CPS331 Lecture: Neural Networks last revised October 26, 2018

Objectives:

1. To introduce the overall concept of neural networks
2. To discuss supervised learning in neural networks
3. To show some other kinds of NN models

Materials:

1. Projectable of Stillings p. 273
2. Projectables of PDP 27, 28 (Jets and Sharks)
3. Handout of Jets and Sharks data (PDP 27)
4. Demonstration of Jets and Sharks - executable program
5. Projectable of single perceptron
6. Projectable of example (AND behavior)
7. Perceptron demo
8. Letter-Learner demo (in Sheepshaver)
9. Projectable of general structure of a classifier
10. Projectable of a one output two input two hidden feed-forward network (not layered)
11. Projectable of a one output two input two hidden layered network
12. Projectable of step and logistic transfer functions
13. xornet learning program
14. Projectable of NIM Learning Architecture
15. Projectable of NIM Learning Example
16. NIMNet demo
17. ShapeAssociator demo with patterns
18. Projectable of Figures 1.2, 1.5 from Goodfellow et al.

I. Introduction

A. We have looked at one approach to learning that draws its inspiration from biological learning - genetic algorithms/programming. Today we look at another - an approach based on the very structure of the human and animal brain.

1. This approach goes by a number of names

a) Connectionist models or connectionism

b) Parallel distributed processing (PDP)

c) Neural networks

2. Though connectionism originated with the study of the physical structure of the brain and nervous system, it is not necessarily restricted to exact imitations of them

B. In order to understand the origin of connectionist models, it might be helpful to look briefly at some of what we know about the physical structure of the brain and nervous system.

1. The brain and nervous system are composed of cells known as NEURONS.

PROJECT: Stillings page 273

a) The dendrites are capable of receiving signals from other neurons, and the axon passes signals to other neurons. That is, signals travel from the dendrites to the cell body (soma), and from the cell body to the axon. (Typically, the signals received by the dendrites come from the axons of other neurons, except in the case of neurons that are part of our sense organs.)

b) The neuron does not simply transmit signals from one end to the other, however. Very weak signals at the dendrites cause no output at the axon. However, under sufficient stimulus at its dendrites the neuron FIRES, causing a uniform, strong signal to propagate down its axon. (This, then, is a form of amplification of the signal: very weak signals are ignored, while less weak signals are converted into strong ones.) The neuron's firing RATE is a measure of its degree of activation. Certain computational processes control the firing.

(1)The probability or rate of firing of the neuron is governed by the strength of the signal(s) at its dendrites.

(2)Separate signals arriving at the same time at different dendrites are summed, so that their effect is the same as that of one stronger signal arriving at a single dendrite. This is called spatial summation.

(3)Also, separate signals arriving at different times at the same dendrite are summed, so that their effect is the same as that of a single, stronger signal arriving at that dendrite - provided that the interval between signals is short. This is called temporal summation.

2. The connections between the axon of one neuron and the dendrites of others play a crucial role in neural processes. Within the neuron itself, signals are propagated electrically; but at the connections they are propagated chemically. These connections are known as SYNAPSES, and are actually of two different kinds:

a) Excitatory synapses: activity of the presynaptic neuron increases the rate or probability of firing of the postsynaptic neuron.

b) Inhibitory synapses: activity of the presynaptic neuron decreases the rate or probability of firing of the postsynaptic neuron. (That is, when summing the signals on the dendrites to determine overall input, activation of an inhibitory synapse is treated as a <u>negative</u> number.)

c) The synapses themselves are of varying degrees of responsiveness. In particular, it seems that, as learning occurs, certain synapses become more sensitive. Thus, activity on the presynaptic side of the synapse becomes increasingly likely to stimulate activity on the postsynaptic side (or inhibit it, as the case may be.)

3. In biological nervous systems, knowledge is not represented by the neurons themselves - but rather by the pattern of CONNECTION of the neurons. Neurons do not have memory - they fire when activated. Memory is embodied in the location and strength of the synapses.

4. In biological brains and nervous systems, information processing is a parallel process, which is often described in terms of SPREADING ACTIVATION. As certain neurons fire, they activate other neurons to which they are connected - which in turn activate other neurons etc. Feedback loops may serve to keep the whole pattern going. Each pattern that a particular portion of the system "knows", then, is related to a particular pattern of activation of the neurons of which it is composed.

5. The following summarizes the key features of neural computation:. (Quoted from Stillings pp. 505-506)

   "In biological systems, parallel computation is achieved by a special kind of hardware, the neural network. A neural network is composed of a large number of discrete units (neurons) each of which has many connections with other units. The information-processing capacity of a single neuron appears to be extremely limited in certain respects. The response time of a neuron is on the order of a few milliseconds ... The range of neuronal signaling frequencies is also quite slow, on the order of hundreds of spikes per second. As far as researchers now know, the connections (synapses) between neurons are either excitatory or inhibitory. That is, information reaching a connection merely serves to excite or inhibit the target unit to a greater or lesser degree. The major consequence of these considerations is that the speed and information transmission capacity of a single neuron is not adequate to support the transmission of complex symbolic information of the kind that is transmitted along the internal and external signal paths of digital computers."

> "Researchers generally agree that these arguments point to the inescapable conclusion that the computational significance of a single neuron lies in its pattern of connections with other neurons. The computing power of patterns of connections is truly impressive. The human visual system, composed of neurons that have a cycle time of a couple of milliseconds, can identify a complex object in about 100 milliseconds, or something like 50 computing cycles, which is utterly insignificant by the standards of electronic digital computing. Connectionist theories in cognitive science try to stay reasonably close to these fundamental facts, without trying to model the nervous system in detail ..."

C. While some of the earliest work in this area was done with hardware simulations of neurons, work today is done using software simulations.

D. Neural networks build on the basic ideas of how the brain is structured.

1. A neural network consists of a collection of simulated neurons. These neurons can be of three kinds

   a) Input neurons - receive input from the outside world

   b) Output neurons - their state is visible to the outside world

   (Actually, in some models a neuron can serve both as an input neuron and an output neuron)

   c) Hidden neurons - neither of the above

2. Each neuron has an activity level - which is represented as a real number, rather than a firing rate. (Note that this is a departure from strictly imitating the biology)

3. Each neuron has an output - also expressed as a real number - which it passes on to other neurons and/or the outside world. The output is expressed as some function of the current activity level. (In many cases, it the same as the current activity level, though in other cases it is some more complex function of the activity level.)

4. Neurons potentially receive input from two sources

    a) The outside world

    b) Other neurons

5. Connections between neurons have a strength or “weight” - typically a real number.

    a) A positive weight represents an excitatory connection.

    b) A negative weight represents an inhibitory connection.

    c) A weight of zero can be used to represent a non-existent connection.

6. The activity of each neuron is periodically updated, as follows:

    a) First, the net input to the neuron is calculated as

    external input (if it is an input neuron) * some scaling factor +

    $\sum$ weight of connection_j * output of neuron_j

    j = all neurons it receives input from

    b) Then the new activity is calculated by applying some function to the net input, and the new output is calculated from the new activity.

