Mitchell B. Fulton

Professor Arden Ruttan

Individual Honors Work - A.I. Project

12 December 2016

## Part 1: A Different Kind of Problem Solving

The concept of Artificial intelligence is a unique one, which requires a fundamentally different approach to problem solving than traditional methods. A.I. is essentially a synthetic construct that is capable of resolving issues that are typically only solvable by human intelligence or human perception. These constructs can be put to a variety of uses, however they are generally used to automate processes that humans can easily perform but traditional computer systems cannot understand. An example of this being image recognition, where a person might easily comprehend that two different images are of the same object while a program would struggle to make the same correlation. Another main area that A.I. excels in are tasks that require human like thought but involve massive amounts of data that would be prohibitively large for a human to examine. When building an A.I. it is important to keep in mind that the programmer is not necessarily constructing a piece of software to solve a problem, but is rather building a system that has some capacity to learn how to solve a problem.

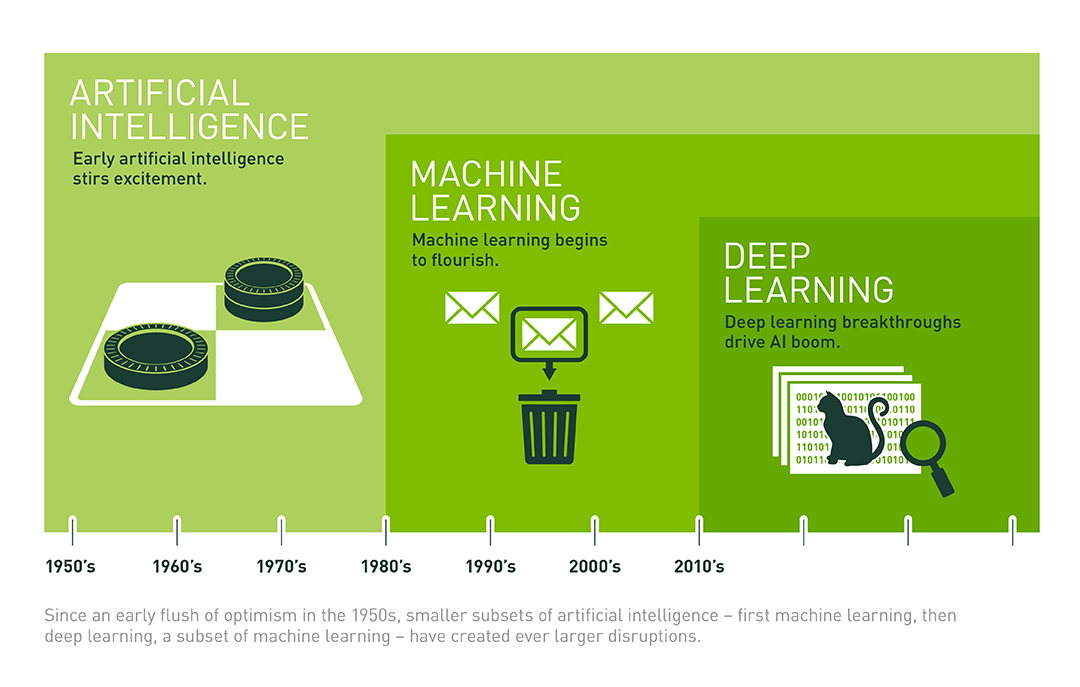


Image courtesy of NVIDIA Blog

One field of A.I. is machine learning which specializes in giving “computers the ability to learn without being explicitly programmed” (Arthur Samuel). Machine learning can be imagined as a more modern “Approach to Achieve Artificial Intelligence”(nvda). This approach involves using self modifying algorithms to predict future events. The basic steps of machine learning are: a generic untrained algorithm or set of algorithms runs against input data, the algorithms output a prediction of the results, the actual results and predicted results are compared to calculate error, the algorithms are then modified based on the calculated error, and then the process begins anew. Each iteration of this loop tunes the algorithms to become more and more accurate at predicting the outcome of events. This means that machine learning has enormous potential to operate under ideal circumstances but has several inherent requirements and drawbacks. To facilitate operation of machine learning enormous amounts of data are required to train the algorithms. Also since the algorithms are based upon historical data they have little to no capacity to predict outliers or abrupt changes.

