GPU-Accelerated Backpropagation with C/CUDA

A presentation by Drew Hans

> *Can a CUDA-enabled graphics processing unit (GPU) be used to reduce the time it takes to train a fully-connected three-layer artificial neural network?*

Research Question

Presentation Overview

- Fundamentals of Machine Learning & Artificial Neural Networks
- GPU-Accelerated Computing with CUDA
- My Research Project: Neuronmancer
- Project Challenges, Successes, & Results

Fundamentals of Machine Learning

Let's cover the basics

Learning

is the process of turning

experience into knowledge.

3 Major Machine Learning Approaches

Supervised Learning

The output you want from the model is known.

Two subgroups:

- Classification
- Regression

Examples:

- Image recognition
- Speech recognition
- Stock price prediction

Unsupervised Learning

The output you want from the model is not known.

One big subgroup:

- Clustering

Examples:

- Netflix recommendations
- Medical imaging
- Gene sequence analysis

Reinforcement Learning

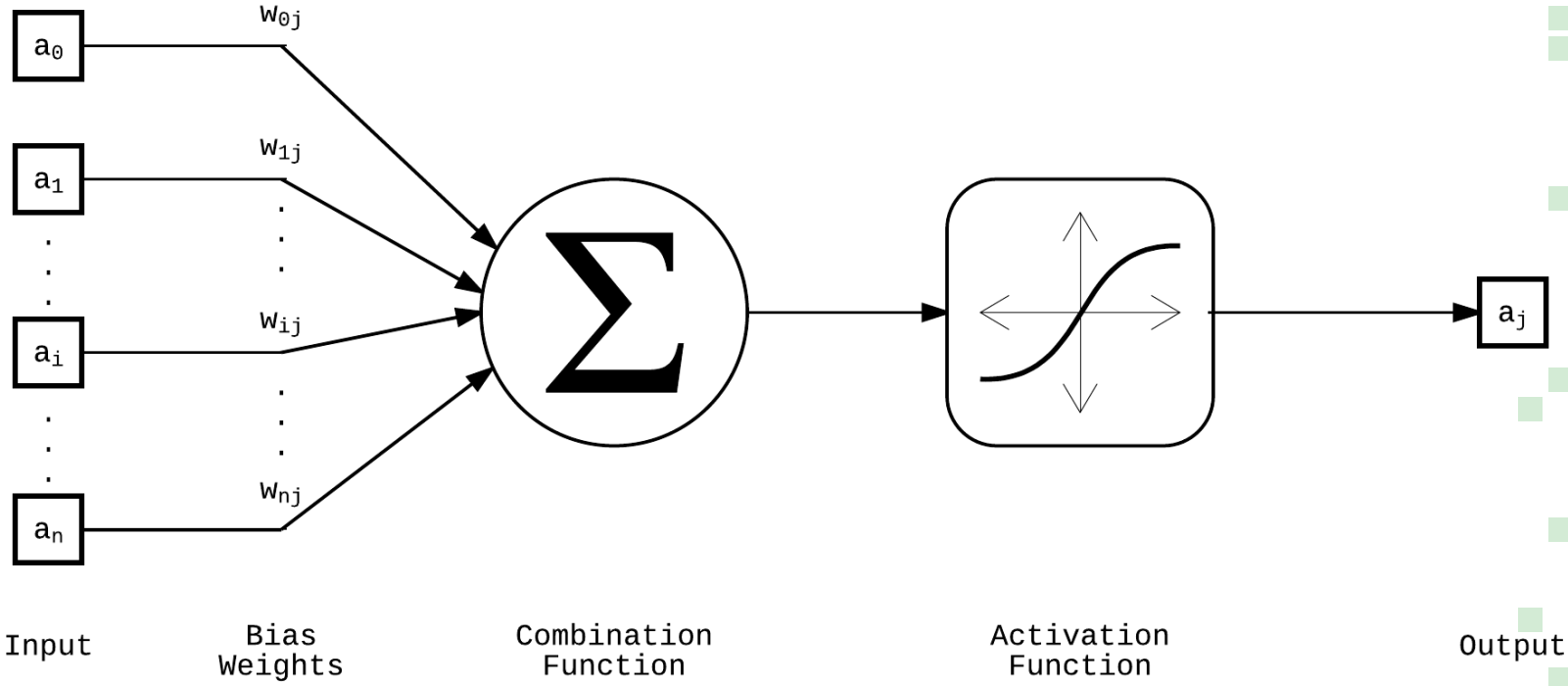
The model has agency, meaning it is able to interact with its environment, and receives "rewards" and "punishments" for its actions.

Examples:

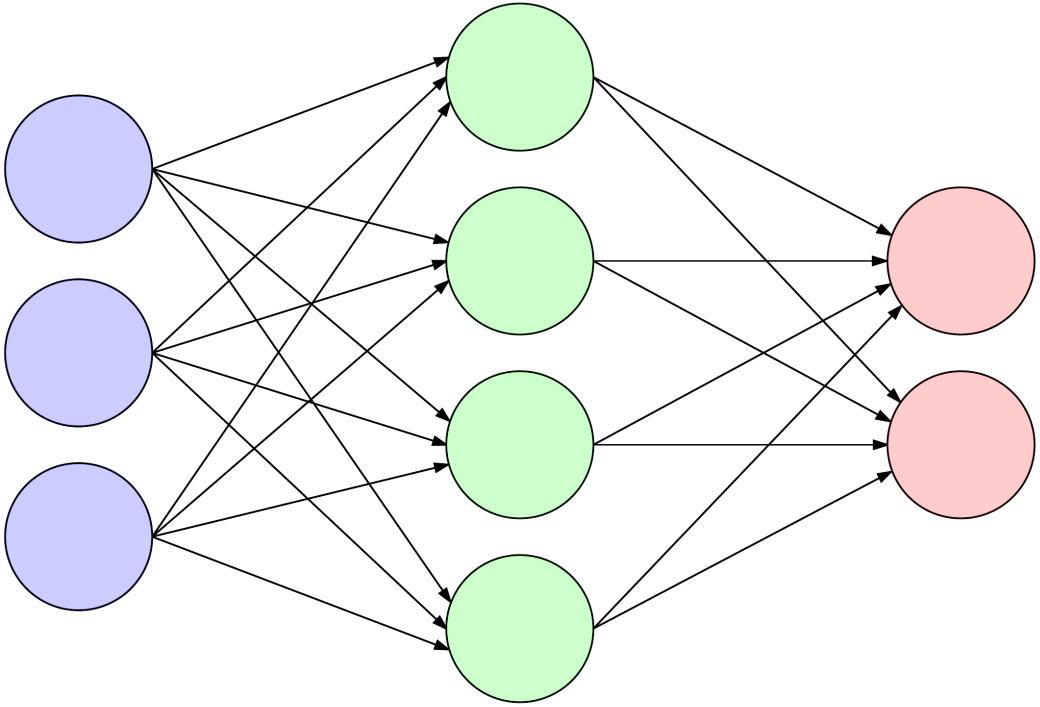
- Playing Checkers
- Playing Go
- Robot Control

