TorchScript and PyTorch JIT

Bram Wasti - Software Engineer

github.com/bwasti

"People are part of the system. The design should match the user's experience, expectations, and mental models."

- Jerome H. Saltzer, M. Frans Kaashoek

Python

Python is for users

Interpreted

Single threaded

Dynamically typed

Duck typed

Strongly typed

Flexible primitive types

Large standard library

Interoperable with C

PyTorch is Python

It looks and feels a lot like numpy, a popular multidimensional array manipulation library

PyTorch is Python

It looks and feels a lot like numpy, a popular multidimensional array manipulation library

```
import numpy as np

A_np = np.random.randn(128,256).astype(np.float32)
B_np = np.random.randn(256,512).astype(np.float32)
C_np = np.matmul(A_np, B_np)

import torch

A_torch = torch.randn(128,256)
B_torch = torch.randn(256,512)
C_torch = torch.matmul(A_torch, B_torch)
```

PyTorch is Python

But it's differentiable

```
import numpy as np

A_np = np.random.randn(128,256).astype(np.float32)
B_np = np.random.randn(256,512).astype(np.float32)
C_np = np.matmul(A_np, B_np)

import torch

A_torch = torch.randn(128,256, requires_grad=True)
B_torch = torch.randn(256,512)
C_torch = torch.matmul(A_torch, B_torch)

C_torch.sum().sum().backward()
```

PyTorch is Python

And has GPU support

```
import numpy as np

A_np = np.random.randn(128,256).astype(np.float32)
B_np = np.random.randn(256,512).astype(np.float32)
C_np = np.matmul(A_np, B_np)

import torch

cuda = torch.device('cuda')

A_torch = torch.randn(128,256, requires_grad=True, device=cuda)
B_torch = torch.randn(256,512, device=cuda)
C_torch = torch.matmul(A_torch, B_torch)

C_torch.sum().sum().backward()
```

PyTorch is Python

To be Pythonic

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

PyTorch is Python

Imperative

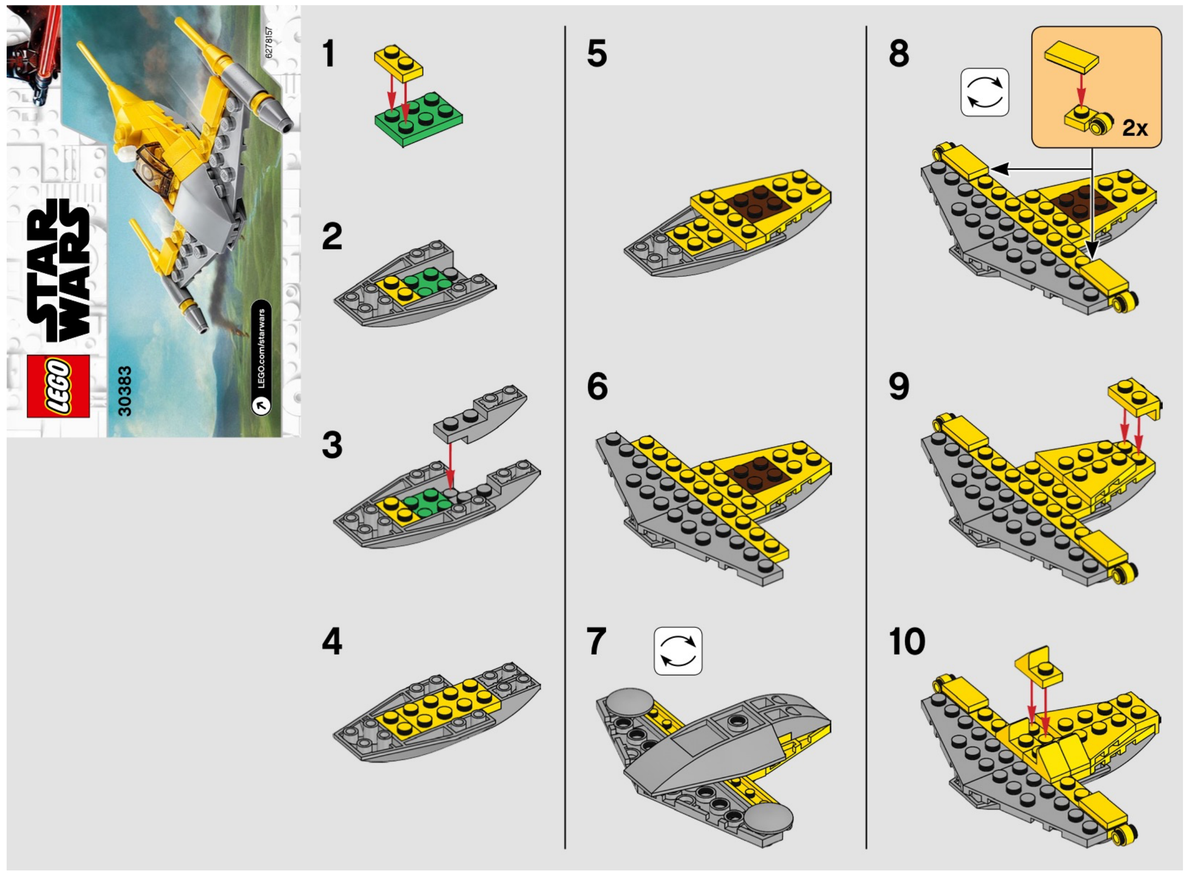
- You change the state of the world with every call
- To contrast, you are not describing a program in the form of a graph