Deep learning is a subfield of machine learning that further specializes in predicting outcomes by mimicking the mind's capacity to understand abstract thoughts and ideas. The most notable technique for deep learning is with the use of artificial neural networks. Organic neural networks found in the brain operate using a collection of neurons and synapses. Think of the neurons as light bulbs and the synapses as wires connecting different bulbs together. Every thought that occurs within the mind is a chain of neurons lighting up, the more abstract or complicated the thought the more neurons it involves. The synapse between each and every neuron has a certain set resistance against passing electrical signals between the neurons it connects. Each time this synapse’s resistance is overcome and electrical signals are passed the resistance diminishes slightly. It is on this principle that memories operate, the more times the mind thinks about a certain concept the easier it becomes to recreate that thought. This is why it is easy for the average person to remember their own name but difficult to remember something they have only heard once. Artificial neural networks perform by emulating their organic counterparts. These networks are comprised of nodes which represent neurons, these nodes are organized into various layers, and weights (synapses) are established to connect the nodes. Input data is fed to each node (after being manipulated by respective weight) on the 1st layer of the network, the value of a node is equivalent to the summation of each input multiplied by its respective weight. For example, if the input was 1 and the weight to a node was .36 then the value of the node would become .36 (assuming there is only one input). This process repeats itself with the next layer of the network where the node value of the previous layer becomes the input for the subsequent layer. After all the layers have been calculated the result is outputted. To insure that the system output is usable an activation function is typically applied (at each layer), which compresses the result down on a scale from 0 to 1.

