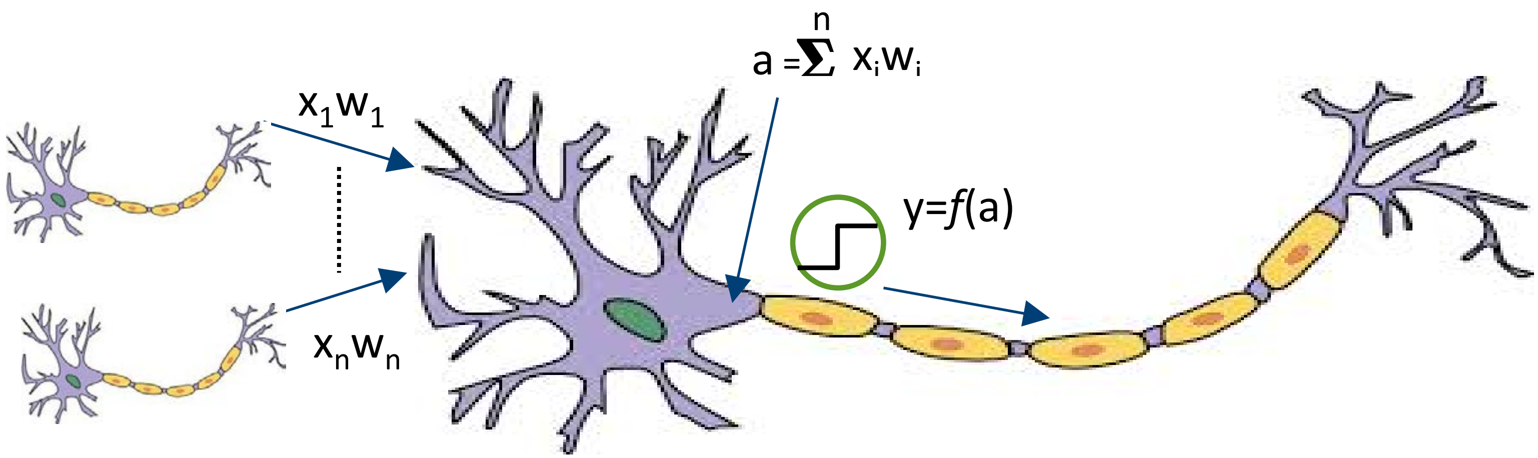
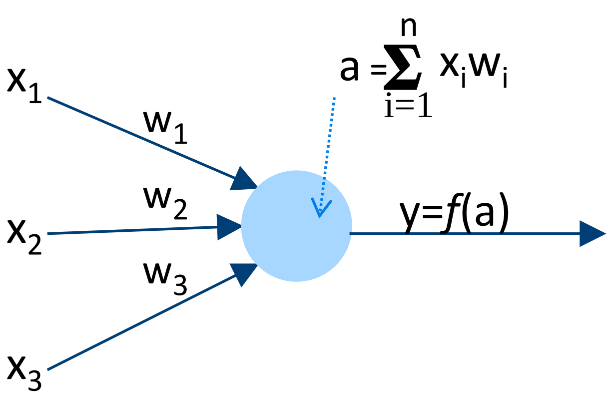
CISC452/CMPE452/COGS 400
Introduction to Artificial Neural Networks

Farhana Zulkernine

Learning in the Neuron



McCulloch and Pitts Neuron Model

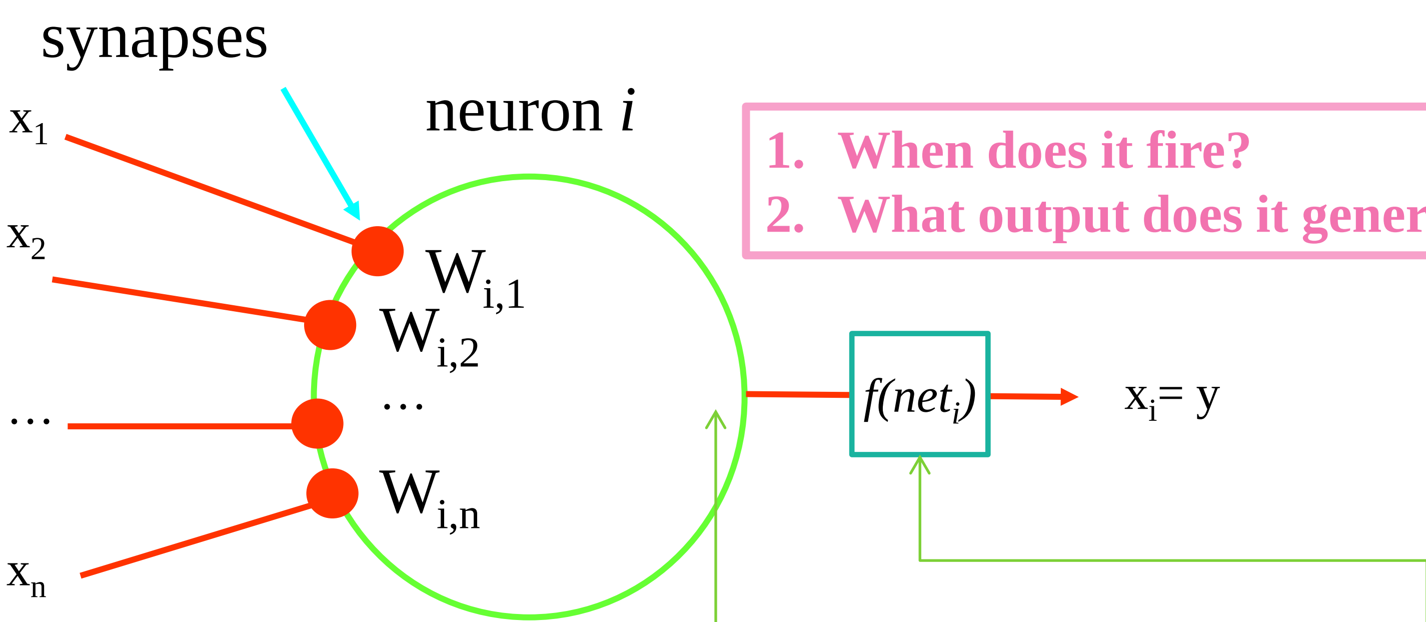


The weights w_i take on *real values* $w_i \in \mathbb{R}$

Activation is the weighted sum of all incoming potentials.

f(a) can be any function that generates a spike (high value) at a given threshold value θ to mimic the scenario of *Action Potential*.

