

# Introduction to PyTorch

CS236 Section, Autumn 2024

Honglin Chen



# What is Pytorch?

*A machine learning framework that accelerates the path from research prototyping to production deployment*

Adapted from Pytorch official tutorial:  
[https://www.youtube.com/watch?v=IC0\\_FRiX-sw](https://www.youtube.com/watch?v=IC0_FRiX-sw)

## Machine learning framework

Deep learning primitives such as data loading, NN layer types, activations, loss functions, and optimizers

Hardware acceleration on NVIDIA GPUs

Libraries for vision, NLP, and audio applications

## Research prototyping

Models are Python code, Automatic differentiation, and eager mode

## Production deployment

TorchScript, TorchServe, quantization

# Overview

## Motivations

Python

NumPy

## Building Blocks

Tensors

Operations

Modules

## Examples

MNIST

## Beyond PyTorch

Tools

High Level Libraries

Domain Specific Libraries

# Motivations

Python vs. NumPy

```
X = [1] * 10000
Y = [0.5] * 10000
Z = [None] * 10000
for i in range(10000):
    Z[i] = X[i] * Y[i]
```

```
# 2.772092819213867 ms
# Interpreter Overhead
# 64 bit
```

```
X = np.full((10000,), 1)
Y = np.full((10000,), 0.5)
Z = X * Y
```

```
# 0.08273124694824219 ms
# Low Level Implementation
# Vectorization
```

# Motivations

NumPy vs. PyTorch

```
X = np.full((10000,), 1)
Y = np.full((10000,), 0.5)
Z = X * Y
```

```
# 0.08273124694824219 ms
# Low Level Implementation
# Vectorization
```

```
X = torch.full((10000,), 1).cuda()
Y = torch.full((10000,), 0.5).cuda()
Z = X * Y
```

```
# 0.3185272216796875 ms
# GPU Acceleration
```

```
Z.sum().backward()
dX = X.grad

# Automatic Differentiation
```

# Building Blocks

TENSORS

# Building Blocks

Tensors / Initialization

```
torch.tensor([5., 3.])  
tensor([ 5.,  3.,]) # defaults to  
torch.float32
```

```
torch.from_numpy(np.array([5., 3.]))  
tensor([ 5.,  3.,], dtype=torch.float64) #  
because numpy defaults to 64bit
```

```
torch.tensor([5., 3.]).numpy()  
array([5., 3.], dtype=float32)
```

# Building Blocks

Tensors / Initialization

```
torch.ones(5, 3)
tensor([[1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.],
        [1., 1., 1.]], dtype=torch.float64)
```



# Building Blocks

Tensors / Initialization

```
torch.randn(5, 3)
```

```
tensor([[ 0.2349, -0.0427, -0.5053],  
        [ 0.6455,  0.1199,  0.4239],  
        [ 0.1279,  0.1105,  1.4637],  
        [ 0.4259, -0.0763, -0.9671],  
        [ 0.6856,  0.5047,  0.4250]])
```

# Building Blocks

Tensors / Initialization

```
torch.ones_like(tensor)
Input: tensor([[ 0.2349, -0.0427, -0.5053],
               [ 0.6455,  0.1199,
 0.4239]])
Output: tensor([[1., 1., 1.],
               [1., 1., 1.],
dtype=torch.float64)
```

# Building Blocks

Tensors / Initialization

```
torch.empty(5, 3)
tensor([[ 0.0000e+00,  2.5244e-29,  0.0000e+00],
        [ 2.5244e-29,  1.4569e-19,  2.7517e+12],
        [ 7.5338e+28,  3.0313e+32,  6.3828e+28],
        [ 1.4603e-19,  1.0899e+27,  6.8943e+34],
        [ 1.1835e+22,  7.0976e+22,  1.8515e+28]])
```

```
# The values are not initialized
```

# Building Blocks

Tensors / Indexing & Reshaping

```
torch.tensor([[5., 3.]][0, :]  
tensor([ 5.,  3.,])
```

```
torch.tensor([[5., 3.]])  
dimension size  
torch.tensor([[5., 3.]])  
tensor([ 5.,  3.,])
```

```
torch.tensor([[5., 3.]])  
torch.Size([1, 2])
```

