Deep Learning Systems

UCSD CSE 291 Differentiable Programming
Tzu-Mao Li
Recall: computational graph
Deep learning systems: each of the node is a multi-dimensional array
How to deep learning

Implement a bunch of operators on multidimensional arrays

torch.add
torch.matmul
torch.nn.Conv2d
torch.nn.ReLU
torch.fft
...

```
[...]
[...]
```

```
[...]
[...]
```
How to deep learning

Implement a bunch of operators on multidimensional arrays

torch.add
torch.matmul
torch.nn.Conv2d
torch.nn.ReLU
torch.fft
...

torch.add
torch.matmul
torch.nn.ConvTransposed2d
torch.nn.ReLU
torch.ifft
...

and their adjoints
How to deep learning

Use taping to record the forward computation
we always record the operations,
optionally, compute the values (aka “eager mode”)

```python
x = torch.tensor(...)  
y = 2 * x
for i in range(10):    
    y = torch.sin(y)
```
How to deep learning

run the tape backwards to compute gradients

```python
x = torch.tensor(...)  

y = 2 * x

for i in range(10):
    y = torch.sin(y)
```

voila! we have a deep learning framework
Why does the deep learning framework strategy work?

- observation 1: most deep learning operations are matrix multiplications or element wise operations
Sidetrack: how to make convolution a matrix multiplication?
Sidetrack: how to make convolution a matrix multiplication?
Sidetrack: how to make convolution a matrix multiplication?
Sidetrack: how to make convolution a matrix multiplication?

image

kernels
Sidetrack: how to make convolution a matrix multiplication?
Sidetrack: how to make convolution a matrix multiplication?

image

... kernels
Sidetrack: how to make convolution a matrix multiplication?

This conversion is often called “im2col”
Sidetrack: how to make convolution a matrix multiplication?

This conversion is often called “im2col”.

A specialized routine to perform this matrix multiplication is called “Winograd convolution”.
Why does the deep learning framework strategy work?

• observation 1: most deep learning operations are matrix multiplications or element wise operations

• therefore if we have an optimized matmul routine, we optimize all deep learning
Why does the deep learning framework strategy work?

• observation 1: most deep learning operations are matrix multiplications or element wise operations

• therefore if we have an optimized matmul routine, we optimize all deep learning

• observation 2: deep learning computation has high arithmetic intensity, thus we do not need to aggressively fuse the computations

• it’s further helped by “batching”: we can process a huge amount of data at once to increase arithmetic intensity
Bells and whistles: graph optimization

given a computational graph of multidimensional arrays (can be the adjoint code) we can often optimize it

how?
Bells and whistles: graph optimization

Bells and whistles: graph optimization

Bells and whistles: graph optimization

Diagram showing the process of enlarging and fusing convolutions in a graph optimization context. The diagram includes nodes labeled 'Input', 'Conv3x3 + Relu', 'Conv1x1 + Relu', 'Add', and 'Relu'. Arrows indicate the flow of data through the network.

For more information, see the source link: https://sampl.cs.washington.edu/tvmconf/slides/2019/Zhihao-Jia-TASO.pdf
Bells and whistles: graph optimization

Bells and whistles: graph optimization

the end result is 30% faster on a V100
Bells and whistles: graph optimization

first, prepare a set of “basic” algebraic rules

<table>
<thead>
<tr>
<th>Operator Property</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( y, z \cdot \text{ewadd}(x, \text{ewadd}(y, z)) = \text{ewadd}(\text{ewadd}(x, y), z) )</td>
<td>( \text{ewadd} ) is associative</td>
</tr>
<tr>
<td>( y, z \cdot \text{ewadd}(y, x) = \text{ewadd}(x, y) )</td>
<td>( \text{ewadd} ) is commutative</td>
</tr>
<tr>
<td>( \text{ewadd}(x, y) = \text{ewmul}(\text{ewmul}(x, y), x) )</td>
<td>( \text{ewmul} ) is associative</td>
</tr>
<tr>
<td>( \text{ewadd}(x, y) = \text{ewmul}(\text{ewmul}(x, y), x) )</td>
<td>( \text{ewmul} ) is commutative</td>
</tr>
<tr>
<td>( \text{ewadd}(x, y) = \text{ewmul}(\text{ewmul}(x, y), y) )</td>
<td>( \text{ewmul} ) is distributive</td>
</tr>
<tr>
<td>( \text{ewadd}(x, y) = \text{ewmul}(\text{ewmul}(x, y), y) )</td>
<td>( \text{ewmul} ) is associative</td>
</tr>
<tr>
<td>( \text{transpose}(\text{transpose}(x)) = x )</td>
<td>( \text{transpose} ) is its own inverse</td>
</tr>
<tr>
<td>( \text{transpose}(\text{ewadd}(x, y)) = \text{ewadd}(\text{transpose}(x), \text{transpose}(y)) )</td>
<td>operator commutativity</td>
</tr>
<tr>
<td>( \text{transpose}(\text{ewmul}(x, y)) = \text{ewmul}(\text{transpose}(x), \text{transpose}(y)) )</td>
<td>operator commutativity</td>
</tr>
<tr>
<td>( \text{transpose}(\text{ewmul}(x, y)) = \text{ewmul}(\text{transpose}(x), \text{transpose}(y)) )</td>
<td>operator commutativity</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(x, \text{matmul}(y, z)) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is associative</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(x, \text{matmul}(y, z)) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is commutative</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is distributive</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is associative</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is commutative</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is distributive</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is associative</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is commutative</td>
</tr>
<tr>
<td>( y, z \cdot \text{matmul}(\text{matmul}(x, y), z) = \text{matmul}(\text{matmul}(x, y), z) )</td>
<td>( \text{matmul} ) is distributive</td>
</tr>
</tbody>
</table>

