Relay: a high level differentiable IR

Jared Roesch
TVMConf
December 12th, 2018
This represents months of joint work with lots of great folks:
How do we represent deep learning?

• Build parametric functions which approximate impossible or hard to program functions.

• In order to perform deep learning we need:
  • To represent computation
  • To differentiate
  • To optimize
Existing Approach

- **Resnet, DCGAN**
- **LSTM**
- **Training Loop**

**Computation Graph**

**Tensor Expression IR**

- **LLVM, CUDA, Metal**
- **VTA**

Edge FPGA, Cloud FPGA, ASIC
Existing Approach

Resnet, DCGAN \downarrow \quad \text{LSTM} \quad \downarrow \quad \text{Training Loop} \downarrow

High-Level Differentiable IR

Tensor Expression IR

LLVM, CUDA, Metal \quad \text{VTA}

Edge FPGA \quad \text{Cloud FPGA} \quad \text{ASIC}
for i in range(…):
    inp, hs = …

out, nhs = RNNCell(inp, hs)
Challenges

• How do we represent control-flow, functional abstraction, and recursion?
• How do we represent and optimize training?
• How do we perform end-to-end whole model optimization?
Relay

• **Relay** is the high level IR of the **TVM** stack.

• Generalize computation graphs to **differentiable** programs.

• Enables whole-program optimization for deep learning.

• Composed of new **IR, auto-diff, optimizer**, and **backends**.

• **Relay** is open source.
Initial Results

• **Relay** shows promising initial results when evaluated in inference tasks:

  • We are able fully optimize models such as generative **RNNs**, outperforming **PyTorch** by up to **3x** on model inference.

  • We demonstrate performance comparable to **NNVM** and outperform **TensorFlow** and **TensorFlow Lite**.

  • We show that **Relay** can be executed on **FPGAs**, resulting in up to an **11x** performance improvement over baseline.
Frontend

- DSL
- Model Importer
- Text Format
- On-disk representation

Compiler

- Operator Language
- Optimizer
- Compiled Operators
- Graph Runtime
- Reference Interpreter
- Ahead of time compiler

Execution

- FPGA
- GPU
- CPU
• A functional IR, an ML-like (ReasonML, OCaml, SML, …) language tailored to machine learning.

• Features closures, reference, ADTs, and primitive operators, tensors are the primary value type.

• We can use this to represent full-models including a generative RNN and training loops.

• Functional style makes it possible to analyze and transform as pure data-flow.
RNN
def @generate(n, i, h, ...):
    if (n == 0)
        []
    else
        let (output, new_hidden) =
            @rnn_cell(i, h, ...);
        output + @generate(
            n - 1, output, new_hidden, ...)

Parameters

Functional style loop
Typing

• Typing these programs introduces a few challenges:
  
  • Need static Tensor shape information to match accelerator primitives, optimize aggressively, and provide better errors.
  
  • Provide flexible typing for operators which contain shape input and output relationships such as broadcast, flatten, concat, squeeze, and more.
Tensor : (BaseType, Shape) -> Type

Float : (Width: Int, Lanes: Int) -> BaseType

f32 = Float<32, 1>

Tensor<f32, (32, 3, 32, 32)>

4-d Tensor
N * Channels * Height * Width
Type Relation

- Operators, the primitive building block of machine learning, are hard to type check (e.g. preconditions must hold over input tensors).

- A call can contain a series of relations which must hold over the input types.
  
  - Enables very flexible typing of operators.

- For example can implement variable arguments using relations (concat) and input/output relationships (broadcast).
For example we can type broadcasting addition:

```haskell
add : 
forall (Lhs: Type, Rhs: Type, Out: Type),
(Lhs, Rhs) -> Out
where Broadcast(Lhs, Rhs, Out)
```

Broadcasting is a tricky rule often employed in machine learning:

```haskell
Broadcast(Tensor<f32, (3, 4, 5)>, Tensor<f32 (n, 3, 4, 5), Tensor<f32, (n, 3, 4, 5)>)

Broadcast(Tensor<f32, (1, 5)>, Tensor<f32, (n, 5)>, Tensor<f32, (n, 5)>)
```
Or more complex constraints such as:

```
concat :
    forall (Args: Type, Out: Type),
    (Args) --> Out
where IsTuple(Args), Concat(Args, Out)
```
Optimizations

