



CogSci 109 lecture 16

*Introduction to neural networks
and applications II*



Outline for today...

- Announcements
- Yet more homework hints
- Review of last time
- An Application
- Artificial Neural Networks introduction
 - **Mainly qualitative, more quantitative**
- Examples



Announcements

- Homework 4 HW4 extension to Friday either in TA boxes or in office hours
- Returning of hw2, hw3, midterm
 - **All today!!!**
- Email list add - email with the subject, if you want to be on the list:
 - **COGSCI109 LIST ADD**



Outline for today...

- Announcements
- ***Yet more homework hints***
- Review of last time
- An application
- Artificial Neural Networks introduction
 - **Mainly qualitative, more quantitative**
- Examples



Homework hints...

- First a few notes
 - **About fminsearch**
 - **About the final question**
- Q&A (~5min-10min max)



Outline for today...

- Announcements
- Homework hints
- **Review of last time**
- *An application*
- *Artificial Neural Networks introduction*
 - **Mainly qualitative, more quantitative**
- Examples



But is there another way?

- As cognitive scientists you might want to create a fit to very nonlinear difficult data, and the methods we have used may have difficulty
- You may want to model cognition and performance of large groups of structure in the brain rather than just behavior
 - **Gosh I wish there was a *model* for these sorts of concepts...**

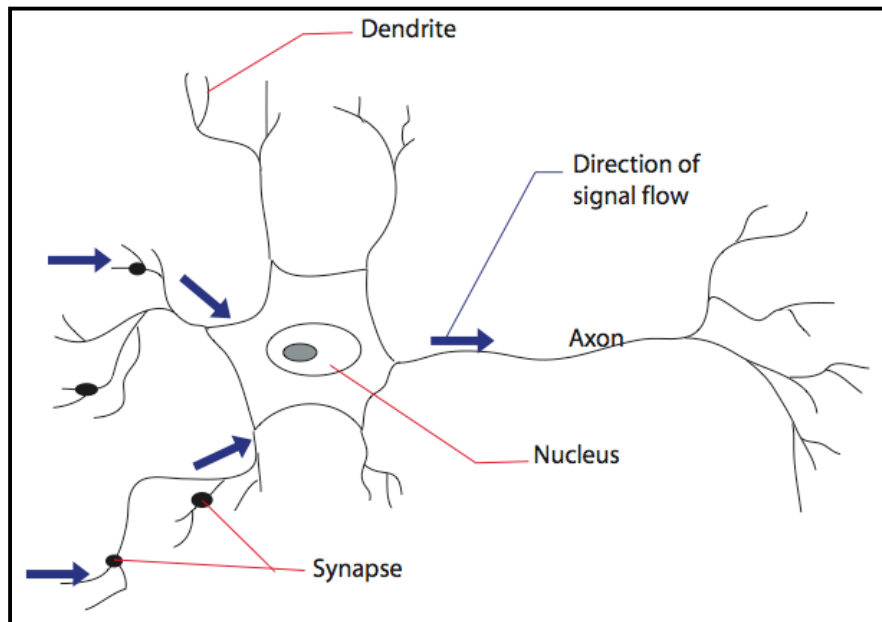


There is! Artificial Neural Networks (A.N.N.'s)