An Artificial Neuron



1. **When does it fire?**
2. **What output does it generate?**

Net input signal

$$a = \text{net}_i(t) = \sum_{j=1}^{n} w_{i,j}(t)\,x_j(t)$$

Output

$$x_i(t) = f_i(net_i(t)) \;\text{ or }\; y = f(a)$$

The Activation Function

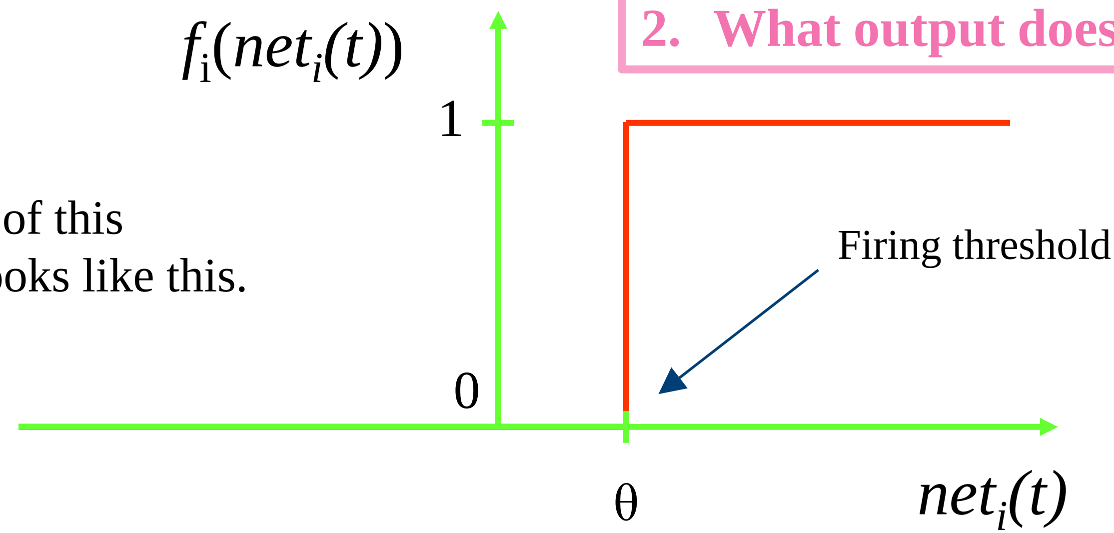
One possible choice is a threshold function:

Therefore, we call this a threshold neuron.

$$f_i(\text{net}_i(t)) = 1, \quad \text{if} \quad \text{net}_i(t) \geq \theta$$
$$= 0, \quad \text{otherwise}$$

1. **When does it fire?**
2. **What output does it generate?**

The graph of this function looks like this.



Output Functions $f(net_i)$

- The simplest node functions are:

1. **Identity**, $f(\text{net}) = \text{net}$, and its non-negative variant $f(\text{net}) = \max(0, \text{net})$

2. Constant functions $f(\text{net}) = c$

3. Signum function

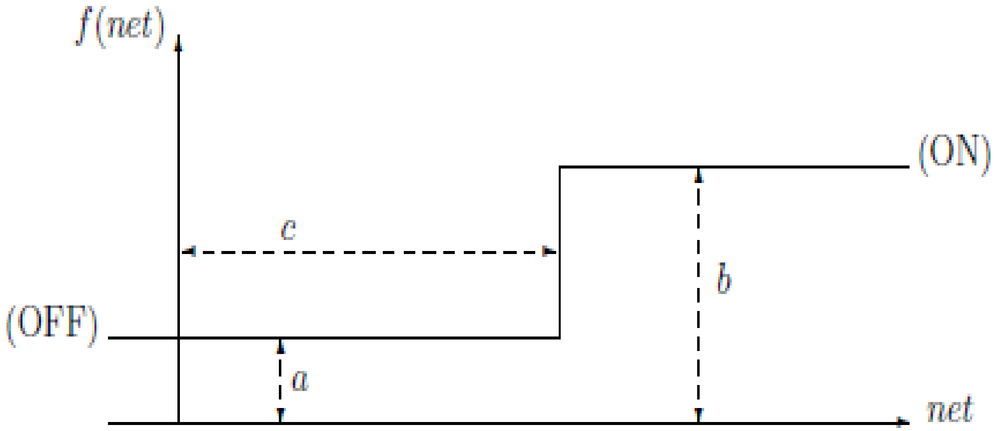
$$f(net) = \begin{cases} +1 & \text{if } net > 0 \\ -1 & \text{if } net < 0 \\ 0 & \text{if } net = 0 \end{cases}$$

4. Step Function

- Simplest function that captures the idea of a "firing threshold"
- Can be used as a class identifier
- Problem: Very small change in $net_i(t)$ can cause a spike and hence change the output

$f(\text{net}) = a$ if $net < c$

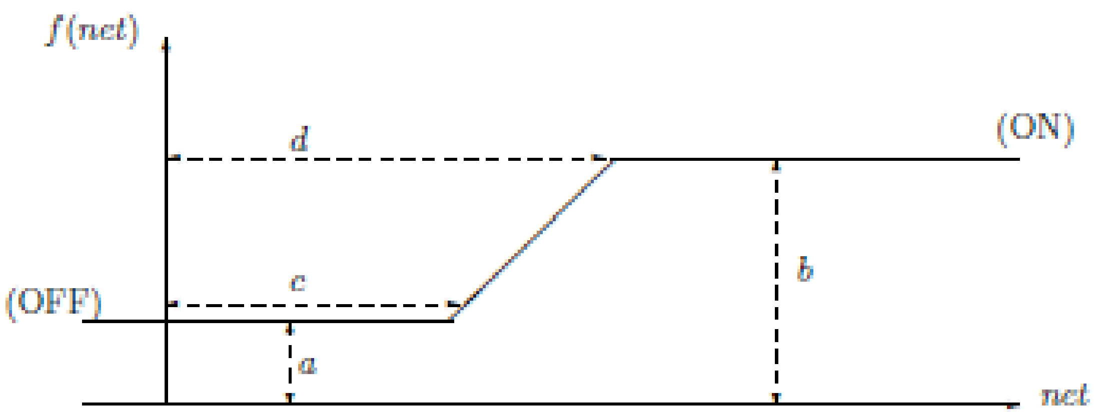
$= b$ if $net >= c$



5. a) Ramp Function

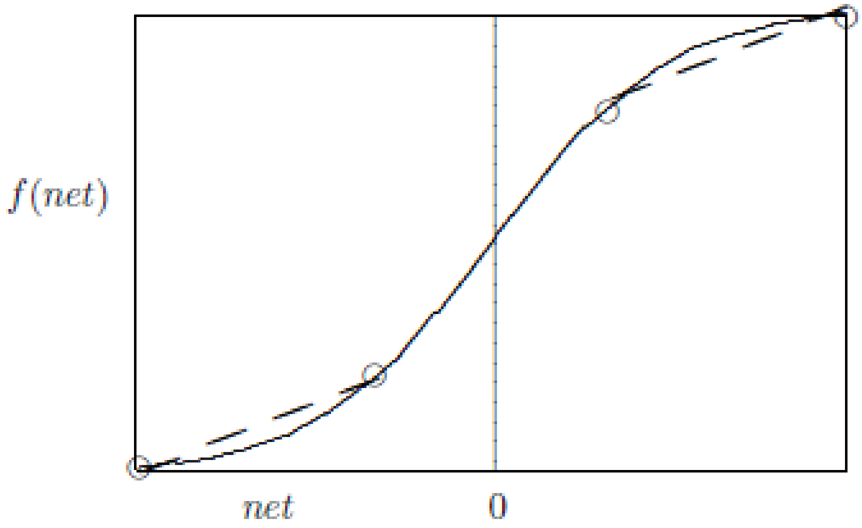
- The ramp function is continuous and almost everywhere differentiable in exchange of the simple ON/OFF description of the output.

