Create - Performance Task

2a. In narration of video.

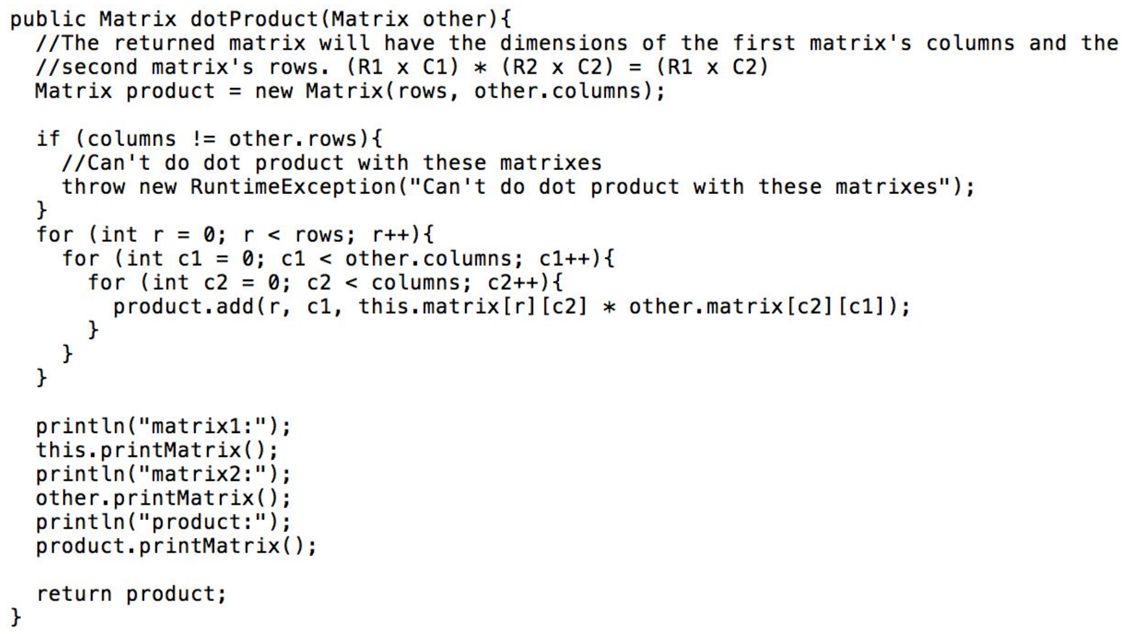
2b.

During development, there were difficulties almost every step. I had to implement version control to track errors at specific stages. Also, for making my program work correctly, I periodically tested individual sections using test inputs and print statements.

The most difficult challenge was programming the graphics for the neural network diagram and tables of values so three layer neural networks of any dimension would be able to call the `displayNeuralNetwork` function and be displayed properly. This section required a particularly large amount of code and I constantly calculated and created mathematical functions for determining the width and height of text boxes and shapes. Throughout debugging, I had to implement test code and inputs to make sure the graphics worked.

I faced another huge difficulty when an error in the Pong game I program made the ball always get stuck in a infinite loop. I spent two long nights testing by disabling parts of code, adding print statements, and writing flowcharts. I resolved the issue by switching the order of two lines correcting the order of execution of the program, and I created less of a delay between the movement of the ball and the calculations that determine the ball's movement.

2c.



