# PyTorch

## Components and Applications

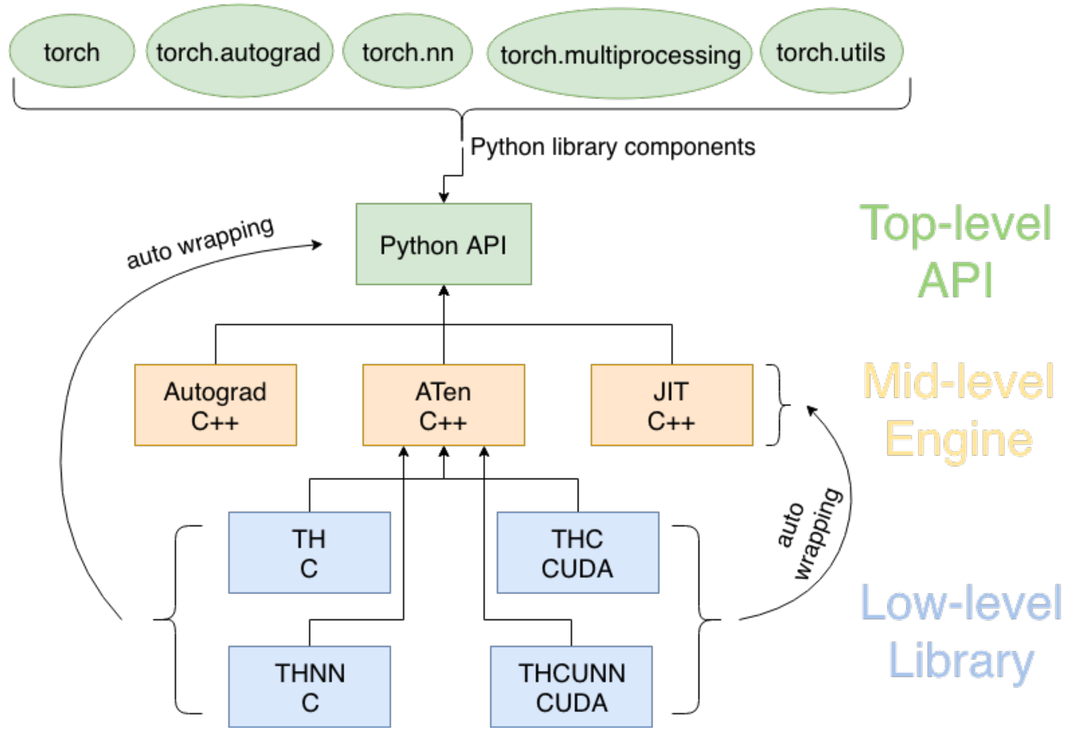
Mengqi

# Agenda

- PyTorch API
- PyTorch Python API Components
- Short PyTorch tutorial

# PyTorch API

- PyTorch is written mostly with C++ and CUDA
- Python is the main way of interacting with PyTorch
- PyTorch has API for Python, C++ and Java.




https://se.ewi.tudelft.nl/desosa2019/chapters/pytorch/#development-view

# Python Library

✦ PyTorch as a project has multiple Python libraries, including:

- torch
- torchaudio, torchtext, torchvision, pytorchvideo
- torcharrow(beta)
- torchdata(beta)
- torchrec
- torchserve

# Python Library

## torch

- ✦ torch provides an optimized tensor library for deep learning using both CPUs and GPUs
- ✦ The most important components are:
  - torch (tensor based data structure and operations, like numpy)
  - torch.autograd (do what its name suggests: automatic differentiation)
  - torch.jit (torchscript, a bridge between python and production env)
  - torch.nn and torch.nn.functional (provide common neural structures/layers)
  - torch.multiprocessing (wrapper of python multiprocessing module)
  - torch.utils (dataloader, and other utility functions)

BACK

# Python Library

## torchaudio, torchtext, torchvision, pytorchvideo

- ✦PyTorch's libraries for special types of data.
- ✦Provide tools from loading data into tensor, data transformations, to pre-trained models.
- ✦These librarys are integrated with autograd
- ✦example applications:
  - torchaudio: speech command classification
  - torchtext: text classification
  - torchvision: object detection in image, image classification
  - pytorchvideo: video action recognition

# Python Library

## torcharrow

- Library for data pre-processing in PyTorch models.
- Basically is a PyTorch version of Pandas that provides better integration with torch.
- Could provide better hardware acceleration in the future (GPU).