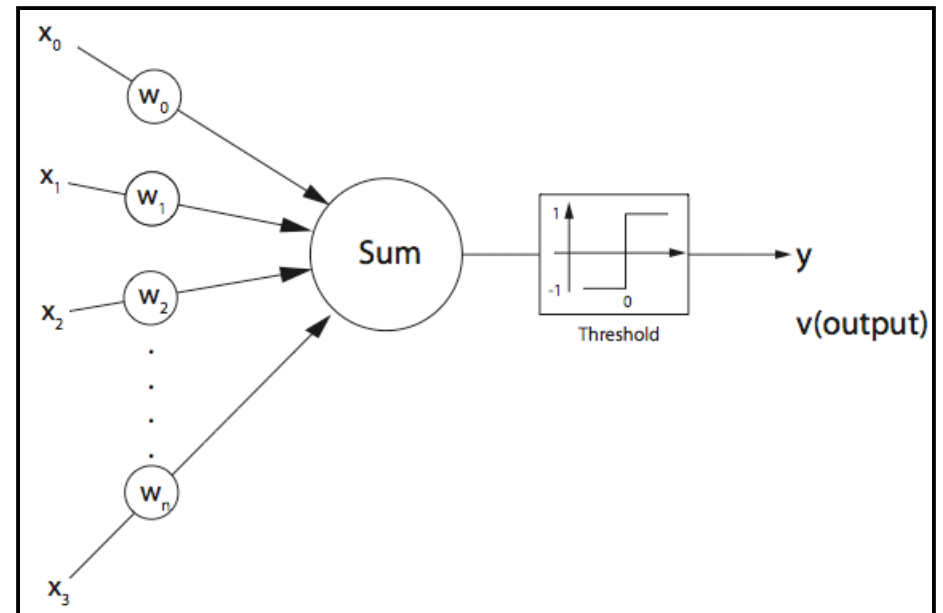
- A study of nature leads to a useful model
 - **Something about the organization of the structures of the brain allows us to solve complex problems with ease, adapt to new situations, and deal with large errors, incomplete information and faults (brain injury)**
 - **Artificial Neural Networks are an attempt to simulate by mathematical means an idealized representation of the basic elements of the brain and their**
 - Functionality
 - Interconnections
 - Signal processing
 - Self-organization capabilities

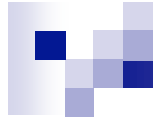
Brief review of neuronal structures and relation to ANNs

A simplified biological neuron



Classic threshold logic unit





One neuron alone is not where the true power lies

- Electrical impulses travel along the axons and are transmitted to other neurons via synaptic connections
- If enough incoming pulses arrive in a particular neuron in a given amount of time, the neuron fires, transmitting a new electrical impulse down its axon
- This is a fairly slow process (relative to computer architecture) for a single neuron, but...



Why is a neural structure so powerful?

- Massively parallel
 - **Parallel vs. serial demo**
- Very fault tolerant
 - **When you for example are writing a program and miss a \cdot^{\wedge} or misspell a variable, that is a fault, brain is less sensitive to that kind of thing since many neurons contribute to the same computation**
- Low power consumption
 - **Brain consumes *orders of magnitude less energy* than any known digital technology for similar elementary operations (logic, for example)**
- 10^{11} neurons, and $\sim 10^{15}$ connections
 - ***Plasticity of the brain* - adaptation of connectivity patterns which allows us to learn**
 - **Compare 10^3 - 10^5 connections of each neuron to others with ~ 10 for a digital logic circuit**
- Highly interconnected nature



The double-edged sword of A.N.N.'s

- A.N.N.'s solve problems in very different ways from usual computer programming
 - **No series of precise instructions (program) for the machine to execute**
 - **ANN is more adaptive, self-organizing progressively to approximate the solution**
 - Frees the problem solver from having to specify the steps to a solution
 - Also *hides* the steps to the solution, so you may not learn how a problem is being solved by a person in an experiment for example, you can just model it in a way that predicts the answer
 - **Example - two volunteers, sentence comprehension**
 - **So take-home message - as always with modeling, use the ANN model with care, consider the application and you are likely to gain many useful insights using them**



A.N.N.'s are best at...

- ANN's are best at problems where little or nothing is known, so building a mathematical model is difficult, but there happens to be a great deal of data is available
 - **A.N.N.'s are data-driven**
 - **Can handle systems with local nonlinearities, ie as time goes to infinity, the system does NOT go to infinity, but rather zero or something else - which is most physical systems**
- Some common applications of this type are
 - **pattern classification**
 - **non-linear function approximation and system modeling**
 - **Control**
 - **associative memory**
 - **system prediction**



The basics of Artificial Neurons

- ANN's are made of up many repetitions of the same simple structure, artificial neurons
- 1943, McCulloch and Pitts wrote a very influential paper (which you will read) and introduced:
 - **The Threshold Logic Unit (TLU) also known as a Linear Threshold Gate**
 - Takes real-valued inputs (e.g. 0.243 as opposed to 1 or 0 only), x_i , each input associated with a “weight” w_i (or “synaptic weight”), which represents the contact between two nerve cells
 - Performs a weighted sum of the x 's, and if the sum is larger than a threshold (θ), the neuron outputs a 1, otherwise a 0
 - The neuron will ‘fire’ if the threshold is exceeded, otherwise it does nothing



Artificial Neuron Firing...

