**Author:** Cameron A. Pickerill-Trinitapoli

**ID:** G03573349

**Sources:**

[Wikipedia](http://en.wikipedia.org/wiki/Artificial_neural_network) [on](http://en.wikipedia.org/wiki/Artificial_neural_network) [Neural](http://en.wikipedia.org/wiki/Artificial_neural_network) [Nets](http://en.wikipedia.org/wiki/Artificial_neural_network),  
[AI](http://www.ai-junkie.com/ann/evolved/nnt1.html) [Junkie](http://www.ai-junkie.com/ann/evolved/nnt1.html) [on](http://www.ai-junkie.com/ann/evolved/nnt1.html) [how](http://www.ai-junkie.com/ann/evolved/nnt1.html) [to](http://www.ai-junkie.com/ann/evolved/nnt1.html) [create](http://www.ai-junkie.com/ann/evolved/nnt1.html) [a](http://www.ai-junkie.com/ann/evolved/nnt1.html) [simple](http://www.ai-junkie.com/ann/evolved/nnt1.html) [Neural](http://www.ai-junkie.com/ann/evolved/nnt1.html) [Net](http://www.ai-junkie.com/ann/evolved/nnt1.html),

[The](http://tinyurl.com/mfxhctn) [Clever](http://tinyurl.com/mfxhctn) [Machine](http://tinyurl.com/mfxhctn) [on](http://tinyurl.com/mfxhctn) [BackProp](http://tinyurl.com/mfxhctn) [and](http://tinyurl.com/mfxhctn) [Gradient](http://tinyurl.com/mfxhctn) [Descent](http://tinyurl.com/mfxhctn)

David Miller on [Neural](https://vimeo.com/19569529) [nets](https://vimeo.com/19569529) [in](https://vimeo.com/19569529) [C++](https://vimeo.com/19569529), and [Care](https://vimeo.com/56882963) [and](https://vimeo.com/56882963) [Training](https://vimeo.com/56882963) [of](https://vimeo.com/56882963) [Your](https://vimeo.com/56882963) [Neural](https://vimeo.com/56882963) [Net](https://vimeo.com/56882963)

& Epp on directed graphs.

**Simple Neural Network**

This project is an example of a simple neural network that can be used flexibly to solve various types of problems. This project is a simple extension of project 2, which used this same Neural Network concept and code in order to implement a simple logic circuit. This project is meant to demonstrate that the concept, as well as the code, can easily be scaled to a more complex problem. In this case the Neural Net is being used as a controller for a non-player character in a hypothetical video game. The video game in question will be referred to as the “primary program” from this point on. The inputs represent the distance from the player, and the actions the player is taking. Although inputs and outputs are limited for this example, the program can easily take a wider range of input and output values in order to demonstrate a variety of actions. In a real game the outputs would correspond to actions particular to that program and the target would be the result of those actions instead of a fixed response. By making our target value the desired reaction (the player losing hitpoints, or the non-player character surviving for example) we can teach the neural net to choose between different output values (actions it can take) based upon its past experience with the effectiveness of those choices.

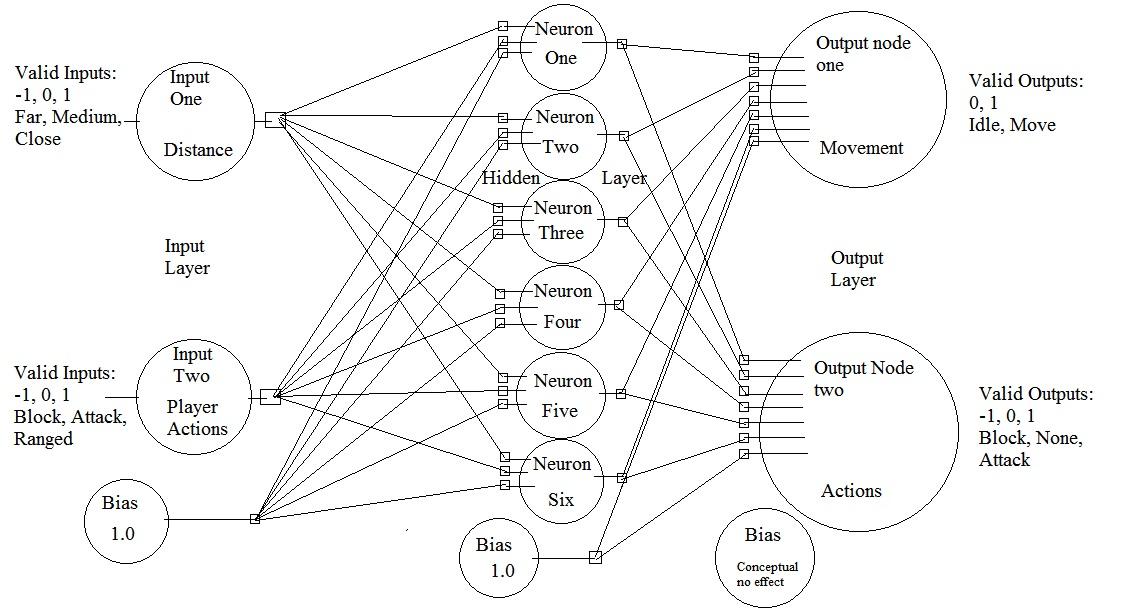
This method of course has its downsides, for example it is much more resource intensive than a fixed algorithm, and may not always utilize the optimal method. However, for a project in which you wish the behavior to change in response to the situation or in which you have no certain knowledge of what the best solution is it provides you a result you cannot obtain with a fixed algorithm.

