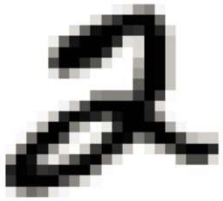
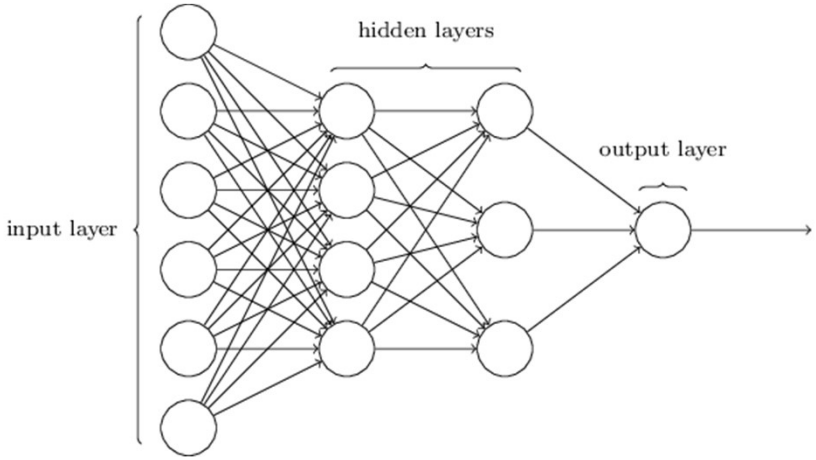
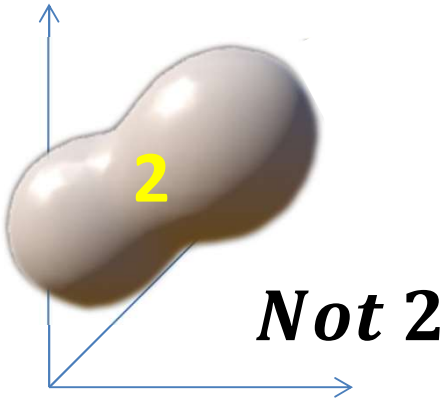
# The MLP as a classifier



784 dimensions
(MNIST)