Artificial Neural Networks

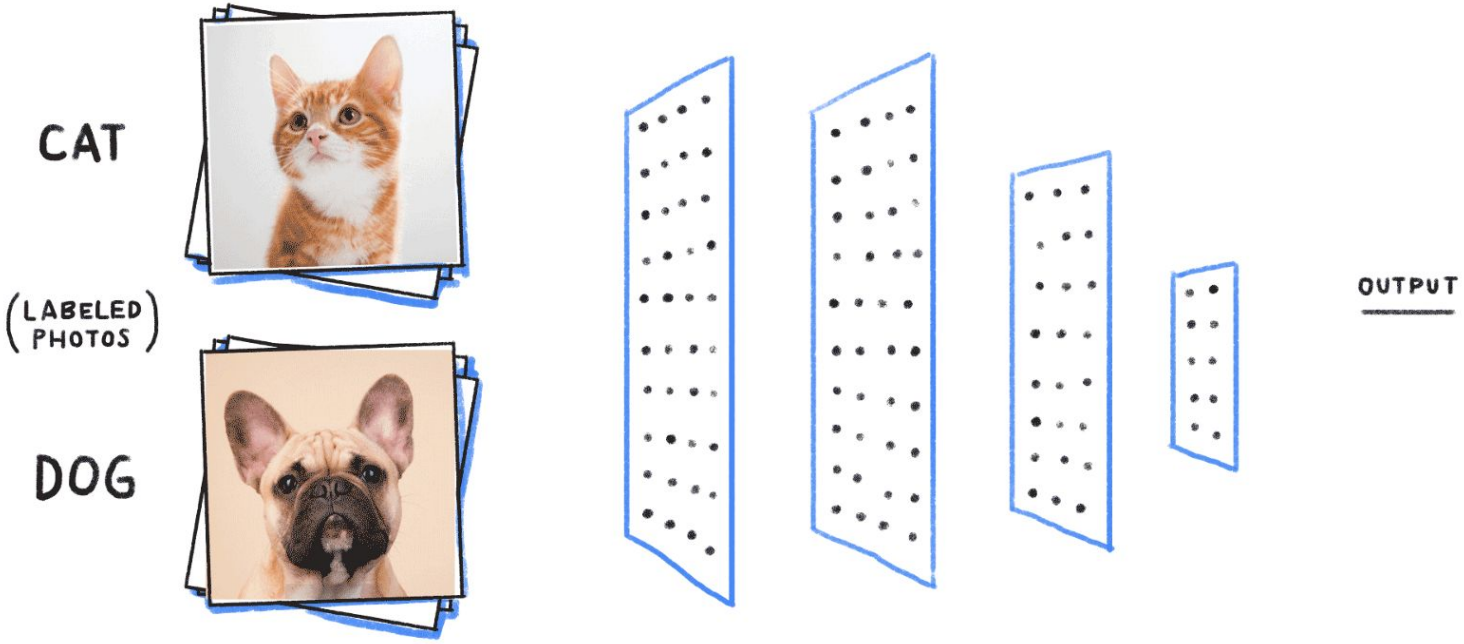
A type of machine learning model



A mathematical model for a single neuron



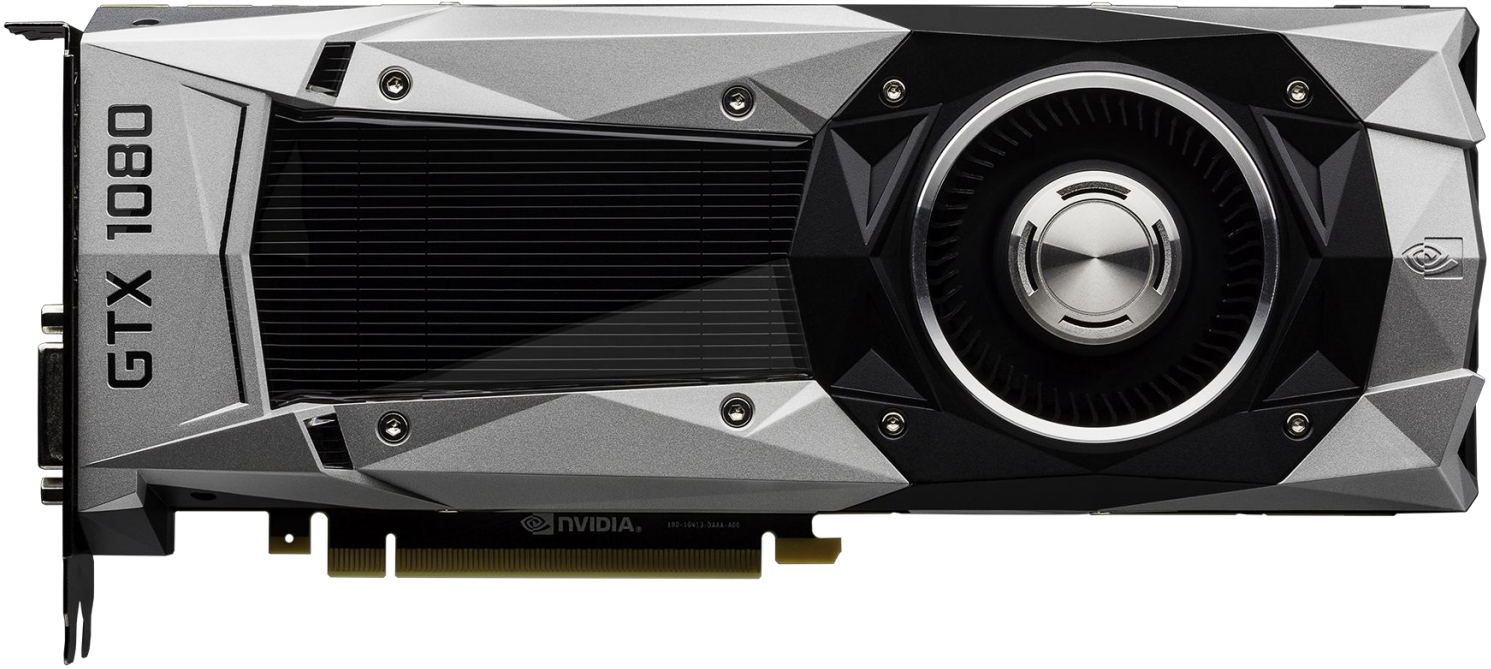
Visualization of a simple 3-layer feedforward artificial neural network



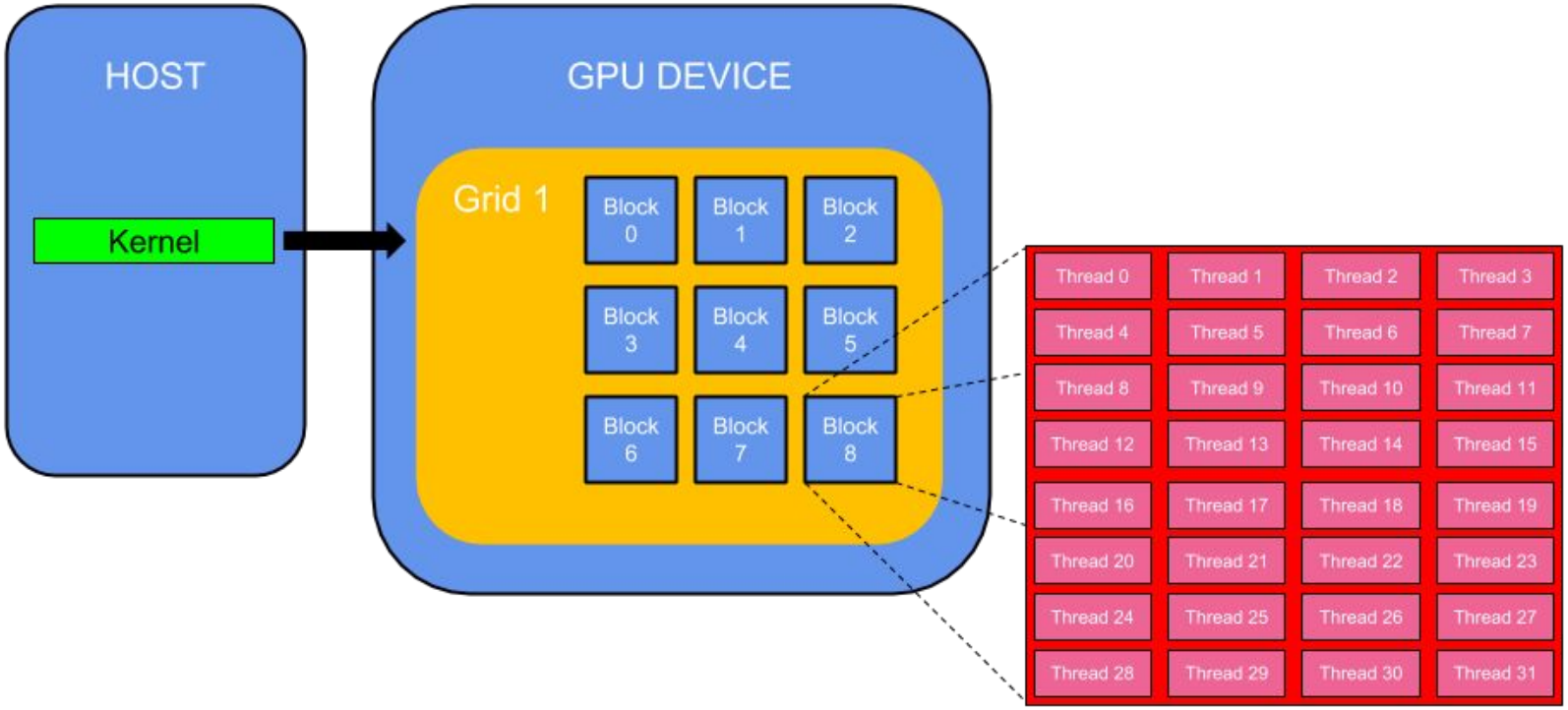
Visualization of a 4-layer neural network that identifies cats!

