

# Extra Credit Project: Parallelizing Feed Forward Neural Networks

**EXTRA CREDIT, GIVE YOU?**



**PASS CLASS, I MUST!**

1. Project Description

2. Machine Learning Background

3. Deliverables and Submission  
Details

# 1. Project Description

# Project Description

- You are hired as a C programming specialist by a company that does **machine learning**.
- They provide you with the following machine learning code: a feed forward neural network implemented in C.
- The company wants to increase the performance of the model by introducing parallel computing into the code.
- You need to first understand how a feed forward neural network works (by reading these slides).
- You then need to work to parallelize the C code using techniques you learned in this class (either threading or with forks).

## 2. Machine Learning Background

# Understanding Feed Forward Neural Networks


- A feed forward neural network in its most basic form does the following:
  1. Takes in an array as **input**.
  2. Passes the array through different layers of the network.
  3. Produces an array as **output**.

In this project we'll be working with a very basic neural network. However, its important to note the fundamental principles you are learning here are used in state-of-the-art machine learning models like ChatGPT.

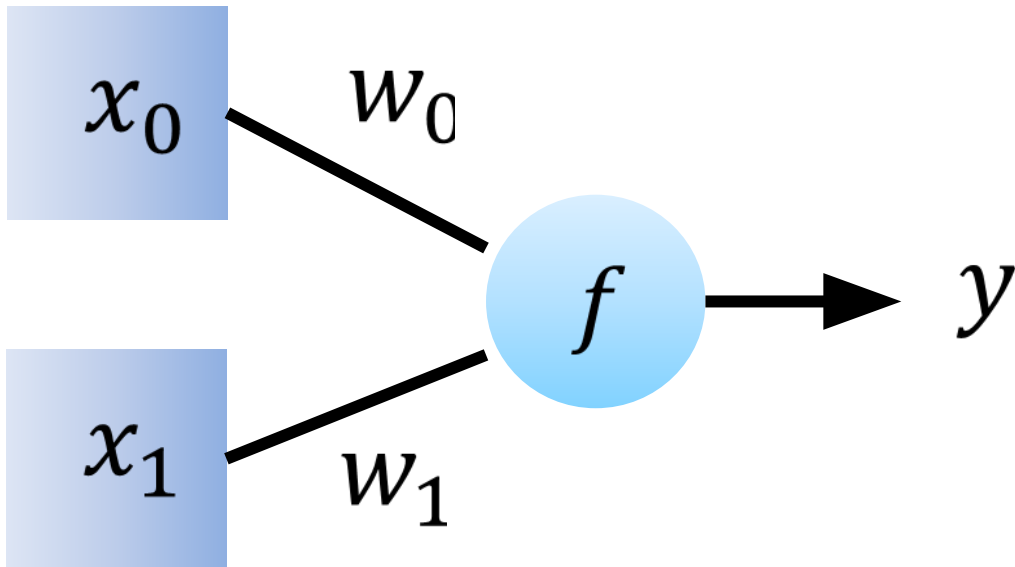
# Side Note: Drawing Machine Learning Models

 = Input to our model,  $x_0$

 = Connection Weight,  $w_0$

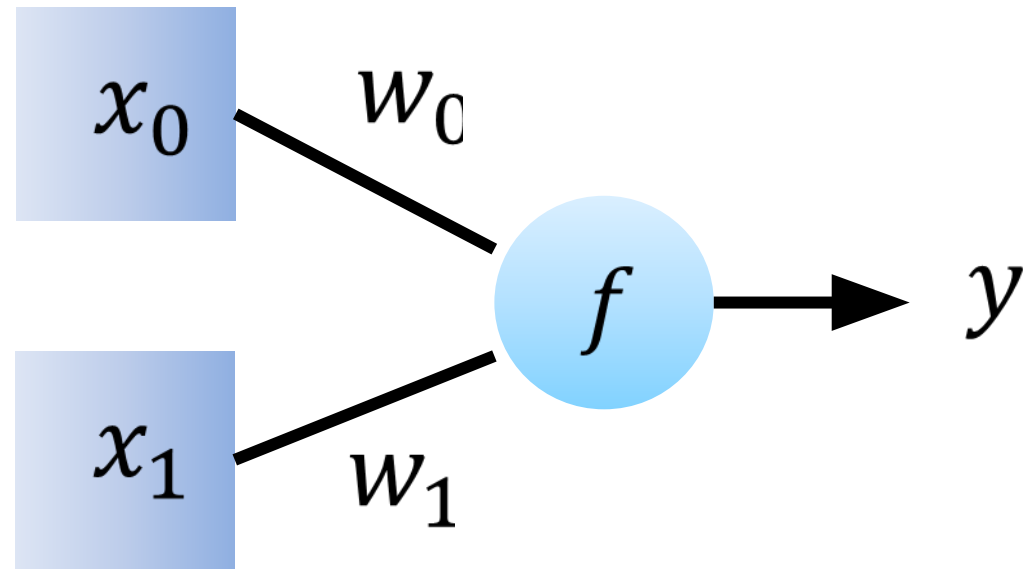
 = Summation AND  
Activation Function,  $f$

# The Basic Neuron Structure



- Every neural network is made up of neurons.
- The neurons take in an input.
- A weighting is applied to the input.
- Then an activation function is applied.
- A single output is produced.

