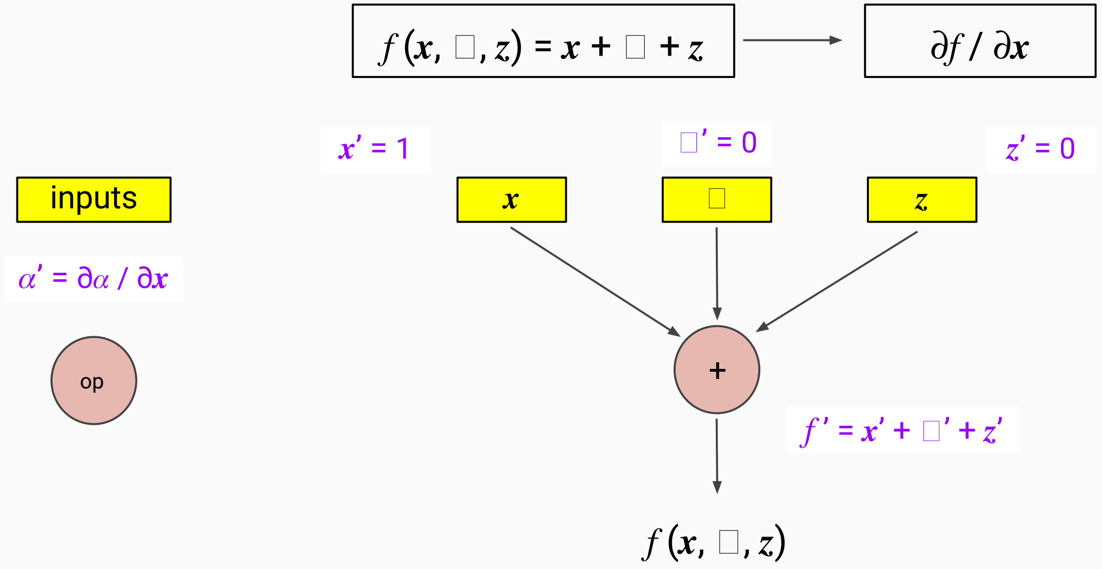
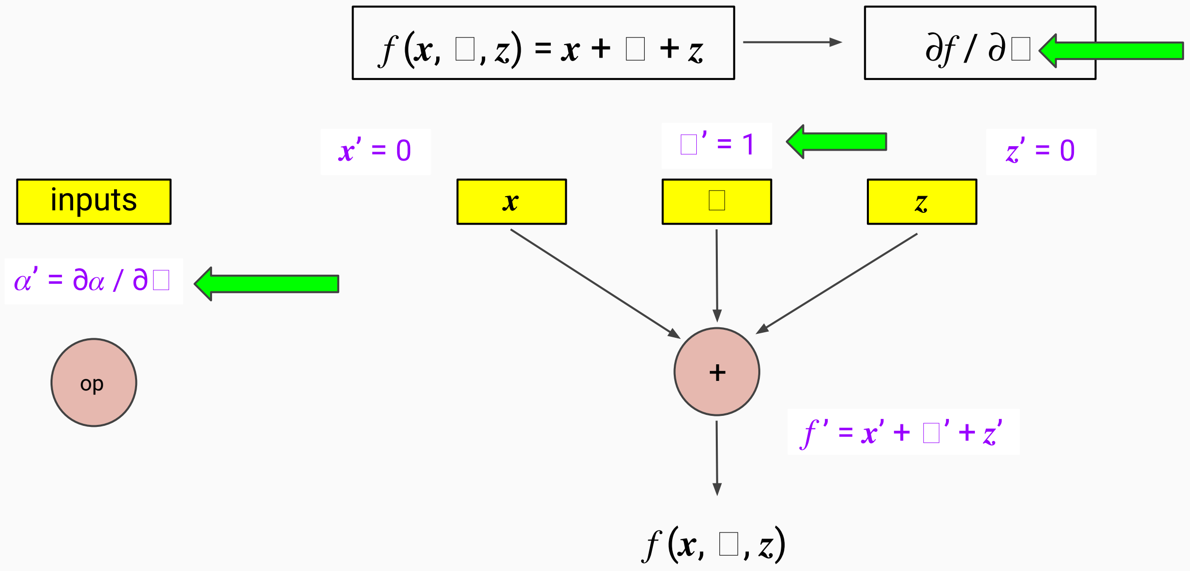
Vectorized forward mode AD in clad

