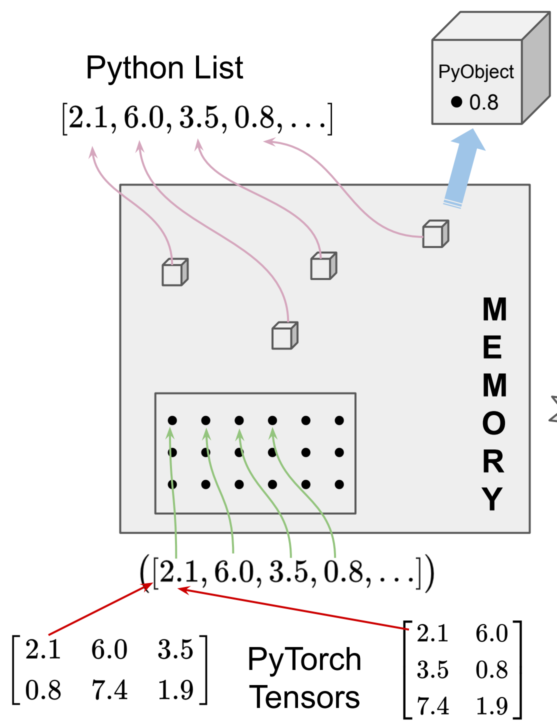
- **Dense Strided Tensors**



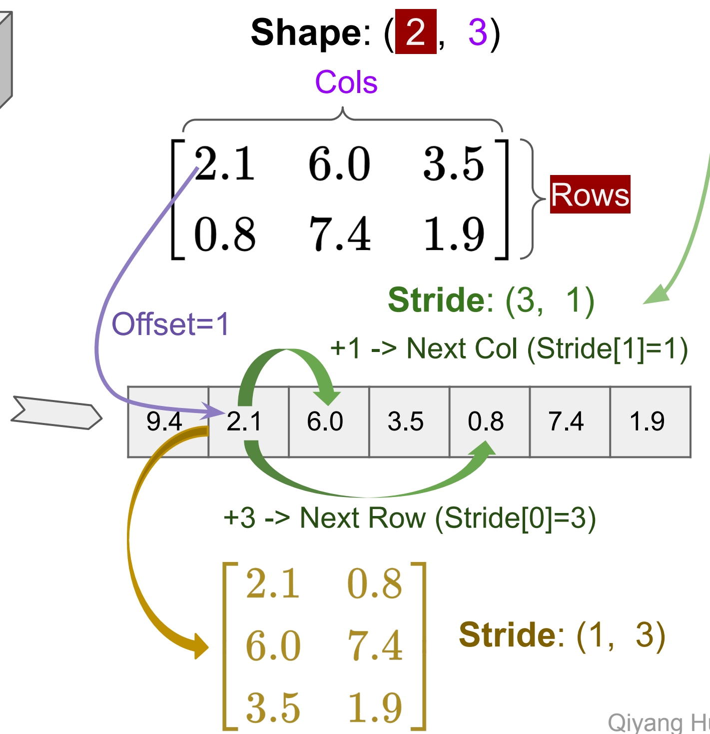
sizes	(D,H,W)
dtype	integer
device	cuda:0
layout	strided
strides	(H*W,W,1)

Other tensor type: sparse, quantized, encrypted, XLA …

- Contiguous & Unboxed



- Size, offset, stride



Qiyang Hu

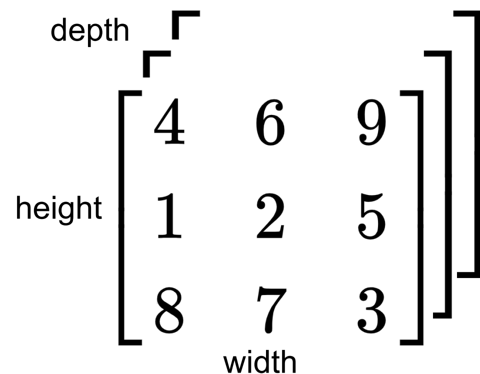
Colab Hands-on

bit.ly/learning_pytorch

Automatic differentiation

- Autograd package
  - Track all operations of tensors
  - Compute derivatives analytically via back-prop
  - Natively loaded in torch module
  - Can be used in other scientific domains
- Simple usage
  - Set tensor’s .requires_grad as TRUE
    - Tensor’s creation function recorded in .grad_fn attribute
    - Gradient accumulated into .grad attribute
  - Call .backward()
- Stop a tensor from tracking history
  - Wrap the code block in with torch.no_grad()
  - .detach()

Tensor and its metadata



sizes	(D,H,W)
dtype	integer
device	cuda:0
layout	strided
strides	(H*W,W,1)
requires_grad	True/False
grad	
grad_fn	

Optimizers in PyTorch

- torch.optim package
  - Provides various optimization algorithms
  - Construct an optimizer object
    - optimizer = optim.SGD(model.parameters(), lr=0.01)
  - Need to move model to GPU before constructing optimizers
  - Must zero the gradient explicitly:
    - optimizer.zero_grad()
  - Take an optimization step:
    - optimizer.step() in GD method
    - optimizer.step(closure) in CG or LBFGS method
  - Optional: adjust the learning rate based on the number of epochs.
    - optimizer.lr_scheduler

Neural Networks in PyTorch

- torch.nn package
  - Contains all building blocks for neural network related work
  - nn.functional and nn.Module
- Define a network
  - For simple networks: concatenate modules through a nn.Sequential container
  - For complex networks: Subclassing nn.Module
- nn.Module package expects first index as batch size of samples
  - Need to reshape the input by .unsqueeze()
  - Use Dataset and DataLoader
- Loss functions in torch.nn:
  - nn.MSELoss (regression), nn.BCELoss (binary classification), nn.CrossEntropyLoss (multiclass classification)

Initialization in PyTorch

- Weight initialization workflow:
  - Determine which layer uses which initialization methods (**`from torch.nn.init`**)
  - After instantizing the model, run the initialization function
- Initialization methods in PyTorch:
  - Constant initialization: constant_, eye_
  - Random initialization: uniform_, normal_
  - Xavier initialization: (xavier_uniform_, xavier_normal_)
  - Kaiming initialization(kaiming_uniform_, kaiming_normal_)
  - Special requirement initialization: orthogonal_, sparse_
- Default initialization in PyTorch:
  - A uniform distribution bounded by 1/sqrt(in_features)
  - May need a customized initialization strategy for specific problem (e.g. training with 2nd-order gradients)

PyTorch

Tensorflow

JAX

Qiyang Hu

OARC Workshop Survey

http://bit.ly/3Wo6AIu