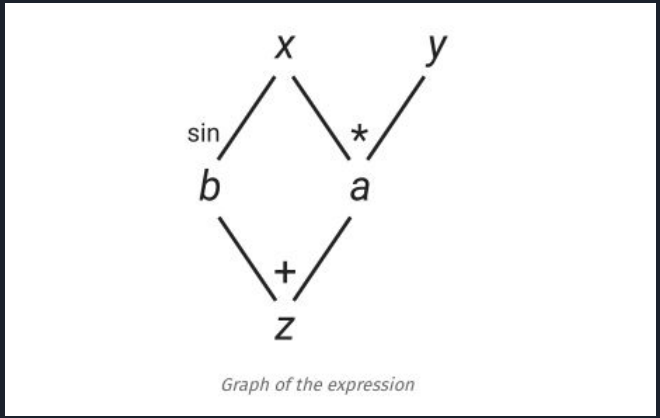
Adding support for differentiating with respect to multi-dimensional arrays(or pointers) in reverse mode.

By - Rishabh Bali

Basics of Automatic Differentiation

- Aims to produce a procedure that calculates the derivative of a given mathematical function w.r.t to one or many input variables

- It does so by breaking down the mathematical function into a computation graph divided into some primitive operations.



Graph of the expression

Eg : Consider the function

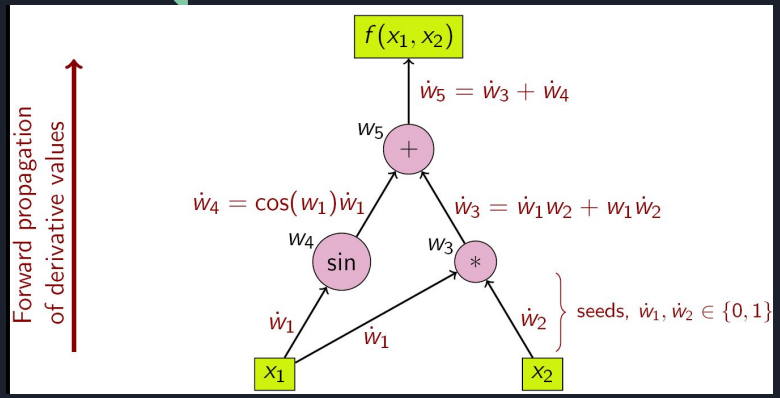
Z = xy + sin(x)

The Chain Rule

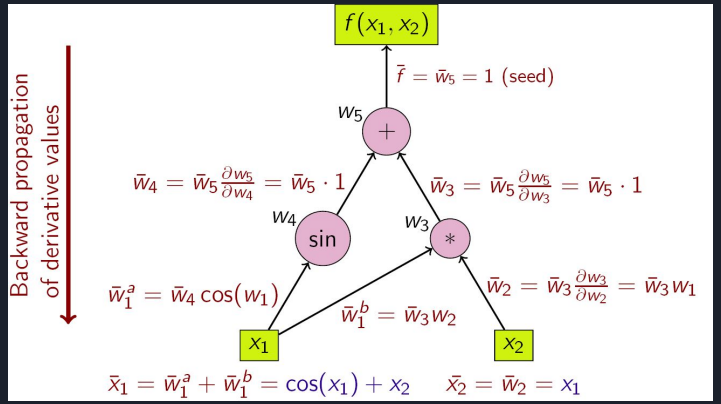
- Fundamental to Automatic Differentiation is the chain rule which helps us calculate the derivative of the dependent variable by calculating the partial derivative of the decomposed functions

$$\frac{\partial w}{\partial t} = \sum_i \left(\frac{\partial w}{\partial u_i} \cdot \frac{\partial u_i}{\partial t} \right)$$
$$= \frac{\partial w}{\partial u_1} \cdot \frac{\partial u_1}{\partial t} + \frac{\partial w}{\partial u_2} \cdot \frac{\partial u_2}{\partial t} + \cdots$$

Forward Vs Reverse Mode AutoDiff



- Flow of derivatives is in the direction of the computation.
- Need to calculate derivative w.r.t each independent variable.



- Flow of derivatives is in the direction opposite to the normal flow of computation.
- Even though we can calculate derivative in one shot we need more memory to store intermediate variables

What is clad and how does it work ?