GPU-Accelerated Computing with CUDA

A parallel programming model



This is a CUDA-enabled GPU device - the GTX 1080



Visualization of a simple 3-layer feedforward artificial neural network

```c
int main(void) {

    const int size = 5;

    const int a[size] = { 1, 2, 3, 4, 5 };
    const int b[size] = { 10, 20, 30, 40, 50 };
    int c[size] = { 0 };

    // Add vectors sequentially
    for (int i = 0; i < size; i++) {
        c[i] = a[i] + b[i];
    }

}// end main function
```

A simple C program for adding array elements sequentially.

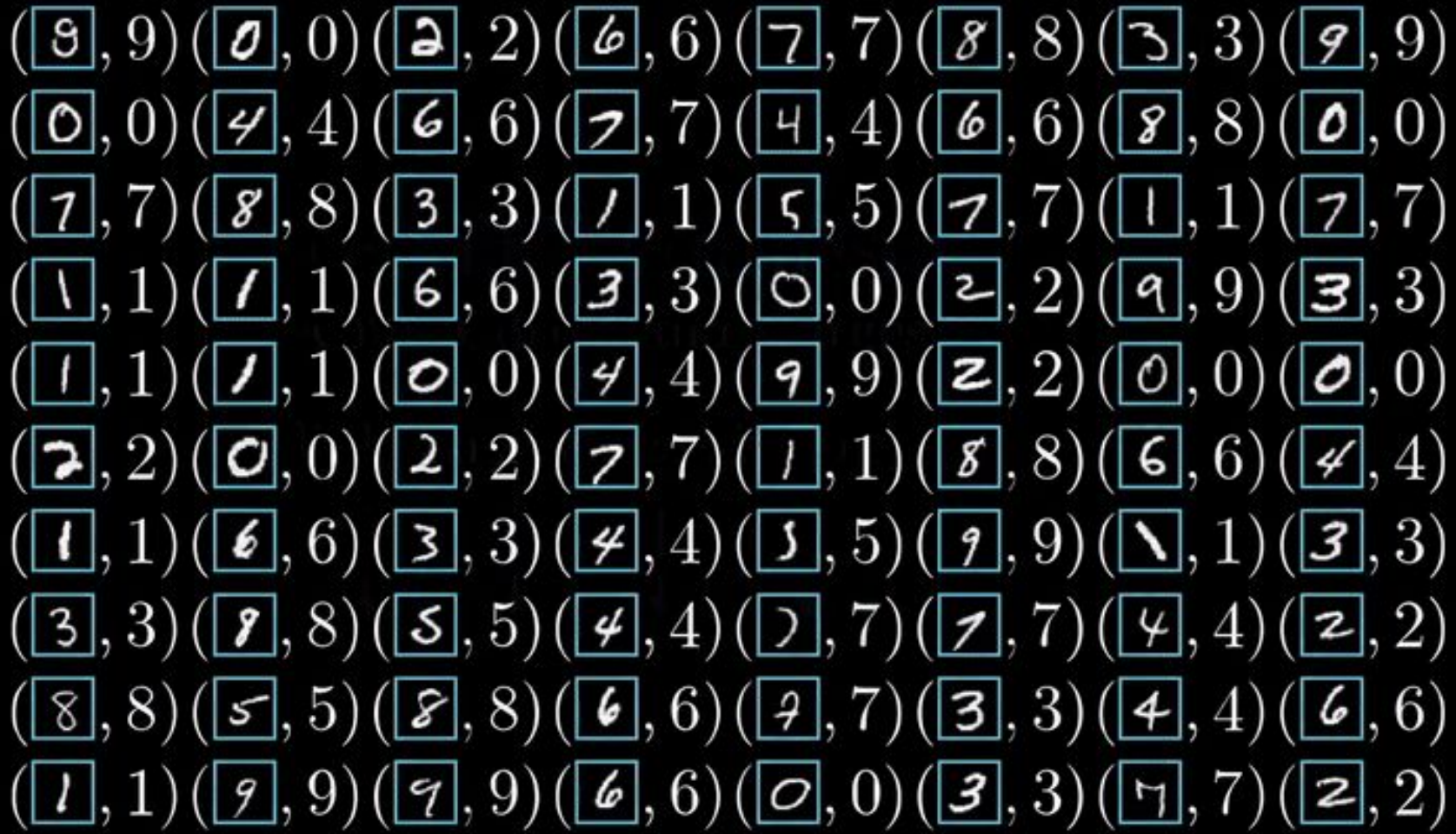
```cpp
int main(void) {
    const int size = 5;

    const int host_a[size] = { 1, 2, 3, 4, 5 };
    const int host_b[size] = { 10, 20, 30, 40, 50 };
    int host_c[size] = { 0 };

    int* dev_a = 0;
    int* dev_b = 0;
    int* dev_c = 0;

    cudaSetDevice(0); // run kernels on GPU 0

    // allocate GPU memory for device copies of a, b, c
    cudaMalloc((void**)&dev_a, size * sizeof(int));
    cudaMalloc((void**)&dev_b, size * sizeof(int));
    cudaMalloc((void**)&dev_c, size * sizeof(int));

    // copy a and b values from host memory into allocated GPU memory
    cudaMemcpy(dev_a, host_a, size * sizeof(int), cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, host_b, size * sizeof(int), cudaMemcpyHostToDevice);

    // add array values in parallel (launch 1 block with "size" threads)
    kernel_add<<< 1, size >>>(dev_c, dev_a, dev_b);

    // copy c values from GPU memory into host memory
    cudaMemcpy(host_c, dev_c, size * sizeof(int), cudaMemcpyDeviceToHost);

    // free GPU memory once we're done with it
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
}// end main function
```

```cpp
__global__
void kernel_add(int* dev_c,
                const int* dev_a,
                const int* dev_b ) {
    int i = threadIdx.x;
    dev_c[i] = dev_a[i] + dev_b[i];
}// end kernel_add cuda kernel
```