TASO: Optimizing Deep Learning Computation with
Automatic Generation of Graph Substitutions

Zihao Jia
Stanford University
zihaoli@c.stanford.edu

Oded Padon
Stanford University
opadon@cs.stanford.edu

James Thomas
Stanford University
jthomas@stanford.edu

Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu
next, enumerate a bunch of “potential” graph rewrites

check whether they are correct using the algebraic rules (using “theorem provers” such as Z3)
Bells and whistles: graph optimization

can get a list of graph rewrite rules
to rewrite computation into equivalent forms

TASO: Optimizing Deep Learning Computation with
Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu
Bells and whistles: graph optimization

given an input computational graph, apply a sequence of rewrites to minimize a “cost model” (e.g., measure how much time each operator costs in a hardware)
Bells and whistles: graph optimization

the cost minimization can be casted as a discrete search problem (can be solved by A*, beam search, MCTS, etc)

TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu
A few other cool papers in this domain

Exocompilation for Productive Programming of Hardware Accelerators

Yuka Ikarashi*  
MIT CSAIL, USA

Gilbert Louis Bernstein*  
UC Berkeley, USA

Alex Reinking  
UC Berkeley, USA

Hasan Genc  
UC Berkeley, USA

Jonathan Ragan-Kelley  
MIT CSAIL, USA

Verified Tensor-Program Optimization Via High-Level Scheduling Rewrites

AMANDA LIU, Massachusetts Institute of Technology, USA

GILBERT LOUIS BERNSTEIN, University of California, Berkeley, USA

ADAM CHILIPALA, Massachusetts Institute of Technology, USA

JONATHAN RAGAN-KELLEY, Massachusetts Institute of Technology, USA
Bells and whistles: Triton

```python
@triton.jit
def add_kernel(x_ptr, # *Pointer* to first input vector.
y_ptr,    # *Pointer* to second input vector.
output_ptr, # *Pointer* to output vector.
n_elements, # Size of the vector.
BLOCK_SIZE: tl.constexpr, # Number of elements each program should process.
            # NOTE: `constexpr` so it can be used as a shape value.
):
    pid = tl.program_id(axis=0) # We use a 1D launch grid so axis is 0.
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    mask = offsets < n_elements
    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    output = x + y
    tl.store(output_ptr + offsets, output, mask=mask)
```

basically CUDA, but operates at “block” instead of “thread” level
this enables much easier warp-level operations
PyTorch 2 generates Triton code from the computational graph

https://triton-lang.org/main/getting-started/tutorials/01-vector-add.html
Jax

- basic idea: construct computational graph where all graph operators are numpy functions
- reuse TensorFlow’s graph optimizer and code generator (XLA)
- profit
A unique idea in Jax: vmap

adds extra dimensionality to the inputs of a function

vector-vector dot product

\[
\text{vv} = \lambda x, y: \text{jnp.vdot}(x, y) \quad \# \quad ([a], [a]) \rightarrow []
\]

\[
\text{mv} = \text{vmap}(\text{vv}, (0, \text{None}), 0) \quad \# \quad ([b,a], [a]) \rightarrow [b] \quad \text{matrix-vector multiplication}
\]

\[
\text{mm} = \text{vmap}(\text{mv}, (\text{None}, 1), 1) \quad \# \quad ([b,a], [a,c]) \rightarrow [b,c] \quad \text{matrix-matrix multiplication}
\]
A fun trivia: do TensorFlow/PyTorch obey cheap gradient principle?

\[
\begin{align*}
\text{let } A : \mathbb{R}^{n \times n} = \text{diag}(x) \text{ in } \\
& \text{tr}(A) + \text{tr}(A) + \cdots + \text{tr}(A) \\
\text{diag}(x) &= \begin{bmatrix}
x & 0 & 0 \\
0 & x & 0 \\
0 & 0 & x \\
\vdots & \vdots & \ddots
\end{bmatrix} \\
\text{tr}(A) &= \sum_{i=0}^{n} A_{ii}
\end{align*}
\]
A fun trivia:
do TensorFlow/PyTorch obey cheap gradient principle?
A fun trivia:
do TensorFlow/PyTorch obey cheap gradient principle?
Tensorflow/PyTorch’s reverse mode violates cheap gradient principle

\[
\frac{O(kn^2)}{O(n^2 + kn)} = O(\min(k, n)) > O(1)
\]
Empirical verification

**PyTorch**

- runtime (sec)
- ratio

**TensorFlow**

- runtime (sec)
- ratio

\[ \text{let } A : \mathbb{R}^{n \times n} = \text{diag}(x) \text{ in } \underbrace{\text{tr}(A) + \text{tr}(A) + \cdots + \text{tr}(A)}_{k} \]

\( n = 20,000 \)
What’s wrong?

• the adjoint of a trace (O(N) operation) requires creating a N^2 matrix

• autodiff needs to be **sparsity-aware**, even when dealing with dense arrays

\[
\text{tr}(A) = \sum_{i=0}^{n} A_{ii}
\]

\[
\text{diag}(x) = \begin{bmatrix}
x & 0 & 0 \\
0 & x & 0 \\
0 & 0 & x \\
\vdots & \ddots & \ddots
\end{bmatrix}
\]

this needs to be sparse
Sparsity-aware, pure functional autodiff

- the adjoint of a trace (O(N) operation) requires creating a N^2 matrix k times
- autodiff needs to be sparsity-aware, even when dealing with dense arrays

Differentiating A Tensor Language

GILBERT BERNSTEIN, University of California, Berkeley
MICHAEL MARA, Stanford University
TZU-MAO LI, Massachusetts Institute of Technology
DOUGAL MACLAURIN, Google
JONATHAN RAGAN-KELLEY, Massachusetts Institute of Technology