$$f(net) = \begin{cases} a & \text{if } net \leq c \\ b & \text{if } net \geq d \\ a + \frac{(net-c)(b-a)}{d-c} & \text{otherwise} \end{cases}$$



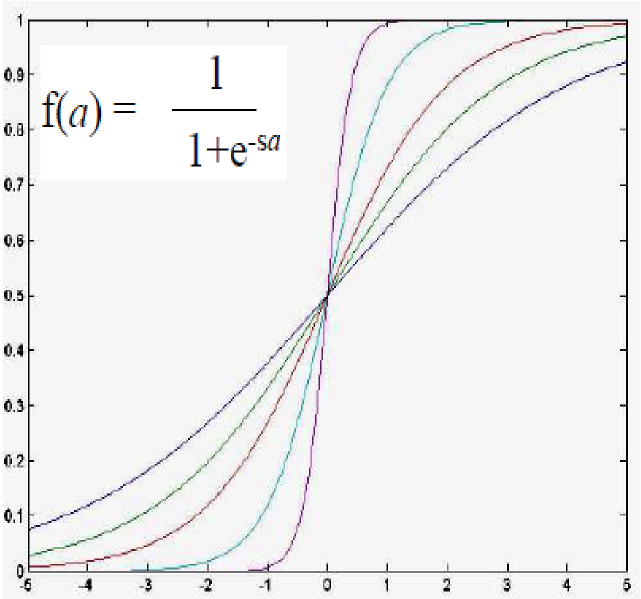
5. b) Piecewise Linear Functions

- Consist of finite number of linear segments, and are thus differentiable almost everywhere.
- Easier to compute than general nonlinear functions such as sigmoid functions.
- Can be used to avoid sudden change in output.



6. Sigmoid Function

- These functions are continuous and differentiable everywhere, and asymptotically approach saturation values (0 and 1 as shown in the picture)
- The parameter s controls the slope of the sigmoid function. Greater s value will give steeper curve.

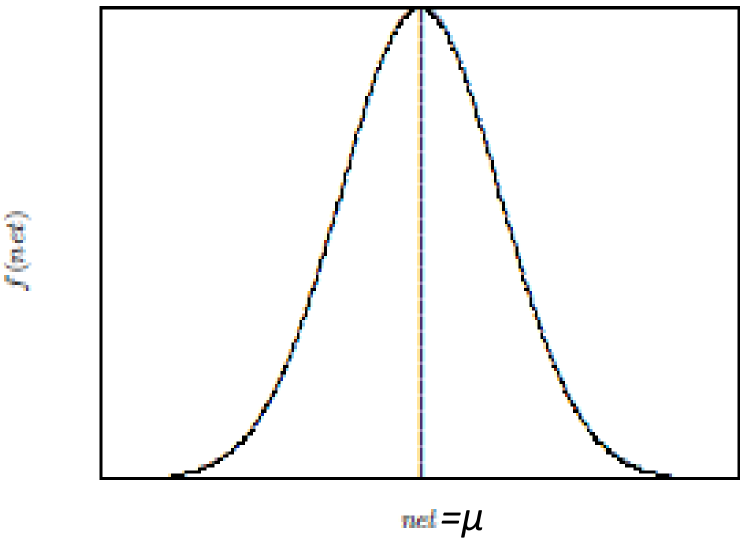


$$\lim_{net \to +\infty} f(net) = 1$$

$$\lim_{net \to -\infty} f(net) = 0$$

7. Gaussian Functions

- Continuous bell-shaped functions.
- Also called 'radial-basis' function.
- f(net) asymptotically approaches 0 (or some constant) for large magnitudes of net, with a single maximum for net = μ, say $\mu = 0$.

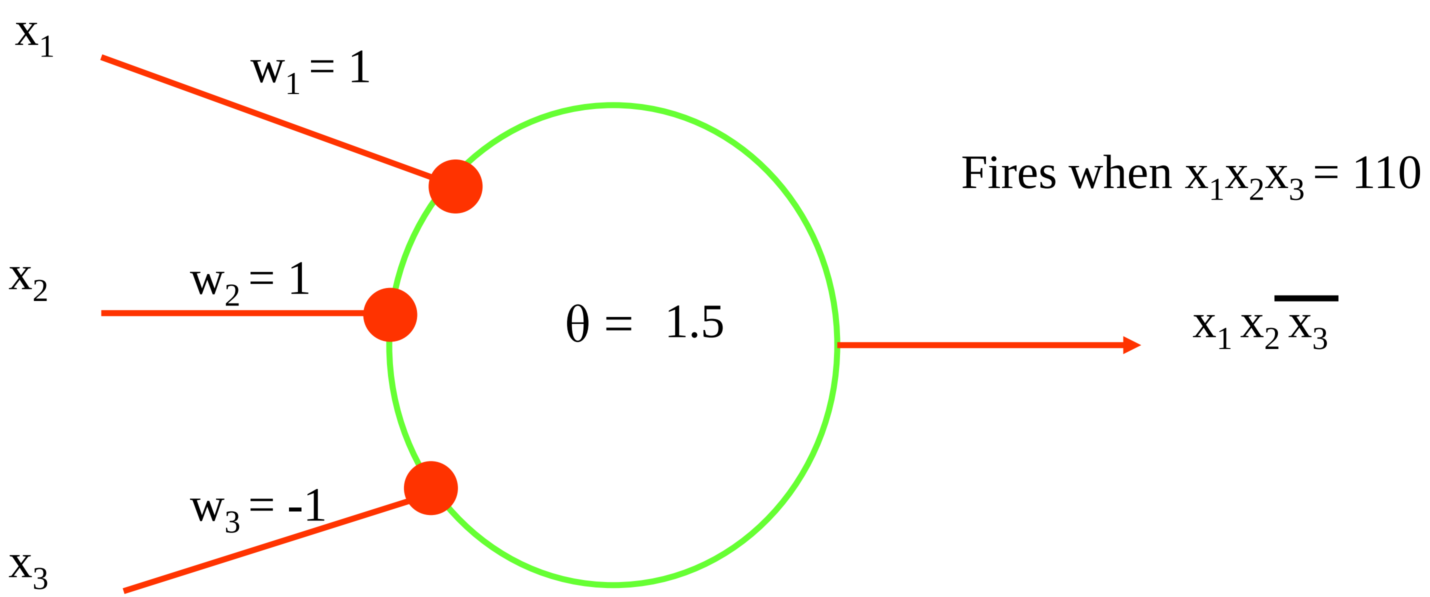


$$f(net) = \frac{1}{\sqrt{2\pi}\sigma} exp[-\frac{1}{2}(\frac{net - \mu}{\sigma})^2]$$

