
PyTorch

1.0 release + next steps

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Trevor Killeen, Francisco Massa, Adam Lerer, James Bradbury, Zeming Lin, Natalia Gimelshein, Christian Sarofeen, Alban Desmaison, Andreas Kopf, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, ...

Deep learning framework

~~Deep learning framework~~

~~Deep learning framework~~

NumPy

```
import numpy as np

a = np.array([[1, 2, 3],
              [4, 5, 6]], dtype=np.float)
b = np.array(2, dtype=np.float)

c = a + b

d = c[:, [0, 2]]

assert d.shape == (2, 2)

e = d @ np.random.normal(size=(2, 2))
```

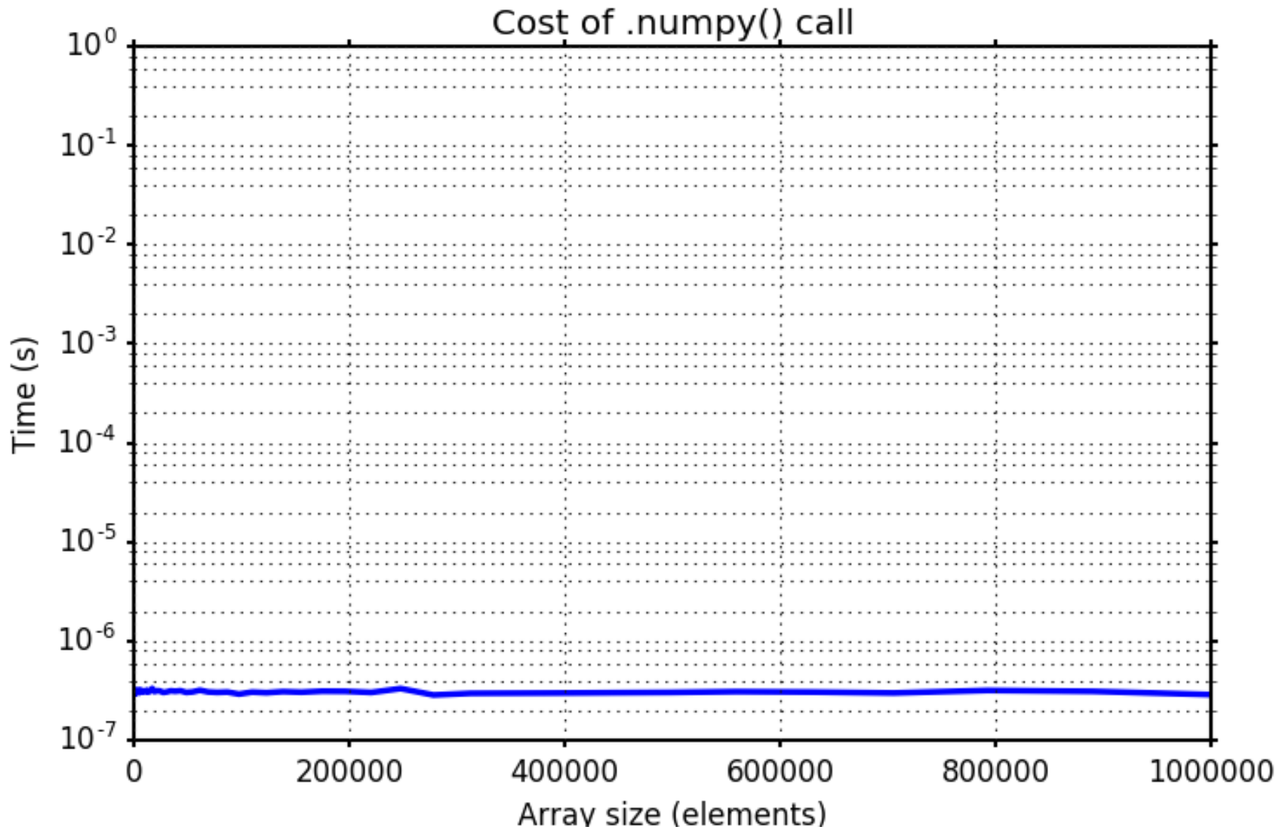
```
import torch

a = torch.tensor([[1, 2, 3],
                  [4, 5, 6]], dtype=torch.float)
b = torch.tensor(2, dtype=torch.float)

c = a + b

d = c[:, [0, 2]]

assert d.shape == (2, 2)

e = d @ torch.randn(2, 2, dtype=torch.float)
```