- Clad enables automatic differentiation of mathematical functions in C++
- It is an open source Clang plugin based on LLVM.
- Clad does this by parsing and transforming the abstract syntax tree (AST).
- Clad support both forward and reverse mode automatic differentiation currently along with computation of hessian and jacobian matrices.

Clad’s API for Reverse Mode AD

A simple example to show differentiation w.r.t all input variables in reverse mode.

```
#include <iostream>
#include "clad/Differentiator/Differentiator.h"

double func(double a, double b) {
return a*b;
}

int main() {
// Use clad::gradient to get the differentiated function;
//here we are differentiating w.r.t to all variables.
auto d_fn = clad::gradient(func);
// Variables to store the derivatives
double d_a = 0;
double d_b = 0;
// Executing the differentiated function
d_fn.execute(/*Value of a*/2, /*Value of b*/3,  &d_a, &d_b);
}
```

Or we can chose the independent variables for differentiation

```
#include <iostream>
#include "clad/Differentiator/Differentiator.h"

double func(double a, double b) {
return a*b;
}

int main() {
// Use clad::gradient to get the differentiated function;
//here we differentiate w.r.t to a
auto d_fn = clad::gradient(func, "a");
// Variable to store derivative w.r.t a
double d_a = 0;
// Executing the differentiated function
d_fn.execute(/*Value of a*/2, /*Value of b*/3,  &d_a);
}
```

Reverse Mode and differentiating w.r.t arrays

Differentiating w.r.t single dimensional arrays

```
#include <iostream>
#include "clad/Differentiator/Differentiator.h"

double fn(double arr[2]) {
    return 2* arr[0] * arr[1];
}

int main() {
// Use clad::gradient to get the differentiated function;
//here we are differentiating w.r.t to all variables.
auto d_fn = clad::gradient(fn);
double arr[2] = {1, 2};
// Empty array to store the derivatives
double d_arr[2] = {0};
// Executing the differentiated function
d_fn.execute(arr, d_arr);
}
```

Task 1 : Enable support for differentiation w.r.t to multi-dimensional arrays in reverse mode.

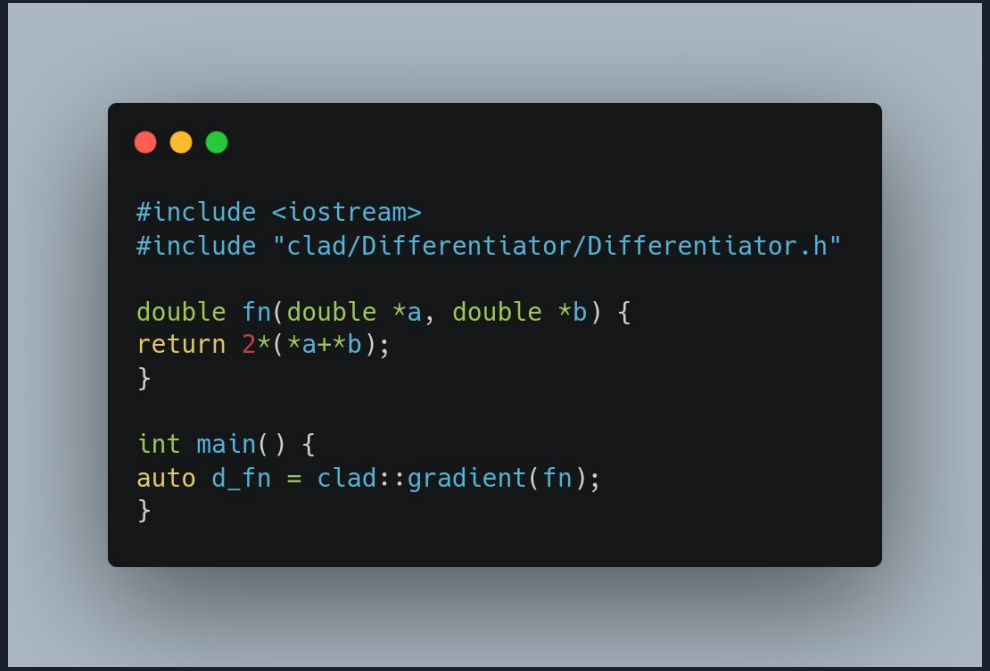
Example for differentiation w.r.t multi-dimensional arrays

```cpp
#include <iostream>
#include "clad/Differentiator/Differentiator.h"

double fn(double arr[5][5]) {
 double res = 1 * arr[0][0] + 2 * arr[1][1] + 4 * arr[2][2];
 return res * 2;
}

int main() {
 auto d_fn = clad::gradient(fn);

 double arr[5][5] = {{1, 2, 3, 4, 5},
                     {6, 7, 8, 9, 10},
                     {11, 12, 13, 14, 15},
                     {16, 17, 18, 19, 20},
                     {21, 22, 23, 24, 25}};

 double d_arr[5][5] = {};
 d_fn.execute(arr, d_arr);
 std::cout << "Derivative of d_fn wrt arr[0][0]: " << d_arr[0][0] << "\n"; // 2
 std::cout << "Derivative of d_fn wrt arr[1][1]: " << d_arr[1][1] << "\n"; // 4
 return 0;
}
```