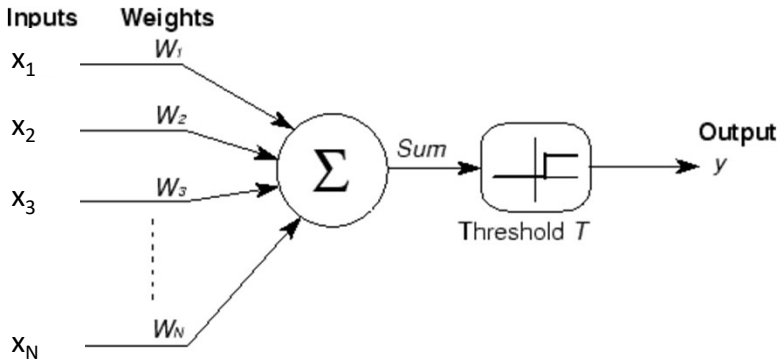


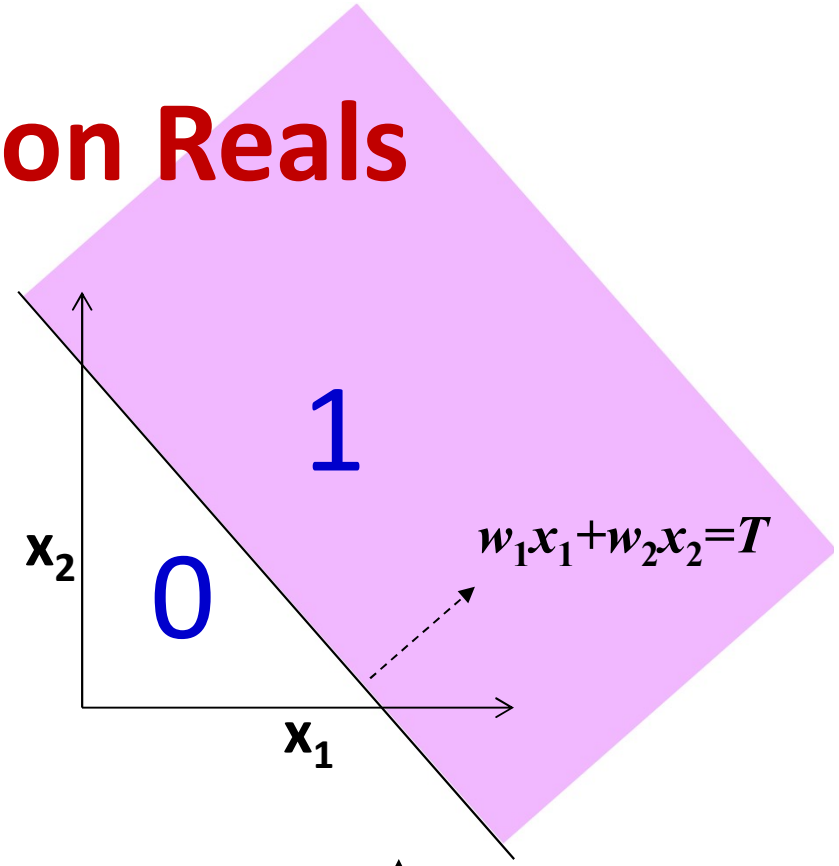

784 dimensions

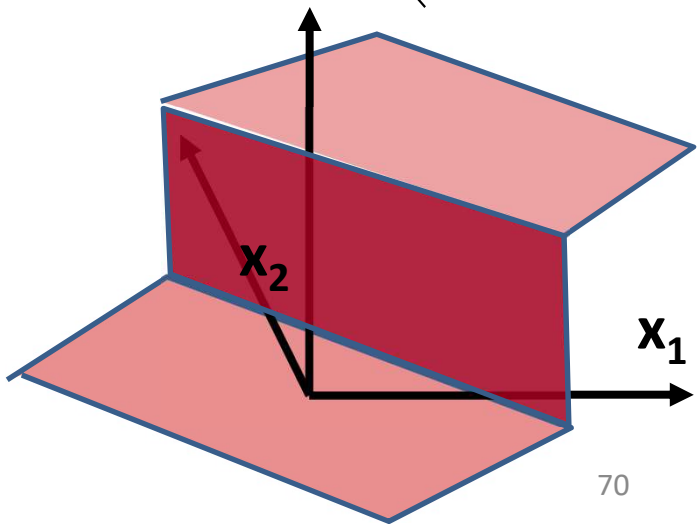
- MLP as a function over real inputs
- MLP as a function that finds a complex “decision boundary” over a space of *reals*