Binary Analogy: Threshold Logic Units (TLUs) – AND Gate

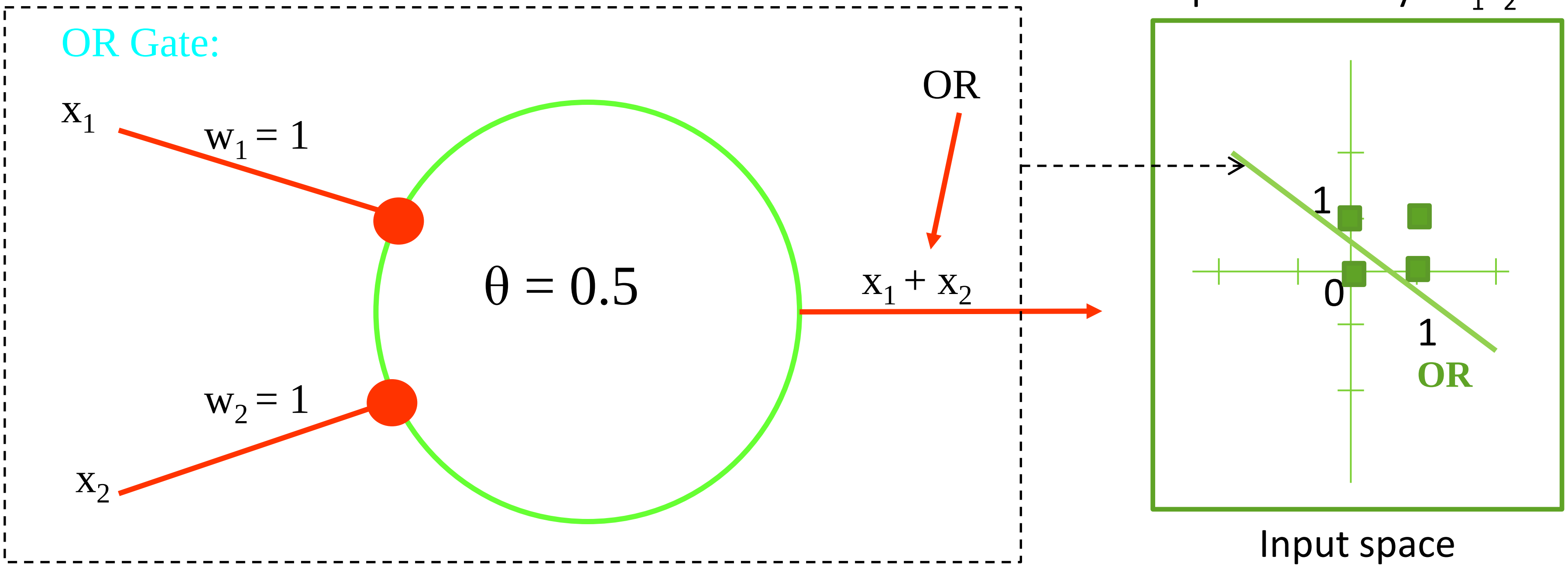
TLUs are similar to the threshold neuron model, except that ***TLUs only accept binary inputs*** (0 or 1).