# Building Blocks

Tensors / Broadcasting

```
X = torch.ones((3, 3, 3))
Y = torch.ones((1, 1, 3))
Z = X * Y
Z.size()
```

```
torch.Size([3, 3, 3])
```

```
#
```

```
https://pytorch.org/docs/stable/notes/broadcasting.html
```

# Building Blocks

Tensors / Devices

```
if torch.cuda.is_available():
    device = torch.device("cuda")           # a CUDA device object
    x = torch.ones(2, device=device)        # directly create a tensor on GPU
    y = torch.ones(2).to(device)            # or just use strings
    `x.to("cuda")`
    z = x + y
    print(z)                                # z is on GPU
    print(z.to("cpu", torch.double))        # to('cpu') moves array to CPU

# `x.cuda()` and `x.cpu()` also works
```

# Building Blocks

Operations / Primitives

```
torch.tensor([5., 3.]) + torch.tensor([3., 5.])  
tensor([ 8.,  8.,])
```

```
z = torch.add(x, y)  
torch.add(x, y, out=z)  
y = y.add_(x) # inplace y += x
```

```
torch.tanh(y)  
torch.stack([x, y])
```

# <https://pytorch.org/docs/stable/torch.html>

# Building Blocks

Operations / Functional

```
import torch.nn.functional as F

X = torch.randn((64, 3, 256, 256))
W = torch.randn((8, 3, 3, 3))

out = F.conv2d(X, W, stride=1, padding=1)

# Like SciPy
# https://pytorch.org/docs/stable/nn.functional.html
```



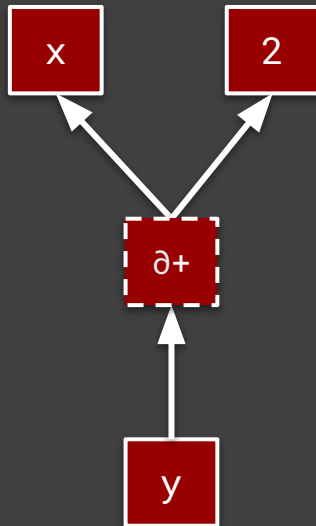
# Building Blocks

Operations / Automatic Differentiation

Computation as a graph built at runtime

```
x = torch.ones(2, 2, requires_grad=True)
tensor([[1., 1.],
        [1., 1.]], requires_grad=True)

y = x + 2
tensor([[3., 3.],
        [3., 3.]], grad_fn=<AddBackward0>)
```



# Building Blocks

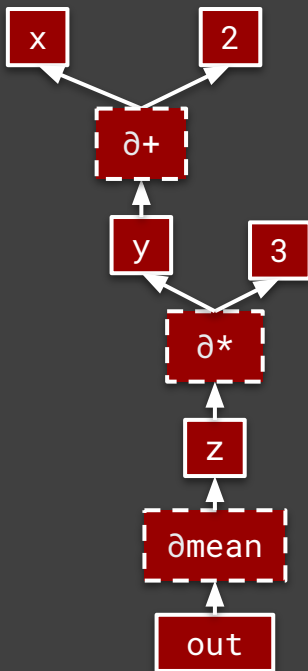
Operations / Automatic Differentiation

```
z = y * 3  
out = z.mean()
```

```
tensor(9., grad_fn=<MeanBackward1>)
```

```
out.backward() # Must be scalar  
print(x.grad) # Only leaf nodes have grad
```

Gradient w.r.t. the input Tensors is computed step-by-step from loss to the top in reverse



# Building Blocks

Operations / Automatic Differentiation

```
x.requires_grad # True
(x ** 2).requires_grad # True

# Keeping track of activations is expensive

with torch.no_grad():
    (x ** 2).requires_grad # False

(x.detach() ** 2).requires_grad # False
```

