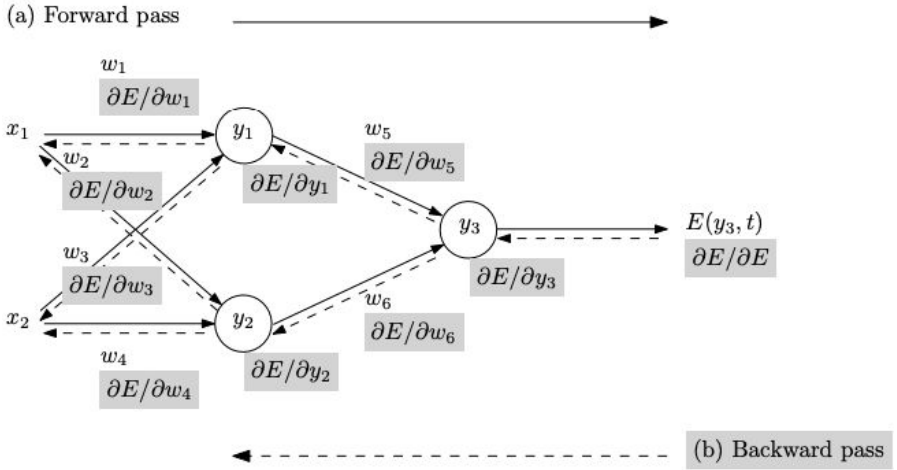
Improve automatic differentiation of object-oriented paradigms using Clad

Petro Zarytskyi
Google Summer of Code
Julius-Maximilians-Universität, Germany
Mentors: Vassil Vassilev, David Lange

COMPILER
C | R
RESEARCH

Introduction: Automatic Differentiation

Automatic differentiation is a method of differentiation of functions expressed as procedures. It involves breaking up the function into simple operations and applying chain rule to each one of them. This can be done both ways: from the input to the output (forward mode) and vice versa (reverse mode). This project focuses on the second approach which is more efficient for computing gradients. In reverse mode, we need two passes: a forward pass to store the intermediate values of all the variables and a backward pass to compute derivatives.



COMPILER C | R RESEARCH

Examples: what didn't work

Original code

```
double f(double* x, double y) {
    g(x, y); // g(double*, double)
    double z = x[3] + y;
    return z;
}
```

Code differentiated by Clad

```
void f_grad(...) {
    // double _t0 = x; ??
    g(x, y);
    double _d_z = 0.;
    double z = x[3] + y;
    ...
    // x = _t0; ??
    ...
}
```

COMPILER C | R RESEARCH

Examples: what didn't work

Original code

```
double f(double* x, double y) {
    g(x, y); // g(double*, double)
    double z = x[3] + y;
    return z;
}
```

C-arrays are not copyable!

Code differentiated by Clad

```
void f_grad(...) {
    // double _t0 = x; ??
    g(x, y);
    double _d_z = 0.;
    double z = x[3] + y;
    ...
    // x = _t0; ??
    ...
}
```

What doesn’t work

The same goes with all non-copyable types:

- std::initializer_list
- std::unique_ptr, std::shared_ptr, etc.
- Other STL and user-defined types.

Note: Methods of such classes follow the same logic as plain functions.

Approach 2: nice partial solution

Let’s say we have analysed g and we know that it only affects the third element of x

Original code

```
double f(double* x, double y) {
    g(x, y); // g(double*, double)
    ...
    return z;
}
```

Code differentiated by Clad

```
void f_grad(...) {
    double _t0 = x[3];
    g(x, y);
    ...
    x[3] = _t0;
    ...
}
```

Approach 2: nice partial solution

Let's say we have analysed g and we know that it only affects the third element of x

Original code

```
double f(double* x, double y) {
    g(x, y); // g(double*, double)
    ...
    return z;
}
```

Code differentiated by Clad

```
void f_grad(...) {
    double _t0 = x[3];
    g(x, y);
    ...
    x[3] = _t0;
    ...
}
```

We decided to focus on Approach 1 for now!

COMPILER C | R RESEARCH

Approach 1: general solution

We introduce a new type of non-copyable types that can automatically store and restore multiple objects at the same time

Original code

```
double f(double* x, double y) {
    g(x, y); // g(double*, double)
    ...
    return z;
}
```

Code differentiated by Clad

```
void f_grad(...) {
  clad::smart_tape _t0;
  g_reverse_forw(x, y, _d_x, 0., _t0);
  ...
  _t0.restore();
  g_pullback(...);
  ...
}
```

COMPILER C | R RESEARCH

Approach 1: general solution

Original code

```
double g(double* x, double y) {
    for (int i = 0; i < 9; ++i) {
        x[i] *= y;
    }
    return x[3];
}
```

Code differentiated by Clad

```
double g_reverse_forw(..., clad::smart_tape &tape) {
    unsigned long _t0 = 0UL;
    int _d_i = 0;
    for (int i = 0; ; ++i) {
        {
            if (!(i < 9))
                break;
        }
        _t0++;
        tape.store(x[i]);
        x[i] *= y;
    }
    return x[3];
}
```

Approach 1: Progress

Progress so far:

- Implemented clad::smart_tape, enabled its usage in reverse_forw functions.
- Took different approaches to minimize the usage of clad::smart_tape when unnecessary, including the analysis of function's signature, avoiding storing local variables, run-time analysis.
- Tested in different scenarios, including nested revere_forw functions, member functions, local variables, etc.
- Did a private pass over the implementation with Vassil, re-implemented clad::smart_tape as an interval-based map.
- Improvements in TBR: internal refactorings, improved support for arrays and pointers, added support for undefined VarData, optimized the visitation process, and almost implemented support for pointer reassignments.

Other progress

In total, 18 merged PRs and 2 open

- Improved reverse_forw mode: made the scheduling static-only, removed a lot of unused statements, made the naming consistent, and improved the support for pointers.
- Enabled support for all (non-linear) constructor_pullback generation.
- Opened a PR that enables support for constructors of derived classes and std::forward, and also enables native support for the constructors of std::pair constructors.
- Fixed a regression in the JIT-ing time (#1371) in Clad by introducing static scheduling of partial pullbacks.
- Enabled static scheduling of derivatives in the forward mode and reverse_forw functions. Made handling parameters differentiability consistent and centralized.
- Fixed the SmallPT ray-tracer demo in Clad. Updated it to showcase the reverse mode instead of the forward mode.

Thank you!