Abstract

Ethan Tanen

This paper explores the artificial neural network, a network of simple computational units connected by weights. The artificial neural network nearly parallels the biological brain, in that each node in the network receives inputs from nodes in the previous layer, computes an output based on the inputs and propagates the signal to the next layer in the network. The signal eventually arrives at the output nodes where the result is processed. The paper, more specifically, explores a neural network trained to perform hand written digit classification based on data gathered from the MNIST dataset. A single serial implementation and two parallel implementations - an openmp implementation and a cilk implementation - are taken into consideration for the analysis. The number of hidden nodes was increased by five for each new trial. The number of hidden nodes directly effects the amount of work that can potentially be completed in parallel. Execution time for the three networks increased linearly. The parallel implementations, on average, complete the training process in half as much time. The cilk implementation has unpredictable performance when looked at from an accuracy perspective. The openmp implementation performs consistently better than the serial version albeit significantly faster.

Introduction

Ethan Tanen

Artificial intelligence has been sought after and researched since the dawn of computing. Artificial intelligence, in the colloquial sense of the word, refers to a machines ability to parallel the cognitive function of a human brain. This ranges from speech recognition to disease diagnosis and maneuvering through and manipulating a machines local environment. Up until recently technology was not at a point where convincing artificial intelligence like behavior could be successfully implemented. The most convincing forms of artificial intelligence in the 20th century took the form of simple chat bots which were hardcoded to produce a response from a specific input. As recently as the first decade of the 21st century, computer hardware has allowed for machine learning to impact many fields. A widely used technique is the artificial neural network.

Neural networks have been a popular technique in the pursuit of artificial intelligence and machine learning for some time now. Neural networks, on a broad scale, parallel the architecture of the biological brain. The network consists of a series of interconnected and simple computational units that work together to produce a desired result from a given input. Each node in the neural network receives inputs from nodes in the previous layer and produces an output based on the nodes activation function. What is neat about neural networks is their ability to reorganize their internal structure to produce more accurate results. This ability to self update has brought about a paradigm shift in the way software engineers construct their programs. When using a neural net to solve a problem, the solution does not need to be hardcoded into the program, the program can learn how to solve the problem itself.

The neural networks lifecycle can be broken down into the training phase and the application phase. During the training phase the network begins as a series of nodes connected by sudo randomly assigned weights. A series of test inputs are fed to the net. Once the signal has propagated to the output nodes the values at the output nodes are compared to the target outputs and passed into an error function. The net than works to minimize the error in the system by modifying the weights. The exact mechanism for calculating the error and updating the weights varies from system to system. After the networks error falls below a predefined threshold the network can move into the application phase of its life cycle. The set of weights for the net can be copied into any other network with the same activation function, node count and error function to produce identical results.

A common application of neural networks is image recognition. An easy problem to tackle of this form is digit recognition due to a high availability of data and resources. The MNIST data set consists of 60,000 training examples of hand written digits. The images come in part from high school students as well as from employees of the United States Census Bureau yielding a wide variety of handwritings. The parent program for the network receives an image which is then deconstructed layer by layer into a single dimensional array for use as input to the network. The network outputs an array of ten numbers that represent the digits 0-9.

Design

I implemented a feed forward artificial neural network with backpropagation that consists of three layers that are conventionally named the input layer, the hidden layer and the output layer respectively. The network is of type “feed forward” because the flow of information is unidirectional, from the input nodes to the output nodes. The node count in each layer is variable. The node count in the input and output layers are determined by the input vector and output vector. There is not a strict method for determining the number of nodes in the hidden layer.

Backpropagation is the name given to a simple algorithm used to minimize the error function. The algorithm determines the contribution of every node to the total error of the system by calculating the partial derivative of each node with respect to the error function. The weights between nodes are updated by a small amount each iteration so as to ensure the minimum error is not passed or oscillated around. The following error function was used for the netowork:

This is a popular choice for the error function because it is easily differentiable. Another name for this algorithm is gradient descent. If we were to picture what the algorithm is doing graphically, we would first note that the error function is quadratic. If we imagine the current state of the network as a ball on the graph of the error function than with each iteration of the algorithm the gradient at the balls current position is calculated, then the ball is moved a small bit in the opposite direction which results in the illusion that the ball is rolling downhill. In practice its slightly more complicated because there is a chance that the ball could get stuck in a local minimum resulting in the networks inability to train any further.

The activation of each neuron is determined by the following equation:

and the output of each neuron is determined by the following activation function:

The sigmoid activation is commonly used because extremely high and low activation values result in an output near 1 and 0 respectively. This is highly similar to the original perceptron implementations of neural nets in the twentieth century.

As can be seen, the program requires many multiplicative operations. This is an obvious scenario in which parallel work could greatly reduce run time. I implemented two parallel versions of the net, one with openmp and the other with cilk.