# The Basic Neuron Structure Math



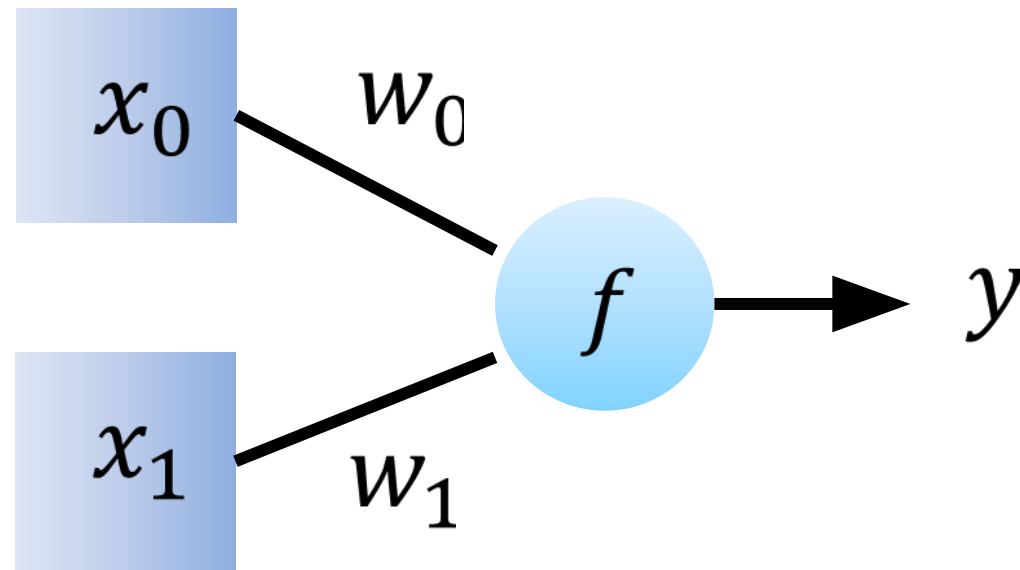
$$y = f(x_1, x_2)$$

$$y = f(x_1 w_1 + x_2 w_2)$$



For function  $f$  we use sigmoid so that output  $y$  is:

$$y = \frac{1}{1 + e^{-x_0 w_0 - x_1 w_1 - b}}$$



$$f = \text{Sigmoid}$$

# Neuron Structure in C:

```
//neuron struct
typedef struct {
    int inputSize;
    double* w; //weight parameter
    double b; //bias parameter
} neuron;
```

# Building a Single Neuron :

- A single neuron is created with randomly initialized weights and a bias term using the following function:

```
//Build the neuron
neuron InitializeNeuron(int inputSize){
    double* w = (double*)malloc(inputSize * sizeof(double));
    double b = randomRangeZeroAndOne();
    //start the network with random values
    for(int i=0;i<inputSize;i++){
        w[i] = randomRangeZeroAndOne();
    }
    neuron n = {inputSize, w, b};
    return n;
}
```

