PYTORCH

PYTORCH

a Python library for next-generation (not only ML) research

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

```python
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))
```

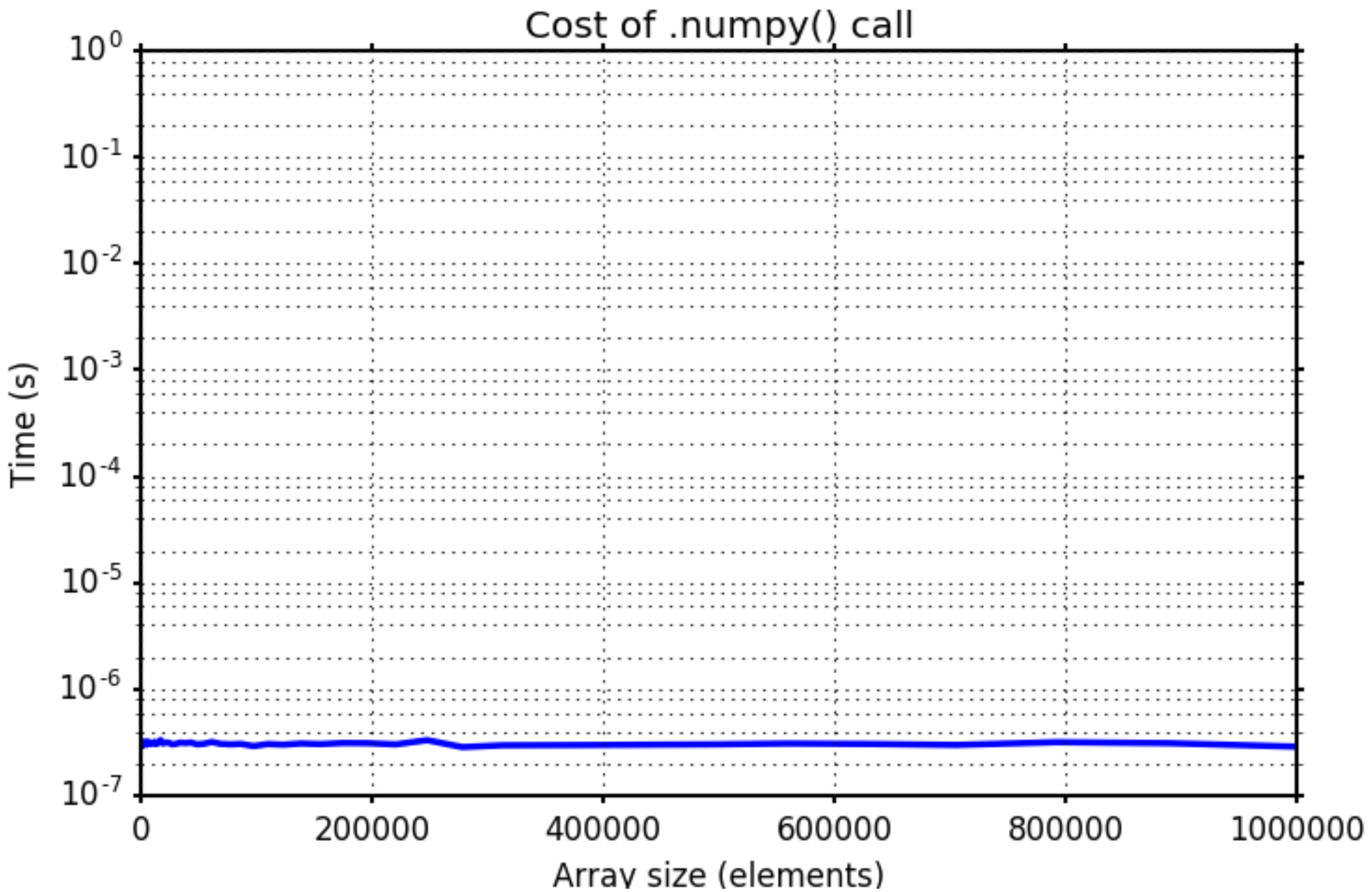
```python
import torch

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