    As we shall see soon, this function is often <u>non</u>-linear

    c) Frequently - but not always - all neurons are updated in parallel - that is, the net input to each is calculated using the “old” activation levels, and then each neuron is set to a new activation level based on the previously-calculated net input

II.Some Capabilities of Neural Networks

A. To illustrate some of the capabilities of neural networks, we will use an example from an early (and very influential) book on neural networks by Rumelhardt and McClelland. It illustrates an approach they called “Interactive Activation and Competition”, and can be used for information retrieval

1. This network is used to hold a database describing two street gangs: he Jets and the Sharks (a takeoff on West Side Story)

   PROJECT, HANDOUT: Data table (PDP vol 1 page 27)

2. The network consists of 68 units, grouped into 7 "pools".

   PROJECT: PDP page 28

   a) One pool represents the individuals; each of the other pools represents one column in the table. (Note that the unit for the individual is distinct from the unit for the person's name; corresponding to the fact that we can recognize an individual but not be able to recall his/her name.)

      (1)Most of the neurons serve as both input and output neurons

      (2)But the neurons that correspond to individuals are hidden

   b) Each non-hidden neuron represents one of the possible values in that column. Thus, there are 27 neurons in the name pool, 2 in the gang pool, three in the age pool etc.

      Each non-hidden neuron can receive external input (corresponding to that feature being known to be present)

   c) The interconnection weights are setup to model the database, on the following basis:

(1)There is a bidirectional connection of weight 1.0 between each individual and each of the six units representing the values associated with him.

(2)There is a bidirectional connection of weight -1.0 between each neuron and each other neuron in the same pool. (This represents competition between the values - e.g. if an individual is a Jet then he is not a Shark, etc.)

d) At any point in time, each neuron has a certain degree of activation.

(1)Initially, the activation for all neurons is set to a "rest" level.

(2)Only a neuron that has a positive activation has any impact on neurons it is connected to. The output that it passes to other neurons is either equal to its activation or 0 if its activation is negative.

3. The model executes iteratively. On each cycle the activation of each neuron is updated as discussed previously.

a) The net input is calculated based on any external input plus the sum of the positive activation levels of all neurons it has a correlation with minus the sum of the positive activation levels of all neurons it is competing with. (Neurons it is connected to that have negative activation levels are ignored.)

b) The new activation of the unit is then calculated by a formula that has the following effect:

(1)If the net input to the unit is near zero, then the neuron's activation is adjusted so as to make it "drift" toward its resting level.

(2)If the net input is a significant positive value, then the neuron's activation is increased (but not above a fixed upper limit.)

(3)If the net input is a significant negative value, then the neuron's activation is decreased (but not below a fixed lower limit.)

(4)In any case, the changes made on any one cycle are small.

(5)To calculate the response to any given stimulus, many repetitions of the above cycle are used - dozens at least.

c) Over the course of these repeated iterations, the system will tend to settle into an equilibrium in which:

(1)Those units that have positive external input are quite active.

(2)Those units that are correlated with these units are also active.

(3) Other units are inactive.

B. DEMONSTRATION:

1. Look up information on Art

Show activations - note how similar individuals (individual, not name, become somewhat active.)

a) Clyde is also a Jet in his 40's with a Junior High education who is Single - just like Art, except is occupation is different

b) Ralph is also a Jet with a Junior High education who is single and a Pusher - just like Art, except that Art is in his 40's and Ralph in his 30's

Observe that we tend to do something similar - if we think about a given person, we tend to also think about other people who are similar

2. Demonstrate trying to find out name and gang of a Pusher in his 40's.

3. Demonstrate trying to find out the name and gang of a Bookie in his 20’s. Note that this time two individuals are reported

4. Now we will consider a situation in which we have a description that is actually partially incorrect.

   Try 40s, College, Burglar

   a) Note that there is no one who exactly matches

   b) Finds two close matches

      Don - 30s rather than 40s
      Earl - HS rather than College

5. Now try a Pusher in his 30’s.

   a) There are five potential answers

   b) However, the network’s answer “misses” three people. Who?

      ASK

      Ralph, Nick, Dave

   c) The two individuals who were found (Phil and Ol) have identical descriptions - in addition to being Pushers in their 30’s they are Sharks, have a College education, and are Married. What in effect happens is their being so similar causes the network to miss differing individuals who match the original query

      Ralph - a Jet, JH, Single
      Nick - a Shark, but HS and Single
      Dave - a Shark, but HS and Divorced

6. Demonstrate generalizing typical Shark

   a) Most of the Sharks are in their 30s (9 out of 12)

   b) Most of the Sharks have a HS education (7 out of 12)

   c) Half the Sharks are married

   d) But there are no Sharks in their 30s who have a HS education and are married

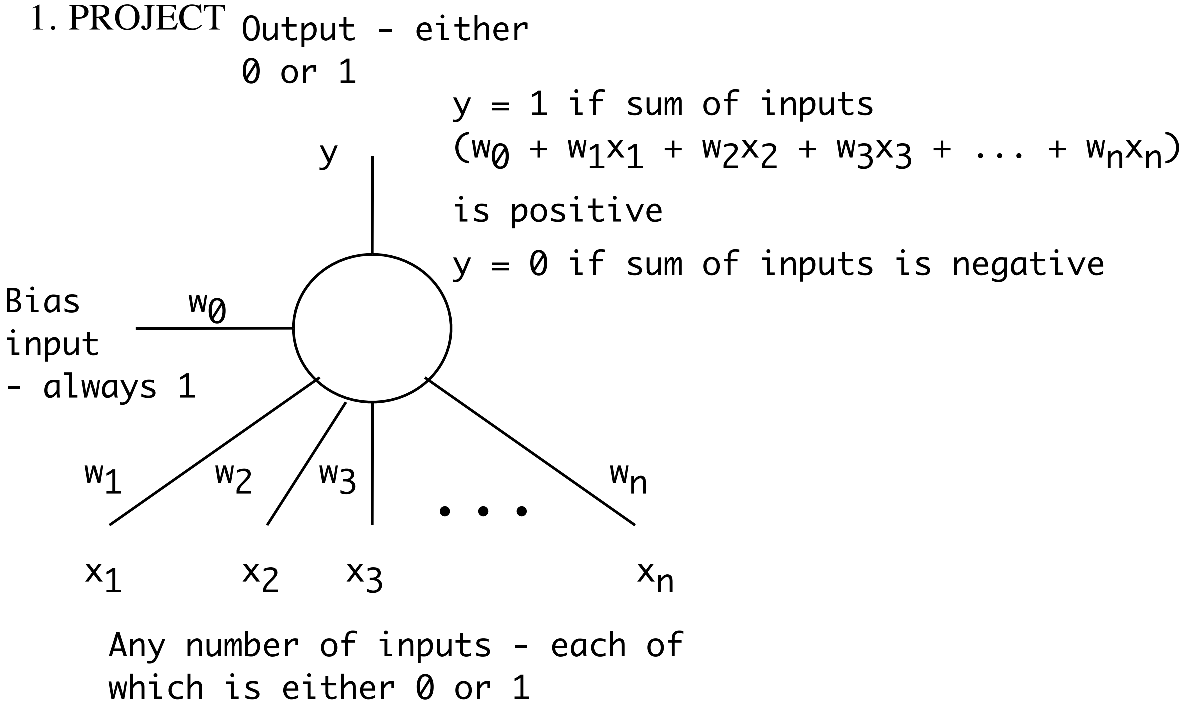
   e) The network ends up finding two “stereotypical” Sharks who match on everything but education. Do we ever do something similar with stereotypes?