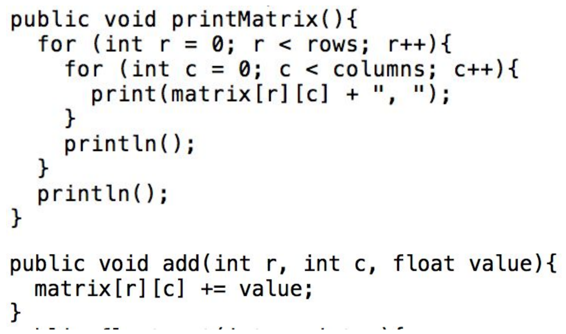
```
public Matrix dotProduct(Matrix other){
  //The returned matrix will have the dimensions of the first matrix's columns and the
  //second matrix's rows. (R1 x C1) * (R2 x C2) = (R1 x C2)
  Matrix product = new Matrix(rows, other.columns);

  if (columns != other.rows){
    //Can't do dot product with these matrixes
    throw new RuntimeException("Can't do dot product with these matrixes");
  }
  for (int r = 0; r < rows; r++){
    for (int c1 = 0; c1 < other.columns; c1++){
      for (int c2 = 0; c2 < columns; c2++){
        product.add(r, c1, this.matrix[r][c2] * other.matrix[c2][c1]);
      }
    }
  }

  println("matrix1:");
  this.printMatrix();
  println("matrix2:");
  other.printMatrix();
  println("product:");
  product.printMatrix();

  return product;
}
```

The `dotProduct` algorithm takes the `matrix` object calling the function and another `matrix` object provided to the function and returns the dot product of the two matrices. In order to find the dot product of the two matrices (essentially two dimensional arrays) the algorithm must first use logic to check if the amount of columns in the first `matrix` equals the amount of rows in the second `matrix`. Then, it uses iteration in three `for` loops to calculate and assign values to the `product matrix`. This algorithm is critical for the program because it feeds forward the data through the neural networks and prints data to the console.

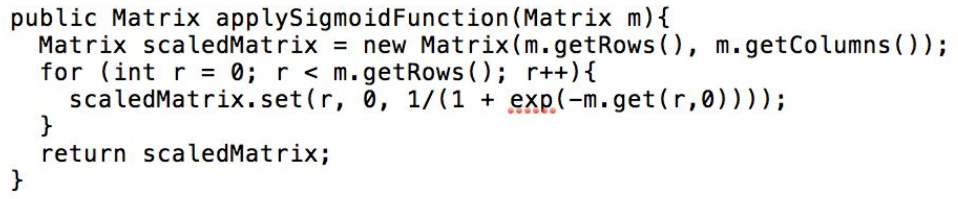


```
public void printMatrix(){
  for (int r = 0; r < rows; r++){
    for (int c = 0; c < columns; c++){
      print(matrix[r][c] + ", ");
    }
    println();
  }
  println();
}

public void add(int r, int c, float value){
  matrix[r][c] += value;
}
```

For this algorithm to function properly it needs to use other functions I programmed. It uses the `printMatrix` function for printing neural network data. This function consists of nested `for` loops which iteratively and logically print every element of the `matrix` object in correct order. Also, in calculating the correct values for the `product matrix`, the algorithm must utilise the `add` function which adds a floating point value given to the function through its parameters to a certain index in the `matrix` object that calls the function.

2d.



```
public Matrix applySigmoidFunction(Matrix m){
  Matrix scaledMatrix = new Matrix(m.getRows(), m.getColumns());
  for (int r = 0; r < m.getRows(); r++){
    scaledMatrix.set(r, 0, 1/(1 + exp(-m.get(r,0))));
  }
  return scaledMatrix;
}
```

One abstraction I programmed to facilitate the complexity of my program is a function called `applySigmoidFunction`. It returns a `matrix` object that has the same dimensions of the `matrix` object calling it and uses the values from the `matrix` object calling it to fill the returned `matrix` object with output values of a sigmoid function. `ApplySigmoidFunction` uses logic when it checks the `matrix` object calling it to make sure the `matrix` object has only one column, so the function isn't called by a `matrix` object responsible for holding the neural network weight values. It also uses mathematics when it inputs values into a sigmoid function and outputs the values into the `matrix` object getting returned. Implementing this abstraction improves the readability of the program and reduces the amount of total code and redundancy. Instead of writing the same lines of code at every place that requires it, I just call the function `applySigmoidFunction`. Furthermore, if I need to add to this type of code or debug it in the future I only have to focus on the body of this function and changes I make there would affect the rest of the program. This increase efficiency of programming the project.