Example: AND Gate with θ = 1.5



TLUs and Linear Separability – OR

Each point is represented by x_1x_2

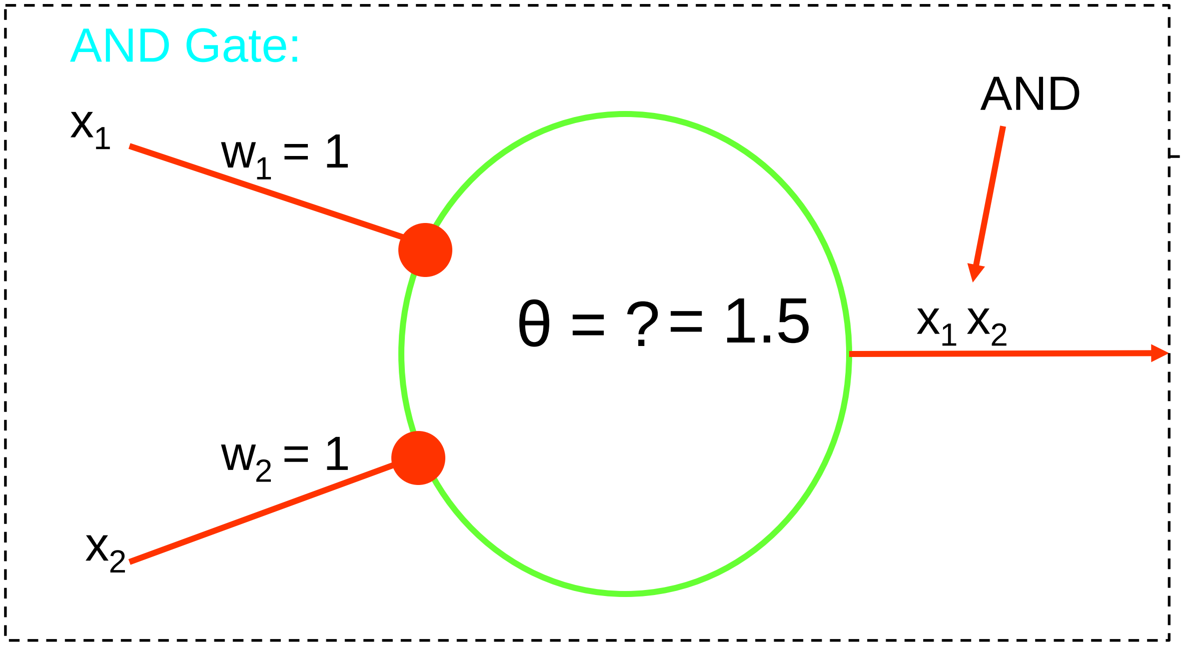


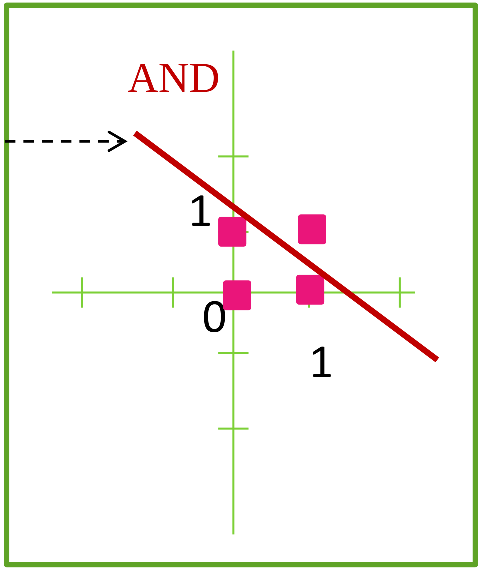
Input space

OR Gate: 00= 0 and 01, 10, 11 = 1

TLUs and Linear Separability

Each point is represented by x_1x_2





AND Gate: 00, 01, 10= 0 and 11 = 1

Linear Separability as Functional Mapping

- To explain linear separability, let us consider the function f: $R^n \rightarrow \{0, 1\}$ with.

$$
\begin{aligned}
f(x_1, x_2, \ldots, x_n) &= 1, \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq \theta \\
&= 0, \quad \text{otherwise}
\end{aligned}
$$

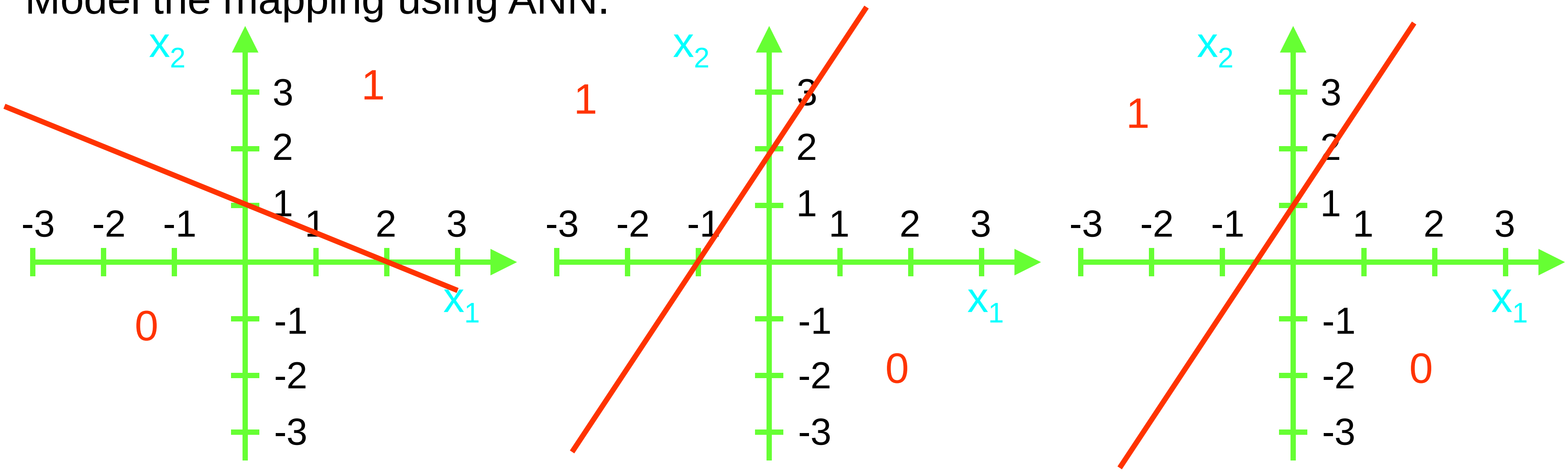
where x_1, x_2, …, x_n represent real numbers (not TLU).

This is the exactly the function that our threshold neurons use to compute their output from their inputs.

Examples – Linear Separability

Let's say we have a two-dimensional Input space (x1 and x2 and n = 2) and the following lines separate the output categories into 1 and 0.
Model the mapping using ANN.

$$f(x_1, x_2, \ldots, x_n) = 1, \quad \text{if} \quad \sum_{i=1}^{n} w_i x_i \geq \theta$$
$$= 0, \quad \text{otherwise}$$



For **θ = 2**, $w_1 = 1$, $w_2 = 2$

For **θ = 2**, $w_1 = -2$, $w_2 = 1$

For **θ = 1**, $w_1 = -2$, $w_2 = 1$

Equation of the straight line $\mathbf{w_1 x_1 + w_2 x_2 = \theta}$
To model it using an ANN, we need to find the weights for a given threshold.

Linear (cont…)

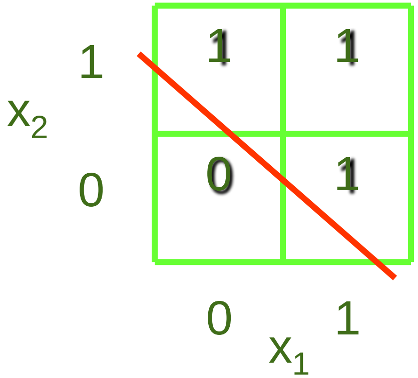
- Therefore, it means that by changing θ and the weights, the line can be aligned in any direction on the 2-dimensional surface to separate two categories of input based on the outputs 1 and 0.
- Here θ is called the bias.
- Training a network means finding the best fitting values of θ and the weights to categorize a given set of values correctly when the categories are already known.

Linear Separability (cont…)

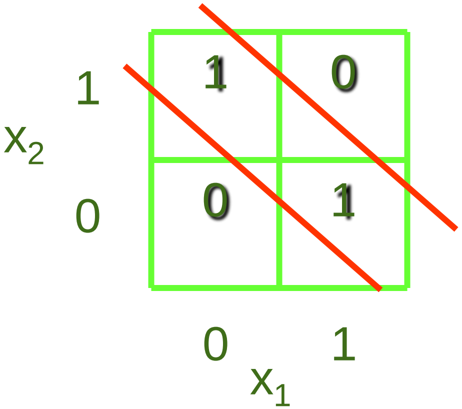
- As we have seen, a two-dimensional input space can be divided by any straight line.
- A three-dimensional input space can be divided by any two-dimensional plane.
- In general, an *n-dimensional input space can be divided by an (n-1)-dimensional plane* or hyperplane.
- Of course, for $n > 3$ this is hard to visualize.

What if the data are not linearly separable?