III. **Perceptrons**

A. Early work with neural networks (in the 1950's and 60's) centered around a kind of artificial neurons known as “perceptrons” (sometimes built as special hardware)

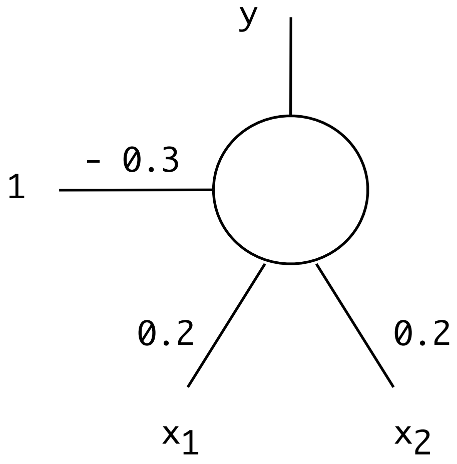
B. We will consider an extremely simple example based on this work, in which there is just one neuron - hence just one output. The model we will talk about, then, looks like this (using more modern terminology).

1. PROJECT


Output - either 0 or 1
y
y = 1 if sum of inputs
$(w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n)$
is positive
y = 0 if sum of inputs is negative
Bias input - always 1
w_0
w_1
w_2
w_3
• • •
w_n
x_1
x_2
x_3
x_n
Any number of inputs - each of which is either 0 or 1

2. Example: Consider the following instance of this pattern, having just two inputs:

PROJECT (And behavior)



Let's consider how this network behaves.

DEMO with one-node perceptron (non learning) software - set up weights as above

a) Suppose both inputs are 0 - what is the output?

ASK; DEMO

b) Now suppose one input is 0 and the other is 1. What is the output now?

ASK; DEMO both possibilities

c) Finally, suppose both inputs are 1. Now what is the output?

ASK; DEMO

d) That is, we can characterize the behavior of this network by saying its output is 1 just when both of its inputs are 1. (That is, it corresponds to an AND gate in digital logic).

3. At this point, you might be tempted to say big deal! But what is interesting about a simple network like this is that it can learn.

a) The basic idea is this: if we have a network whose behavior is not what we want, we can train it by adjusting its weights according to the following perceptron learning rules:

(1)For input patterns for which the output is what we want, we leave the weights alone.

(2)For input patterns where the output is not what we want, we adjust the weight associated with an input that is 1 for that pattern by a small multiple of the <u>error</u>. (This includes the bias weight, because the bias input is always 1)

(a)If the output is 0 when we desire 1, the error is +1, so we adjust the weight for each input that is 1 by a small positive amount.

(b)If the output is 1 when we desire 0, the error is -1, so we adjust the weight for each input that is 1 by a small negative amount.

b) Let's follow this approach to retrain this network so that it's output is 1 just when both inputs are 0 (corresponding to a NOR gate in digital logic)

Of course, it's easy to figure out by hand a set of changes that would accomplish what we want.

ASK

(1)But what we want to do is to have the network learn.

(a)DEMO - turn learning on

(b)Train to have output 1 just when both inputs are 0.

i) Discuss rationale for each weight adjustment before actually clicking “Wrong”

ii) Note that we are using a training rate of 0.5

(c)Although these are not the weights we might have found by hand, we can now see that the network has learned the desired behavior!

C. Learning of this sort becomes more interesting when the problem has more inputs and more outputs.

1. An Example

DEMO LetterLearner

a) Explain structure

(1)Input - box is treated as a 5 x 5 array of boxes - 1 if any part of the drawn line passes through it (therefore 25 inputs in all).

(2)One neuron per letter (therefore 26 in all)

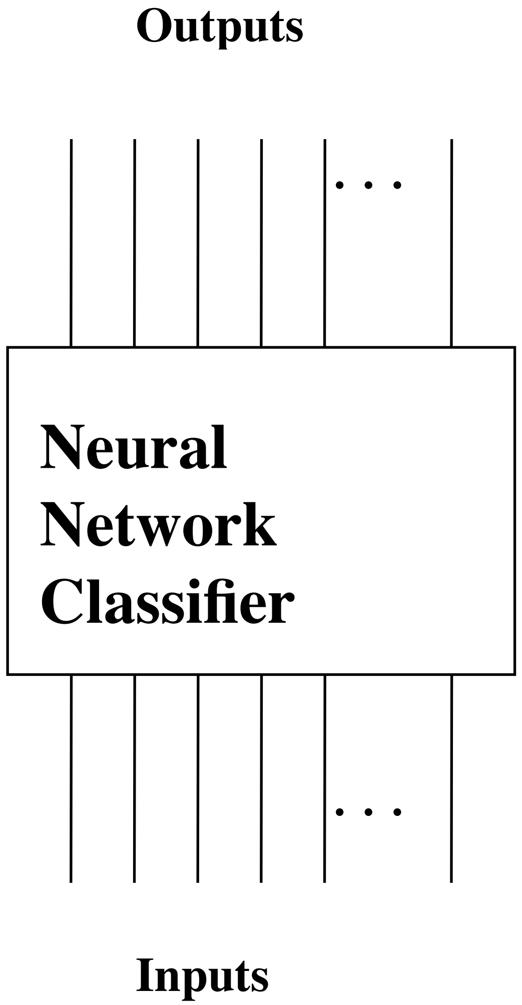
(3)A trainable weight connecting each of the 26 neurons to each of the 25 inputs (therefore 650 weights in all)

(4)When guessing, if there is one neuron that has a positive output that is significantly higher than the second best, it is chosen.

b) Train network to recognize A through D, then test with slight variants - note that further training is sometimes needed. (It should struggle with C/D due to low resolution of input)

2. The first example we did and this example are a type of neural network known as a classifier. We have already seen something similar in the symbolic paradigm. Recall the general structure of a classifier:

PROJECT



a) There are as many nodes as there outputs, with a trainable bias for each, and a trainable weight from each input to each output.

b) The output is interpreted as representing the classification of the input pattern

Example: The letter-learner program classified an input pattern representing a hand-drawn letter as a particular letter of the alphabet.

c) Classifiers are trained using supervised learning, involving a set of training data, which consists of pairs representing an input and the desired output. .

Example: For the letter-learner program, the training data consisted of patterns representing hand-drawn letters and the letter each represented.