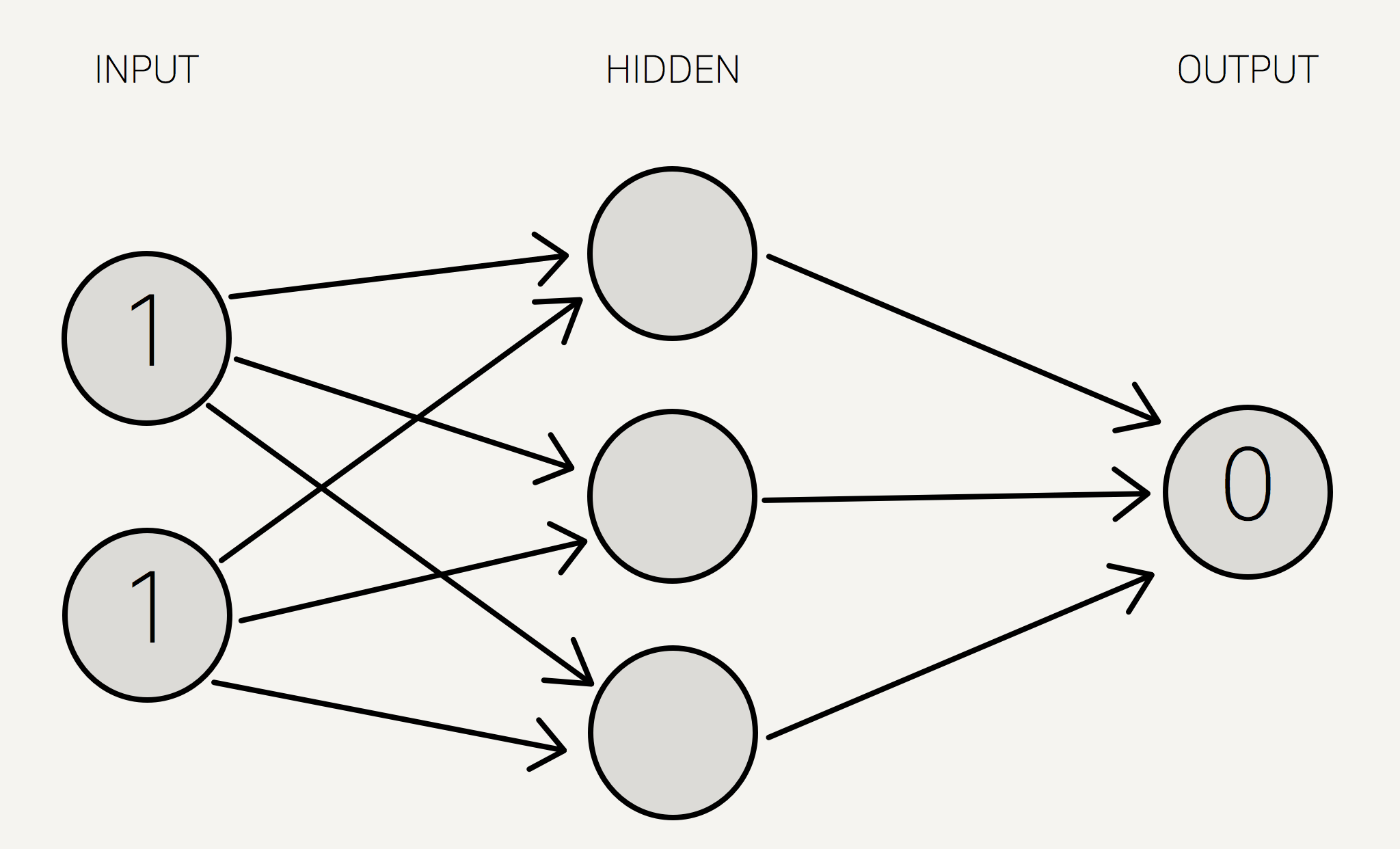


Image courtesy of StevenMiller Github tutorial.

Multiple ways exist to train neural nets, however for this program we will be using backpropagation exclusively. Backpropagation involves calculating the total error of the output of a network, then correcting the weights between the various parts of the network to account for said error. The further back the weight is from the output is, the more complicated the calculation since the effects of the error correction must be determined for each specific node and all nodes that depend on said node for input. We will discuss backpropagation and the functioning of neural nets later in section 3 in greater detail.

## Part 2: Development & Purpose of Program

The original vision for my implementation of artificial intelligence into a program was to create a sort of systems manager program. One that could maintain the integrity of a system while simultaneously pursuing some sort of goal. The system would be forced to monitor a set series of variables that represented varying aspects of system health, and would make decisions that would have an effect on those variables. The goal was to get the manager A.I. to learn that making decisions that further progress towards the goal are not always the best solution. For example, say the goal was finding a pair of lost glasses. The simplest solution is to look for the glasses until they are found. However if all a person does is look for the glasses they will eventually experience issues such as not going to work, or starving to death because they refuse to eat until they find the missing glasses.

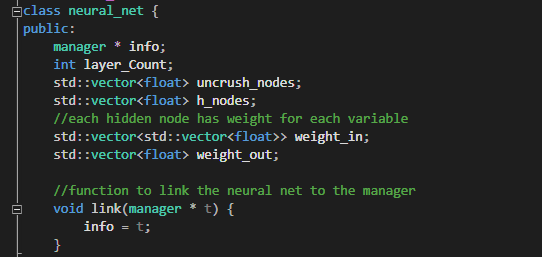
As time progressed over the semester the vision for the project changed and the goal of the program shifted slightly. Instead of preserving the health of a system and pursuing a goal the role of the manager became deciding whether or not a decision would positively or negatively affect a system. The manager would like before be given a series of variables to monitor, and then a series of randomly generated decisions would be run through the manager and executed if they were approved. Let us consider a system that needs to monitor two variables. As the system initializes the status of the system variables would be set to some constant, 100 for instance. The first decision would then be generated, each decision is comprised of two random floats in between a specified range. With each float corresponding to an effect on the system variables if the decision is executed. An example decision might contain a +24 for variable one, and a -12 for variable two. If this decision was executed the system would go from a status of 100, 100 to 124, 88. After each executed decision the variables in the system would be checked to make sure that they were still above zero, if they weren’t then the manager would be deemed a failure and the program would end. The goal of this version of the project was to create a manager A.I. capable of not only realizing what constitutes a good decision for the system but also with the capacity to make the comparatively worse choice if the status of the system demanded it. An example of this being the manager refusing to execute a decision that has a large net positive (86, -7) but would crash the system (i.e. if the 2nd variable of the system was only at 5).