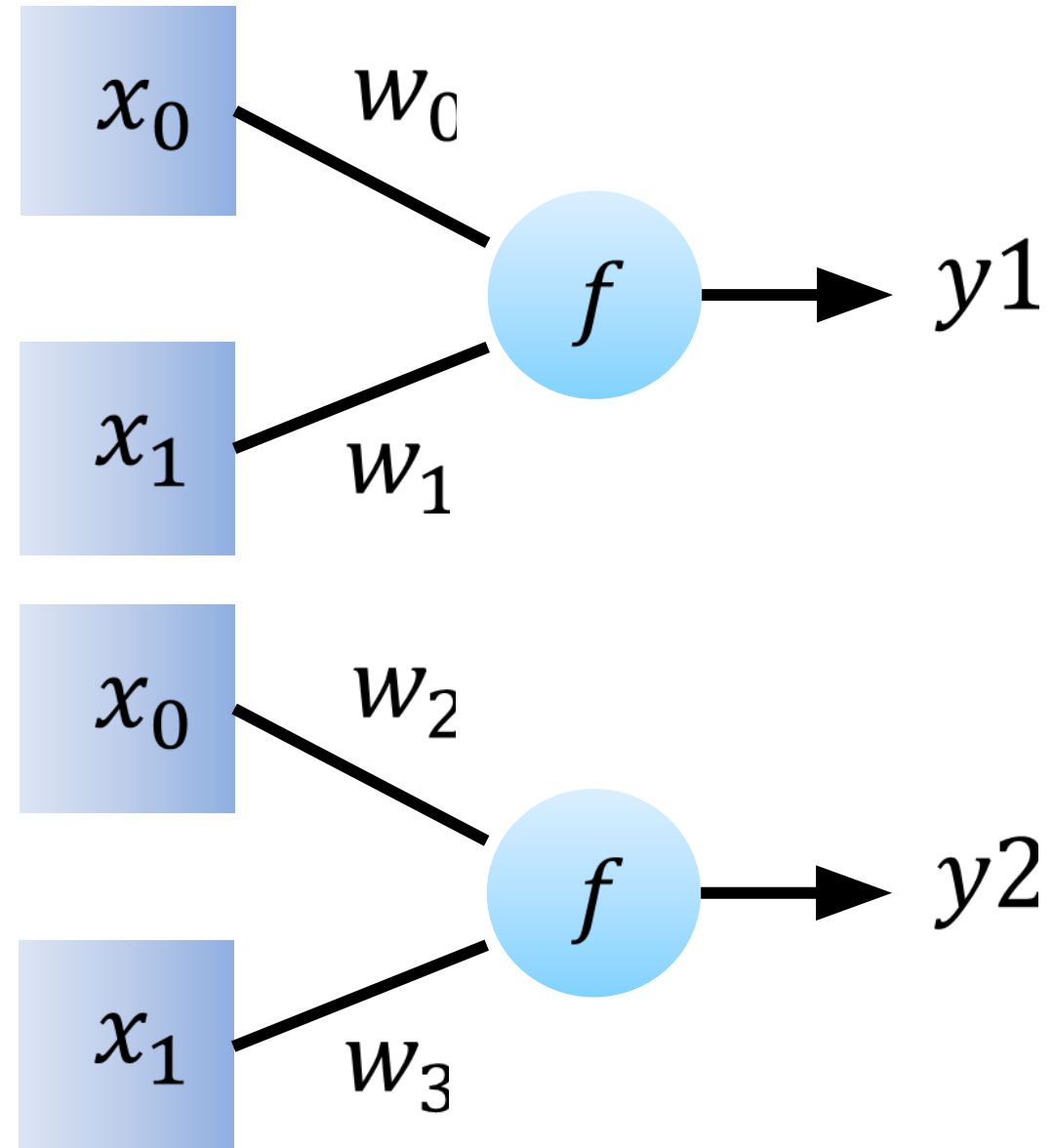
# Doing a forward pass on a single neuron:

- Once the neuron is created, we want to do what is called a forward pass: we apply some **input** to the neuron compute some **output**.
- See the equation in slide 9 for an example of this when we only have two inputs.

```
//compute forward pass on the neuron
double ForwardPassNeuron(neuron* n, double* x){
    //Get the input size
    int inputSize = (*n).inputSize;
    //memory for the solution
    double output = 0;
    //first apply the weight
    for(int i = 0; i < inputSize; i++){
        output = output + (*n).w[i]*x[i];
    }
    //apply the bias
    output = output + (*n).b;
    //apply an activation function
    output = 1/(1+exp(-output));
    return output;
}
```

# Making a **layer** of neurons

- We can then put multiple neurons together to make a layer.
- Each neuron takes the SAME input but can have different weights and biases that it applies to the input.