# A Perceptron on Reals



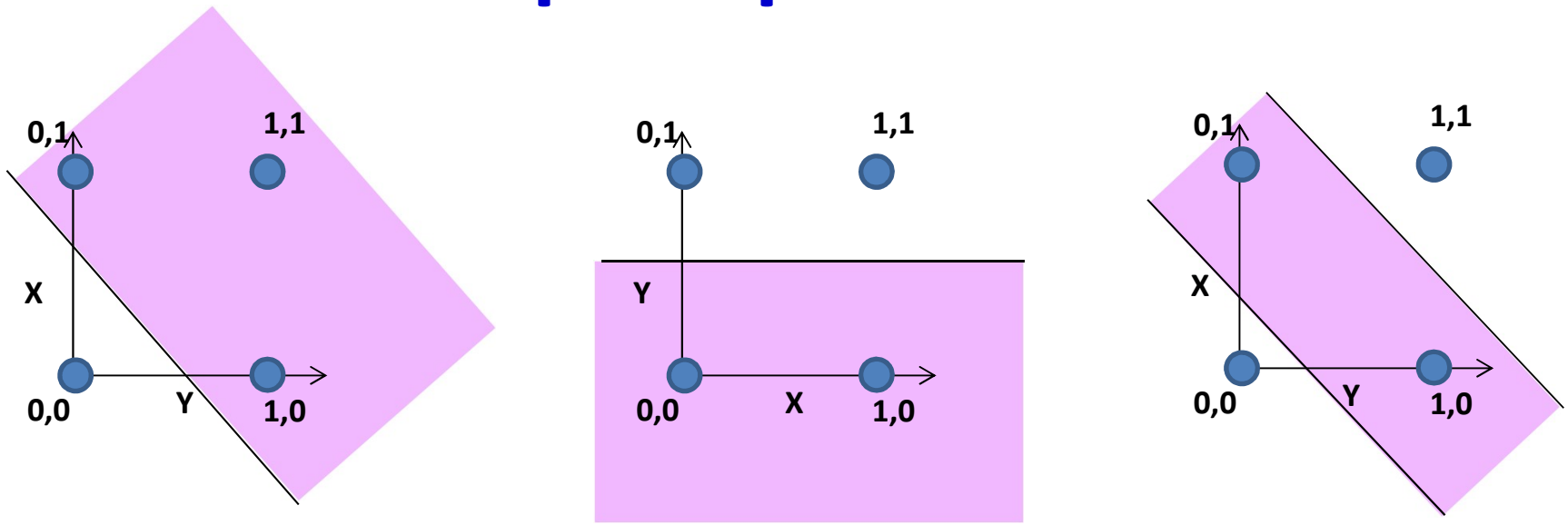

$$y = \begin{cases} 1 \text{ if } \sum_i w_i x_i \geq T \\ 0 \text{ else} \end{cases}$$



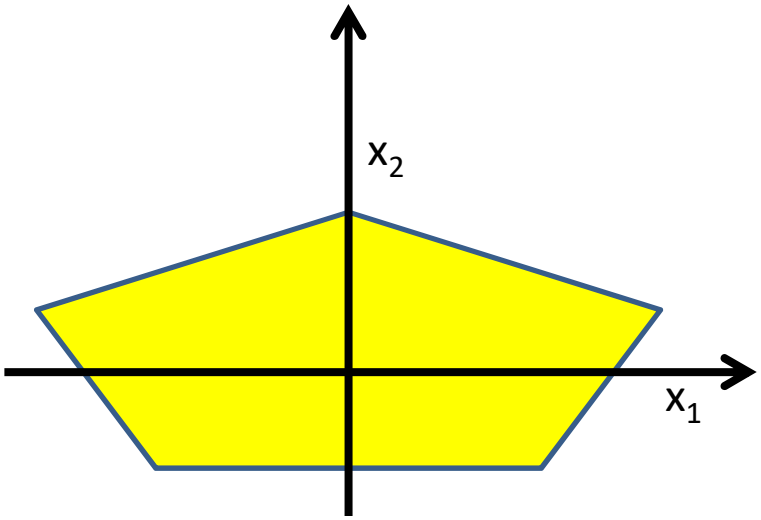

- A perceptron operates on *real*-valued vectors
