Instead of Rewriting Foreign Code for Machine Learning, Automatically Synthesize Fast Gradients

William S. Moses

Valentin Churavy

wmoses@mit.edu
NeurIPS ’20 Poster Preview
December 2020
Differentiation & Machine Learning

• Computing derivatives is key to many machine learning algorithms

• Existing approaches:
  • Rewrite all code in a differentiable DSL (PyTorch, TensorFlow, Taichi, etc)
  • Manually writing gradient functions

• Hinders application of ML to new domains

• Automatic differentiation (AD) aims to close this gap
Existing Automatic Differentiation Pipelines
Optimization & Automatic Differentiation

Differentiating before optimizing can create asymptotically slower gradients.

\[
O(n^2) \quad \text{for } i=0..n \{ \\
\quad \text{out}[i] /= \text{mag}(\text{in}) \}
\]

\[
O(n) \quad \text{Optimize} \quad \text{res} = \text{mag}(\text{in}) \quad \text{for } i=0..n \{ \\
\quad \text{out}[i] /= \text{res} \}
\]

\[
O(n) \quad \text{AD} \quad \text{d_res} = 0.0 \quad \text{for } i=n..0 \{ \\
\quad \text{d_res} += \text{d_out}[i]... \\
\quad \text{d_mag}(\text{d_in}, \text{d_res})
\}
\]

\[
O(n^2) \quad \text{AD} \quad \text{for } i=0..n \{ \\
\quad \text{out}[i] /= \text{mag}(\text{in}) \\
\quad \text{d_res} = \text{d_out}[i]... \\
\quad \text{d_mag}(\text{d_in}, \text{d_res})
\}
\]

\[
O(n^2) \quad \text{Optimize} \quad \text{for } i=n..0 \{ \\
\quad \text{d_res} = \text{d_out}[i]... \\
\quad \text{d_mag}(\text{d_in}, \text{d_res})
\}
\]
Enzyme Approach

Perform AD on **optimized, language-independent** representation!
Challenges of Low-Level AD

- "AD is more effective in high-level compiled languages (e.g. Julia, Swift, Rust, Nim) than traditional ones such as C/C++, Fortran and LLVM IR [...]" – Innes

- Low-level code lacks information necessary to compute adjoints

- Created new interprocedural analyses to derive information and optimize

```c
struct Type {
    double;
    int*;
}
x = Type*;
```
Speedup of Enzyme

Enzyme is \textbf{4.2x faster} than Reference!
import torch
from torch_enzyme import enzyme

# Create some initial tensor
inp = ...

# Apply foreign function to tensor
out = enzyme("test.c", "f").apply(inp)

# Derive gradient
out.backward()
print(inp.grad)

import tensorflow as tf
from tf_enzyme import enzyme

inp = tf.Variable(…)

# Use external C code as a regular TF op
out = enzyme(inp, filename="test.c", function="f")

# Results is a TF tensor
out = tf.sigmoid(out)

// Input tensor + size, and output tensor
void f(float* inp, size_t n, float* out);

// diffe_dupnoneed specifies not recomputing the output
void diffef(float* inp, float* d_inp, size_t n, float* d_out) {
    __enzyme_autodiff(f, diffe_dup, inp, d_inp, n, diffe_dupnoneed, (float*)0, d_out);
}
Enzyme

- Tool for performing reverse-mode AD of statically analyzable LLVM IR
- Differentiates code in a variety of languages (C, C++, Fortran, Julia, Rust, Swift, etc)
- 4.2x speedup over AD before optimization
- State-of-the-art performance with existing tools
- PyTorch-Enzyme & TensorFlow-Enzyme lets researchers use foreign code in ML workflow
- Open source (enzyme.mit.edu & join our mailing list)
- For more information come to our poster & spotlight presentation!