Eager

- Things run when you expect them to
- Things fail where you expect them to

PyTorch is Lego

PyTorch is Lego



PyTorch is Lego

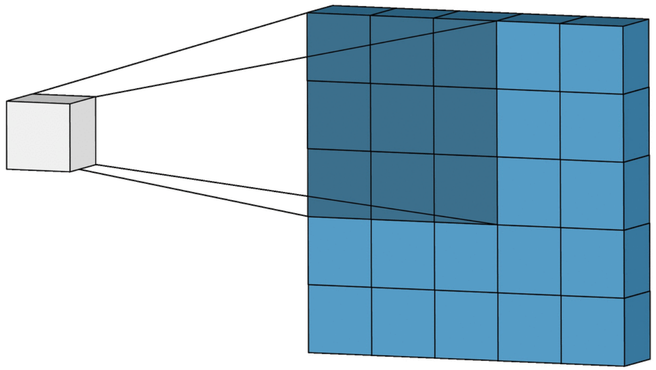
PyTorch ditched the idea of schematic machine learning

It was extremely successful in doing so

How does it work?

PyTorch: Internals

Tensor library



```
} else if (block_size == 16) {
  // unrolling 2 times
  for (int rangeIndex = 0; rangeIndex < output_size; ++rangeIndex) {
    float* op = &out[rangeIndex * block_size];
    __m256 vop0 = _mm256_setzero_ps();
    __m256 vop8 = _mm256_setzero_ps();
    if (dataInd + lengths[rangeIndex] > index_size) {
      return false;
    }
    for (int start = dataInd; dataInd < start + lengths[rangeIndex];
         ++dataInd) {
      const int idx = indices[dataInd];
      if (idx < 0 || idx >= data_size) {
        return false;
      }
      float wgt = 1.f;
      if (weights) {
        wgt = weights[IS_WEIGHT_POSITIONAL ? (dataInd - start) : dataInd];
      }
      __m256 vwgt = _mm256_set1_ps(wgt);
      const float* ip = &input[idx * fused_block_size];
      const int next_T0 = (dataInd < index_size - prefdist_T0)
          ? (dataInd + prefdist_T0)
          : dataInd;
      const int idx_pref_T0 = indices[next_T0];
      if (idx_pref_T0 < 0 || idx_pref_T0 >= data_size) {
        return false;
      }
      const float* ip_next_T0 = &input[idx_pref_T0 * fused_block_size];
      vop0 = _mm256_fmadd_ps(vwgt, _mm256_loadu_ps(ip + (0)), vop0);
      _mm_prefetch(
          reinterpret_cast<const char*>(&ip_next_T0[0]), _MM_HINT_T0);
      vop8 = _mm256_fmadd_ps(vwgt, _mm256_loadu_ps(ip + (8)), vop8);
      // skip unnecessary prefetch of (&ip_next_T0[8])
    }
    if (!normalize_by_lengths || lengths[rangeIndex] == 0) {
      _mm256_storeu_ps(&op[0], vop0);
      _mm256_storeu_ps(&op[8], vop8);
    } else {
      __m256 vlen_inv = _mm256_set1_ps(1.0f / lengths[rangeIndex]);
      _mm256_storeu_ps(&op[0], _mm256_mul_ps(vop0, vlen_inv));
      _mm256_storeu_ps(&op[8], _mm256_mul_ps(vop8, vlen_inv));
    }
  }
}
```

PyTorch: Internals

Autograd engine

```
import torch


def foo(x, y):
    if x.max() > y.max():
        r = x
    else:
        r = y
    return r

A = torch.randn(256, requires_grad=True)
B = torch.randn(256, requires_grad=True)
C = foo(A, B)
C.sum().backward()
```

A graph is created on the fly


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

New Needs

Mobile

Tight loops

Tiny models

Memory bound models

Oversubscribed boxes

Accelerators

Solution

We don’t want to ditch the programming model that has served us so well

We do want portability and performance

A portable optimizing compiler for Python!