A simple C / CUDA program for adding array elements sequentially.

Neuronmancer (v3)

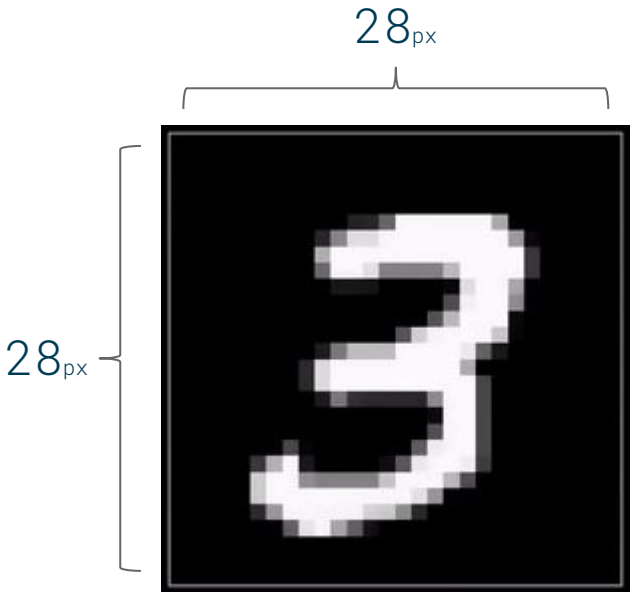
My research project

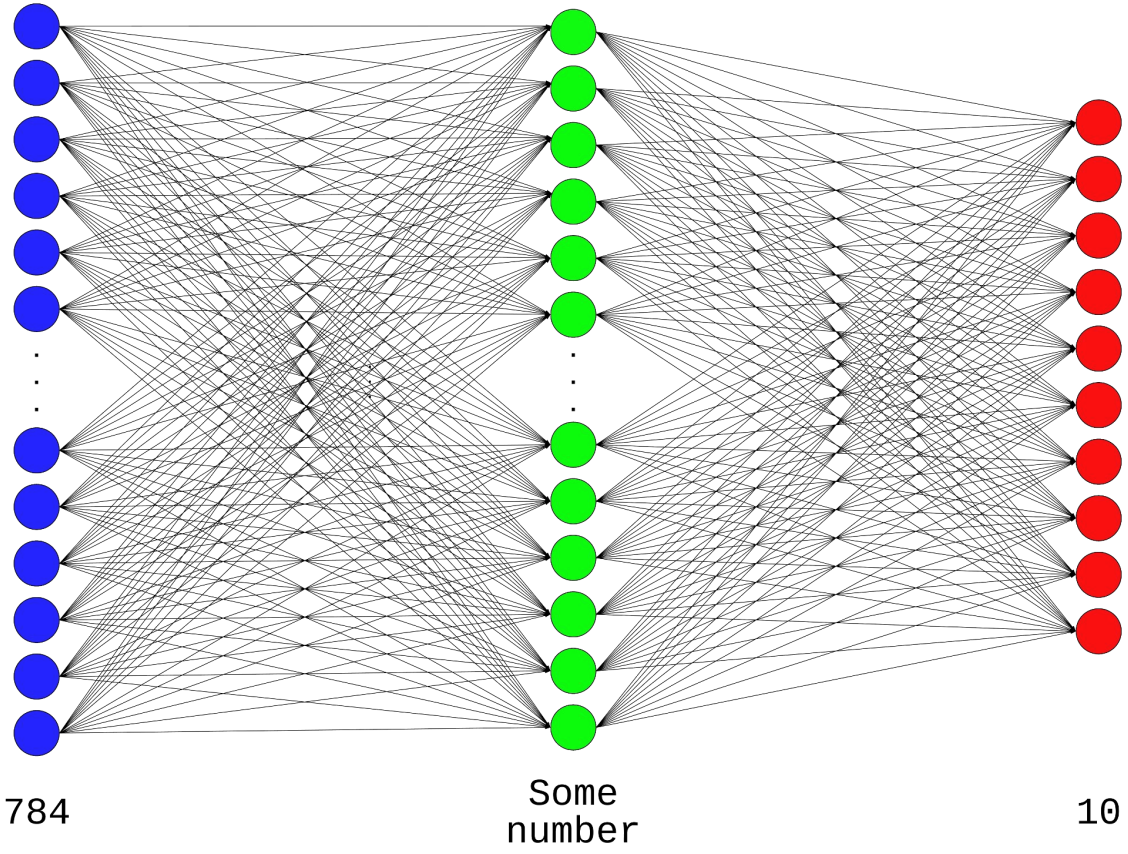


MNIST DATASET SAMPLES

28 x 28 = 784 pixels
in a single image.