  - **This is a *linear classifier***

# Boolean functions with a real perceptron




- Boolean perceptrons are also linear classifiers
  - Purple regions are 1

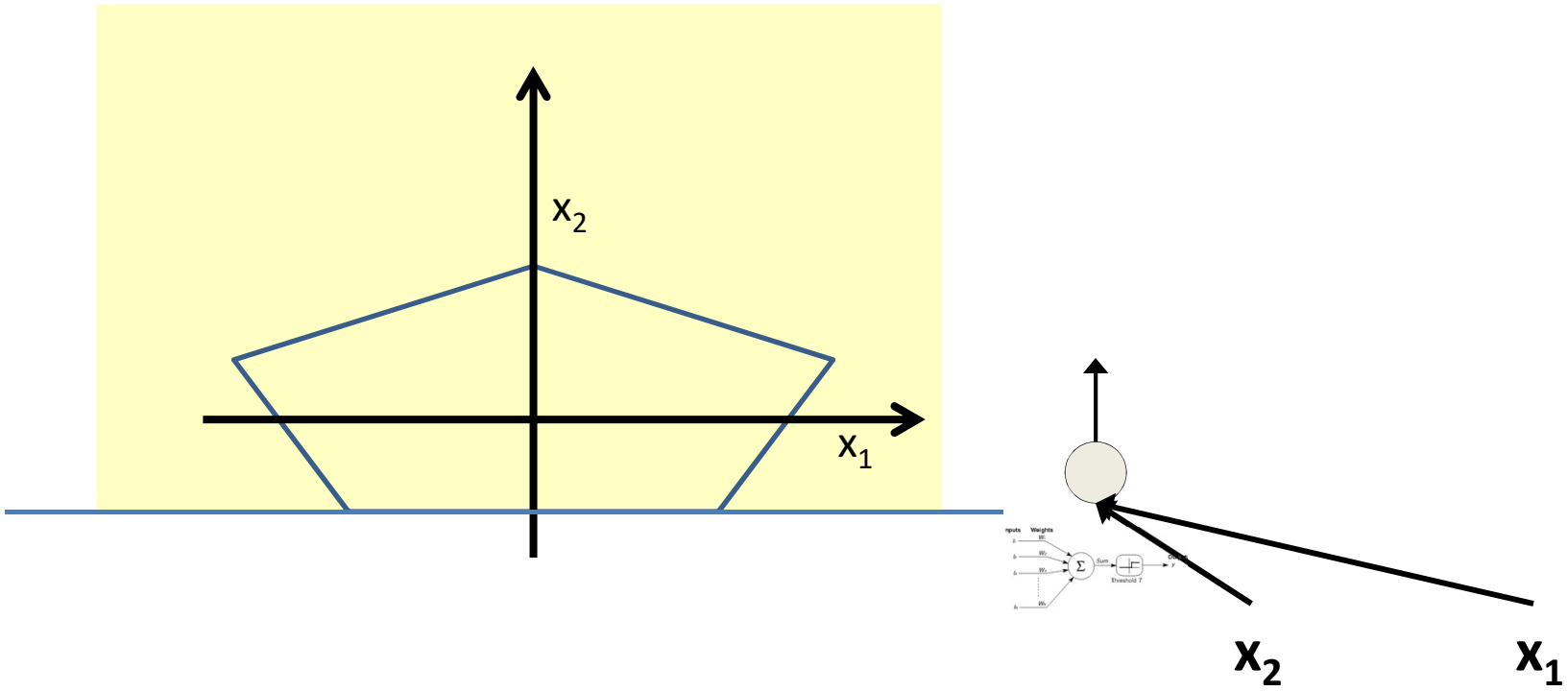
# Composing complicated "decision" boundaries