```python
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)
```

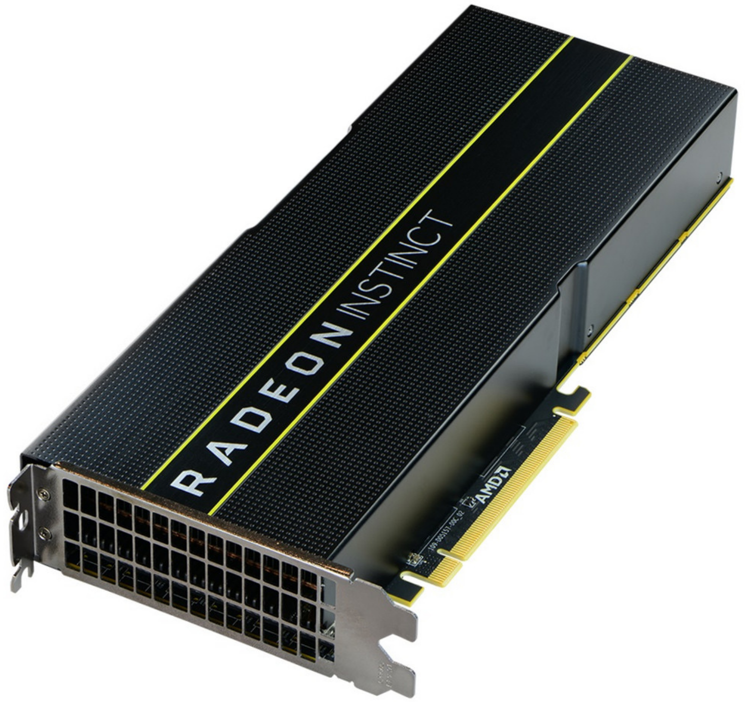

Cost of .numpy() call
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
Time (s)
0
200000
400000
600000
800000
1000000
Array size (elements)

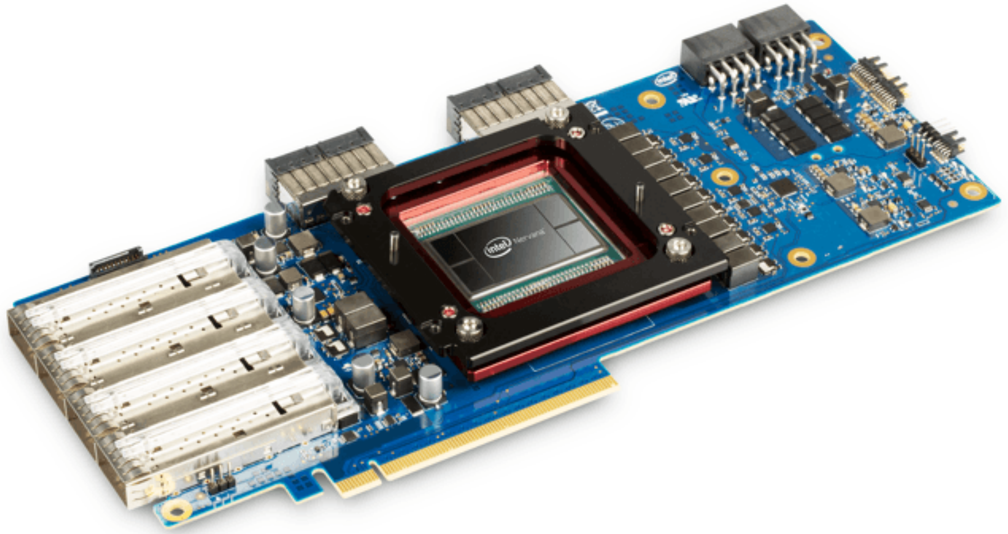
5 µs!