As this program relates mathematically to the course, a Neural Network is a directed graph of variable size. This program may have any number of inputs, any number of hidden nodes, and any number of outputs. The input nodes feed into the hidden layer of neurons, and when some activation weight is achieved the hidden layer modifies the input and produces an output. This is in nature similar to a function as some input goes into the hidden layer, and we should reliably get some particular output from the output layer in response to it. What exactly goes on in the hidden layer varies from one data set to the next as the point is to solve a problem where the function that must be applied to the input in order to acquire the desired output is unknown. It’s worth noting that artificial neural networks of three layers are roughly O(n^2), I say roughly because the order complexity of a Neural Network can increase when variables (such as output nodes) are changed, and methods for calculating the complexity of Artificial Neural Nets are still a fairly active subject for research papers.

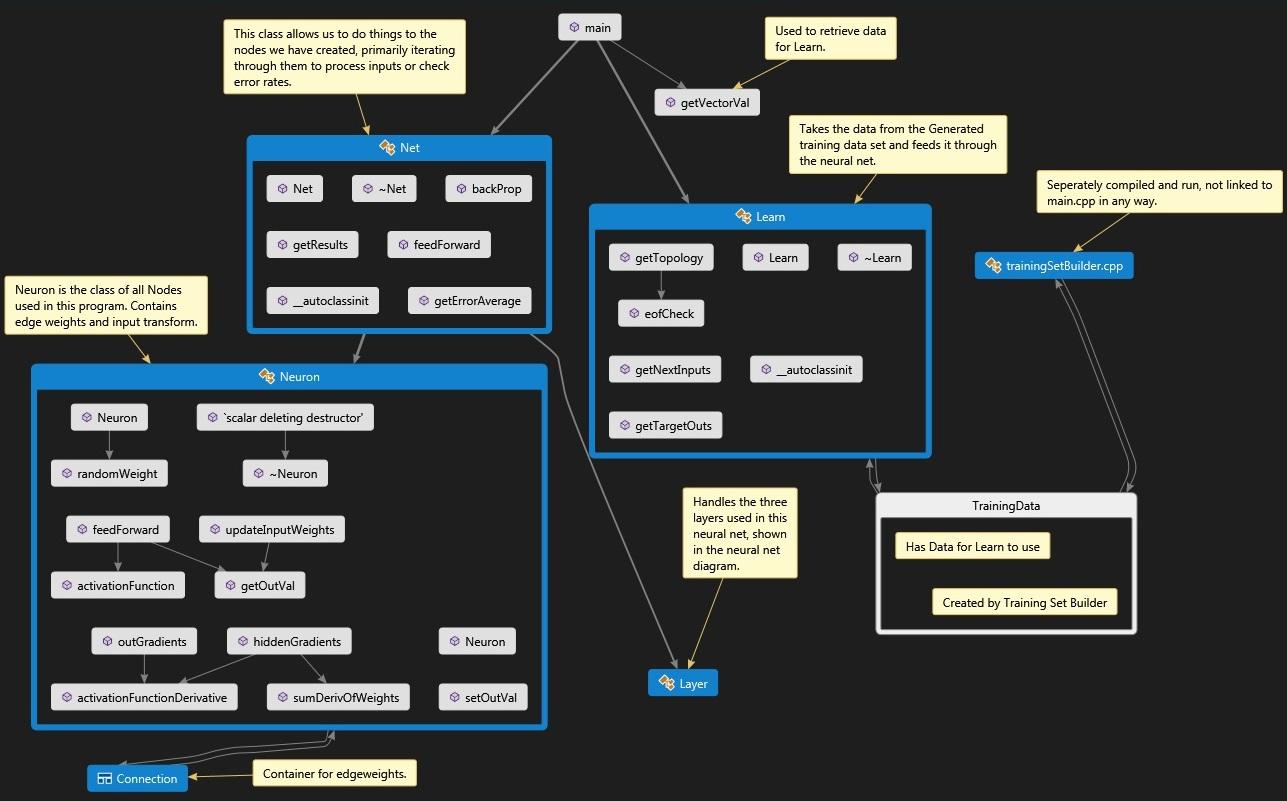
This directed graph also has weighted edges in order to simulate neuron activation. The exact edge weight values are randomized at run time, and then a gradient descent algorithm is applied in order to work towards the correct solution based on the current error rate. In this way the Neural Net effectively fumbles about for an answer using a method of checking whether it is it is getting “warmer,” or “colder,” like might be seen in a childrens game. When the net starts to get “warmer” it is given more weight in that direction (called momentum in the notes within the code), allowing it to continue in that direction more and more strongly until it closely approaches the target value.

