Super intelligence

New Mexico Super
supercomputing Challenge
Final Report
Friday, March 17, 2006

Team Number 44
Las Cruces High School

Jorge Palma
Jeremy Wilson
Elizabeth Jurado

Teacher: Mr. Marez
Mentor: Nick Bennett

Short Statement

This project was very challenging. The way the program operates has changed over five time during the course of this year. The two biggest changes happened in February. The idea for our project came from a book called On Intelligence. This book taught us that sequences of patterns were the key to truly usefull Neural Networks.

Summary

By analyzing a book called On Intelligence by Jeff Hawkins we have come up with a type of neural network that functions more like the brain. Neural networks use training to give a desired response to an input. The brain dose not have a training mechanism with all the correct answers, it self organizes itself depending on what input it receives. The brain is also build on a hierarchy that uses feedback. Not feedback from the senses but through the entire brain with feedback connections. With these feedback connections the brain can remember different sequences of patterns. The best description of the neural network we have developed would be an unsupervised learning neural network without templates. Regular unsupervised learning neural networks arrange its nodes so that ones that are more like one type pattern are closer to one side. Weights of each node in our neural network start out random so different nodes can fire to different patterns. The weights connecting to a node (not the weights of a single node) adjusts if that node "has" fired. It adjusts them by weakening those that were off when it fired and strengthening those that were on when it fired. The weights change to match its input. How much they change depend on the number of nodes that fired in the next hidden layer (this is the feedback of our neural network). Adjustments do not depend on desired output. As it is exposed to more patterns it learns them by maxing the strengths to the extreme because not all of the nodes fired for one pattern it can learn more patterns.

Software Used

-Java (NetBeans)
-OpenOffice
-Microsoft Work
-Paint
-notepad

Procedure

1) The program will first look at patterns of music.
2) The state of all the nodes will be displayed in an array of nodes.
3) We will check to see whether the pattern of the nodes firing stabilizes.
4) We'll show a different pattern of music a repeated amount of times.

Results None

We have no results because it has taken a very long time to find the right approach to our problem and to program it. The results we would have liked to have gotten would be that the firing of the nodes furthest away from the input layer would stabilize after being exposed to many patterns repeatedly proving that it had learned a sequence of patterns.

Conclusion

We have concluded that this program will probably become the basis of future artificial intelligence.

What we are Most Proud Of

Our program of course. Out program was made and remade several times and every time we changed something about it it improved. At first we had 2 separate ideas as to how we should simulate intelligence. Our first idea was to simply write out a program that would have every single possibility of a form of typing. And if the sentence that was not in already predetermined appeared the program would simply respond with a “I'm sorry I didn't get that” then it was added that maybe instead of recognizing words we have it recognize images. Then the problem that if this was done the program would have to archive every single image that appeared and would not be able to recognize it other wise. So then we all agreed that we should simply write a program that could emulate the brain. After a long time we finally we got the program to show the states of all the nodes which is interpreted by our very own Jeremy Wilson.

Special Thanks To

Mr. Marez our good Teacher: thought us how to use netbeans to create Java program.
Nick Bennett: helped us debug our Java program
Dr. Shaun Cooper: He advised us

References

Hawkins, Jeff. On Intelligence. Henry Holt and Company. New York 2004

Program

```
import javax.swing.*;
/*
 * TestBrain.java
 *
 * Created on October 30, 2005, 4:41 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management node. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

/**
 *
 * @author Gregory Marez
 */
public class TestBrainNode {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        BrainArray jeremy = new BrainArray();
         jeremy.runBrainNodes();
         jeremy.inputVariables();

        // TODO code application logic here
    }
}
```