Example: For the example in the book, the training data consisted of actual data from previous students.

d) Sometimes, the training data is divided into two parts. One part is used to actually train the network; the other is used to test the network to see whether sufficient learning has taken place. (In the case of letter learner, we could create new testing data by hand-drawing letters whose meaning we know.)

e) It is the expectation that, once the classifier has been successfully trained, it can correctly classify a new set of data based on patterns it has "learned" from the training data. (Of course, some input leading to the classification must have occurred in the training data; the network is expected to recognize similar patterns.)

Example: Letter learner's ability to recognize various hand-drawn versions of a letter like A having "seen" other hand-drawn A's. In this case, of course, we expect the classification to be correct unless the input represents a really sloppy letter! (The predictions during would not be expected to be perfect, of course!

3. Neural network classifiers are used in practice in many places.

IV.Multi-Layer networks and Backpropagation

A. Impressive as the ability to learn may seem, in the case of single-layer networks like the perceptron it is limited. It is easy to show that there are some patterns a perceptron cannot learn,because there are no weights that produce the desired result.

1. Example: Can we train a 2-input 1-output perceptron to produce the output 1 just when both inputs are different, and 0 if they are the same? (XOR gate behavior)

a) No. To see why, consider the following:

(1)For the input pattern 0, 0, we want output 0. This means that we must have $w_0 < 0$. (Since the other two weights are not involved in producing the output since the inputs are 0.)

(2)For the input pattern 0, 1, we want output 1. This means that we must have $w_0 + w_2 > 0$.

(3)For the input pattern 1, 0, we want output 1. This means that we must have $w_0 + w_1 > 0$.

(4)Finally, for the input pattern 1, 1, we want output 0. This means that we must have $w_0 + w_1 + w_2 < 0$.

b) But there is no set of possible values for the weights that can meet all these criteria. To see this:

From $w_0 + w_1 > 0$ and $w_0 + w_2 > 0$ we get (by adding them)
$w_0 + w_0 + w_1 + w_2 > 0$.
But we want $w_0 < 0$ and $w_0 + w_1 + w_2 < 0$. That means that two negative numbers would have to add to a positive sum, which cannot be!

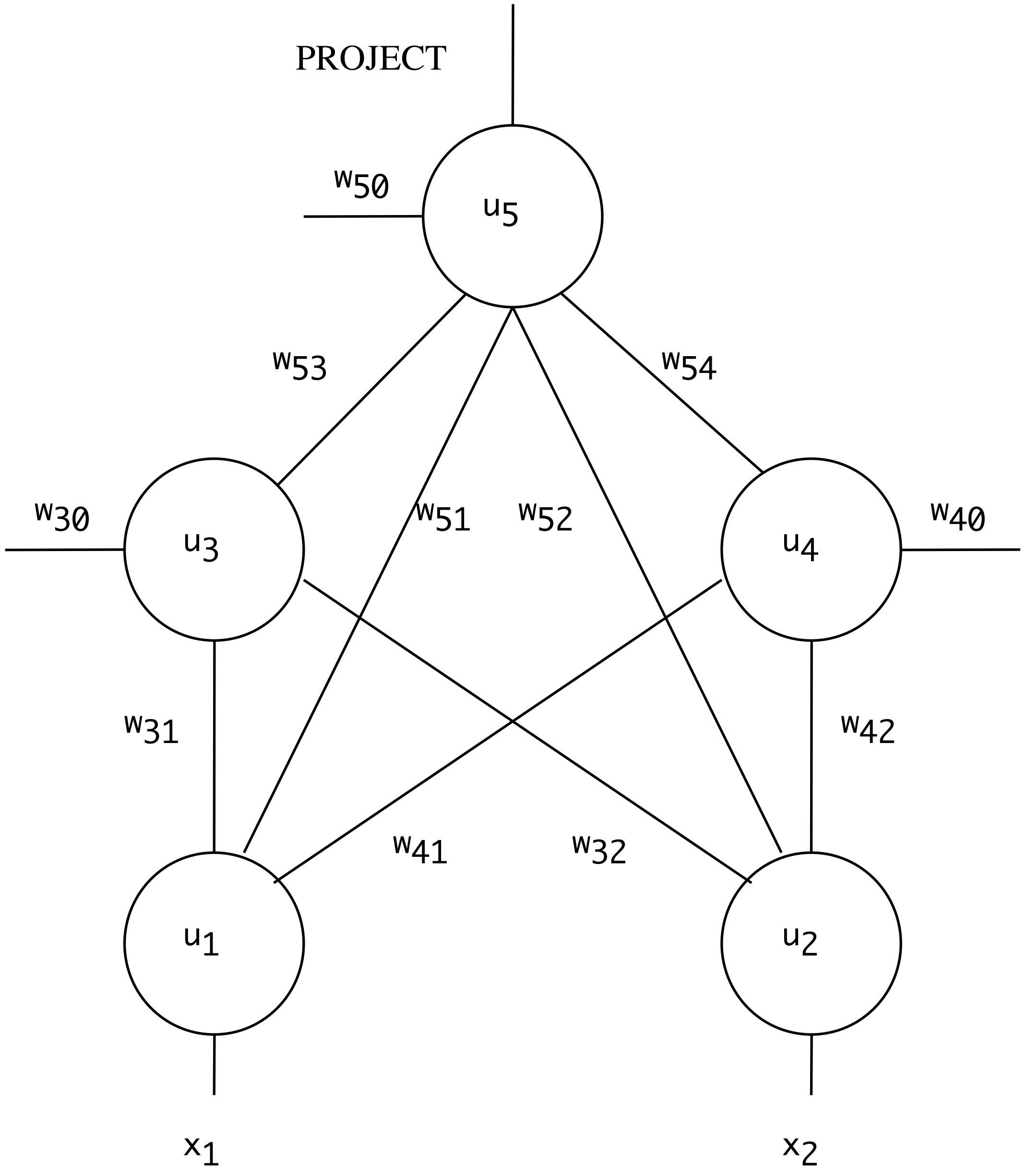
c) There are, in fact, 16 possible one output functions of two variables. Of these, 14 can be produced by a single perceptron, and 2 cannot.

2. The initial perceptron work did not go very far, and essentially died when Minsky and Papert published a book (1969) in which they proved that the capabilities of these devices were severely limited - using a problem similar to the one we just used plus some other more complex examples.