~~Deep learning framework~~

NumPy + ???

NumPy integration

```
import torch

x = torch.ones((2, 2), dtype=torch.double)
print(x)
# tensor([[ 1., 1.]
#         [ 1., 1.]], dtype=torch.float64)

y = x.numpy()
print(y)
# array([[ 1.,  1.],
#        [ 1.,  1.]])

z = torch.from_numpy(y)
print(z)
# tensor([[ 1., 1.]
#         [ 1., 1.]], dtype=torch.float64)
```

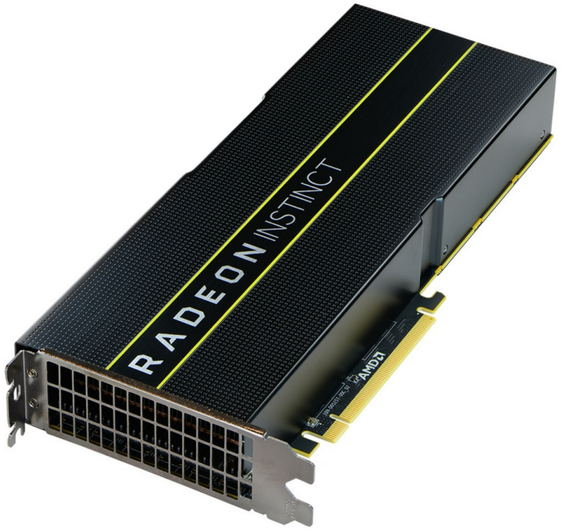

Cost of .numpy() call
Time (s)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
0
200000
400000
600000
800000
1000000
Array size (elements)

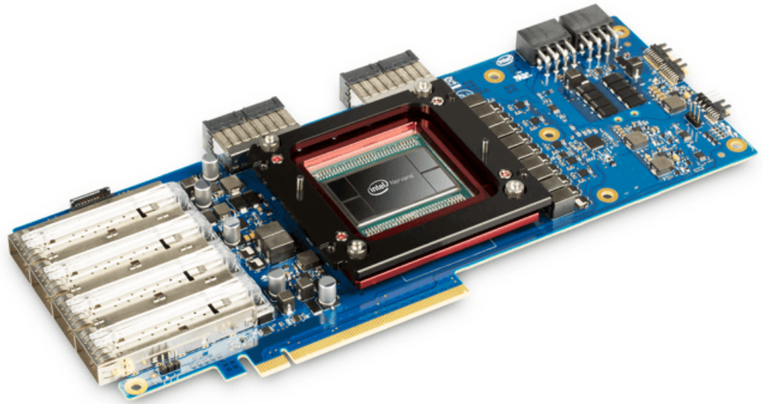
5 µs!

```
x += 1
print(arr)

# array([[ 2.,  2.],
#        [ 2.,  2.]])

np.add(arr, 1, out=arr)
print(x)

# tensor([[ 3., 3.]
#         [ 3., 3.]], dtype=torch.float64)
```