Program2

```
import javax.swing.*;
/*
 * TestBrainArray.java
 *
 * Created on April 3, 2006, 3:00 PM
 *
 * To change this template, choose Tools | Options and locate the template under
 * the Source Creation and Management nodes. Right-click the template and choose
 * Open. You can then make changes to the template in the Source Editor.
 */

/**
 *
 * @author Jeremy Wilson
 */

   public class BrainArray {
   int down,n=0;
   int [] up;
   double time;
   BrainNode [][] nodes;

/** Creates a new instance ofBrainNode */

   public BrainArray ()
   {
     nodes = new BrainNode[20][20];
         for(int i=0; i<20; i++)
         {
               for(int j=0; j<20; j++)
               {
                  System.out.println(" "+i+" "+j);
                nodes [i][j] = new BrainNode(20);

               }
         }
   }
//
// ----------------------------------------------------------
//
   void runBrainNodes ()
   {
     nodes = new BrainNode[20][20];
         for(int i=0; i<20; i++)
         {
               for(int j=0; j<20; j++)
               {
```

```
                nodes [i][j].Pow2(n);
                nodes [i][j].addStrengths();
                nodes [i][j].getNodeValueBrainCell();
                nodes [i][j].changeStrengths(down,up);
               }
          }
     }
//
// ------------------------------------------------------------
//
    void inputVariables()
    {
       System.out.println("In program Run-1"); // ------------  #1
       for(int i=1; i<20; i++)
       {
          System.out.println("In program Run-2");// ------------  #2
          for(int j=1; j<20; j++)
             {
              down = down + nodes[i][j].getNodeValueBrainCell();
             }
          for(int j=1; j<20; j++ )
             {
              nodes[i+1][j].changeStrengths(down, up);
              nodes[i][j].Pow2(n);
             }
         down = 0;
         n++;
         if (n == 20)
            n = 0;
       }
//
// ------------------------------------------------------------
//
      System.out.println("In program Run-4");// ------------  #4
       for(int i=1; i<20; i++)
       {  System.out.println("In program Run-5");// ------------  #5
          for(int j=1; j<20; j++)
             {
              up[j] = nodes[i][j].getNodeValueBrainCell();
             }
          for(int j=1; j<20; j++)
             {
              if (i > 1)
              nodes[i-1][j].changeStrengths(down, up);
             }
       }time = time + 1/30;
        System.out.println("In program Run-6");// ------------  #6
    }
}
```

Program3

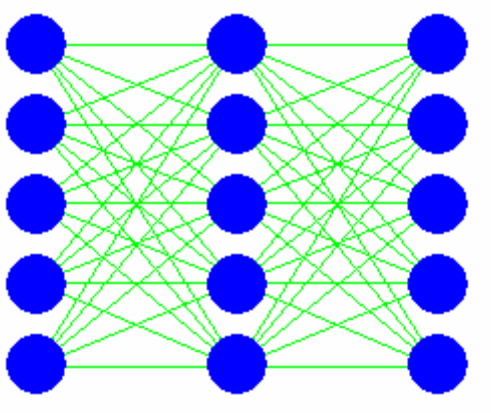
```
import javax.swing.*;
/*
 * Title: BrainNode
 * Programmer: Jeremy Wilson
 * Class Names: BrainNode - Constructor BrainNode ( i )
 *
 * Version: 1.0
 * Method name
 * ACTION METHODS
 *        1. addstrengths           - Add strengths that correspond to an upInput value of "1"
 *        2. getNodeValuebrainCell   - Compare sumStrengths to threshold
 *        3. changestrengths         - Change strength variables when done
 *
 * Date:4/3/06
 * Description: This Class will model functions of a brainCell.
 */
public class BrainNode {
    int switchStrengths,x,x2,y,y2,q,n,k,upInput[],downInput,
    strengths[],strengths2[],sumStrengths,brainCell,threshold=1000;
    String row[],output;

    /** Creates a new instance of BrainNode */
    public BrainNode (int i)
    {
    this.inStrengths();
    }

//
// Create random strength values ---------------------------------
//
 public void inStrengths ()
 {
    for (int i = 0; i < strengths.length;i++){
         x = (int) (Math.random() *10);
         y = (int) (Math.random() *150);
         if (x >= 5)
            y = y + 350;

         x2 = (int) (Math.random() *10);
         y2 = (int) (Math.random() *150);
         if (x2 >= 5)
            y2 = y2 + 350;

         strengths [i] = y;
         strengths2 [i] = y2;}
 }
```

```
//
// --------------------------Pow n ------------------------------
//
 public int Pow2 (int n)
 {
        if (n == 1)
           k = 1;
        if (n == 2)
           k = 2;
        if (n == 3)
           k = 4;
        if (n == 4)
           k = 8;
        if (n == 5)
           k = 16;
        if (n == 6)
           k = 32;
        if (n == 7)
           k = 64;
        if (n == 8)
           k = 128;
        if (n == 9)
           k = 256;
        if (n == 10)
           k = 512;
        if (n == 11)
           k = 1024;
        if (n == 12)
           k = 2048;
        if (n == 13)
           k = 4096;
        if (n == 14)
           k = 8192;
        if (n == 15)
           k = 16384;
        if (n == 16)
           k = 32768;
    return k;
 }

//
// Add strengths that correspond to a upInput value of "1"
//
 public void addStrengths ()
 {
    q++;
    for (int i = 0; i < upInput.length;i++)
     {
```