Python support

Interpreted

Single threaded

Dynamically typed

Duck typed

Strongly typed

Flexible primitive types

Large standard library

Interoperable with C

Things a portable optimizing compiler likes

~~Interpreted~~

~~Single threaded~~

~~Dynamically typed~~

~~Duck typed~~

Strongly typed

~~Flexible primitive types~~

~~Large standard library~~

~~Interoperable with C~~

Introducing: TorchScript

Subset of Python

Statically typed

Makes choices about standard library support

Has PyTorch support built in

Integrates with Python

```
import torch

@torch.jit.script
def foo(x, y):
    if x.max() > y.max():
        r = x
    else:
        r = y
    return r
```

TorchScript is Portable

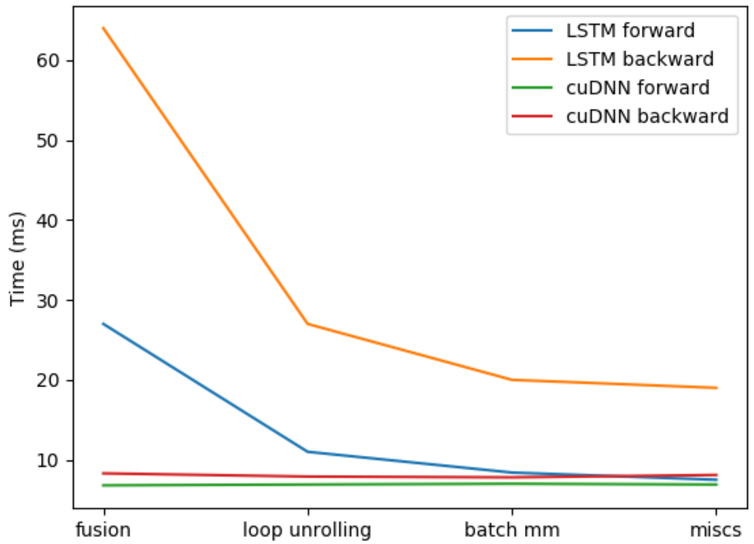
Save and load models anywhere

```python
import torch
import io

class MyModule(torch.nn.Module):
    def forward(self, x):
        return x + 10

m = torch.jit.script(MyModule())

# Save to file
torch.jit.save(m, 'scriptmodule.pt')

# Save to io.BytesIO buffer
buffer = io.BytesIO()
torch.jit.save(m, buffer)
```

```cpp
// Load in C++
std::string module_path = "scriptmodule.pt";
torch::jit::script::Module module = torch::jit::load(module_path);
```

TorchScript is Performant (theoretically)

Performance is largely enabled




```
import torch

A = torch.randn(256,256)
B = torch.randn(256,256) / 1000

@torch.jit.script
def foo_scripted(a, b):
  for _ in range(100):
    a = torch.matmul(a, b)
  return a


def foo(a, b):
  for _ in range(100):
    a = torch.matmul(a, b)
  return a

import time
t = time.time()
for _ in range(100):
  foo_scripted(A,B)
print(f"scripted: {(time.time() - t)*1000:.2f}ms/iter")

t = time.time()
for _ in range(100):
  foo(A,B)
print(f"unscripted: {(time.time() - t)*1000:.2f}ms/iter")
```

```
scripted: 4827.75ms/iter
unscripted: 5748.14ms/iter
```

TorchScript IR

SSA

Support for if and loops

Block arguments rather than Phi-nodes

Function calls [coming soon]

Construction from various sources

```
@torch.jit.script
def foo_scripted(a, b):
  for _ in range(100):
    a = torch.matmul(a, b)
  return a

print(foo_scripted.graph)
```

```
graph(%a.1 : Tensor,
      %b.1 : Tensor):
  %5 : bool = prim::Constant[value=1]() # test.py:8:3
  %2 : int = prim::Constant[value=100]() # test.py:8:18
  %a : Tensor = prim::Loop(%2, %5, %a.1) # test.py:8:3
    block0(%6 : int, %a.6 : Tensor):
      %a.3 : Tensor = aten::matmul(%a.6, %b.1) # test.py:9:9
      -> (%5, %a.3)
  return (%a)
```

TorchScript IR

Graph fusion

Algebraic rewrites

Transpose sinking

Branch: master ▾ **pytorch** / torch / csrc / jit / **passes** /

Create new file | Upload files | Find file | History

suo and **facebook-github-bot** serializing function calls (#23799) ... Latest commit 755f91b 16 hours ago