By Vaibhav Thakkar

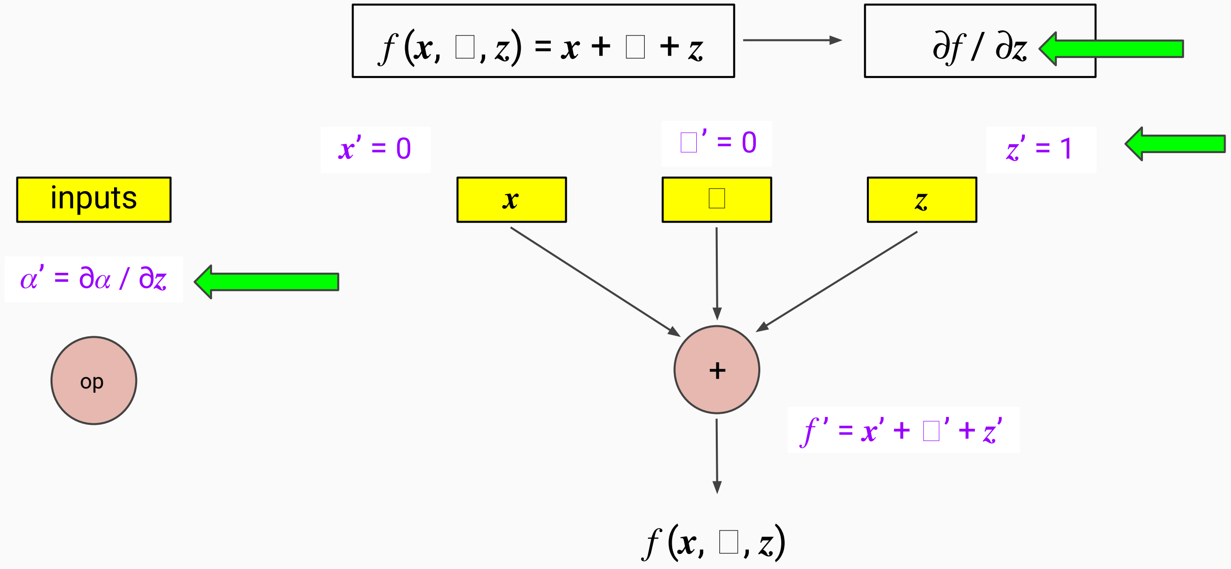
Forward mode AD



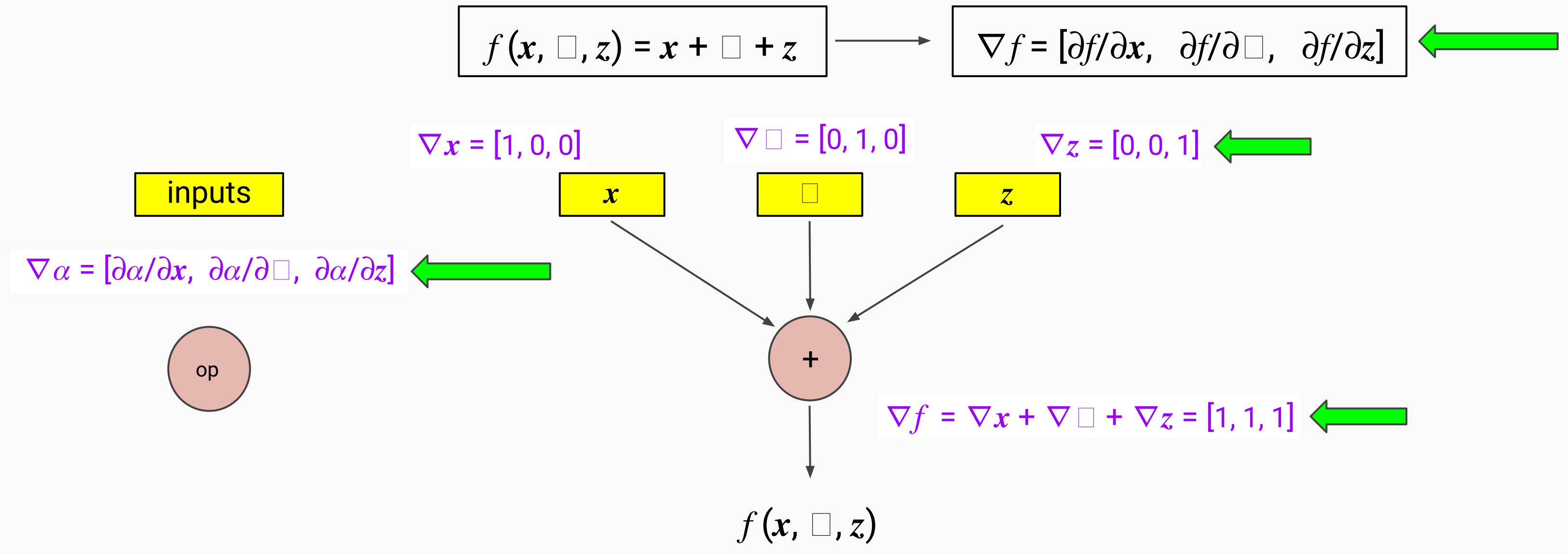
Forward mode AD



Forward mode AD



Vectorized Forward mode AD



$f(x, □, z) = x + □ + z \longrightarrow \boxed{\nabla f = [\partial f/\partial x,\ \partial f/\partial □,\ \partial f/\partial z]}$

$\nabla x = [1, 0, 0]$

$\nabla □ = [0, 1, 0]$

$\nabla z = [0, 0, 1]$

$\nabla \alpha = [\partial \alpha/\partial x,\ \partial \alpha/\partial □,\ \partial \alpha/\partial z]$

$\nabla f = \nabla x + \nabla □ + \nabla z = [1, 1, 1]$