• We implement various optimizations over these programs including:

• Standard Optimizations
  • Fusion
  • Constant Propagation

• Accelerator Specific Optimizations
  • Quantization (see Ziheng's talk)
  • FoldScaleAxis
  • Data Packing
Backends

- Relay
- Graph Runtime
- Interpreter
- AoT Compiler
- FPGA
- GPU
- CPU
Backends

- We implemented multiple execution backends to demonstrate the versatility of Relay as an IR.

- Each backend builds on TVM’s existing low level Tensor IR (HalideIR).

- TVM is used for operators, but the rest of the program must be executed (e.g. allocation, control-flow, recursion).
Operator Compilation

```python
def @my_func(…)
    ...
}
```

TVM → operators.so
Graph Runtime

- TVM’s existing execution pipeline, can execute a subset of Relay programs.
- Requires a graph, a shared library containing operators, and parameters

GraphRTS + operators.so
Interpreter

- A reference interpreter for Relay.
- Implements the reference semantics.
- Uses naive recursive AST traversal for interpreting control flow.
- Uses JIT compilation for operators.
AoT Compiler

- A case study of what Relay IR affords, we built prototype compiler in less than 3 weeks.
- Generates code for CPU/GPU, FPGA support in the future.
- Removes interpretation overhead and enables optimization.
- Written as a pure Python library and uses Relay as dependency.
def @my_func(...) {
    ...
}

f = compile(my_func)
f(…)

librelay_aot_my_func.so

Diagram:
- Standard
- AoT
- Optimize
- Optimize
- LittleCpp
- Clang
• VTA is a target for Relay.

• We can compile high level models written in Frameworks such as MxNet directly to Relay.

• Generic compilation to VTA will be upstreamed soon after the conference.
• VTA is a target for Relay.

• We can compile high level models written in Frameworks such as MxNet directly to Relay.

• Generic compilation to VTA will be upstreamed soon after the conference.
Evaluation

• **Relay supports expressive models:**
  - We demonstrate Relay’s ability to optimize full models such as generative RNNs, beating PyTorch by up to $3x$.

• **Relay provides competitive performance:**
  - We demonstrate better than TensorFlow and on par performance with NNVM on a suite of models.

• **Relay supports customized hardware:**
  - We show how Relay and TVM can be used to execute on FPGA based accelerators, bring $11x$ performance improvement over baseline.
VTA Results

- **DCGAN**
  - CPU: 300 ms
  - FPGA: 50 ms

- **MobileNet**
  - CPU: 150 ms
  - FPGA: 30 ms

- **ResNet-18**
  - CPU: 300 ms
  - FPGA: 50 ms

- **ResNet-34**
  - CPU: 600 ms
  - FPGA: 70 ms

- **ResNet-50**
  - CPU: 700 ms
  - FPGA: 30 ms

**Target**
- CPU
- FPGA
Future Work

- Evaluating **Relay** on training tasks.

- AutoRelay: applying ideas from **AutoTVM** to **Relay**.

- A high-level full differentiable programming language frontend (i.e Python frontend, Haskell DSL).

- Novel analyses and optimizations for DL (e.g automatic differential privacy).

- Non-standard data types (e.g unums, posits).
Lessons Learned

• Using a full program representation we were able to:
  • Rephrase shape inference as type checking.
  • Use **Relay** as platform to develop novel optimizations such as automatic quantization.
  • Execute **Relay** programs via a variety of backends and hardware devices.
  • Demonstrate an increase in expressiveness does not come at the cost of performance.
Conclusion

• **Relay** is a new intermediate representation for optimizing deep learning programs.

• We apply the straightforward insight that machine learning models are just programs.

• This generalization enables support for a greater range of programs, new optimizations, and the ability to target a wide range of devices.

• Excited about production and research collaborations.

http://sampl.cs.washington.edu

http://tvm.ai