- A function f:$\{0, 1\}^n \rightarrow \{0, 1\}$ is linearly separable if the space of input vectors yielding 1 can be separated from those yielding 0 by a linear surface (hyperplane) in *n* dimensions.
- Examples (two dimensions):



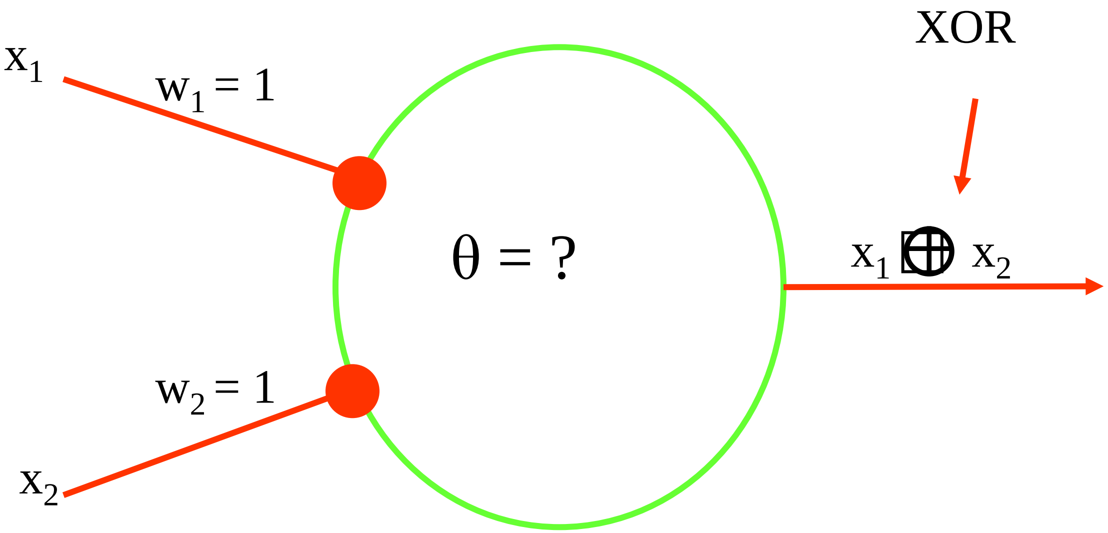
linearly separable (OR)

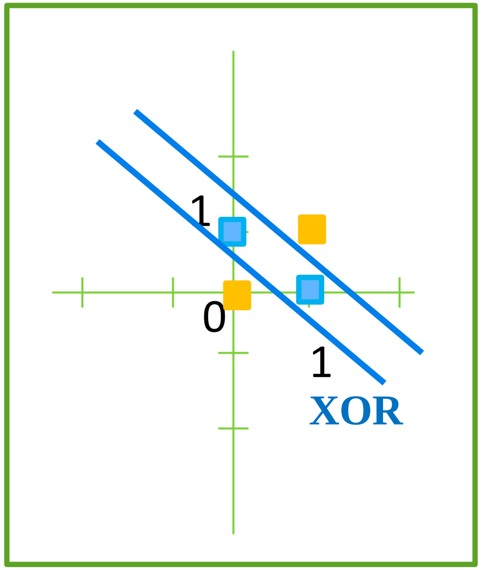


linearly inseparable (XOR)

TLUs cannot implement XOR

Each point is represented by x_1x_2





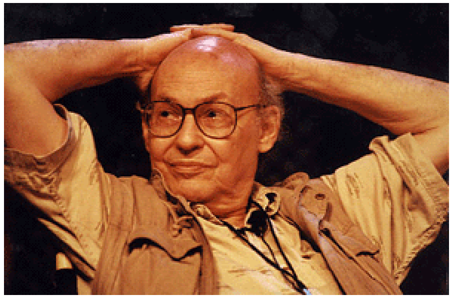
XOR Gate: 00,11=0 and 10,01=1

NOT linearly separable!!!

TLUs CANNOT realize functions that are NOT **linearly separable**.

Minsky and Papert

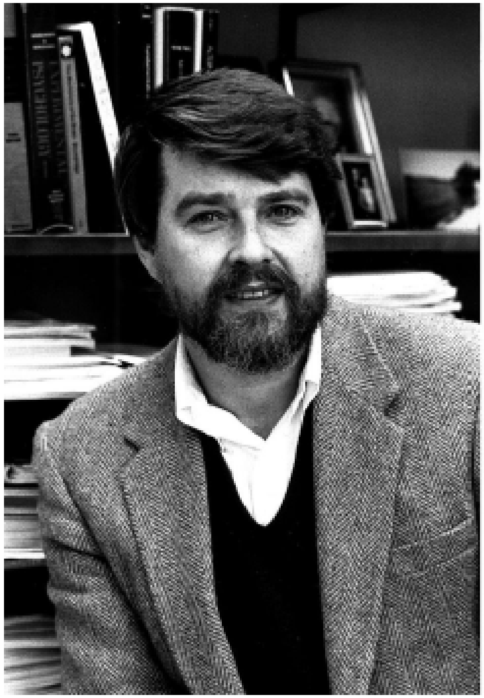
- In 1969, Marvin Minsky and Seymour Papert, two “PSS” researchers at MIT studied the ANNs and revealed that a two-layered (input and output) network cannot handle all logical relations – specifically the XOR.
  - It implies that ANNs lacked the power of a Turing machine.
- Federal funding for ANNs immediately stopped.



Minsky

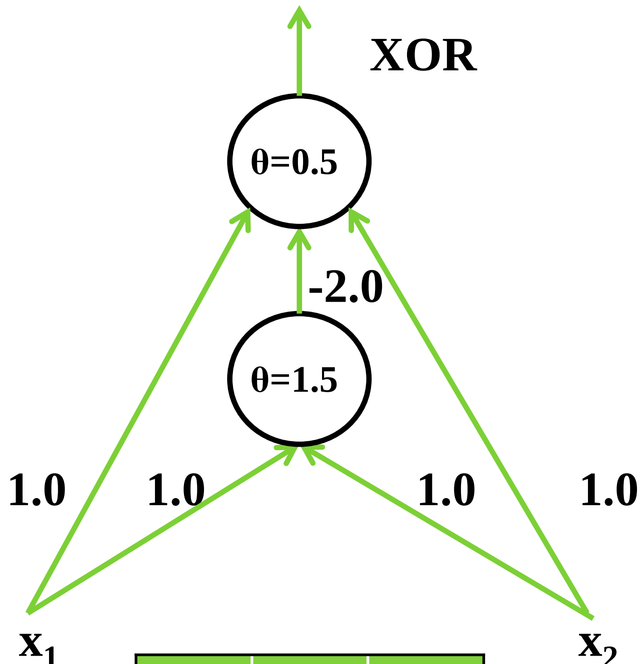
The Big Breakthrough

- David Rumelhart and Jim McClelland developed Parallel Distributed Processing (PDP)



Multilayer Networks with *Inhibition*

- The solution that Rumelhart and McClelland propose is simple: Add a third layer between input and output.
- 3 layers enable creating an XOR gate and handle all logic.
- Note that middle layer neuron inhibits output layer neuron when $x_1 = x_2 = 1$.
  - New for ANN