B. A revival of interest in neural networks came about the late 1980's based on the use of multilayer networks.

1. That is, a network that looks like this can produce all 16 possible functions of two inputs:

PROJECT



Output layer units - depend only on hidden and input layers

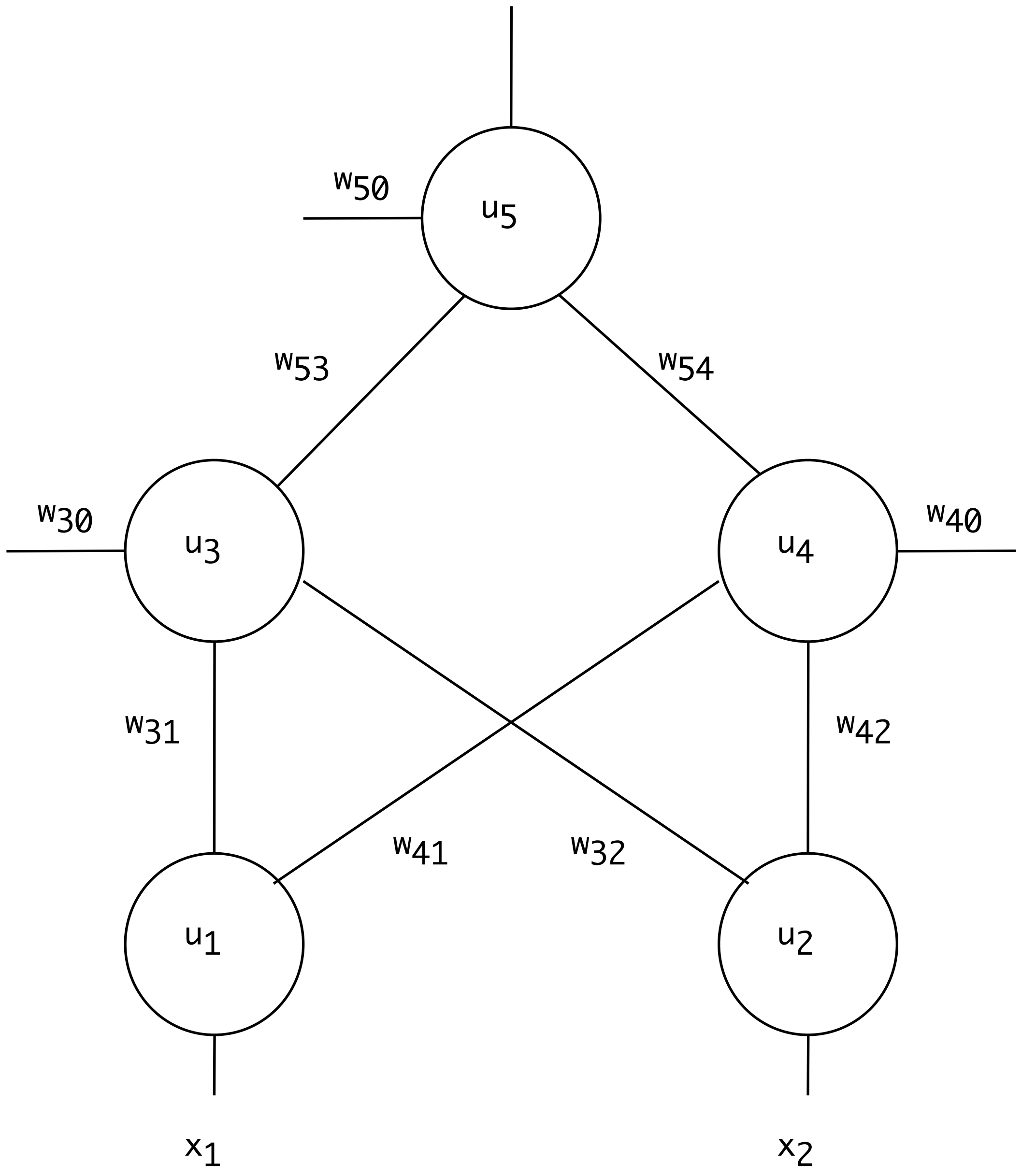
“Hidden” layer - units depend only on input
layer

Input layer -
Units “hardwired” to external
inputs

This is an example of a feed-forward network. By feed-forward, we mean that information flows one way, from the input toward the output.

2. A variant of this known as a layered network. It only allows from one layer to the layer immediately above it:

PROJECT



Output layer units - depend only on hidden layer

“Hidden” layer - units depend only on input layer

Input layer - Units “hardwired” to external inputs

3. It is also possible to create a network with more than one layer of hidden units.

C. A multilayer network can handle the "differing inputs" problem. For example, the following weights work for the layered network :

$w_{30} = -1$; $w_{31} = -2$; $w_{32} = 2$
$w_{40} = -1$; $w_{41} = 2$; $w_{42} = -2$
$w_{50} = -1$; $w_{53} = 2$; $w_{54} = 2$

1. u3 exhibits the following behavior:

x_1	x_2	weighted sum of inputs	output
0	0	-1	0
0	1	1	1
1	0	-3	0
1	1	-1	0

2. u4 exhibits the following behavior:

x_1	x_2	weighted sum of inputs	output
0	0	-1	0
0	1	-3	0
1	0	1	1
1	1	-1	0

3. u5 exhibits the following behavior:

x_1	x_2	u_3	u_4	weighted sum of inputs	output
0	0	0	0	-1	0
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	-1	0

4. Actually, there are <u>many</u> weight patterns that will also work

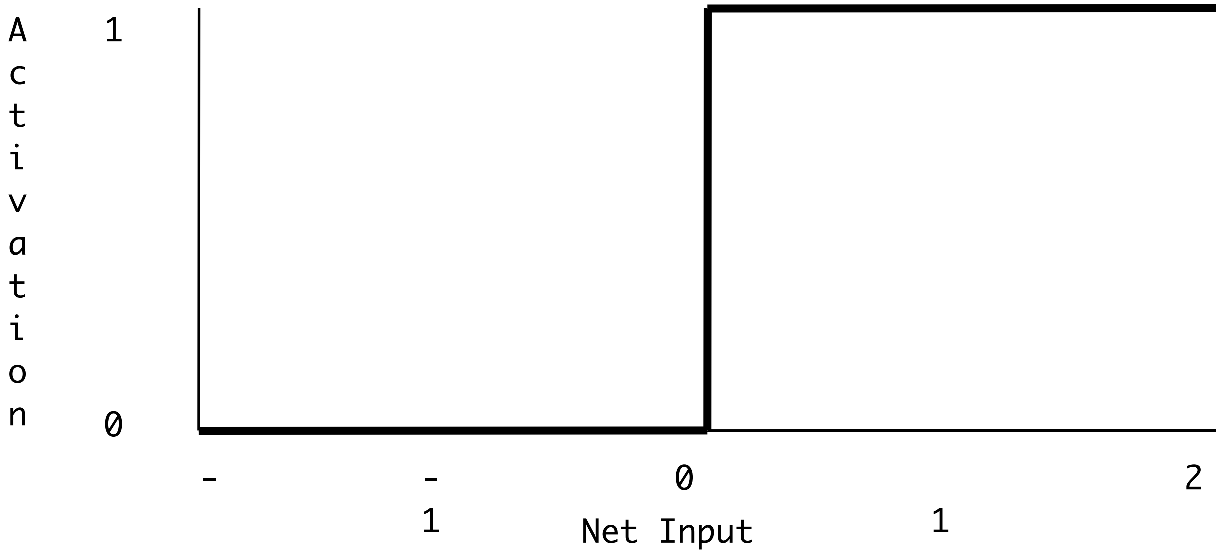
D. A multilayer network like this can not only realize any function of any number of inputs, but also if a feed-forward layered network is used and the “transfer function” (way we calculate activation from the net input) for the node is appropriate it can <u>learn</u> any such function using an algorithm discovered in the 1980's called back-propagation.