```python
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


TITAN V
NVIDIA

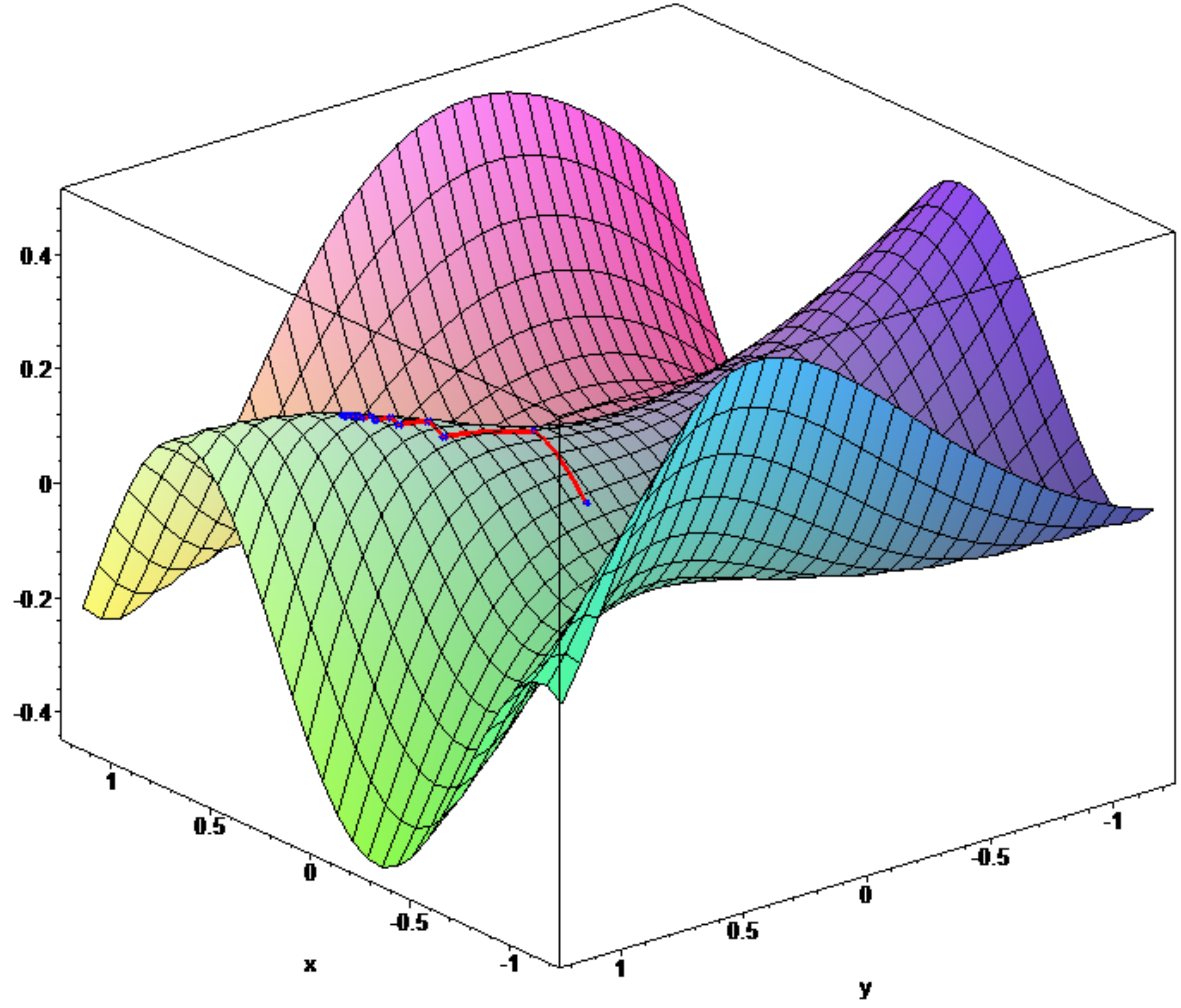

RADEON INSTINCT
AMD



```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]])
```

```python
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.])
```