- **Neuron Activation** is defined by the weighted sum of

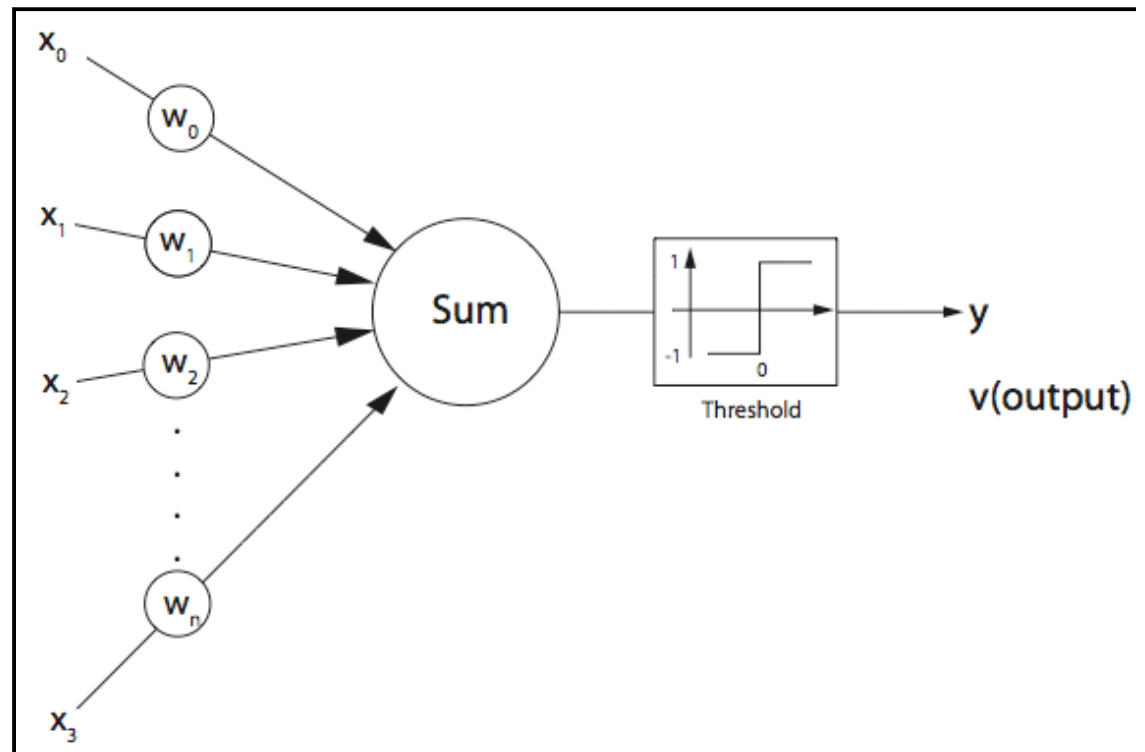
$$Activation = \sum_{i=1}^n w_i x_i = w^T x$$

- And whether the neuron fires is determined by

$$y(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^n w_i x_i \geq \theta, \\ 0 & \text{otherwise} \end{cases}$$

Perceptrons are more general than TLU's *(may contain more than one TLU)*

- So how is this useful?
 - Since it can output a 0 or 1, a single unit perceptron alone can perform many logical operations
 - Demos
 - AND, OR, NOT
 - Combined with more than one TLU, you can have continuous functions, since output of one can be weighted input to another





How does it 'learn?'

- There are multiple learning rules that can be used
- The idea is that the perceptron is 'trained' by beginning with a guess for the weights, giving it an input, it generates an output (0 or 1), then that is compared with the desired output, and the weights are updated according to some rule
 - **i.e. - if it was wrong, change the weights so next time it will be 'less wrong'**
- After the training period, it should respond to certain inputs with reasonable outputs
- Guess what is a popular algorithm for updating the weights?
 - **Yep, gradient descent - usually modified to be conjugate gradient to help with convergence**



Three approaches

1) Supervised learning

- **Net presented with known input/output pattern pairs (training set)**
- **Learning process consists of updating weights at each step so for a given input, error measure betw. Net's output and known target is reduced**
- **AKA associative learning**



