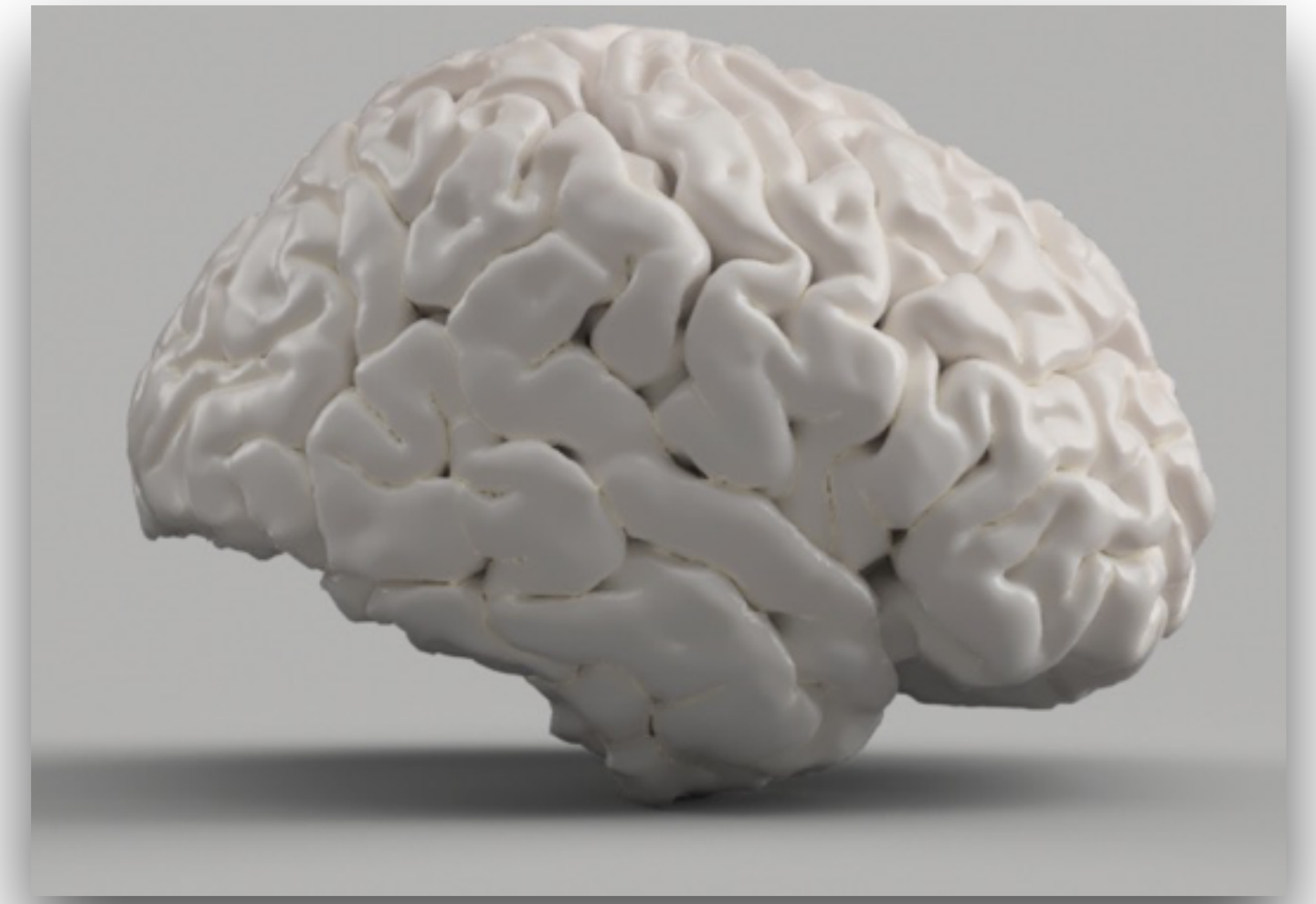
COGS109: Lecture 14



Perceptrons, Threshold logic unit, Artificial Neural Networks

July 27, 2023

Modeling and Data Analysis

Summer Session 1, 2023

C. Alex Simpkins Jr., Ph.D.

RDPRobotics LLC | Dept. of CogSci, UCSD

Announcements

- New cape replacements logistics
- Assignments remaining A2, A3, D5, D6, D7, Q3, Q4, project
- Project checkpoint 2 EDA
- Project meetings to check in

But is there another way?

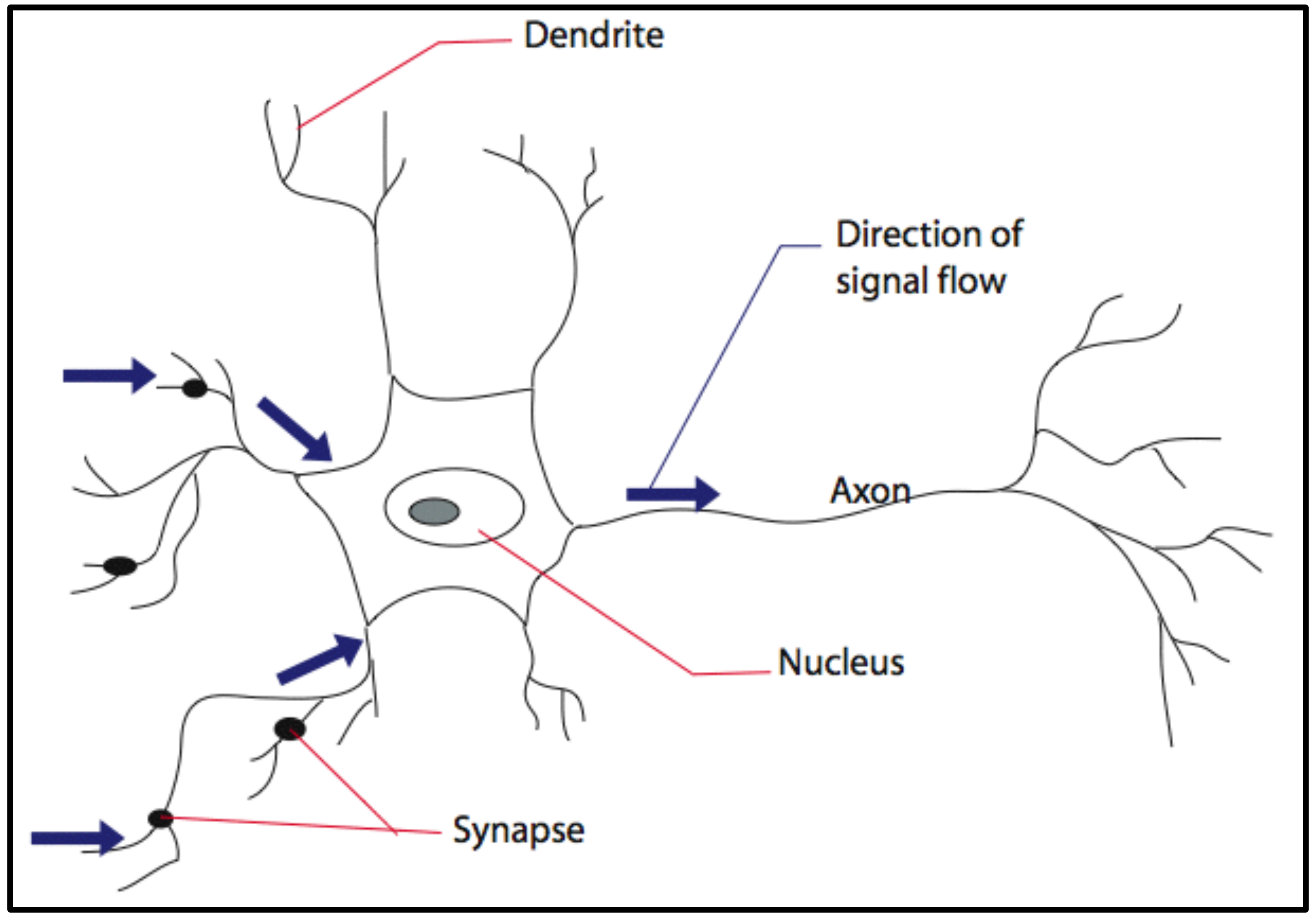
- As cognitive scientists you might want to create a fit to very nonlinear difficult data, and the methods we have used may have difficulty
  - **Or model a system whose properties are not simple, or are difficult to define**
- You may want to model cognition and performance of large groups of structure 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)