Below is a diagram of what the Neural Network will look like during the operation of this particular program for this particular problem.

[[1]](#footnote-0)

The distance from a given target and what action the target is currently taking are read into the neural net, and then the net outputs a reaction to that data. In this example that means that we train the net to respond in specific ways to a given input. This can serve as a fairly flexible substitute to a AI decision tree. The input and output values as well as target values can easily be replaced with any number of similar actions, and this code can be used to power many different non-player character action sets in this way. Additionally more complex behavior can be achieved by feeding the outputs into the primary program and tying them to actions, then using the primary program we can feed the outcome of the actions taken back into the neural net as the outputs displayed in this example, and compare them to the desired outputs. This allows the program to decide what actions to take on its own to best achieve a particular goal based on the mechanics of the primary program. Thus innovative or unique solutions can be created to broad or varying problem sets.

This method may also be used to achieve adaptive solutions to a dynamic problem set. The Neural Net can attempt to work towards a solution and come up with what is in essence a best guess within a high and measurable degree of certitude. This can allow the Net to achieve good answers to particular situations-- input values-- and improve upon them as more data to train the Net on becomes available. By recording what actions are taken by the player in this example, you can test the Net against those actions, and it can improve itself in response to improvements on the part of the player, allowing you to keep your AI relevant instead of leaving some “best strategy” available to beat it. If we give the net feedback on its output values by doing some action in the primary program with the output value, and then comparing the result to the desired outcome we can work towards a degree of effectiveness instead of a specific value. By working towards some degree of effectiveness new output combinations will also be attempted in order to compensate for human players learning to avoid the actions taken in response to outputs.

[[2]](#footnote-1)

Microsoft Visual Studio has been used here in order to create a flow diagram of the program. It clearly shows how the Learn class acts as an interface with trainingSetBuilder.cpp and it’s output; TrainingData. Then the main function passes information between the classes giving Net the training data from Learn, and allowing the Net to learn to solve the problem set. Net acts on neurons using Layer to select the correct group of Neurons (nodes) to perform a given action.

**New concepts in this project.**

This project taught me a lot about the structure of classes in C++. I usually prefer to split my program into multiple .cpp files, however most of the best established methods for creating a simple neural network used single files with multiple classes, and since I lack the experience and time to completely write different code I chose to stick with the all in one file so I’d have a reference and/or easy access to help if anything went wrong. The upshot of this is that I’ve gotten very familiar with writing classes. Most of the rest of the program didn’t actually explore advanced C++, the new concepts in this program were primarily in mathematics such as the hyperbolic tangent function, the gradient descent algorithm used, and a derivative equation I actually don’t fully grasp, mostly because I don’t have enough calculus experience for it to look like anything but a pile of symbols to me. Another valuable thing I learned from this project is how enjoyable it is to code such an interesting and flexible program. I’m looking forward to implementing this in a few of my personal projects as it has a couple of neat uses in generating unique but semi-random content.

Not much went differently than expected although my first idea for an example problem failed (controlling some kind of fictional furnace with pressure, heat, and something else for inputs and outputs). That example was poorly thought out and I had to fall back on a logic gate for project 2. For this project I had some similar thoughts initially but it still didn’t work. Eventually when I was brainstorming for the project proposal I realized I might be able to use a gaming example. That didn’t go well at first either until I thought to draw up a full diagram of the neural net to make sure my setup for the problem was possible which I showed above. This ended up greatly expediting the small amount of rewriting of the program required as well as writing both the project proposal and this write up to a degree.

I also found a number of errors left over from project 2 that I didn’t expect. This accounted for previously unexplained odd behavior with certain problem sets and outputs approaching incorrect specific values while still technically simulating the correct behavior for simple problems. It was an interesting set of issues to solve since solving the first bug led to several other errors becoming much more problematic. However, now the performance and accuracy of the Neural Net have been substantially increased.

**User Guide**

Welcome to Simple NN AI Addon by C.A.P.T.

