Improve automatic differentiation of object-oriented paradigms using Clad

Name: Daemond Zhang
Institute: Tsinghua University
Mentors: Vassil Vassilev, Parth Arora

Abstract

I would like to apply for the **Improve automatic differentiation of object-oriented paradigms using Clad** project.

This project is mainly focused on supporting object-oriented programming features in clad, including several milestones such as differentiation of constructors, differentiation of operator overloads, reference class members, and custom derivatives for object-oriented constructs. It's valuable as no other AD system has done this before, and thus is experimental. But there is no necessity to build from scratch since clad has already supported many c++ syntaxes like differentiation of operator overloads in forward mode etc.

With some previous background in the field of automatic differentiation, I believe I can produce meaningful results through discussion with other contributors in clad organization and my hard work.

Motivation

Clad is an automatic differentiation (AD) clang plugin for C++. Given a C++ source code of a mathematical function, it can automatically generate C++ code for computing derivatives of the function. Clad has found uses in statistical analysis and uncertainty assessment applications.

Object oriented paradigms (OOP) provide a structured approach for complex use cases, allowing for modular components that can be reused & extended. OOP also allows for abstraction which makes code easier to reason about & maintain. Gaining full OOP support is an open research area for automatic differentiation tools.

Supporting object oriented paradigms in clad will allow users to compute derivatives to the algorithms in their projects seamlessly, using an object-oriented model. C++ object-oriented constructs include but are not limited to: classes, inheritance, polymorphism, and related features such as operator overloading.

By enhancing Clad to correctly differentiate when non-differentiable elements are involved and adding support for differentiating special member functions, our work will finally enable users to realize clad's potential by making it compatible with Softsusy and Eigen libraries codebases.

Project Deliverables

1. Add support for differentiation of special member functions
2. Improve support for differentiation of operator overloads
3. Add support for functions which return pointer/reference in the reverse mode
4. Add support for custom derivatives for special member functions
5. Research about way to improve Clad object-oriented differentiable model such as adding attribute functionality to mark a function or a member variable as non-differentiable. This can used to signal Clad to not to try to differentiate such a function / member variable.
6. Extend the test and documentation coverage

Implementation Plan

Add support for differentiation of special member functions

Currently, we mainly lack support for differentiation of constructors, which can be seen as a generalization of the "=" operator. After implementing support for differentiation of constructors, we will be able to perform copy operations on class types similar to those on builtin types, such as "int y = x;" -> "classA y = x".

PR #497 adds support for differentiating copy constructors of class ValueAndPushforward and that too, only the C++ generated default ones. However for other class types and user defined copy constructors, we need to generalize the differentiation of constructors with the help of a constructor pull back function.

I will develop the feature based on the following code.

```
Expr* dValueExpr =
 utils::BuildMemberExpr(m_Sema, getCurrentScope(), dfdx(), "value"); StmtDiff
clonedValueEI = Visit(ILE->getInit(0), dValueExpr).getExpr(); clonedExprs[0] =
clonedValueEI.getExpr();
Expr* dPushforwardExpr = utils::BuildMemberExpr(m_Sema, getCurrentScope(),  dfdx(),
"pushforward"); Expr* clonedPushforwardEI =
 Visit(ILE->getInit(1), dPushforwardExpr).getExpr();
clonedExprs[1] = clonedPushforwardEI;
Expr* clonedILE = m_Sema.ActOnInitList(noLoc, clonedExprs, noLoc).get(); return
StmtDiff(clonedILE);
```

For the ValueAndPushforward class, we need to handle `dfdx().value` and `dfdx().pushforward` separately. Similarly, for other classes, we need to handle all member variables in a similar way.

This generalization will also help to differentiate Move constructors, Default object creation constructors and Parameterised constructors.

Improve support for differentiation of operator overloads

We need to improve support for differentiation of overloaded operators in reverse-mode.

I carefully looked into PR #397 and PR #256, which respectively add support for member functions and the () operator in reverse mode.