1. If we call the weighted sum of inputs into the neuron s, then the output is calculated as output = f(s), where f is some function.

2. The transfer function we have been using thus far is a step function: $f(s) = 1$ if $s .> 0$ or 0 if $s < 0$

   PROJECT

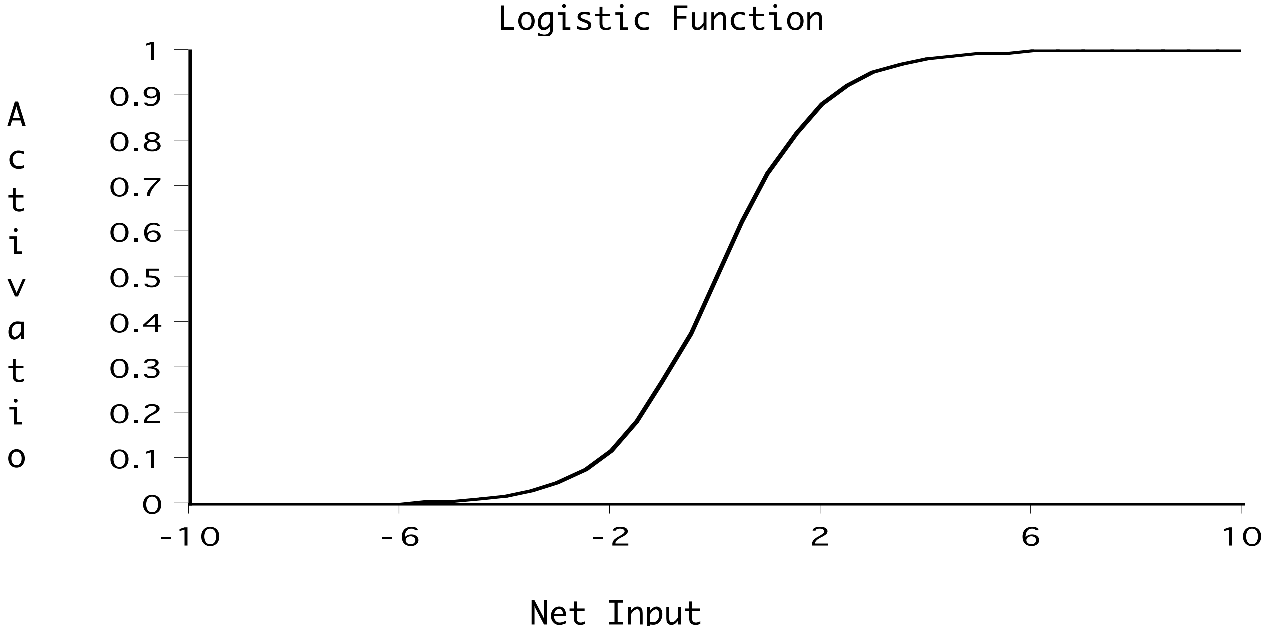
Step Function



3. One that is often used is the logistic function: $f(s) = 1 / (1 + e^{-s})$

   For negative s, this is close to 0, and for large positive s, it is close to 1, so it looks like a smoothed out step function.

   PROJECT



Its derivative has a particularly simple form:

f'(s) = f(s)(1 - f(s))

E. The full derivation of the back-propagation function can be done in an hour or so, but we can quickly give the basic idea.

1. For any given set of inputs, let the error E be the difference between the desired output and the actual output of the network we are training - e.g. if the network outputs 0 when it should output 1, the error is 1. (If there are multiple outputs, then E is a vector and we have to carry out this process for each element.)

2. For each weight in the network, we calculate the partial derivative of the error relative to that weight - e.g. for each weight w_{ij} , we calculate $\partial E / \partial w_{ij}$. Since we want to reduce the error, we adjust the weight by some fraction of this partial derivative times the error - e.g. if the partial derivative is positive and the error is positive, then we want to increase the weight.

   The calculation of the partial derivatives uses the derivative of the transfer function - hence the requirement that the transfer function be differentiable and the desirability of using a transfer function whose derivative is easily calculated.

3. We refer to the fraction used in adjustment as the training rate (often called α). For any given training problem, this value needs to be set so as to achieve a good balance between overshooting the correct value (which happens if α is too large) and taking too long to learn (if it is too small).

4. It is also possible to use a momentum term (β) to further speed learning.

F. DEMO: xornet

1. Train with alpha = 0.5 beta = 1 (Note: training is much quicker with high beta, but will sometimes fail!)

2. Analyze resulting weights

3. Train again

4. Note the large number of cycles needed to train (hence reason we didn't step through in class!)

   However, once a network has been trained, the weight values found can be replicated any number of times - i.e. training needs to be done just once for any given network architecture and set of training data!

G. Many neural network used in practice are multi-layered networks that learn using backpropagation. Applications include:

1. Classifiers.

2. Predictors

3. Controllers

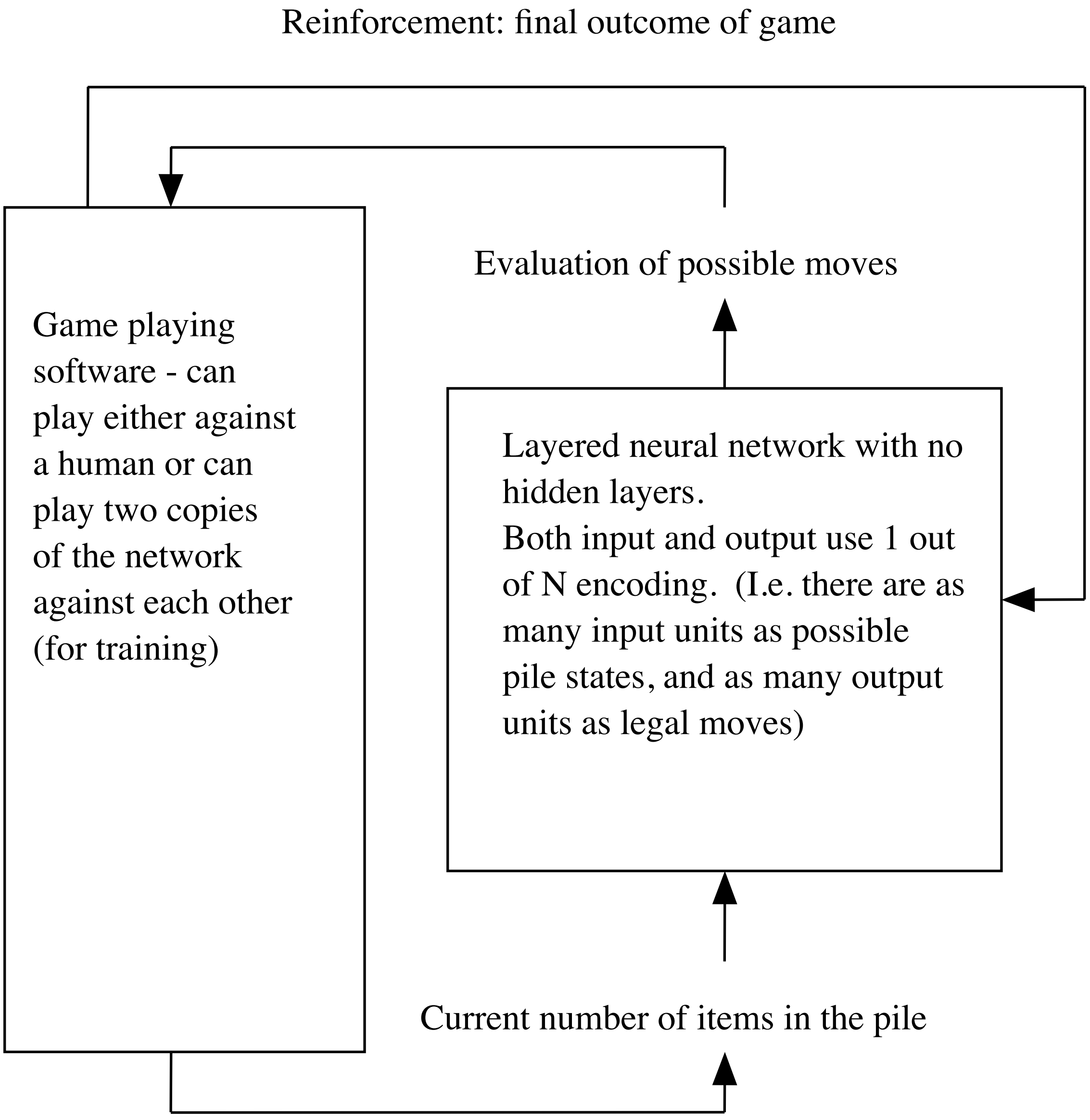
H. Improvements in computational speed have made it practical to train networks with multiple layers - what we call deep networks. (More on this later)

V. **Reinforcement Learning**

A. When we discussed genetic algorithms, we looked at an example of how a genetic algorithm could learn to play the game of NIM. Now we will consider how a neural network can do this.

B. The basic architecture we will use is this

PROJECT


Reinforcement: final outcome of game
Evaluation of possible moves
Game playing software - can play either against a human or can play two copies of the network against each other (for training)
Layered neural network with no hidden layers.
Both input and output use 1 out of N encoding. (I.e. there are as many input units as possible pile states, and as many output units as legal moves)
Current number of items in the pile

1. The network uses 1 out of N encoding for both input and output.

    a) There is one input unit corresponding to each possible number of items in the pile. (E.g. if the maximum size of the pile is 64, there will be 64 input units.) Exactly one of these will be 1, while the remainder will be 0.

    b) There is one output unit corresponding to each possible move. The output unit having the highest activation will have output 1, and represents the move the network will choose, while the remainder will have output 0. (That is, the function for calculating output as a function of activation yields 1 for the highest value and 0 for the others.)

2. Activation is interpreted as "the likelihood of success if this move is chosen", and the move which the network classifies as having the highest likelihood of success is chosen.

3. The aim of learning is for the network to learn to assign the correct likelihood of success to each possible move for each possible pile state.

    Example: If there are 5 items in the pile, and the permissible moves are 1, 2, and 3, we would like the network to assign a likelihood close to 1 to "take 1", and a very small value to "take 2" and to "take 3". (But actually any value for 1 that is greater than the values for 2 and 3 would work)

C. How can we train such a network?

1. In this case, we could use supervised learning, since a knowledgable human can always determine the correct move using the NIM algorithm. But this is not generally the case with interesting games, of course - so we'll pretend we don't know the algorithm.

2. Instead, we need to use reinforcement learning - so the only feedback we can use for training is the ultimate outcome of the game.

3. In the case of the final move, it is possible to train the network as follows:

   a) If the network won the game, then train the final move toward likelihood 1.

   b) If the network lost the game, then train the final move toward likelihood 0.

4. But how do we train the network for the other moves in the game?

   In general, if the network is playing a knowledgeable opponent and loses, it might be a consequence of a bad move it made at any point in the game.

   a) Many of the moves it made might actually have been good - we don't want to train the network to reject them.

   b) But we do want to train the network to reject bad moves.

   c) How can we tell the difference?

      This problem is known as the "credit assignment problem" (or perhaps in this case it should be called "blame assignment".

5. The approach we will take is this.

   a) During the course of the game will keep a record of the following for each move the network made

      (1)The number of items in the pile

      (2)What move the network chose, and the network's evaluation for it

b) At the end of the game

(1)We will train the last move chosen as discussed for above

(2)For each move the network made (except the last one), we will train the network toward making the estimated likelihood of success it gave toward the estimate on the next move (which, being closer to the end of the game, is presumed to be more accurate)

6. Example - using actual data

a) Log from an actual game played after 500 games of training (in which I, as the human, deliberately "threw" the game early, and began moving more or less randomly in the end game when the program obviously "knew what it was doing"

```
PROJECT

Pile contains 35 - You take 1
Pile contains 34 - I take 3 (confidence level 0.50)
Pile contains 31 - You take 2
Pile contains 29 - I take 1 (confidence level 0.52)
Pile contains 28 - You take 1
Pile contains 27 - I take 3 (confidence level 0.50)
Pile contains 24 - You take 2
Pile contains 22 - I take 2 (confidence level 0.55)
Pile contains 20 - You take 3
Pile contains 17 - I take 1 (confidence level 0.66)
Pile contains 16 - You take 1
Pile contains 15 - I take 3 (confidence level 0.74)
Pile contains 12 - You take 2
Pile contains 10 - I take 2 (confidence level 0.76)
Pile contains 8 - You take 3
Pile contains 5 - I take 1 (confidence level 0.82)
Pile contains 4 - You take 1
Pile contains 3 - I take 3 (confidence level 0.91) - I win!
```

(1)At the end of the game,

Take 3 with pile 3 was 0.91 would be trained toward 1
Take 1 with pile 5 was 0.82 would be trained toward 0.91
Take 2 with pile 10 was 0.76 would be trained toward 0.85
Take 3 with pile 15 was 0.74 would be trained toward 0.76
Take 1 with pile 17 was 0.66 would be trained toward 0.74
Take 2 with pile 22 was 0.55 would be trained toward 0.66
Take 3 with pile 27 was 0.50 would be trained toward 0.55
Take 1 with pile 29 was 0.52 would be trained toward 0.50
Take 3 with pile 34 was 0.50 would be trained toward 0.52

(2)Note how the trainings corresponding to the end game serve to strengthen behavior that is already good

(3)Note how the trainings corresponding to the two earliest moves actually train the program in the wrong direction!. But as learning progresses, even moves near the start of the game will be trained properly.