Can now be composed into "networks" to compute arbitrary classification "boundaries"

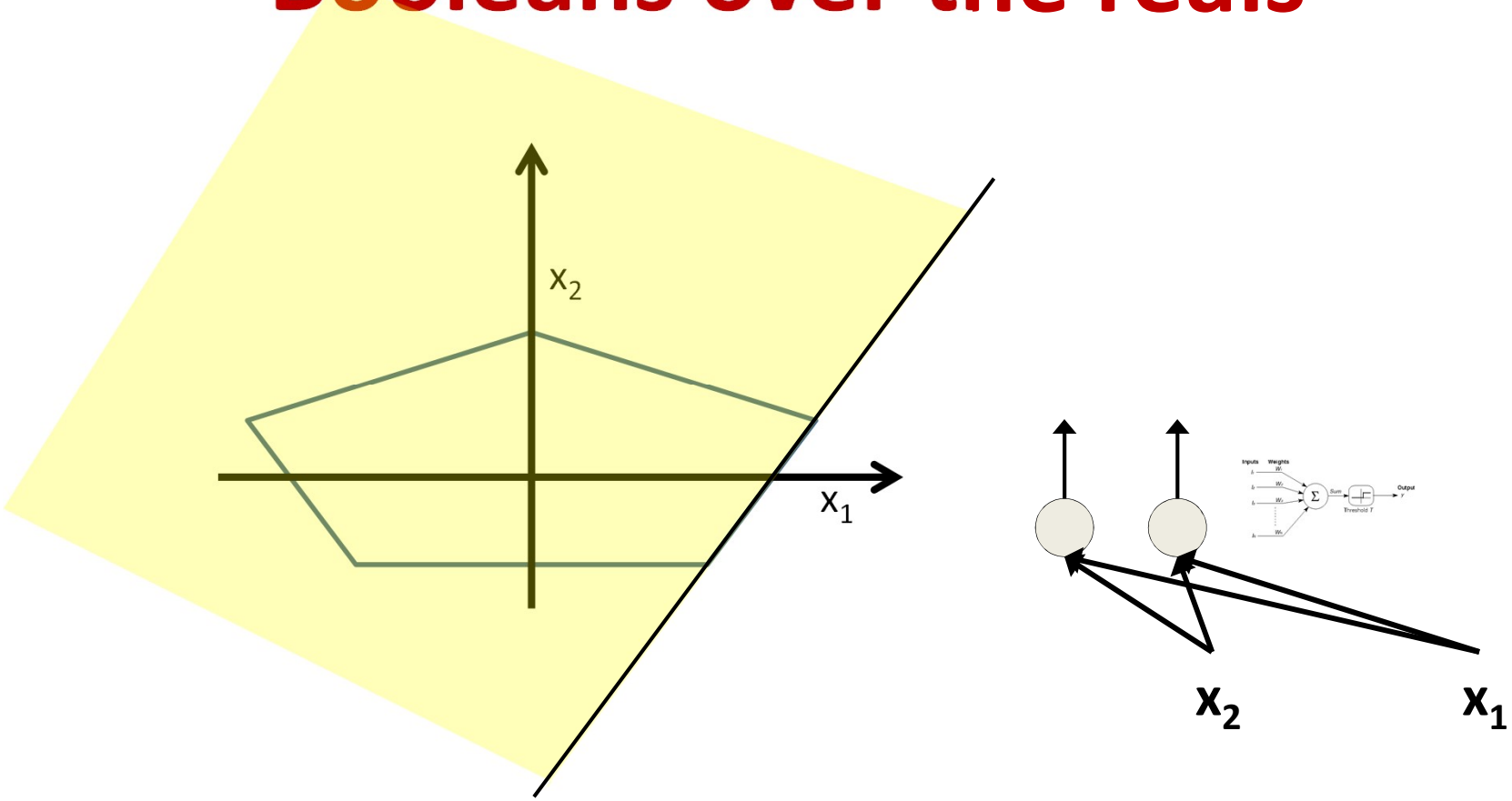
- Build a network of units with a single output that fires if the input is in the coloured area