..		
onnx	Remove type subclassing (#24257)	5 days ago
utils	Cleanup interface of inlineCallTo.	21 days ago
alias_analysis.cpp	Cache node operators to speed up optimization (#24827)	yesterday
alias_analysis.h	Add in-place check to AliasDb	19 days ago
bailout_graph.cpp	fix uninitialized variable in BailOutInserter	last month
bailout_graph.h	Add comments to bailoug_graph.*	2 months ago
batch_mm.cpp	AliasAnalysisKind::CONSERVATIVE/FROM_SCHEMA (#22175)	26 days ago
batch_mm.h	clang format world (#15524)	8 months ago
canonicalize.cpp	s/uniqueName/debugName/ (#22096)	2 months ago
canonicalize.h	Canonicalize order of If and Loop outputs (#20015)	3 months ago
canonicalize_ops.cpp	canonicalize_ops pass bugfix: copy metadata for new output (#23809)	11 days ago
canonicalize_ops.h	clang format world (#15524)	8 months ago
common_subexpression_eliminati...	Dont introduce aliasing in CSE or Constant Pooling (#19576)	4 months ago
common_subexpression_eliminati...	clang format world (#15524)	8 months ago
constant_pooling.cpp	Dont introduce aliasing in CSE or Constant Pooling (#19576)	4 months ago
constant_pooling.h	clang format world (#15524)	8 months ago
constant_propagation.cpp	Better Constant Propagation through Tuples (#22561)	2 months ago
constant_propagation.h	clang format world (#15524)	8 months ago
create_autodiff_subgraphs.cpp	Dont introduce aliasing in CSE or Constant Pooling (#19576)	4 months ago
create_autodiff_subgraphs.h	clang format world (#15524)	8 months ago
dead_code_elimination.cpp	fix dce over loops	last month
dead_code_elimination.h	Fix dead code elimination in onnx export (#22476)	2 months ago
decompose_ops.cpp	Remove more uses of `DimensionedTensorType`	19 days ago
decompose_ops.h	split canonicalize_ops, make a decompose pass (#19988)	3 months ago
erase_number_types.cpp	remove CompleteTensorType	5 days ago

TorchScript IR

Graph fusion

Algebraic rewrites

Transpose sinking

Loop unrolling

Dead code elimination

Common subexpression elimination

Constant pooling

graph_fuser.cpp	Remove more uses of `DimensionedTensorType`	19 days ago
graph_fuser.h	Disable fusion of grad_sum_to_size (#23372)	26 days ago
guard_elimination.cpp	cleanup warnings	8 days ago
guard_elimination.h	Hook up profiled execution in the interpreter (#21799)	2 months ago
inline_autodiff_subgraphs.cpp	Canonicalize all includes in PyTorch. (#14849)	9 months ago
inline_autodiff_subgraphs.h	clang format world (#15524)	8 months ago
inline_fork_wait.cpp	Cleanup interface of inlineCallTo.	21 days ago
inline_fork_wait.h	Trace fork and join calls	7 months ago
inline_forked_closures.cpp	Change compiler to use Load/Stores, then transform to SSA (#21101)	2 months ago
inline_forked_closures.h	Change compiler to use Load/Stores, then transform to SSA (#21101)	2 months ago
inliner.cpp	Cleanup interface of inlineCallTo.	21 days ago
inliner.h	First class functions in IR, inlined eagerly (#21052)	3 months ago
inplace_check.cpp	clang format world (#15524)	8 months ago
inplace_check.h	clang format world (#15524)	8 months ago
insert_guards.cpp	Hook up profiled execution in the interpreter (#21799)	2 months ago
insert_guards.h	InsertGuards pass	3 months ago
lift_closures.cpp	Change compiler to use Load/Stores, then transform to SSA (#21101)	2 months ago
lift_closures.h	Change compiler to use Load/Stores, then transform to SSA (#21101)	2 months ago
liveness.cpp	s/uniqueName/debugName/ (#22096)	2 months ago
liveness.h	Liveness for BailOut graphs	2 months ago
loop_unrolling.cpp	Fix a bug in loop unrolling (#21239)	3 months ago
loop_unrolling.h	clang format world (#15524)	8 months ago
lower_grad_of.cpp	Revert D16161144: [pytorch][PR] Add traces to LowerGradOf and Special...	last month
lower_grad_of.h	clang format world (#15524)	8 months ago
lower_tuples.cpp	Improve the error message for ONNX export (#23317)	27 days ago
lower_tuples.h	clang format world (#15524)	8 months ago
onnx.cpp	Remove type subclassing (#24257)	5 days ago
onnx.h	split onnx passes (#22413)	2 months ago
peephole.cpp	remove CompleteTensorType	5 days ago