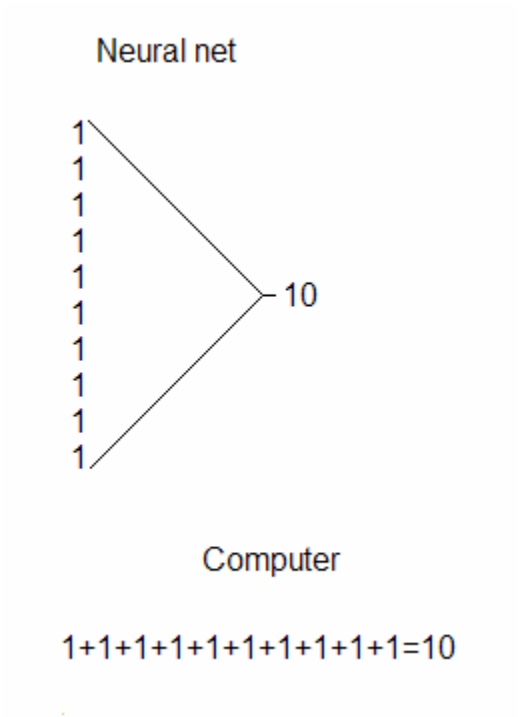
```
        if (q == k)
           switchStrengths++;
           q = 0;

        if (switchStrengths == 3)
           switchStrengths = 1;

        if ((upInput [i] == 1)&&(switchStrengths == 1))
           sumStrengths += strengths [i];
        if ((upInput [i] == 1)&&(switchStrengths == 2))
           sumStrengths += strengths2 [i];
     }
 }
//
//  Compare sumStrengths to threshold  ----------------------------
//
 public int getNodeValueBrainCell ()
 {
    {
     if (threshold < sumStrengths)
         brainCell = 1;
     else
         brainCell = 0;
     }
   return brainCell;
 }

//
//  Change strength variables to better match upInput ------------------------
//

 public void changeStrengths (int downInput, int[] upInput)
 {
    for (int i = 0; i < upInput.length;i++)
     {
        if ((upInput [i]     == 1)
          &&(brainCell       == 1)
          &&(switchStrengths == 1))
                strengths [i] = strengths [i] + downInput;
        if ((upInput [i]     == 0)
          &&(brainCell       == 1)
          &&(switchStrengths == 1));
                strengths [i] = strengths [i] - downInput;
        if ((upInput [i]     == 1)
          &&(brainCell       == 1)
          &&(switchStrengths == 2))
                strengths2 [i] = strengths2 [i] + downInput;
        if ((upInput [i]     == 0)
          &&(brainCell       == 1)
```

```
        &&(switchStrengths == 2));
            strengths2 [i] = strengths2 [i] - downInput;
    }
 }
}
```

Artificial Intelligence has been a series of failures and successes but mostly successes in the beginning of the 21st century. But many scientists have come to miss the point. Neural networks have long been used in A.I. but have failed to live up to expectations. The reason that they haven't is because none of them are the type found in the brain.