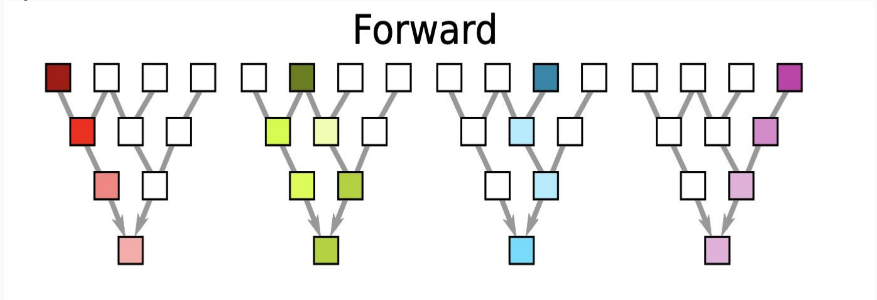
$f(x, □, z)$

Vectorized Forward Mode AD

Problem

For computing gradient of a function with *n*-dimensional input - forward mode requires *n* forward passes, 1 for each input.

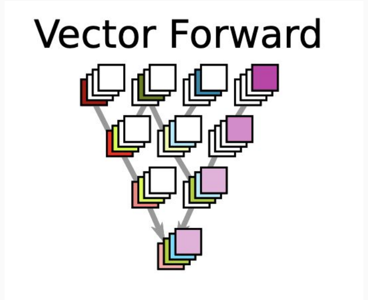
Can we instead compute the complete gradient in one pass?



Proposed Solution

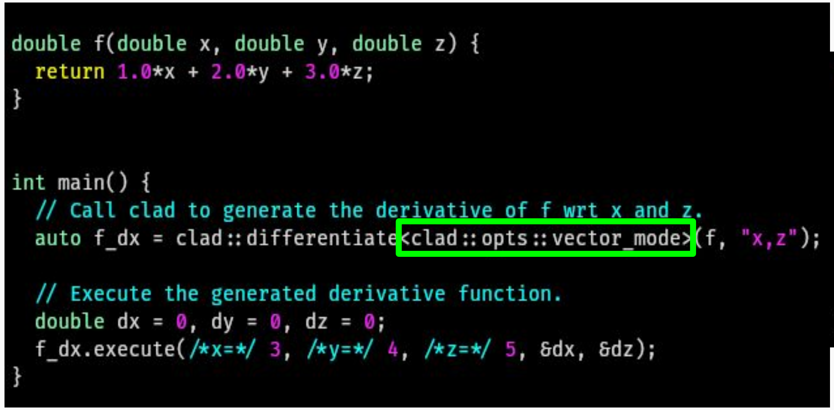
Instead of accumulating a single scalar value of derivative with respect to a particular node - maintain a gradient vector at each node.