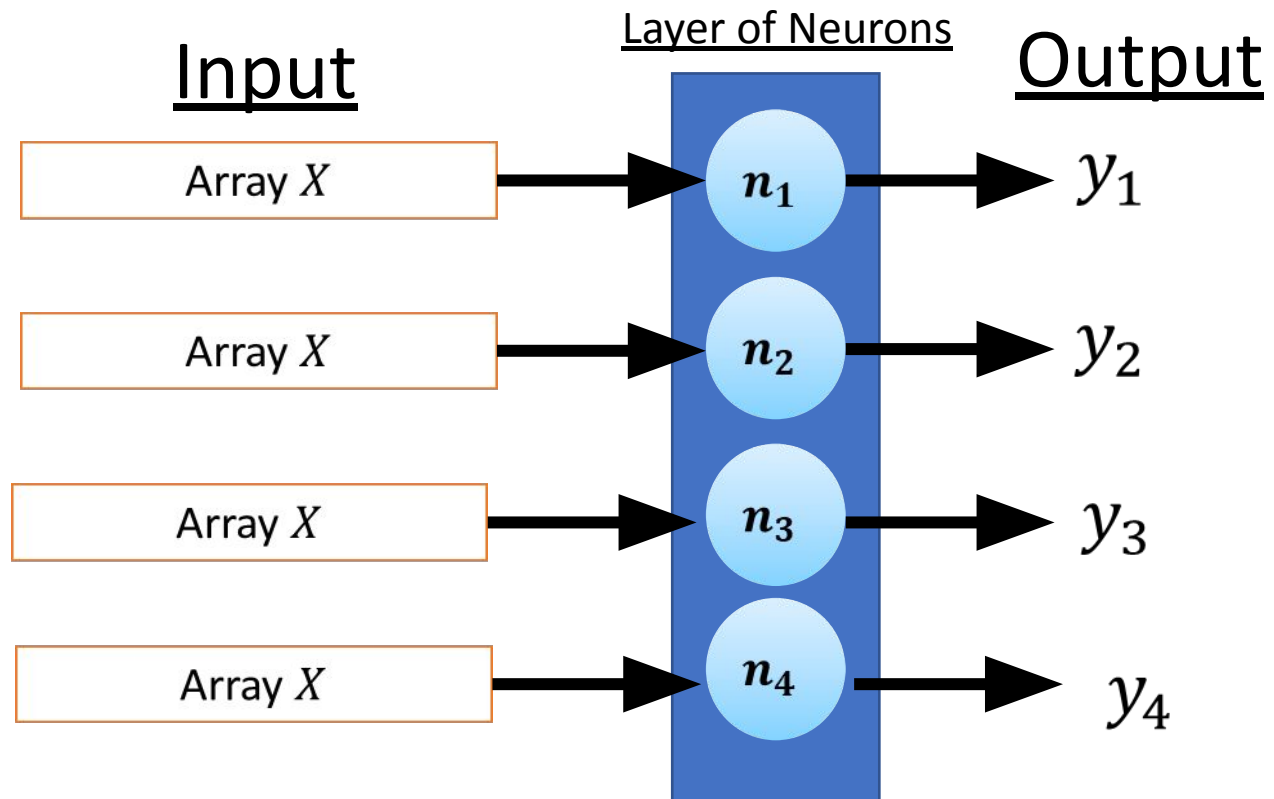


# Layer Structure in C:

```
//layer struct
typedef struct {
    int inputSize;
    int numNeurons;
    neuron* nArray;
} layer;
```

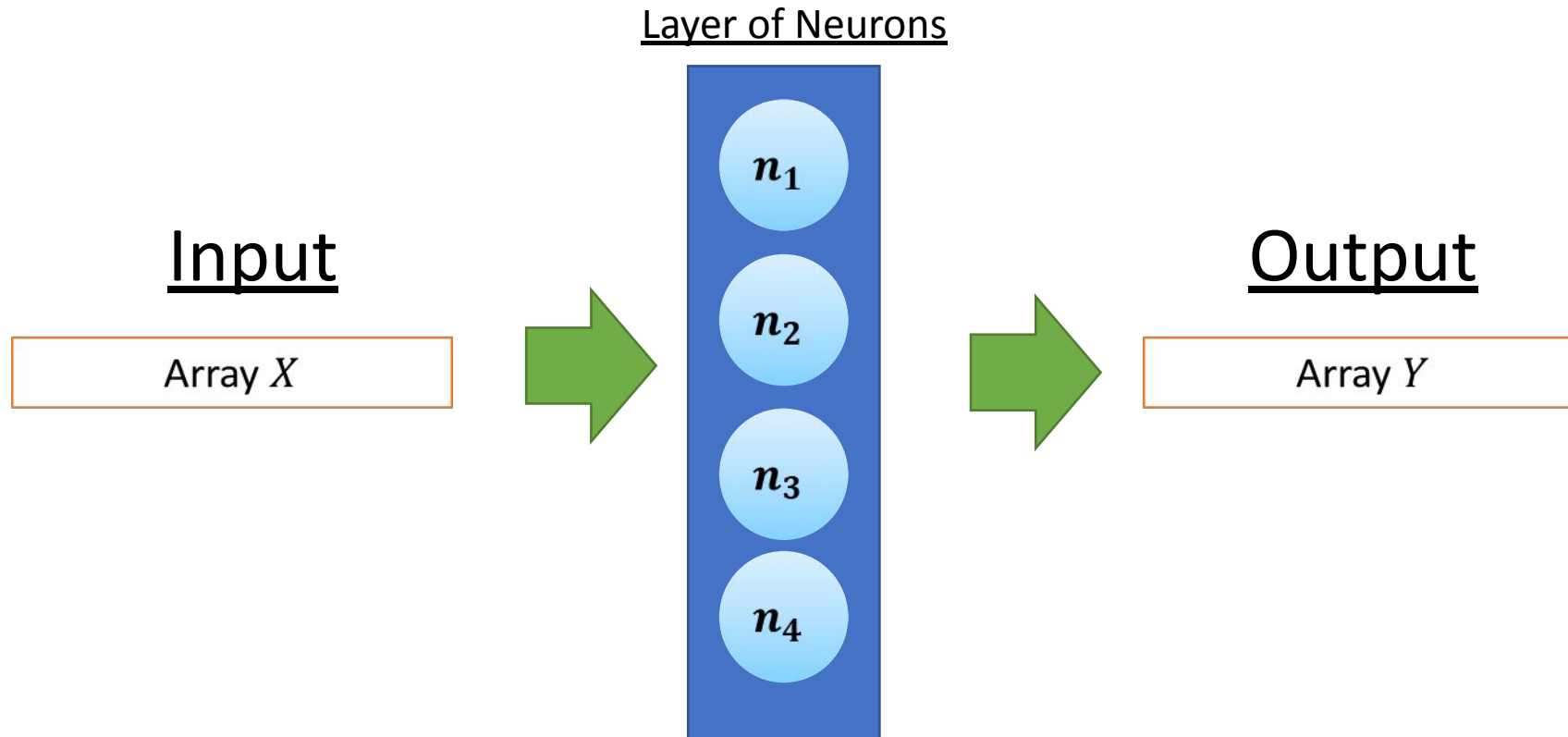
# Layers of Neurons

- The basic idea is that a layer takes in an array as input.
  - The array is passed through each neuron.
  - Each neuron produces one output value.