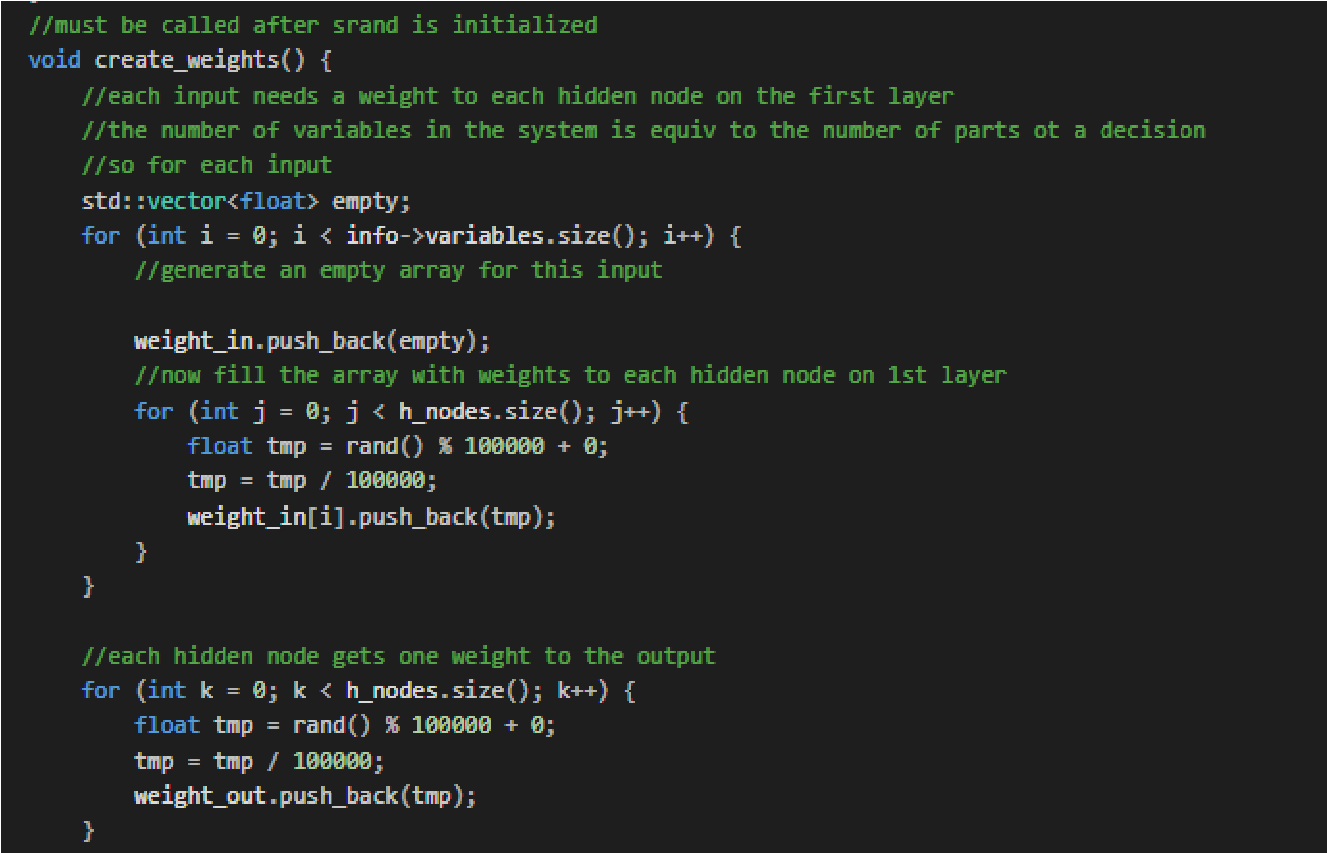
During the semester considerable progress was made towards this type of system however there were two major issues that stopped this version of the project from being the final implementation. The first issue was that there were no consequences for the system refusing to make a decision, subsequently the manager tended to reject most decisions and wait for better random number generation inside from the decision generator. This behaviour defeats the purpose of teaching a manager to make difficult choices to manage a system, and makes such a manager unfit for real world implementation. A sort of penalty system had to be devised that would force negative repercussions on the system should the manager refuse to execute a decision. The solution to this problem was actually quite simple, if the manager decides to execute a decision the system variables are summed with the corresponding decision modifiers, if the manager rejects a decision all variables in the system are penalized by a set amount. By creating a situation where there is always a negative to refusing to take action (even if the incentive is small) the manager A.I. was forced to make decisions more carefully and prevented it from simply idling. The second major issue proved the death for this iteration of the manager A.I. The second iteration of the program had intelligence implemented through an artificial neural network. Which poses a significant problem since artificial neural networks require training to become effective and intelligent. For the 2nd iteration of the program however there was no easy way to train the network. Training through back propagation requires that there be a quantifiable right or wrong answer to each decision that was executed, and for a program such as this where each decision is randomly generated and whether or not the decision is “correct” depends on the status of the system it is very difficult to determine if there is a right or wrong choice. Let alone determining the extent of the wrongness so that the margins of error can be used for calculations. Due to the difficulties in determining a “right” answer and the impossibilities of determining how correct an answer was it quickly became apparent that this iteration of the project was doomed since their was no way to train the manager.