Initialised by a 1-hot vector for each input node

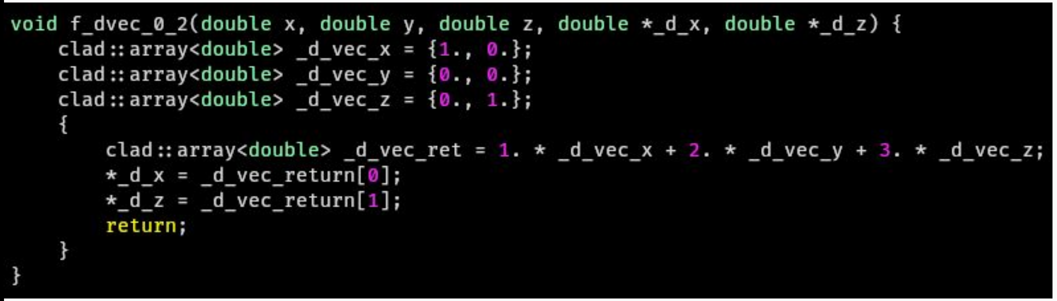


Progress till now

Updated clad interface

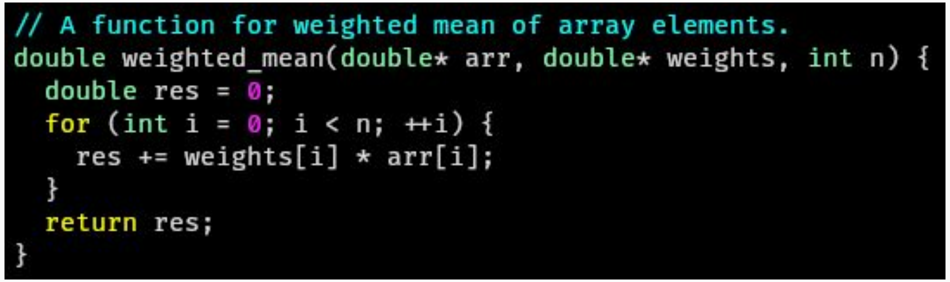


```cpp
double f(double x, double y, double z) {
  return 1.0*x + 2.0*y + 3.0*z;
}


int main() {
  // Call clad to generate the derivative of f wrt x and z.
  auto f_dx = clad::differentiate<clad::opts::vector_mode>(f, "x,z");

  // Execute the generated derivative function.
  double dx = 0, dy = 0, dz = 0;
  f_dx.execute(/*x=*/ 3, /*y=*/ 4, /*z=*/ 5, &dx, &dz);
}
```



```cpp
void f_dvec_0_2(double x, double y, double z, double *_d_x, double *_d_z) {
    clad::array<double> _d_vec_x = {1., 0.};
    clad::array<double> _d_vec_y = {0., 0.};
    clad::array<double> _d_vec_z = {0., 1.};
    {
        clad::array<double> _d_vec_ret = 1. * _d_vec_x + 2. * _d_vec_y + 3. * _d_vec_z;
        *_d_x = _d_vec_return[0];
        *_d_z = _d_vec_return[1];
        return;
    }
}
```