✅ Accelerator support


RTX 2080Ti
NVIDIA

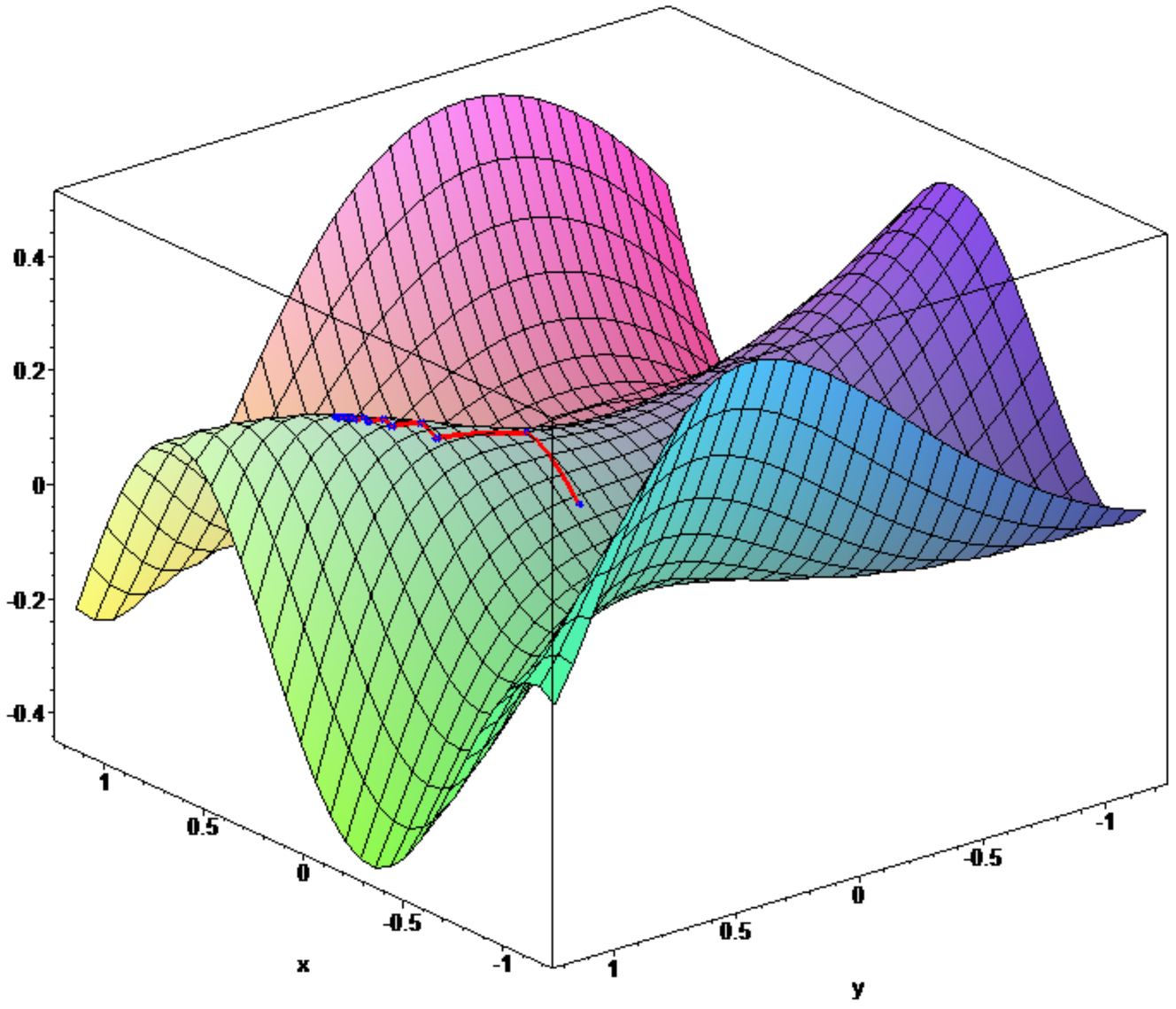

RADEON INSTINCT



```python
import torch



x = torch.randn((2, 2))
y = torch.randn((2, 2))

z = x + y

print(z)

# tensor([[1.4689,0.2254],
#         [1.3166,1.5713]])
```

```
import torch

dev = 'cuda:0' if torch.cuda.is_available() else 'cpu'

x = torch.randn((2, 2), device=dev)
y = torch.randn((2, 2)).to(x.device)

z = x + y # Runs on GPU!

print(z)

# tensor([[1.4689,0.2254],
#         [1.3166,1.5713]])
```