Three approaches

2) Unsupervised learning

- **Still and input/output relationship that the Net must learn to reproduce but no feedback as to whether the net's output is correct or not**
- **Similarities in patterns and features of the set must be discovered by the net in a *self-organizing manner***



Three approaches

3) Reinforcement learning

- **The ‘teacher’ signal from the input/output pair is not a measure of error, but an evaluation of the result as ‘wrong’ or ‘right’ direction**
- **The net at each step of the training period updates to reduce the ‘wrong’ responses**



Supervised learning: Perceptron learning rule (Rosenblatt 1957)

1. Initialize weights and threshold randomly
2. Present an input vector to the neuron
3. Evaluate the output of the neuron
4. Evaluate the error of the neuron and update the weights according to :

$$w_i^{t+1} = w_i^t + \eta(d - y)x_i$$

- 1. Where d is the desired output, y is the actual output of the neuron, and $\eta(0 < \eta < 1)$ is a parameter called the step size**
5. Go to step 2 for a certain number of iterations or until the error is less than a pre-specified value

■ Computing "and":

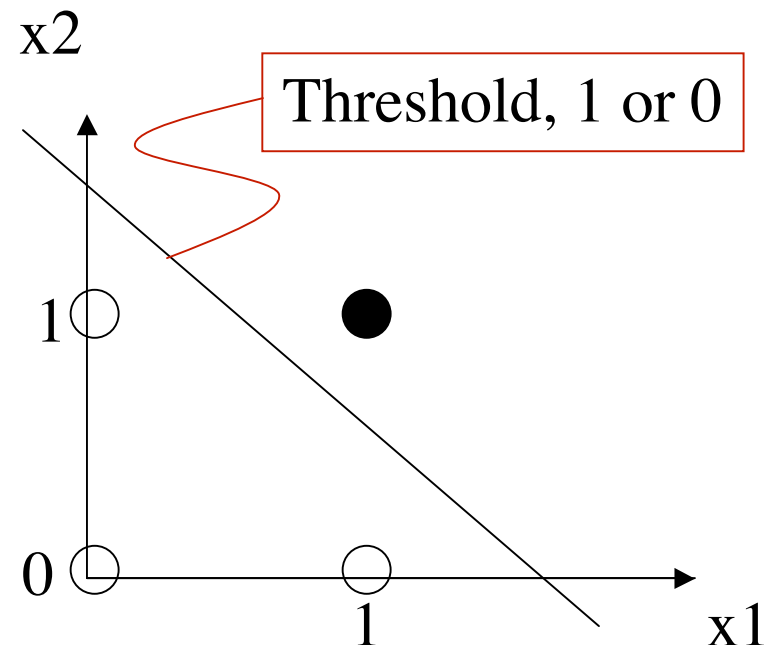
- 'And' review
- There are n inputs, each either a 0 or 1. To compute the logical "and" of these n inputs, the output should be 1 if and only if all the inputs are 1. This can easily be achieved by setting the threshold of the perceptron to n . The weights of all edges are 1. The net input can be n only if all the inputs are active.
- For every input we could compute the output, or we could realize that the activation function is