# Layers of Neurons

- We can simplify the picture a little bit and simply say a layer takes in an array as input and produces an array as output:



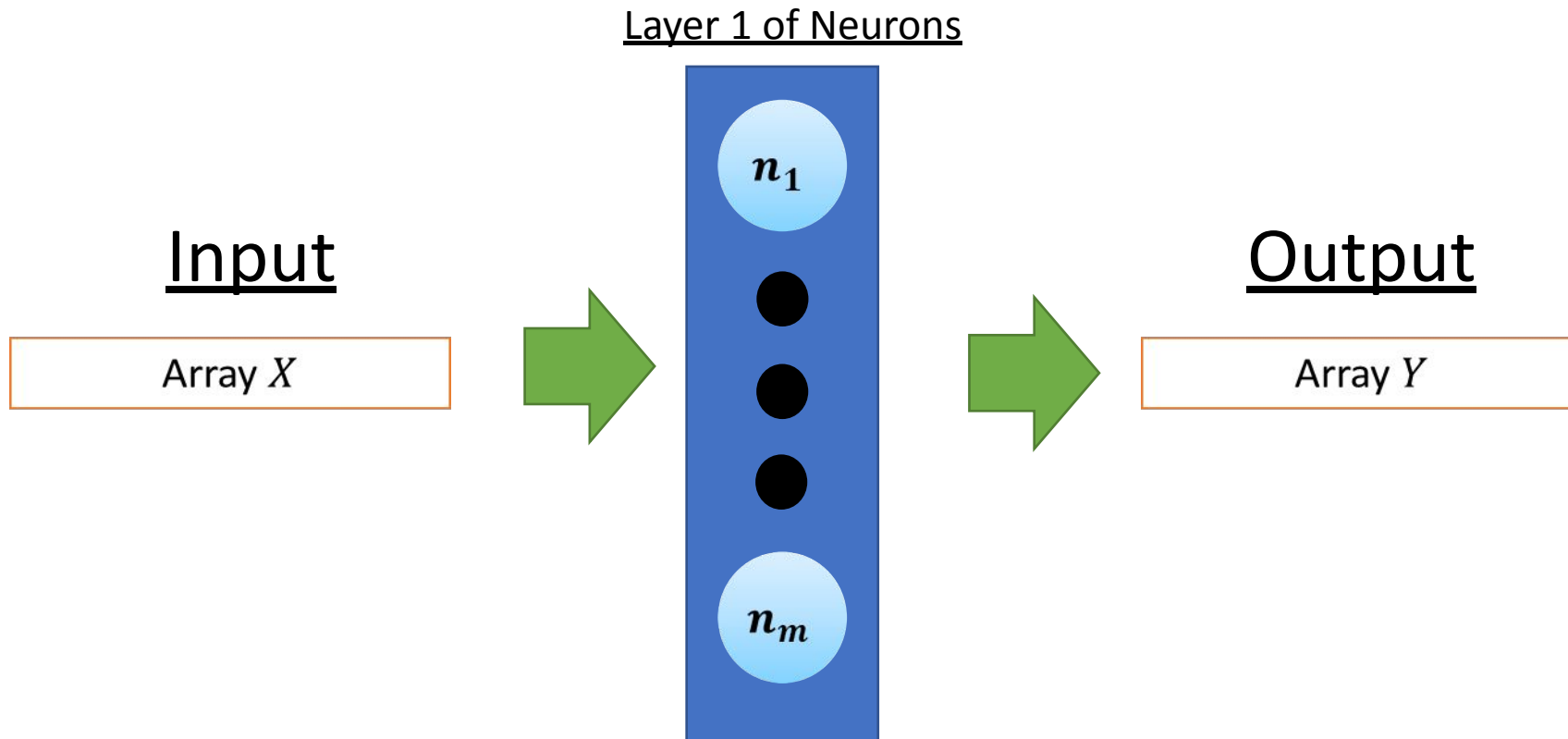


# Using Multiple Layers

- Machine learning engineers don't just want one layer.
- They want to use multiple layers of neurons as multiple layers allow for more complex network behavior (in a grossly simplified way think of more layers as “more” intelligence”).
- Therefore, our picture looks like the following (see next slide).

