FinLin: A C++ Library for Finite Linear Algebra

Thomas Kaldahl
Rice University
MATH 354
12–4–2020

1 Preface

Originally, Camille Kao and I initially elected to produce and demonstrate an artificial intelligence trained by machine learning, capable of recognizing handwritten digits. Given the high work load combined with the circumstances created by the pandemic, we decided that this was no longer possible. I however still had already created from scratch a library for C++ capable of performing calculations with vectors and matrices. I present this along with a description of machine learning.

2 What is machine learning?

Machine learning is a hot topic of modern computer science. It is used to create artificial intelligences capable of categorizing data and solving problems that would normally be very difficult to program by hand.

A program is a function; it takes an input and outputs something in return. Machine learning is performed by creating a scheme to describe a wide variety of functions with a set of numerical parameters that can be adjusted to make the function output what is needed based on its input. A variety of algorithms may be used to produce a set of parameters that is optimal.

2.1 Neural networks

One of the most common schemes for creating such arbitrary programs is known as a neural network. While common diagrams often resemble some sort of network, the math in the background is really just linear algebra.

Firstly, a set of inputs can be described as a vector. An image, for example, can be encoded as a list of numbers — each describing a pixel's brightness — which can then become the components of a vector. The neural network then applies a matrix transformation on this vector to get a new vector. This resulting vector is then fed through a non-linear function that acts on each of the components separately. This process is then repeated, applying matrix transformations and then the non-linear function until a final result is obtained.

The use of a simple non-linear function ensures that the function can extend beyond merely being linear, but the fact that the neural network is largely just linear algebra makes it very fast to compute.

3 Optimizing speed

A computer's central processing unit (CPU) is what most computer code has to go through to be executed. However, most computers today only have around 8 threads — components that can execute the code at any given time. The central nature of the CPU allows it to perform complex calculations, but there is another way.

The graphics processing unit (GPU) is, as the name suggests, intended for processing graphics. Computer monitors have a vast array of pixels that need to be individually. If the CPU did all of the work to determine the correct values, the entire computer would have to wait for the screen to be rendered before doing any other tasks. The GPU is therefore designed to perform on the order of thousands of calculations at a time, in parallel. The calculations the GPU can perform are more limited, but the power comes from the number of threads it has.

3.1 Linear Algebra

Simple algorithms for multiplying a matrix and a vector are often iterative. The CPU does the work of going through, component by component, determining the output of the product. A more complex algorithm can be made to utilize the GPU, assigning one component of the vector to one thread of the GPU. That way, the time it takes to perform a matrix multiplication is only dependent on how much time it takes to calculate one component, not all.

4 FinLin

I have developed FinLin, a library for C++, capable of performing various linear algebra calculations. It takes advantage of the computer's GPU using the program OpenCL.

4.1 Scalars

FinLin works with complex fields. The library first defines various operations on complex scalars, like the addition, multiplication, the complex conjugate, square root, and more. On the next page are some scalar operations, as outputted.

Source code:

```
Sca a = Sca(2, 3);
Sca b = Sca(4, -3);
printf("a is equal to %s\n", a.string());
printf("b is equal to %s\n", b.string());
printf("a + b = %s\n", (a + b).string());
printf("a - b = %s\n", (a - b).string());
printf("a * b = %s\n", (a * b).string());
printf("a / b = %s\n", (a / b).string());
printf("a ^ b = %s\n", a.pow(b).string());
printf("conjugate of a = %s\n", a.conj().string());
printf("sqrt(a + b) = %s\n", (a + b).sqrt().string());
```

Result:

```
a is equal to 2 + 3i
b is equal to 4 + -3i
a + b = 6
a - b = -2 + 6i
a * b = 17 + 6i
a / b = -0.040 + 0.720i
a ^ b = 3212.384 + 269.671i
conjugate of a = 2 + -3i
sqrt(a + b) = 2.449
```

4.2 Vectors

A type is used for abstraction, the "Scalar List." It is the type that is used to build the vector type, which is capable of handling vector addition, scaling by a vector, and standard inner product calculations.

These calculations are done with the computer's GPU. For each component in a vector addition, there is one GPU thread calculating the sum.

For inner products, one thread calculates the product between two components, then an algorithm uses the GPU threads to repeatedly cut the size of the list of numbers in half, preserving the sum of the numbers in the list, until the sum is actually computed.

Source code:

```
Sca a = Sca(5, 4);
ScaList l = ScaList();
l += Sca(1, 2);
l += Sca(2, 1);
l += Sca(-1, 2);
ScaList m = ScaList();
m += Sca(4, 2);
m += Sca(2, -1);
m += Sca(-1, 1);
Vec v = Vec(l);
Vec w = Vec(m);
printf("a is equal to %s\n", a.string());
printf("v is equal to %s\n", v.string());
printf("w is equal to %s\n", w.string());
printf("a v = %s\n", (a * v).string());
printf("v + w = %s\n", (v + w).string());
printf("v - w = %s\n", (v - w).string());
printf("||v|| = %s\n", v.norm().string());
printf("w normalized = %s\n",
w.normal().string());
printf("<v, w> = %s\n", (v * w).string());
```

Result:

```
a is equal to 5 + 4i
v is equal to <1 + 2i, 2 + 1i, -1 +
2i>
w is equal to <4 + 2i, 2 + -1i, -1 +
1i>
a v = <-3 + 14i, 6 + 13i, -13 + 6i>
v + w = <5 + 4i, 4, -2 + 3i>
v - w = <-3, 0 + 2i, 0 + 1i>
||v|| = 2.900 + 1.552i
w normalized = <0.907 + 0.305i, 0.394
+ -0.271i, -0.182 + 0.242i>
<v, w> = 9 + 5i
```

4.3 Matrices

In the current iteration of this library, matrices are only capable of transforming vectors. Matrix multiplication, addition, and attributes like determinants and determination of diagonalizability are planned, but not yet implemented.

Source code:

```
double *nums = (double*)malloc(40 *
sizeof(double));
for(int i = 0; i < 40; i++) nums[i] = i;
Mat A = Mat(5, 4, nums);
ScaList l = ScaList();
l += Sca(2, 3);
l += Sca(5, 7);
l += Sca(11, 13);
l += Sca(15, 17);
Vec v = Vec(l);
printf("v is equal to %s\n", v.string());
printf("A is equal to %s\n", A.string());
printf("A v = %s\n", (A * v).string());
```

Result:

```
v is equal to <2 + 3i, 5 + 7i, 11 + 13i, 15 + 17i>
A is equal to [
 0 + 1i   2 + 3i   4 + 5i   6 + 7i
 8 + 9i   10 + 11i   12 + 13i   14 + 15i
 16 + 17i   18 + 19i   20 + 21i   22 + 23i
 24 + 25i   26 + 27i   28 + 29i   30 + 31i
 32 + 33i   34 + 35i   36 + 37i   38 + 39i
]
A v = <-64 + 345i, -120 + 929i, -176 + 1513i, -232 +
2097i, -288 + 2681i>
```

4.4 Planned features

Given the circumstances, I wasn't able to implement all of the features of this library that were initially planned. Upcoming features include the ability to programmatically test for features of vector lists, for example, determining if a list is a basis, orthonormal, etc. I also wanted to implement the Gram Schmidt algorithm as a usable function in the code. Furthermore, matrix multiplication is something that is difficult to add in, but is necessary to make this library more complete. I do plan to use this library in the future for other projects, and I will likely distribute the code on the internet once I am finished.

4.5 Source Code

While this is not a computer science class, I feel it wouldn't hurt to provide the source code. It is available at `www.thomaskaldahl.com/finlin`