$$\text{Activation} = \sum_{i=1}^n w_i x_i \geq \theta$$

$\theta = \text{threshold}$

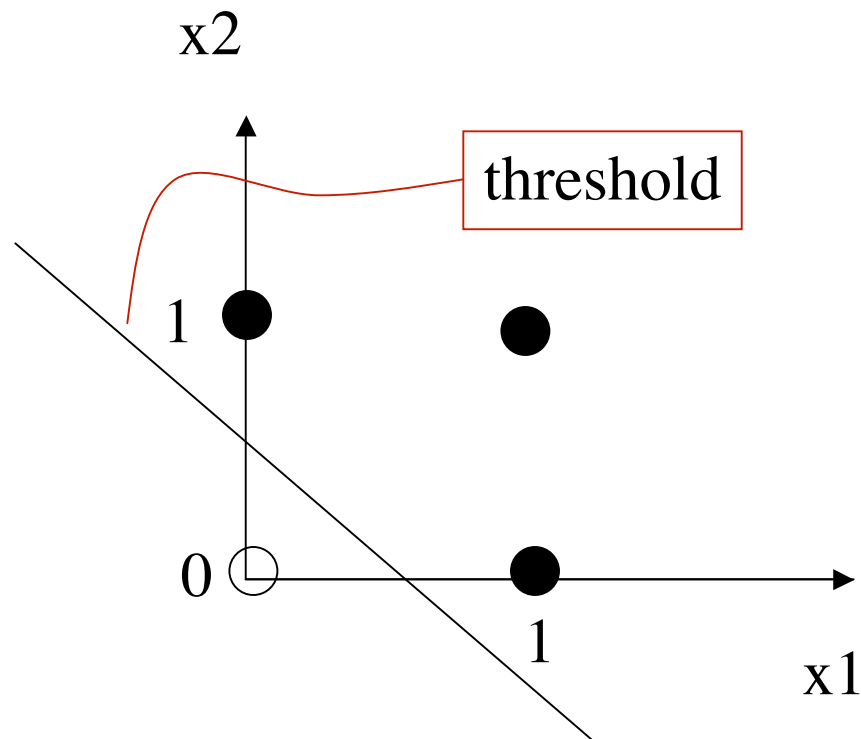
And thus we can define a threshold line:

$$\begin{aligned} w_1 x_1 + w_2 x_2 &= \theta \\ x_2 &= \theta / w_2 - w_1 x_1 / w_2 \\ w_1 &= \text{const} \\ w_2 &= \text{const} \end{aligned}$$



■ Computing "or":

- 'Or' review
- It is also simple to see that if the threshold is set to 1, then the output will be 1 if at least one input is active. The perceptron in this case acts as the logical "or".





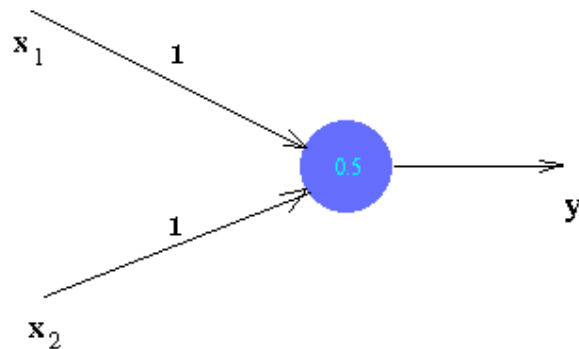
■ Computing "not":

- 'Not' review
- The logical "not" is a little tricky, but can be done. In this case, there is only one boolean input. Let the weight of the edge be -1 , so that the input which is either 0 or 1 becomes 0 or -1 . Set the threshold to 0. If the input is 0, the threshold is reached and the output is 1. If the input is -1 , the threshold is not reached and the output is 0.