Other operator overloads are similar to this in that they involve an indirect call to member

functions. Some key points are as follows:

- Using CXXOperatorCallExpr to handle operator overloads and obtain the 'this'

  pointer

```
StmtDiff ForwardModeVisitor::VisitCallExpr(const CallExpr* CE) {
 ...
 if (auto OCE = dyn_cast<CXXOperatorCallExpr>(CE))
 baseOriginalE = OCE->getArg(0);
 ...
}
```

- We can use a separate variable to store the derivative of the function value with respect to implicit this pointer and use some other variables to contain derivatives of all the call arguments.

```
StmtDiff ForwardModeVisitor::VisitCallExpr(const CallExpr* CE) {
 ...
 // Stores differentiation result of implicit `this` object, if any.  StmtDiff
baseDiff;
 ...
 // Add base derivative expression in the derived call output args list
DerivedCallOutputArgs.push_back(BuildOp(UnaryOperatorKind::UO_AddrOf,
baseDiff.getExpr_dx()));
 ...
}
```

Add support for functions which return pointer/reference in the reverse mode

I looked into this PR, but it seems like a wrong usage. We will work out a more detailed example of code about what we would like to see.

Add support for custom derivatives for special member functions

For example, we would like to support custom derivatives of constructors. Currently, clad can support specifying custom derivatives of class functions. For instance,

```
namespace clad {
namespace custom_derivatives {
```

```
namespace class_functions {
template <typename T>
void clear_pushforward(::std::vector<T>* v, ::std::vector<T>* d_v) {
 d_v->clear();
 v->clear();
}
template <typename T>
void resize_pushforward(::std::vector<T>* v, unsigned sz, ::std::vector<T>* d_v,
unsigned d_sz) {
 d_v->resize(sz, T());
 v->resize(sz);
}
template <typename T, typename U>
void resize_pushforward(::std::vector<T>* v, unsigned sz, U val,
 ::std::vector<T>* d_v, unsigned d_sz, U d_val) {  d_v->resize(sz, d_val);
 v->resize(sz, val);
}
} // namespace class_functions
} // namespace custom_derivatives
} // namespace clad
```

I suppose that special member functions are similar in this regard, as they also require finding user-defined functions in the context, as in the following code:

```
Expr* DerivativeBuilder::BuildCallToCustomDerivative(
 DeclarationNameInfo DNI, llvm::SmallVectorImpl<Expr*>& CallArgs,  clang::Scope* S,
clang::DeclContext* originalFnDC,
 bool forCustomDerv /*=true*/, bool namespaceShouldExist /*=true*/) {  ...
 LookupResult R(m_Sema, DNI, Sema::LookupOrdinaryName);
 if (DC)
 m_Sema.LookupQualifiedName(R, DC);
 Expr* OverloadedFn = 0;
 if (!R.empty()) {
 // FIXME: We should find a way to specify nested name specifier  // after finding the
custom derivative.
 Expr* UnresolvedLookup =
 m_Sema.BuildDeclarationNameExpr(SS, R, /*ADL*/ false).get();
 llvm::MutableArrayRef<Expr*> MARargs =
 llvm::MutableArrayRef<Expr*>(CallArgs);
 SourceLocation Loc;
 if (noOverloadExists(UnresolvedLookup, MARargs)) {
 return 0;
 }
 OverloadedFn =
 m_Sema.ActOnCallExpr(S, UnresolvedLookup, Loc, MARargs, Loc).get();  }
 ...
}
```

In addition, we also hope to allow class designers to specify custom derivatives of class

functions directly within the class.

```
class UserDefinedClass {
 public:
 double memFn(double i, double j) { ... }

 double memFn_pushforward(double i, double j, double d_i, double d_j) { ... }  void
memFn_pullback(double i, double j, clad::array_ref<double> d_i,
clad::array_ref<double> d_j) { ... }
}
```

Regarding this goal, I need to explore further to come up with a more detailed

design plan.