A	B	D
0	0	0
0	1	1
1	0	1
1	1	0

Activation as a Vector Product

- The net input signal is the sum of all inputs:

$$\text{net}_i(t) = \sum_{j=1}^{n} w_{i,j}(t) x_j(t)$$

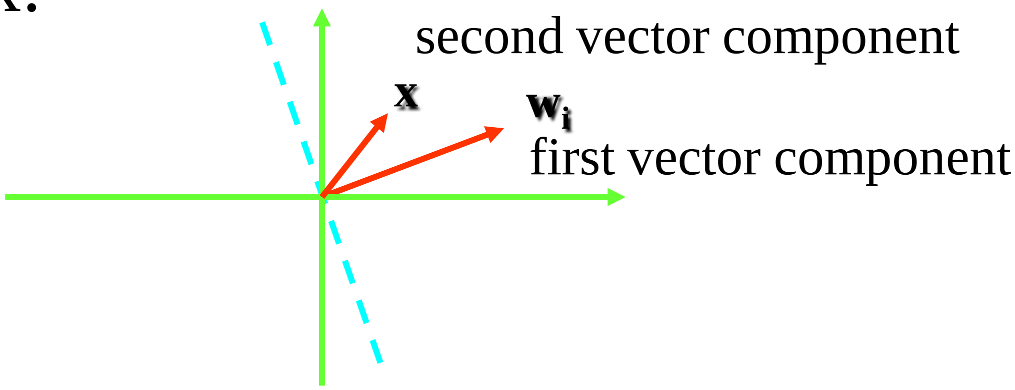
This can be viewed as computing the **inner product** of the **vectors** $\mathbf{w_i}$ and $\mathbf{x}$: https://en.wikipedia.org/wiki/Dot_product

$$\text{net}_i(t) = \| w_i(t) \| \cdot \| x(t) \| \cdot \cos\alpha$$

where α is the **angle** between the two vectors.

Capabilities of Threshold Neurons

- Let us assume that the threshold $\theta = 0$ and illustrate the function computed by the neuron for sample vectors w_I and x:



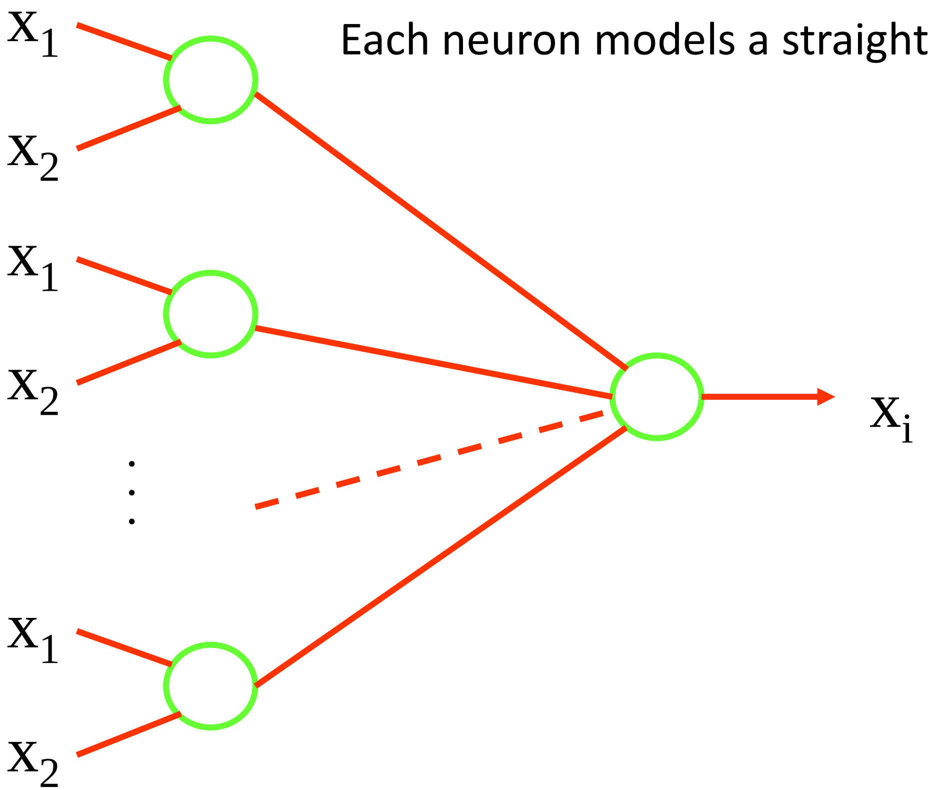
- Since the inner product is positive for $-90^o < \alpha < 90^o$, in this example the neuron's output is 1 for any input vector **x** to the right of or on the dotted line (on the same side of the line as ***w***), and 0 for any other input vector.

What about complex functions?

- Just like Threshold Logic Units for XOR, we can combine multiple artificial neurons in multiple layers to form networks with increased capabilities.
  - For example, in a two-layer network, *n* input neurons in the first layer can send inputs to a single neuron in the second layer.
  - The neuron in the second layer can implement an AND function i.e., output 1 when all are 1.

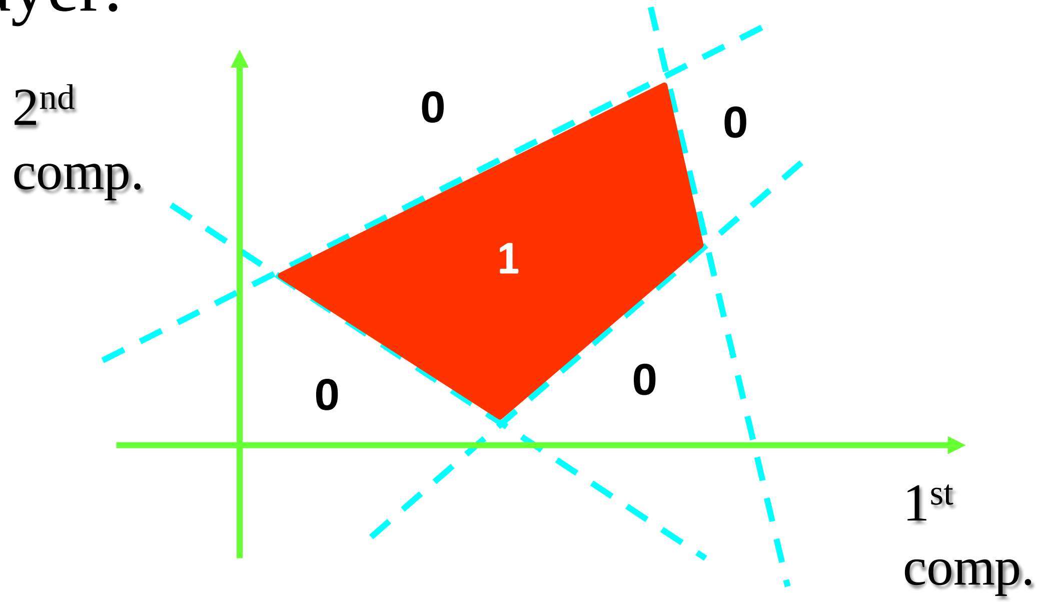
Capabilities (cont…)

- What kind of function can such a network realize?