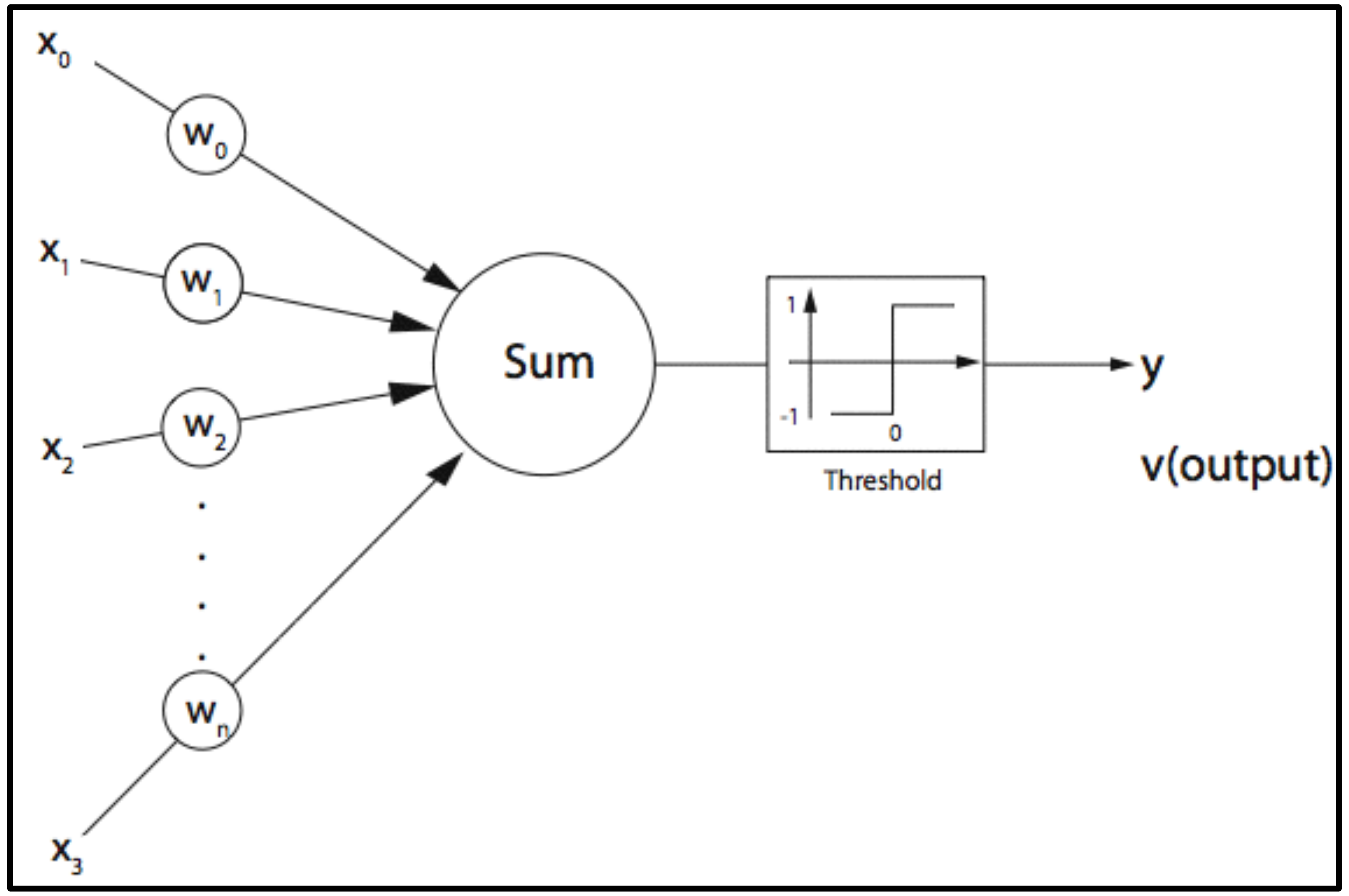
- A study of nature leads to a useful model
  - **Something about the organization of the structures of the brain allows us to solve complex problems with ease, adapt to new situations, and deal with large errors, incomplete information and faults (brain injury)**
  - **Artificial Neural Networks are 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 .^ 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**

Why is a neural structure so powerful (II)?

- 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 ~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**

Look not for the panacea of modeling, look for what's useful for your purposes

- (**Panacea** - a solution or remedy for all difficulties or diseases)
- 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
  - **Mouse example**

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**
- 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**

The threshold logic unit (TLU)

- Takes real-valued inputs (e.g. 0.243 as opposed to 1 or 0 only), $\boldsymbol{x_i}$, each input associated with a "weight" $\boldsymbol{w_i}$ (or "synaptic weight"), which represents the contact between two nerve cells
- Performs a weighted sum of the $\boldsymbol{x}$'s, and if the sum is larger than a threshold (theta), 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