7. Of course, for training purposes, we want the network to "play" against a simulated opponent, not a live human. (Since successful training takes <u>many</u> games). What sort of opponent should we use?

   a) One possibility would be to have the network play against a simulated opponent who plays the game perfectly, using the NIM algorithm.

      However, this turns out to require many training games because, when the network is first learning, it plays so poorly that it seldom sees unsafe pile states, and thus it cannot learn the correct move to make in such a state because it doesn't see it.

   b) Another possibility is to have the network play against a simulated opponent who plays randomly some or all of the time.

      However, this turns out not to work, since the network can often get away with making bad moves, so it doesn't really learn what moves are bad.

c) A third possibility is to have the network play against a simulated opponent who makes the “right” move some percentage of the time, and a random move the rest of the time. While this can work, it presumes (as does the first case) a knowledge of what the “right” move is, so it’s not useful for interesting games.

d) The approach used by the program about to be demonstrated is to have the network play against another copy of itself.

   That is, the same set of weights is used for both players, but is only trained based on the experience of one of the players.

   (1)When first starting out, the network does play randomly.

   (2)However,. as the network learns, the “opponent” plays better and better - but this is not a problem since the “trainee” makes the “right” move many of the times.

8. Demonstration

a) Play against initial opponent - deliberately make bad moves

b) Train 1000 games, then play again - intentionally making a wrong move at the start if necessary to give the network a chance.

VI. **Auto-Associative Networks**

A. The network structure we have been using (feed-forward layered network) is the one most frequently used in machine learning. But there are other kinds of networks that can be used for other purposes.

B. Another interesting type of neural network is the auto-associator. An auto-associator can be thought of as a pattern classifier where each pattern is associated with itself (i.e. for training the input and output are the same)

1. Although auto-associative networks are considered a form of supervised learning, they are not trained in the same way backpropagation networks are trained.

2. Instead, the weights in an auto-associative network are set once and for all at program startup based on the patterns to be used.

3. The “Jets and Sharks” example we did at the start of this series of lectures is a form of auto-associator.

C. We will now look at another form of auto-associator: the Hopfield network.

1. Demo ShapeAssociator

a) Show stored patterns (select each in turn using Set To Training Pattern in left window)

b) Demonstrate how it can reconstruct a junked pattern
(Use load input in File menu to select one of the junked patterns, then push Set Input, then cycle)

c) Demonstrate how it can reconstruct a partial pattern
(Use load input in File menu to select one of the partial patterns, then push Set Input, then cycle)

d) Demonstrate how it sometimes can reconstruct a pattern from even a single dot! (Use load input in File menu to select single dot, then push Set Input, then cycle)

e) Demonstrate how it sometimes “creates” a pattern. (Use load input in File menu to select totally random, then push Set Input, then cycle)

2. How does it work?

a) One neuron per element in the pattern - in this case 100 since a pattern is 10 x 10

b) Each neuron is connected to each other neuron, but not to itself. Hence, in this case there are 9900 connections, each with a weight. (But connections are considered bidirectional, so if the weight from neuron i to neuron j is some value, then the weight from neuron j to neuron i will have the same value.)

c) A pattern to be stored or tested is represented as a vector of bipolar values - +1 if the pixel is black and -1 if it is white.

d) When the program starts up, it stores a collection of training patterns in the network. As each training pattern is stored

(1)The weight between two neurons corresponding to picture elements that are the same (both white or both black) is increased.

(2)The weight between two neurons corresponding to picture elements that are different (one white and one black) is decreased.

(3)The overall value of each weight is the sum of contributions from each pattern.

(a) Example: Suppose we have a 4 neuron network (hence 12 weights). Suppose we want to store the following three patterns

+1 -1 -1 +1

+1 -1 +1 -1

-1 +1 +1 -1

(b) The weights are set as follows

Pattern	w_{12} w_{21}	w_{13} w_{31}	w_{14} w_{41}	w_{23} w_{32}	w_{24} w_{42}	w_{34} w_{43}
+ - - +	-	-	+	+	-	-
+ - + -	-	+	-	-	+	-
- + + -	-	-	+	+	-	-
overall	-3	-1	+1	+1	-1	-3

(4) Once the weights are set, they do not change

e) When a test pattern is presented, the activation of each neuron is initially set to the value in the pattern (+1 or -1).

(1) On subsequent cycles, a new activity is calculated for a neuron as a weighted sum of the activities of the other neurons. If this sum is > 0, the neuron's activity is set to +1; if it is < 0, it is set to -1, and if it is 0, it is left alone.

(2) The order in which neurons are updated is random

Example: suppose we are given the test pattern - - - -:

Initial neural activities: - - - -

Update (say) neuron 2

Weighted sum of activities is w_{21}*(-1) + w_{23}*(-1) + w_{24}*(-1) = -3*(-1) + 1*(-1) + -1*(-1) = 3 - so neuron 2 changes to +1

New activities: - + - -

Update (say) neuron 3
Weighted sum of activities is w_{31}*(-1) + w_{32}*(+1) + w_{34}*(-1) = -1*(-1) + 1*(+1) + -3*(-1) = 5 - so neuron 3 changes to +1

New activities: - + + -

This is a stable pattern (in fact, one of our training patterns)

(3) Of course, normally a network won't settle into a stable state quite so quickly! In our example, each press of the cycle button updated 100 neurons, and it usually took several presses of the button to establish a stable state.

VII. Deep Learning

A. While Deep Learning methods are not limited to use with neural networks, the term is often taken as implying the use of a neural network. We will focus our discussion here to this sort of deep learning.

B. One writer describes deep learning this way: "Deep learning is a particular kind of machine learning that achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts." (Ian Goodfellow, Yoshua Bengio, and Aaron Courville *Deep Learning*. MIT Press, 2016 p. 9.) The writers illustrate this by the following figure:

PROJECT 1.2 from above

C. In traditional neural network architectures, the features of the problem that are to be considered are decided on by the designer, and then the network learns the weights associated with various features. In deep learning, the representation itself is learned.

Moreover, the network generally uses a layered architecture, in which the representation that is learned at each layer depends on the features learned at the layer below.

The authors of the book I just cited illustrate this this way:

PROJECT Figure 1.5

D. Such an architecture may have myriads of trainable weights, which could make learning take an infeasible amount of time.

1. Back-propagation - the standard algorithm for training layered neural networks - is based on considering $\partial E/\partial w$ for each weight w. But the partial derivative takes into account not only the impact of the weight on the output of the neuron, but also the impact of this on the overall output (often passing through several layers.) The computational effort grows exponentially with the number of layers.

2. Increasingly-fast computational platforms (e.g. highly parallel systems based on GPUs) and other methods must be used to make the computational cost of learning feasible.

3. Of course, once a network has learned a set of weights for a given problem, one can quickly copy the learned set of weights into a new network - so that once one network has learned a solution to a given problem, it is inexpensive to create many more that "know" the same solution.

E. Deep learning has numerous practical applications in systems such as those that recognize images or speech, self-driving cars, or systems such as Watson or Alpha-Go.