After the collapse of the 2nd iteration of the project, the 3rd and final iteration was born by slightly modifying the purpose of the systems and networks in the 2nd iteration. The goal of iteration 3 was to create an A.I. capable of mimicking a mathematical function on a set of data. The system uses the same random decision generator as before and then plugs the data it generated into both the neural network for the manager and into an algorithm. Each one generates a respective result, and then the results are compared for any differences. The algorithm based result is the correct result, since the goal is to train the neural network to mimic the functionality of the algorithm. Error is then determined through the comparison and the neural network is back propagated to in an attempt to train it. Then the a new decision is generated and the process repeats itself until the network can replicate the results of the algorithm within a certain margin of error.

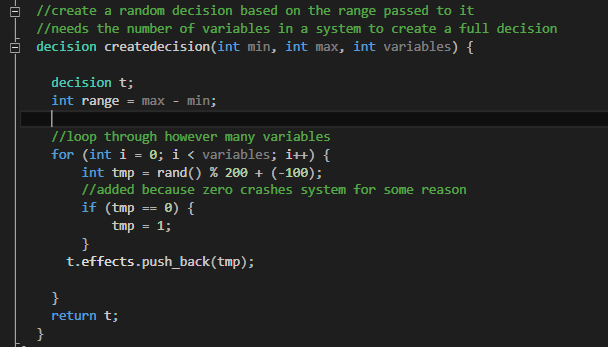
Long story short a lot of code was written for this project and never used, and the final implementation of the project is a single layer neural network that is designed to guess the results of an algorithm.

## Part 3: Implementation & Code Explanation

The class that holds all the information for the artificial neural network can be seen below. The vector of floats h\_nodes serves to represent a single layer of hidden nodes in the neural network. If this project were to be expanded into a multi layer neural network this would have to change to a vector of vectors or something similar to represent multiple hidden node layers. Uncrush\_nodes serves as a placeholder to save the value of the nodes before they are “crushed” down to between 0-1 for calculations. The vector of floats weight\_in and out represent the weights from input to nodes and nodes to output respectively. Note: weight\_in is a vector of vectors to accommodate multivariable inputs into the system.

Before running the program some initialization must be handled, the vectors must be populated the correct number of elements according to the current setup of the system, and those elements must be filled with random numbers to start up the neural network. Note: the random number range is arbitrary and can be tweaked.

Now that the network is initialized, we must generate data to be run through said network.

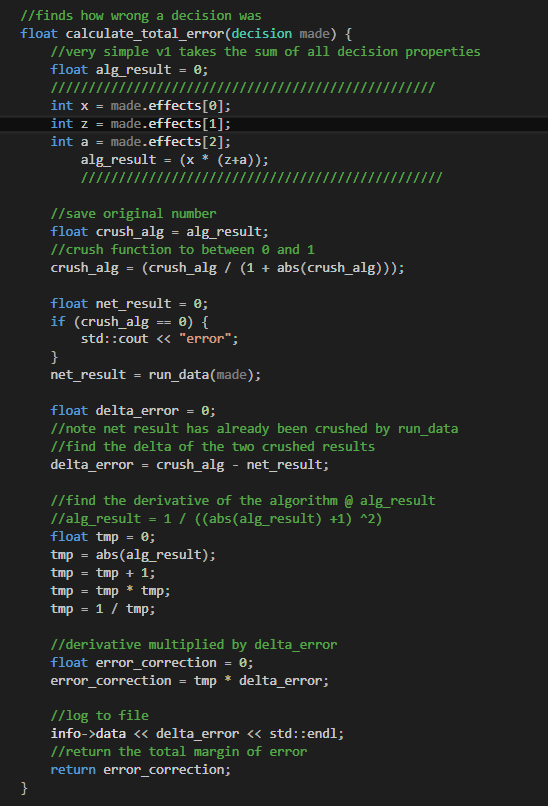


Below is where the data gets fed into the neural network and a result is produced. Here are the basic steps:

