C++ Linear Algebra Library

Luke Wood

Project Description

I began this project in order to begin working on a neural network project in C++. The library currently allows for you to easily create matrices and perform basic operations on these matrix objects.

Implementation

Each matrix object consists of a pointer to an integer pointer, or as some people like to think of it an array of integer arrays. Upon creation of the matrix object, all of the values in the matrix are initialized to zeros. When errors are encountered regarding the dimensions of a matrix, a matrix holding the value [0] is returned.

Code

The Code below displays the implementation full source code of the matrix class as well as various relevant operators.

Matrix.h

```
#ifndef MATRIX_H
#define MATRIX_H
#include <ostream>
using std::ostream;
namespace linear
{

class matrix
{

	friend ostream& operator<<(ostream&,const matrix&);
	friend matrix operator*(const matrix&,const matrix&);
	friend matrix operator*(int scalar, const matrix&);

		private:
			int** rows;
			int height;
			int width;
		public:
			matrix(const matrix&);
			matrix();
			matrix(int,int);
			~matrix();

			int* operator[](int) const;
			matrix operator*(int) const;
			matrix operator+(const matrix&) const;
			matrix operator-(const matrix&) const;
			matrix operator+(int) const;
			matrix operator-(int) const;

			matrix operator=(const matrix&);



			int getHeight() const;
			int getWidth() const;

			int dot(const matrix&) const;

			//Transpose
			matrix T() const;
};

linear::matrix operator*(const linear::matrix&, const linear::matrix&);
linear::matrix operator*(const int, const linear::matrix&);
ostream& operator<<(ostream&,const matrix&);

}
#endif
```

Matrix.cpp

```
#include "matrix.h"
#include <ostream>
using linear::matrix;
using std::ostream;
ostream& linear::operator<<(ostream& stream, const matrix& toprint)
{
	stream<<"[";
	for(int i = 0; i < toprint.getHeight(); i++)
	{
		for(int j = 0; j < toprint.getWidth(); j++)
		{
			if(j!=toprint.getWidth()-1)
			{
				stream<<toprint[i][j]<<" ";
			}
			else
			{
				stream<<toprint[i][j];
			}
		}
		if(i!=toprint.getHeight()-1)
		{
		stream<<"\n ";
		}
	}
	stream<<"]\n";
	return stream;
}

matrix linear::operator*(int scalar, const matrix& tomult)
{
	matrix m(tomult);
	for(int i = 0; i < m.getHeight(); i++)
	{
		for(int j = 0; j < m.getWidth(); j++)
		{
			m[i][j] = m[i][j] * scalar;
		}
	}
	return m;
}

int matrix::getHeight() const
{
	return height;
}
int matrix::getWidth() const
{
	return width;
}

matrix::matrix(int iheight, int iwidth)
{
```

```
        height = iheight;
        width = iwidth;
        rows = new int*[iheight];
        for(int i=0; i < iheight; i++)
        {
                rows[i] = new int[iwidth];
                for( int j = 0; j < iwidth; j++)
                {
                        rows[i][j] = 0;
                }
        }
}

matrix::matrix()
{
        width = 1;
        height = 1;
        rows = new int*[1];
        rows[0] = new int[1];
        rows[0][0] = 0;
}

matrix::matrix(const matrix& copyFrom)
{
        height = copyFrom.getHeight();
        width = copyFrom.getWidth();
        rows = new int*[height];
        for(int i = 0; i < height; i++)
        {
                rows[i] = new int[width];
                for(int j = 0; j < width; j++)
                {
                        rows[i][j] = copyFrom[i][j];
                }
        }
}

matrix matrix::operator+(const matrix& toAdd) const
{
        if(width!= toAdd.width && height != toAdd.height)
        {
                matrix m(height,width);
                for(int i = 0; i < height; i++)
                {
                        for(int j = 0; j < width; j++)
                        {
                                m[i][j] = rows[i][j] + toAdd[i][j];
                        }
                }
                return m;
        }
        return matrix();
}


matrix matrix::operator-(const matrix& toSub) const
{
        if(width== toSub.width && height == toSub.height)
        {
                matrix m(height,width);
```

```
                for(int i = 0; i < height; i++)
                {
                        for(int j = 0; j < width; j++)
                        {
                                m[i][j] = rows[i][j] - toSub[i][j];
                        }
                }
                return m;
        }
        else
        {
                return matrix();
        }
}

matrix matrix::operator+(int toAdd) const
{
         matrix m(height,width);
         for(int i = 0; i < height; i++)
         {
                 for(int j = 0; j < width; j++)
                 {
                          m[i][j] = rows[i][j] + toAdd;
                 }
         }
         return m;
}

matrix matrix::operator-(int toSub) const
{
         matrix m(height,width);
         for(int i = 0; i < height; i++)
         {
                 for(int j = 0; j < width; j++)
                 {
                          m[i][j] = rows[i][j] + toSub;
                 }
         }
         return m;
}

matrix matrix::operator=(const matrix& copyFrom)
{
        if(!(this == &arg))
        {
                for(int i = 0; i < copyFrom.getHeight(); i++)
                {
                        delete[] rows[i];
                }
                delete[] rows;
                height = copyFrom.getHeight();
                width = copyFrom.getWidth();
                rows = new int*[height];
                for(int i = 0; i < height; i++)
                {
                        rows[i] = new int[width];
                        for(int j = 0; j < copyFrom.getWidth(); j++)
                        {
                                rows[i][j] = copyFrom[i][j];
                        }
```

```
                }
        }
        return *this;
}

matrix::~matrix()
{
for(int i = 0; i < height; i++)
{
                delete[] rows[i];
}
delete[] rows;
}

matrix matrix::T() const
{
        matrix m(width, height);
        for(int i =0; i < height; i++)
        {
                for(int j =0; j< width; j++)
                {
                        m[j][i] = rows[i][j];
                }
        }
        return m;
}

int matrix::dot(const matrix& other) const
{
        if(height == other.getHeight() && width == other.getWidth())
        {
                int dotProd = 0;
                for(int i = 0; i< height; i++)
                {
                        for(int j = 0; j < width; j++)
                        {
                                dotProd += (rows[i][j] * other[i][j]);
                        }
                }
                return dotProd;
        }
        return -1;
}

matrix linear::operator*(const matrix& first,const matrix& second)
{
        if(first.getWidth() != second.getHeight())
        {
                matrix m(1,1);
                m[0][0] = 0;
                return m;
        }
        matrix m(first.getHeight(),second.getWidth());
        for(int i = 0; i < first.getHeight(); i++)
        {
                for(int j = 0; j < second.getWidth(); j++)
                {
                        int sum = 0;
                        for(int k = 0; k < second.getHeight(); k++)
                        {
```

```
                                sum = sum + (first[i][k]*second[k][j]);
                        }
                        m[i][j] = sum;
                }
        }
        return m;
}
matrix linear::matrix::operator*(int scalar) const
{
        matrix m(height,width);
        for(int i = 0; i < height; i++)
        {
                for(int j = 0; j < width; j++)
                {
                        m[i][j] = rows[i][j] * scalar;
                }
        }
        return m;
}

int* linear::matrix::operator[](int index) const
{
        return rows[index];
}
```

Results and Discussion

Implementing this matrix class proved to be more difficult than I initially believed that it would be. I encountered various errors typically due to oversights in memory management.

Concluding Remarks

Now that this project is done, I have begun working on my neural network project which I should be finishing in the next few months and will hopefully provide some real world functionality.