Capabilities (cont…)

- Assume that the dotted lines in the diagram represent the input-dividing lines implemented by the neurons in the first layer:



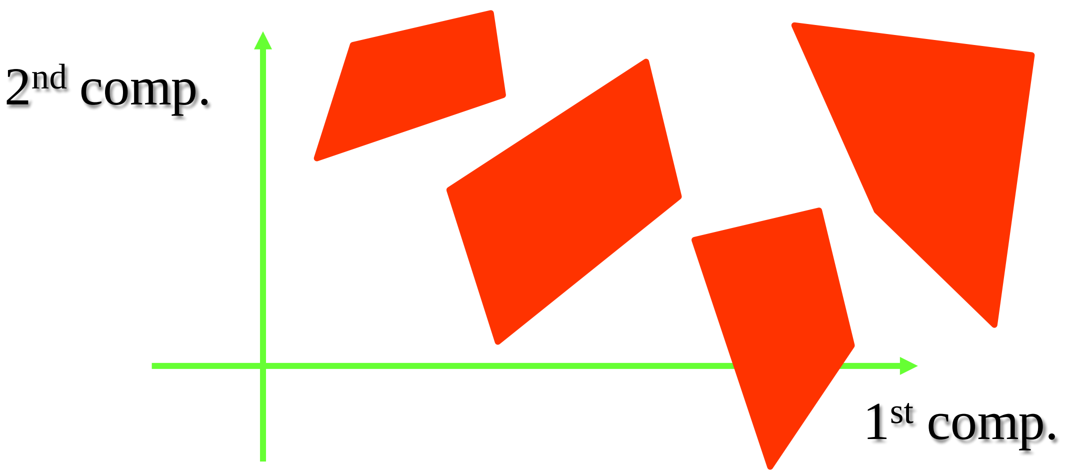
Then, for example, the second-layer neuron could output 1 if the input is within a **polygon**, and 0 otherwise.

Capabilities (cont…)

- The more neurons there are in the first layer, the more vertices can the polygons have.
- With a sufficient number of first-layer neurons, the polygons can approximate any given shape.
- The more neurons there are in the second layer, the more of these polygons can be combined to form the output function of the network.

Capabilities (cont…)

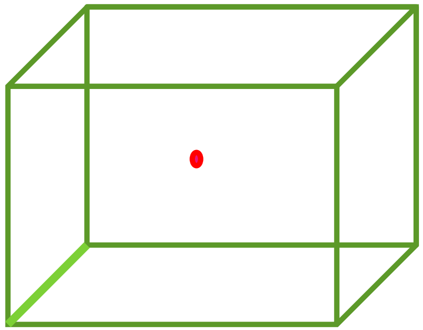
- Assume that the polygons in the diagram indicate the input regions for which each of the **second-layer** neurons yields output 1:



Then, for example, **the third-layer** neuron could output 1 if the input is within **any of the polygons**, and 0 otherwise. Example application ???

Problem

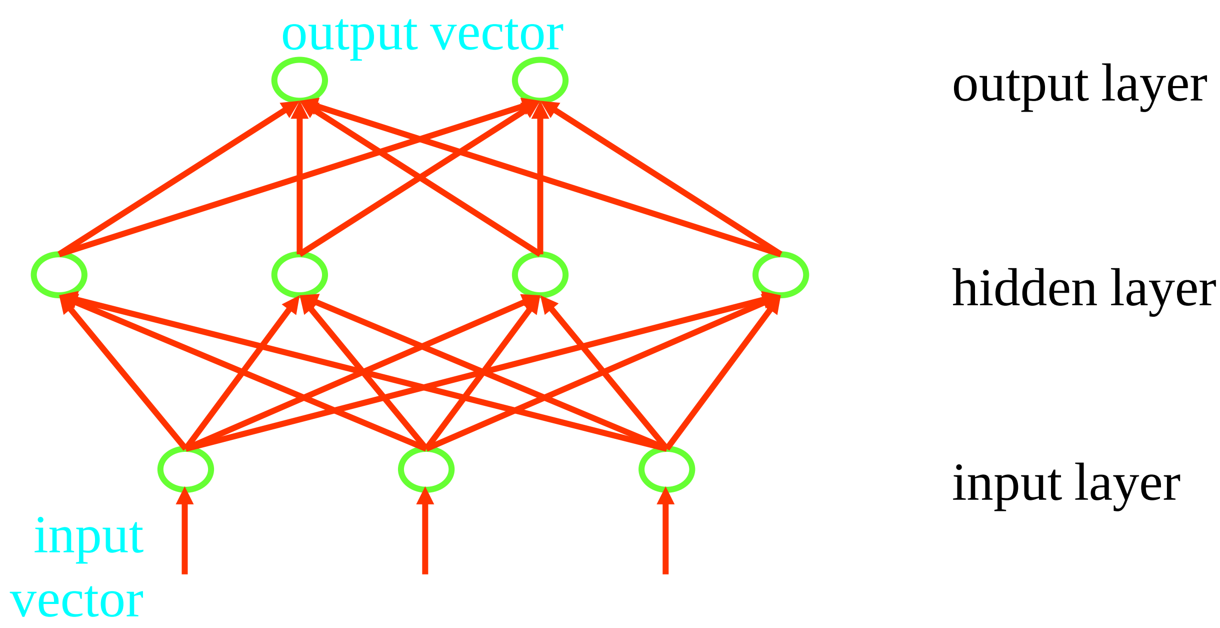
1. Draw an ANN to find if a point lies **inside** a hollow box. Explain how it would work.



2. Draw an ANN that can detect whether a point lies on the **surface** of a pyramid. Explain how it would work.

Terminology

- Example: Network function f: $R^3 \rightarrow \{0, 1\}^2$ means that the network takes 3-dimensional real values as input and generates two dimensional binary output values.



Terminology

- Usually, we draw neural networks in such a way that the input enters at the bottom and the output is generated at the top.
- Arrows indicate the direction of data flow.
- The *input layer* just contains the input vector and *does not perform any computations other than distributing inputs to the next layers (used optionally)*.
- The intermediate layers, termed *hidden layers*, receives input from the input layer or previous hidden layers and *sends output to the final output layer*.
- After applying their *activation function*, the neurons in the final *output layer* generate the output vector.