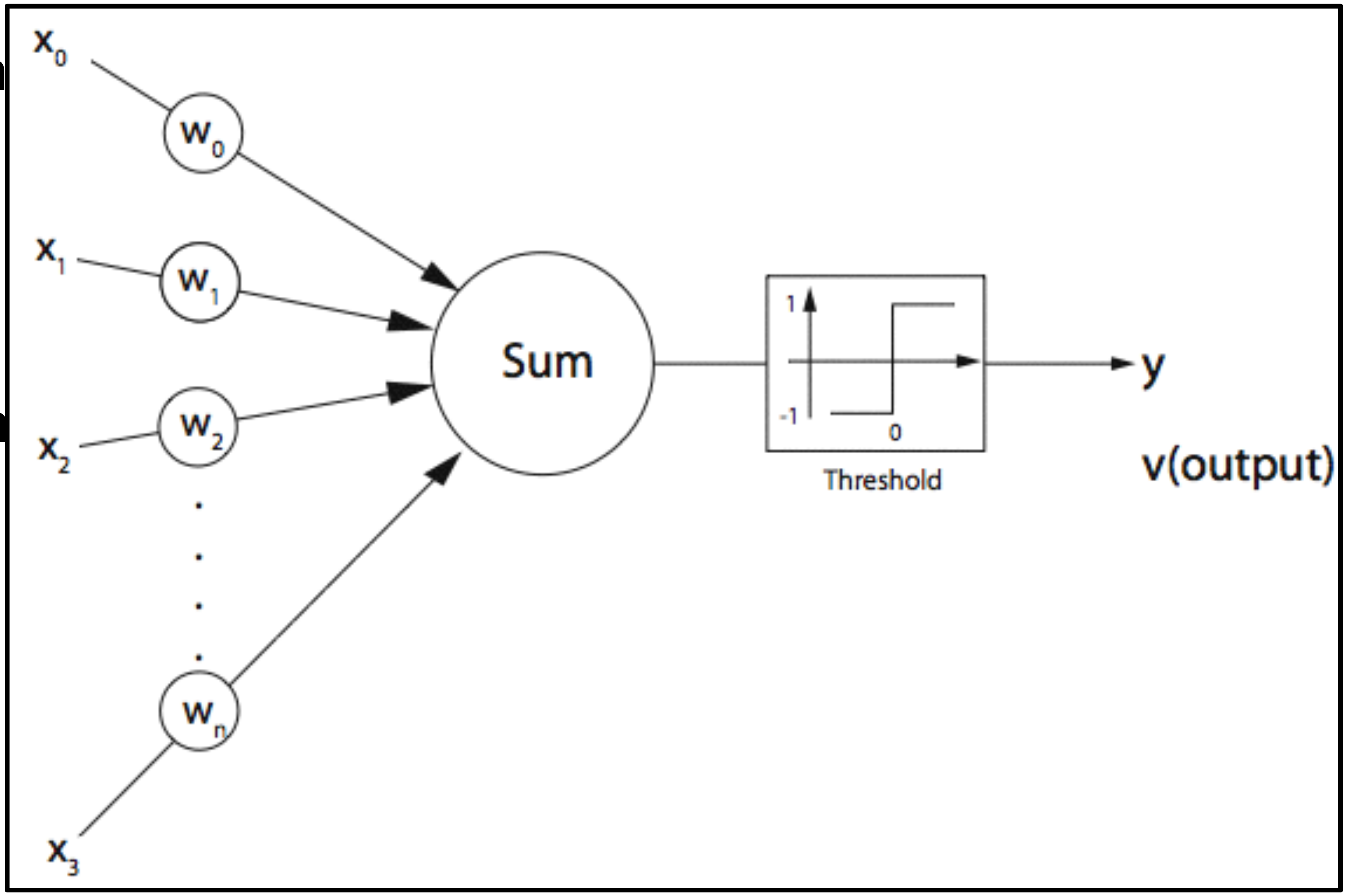
$$\boxed{Activation = \sum_{i=1}^{n} w_i x_i = w^T x}$$

- And whether the neuron fires is determined by

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

Perceptrons are more general than TLU's

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



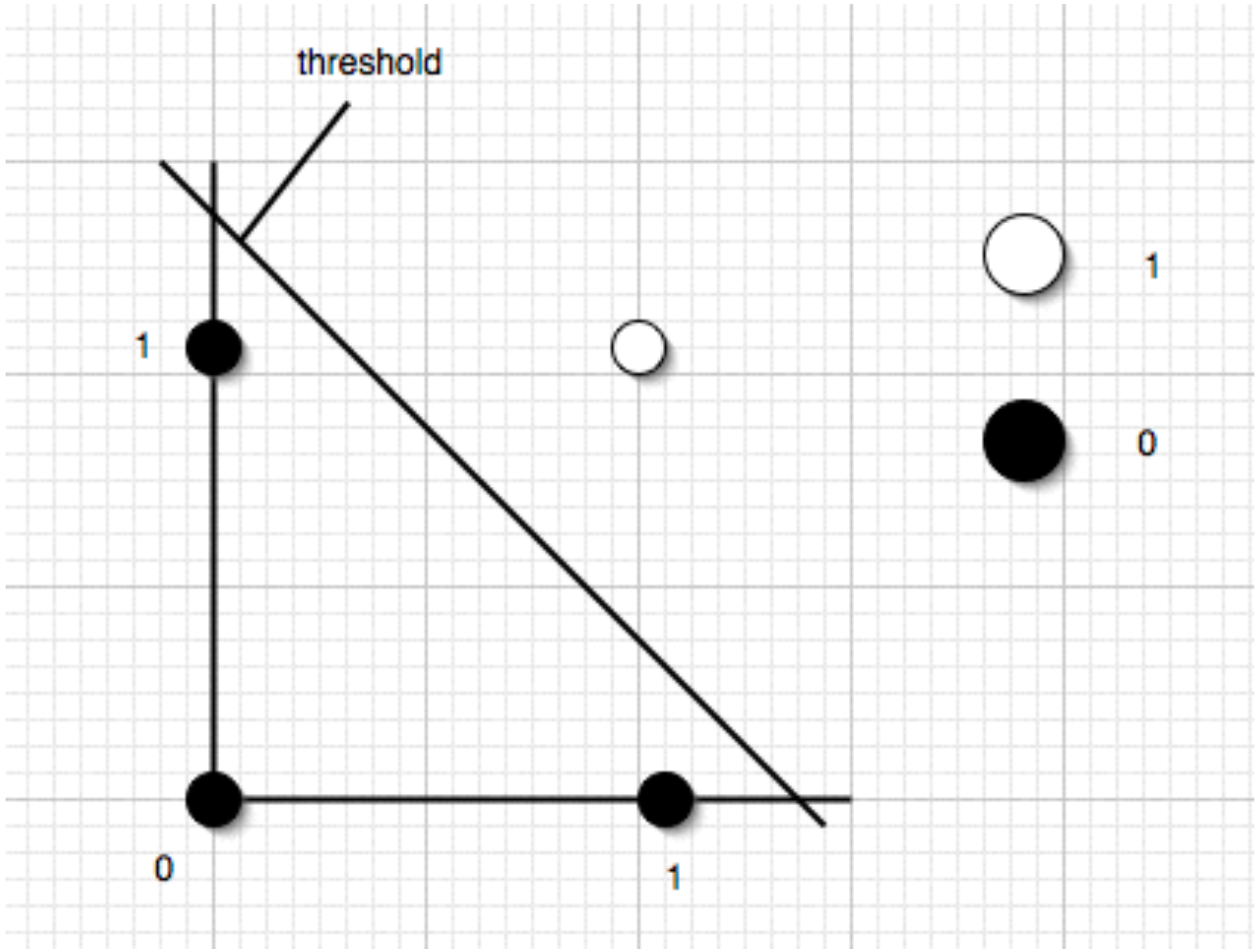
How does it ‘learn?’