Task 2 : Add support for differentiating w.r.t pointers in reverse mode

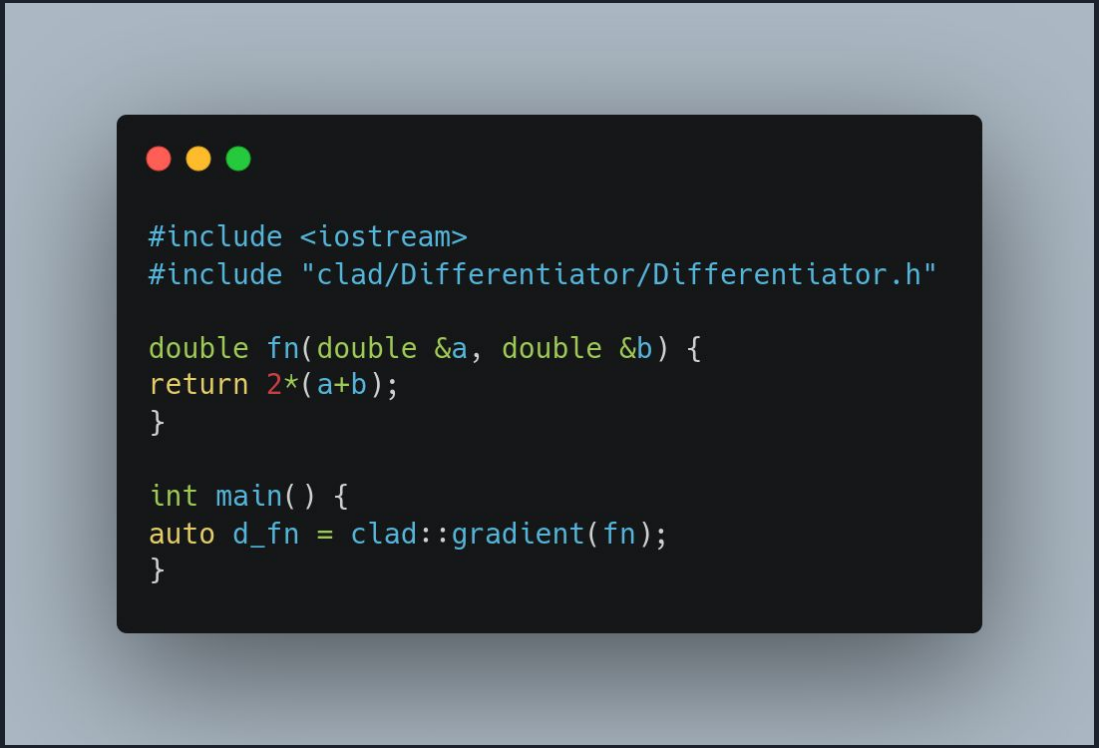
- Reverse Mode in clad doesn’t support differentiation w.r.t pointers.
- The only way around this is to convert pointers to references and then differentiate using clad.

Differentiating w.r.t pointers in reverse mode



```
#include <iostream>
#include "clad/Differentiator/Differentiator.h"

double fn(double *a, double *b) {
return 2*(*a+*b);
}

int main() {
auto d_fn = clad::gradient(fn);
}
```

Alternative Way : Pass variables by Reference



```
#include <iostream>
#include "clad/Differentiator/Differentiator.h"

double fn(double &a, double &b) {
return 2*(a+b);
}

int main() {
auto d_fn = clad::gradient(fn);
}
```

Main Goals of this project :

- Add support for differentiating w.r.t to multidimensional arrays in reverse mode.
- Add support for differentiating w.r.t pointers in reverse mode.
- Support the implementation with tests and documentation.

Thank You