# Building Blocks

Operations / nn

```
import torch.nn as nn

X = torch.ones((64, 3, 256, 256))

conv = nn.Conv2D(in_channels=3,
                 out_channels=8,
                 kernel_size=3,
                 stride=1,
                 padding=1)

out = conv(img)
```

```
import torch.nn.functional as F

X = torch.randn((64, 3, 256, 256))
W = torch.randn((8, 3, 3, 3))

out = F.conv2d(X, W,
               stride=1, padding=1)

# Inherits from nn.Module
# Implemented using functional
# Stores internal states
```

# Building Blocks

Operations / Module

```
import torch.nn as nn

X = torch.ones((64, 3, 256, 256))

conv = nn.Conv2D(in_channels=3,
                  out_channels=8,
                  kernel_size=3,
                  stride=1,
                  padding=1)
```

```
# Move the module to GPUs
conv.cuda()

# Saves states
conv.state_dict()

# Saves trainable states
conv.parameters()

# Recursively visit child modules
conv.apply(weight_init)
```

# Examples

MNIST

airplane

automobile

bird

cat

deer

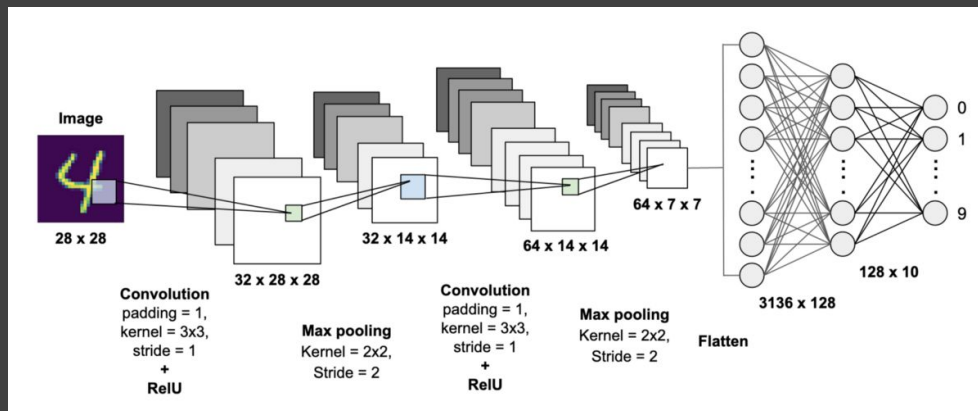
dog

frog

horse

ship

truck



# Example

MNIST

**Preprocessing**

**Dataloader**

**Network**

**Optimizer**

**Training**

# Examples

MNIST / Preprocessing

```
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Convert to Torch Tensor and perform normalization
# https://pytorch.org/vision/stable/transforms.html
# e.x Color Jitter, Five Crops
```



# Examples

MNIST / Dataloader

```
Import torch
import torchvision

trainset = torchvision.datasets.CIFAR10(
    root='./data', train=True,
    download=True, transform=transform)

# Dataloaders are python iterators
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=8,
    shuffle=True, num_workers=2)
```

# Examples

MNIST / Network

```
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

# Examples

MNIST / Network

```
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        ...
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = torch.flatten(self.pool(F.relu(self.conv2(x))))
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

# Examples

MNIST / Optimizer

```
import torch.optim as optim

# Instantiate nn.Module (Use default weights)
net = Net().to("cuda")

# Define loss function
criterion = nn.CrossEntropyLoss()

# Create optimizer: https://pytorch.org/docs/stable/optim.html
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

# Examples

MNIST / Training

```
net.train() # Set to training mode (there is also `net.eval()`)  
  
for epoch in range(2):  
    for inputs, labels in trainloader:  
        # zero the parameter gradients  
        optimizer.zero_grad()  
        # forward + backward + optimize  
        outputs = net(inputs.to("cuda"))  
        loss = criterion(outputs, labels.to("cuda"))  
        loss.backward()  
        optimizer.step()
```


# Examples

MNIST / Recap