- 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’**
  - **think about our discussions of error criteria**
- 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**

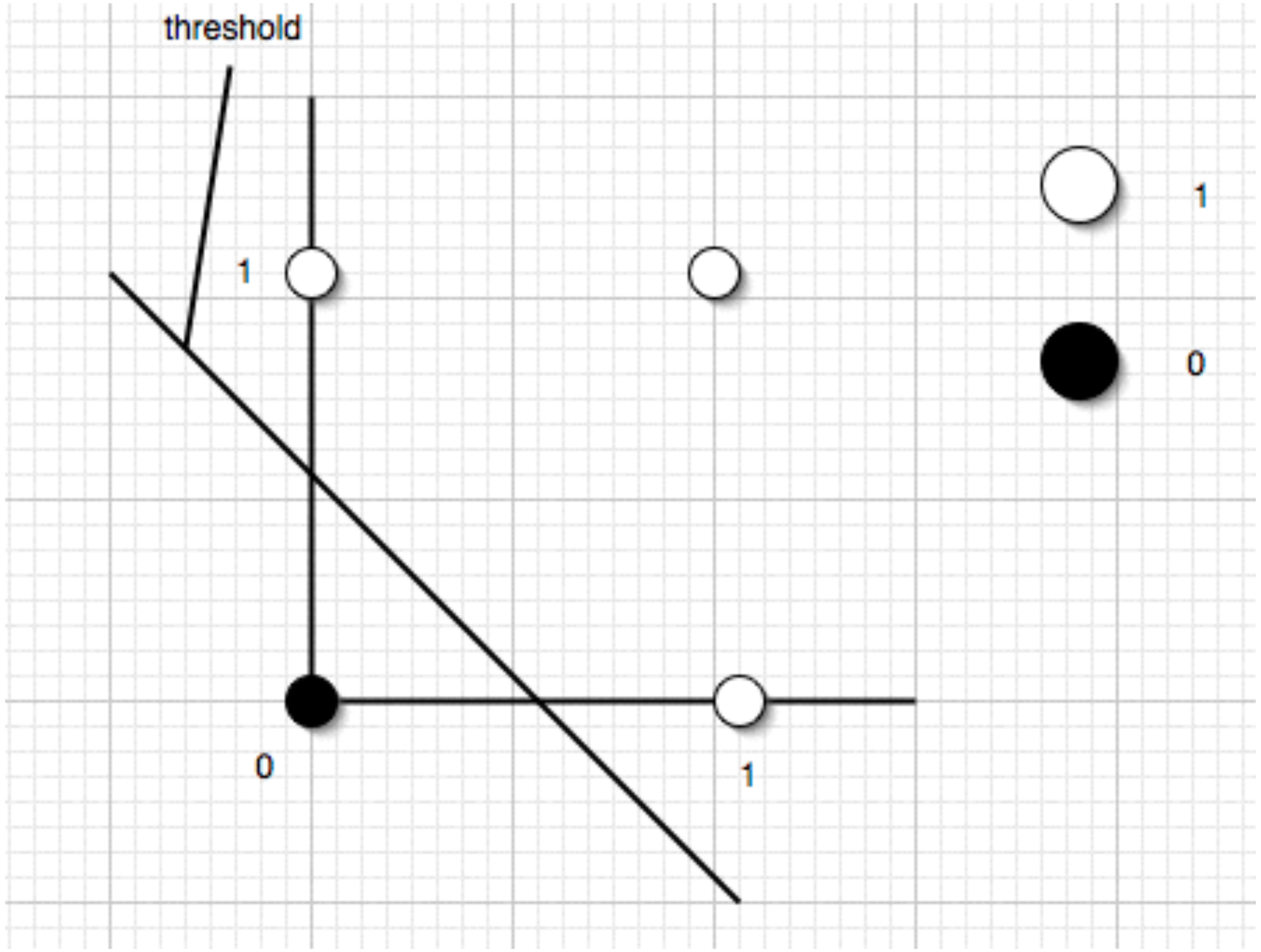
Perceptron learning rule

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 : $\boxed{w_i^{t+1} = w_i^t + \eta(d - y)x_i}$
   1. **Where d is the desired output, y is the actual output of the neuron, and is a parameter called the step size** $\eta(0 < \eta < 1)$
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.


threshold
1
0
1
1
0

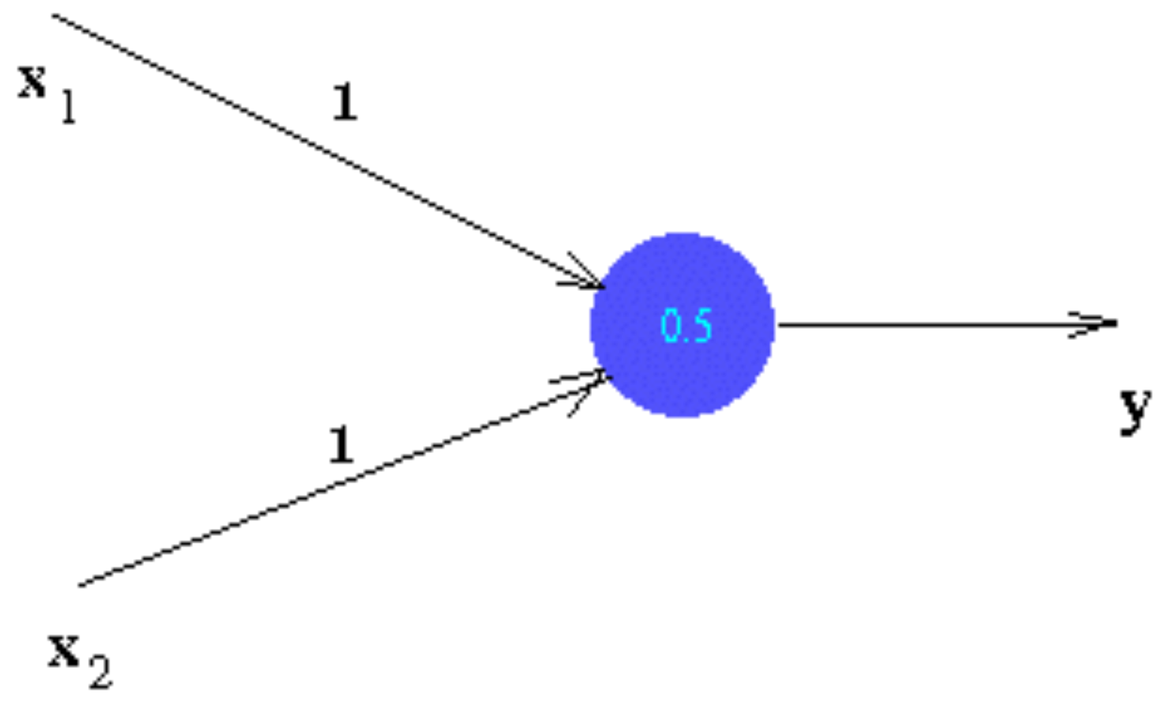
- **Computing "or":**
  - ‘Or’ revieww
  - It is also simple to see that if the threshold is set to 1, then the output will be 1 if at least one input is active. The perceptron in this case acts as the logical "or".


threshold
1
0
1
1
0

- **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

Never fear, we can make more!