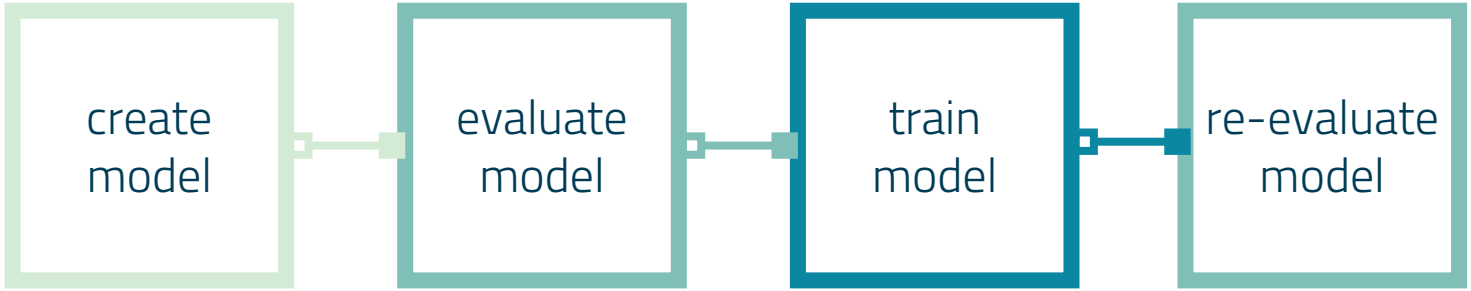
10 possible classifications
for any given image.