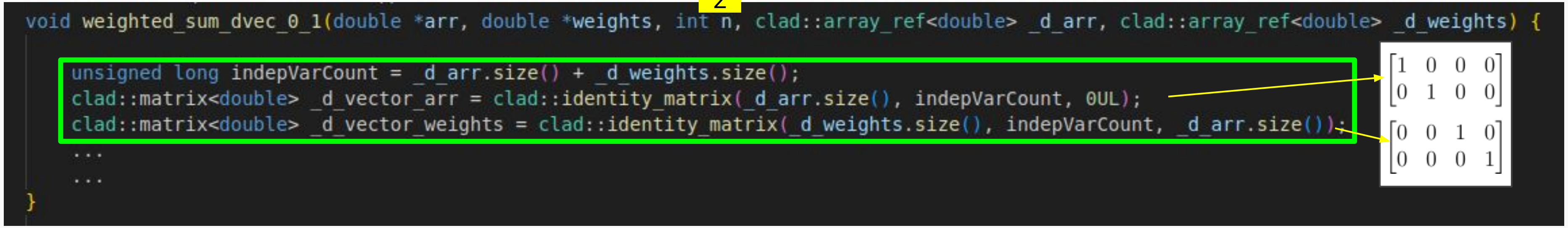
Differentiating array parameters



```
// A function for weighted mean of array elements.
double weighted_mean(double* arr, double* weights, int n) {
  double res = 0;
  for (int i = 0; i < n; ++i) {
    res += weights[i] * arr[i];
  }
  return res;
}
```

- Each arr[i] is a separate independent variable which needs to maintain a vector - this means we need a matrix to store ***_d_vector_arr***.
- Can be multiple array parameters, so multiple matrix instances.

2



```
void weighted_sum_dvec_0_1(double *arr, double *weights, int n, clad::array_ref<double> _d_arr, clad::array_ref<double> _d_weights) {

    unsigned long indepVarCount = _d_arr.size() + _d_weights.size();
    clad::matrix<double> _d_vector_arr = clad::identity_matrix(_d_arr.size(), indepVarCount, 0UL);
    clad::matrix<double> _d_vector_weights = clad::identity_matrix(_d_weights.size(), indepVarCount, _d_arr.size());
    ...
    ...
}
```

$$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\end{bmatrix}$$

$$\begin{bmatrix}0 & 0 & 1 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

Major Features added

- Support for vectorized forward mode for functions containing any of the following:
  - Arithmetic operations
  - Variable assignments
  - Control flow (if statements / loops)
- Restructured ForwardModeVisitor classes to separate out the logic from basic forward mode AD.
- Improved the interface of *clad::differentiate* to take bit-masked options and allowing user to specify multiple input params for differentiation.
- Fixed all LLVM assertions errors when using vector mode
  - Required generating an overload function

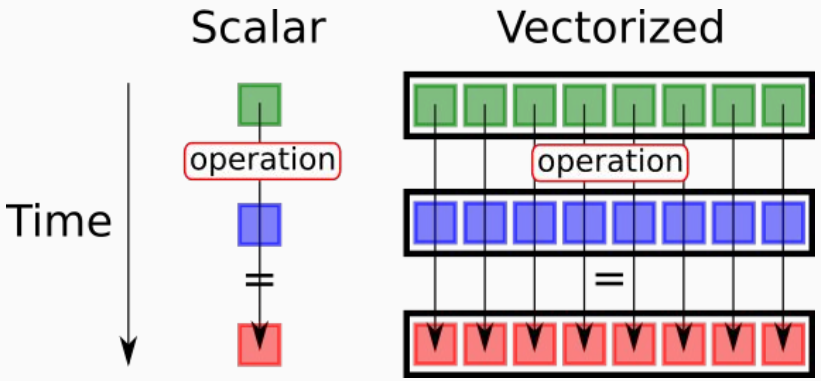
Major Features added

- Adding support for differentiation array parameters
  - Required adding a clad::matrix class along with benchmarks.
- Documentation and demo examples for vector mode.
- Some utilities like adding clang-format and clang-tidy in GitHub checks to ensure code quality.

Next Goal

Improving efficiency

- Current implementation is for vectorization at algorithmic level.
    - To achieve performance speedups - we need to perform operations in parallel at hardware level by instructing the compiler that it is safe to vectorize these operations.



Future Goals

Missing features

- Adding support for differentiating function with call expressions.
    - *std::exp, std::sin, … custom_defined_fn (x, y, z)*
- Object oriented feature support - differentiating methods and functors.
- Improving compute and memory efficiency by activity analysis (enzyme also does this).
- Reverse vector mode.
    - *General* reverse mode AD - traverse from single output to all inputs.
    - *Vectorized* reverse mode AD - traverse from multiple output to all inputs.

Questions ?