Introduction to Cognitive Science
Assignment 2: The Associative Net

Due Tuesday October 27th, 2009, hardcopy, in class

How is it that you notice a mouse behind the breadbox when all that is visible is its tail? How come you notice when somone mentions your name in the midst of a buzz of noisy conversation at a Cocktail Party—whatever *that* is—even when you have no idea what they said about you? What exactly is going on when you cannot remember the name of someone you are talking to, but know that it will come to you in a minute, and it does?

These are examples of "retrieval from partial information", "recognition from noisy input", and "content addressable memory". They are all best understood in terms of massively parallel distributed processing—(M)PDP—using what are somewhat metaphorically called "Neural Networks."

Since even a two dimensional mouse represents rather a lot of information, we are going to look at recognition etc. of "one-dimensional" mice, represented by bit-vectors, or ordered sequences of 0s and 1s. (Maybe noticing a snake behind the toilet bowl would have been a better example.)

1 The Associative Net

The network you are going to work with was invented by Longuet-Higgins, Buneman, and Willshaw (see Hinton and Anderson (eds.) 1981). It is called the "Associative Net." This device illustrates three basic properties of network models which are characteristic of mechanisms involved in phenomena of human memory and attention like those mentioned above. namely:

- Non-localized storage ("Distributivity")
- Ability to recover complete stored patterns from partial or noisy input ("Graceful Degradation").
- Ability to work even in the face of damage ("Holographic Memory").

Consider the following network, connecting 16 input units each of which has a directed connection to all and only 16 output units. The connection strengths all start at 0. Input is along the top, output from the righthand side.

vvvvvvvvvvvvvvvv
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
0000000000000000>
Input and output patterns consist of binary vectors. So the input and output units have activities of 1 or 0. To associate an input pattern with an output pattern, we simply set to 1 all of the connections between a unit that is on in the input pattern and a unit that is on in the output pattern. If the connection was already 1, we leave it alone. Connections can never go back from 1 to 0. We will consider the "auto-associative" case where the input and the output vectors are the same—since this is the easiest case to look at the effects of noise and damage in.

Below is the net storing the pattern 0110001110101001 as input in association with itself as output:

```
0110001110101001
vvvvvvvvvvvvvvvv
0000000000000000>0
0110001110101001>1
0110001110101001>1
0000000000000000>0
0000000000000000>0
0000000000000000>0
0110001110101001>1
0110001110101001>1
0110001110101001>1
0000000000000000>0
0110001110101001>1
0000000000000000>0
0110001110101001>1
0000000000000000>0
0000000000000000>0
0110001110101001>1
```

To retrieve an association, we sum the total input, x, received by each output unit from all the on switches in its row. Then we compare x with the number y of active input units, and threshold at y. That is, if $x = y$, that output fires, or is turned on. If $x < y$, the output unit does not fire, but remains off. (If $x > y$ there is something wrong with your program!)

Satisfy yourself that this procedure yields the same pattern 0110001110101001 as output from the memory shown above:

```
Input:   0110001110101001
Totals:  0880008880808008
Output:  0110001110101001
```

More interestingly, if we input the "damaged" or "partial" pattern 0000001110101001, we still get 0110001110101001 as output, though with lower signal strength (6 instead of 8), and therefore a lower threshold. Check this—you should get:

```
Input:   0000001110101001
Totals:  0660006660606006
Output:  0110001110101001
```

This is a bit like recognizing a mouse, snake, etc. when its head is behind the breadbox, or whatever.

Even more interestingly, if we "damage" a bit—say (7,7)—in the above memory, then from the original pattern 0110001110101001 we can still recover the complete associate, although this time we have to explicitly lower the threshold until we get something we know has been stored:

```
       |
       v
0000000000000000
0110001110101001
0110001110101001
0000000000000000
0000000000000000
0000000000000000
0110001110101001
0110001010101001<-
0110001110101001
0000000000000000
0110001110101001
0000000000000000
0110001110101001
0000000000000000
0000000000000000
0110001110101001
                 v
Input:   0110001110101001
Totals:  0880008780808008
Output:  0110001110101001
                 ^
```

This again is somewhat like biological memory. Damage does not result in the loss of a specific memory, but to a general degradation of recall in the form of increased noise.

The homework asks you to investigate the Associative Net. As usual, you could write a program, but you can easily be the computer yourself. You can get full credit either way.

2 What you have to do

Here are some 16 bit vectors with a random 8 bits set to 1—that is, with 1s and 0s equiprobable, $p = .5$. Don't worry, there are far more than you need! (It is possible there are some duplicates, but its so unlikely I haven't checked.)