High-performance Automatic Differentiation


0.4
0.2
0
-0.2
-0.4
1
0.5
0
-0.5
-1
x
1
0.5
0
-0.5
-1
y

```python
import torch

x = torch.arange(4, requires_grad=True)

def poly(x):
    return x ** 2 + 5 * x + 2

# poly'(x) =  2x + 5
grad_x, = torch.autograd.grad(poly(x), x)

print(x)
# tensor([0., 1., 2., 3.])
print(grad_x)
# tensor([5., 7., 9., 11.])
```

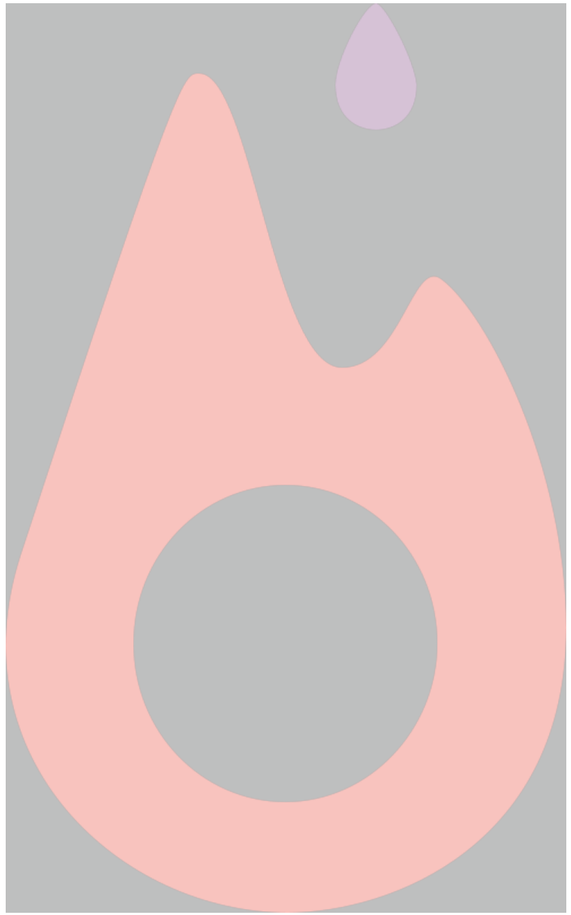
✅ High-level helpers for ML

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

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)
```

```
model.parameters()                                  # Iterator over parameters

model.state_dict()                                  # Weight serialization
model.load_state_dict(serialized_weights)           # and loading

model.eval()                                        # Train/eval mode toggle
model.train()

model.half()                                        # Type casts

model.zero_grad()                                   # Gradient management

model.cuda()                                        # Device changes

model = nn.DataParallel(model)                      # Multi-GPU

model = DistributedDataParallel(model)              # Multi-GPU + Multi-Node
```

✅ Data loading

- Vision
  - MNIST
  - Fashion MNIST
  - COCO (captioning and detection)
  - LSUN Classification
  - ImageFolder (generic classification dataset format)
  - Imagenet-12
  - CIFAR10 and CIFAR100
  - STL10
  - SVHN
  - PhotoTour
- Text
  - SST (sentiment analysis)
  - IMDb (sentiment analysis)
  - TREC (question classification)
  - SNLI (entailment)
  - Wikitext-2 (language modeling)
  - Abstract/generic support for machine translation