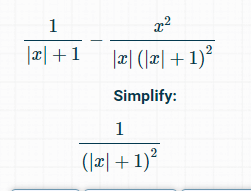
1. Each input is multiplied by the weight to a node. The sum of all results of this operation for each input are then saved as the value of the hidden node.
2. The hidden node values are then saved into uncrush\_nodes and then they are crushed using a fast sigmoid function 1/ (1+(abs(x)). Image courtesy of derivative calculator
3. Each crushed node value is multiplied by the weight to the output layer, and then the sum of all these operations is saved to the output.
4. The output is then crushed using the same function.

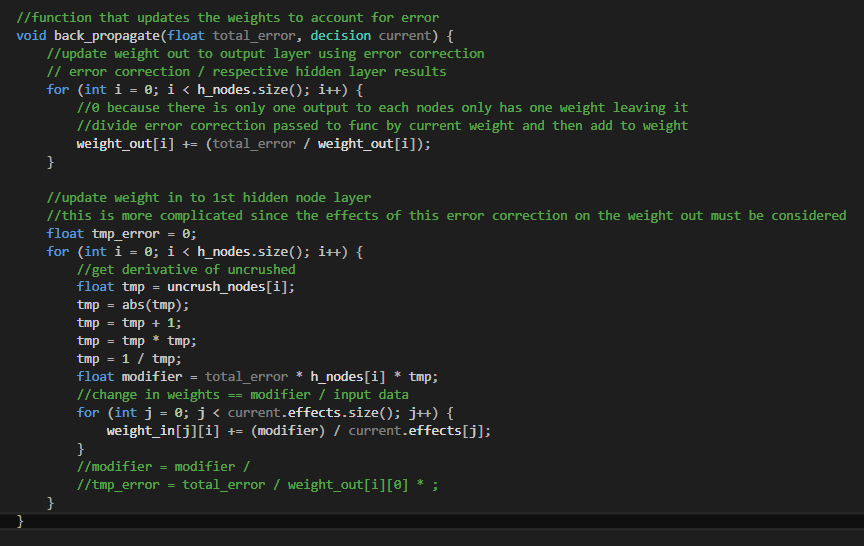
## 

After determining the network's result the error must be calculated by comparing the network result to the algorithmic result. This produces the delta error (alg - net), which is then multiplied by the derivative of the crush function to get an error correction value to pass to the backpropagation. This function is also responsible for gathering data on the progress of the network’s learning curve by exporting the delta error for each iteration to a data file.

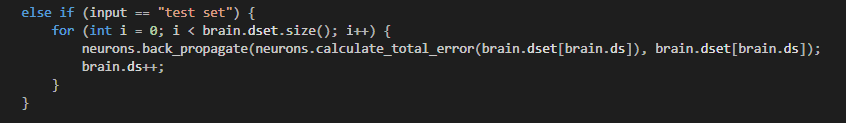


Once the error correction has been calculated it is passed to the back\_propagate function which adjusts the weights in the neural network to train it. The basic steps to updating weights are:

1. Divide the error correction number by each weight to the output layer, and then add the result to the original weight.
2. Find the derivative of the uncrushed h\_node value. In this case the uncrushed value is run through the derivative of the fast sigmoid function.  
   Calculations and images courtesy of derivative calculator  
   
3. Multiply the crushed h\_node by the result of step 3
4. Multiply the result of step 4 by error correction
5. Sum the original weight with itself and the product of step 4 times the input. (note that in this case input are the variables generated by decision, in a multi level network this would be the next node layer)  
   Weight += weight \* ((step4) /input)



Finally below all of the function calls necessary to run the above steps can be seen.