1101001000111001	0100010110011101	1110100100110001
0011100101110100	1000111001010011	1000001100111011
0111000111001001	0011011001100101	1100010100011101
1010001000110111	1101010011110000	1101011001101000
1011000010011011	1000110011001110	0000100101111101
1010001001101110	1010010011110100	0010011000101111
0110101000110101	1101010100100011	1110001001001101
0111000101110100	0101011010000111	1000001101101110
1000001111101010	0101011010100101	1011110010010001
1100110101100100	1010011110010100	1010011011100100
1001011110100001	1011110000011001	0011110111000010
0110010010100111	0110011010101100	0110010001111001
0111110101000001	1001100001110011	1101100010110010
1001010000110111	1110011100010100	1010011100101001
0100110110100011	0011110010100101	1101110000100110
1100110001010101	1011100100110010	0101100000110111
0011001011001110	1010000111000111	1000000101011111
1100100001111100	0001110100111010	1011100110011000
0011111000101010	1101010001101010	0011001000111110
1011101001101000	1110110001000011	0111001010110001

Customize this set by crossing out the first n vectors where n is the last digit in your student ID.

1. Either implement, or just draw neatly by hand, a 16 by 16 Associative net. (If you draw it you may want to make some copies of the empty net. Computers are good for doing this as well)
2. Load the net with the first vector in the set after the ones you have crossed out, autoassociating this vector as input with itself as output. (If you are writing a program, you should still use the above random vectors.)
3. Retrieve the same vector from the net, showing the total outputs on each output unit and the translation into a binary vector. (You can either write

the totals and binaries on the right of each row of the network as in the first example, or as a triple of input, totals, and output, as in later cases.)

4. Change the first 1 in the input vector to 0 and show that the complete pattern is still retrieved, giving totals and binary result again.
5. Using further vectors in sequence (you will not need many!) you are to estimate how well the net retrieves associations by iterating the following procedure.

   (a) Add a new auto-association to the same network for the next vector on the list after the one you just added.
   (b) Retrieve *all* the previously stored patterns including the new one from the network.
   (c) Do this until there is some output which has at least one bit wrong.

   Give all your working, showing input, total, and output, for each retrieval. State the number of patterns you managed to store up to and including the last one which caused the error.
6. Ideally you should repeat the whole thing using several different sets of associations with fresh empty 16x16 nets, so you get an idea of how reliable this number is. (We will use the class average as a really good estimator). You do not need to give all your working for further sets, if you do any, although you may give as much as you wish.
7. By making experiments along the same lines with fresh nets and sequences of vectors with higher and lower densities of 1s, try to show how this probability affects the number of associations that can be stored without more than 1 error.

   Here is a bunch more vectors with four bits set, to help you, again far more than you need. Again, get rid of the first n where n is the last digit of your student ID.

1000000010000101	0001000110001000	1100000100000001
1010000010001000	0000010100001100	0000000010010011
0100001100000100	0000000011011000	0110100000000010
1000001010010000	0000001101000010	0000010000100101
1000110000000001	0000110100000001	0000011100010000
0100000100001010	0000001000001110	0000000010101010
0001000010010001	1000100100100000	0000001010101000
0000100110000100	0001001001000010	0000011100000100
1011000010000000	0010100100001000	0000000101001001
0000100100010100	1010000000100001	0100000000011100
0110000100000100	1000000110000100	0001010010000100
1000110001000000	1100000000000011	0000010001010010
0000100010001001	0000000100010101	0010000011000010
0110101000000000	0011000000110000	1101000001000000
0010000000001011	0100011000010000	0000011010000100
0100010000001010	0011100000000001	0011010010000000
1000010000001100	1100000110000000	1101000010000000
0000010011000100	0100100000000110	0001010010000010
1000000011010000	0001000000101100	0001100100100000
0001100001000001	1001001000000010	0110100010000000

(If you need vectors with other densities, just fake them up by making some appropriate number of these 0s into 1s, or 1s into 0s.)

Give your working—you can use intelligent guesswork as well as experiment for this part *if you tell us your reasoning*.

8. You have seen that the Willshaw net can recover the entire associated output for an input that has had one or more bits "hidden" or "damaged." Try to determine how the number of associations stored affects this ability to retrieve the complete associate from damaged input in comparison with retrieval from perfect input. Again, experiments or intelligent guesswork can be used but give your evidence in both cases.
9. In some sense the Associative Net "hallucinates" the whole snake when it "sees" part of it. There is no difference apart from the noise level. But it is important for a mongoose to know the difference between a snake and a snake behind the toilet-bowl, if mongooses are not to spend a lot of time bumping into things. What does this suggest to you concerning the possible psychological reality of the associative net? What is it good for if

it can't tell the difference between a snake and a snake behind the toilet? (*Hint:* Snakes are important things to notice, compared with other objects, so you may want to think in terms of a mechanism that separates *becoming aware* of snakes from the business of deciding *what to do* in a particular situation.)

Note: Write your name, school ID, and email address at the top of your homework. If you have multiple pages, *staple them together!*

Reference

Willshaw, David: (1981), "Holography, Association and Induction," in Geoffrey Hinton and James A. Anderson, (eds.), *Parallel Models of Associative Memory*, Hillsdale NJ: Erlbaum. 83-104.

mjs October 18, 2009