TorchScript JIT Profiler

Profiler

- Run the program a couple of times
- Record the shapes of that program

```
graph(%a.1 : Tensor,
      %b.1 : Tensor,
      %c : Tensor):
  %3 : int = prim::Constant[value=1]()
  %4 : int = prim::Constant[value=2]() # test_jit.py:4232:25
  %5 : int = prim::Constant[value=3]() # test_jit.py:4233:25
  %6 : int = prim::Constant[value=0]() # test_jit.py:4235:47
  %7 : Tensor = prim::profile(%b.1)
  %e.1 : Tensor = aten::add(%7, %5, %3) # test_jit.py:4233:21
  %9 : Tensor = prim::profile(%a.1)
  %10 : Tensor = prim::profile(%b.1)
  %f.1 : Tensor = aten::sub(%9, %10, %3) # test_jit.py:4234:21
  %12 : Tensor = prim::profile(%f.1)
  %13 : Tensor = prim::profile(%e.1)
  %14 : Tensor = aten::add(%12, %13, %3) # test_jit.py:4235:36
  %15 : Tensor = prim::profile(%14)
  %16 : Tensor = aten::clamp(%15, %6, %4) # test_jit.py:4235:24
  %17 : Tensor = prim::profile(%16)
   = prim::profile()
  return (%17)
```

TorchScript JIT Profiler

Profiler

- Run the program a couple of times
- Record the shapes of that program
- Insert "guard" nodes and specialize parts of the graph

```
graph(%a.1 : Tensor,
      %b.1 : Tensor,
      %c : Tensor):
  %20 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%b.1)
  %19 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%a.1)
  %18 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%b.1)
  %3 : int = prim::Constant[value=1]()
  %4 : int = prim::Constant[value=2]() # test_jit.py:4232:25
  %5 : int = prim::Constant[value=3]() # test_jit.py:4233:25
  %6 : int = prim::Constant[value=0]() # test_jit.py:4235:47
  %e.1 : Tensor = aten::add(%18, %5, %3) # test_jit.py:4233:21
  %22 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%e.1)
  %f.1 : Tensor = aten::sub(%19, %20, %3) # test_jit.py:4234:21
  %21 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%f.1)
  %14 : Tensor = aten::add(%21, %22, %3) # test_jit.py:4235:36
  %23 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%14)
  %16 : Tensor = aten::clamp(%23, %6, %4) # test_jit.py:4235:24
  %24 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%16)
  return (%24)
```

TorchScript JIT Profiler

Profiler

- Run the program a couple of times
- Record the shapes of that program
- Insert "guard" nodes and specialize parts of the graph
- Coalesce guard nodes

```
graph(%a.1 : Tensor,
      %b.1 : Tensor,
      %c : Tensor):
  %20 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%b.1)
  %19 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::Guard(%a.1)
  %3 : int = prim::Constant[value=1]()
  %4 : int = prim::Constant[value=2]() # test_jit.py:4232:25
  %5 : int = prim::Constant[value=3]() # test_jit.py:4233:25
  %6 : int = prim::Constant[value=0]() # test_jit.py:4235:47
  %e.1 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::add(%20, %5, %3)
  %f.1 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::sub(%19, %20, %3)
  %14 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::add(%f.1, %e.1, %3)
  %16 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::clamp(%14, %6, %4)
```

TorchScript JIT Profiler

Profiler

- Run the program a couple of times
- Record the shapes of that program
- Insert "guard" nodes and specialize parts of the graph
- Coalesce guard nodes
- Insert bailout paths to the IR and let the optimizer deal with the resultant graph

```
graph(%a.1 : Tensor,
      %b.1 : Tensor,
      %c : Tensor):
  %27 : int = prim::BailoutTemplate_0()
  %25 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::BailOut[index=0](%27, %b.1, %a.1)
  %26 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::BailOut[index=1](%27, %a.1, %25)
  %3 : int = prim::Constant[value=1]()
  %4 : int = prim::Constant[value=2]() # test_jit.py:4232:25
  %5 : int = prim::Constant[value=3]() # test_jit.py:4233:25
  %6 : int = prim::Constant[value=0]() # test_jit.py:4235:47
  %e.1 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::add(%25, %5, %3)
  %f.1 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::sub(%26, %25, %3)
  %14 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::add(%f.1, %e.1, %3)
  %16 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::clamp(%14, %6, %4)
  return (%16)
with prim::BailoutTemplate_0 = graph(%a.1 : Tensor,
      %b.1 : Tensor,
      %c : Tensor):
  %3 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::BailOut[index=0](%b.1, %a.1)
  %4 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = prim::BailOut[index=1](%a.1, %3)
  %5 : int = prim::Constant[value=1]()
  %6 : int = prim::Constant[value=2]() # test_jit.py:4232:25
  %7 : int = prim::Constant[value=3]() # test_jit.py:4233:25
  %8 : int = prim::Constant[value=0]() # test_jit.py:4235:47
  %e.1 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::add(%3, %7, %5)
  %f.1 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::sub(%4, %3, %5)
  %11 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::add(%f.1, %e.1, %5)
  %12 : ProfiledTensor(dtype = Double, requires_grad = 0 , shape = (2, 2) = aten::clamp(%11, %8, %6)
  return (%12)
```