In order to test the implementation of this program please simply run testit.sh and then examine the resulting “outfile.txt”. You should be able to clearly see the process by which the outputs rapidly approach the target output corresponding with a given input, and eventually reaching an error rate of less than 1%. In order to configure the training file for your specific output simply edit the trainingSetBuilder.cpp file to create inputs and outputs appropriate for your project. For the most part it should be possible to simply read outputs from the neuralNet into your program and take appropriate action once you have supplied suitable training material to prepare the Neural Network for your task. More Inputs or outputs may be added as needed although if the number of outputs is increased you will also need to add more target values which requires some experience in C++ beyond simply editing values. If you wish to use your own training data builder or some previously created set of data simply convert it to share the same format as the trainingSet1.txt found in the project data folder. In order for a file to be automatically read by the neuralNet program it must replace the “trainingSet1.txt” with the same filename, extension, and path. Otherwise the path will need to be edited within the main.cpp file.

Keep in mind that the topology may need to be altered for various problems; for example, the hidden layer (the second argument in line 10, topology: 2 6 2 of trainingSetBuilder.cpp ) should be increased if the number of inputs or complexity of the problem is significantly increased. To evaluate the effectiveness of such an action run testit.sh and check the error rate of a problem at a high round number (greater than 1000), if the error rate increases after raising the number of nodes in the hidden layer, then the previous value was sufficient. This requires some experimentation as the actions of neurons in the hidden layer are not the same from one problem to the next.

Please note: this program is not intended for for a user without some minor c++ experience and access to a code editor. This program is meant for use as an add on to an existing program and not as a standalone build. C.A.P.T. Is not responsible for anything that happens as a result of using this program without the required skills, nor for anything that may occur to your computer while attempting to use this program.

**Potential Alterations**

If I had to do this project again, I think I would make a number of changes to the build. First, I would seperate most or all of the classes into separate .cpp files in order to further compartmentalize the program and increase readability. Then, I would replace the entire input interface with something a little more flexible, and possibly with a user interface option and some debug tools. I would also write a more robust training data creator that could create a variety of problem sets based on user input instead of a specific one. Plus, I would convert the program to work towards a desired result rather that specific value (by taking the output, doing something with it using a small test program, and then using the test program to create a “final” output to compare against the target output. This change is fairly minor but one that must be done to use the program to solve problem sets that are not purely data crunching.

If I were to extend the program I would add in the ability for the trained Neural Network to be saved, loaded, and then improved via additional training. As well as a method for taking incoming data and converting it to training data format I’d also add in options for the algorithm used to improve the program as the randomness of a genetic algorithm method can be more interesting for dynamic live problems even if it achieves a similar result to a good gradient descent algorithm. Genetic algorithms are also inherently good at avoiding local maxima, where as I would need to make some improvements to the gradient descent algorithm in order to give it similar improvements, and those improvements would require some time for me to learn how to implement. I would also make the training data set building more flexible to allow it to dynamically increase or decrease both inputs, hidden layer nodes, outputs, and target values.

**Time Allotment**

This project has been pretty much on track as planned and this makes a lot of sense as projects 1 and 2 were on topics I had little experience in and that stretched my programing ability far past what I’m comfortable coding from scratch normally. Project 3 is a small improvement to project 2, combined with significant bugfixing and a substantial writeup improvement. This has meant both that I have only had to write a few fresh lines of code, and that I was able to work with something I have become somewhat familiar with from my time working on project 2. In total I spent approximately 6 hours on the program, about 5 of which were spent debugging. I expect to have spent 6-8 hours on this write up by the time I finish it.

**Final Notes**

This really depends on how we define the customer expectations. If they expected to receive what I proposed for each project proposal I think my hypothetical customer would be satisfied. This write up fairly clearly demonstrates what the program can do, how it does it, and how to use it. I further feel that this fairly accurately lives up to the expectations of what it --the program --should do. I don’t think they would be 100% satisfied, as it is hard to match an end result perfectly with an initial proposal, or at least I lack this ability as of now and it is likely they will expect something somewhat different than what we actually agreed upon depending on their perspective on programming and computers. It will definitely work as intended though so I doubt they would be dissatisfied with the work.

If I was publishing to a magazine I would include more work, primarily a specific example of this hypothetical game problem or a different problem that only requires a training data set alone to fully demonstrate the program. This program is written as if for a customer with an at least partially working knowledge of a primary program, where-as for a magazine I feel I would need a concrete example for this program as readers might not be familiar with the hypothetical program, where-as a customer would already know what they wanted the program for, and what they wanted out of the program.

1. Fig1 and fig2 can be found in the document folder as .jpg files for higher resolution [↑](#footnote-ref-0)
2. Fig1 and fig2 can be found in the document folder as .jpg files for higher resolution [↑](#footnote-ref-1)