Clad - Clang plugin for Automatic Differentiation

Aleksandr Efremov

Mentors: Vassil Vassilev, Oksana Shadura
What automatic differentiation is

- Technique for evaluating the derivatives of mathematical functions
- Applicable to computer programs (e.g. C++ code)
- Alternative to numerical differentiation
Numerical differentiation

\[ f'(x) = \lim_{h \to 0} \frac{f(x+h)-f(x)}{h} \]

Finite difference method:

\[ f'(x) \approx \frac{f(x+h)-f(x)}{h} \]

(for small value of \( h \))
Numerical differentiation

\[ f'(x) \approx \frac{f(x+h) - f(x)}{h} \]

Disadvantages:

- Need to evaluate \( f \) twice
- Slow gradient computation
- Prone to numerical errors
- How to select \( h \)?

[Wikipedia, Numerical differentiation]
Automatic differentiation is

- Applies basic rules of symbolic differentiation
- To the source code of the original function
- The result is the code of the function that computes value of the derivative
- Without additional precision loss
- Without inefficiently long expressions

Baydin et al., Automatic Differentiation in Machine Learning: a Survey, 2018
Alternative to symbolic differentiation

- Not limited to closed-form expressions
- Can take derivatives of algorithms (conditionals, loops, recursion)

![Diagram of automatic differentiation]

\[ y = f(x) \]
\[ df(x) \]
\[ y' = f'(x) \]
Automatic differentiation

- **Forward mode AD** algorithm allows to compute derivatives w.r.t. any (single) variable
- Theoretical result guarantees that the derivative can be computed in at most \(2.5\) times more arithmetic operations than the original function
- Propagates derivatives from the dependent towards the independent variables

\[
f(x_1, x_2) = \sin(x_1) + x_1 x_2
\]

[Wikipedia, Automatic differentiation]
Computing gradients

- What if we want to compute a gradient (vector of derivatives w.r.t. every variable)?
- For a function of $N$ inputs, we have to call $N$ functions produced by the forward mode.
- Total complexity is $O(2.5^*N^*M)$, where $M$ is the complexity of the original function.
- **Reverse mode** allows to compute gradients with complexity of $O(4^*M)$, independently of $N$, which is much more efficient for functions with many parameters.
Automatic differentiation

- **Reverse mode AD** allows to compute gradients
- Gradients can be computed in **at most 4 times** more arithmetic operations than the original function *(independently of the number of input variables)*
- Propagates derivatives from the independent variables towards the result of the function

\[ f(x_1, x_2) = \sin(x_1) + x_1 x_2 \]

[Wikipedia, Automatic differentiation]

---

Aleksandr Efremov  
DIANA Meeting, 08.10.2018
AD implementations

- Source transformation
- Operator overloading
- Several implementations exist, see: http://www.autodiff.org/?module=Tools
Automatic differentiation in Clad

- Clad is a **Clang compiler plugin**
- Performs C++ **source code transformation**, based on Clang AST

```cpp
double f(double x) {
    return x * x;
}
```
Automatic differentiation in Clad

- Given some C++ function $f$
- *(no source modification needed, but must be visible for the compiler)*
- User specifies the function, independent variables, differentiation mode
- Clad performs the transformation *(in compile time)*
- Another C++ function for $f'$ is produced
Automatic differentiation in Clad

double f(double x) { return x*x; }

... clad::differentiate(f, 0);
...

double f_darg0(double x) { return 1*x + x*1; }

std::cout << f_darg0(1) << '\n'; // Prints 2
Clad capabilities

- Most C++ constructs are (will be) supported
- As long as the function is differentiable

Example: temporary variable declarations

```c++
double f(double x, double y) {
    double t = x*x;
    return t + y;
}

double f_darg0(double x, double y) {
    double dt = 1*x + x*1;
    return dt + 0;
}
```
Clad capabilities

Example: loops

double pow(double x, int n) {
    double r = 1;
    for (int i = 0; i < n; i++)
        r = r*x;
    return r;
}

double pow_darg0(double x, int n) {
    double dr = 0;
    double r = 1;
    for (int i = 0; i < n; i++) {
        dr = dr*x + r*1;
        r = r*x;
    }
    return dr;
}
Contributions

- Implemented **Reverse mode AD** for **efficient gradients computation**
- Extended the functionality to support **control flow** and **variable reassignments**
- Integrated Clad into **ROOT** (CERN’s framework for data analysis), available through **TFormula**, to be used in minimization and fitting
Potential applications

Any gradient-based optimization methods, for example:

- Function minimization
- Backpropagation for machine learning
- Fitting models to data

In ROOT:

- Minuit
- TMVA
- RooFit
Future work

- Support more C++ constructs, enable differentiation of any C++ code in general
- Improve user interface to allow more expressiveness for independent variables, differentiation mode specification
- Support OpenCL/CUDA code differentiation
- Implement advanced features of forward/reverse mode AD to produce optimal code
- Efficient differentiation of functions with multiple outputs (Jacobians), Hessians
Thank you!