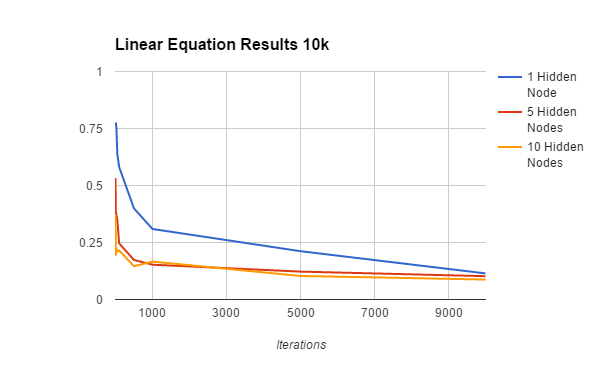


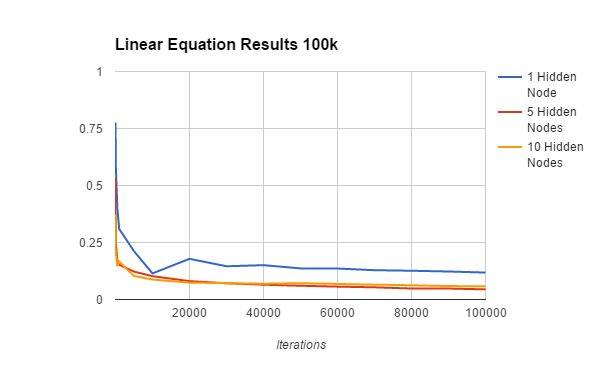
## Part 4: Results and Data Analysis

To begin testing the algorithm will be set to a simple linear equation as a proof of concept. To determine the progress being made towards having the neural network emulate the algorithm we will be charting the delta error of the algorithm’s crushed number and the net’s crushed number (similar to percent error). One hundred thousand random numbers were passed through the system with 3 different numbers of hidden nodes (artificial neurons) in the first layer. The results were rather interesting; it appears that more hidden nodes in the system causes the final margin of error to decrease. Even a single node was capable of mimicking the linear algorithm, albeit with a lesser degree of accuracy than multi node setups. It is important to note that the weights for the initialization of the network and the input numbers were all the result of randomization and were unique to each trial run(a new set of input data was used for each different configuration). Regardless this data proves that the program is capable of reducing its margin of error based upon backpropagation. All 3 configurations also demonstrated a principle of diminishing returns where the network’s jumps in accuracy across iterations steadily decreased towards zero. In essence the program was still becoming smarter, but the rate of its learning declined with each pass.

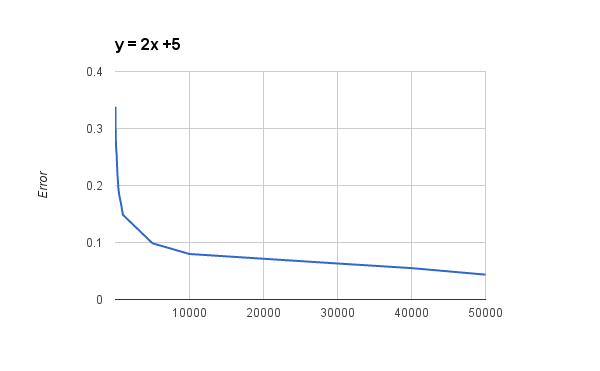
Y = X

Where x is input and y is output.

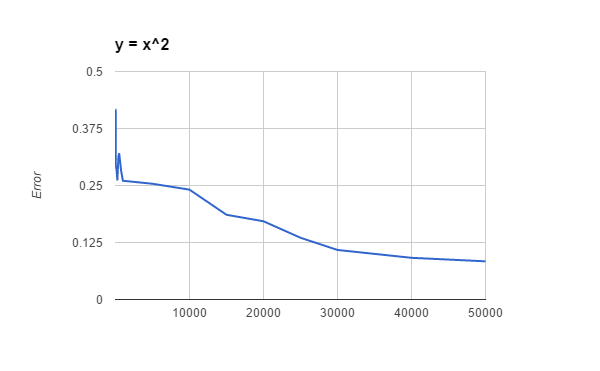




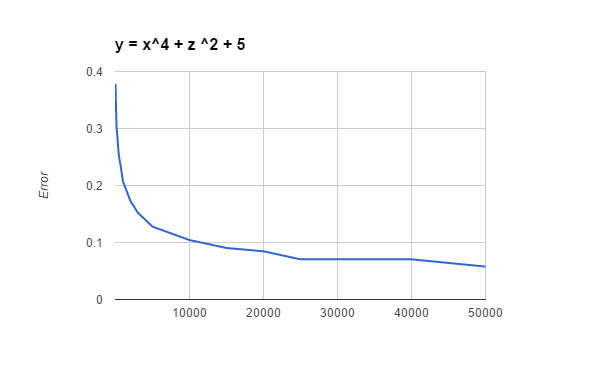
Now that it has been established that the network is capable of learning, we will begin testing more complex algorithms. For each of these tests once again randomized data will be used in a network with 10 hidden nodes.