```
... transforms.Compose( ... # Define preprocessing transforms
... torch.utils.data.DataLoader( ... # Create Dataloader
... def Net(nn.Module): ... # Define Network
... criterion = nn.CrossEntropyLoss() ... # Define loss function
... optim.SGD(net.parameters(), ... # Create Optimizer
... for x, y in trainloader: ... # Iterate over Dataloader
... outputs = net(inputs) # Forward Pass
... criterion(outputs, labels) ... # Compute Loss
... optimizer.zero_grad() ... # Zero out gradients
... loss.backward() ... # Back Propagate
... optimizer.step() ... # Update weights
```

# Beyond PyTorch

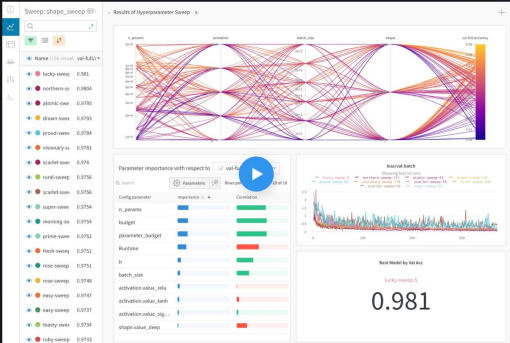
Tools / Keep Track of experiments, artifacts




 **Weights & Biases** Products ▾ Resources ▾ Enterprise Docs Pricing Login Sign Up


## Developer-first MLOps platform

Build better models faster with experiment tracking, dataset versioning, and model management

[SIGN UP](#) [REQUEST DEMO](#)



 DOCS COMMUNITY CODE  




## An open source platform for the machine learning lifecycle


### Latest News

- MLflow 1.20.2 released! (03 Sep 2021)
- MLflow 1.20.1 released! (26 Aug 2021)
- MLflow 1.20.0 released! (26 Aug 2021)
- MLflow 1.19.0 released! (14 Jul 2021)


[News Archive](#)




WORKS WITH ANY ML LIBRARY, LANGUAGE & EXISTING CODE



RUNS THE SAME WAY IN ANY CLOUD



DESIGNED TO SCALE FROM 1 USER TO LARGE ORGS



SCALES TO BIG DATA WITH APACHE SPARK™

# Beyond PyTorch

High Level Libraries / Distributed & Mixed Precision Training

## LIGHTNING CODE

```
# Train
model = LitAutoEncoder()
trainer = pl.Trainer(tpu_cores=8)
trainer.fit(model, mnist_train, mnist_val)
```

GPU available: True, used: False  
TPU available: True, using: 8 TPU cores  
training on 8 TPU cores

Epoch 2:

## LIGHTNING CODE

```
# model
class LitAutoEncoder(pl.LightningModule):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(n.Linear(28 * 28, 64), nn.ReLU(), nn.Linear(64, 3))
        self.decoder = nn.Sequential(nn.Linear(28 * 28, 64), nn.ReLU(), nn.Linear(64, 28 * 28))

    def forward(self, x):
        embedding = self.encoder(x)
        return embedding

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters, lr=1e-3)
        return optimizer

    def training_step(self, train_batch, batch_idx):
        x, y = train_batch
        x = x.view(x.size(0), -1)
        z = self.encoder(x)
        x_hat = self.decoder(z)
        loss = F.mse_loss(x_hat, x)
        self.log('train_loss', loss)
        return loss

    def validation_step(self, val_batch, batch_idx):
        x, y = val_batch
        x = x.view(x.size(0), -1)
        z = self.encoder(x)
        x_hat = self.decoder(z)
        loss = F.mse_loss(x_hat, x)
        self.log('val_loss', loss)

    def backward(self, trainer, loss, optimizer, optimizer_idx):
        loss.backward()
```



# Beyond PyTorch

Domain Specific Libraries / Graph, RL, Probabilistic Programming



NEWS

DOCS

**PyG is the ultimate library  
for Graph Neural Networks**

Build graph learning pipelines with ease.

JOIN SLACK



Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

[View documentation >](#)

[View on GitHub >](#)

**PYRO**

Deep Universal Probabilistic Programming