```
if opt.dataset in ['imagenet', 'folder', 'lfw']:
    dataset = dset.ImageFolder(root=opt.dataroot,
                               transform=transforms.Compose([
                                   transforms.Scale(opt.imageSize),
                                   transforms.CenterCrop(opt.imageSize),
                                   transforms.ToTensor(),
                                   transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                               ]))
elif opt.dataset == 'lsun':
    dataset = dset.LSUN(db_path=opt.dataroot, classes=['bedroom_train'],
                        transform=transforms.Compose([
                            transforms.Scale(opt.imageSize),
                            transforms.CenterCrop(opt.imageSize),
                            transforms.ToTensor(),
                            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                        ]))
elif opt.dataset == 'cifar10':
    dataset = dset.CIFAR10(root=opt.dataroot, download=True,
                           transform=transforms.Compose([
                               transforms.Scale(opt.imageSize),
                               transforms.ToTensor(),
                               transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
                           ]))
elif opt.dataset == 'fake':
    dataset = dset.FakeData(image_size=(3, opt.imageSize, opt.imageSize),
                            transform=transforms.ToTensor())
```

```
class MNIST(torch.utils.data.Dataset):
    def __init__(self, root, train=True, transform=None, target_transform=None):
        self.root = os.path.expanduser(root)
        self.transform = transform
        self.target_transform = target_transform

        self.download()
        self.data, self.labels = torch.load(
            os.path.join(self.root, self.processed_folder, MNIST.training_file))

    def __getitem__(self, index):
        img, target = self.data[index], self.labels[index]

        # doing this so that it is consistent with all other datasets to return a PIL Image
        img = Image.fromarray(img.numpy(), mode='L')

        if self.transform is not None:
            img = self.transform(img)
        if self.target_transform is not None:
            target = self.target_transform(target)

        return img, target

    def __len__(self):
        return len(self.data)
```

```
train_loader = torch.utils.data.DataLoader(
        train_dataset, batch_size=args.batch_size, shuffle=True,
        num_workers=args.workers, pin_memory=True)

for data_batch, label_batch in train_loader:
  ...  # train!
```

A single class for:

- multiprocess parallel data loading
- shuffling
- batching
- memory locking (for faster CUDA transfers)


PyTorch 1.0

Research ➡ Deployment

But what *deployment* really is?

PyTorch *Eager mode*

✅ Simple to write

✅ Simple to debug

❌ Hard to *deploy*

PyTorch *Script mode*

✅ Still Python

✅ Exportable

✅ Optimizable

⚠️ Only a subset

What works:

- ✅ Tensors
- ✅ Integral and floating-point scalars
- ✅ `if/while/for`
- ✅ `print`
- ✅ Strings
- ✅ Tuples
- ✅ Lists
- ✅ Function calls
- 🚧 ... much more coming

PyTorch *Eager*

`torch.jit.trace/script`

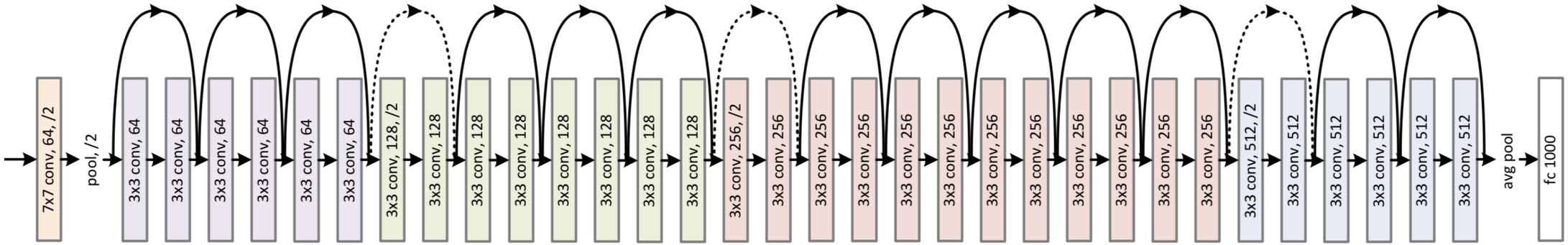
PyTorch *Script*

`torch.jit.trace`

✅ No code changes required

⚠️ Has to run your code on an example

⚠️ Control flow is inlined



Deep Residual Learning for Image Recognition, Kaiming He et al. https://arxiv.org/abs/1512.03385