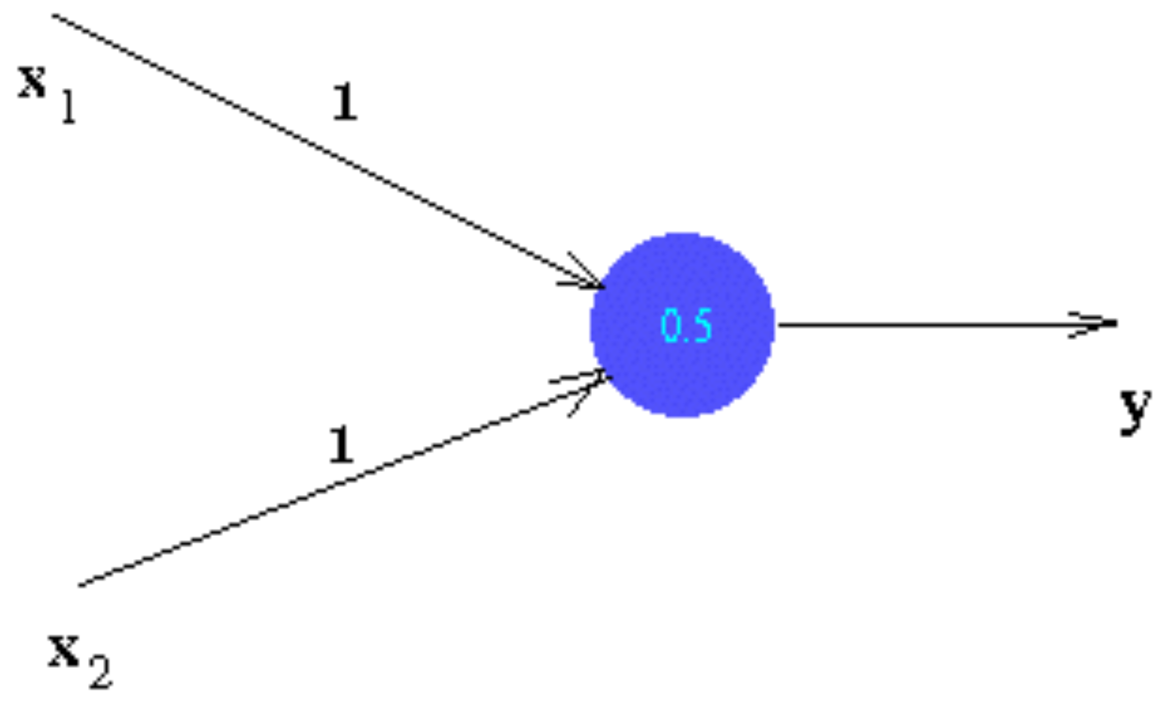
- That’s why combining more than one makes neural networks more general for solving problems

Generic PLA code (not using neural network toolbox or module)

- Original TLU’s did not have learning rule - weights had to be designed
- 50’s Rosenblatt’s main contributions were the perceptron learning rule
- Demo/explanation
  - **Binary classifier**

Limitations of a single neuron

- XOR problem -
  - build a single layer, single unit 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

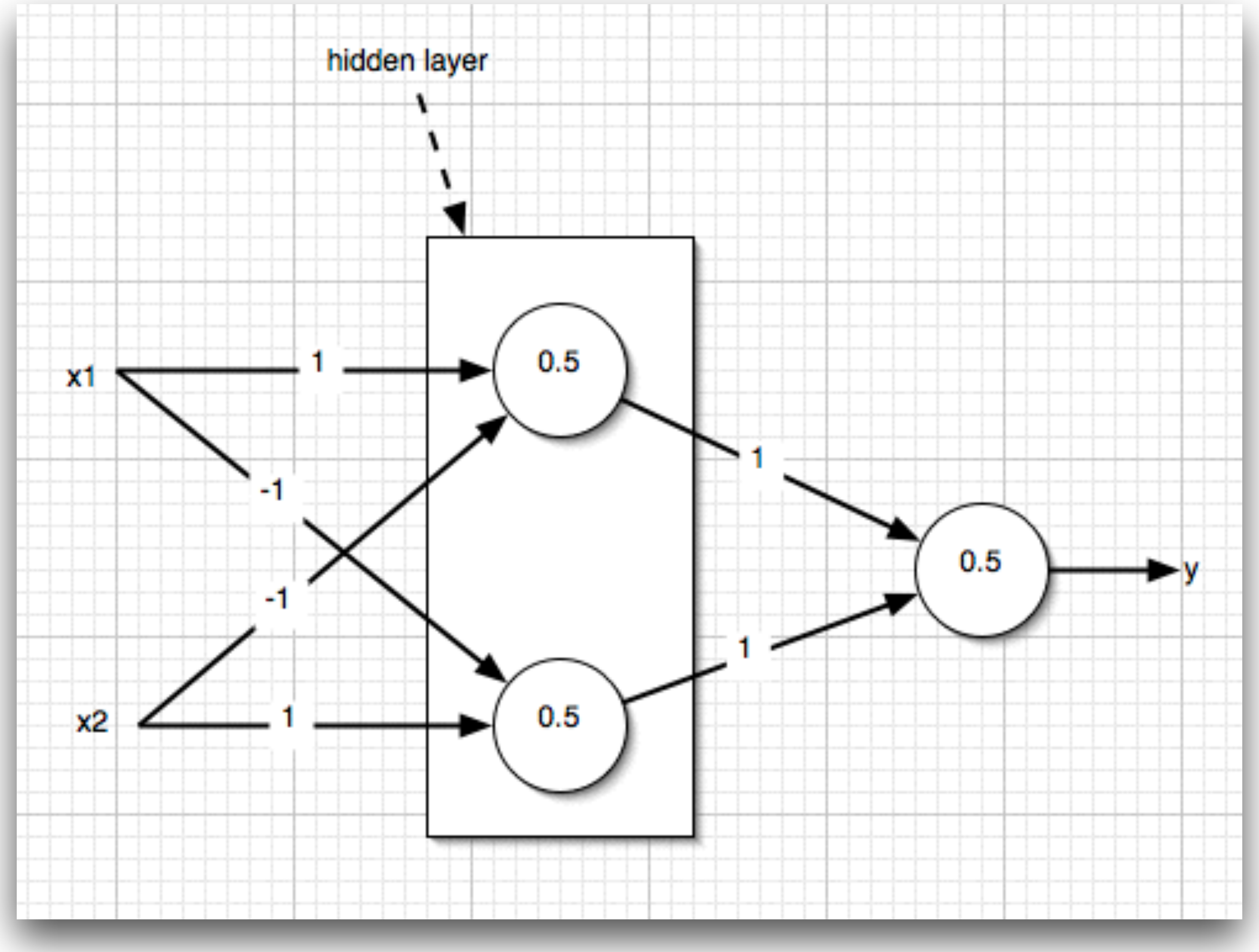
•If the inputs are both 0, then net input is 0 which is less than the threshold (0.5). So the output is 0 - desired output.
•If one of the inputs is 0 and the other is 1, then the net input is 1. This is above threshold, and so the output 1 is obtained.
•But the given perceptron fails for the last case

Limitations of single layer perceptrons (II)

- Widely publicized in the book Perceptrons [MiPa69] by Marvin Minsky and Seymour Papert
- It was not until the 1980s that these limitations were overcome with im- proved (multilayer) perceptron networks and associated learning rules
  - The funding and thus literature for ANN's slowed to a crawl until then!