Limitations of a single neuron

■ XOR problem -

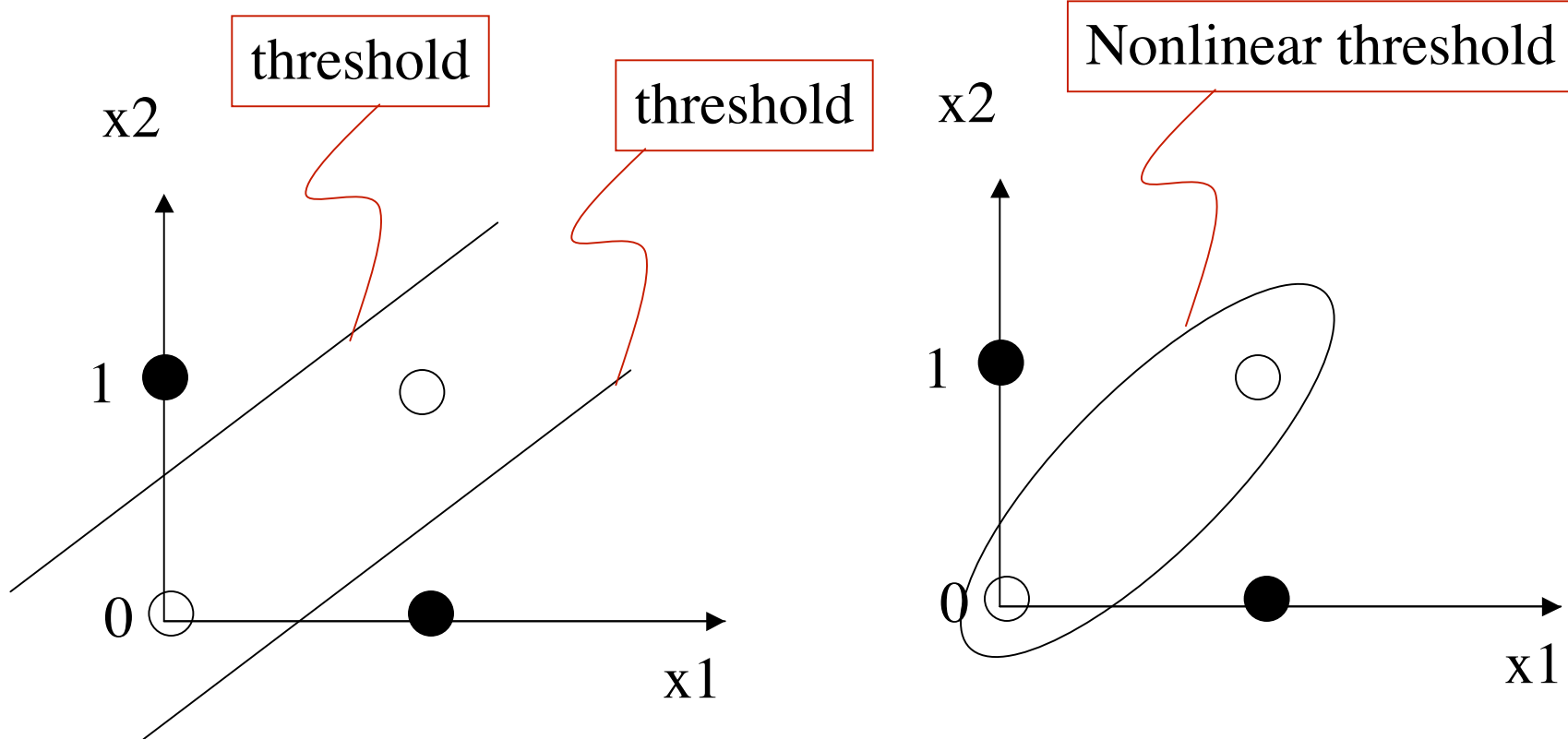
- build a perceptron which takes 2 boolean inputs and outputs the XOR of them. What we want is a perceptron which will output 1 if the two inputs are different and 0 otherwise.
- Consider the following perceptron as an attempt to solve the problem



Input	Input	Desired Output
0	0	0
0	1	1
1	0	1
1	1	0

- If the inputs are both 0, then net input is 0 which is less than the threshold (0.5). So the output is 0 - desired output.
- If one of the inputs is 0 and the other is 1, then the net input is 1. This is above threshold, and so the output 1 is obtained.
- But the given perceptron fails for the last case, and you can't make it give the desired output for that case, given this single perceptron

What would the input space look like with XOR and 2 inputs?



The separation can either be two linear separations or one nonlinear separation



Never fear, we can make more!

- That's why combining more than one makes neural networks more general for solving problems
 - **The concept of basis functions**

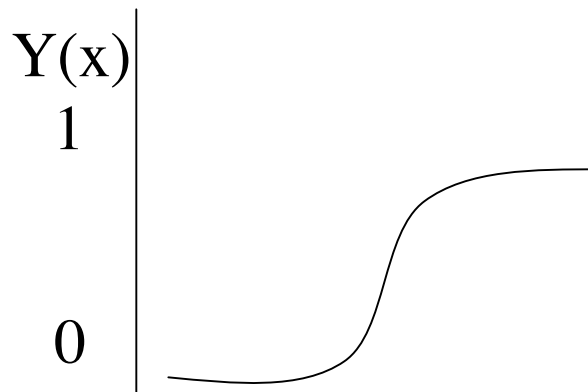
Using ANN's to create continuous outputs

- The activation function does not have to be only a threshold, it can also be a continuous function between 0 and 1, -1 and 1, etc

$$y(x) = g\left(\sum_{i=1}^n w_i x_i\right)$$

- A popular function is the SIGMOID FUNCTION

$$y(x) = \frac{1}{1 + e^{-(bx+c)}}$$





ANN examples

- <http://diwww.epfl.ch/mantra/tutorial/english/perceptron/html/>