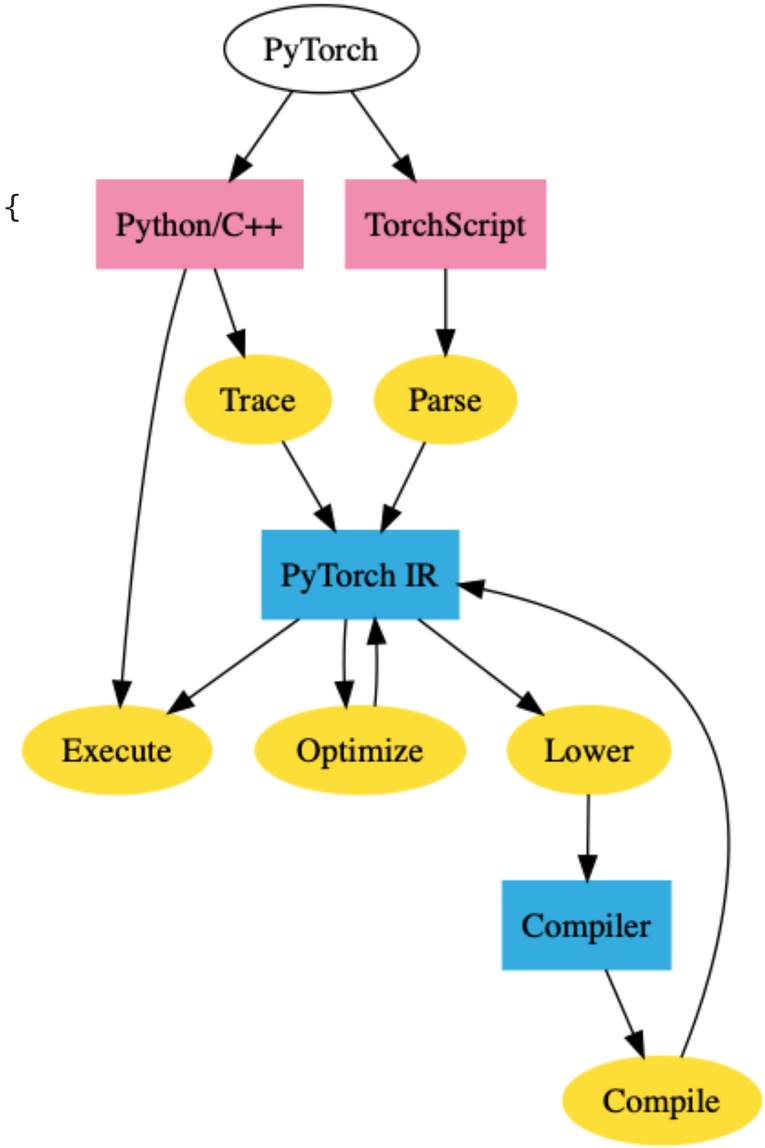
TorchScript JIT Backends

Custom backends

- Register a pass
- Register an operator

```
RegisterPass pass([](std::shared_ptr<Graph>& g) {
  if (fusion_enabled) {
    FuseLinear(g);
    FuseSupportedOps(g);
  }
});
```

```
auto options = c10::OperatorOptions();
options.setAliasAnalysis(AliasAnalysisKind::PURE);
RegisterOperators op({Operator(
    getTVMSymbol(),
    [](const Node* node) {
      auto cc = std::make_shared<TVMCompiler>(
          node, opt_level, strict, device_type, device, host);
      return [cc](Stack& stack) {
        RECORD_FUNCTION("TVM", std::vector<c10::IValue>());
        cc->run(stack);
        return 0;
      };
    },
    options)});
```