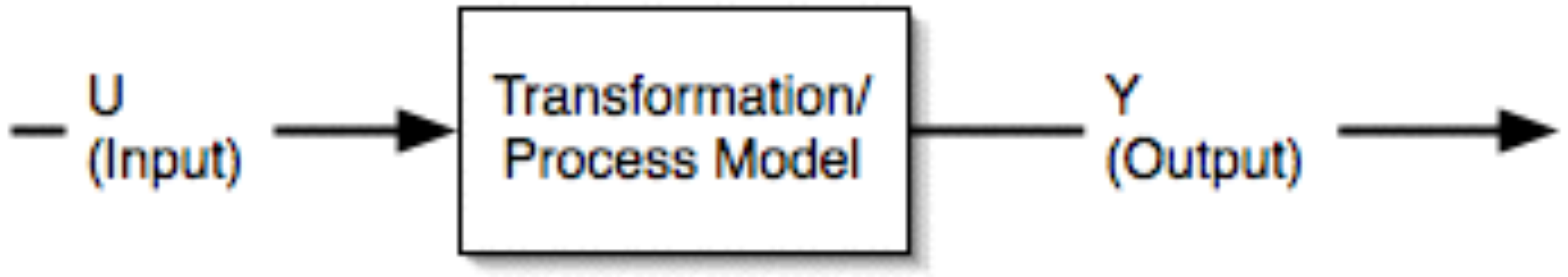
How do we resolve this?

- Feedforward multilayer networks
  - **Simple implementation**
  - **Computational capability**
  - **Input-output data**
  - **No feedback (signals only travel forward)**
- It can be shown that by connecting together multiple TLU's in a two layer network we can solve the XOR problem
  - **Implements two linear decision boundaries**

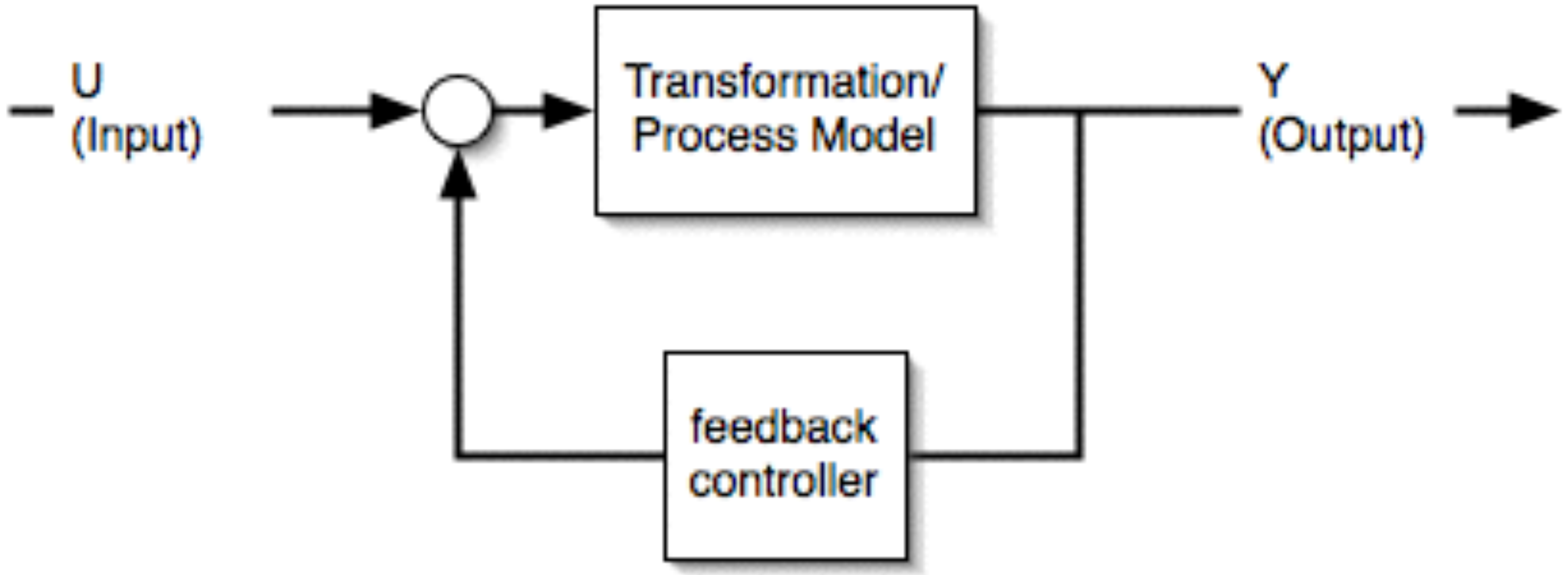


An important concept…

- Feedforward system



- Feedback system



Feedforward-feedback example

- Position control of a motor angle or human limb joint angle
- Path planning
- Feedforward has advantages and drawbacks
  - **Main drawback - model is never perfect, and noise can cause severe drift over time, leading to inaccuracies**
    - *Any small error in the model tends to cause massive inaccuracies*
    - *Any disturbances cause errors - noise or external inputs*
  - **Advantage - simplicity in computation and sensor requirements**
- Feedback has advantages of robustness and error correction