PyTorch 1.0

🚧 *This is a release candidate!*

🕑 *Stable* release in a few months

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`

⬇

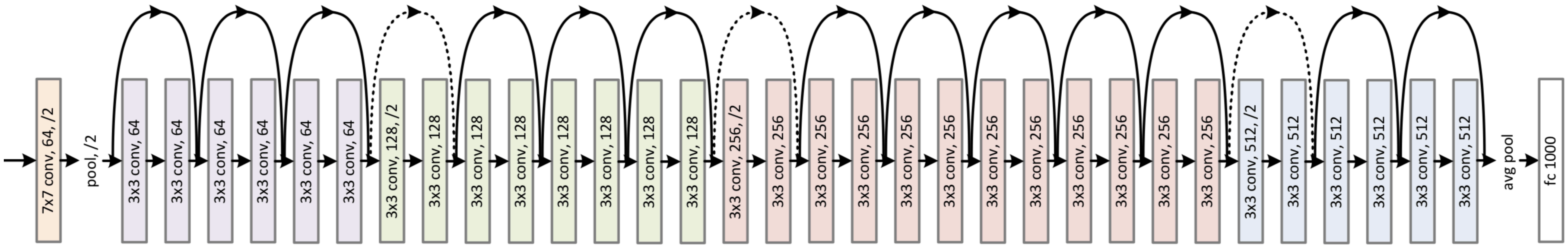
PyTorch *Script*

`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

```python
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
```

```python
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

```python
@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

41ms ➡ `torch.jit.script` ➡ 17ms

For an LSTM variant

41ms ➡ `torch.jit.script` ➡ 17ms

2.4x 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

```python
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

```python
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'))
```

```python
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
```

```cpp
#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;
};
```

```python
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")
```

```cpp
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

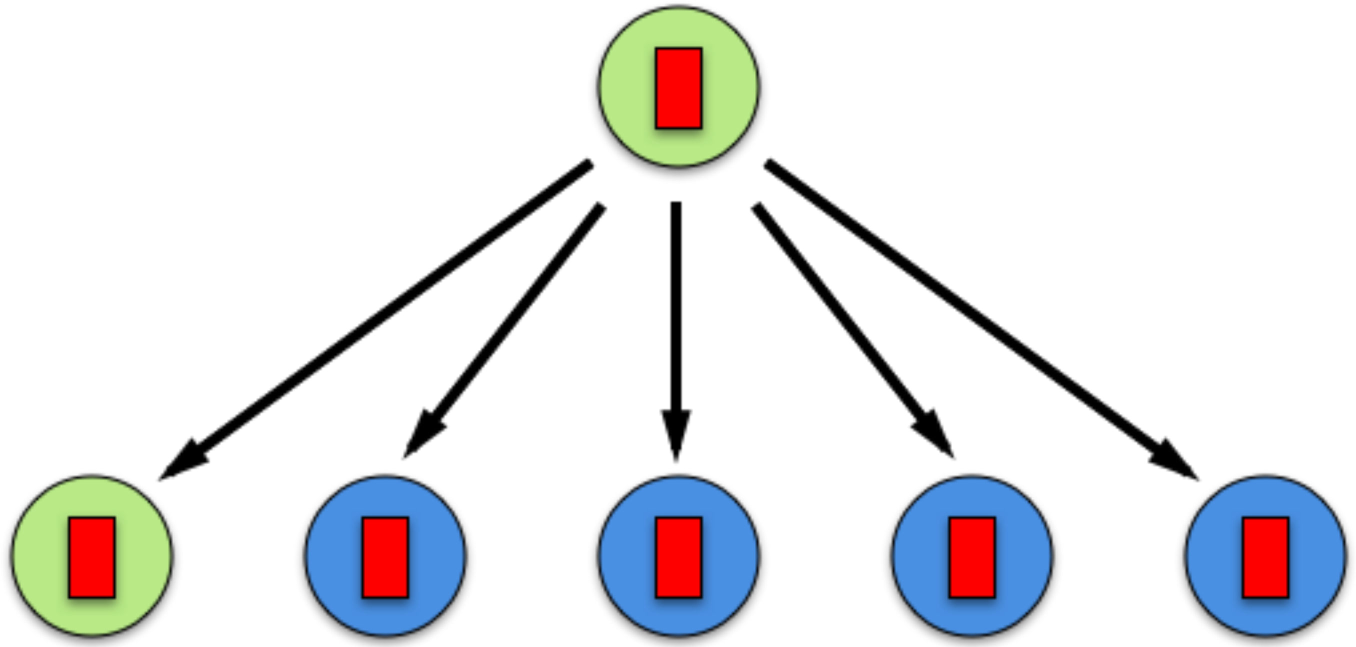
Asynchronous operation

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

UNIVERSITÉ 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