```
convolutions = [
  nn.Conv2d(64, 64, kernel_size=3),
  nn.Conv2d(64, 64, kernel_size=3),
  nn.Conv2d(64, 128, kernel_size=3, stride=2),
  nn.Conv2d(128, 128, kernel_size=3, stride=2),
]

def model(x):
    for conv in convolutions:
        x = torch.relu(conv(x))
    return x
```

```
convolutions = [
  nn.Conv2d(64, 64, kernel_size=3),
  nn.Conv2d(64, 64, kernel_size=3),
  nn.Conv2d(64, 128, kernel_size=3, stride=2),
  nn.Conv2d(128, 128, kernel_size=3, stride=2),
]

def model(x):
    x = torch.relu(convolutions[0](x))
    x = torch.relu(convolutions[1](x))
    x = torch.relu(convolutions[2](x))
    x = torch.relu(convolutions[3](x))
    return x
```

```
import torchvision


model = torch.jit.trace(
    torchvision.models.resnet50(pretrained=True),
    args=(torch.randn(1, 3, 224, 224),))
```

`torch.jit.script`

✅ Program just as if you were writing Python

✅ Control flow is recovered correctly

⚠️ Restricted to a subset

```
@torch.jit.script
def lstm(x : Tensor,
         hidden : (Tensor, Tensor),
         w_ih : Tensor,
         w_hh : Tensor) -> (Tensor, (Tensor, Tensor)):

  outputs = []
  hx, cx = hidden

  for step in range(x.size(0)):
    hx, cx = lstm_cell(x[step], (hx, cx), w_ih, w_hh)
    outputs.append(hx)

  return torch.stack(outputs, dim=0), (hx, cx)
```

`trace` and `script` mix seamlessly …

`trace` and `script` mix seamlessly ...

and still allow you to call back to Python!

All *TorchScript* programs can be exported and run from native C++ environments!

```
auto model = torch::jit::load(path);
auto input = torch::randn({1, 3, 224, 224});
auto output = model->forward(inputs).toTensor();
```

Performance optimizations

Once a builtin doesn't fit what you're doing the perf drops ~5x

For an LSTM variant

23ms ➡ `torch.jit.script` ➡ 6ms

For an LSTM variant

23ms ➡ `torch.jit.script` ➡ 6ms

3.5x speedup!

C++ extensions/interface (beta!)

```
#include <torch/extension.h>


torch::Tensor compute(torch::Tensor x, torch::Tensor y) {
  auto z = torch::empty_like(x);
  x.mul_(2);
  compute_kernel<<<2, 4>>>(x.data<float>(),
                           y.data<float>(),
                           z.data<float>());
  return z;
}

PYBIND11_MODULE(TORCH_EXTENSION_NAME, m) {
  m.def("compute", &compute);
}
```

Setuptools

```
from setuptools import setup
from torch.utils.cpp_extension import BuildExtension, CUDAExtension

setup(
    name='extension',
    packages=['extension'],
    ext_modules=[CUDAExtension(
        'extension', ['extension.cpp', 'extension.cu']
    )],
    cmdclass=dict(build_ext=BuildExtension))
```

JIT loading

```
module = torch.utils.cpp_extension.load(
    name='extension',
    sources=['extension.cpp', 'extension.cu'])

module.compute(
    torch.ones(3, 4, device='cuda'), torch.randn(4, 5, device='cuda'))
```

```
import torch

class Net(torch.nn.Module):
    def __init__(self):
        self.fc1 = torch.nn.Linear(8, 64)
        self.fc2 = torch.nn.Linear(64, 1)


    def forward(self, x):
        x = torch.relu(self.fc1.forward(x))
        x = torch.dropout(x, p=0.5)
        x = torch.sigmoid(self.fc2.forward(x))
        return x
```