Research ways to improve Clad object-oriented differentiable model

We will create a formal specification model document of the meaning of differentiating a class object in Clad. Detail in the formal specification what is and what is not allowed in a differentiable class.

Then we can do some improvements such as adding attribute functionality to mark a function or a member variable as non-differentiable. This can be used to signal Clad not to try to differentiate such a function/member variable.

This section involves experimental research and requires a deeper discussion of architecture design. Therefore, we will design a more detailed plan during the summer.

Extend the test and documentation coverage

For the above milestones, I will add corresponding tests and documentation once they are

completed.

Timeline

I would like to divide this project into 5 phases. I saw on the website that the Mentor availability is from June to October. My availability is flexible enough. Though I designed a schedule based on Google's timeline, I will adjust it based on mentors' availability.

Phase	Tasks
May 4 - May 28	During this Community Bonding Period, I will learn more about the content of the Clad project, discuss with mentors and other contributors to explore more detailed designs for the above milestones.

May 29 - June 2	Implement differentiation of constructors
June 5 - June 9	Add tests and documentation for differentiation of constructors
June 12 - June 16	Implement differentiation of operator overloads
June 19 - June 23	Add tests and documentation for differentiation of operator overloads
June 26 - June 30	Implement differentiation of functions which return pointer/reference in the reverse mode
July 3 - July 7	Add tests and documentation for differentiation of functions which return pointer/reference
July 10 - July 14	Implement differentiation of custom derivatives for special member functions
July 17 - July 21	Add tests and documentation for differentiation of custom derivatives for special member functions
July 24 - July 28	Create a formal specification model document of the meaning of differentiating a class object in Clad.
July 31 - Aug 4	Add attribute functionality to mark a function or a member variable as non-differentiable
Aug 7 - Aug 18	Do more research on ways to improve Clad object-oriented differentiable model
Aug 22- Aug 28	Buffer-time. Test and submit final code and project summaries.

Obligations

I will be available to work full-time during the project period. I believe that communication is a vital aspect of any project and to ensure that the status of the project is communicated properly, I will contact my mentors at least once a week to update them on how the work is progressing, discuss current problems and ask for their suggestions for the problems at hand.

Development Experience

Internship experience

Taichi Graphics

Software Engineer Intern

Taichi Graphics is the company behind Taichi Lang, one of the most popular parallel programming languages for high-performance numerical computation. (21.7k Github stars)
Developed an autodiff system for ndarray data structure, as well as a gradient-check feature. Improved gradient quality and availability of the autodiff module, making Taichi Lang a more powerful tool to build high-performance programs.

Research experience

Tsinghua University
Research Intern

Learn and study parallel programming, distributed computing, compiler design, etc.
Working on accelerating FasterMoE, a mixture-of-experts deep learning system with multiple GPUs.

Competition experience

Tsinghua University SCC(Student Cluster Competition) Team

Took responsibility for the coding challenge part of the ISC 2022 Student Cluster Competition. Altered the Xcompact3d FORTRAN code to support NVIDIA DPU with the non-blocking collective offload.

Why this project

I love research that has a practical impact in our real world while also being related to mathematics. I believe that the clad project fits these criteria and is elegant. So I hope to explore the system and apply some of the skills I gained from previous experiences to this project.

Feasibility of the project

The programming environment for clad is already fully set up, and the examples, tests and benchmarks in the repository, as well as my own modifications, are all working properly.

I have worked on another AD system through my internship in Taichi Graphics. Their system was based on source code transformation in compilation time, which is similar to clad, and I added support for numerical check and built a source code transformation AD system for their new "ndarray" data structure.

I have read and run the existing clad source code and submitted some PRs, thus

having a good understanding of the system.

I have had several video meetings with my supervisors to discuss ideas and tasks. The technical details below are the result of many combinations with my supervisors.

About Me

Name: Daemond Zhang
Email: daemondzh@gmail.com
Resume: google doc
Phone number: +86 18755314010
Time Zone: GMT+1
University Name: Tsinghua University