The original cilk implementation attempted to exploit the dot product computation needed to calculate the activation of a node. Cilk\_for loops were used instead of the serial for. This proved to be not worthwhile due to the amount of overhead needed to setup the parallel region each time the forward propagation loop was run. The second and final implementation exploited the same region of the program to produce a faster runtime for the program. Rather than using the cilk\_for, cilk’s array notation was used which allows the computer to use the vector unit more efficiently. In doing so, a portion of the dot product calculation can be completed in parallel.

The openmp implementation attempted to exploit the neural network higher up in the abstraction. Before continuing it is necessary to introduce the term ‘epoch’ which is a number that represents the number of times the program has run through the training set. The openmp implementation broke the loop that facilitates the number of epochs into two separate threads that would run simultaneously on half as many epochs. Each thread is given its own set of weight arrays in order to prevent race conditions. After the training period completes the two sets of weight arrays are combined by taking the average of a weight and its counterpart in the other array. A serious concern for this implementation is that the two parallel neural networks make restructure their innards completely differently.

Analysis

In order to gauge the performance of the different implementations, data was retrieved on the speed of training the net, error in the system after training the net and and accuracy of the net during the application phase. The number of hidden nodes varies between trials. The number of hidden nodes will directly effect the amount of total work in the system and the amount of work potentially completed in parallel. The timer for the training phase begins just before the program enters the training loop and ends either when the nets error dips beneath the threshold or when the program runs through all the requested epochs. To calculate error in the system after completing the training phase, a series of ten images that the network has not been exposed to yet are fed to the network and system error is calculated exactly the same as in the training phase. The accuracy of the net during the application phase is analyzed by creating a vector of the outputs that correspond to the desired target and then calculating the error of this set using the same error function.

The graph above displays the hidden node count of the system and the corresponding time needed for execution; measured in milliseconds. The three different networks were tasked with training on three test images at an epoch count of 20,000. The execution time has linear correlation to the number of hidden nodes. For small hidden node counts the difference in time is nominal but as the hidden node count increases it becomes very clear that the parallel implementations are much faster. The parallel implementations run on average 50% faster than the serial implementation.

The graph above displays the relationship between the hidden node count and the total error in the system. The graph, as mentioned above, was tasked with training on three different test images at an epoch cap of 20,000. The resulting network was passed to the testing program where the error in the system was calculated based on an input of ten test images. The images that the three nets were trained on were the same. The same is true for the images that the nets were tested on. The graph shows a steady decline in the system error as the hidden node count increases, at least in the case of the serial implementation and the openmp implementation. The cilk implementation seems to steadily increase in error. It is reasonable to assume that the network error should remain consistent across implementations. This anomaly is either due to the mysterious workings of cilk or an issue with the implementation. Numerous tests have been conducted to ensure that the system is working correctly but it still begs the question: Why is the data inconsistent?

The above graph shows the relationship between the number of hidden nodes and the error between the target result and the actual output from the net. As mentioned above, this was completed by creating a one dimensional array of the networks outputs, only considering the values of the nodes that correlate to the target array. For example, if the target was sixth than only the sixth node in the output layer was added to this array. This array was than passed through the error function. The target for each element in this array is 1. The accuracy of the net in this regard remains constant for the serial and openmp implementation while steadily increasing for the cilk implementation. The mystery remains the same for the cilk case while the results for the serial and openmp implementations makes perfect sense. The system was only run through 20,000 epochs to ensure that the execution time for the training phase remained sufficiently low. This resulted in the networks inability to reach the error threshold before the number of epochs had expired. The networks were trained on the same data for the same number of iterations resulting in very similar outputs across variations in hidden node count.

Conclusion

The neural network is a fascinating piece of code due to its ability to parallel the biological brain and its ability to self update or learn. A neural network boils down to a long series of dot products and other multiplicative operations. By parallelizing the work, the network can easily double in speed while keeping the error consistent. This means that with a parallel implementation the network can be trained on more datasets quicker allowing for an increasingly more accurate network.

This was my first exposure to neural networks which makes for much room for improvement in future implementations. The program could be restructured to allow for more parallel work. For example, rather than passing the training set in one image at a time, the input could be a large two dimensional array that holds information for every input. This array with a sufficiently advance linear algebra unit would decrease the forward pass of the net substantially. The same is true for the backpropagation portion of the net but the details are far more nuanced. The net could also include tools such as “momentum” which is a term in the backpropagation algorithm that causes the hypothetical ball mentioned above to move quicker and quicker if the ball rolls in a similar direction as the iteration before. This causes the network to minimize the error function much quicker.

In the future, I would like to re-implement the code base with these new features. I would also like to redo the architecture of the program to make it more readable. Currently the bulk of the work is done in the main method. In future implementations the work would be distributed to other more refined functions. As a side project, an application that would allow users to draw their own digits to be test the neural network would be neat.

In terms of analysis, there remains a substantial amount of information that can still be extruded from the net. For example, questions such as: How does layer count, learning rate and input size effect accuracy and speed?