A Fast Quantum Circuit Simulator

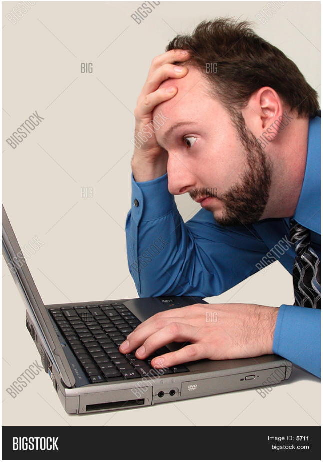
Pagonis Alexandros

Hack-Q-Thon (Q-munity)

May 29, 2022

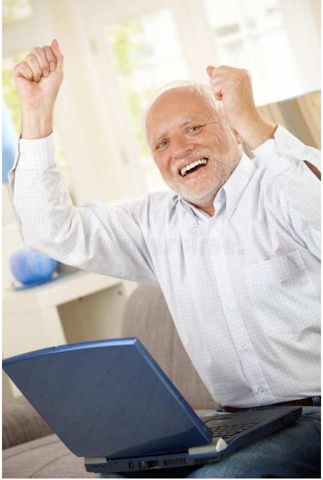
A fast simulator

- Are you tired of your slow python-written Quantum Simulator?
- 10 Qubits seem like a pipe-dream to you?
- Do you want to optimize your Quantum Circuits in a jiffy?



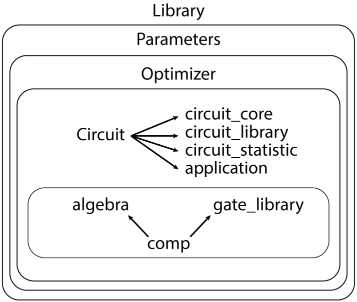
Designing Principles

1. Choose a fast language: C++ seems perfect.
2. Keep it simple (imitate PennyLane's syntax)
3. Keep it flexible and organized.
4. Make it open-source
5. Make it fast. Squeeze every bit of performance.



What it contains

1. A fast complex number and linear algebra library
2. A basic gate library
3. A circuit class with core utilities
4. A circuit back-propagation algorithm
5. Built-in Circuit Optimizers



What do we need? (Part 1)

Compress the layers into as few as possible. Keep track of qubits' last available position. Push everything to the left.

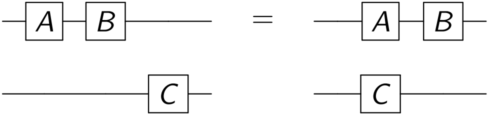


Figure: Layer compression

What do we need? (Part 2)

A **permutation** algorithm to apply layer-gate on our state vector (fill empty ”wires” with Identity)

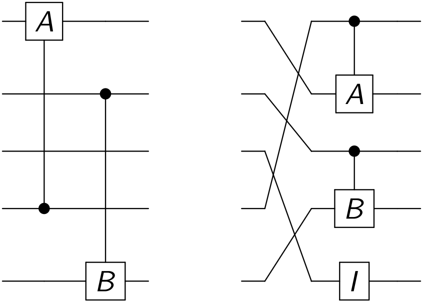


Figure: Permutation Algorithm

What do we need? (Part 3)

A good **Back-Propagation** Algorithm for parametric circuit optimization

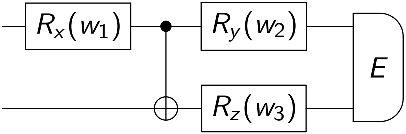
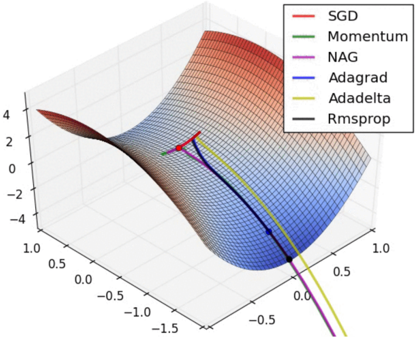


Figure: Find ∇E

What do we need? (Part 4)

Built-in **optimizers** (Adam, GD, Adagrad for the moment)



Example (Part 1)

Define parameters. Use an extra qubit to produce pseudo-non-linearity

```
#define N_QUBITS 5        // 4+1 ancillary
#define DEPTH        6
#define MY_EPOCHS 2500

// predefined in gate_library
SqMatrix Y = matrix_Y;
SqMatrix X = matrix_X;
SqMatrix Z = matrix_Z;
SqMatrix I = matrix_I;
```

Example (Part 2)

Construct H2 Hamiltonian (Operator '%' is tensor product)

```
SqMatrix Hamiltonian =
  0.04523279994605781 * Y%Y%Y%Y
+ 0.04523279994605781 * X%X%Y%Y
+ 0.04523279994605781 * Y%Y%X%X
+ 0.04523279994605781 * X%X%X%X
- 0.8105479805373281  * I%I%I%I
- 0.2257534922240358 * Z%I%I%I
+ 0.1721839326191543 * I%Z%I%I
+ 0.1209126326177663 * Z%Z%I%I
- 0.2257534922240358 * I%I%Z%I
+ 0.1746434306830043 * Z%I%Z%I
+ 0.1661454325638241 * I%Z%Z%I
+ 0.1721839326191543 * I%I%I%Z
+ 0.1661454325638241 * Z%I%I%Z
+ 0.1689275387008790 * I%Z%I%Z
+ 0.1209126326177663 * I%I%Z%Z;
```

Example (Part 3)

Construct Neural Network with alternating layers

```
// Param Layer / Entang Layer...
// --RY--o---X-----RY--o---X--------
// --RY--X-o-|-----RY--X-o-|---etc--
// --RY----X-o-----RY----X-o--------
void make_circuit(Circuit &c, std::vector<c_type> weights) {
        for(unsigned int i = 0; i < DEPTH; ++i) {
                // Parameter Layer
                for(unsigned int j = 0; j < N_QUBITS; ++j)
                        c.RY(j, weights[i*DEPTH + j]);
                // Entanglement Layer
                for(unsigned int j = 0; j < N_QUBITS-1; ++j)
                        c.CZ(j, j+1);
                c.CZ(N_QUBITS-1, 0);
        }

}
```

Example (Part 4)

Pass it to ADAM optimizer

```
int main()
{
    // initialize random weights
        std::vector<c_type> weights = random_weights(N_QUBITS * DEPTH);

        // optimize
        // qubits, layout, Observable, Obs_qubits,
        // initial weights, epochs
        weights = ADAM( N_QUBITS, make_circuit,
                        Hamiltonian, {0,1,2,3}, weights , MY_EPOCHS);


        return 0;
}
```

Example (Part 5)

Compile code:

```
/tests$ g++ -O3 H2_hamiltonian.cpp
/tests$ time ./a.out
```

Execute and measure time:

```
Epoch:  2410    <E>: -1.7521
Epoch:  2420    <E>: -1.75228
Epoch:  2430    <E>: -1.7525
Epoch:  2440    <E>: -1.75278
Epoch:  2450    <E>: -1.75311
Epoch:  2460    <E>: -1.75349
Epoch:  2470    <E>: -1.75391
Epoch:  2480    <E>: -1.75439
Epoch:  2490    <E>: -1.75491
Epoch:  2500    <E>: -1.75548

real    0m0.575s
user    0m0.566s
sys     0m0.009s
```

Figure: Wow..

End

Thank you for your time