# Booleans over the reals



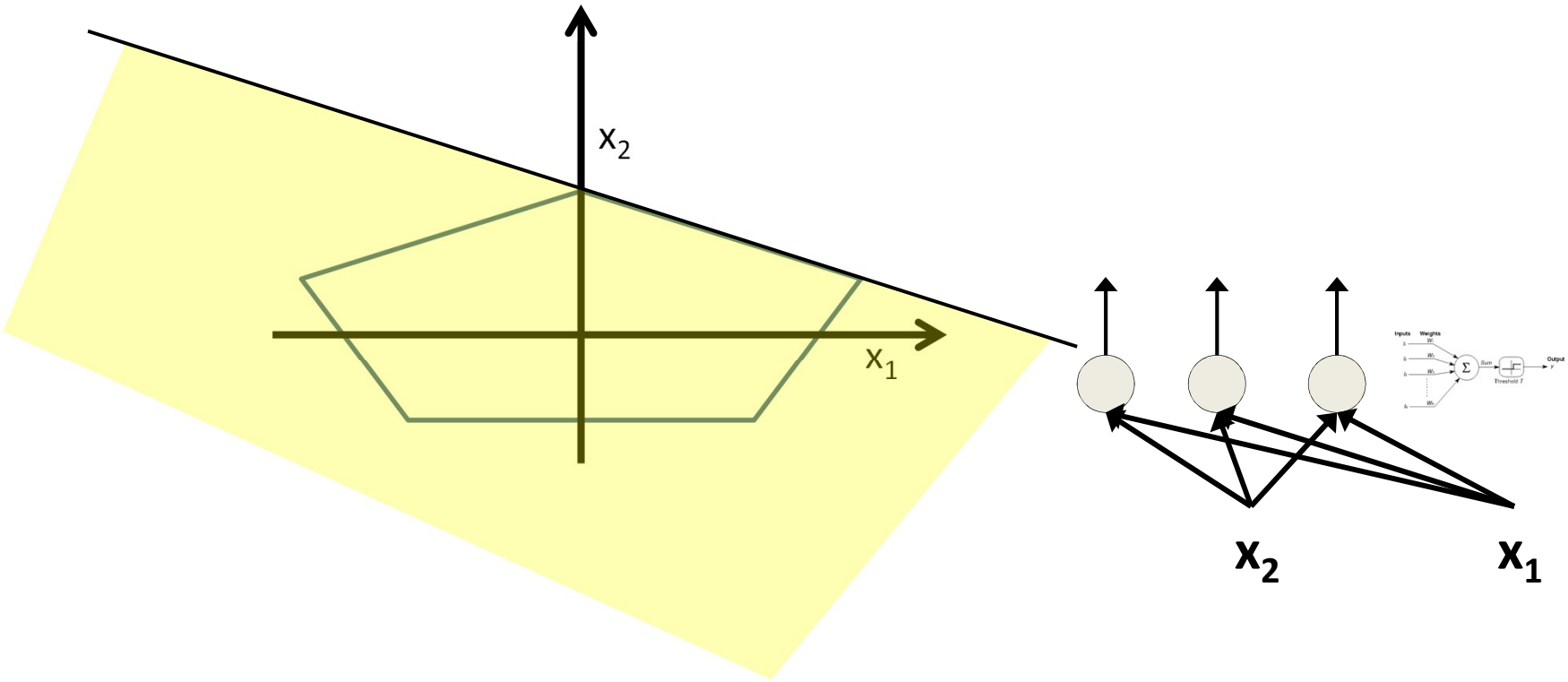

- The network must fire if the input is in the coloured area

# Booleans over the reals



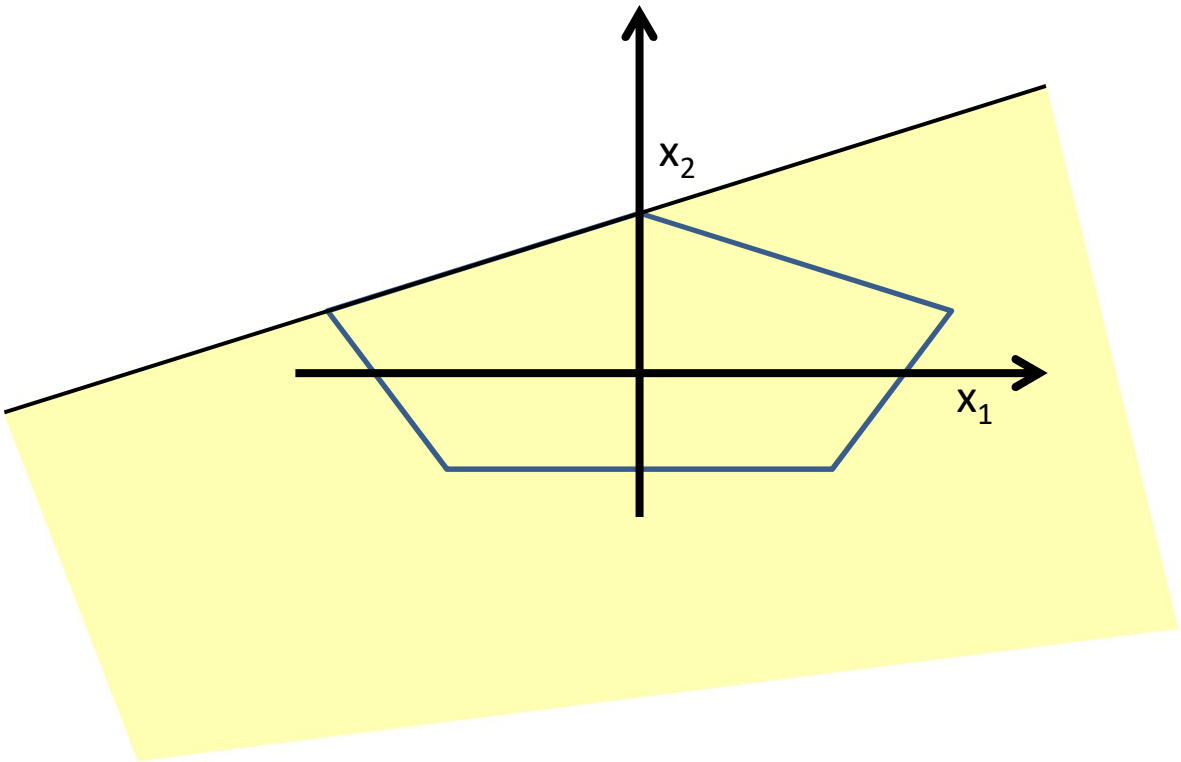

- The network must fire if the input is in the coloured area