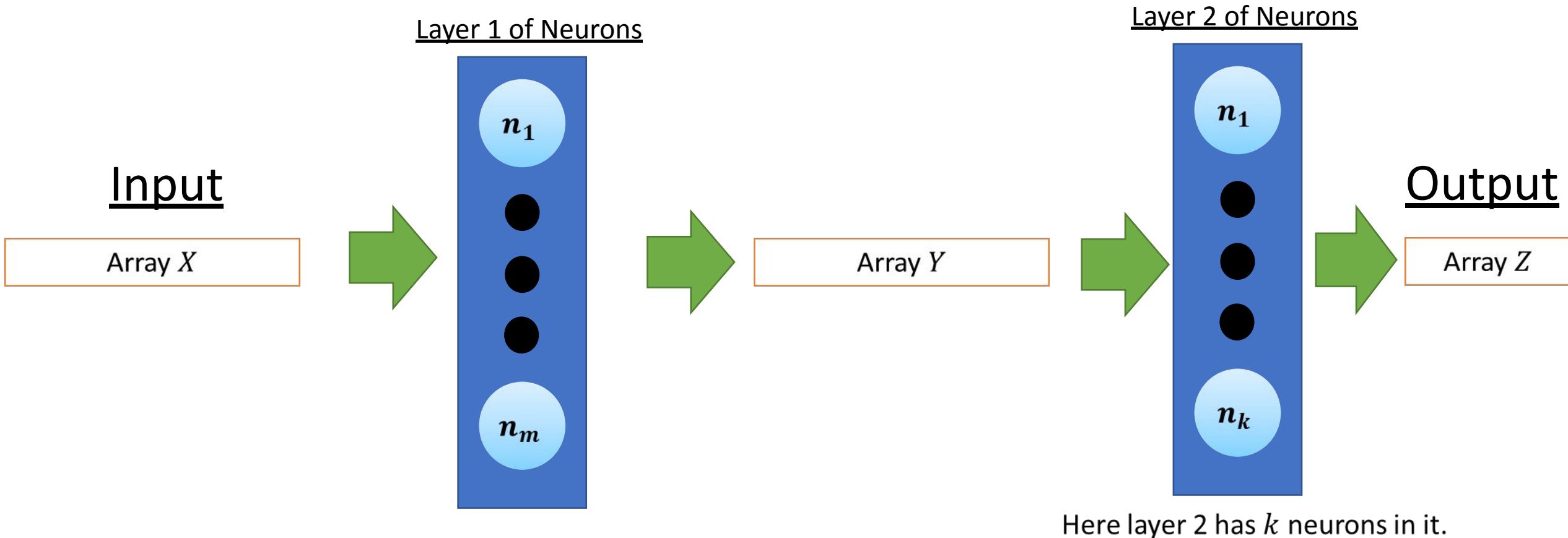
# Multiple Layers (1)

- First again consider a single layer with  $m$  neurons in it. We can draw it like this:



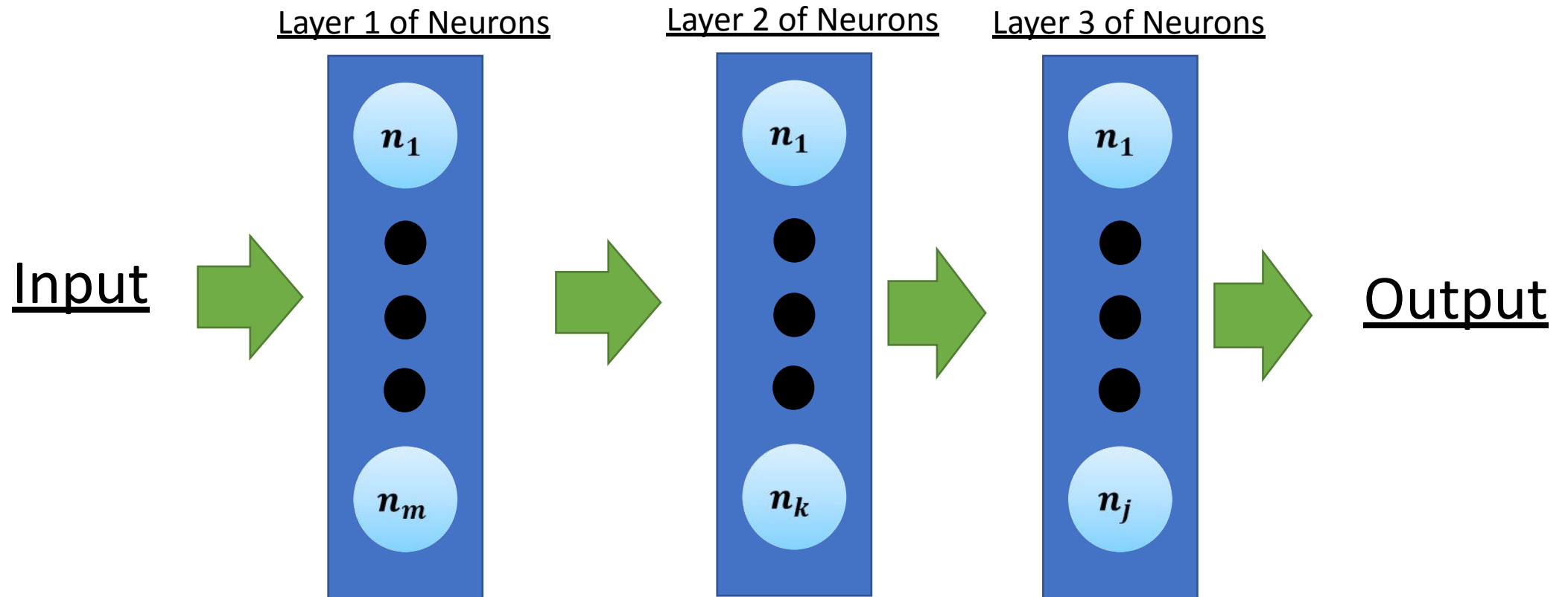
# Multiple Layers (2)