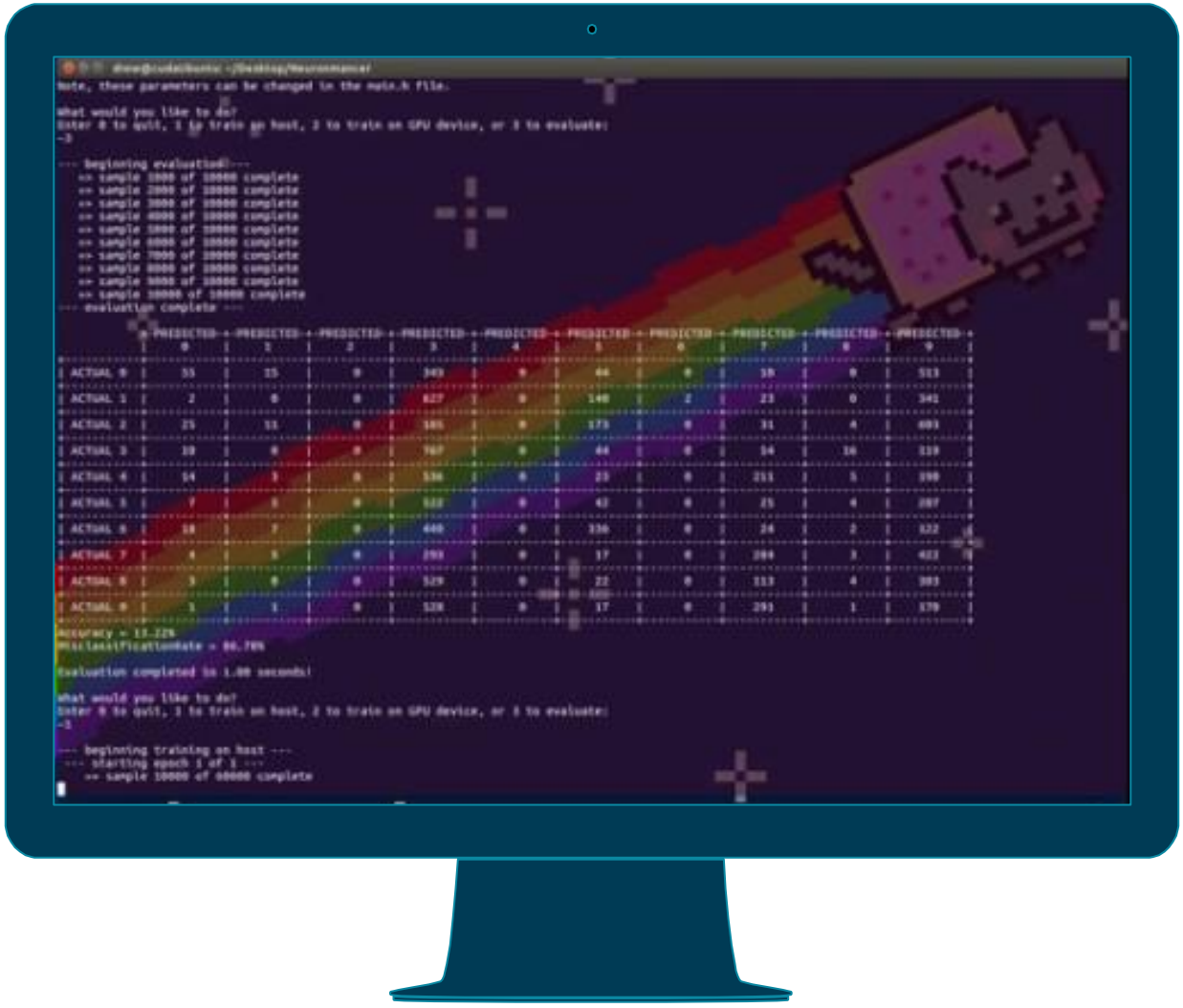




Visualization of neural network models created by my program

MY DATA GATHERING PROCESS

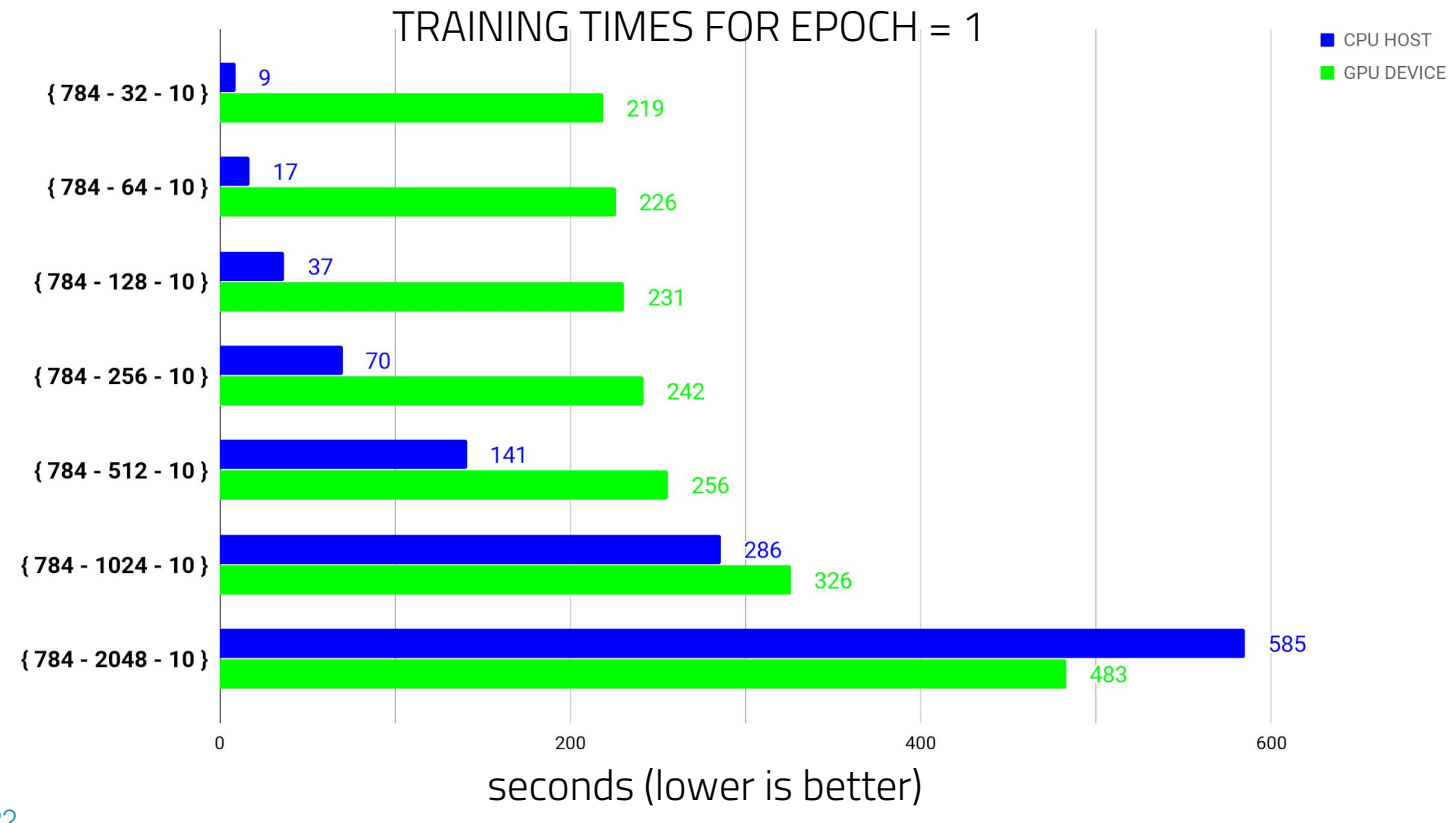



Note, these parameters can be changed in the main.h file.
What would you like to do?
Enter 0 to quit, 1 to train on host, 2 to train on GPU device, or 3 to evaluate:
--- beginning evaluation ---
=> sample 1000 of 10000 complete
=> sample 2000 of 10000 complete
=> sample 3000 of 10000 complete
=> sample 4000 of 10000 complete
=> sample 5000 of 10000 complete
=> sample 6000 of 10000 complete
=> sample 7000 of 10000 complete
=> sample 8000 of 10000 complete
=> sample 9000 of 10000 complete
=> sample 10000 of 10000 complete
--- evaluation complete ---
	PREDICTED	PREDICTED	PREDICTED	PREDICTED	PREDICTED	PREDICTED	PREDICTED	PREDICTED	PREDICTED	PREDICTED
	0	1	2	3	4	5	6	7	8	9
ACTUAL 0	35	15	0	343	0	44	0	10	0	513
ACTUAL 1	2	0	0	827	0	140	2	23	0	341
ACTUAL 2	25	11	0	185	0	173	0	31	4	603
ACTUAL 3	19	0	0	767	0	44	0	14	16	119
ACTUAL 4	14	1	0	536	0	23	0	211	1	196
ACTUAL 5	7	1	0	512	0	42	0	25	4	287
ACTUAL 6	18	7	0	448	0	336	0	24	2	122
ACTUAL 7	4	1	0	293	0	17	0	284	3	423
ACTUAL 8	3	0	0	529	0	22	0	113	4	303
ACTUAL 9	1	1	0	538	0	17	0	291	1	170
Accuracy = 13.22%
Misclassification rate = 86.78%
Evaluation completed in 1.00 seconds!
What would you like to do?
Enter 0 to quit, 1 to train on host, 2 to train on GPU device, or 3 to evaluate:
--- beginning training on host ---
--- starting epoch 1 of 1 ---
=> sample 10000 of 60000 complete