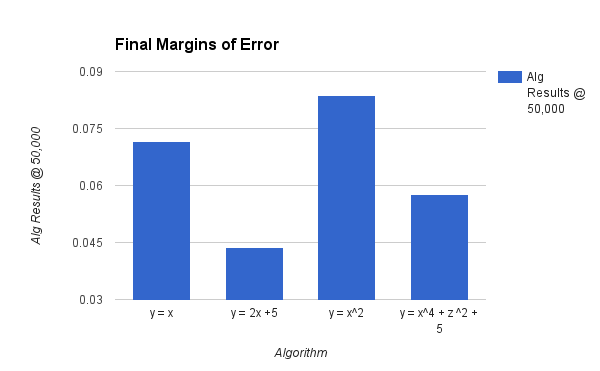
Next we will test an exponential equation



Finally we come to a complex multi variable equation



In all cases of testing it is apparent that the network is capable of learning how to emulate an algorithm but the change in delta error across iterations continuously decrease to a certain unspecified margin of error. It appears that the “margin” varies by algorithm and isn’t necessarily tied to complexity. This is demonstrated by our first tests of a linear algorithm, where the delta error continued to decline from the 10,000 mark to the 100,000 mark but the change in delta error between each iteration became so small as to be negligible.



It appears that adding more variables to the system had little to no affect on the ultimate margin of error.

A warning to those wishing to recreate this test data, the neural network currently throws a fit whenever it is passed a zero as input, or if the algorithm generates a zero, or if the crushed value is a zero. This is due to the fact that multiplication is used inside the neural networks, meaning if a zero is passed at one point it has a tendency to cause the whole system to turn to zero.

## Part 5: Problems

The main issue with creating and managing a project such as this is simply keeping track of everything. It can be hard to wrap one's mind around all the different vectors, nodes, weights, etc, especially if the principles of neural networking and back propagation aren’t fully understood. Another major issue is also debugging, if there is an error in the code relating to the mathematics behind all of this it can be nearly impossible to discover. This due to the fact that programs like this involve hundreds of numbers being affected by dozens of calculations over thousands of iterations. As of the time of this writing there is only one major problem with the program that I am aware of, and that is its intolerance towards zeros. This could likely be fixed by manipulating the algorithms given more time, however that is something that must be addressed in the future.

## Part 6: Future

The program works as demonstrated by the data above, however it is only set up in a single layer neural network. Given more time this could easily be converted to a multilayer neural network however. If I was to do this project over again there are probably two main things I would do differently: 1) I would write it in c# as opposed to c++ just for simplicity’s sake. 2) I would likely use a deep learning api such as nvidia’s sdk or Microsoft’s azure.

## Works Cited:

Copeland, Michael. "The Difference Between AI, Machine Learning, and Deep Learning? | NVIDIA Blog." *The Official NVIDIA Blog*. NVIDIA, 01 Dec. 2016. Web. 15 Dec. 2016. <https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/>.

Miller, Steven. "Steven Miller." *Mind: How to Build a Neural Network (Part One)*. N.p., n.d. Web. 15 Dec. 2016. <http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/>.

"Deep Learning." *NVIDIA Developer*. N.p., 14 Oct. 2016. Web. 15 Dec. 2016. <https://developer.nvidia.com/deep-learning>.

"Microsoft Azure: Cloud Computing Platform & Services." *Microsoft Azure: Cloud Computing Platform & Services*. N.p., n.d. Web. 15 Dec. 2016. <https://azure.microsoft.com/en-us/>.

"Back Propagation Tutorial." *Back Propagation Tutorial*. N.p., n.d. Web. 15 Dec. 2016. <http://www.cse.unsw.edu.au/~cs9417ml/MLP2/>.

Alpaydin, Ethem. *Introduction to Machine Learning*. Cambridge, MA: MIT, 2004. Print.