- Now we can stack the layers so that the **output** of layer 1 will be the **input** to layer 2:



# Multiple Layers (3)

- We can stack more layers together like shown below.
- Remember the basic idea is that a layer is a function that takes an array as **input** and produces an array as **output**.



# Feed Forward Neural Network in C

- When we stack layers of neurons together like this, we call it a feed forward neural network.
- See below for the feed forward neural network implementation.

```
//Feedforward network structure
typedef struct {
    int numLayers;
    //have an array with layers
    layer* lArray;
    //store the final output in this variable
    double* output;
} networkFF;
```

# Feed Forward Neural Network Initialization

- We can create the feed forward network using the following code:

```
//build the network
networkFF InitializeNetwork(int networkInputSize, int numLayers, int* numNeuronsPerLayer){
    //First create the layer array
    layer* layerArray = (layer*)malloc(numLayers * sizeof(layer));
    //Fill in each layer with neurons
    for(int i=0;i<numLayers;i++){
        //Each layer contains an array of neurons
        int currentNumNeurons = numNeuronsPerLayer[i];
        neuron* neuronArray = (neuron*)malloc(currentNumNeurons * sizeof(neuron));
        //fill in the array of neurons
        for(int j=0;j<currentNumNeurons;j++){
            //first layer handled differently
            if(i == 0){
                neuronArray[j] = InitializeNeuron(networkInputSize);
            }
            else{
                //input comes from previous layer's output size
                neuronArray[j] = InitializeNeuron(numNeuronsPerLayer[i-1]);
            }
        }
        //Neurons all build time to make the layer
        layer l = {neuronArray[0].inputSize, currentNumNeurons, neuronArray};
        layerArray[i] = l;
    }
    //lastly fill in the network
    networkFF network = {numLayers, layerArray, NULL};
    return network;
}
```

# Feed Forward Network Output

- To get the output of the network we need to do a **forward pass**:
  1. We apply an input array to each neuron in the first layer.
  2. We then put all neurons outputs from the first layer into an array, let's call it  $S_1$ .
  3. For the second layer in the array, we apply  $S_1$  to each neuron in layer 2.
  4. We then put all neurons outputs from the second layer into an array, let's call it  $S_2$ .
  5. For the third layer in the array, we apply  $S_2$  to each neuron in layer 3.
  6. We then put all neurons outputs from the third layer into an array, let's call it  $S_3$ .
  7. We continue like this until we reach the last layer which gives us our final output.



# Feed Forward Neural Network Output Code

```
//compute a forward pass on the network
void ForwardPassNetworkFF(double* x, networkFF* network){
    //basic variable setup
    int numLayers = (*network).numLayers;
    //allocate memory for the output of each layer of the network
    double** layerOutputs = (double**)malloc((numLayers + 1)* sizeof(double*));
    layerOutputs[0] = x; //first layer output is treated as input x to the network
    //initialize the arrays to hold the output of each layer
    for(int i = 1; i< numLayers+1;i++){
        layer currentLayer = (*network).lArray[i-1];
        int numNeuronsPerLayer = currentLayer.numNeurons;
        layerOutputs[i] = (double*)malloc(numNeuronsPerLayer * sizeof(double));
    }
    //Go through each layer and compute the output
    for(int i=1;i<numLayers+1;i++){
        layer currentLayer = (*network).lArray[i-1];
        int numNeuronsPerLayer = currentLayer.numNeurons;
        //go through each neuron in the layer
        //TODO: instead of processing the neurons in series try using forks and pipes
        //TODO: insteado processing the neurons samples in series, try using threads
        for(int j = 0; j< numNeuronsPerLayer;j++){
            layerOutputs[i][j] = ForwardPassNeuron(&currentLayer.nArray[j], layerOutputs[i-1]);
        }
    }
    //save the final result
    (*network).output = layerOutputs[numLayers];
    //free up the memory (except the zeroth layer which is the input x and the last layer whose output we need)
    for(int i=1;i<numLayers;i++){
        free(layerOutputs[i]);
    }
}
```



# Starter Code: Simple Network

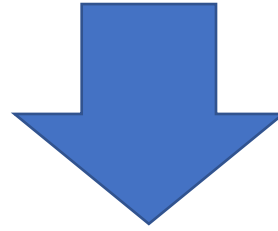
- You are given all of the code shown in the previous slides.
- We also have a simple example of a network producing one output for one input with the following properties:
  - The network has 6 layers. With 4, 3, 7, 8, 2 and 10 neurons in the layers.
  - The input array size to the network is 5 (e.g. 5 elements go in, at the very end 10 elements come out).
- To run the starter code see the following slide.