RESULTS (1 of 2)

Intel(R) Core™ i7-4770K CPU (3.50GHz)

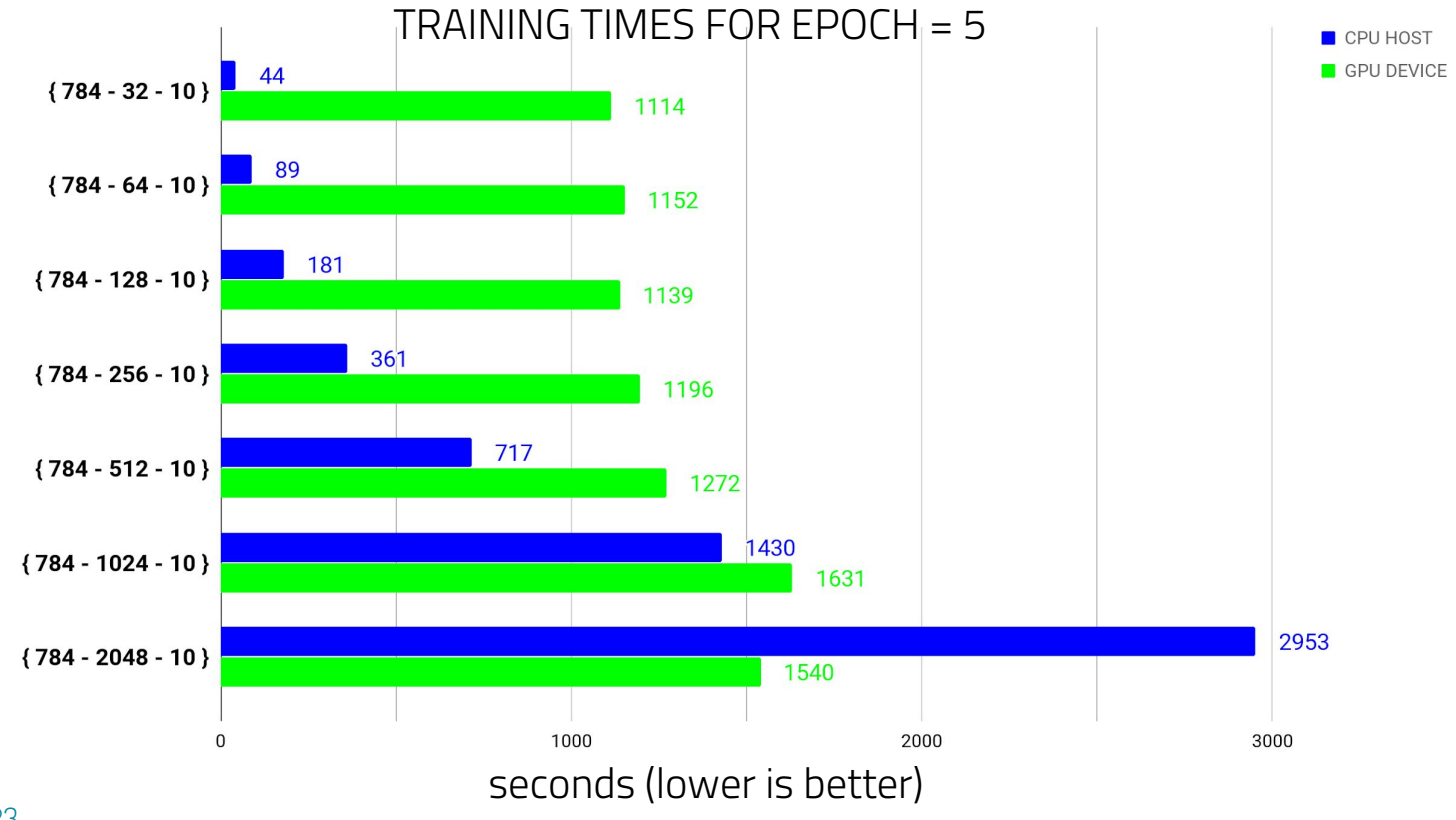
NVIDIA GeForce GTX 670 (1344 CUDA Cores)



RESULTS (2 of 2)

Intel(R) Core™ i7-4770K CPU (3.50GHz)

NVIDIA GeForce GTX 670 (1344 CUDA Cores)



CHALLENGES

- ➔ Working with Visual Studio 2015
- ➔ Learning how to write Makefiles for GNU Make
- ➔ Learning how CUDA-enabled GPUs work
- ➔ Learning how to think like a C programmer
- ➔ ... oh, and those two awesome times I spent an entire weekend rewriting Neuronmancer

Acknowledgments

Special thanks to all my friends who encouraged me when I felt like giving up.

Also, to my professors for giving me the freedom to explore a topic I found interesting.

THANKS!

Any questions?

You can find me at:

 github.com/DrewHans555

 drewhans555@gmail.com

/ / end of presentation