```
#include <torch/torch.h>

struct Net : torch::nn::Module {
  Net() : fc1(8, 64), fc2(64, 1) {
    register_module("fc1", fc1);
    register_module("fc2", fc2);
  }

  torch::Tensor forward(torch::Tensor x) {
    x = torch::relu(fc1->forward(x));
    x = torch::dropout(x, /*p=*/0.5);
    x = torch::sigmoid(fc2->forward(x));
    return x;
  }

  torch::nn::Linear fc1, fc2;
};
```

```
net = Net()

data_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('./data'))

optimizer = torch.optim.SGD(net.parameters())

for epoch in range(1, 11):
    for data, target in data_loader:
        optimizer.zero_grad()
        prediction = net(data)
        loss = F.nll_loss(prediction, target)
        loss.backward()
        optimizer.step()



    if epoch % 2 == 0:
        torch.save(net, "net.pt")
```

```
Net net;

auto data_loader = torch::data::data_loader(
  torch::data::datasets::MNIST("./data"));

torch::optim::SGD optimizer {net.parameters()};

for (size_t epoch = 1; epoch <= 10; ++epoch) {
    for (auto batch : data_loader) {
        optimizer.zero_grad();
        auto prediction = net.forward(batch.data);
        auto loss = torch::nll_loss(prediction, batch.label);
        loss.backward();
        optimizer.step();
    }
    if (epoch % 2 == 0) {
        torch::save(net, "net.pt");
    }
}
```

torch::nn

torch::optim

torch::data

torch::serialize

torch::python

torch::jit

Distributed

New abstractions

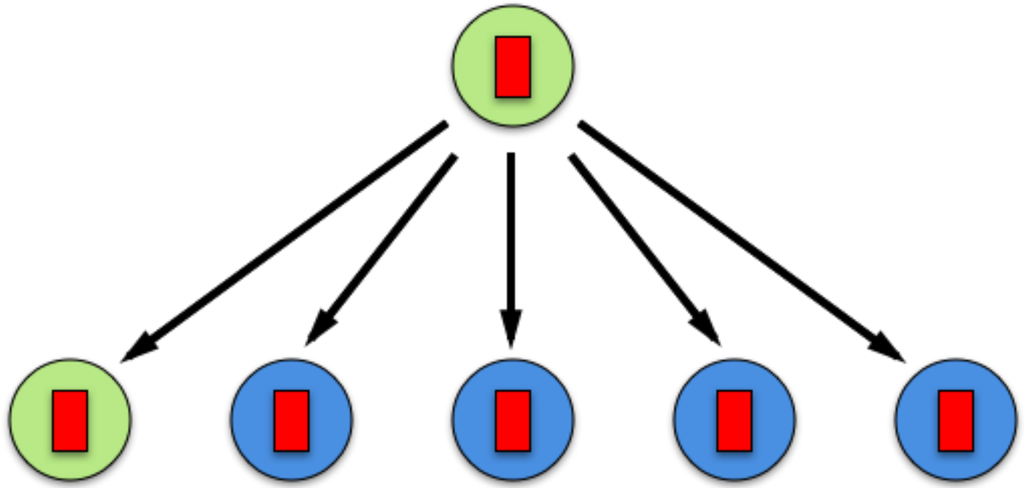
Asynchronous operation

Independent groups

Performance improvements

Fault tolerance

Elastic sizing



Caffe2

With ❤ from


facebook


NVIDIA


salesforce


ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIE
PARIS INSTITUTE OF TECHNOLOGY


Carnegie
Mellon
University


UNIVERSITE
PIERRE & MARIE CURIE
LA SCIENCE A PARIS


UBER


Stanford
University


UNIVERSITY OF
OXFORD


NYU


Inria


1794
ENS
ÉCOLE NORMALE
SUPÉRIEURE


VisionLabs
MACHINES CAN SEE


Digital
Reasoning