A common feedback example - inverted pendulum control

- People standing or walking can be modeled as inverted pendulums

Another example - robotics application

- Big dog video

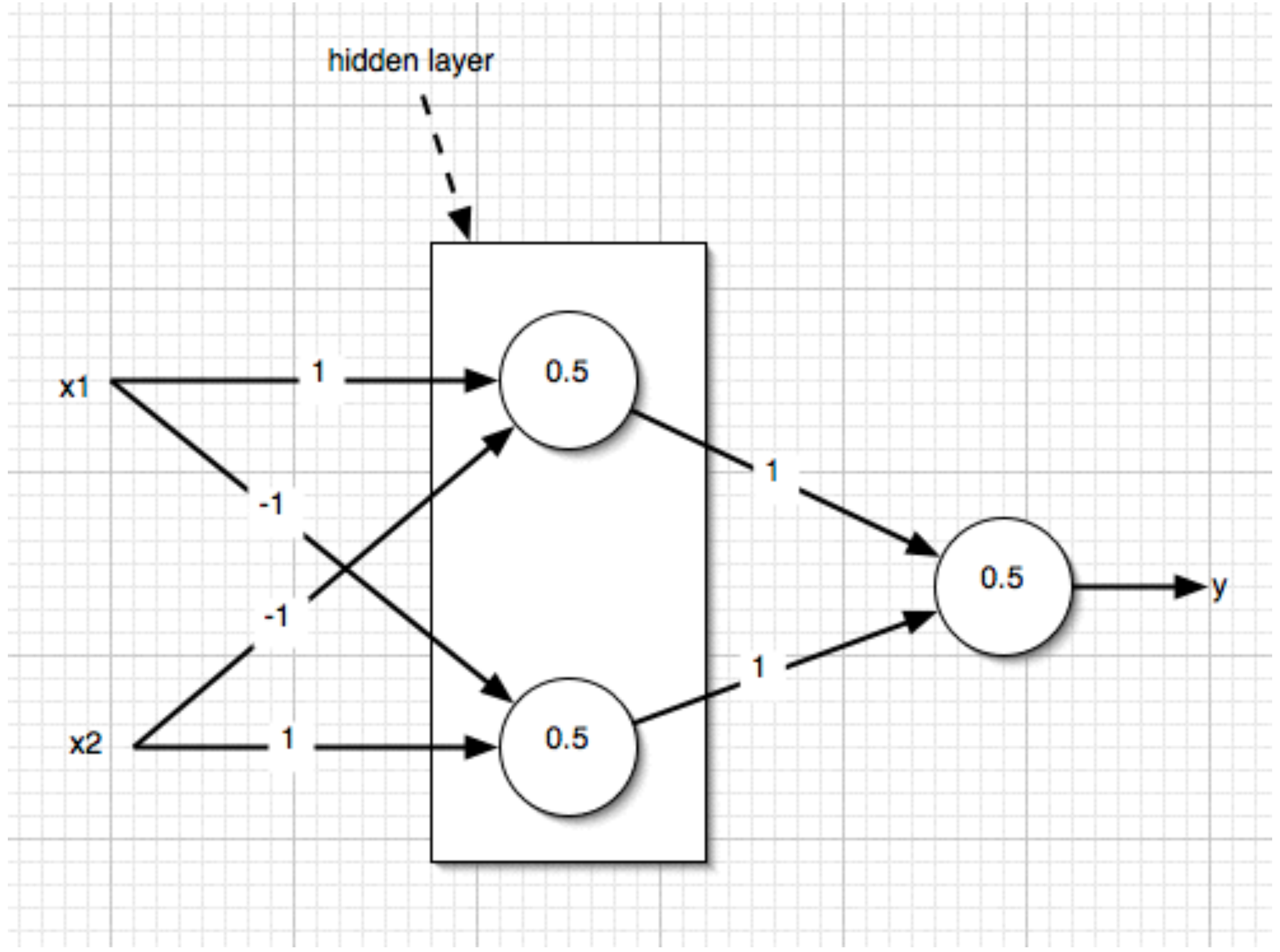
- Littledog video

Back to neural networks…

- Now that we have a concept of feedforward and feedback, and how single unit perceptrons work, let’s move on to combinations of units to multi-layer networks
- More details next time but main applications of ANN’s are
  - **Function fitting**
    - Fit this data without an equation!!!
  - **Classification**
    - blue cat or red cat?

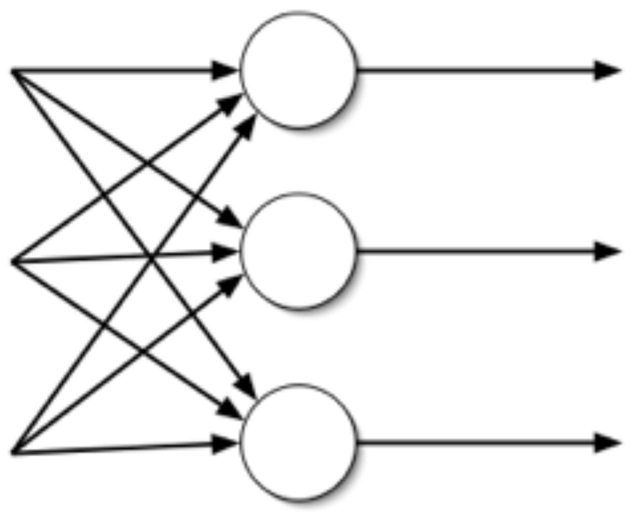
Multilayer networks

- Hidden nodes/layers - intermediate node layers which are NOT directly connected to the outside world (input or output)