# Starter Code: Simple Network

```
void TestSimpleNetwork(){
    //size of the input
    int networkInputSize = 5;
    //layer structure of the network
    int numNeuronsPerLayer[6] = {4, 3, 7, 8, 2, 10};
    //number of layers
    int numLayers = 6;
    //Build the network
    networkFF network = InitializeNetwork(networkInputSize, numLayers, numNeuronsPerLayer);
    //sample input to the network, an array of size 5
    double x[5] = {2, 3, 4, 5, 6};
    //Do a forward pass on the network
    ForwardPassNetworkFF(x, &network);
    //print the output of the network
    for(int i=0;i<10;i++){
        printf("y[%d]=%lf\n", i, network.output[i]);
    }
    printf("Simple neural network code complete!\n");
}
```

# Starter Code: Simple Network

```
kaheel@CentralCompute:~$ gcc NeuralNetwork.c -lm -o test
kaheel@CentralCompute:~$ ./test simple
```



```
y[0]=0.454192
y[1]=0.525985
y[2]=0.356411
y[3]=0.679725
y[4]=0.520044
y[5]=0.406268
y[6]=0.552949
y[7]=0.432589
y[8]=0.521836
y[9]=0.581155
Simple neural network code complete!
```

**Note: Due to the random seed your numbers may look slightly different than mine. This is fine.**

# Starter Code: Big Network

- The simple network uses only ONE input and gives ONE output with a very small network structure.
- In reality machine learning use feed forward neural networks with many layers and hundreds of neurons per layer.
- For example: the Big Network code can be run where 100 samples are passed through a neural network with many neurons per layer.
- You can try the big network code with:

```
kaleel@CentralCompute:~$ ./test big
```

### 3. Deliverables and Submission Details

# Project: TODO

- The goal of this project is to make the big feedforward neural network code run *faster*.
- There are two ways that this can be accomplished. Furthermore, for each way there are two possible solution (using either forks or threading).
- To get credit for this assignment you have to pick either option A or B and then implement one solution correctly.
- **You do not have to do both option A and B. Only do one.**

# Option A: Speed up with Neurons

- In a single layer, each neuron computation is done INDEPENDENT of one another.
1. You can use **fork** to compute the output of half the neurons in the parent process and at the same time compute half the neurons in the child process. You will then need to use pipes to communicate the information back to the parent before going to the computation in the next layer.
  2. You can use **threads** to compute the output of half the neurons in the one thread and at the same time compute half the neurons in another thread. You will then need to synchronization so that both threads have completed the output before moving to the next layer.

# Option B: Speed up with Samples

- We need to run 100 samples through the network. That means we need to take 100 arrays as input and get 100 array outputs. However, each input is INDEPENDENT of one another so you could do each forward pass in parallel.
1. You can use **fork** to compute the output of half the input samples in the parent process and at the same time compute the other half the input samples in the child process. You will then need to use pipes to communicate the information back to the parent before giving the final output.
  2. You can use **threads** to compute the output of half the inputs in the one thread and at the same time compute half the outputs in another thread. You will then need to synchronization so that both threads have completed the outputs before the code finishes.



# Project Deliverables

## 1. **Written report** with the following:

=Screenshot with the output of the big network code run in series (e.g. no parallel speed up) and its corresponding run time in seconds or ms.

AND

=Screenshot with the big network code output run in parallel form (e.g. with the solution you choose to implement) and its corresponding run time in seconds or ms.

AND

=1-2 paragraphs explaining which solution you choose to implement, HOW you implemented the solution and what changes you made to the starter code to implement the solution.

AND

## 2. **Your code** in a single .c file that implements your solution.

## Other important notes

- You ARE allowed to create new variables and new functions to implement your parallel solution.
- Both screenshots in the report must show *THE SAME* output. E.g. if your parallel solution does not give the same output as the original code your solution is wrong.
- You ARE allowed to discuss with other students. However, you are NOT allowed to copy code.
- No using ChatGPT.
- Every student must submit a unique solution. If you are caught cheating on the extra credit assignment, that will result in a 0 for the extra credit assignment AND a 0 for the exam 3 grade.

# How to submit

- Like a real-world coding job, there will be no feedback until you have fully created your end solution. Therefore no Gradescope submission.
- Submit one document (either word or pdf) for the final report, and ONE .c file to HuskyCT.
- Good luck!