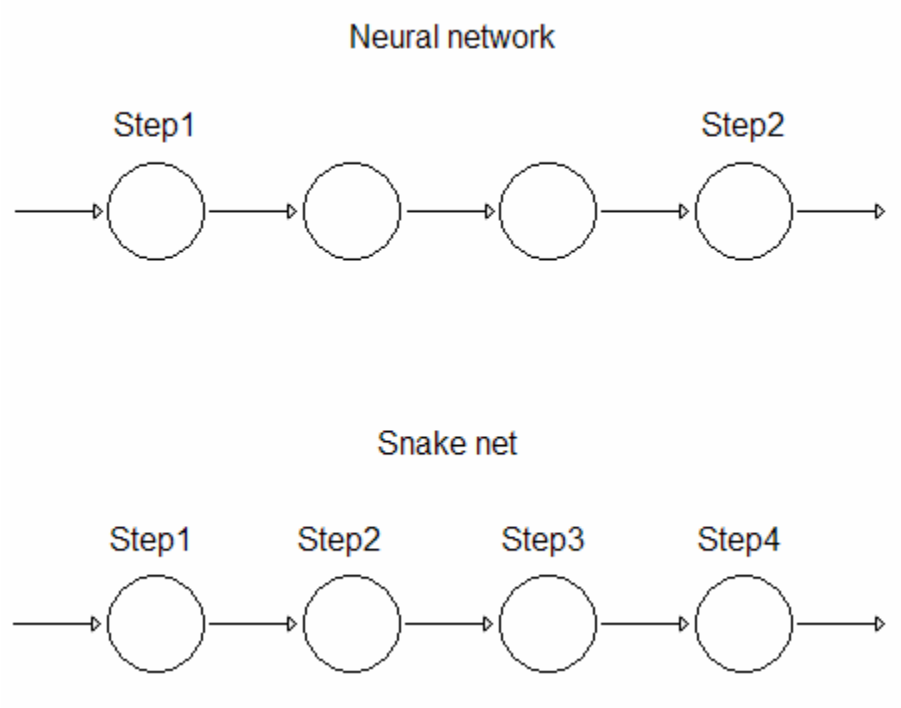


Neural networks are different then computers because they can do operations in one step.



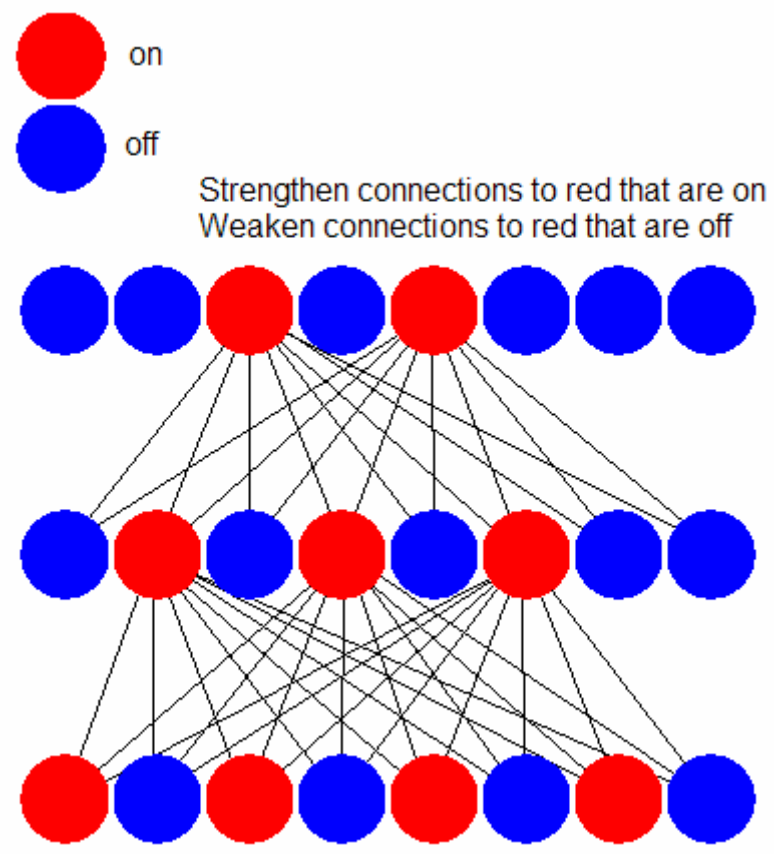
But neural networks are limited in that they don’t recognize sequences of patterns. Some types of neural networks can be given the data of the entire sequence at once but they can’t take the data in sequential order.

In a regular network information goes through all the Hidden layers at once. But in Snake net information goes through one at a time.

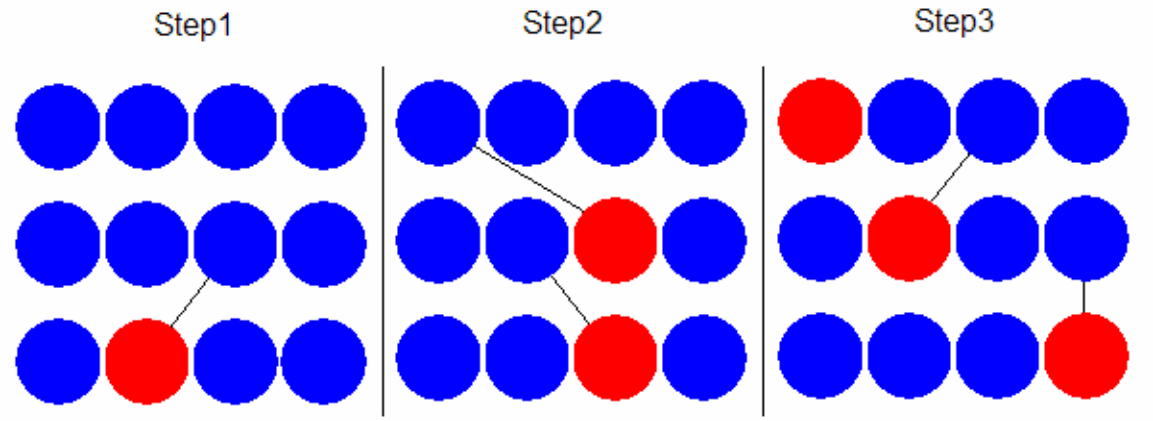


If you were to compare a neural network to talking it would be like saying the whole alphabet all at once. You couldn't understand it. Snake net would be like singing the alphabet. You would hear it nice and clear.