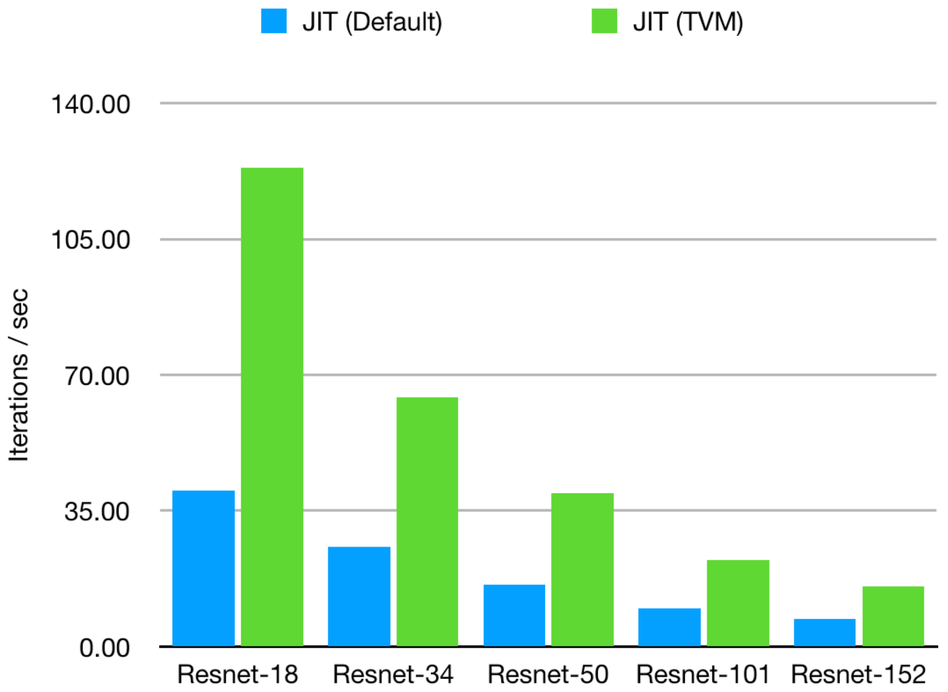


TorchScript JIT Backends

TVM - https://tvm.ai/2019/05/30/pytorch-frontend

- Open deep learning compiler stack for CPUs, GPUs, and specialized accelerators
- Apache-style governance model
- Uses Halide style compute and schedule definitions
- Autotuning

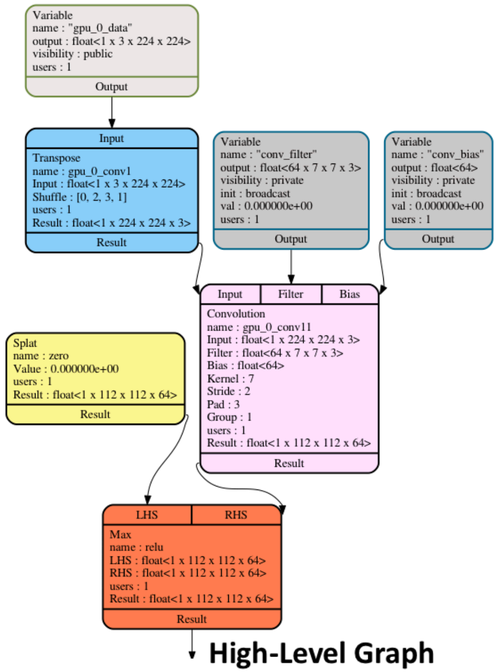
```
# Algorithm
k = tvm.reduce_axis((0, K), 'k')
A = tvm.placeholder((M, K), name='A')
B = tvm.placeholder((K, N), name='B')
C = tvm.compute(
           (M, N),
           lambda x, y: tvm.sum(A[x, k] * B[k, y], axis=k),
           name='C')
```



TorchScript JIT Backends

Glow

- Built for accelerators
- Has its own IR akin to LLVM



High-Level Graph

```
declare {
 %input = weight float<8 x 28 x 28 x 1>, broadcast, 0.0
 %filter = weight float<16 x 5 x 5 x 1>, xavier, 25.0
 %filter0 = weight float<16>, broadcast, 0.100
 %weights = weight float<10 x 144>, xavier, 144.0
 %bias = weight float<10>, broadcast, 0.100
 %selected = weight index<8 x 1>
 ...
 %result = weight float<8 x 10>
}

program {
 %allo = alloc float<8 x 28 x 28 x 16>
 %conv = convolution [5 1 2 16] @out %allo, @in %input,
     @in %filter3, @in %bias0
 %allo0 = alloc float<8 x 28 x 28 x 16>
 %relu = max0 @out %allo0, @in %allo
 %allo1 = alloc index<8 x 9 x 9 x 16 x 2>
 %allo2 = alloc float<8 x 9 x 9 x 16>
 %pool = pool max [3 3 0] @out %allo2, @in %allo0,
     @inout %allo1
 ...
 %deal6 = dealloc @out %allo6
 %deal7 = dealloc @out %allo7
 %deal8 = dealloc @out %allo8
 %deal9 = dealloc @out %allo9
}
```