# Python Library

## torchdata

- ✦Library for creating flexible and performant data pipelines
- ✦Load, sample, shuffle data for neural network. Replace DataLoader
- ✦to solve problems of DataLoader, including better support streaming data

# Python Library

## torchrec

- ✦PyTorch domain library for Recommendation Systems.
- ✦introduced in Feb, 2022
- ✦why separate library for RS? Needs of special data, model structure.
- ✦e.g. matrix factorization RS

# Python Library

## torchserve

- ✦ Tool for serving and scaling PyTorch model in production
- ✦ Integrated with other pytorch components

# Demo Network Define&Training

train an image classifier

- ✦Download dataset CIRAR10 using torchvision
- ✦Define DataLoader

```
import torch
import torchvision
import torchvision.transforms as transforms
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data',
                                        train=True,
                                        download=True,
                                        transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data',
                                       train=False,
                                       download=True,
                                       transform=transform)
testloader = torch.utils.data.DataLoader(testset,
                                         batch_size=batch_size,
                                         shuffle=False,
                                         num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

BACK

# Demo Network Define&Training

train an image classifier

✦The define of neural network

- __init__(): all layers with learnable parameters. Defined by instantiating objects.

- forward(): non-learnable layers and network structure. Defined by calling functions.

- forward() is called by __call__()

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


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)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # flatten all dimensions except batch
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
output_result = net(input_data)
```

BACK

# Demo Network Define&Training

train an image classifier

✦autograd for calculating gradients
- grad data stored on each corresponding tensor
- a computational graph is stored in list
- a very good demonstration of this mechanism
- the gradients are used by optimizer to update model.

✦Each tensor has a grad_fn
- It is correspond to the calculation its tensor is involved in.
- It updates grad parameter of its tensor, and gives the gradient belong to its tensor's predecessor tensor(s)

Equal!

```
l.backward()
```

✦tensor.backward() does all!

```
torch.manual_seed(6)
x = torch.randn(4, 4, requires_grad=True)
y = torch.randn(4, 4, requires_grad=True)
z = x * y
l = z.sum()
```

```
dl = torch.tensor(1.)

dz = l.grad_fn(dl)

dx, dy = l.grad_fn.next_functions[0][0](dz)

print(x.grad, y.grad)
```

```
None None
```

```
_ = l.grad_fn.next_functions[0][0].next_functions[0][0](dx)

_ = l.grad_fn.next_functions[0][0].next_functions[1][0](dy)

print(torch.equal(dx, x.grad))
print(torch.equal(dy, y.grad))
print(z.grad_fn == l.grad_fn.next_functions[0][0])
```

```
True
True
True
```

BACK

# Demo Network Define&Training

train an image classifier

✦The actual training loop
- always reset grad using zero_grad() before calling backward()

✦Optimizer for updating parameters of model
- SGD
- Adam
- torch.optim provides several built-in optimizers.

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

BACK

# Articles Worth Checking

- https://se.ewi.tudelft.nl/desosa2019/chapters/pytorch/#development-view
- https://datawhalechina.github.io/thorough-pytorch/%E7%AC%AC%E5%85%AB%E7%AB%A0/8.2%20%E5%9B%BE%E5%83%8F%20-%20torchvision.html (in chinese)
- https://amsword.medium.com/understanding-pytorchs-autograd-with-grad-fn-and-next-functions-b2c4836daa00
- https://zhuanlan.zhihu.com/p/537868554
- https://zhuanlan.zhihu.com/p/486222256
- https://pytorch.org/blog/introducing-torchrec/
- https://pytorch.org/torchrec/
- https://blog.christianperone.com/2018/03/pytorch-internal-architecture-tour/
- https://blog.christianperone.com/2019/02/pydata-montreal-slides-for-the-talk-pytorch-under-the-hood/