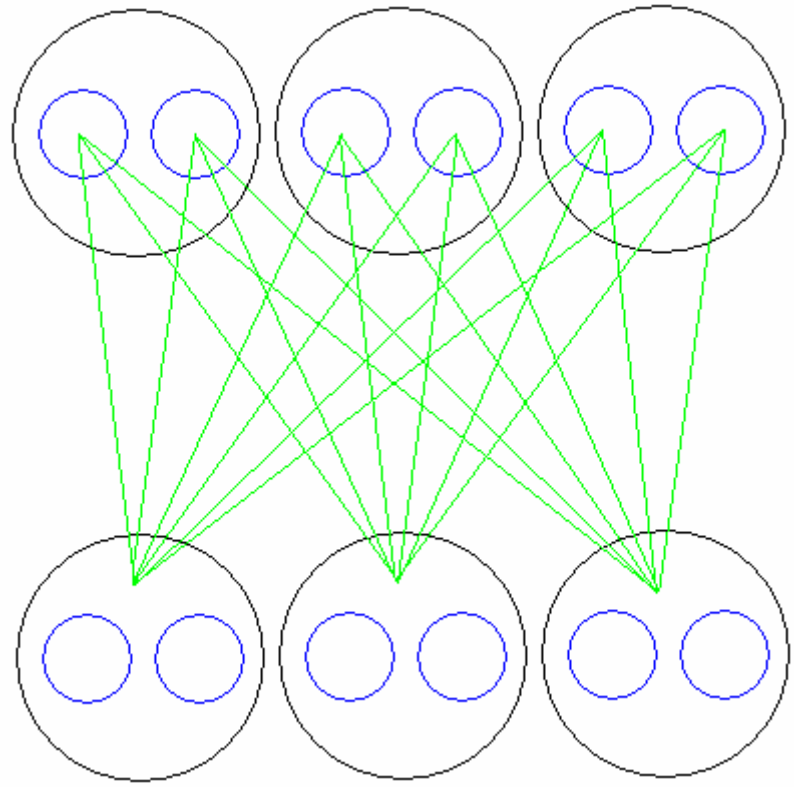
The training of a neural network has always been an essential part of its process. The process Snake net uses to train itself is very different from the way a normal neural network does. It does use an unsupervised training mechanism but it doesn't sort data into groups instead if a nodes threshold is reached it will change the weights connected to it. Strengthening those that are on weakening those that are off. This way hidden layers can be activated sequentially.



The nodes at the bottom of these three pictures are the input. If a nodes threshold is reached (indicated by the line to it) it will be on in the next step. It will also change its weights in the next step. More nodes can be activated but in this example only one is.

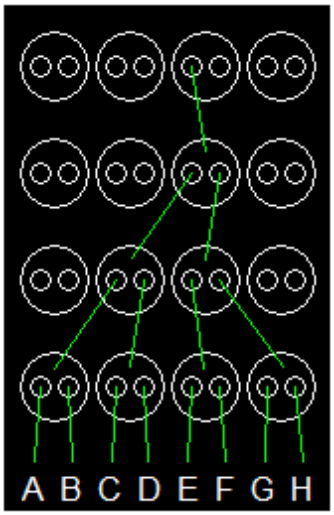


Inside each node there are two sets of weights. These weights sets switch off between each other. The weights change at different speeds in different layers. Faster near the input. Doubling each layer away from it.

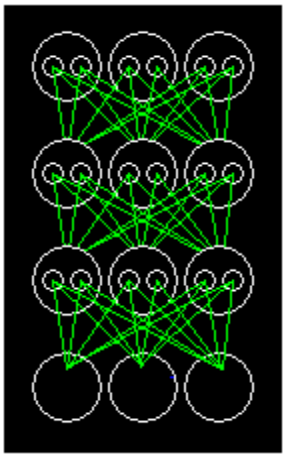


The nodes at the bottom send there singles to
the node at the top not the other way around

Hear is an example of how it would work.



Say letters in order and it will recognize it.



Our program uses neural networks to do this.