Low-Level IR

```
LBB14_1:
  vmovaps 3211264(%rcx,%rax,4), %ymm1
  vmovaps 3211296(%rcx,%rax,4), %ymm2
  vmovaps 3211328(%rcx,%rax,4), %ymm3
  vaddps 6422528(%rcx,%rax,4), %ymm1, %ymm1
  vaddps 6422560(%rcx,%rax,4), %ymm2, %ymm2
  vmovaps 3211360(%rcx,%rax,4), %ymm4
  vaddps 6422592(%rcx,%rax,4), %ymm3, %ymm3
  vaddps 6422624(%rcx,%rax,4), %ymm4, %ymm4
  vmaxps %ymm0, %ymm1, %ymm1
  vmaxps %ymm0, %ymm2, %ymm2
  vmaxps %ymm0, %ymm3, %ymm3
  vmovaps %ymm1, 6422528(%rcx,%rax,4)
  vmovaps %ymm2, 6422560(%rcx,%rax,4)
  vmaxps %ymm0, %ymm4, %ymm1
  vmovaps %ymm3, 6422592(%rcx,%rax,4)
  vmovaps %ymm1, 6422624(%rcx,%rax,4)
  addq $32, %rax
```

Machine Code

In the works

Distributed Primitives - https://github.com/pytorch/pytorch/issues/23110

- RRef[T]
- Future[T]

```
ref.owner() # what is the worker this value lives on

v = ref.local_value() # if ref.owner() is local worker, then
                      # this returns the the underlying value,
                      # otherwise error.
# you can create a ref to a local tensor
t = torch.rand(3, 4)
ref2 = torch.RRef(t)

# in TorchScript, T can be automatically converted to RRef[T]
ref3 : RRef[Tensor] = t
```

```
v = fut.wait() # block the current thread
until v is ready

# local cpu task creation returns a future to
# the computed tensors
fut = torch.fork(lambda x, y:
                   x + y,
                   torch.rand(3, 4),
                   torch.rand(3, 4))
```

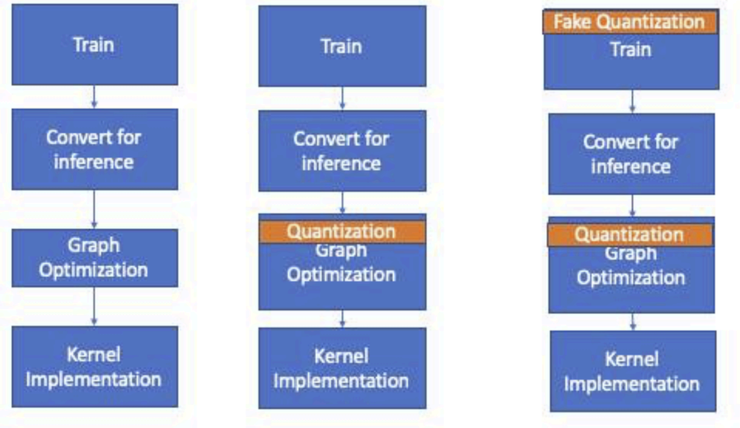
In the works

Quantization - https://github.com/pytorch/pytorch/wiki/Introducing-Quantized-Tensor

- QTensor
- Post training and quantization aware training modules

```
# Check size of Tensor
q.numel(), q.sizes(), ...

# Check whether the tensor is quantized
q.is_quantized

# Get the scale of the quantized Tensor,
# only works for affine quantized tensor
q.q_scale()

# Get the zero_point of quantized Tensor
q.q_zero_point()

# get the underlying integer representation of the QTensor
q.int_repr()
```

Model workflows, from left to right: 1. Non-quantized 2. Post-training quantization 3. Quantization-aware training



For greater accuracy, the quantization API will support modeling the effects of quanitization during training by inserting fake-quantization operators that will then be consumed by the same quantizer described above.

In the works

NamedTensors

- Comes from popular writeup: *Tensor Considered Harmful* by Alexander Rush
- Named axes make code more readable

LazyTensors

- We are missing a lot of simple perf optimizations in eager mode
- More importantly, we are missing all perf with off-host devices with large latencies
- We still want debugability

How To Contribute

Main repo - github.com/pytorch/pytorch

Issues

Pull requests

Contribution guide - https://pytorch.org/docs/stable/community/contribution_guide.html

Code browser - https://bddppq.github.io/codebrowser/pytorch/pytorch/

Ping me! bwasti@fb.com

Thanks :)

https://pytorch.org