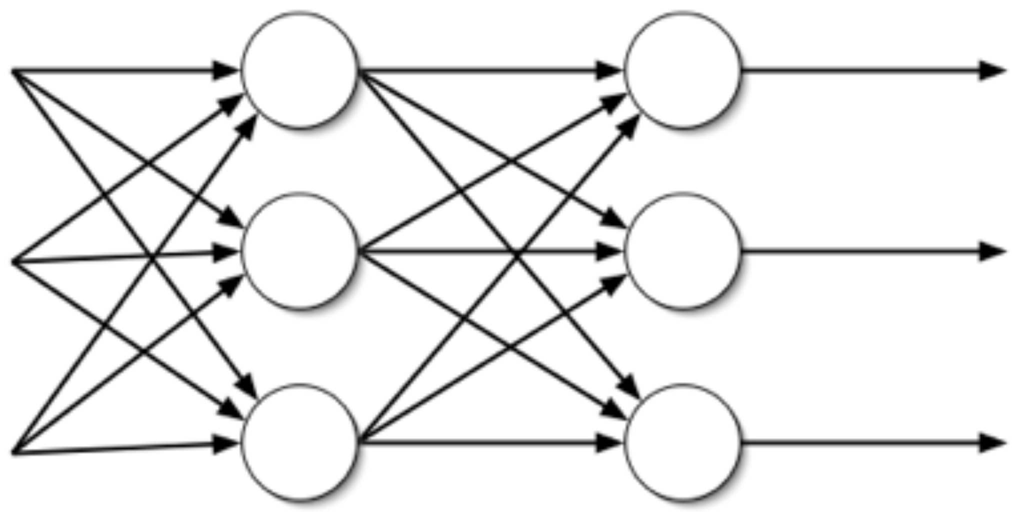


Some typical network topologies

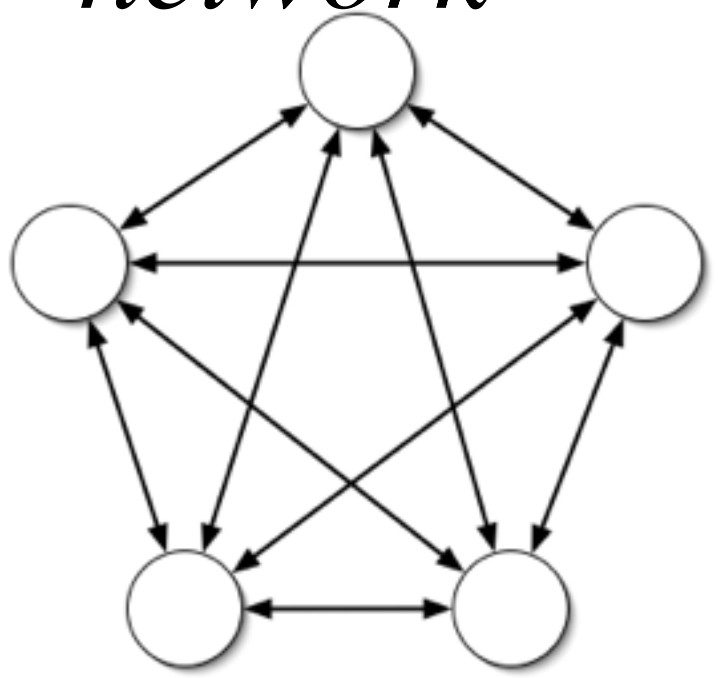
Single layer perceptron



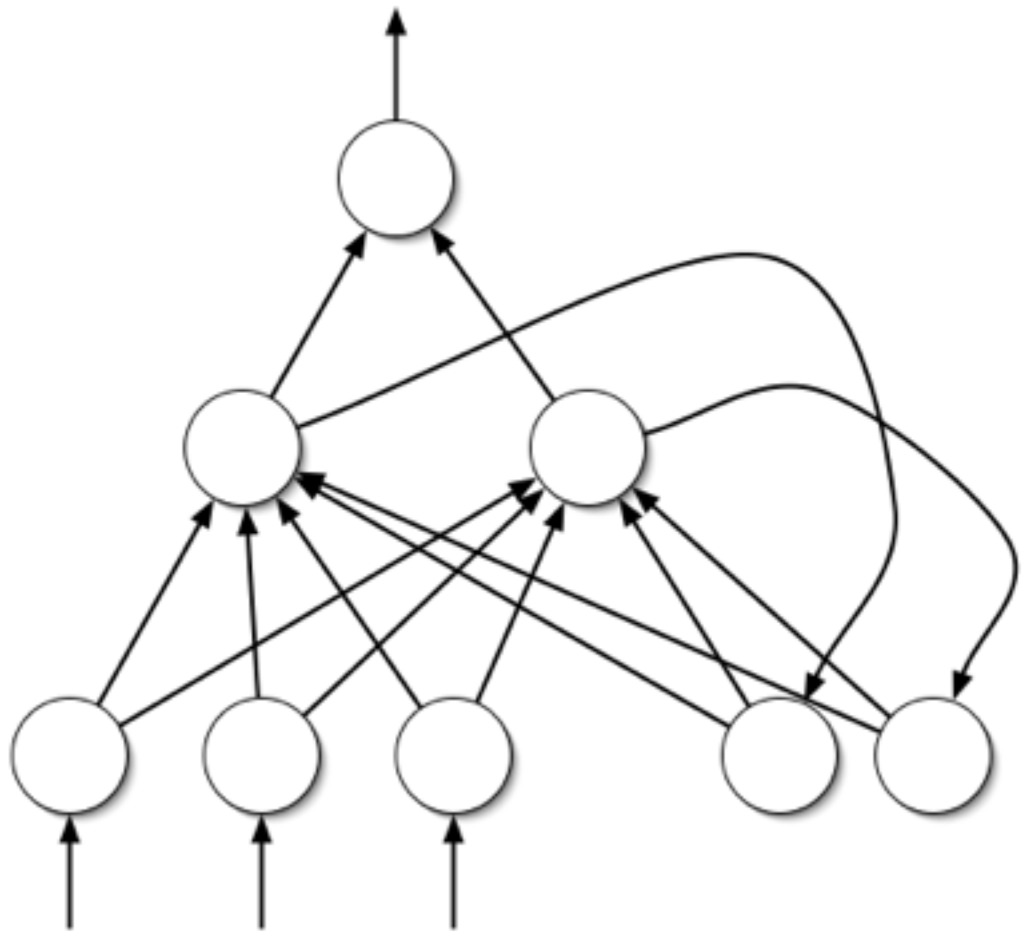
Multi-layer perceptron



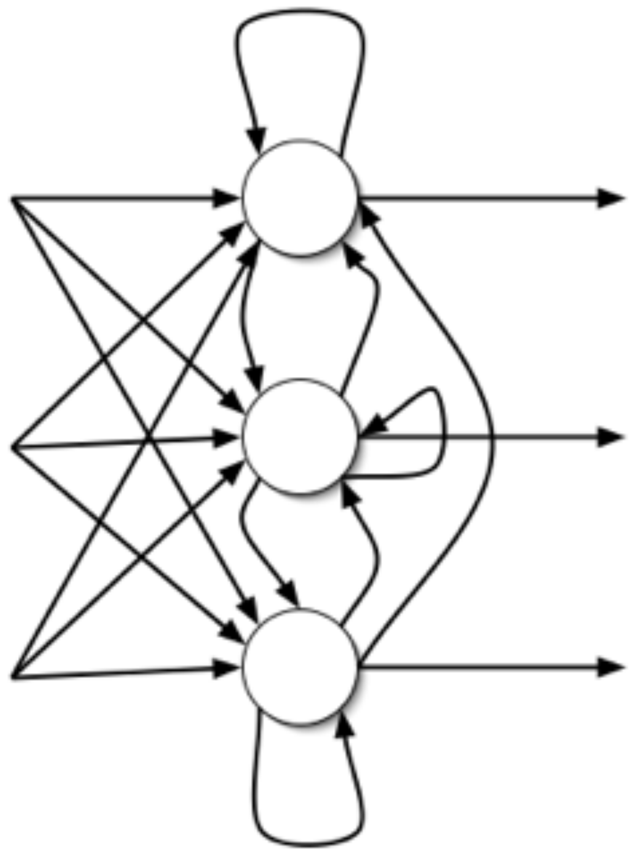
Hopfield network



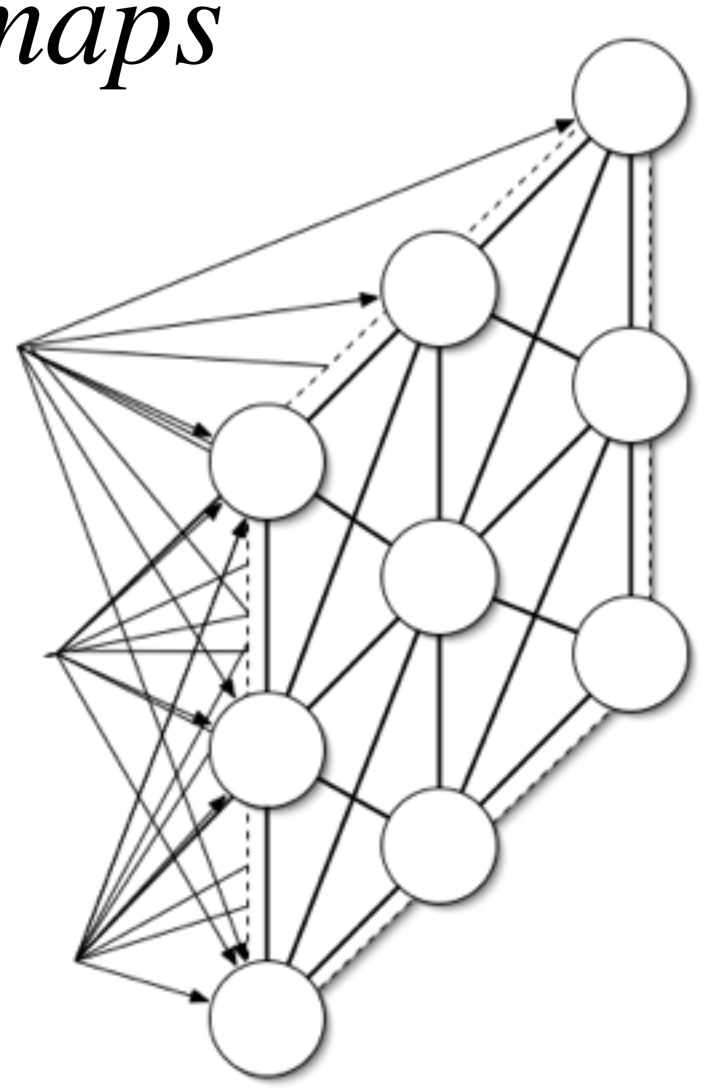
Elman recurrent network



Competitive networks



Self-organizing maps



Other activation function concepts

- Threshold
- Sigmoid
- Logarithmic
- Linear
- Many others

Neural networks in Python

- Tensorflow
- Other tools
- For next time

Neural Network Demos in matlab

- In matlab (you need the Neural Network Toolbox)
  - **nnd2n1 One-input neuron demonstration.**
  - **nnd2n2 Two-input neuron demonstration.**
  - **nnd4db Decision boundaries demonstration.**
  - **nnd4pr Perceptron rule demonstration.**
  - **nnd9sdq Steepest descent for quadratic function demonstration.**
  - **nnd11nf Network function demonstration.**
  - **nnd11bc Backpropagation calculation demonstration**
  - **nnd11fa Function approximation demonstration.**
  - **nnd11gn Generalization demonstration.**