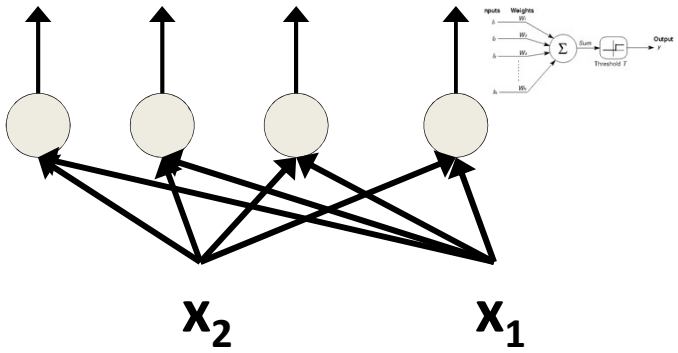
# Booleans over the reals




- The network must fire if the input is in the coloured area

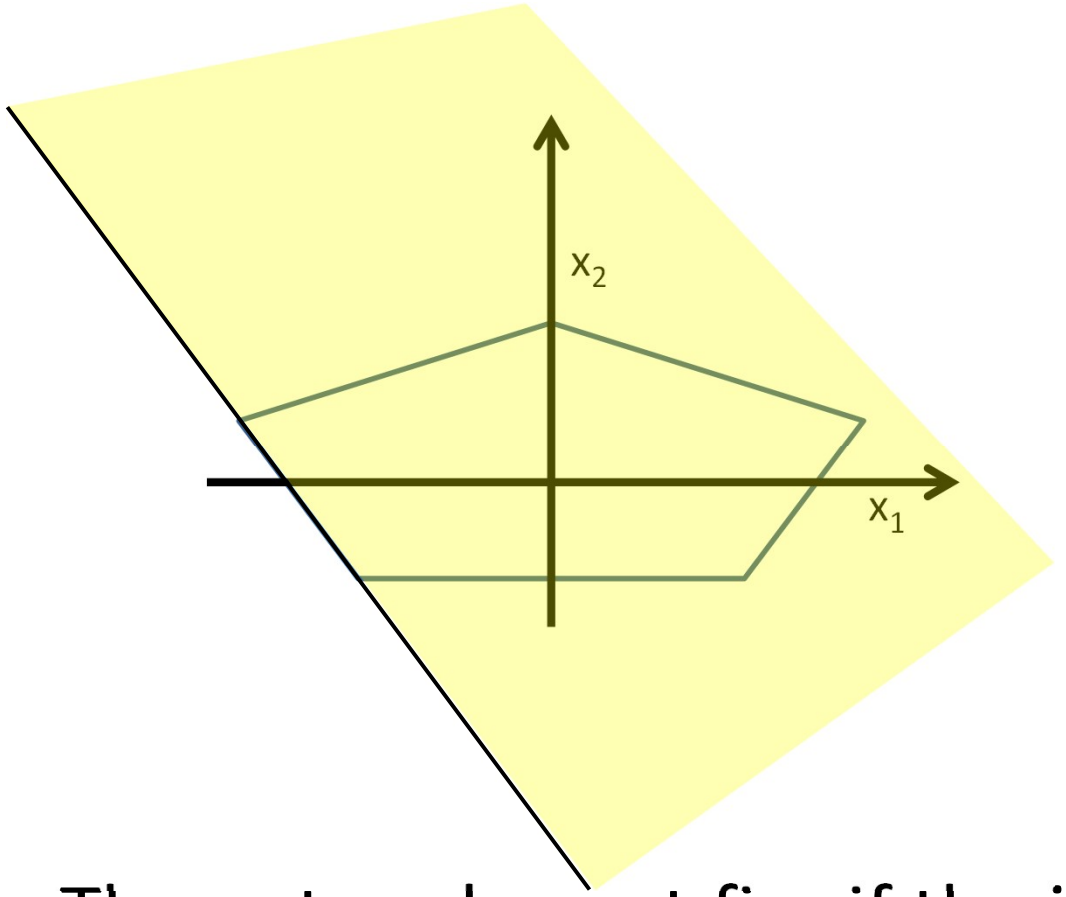
# Booleans over the reals

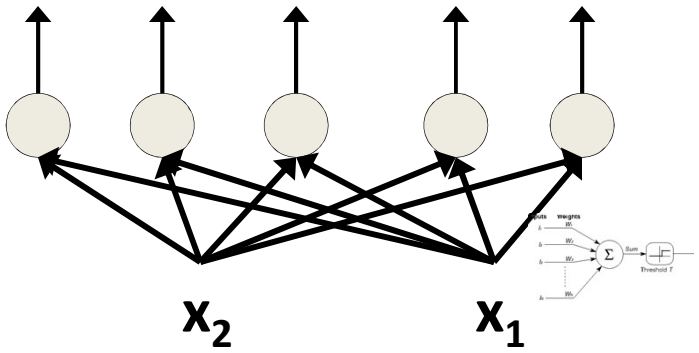



- The network must fire if the input is in the coloured area

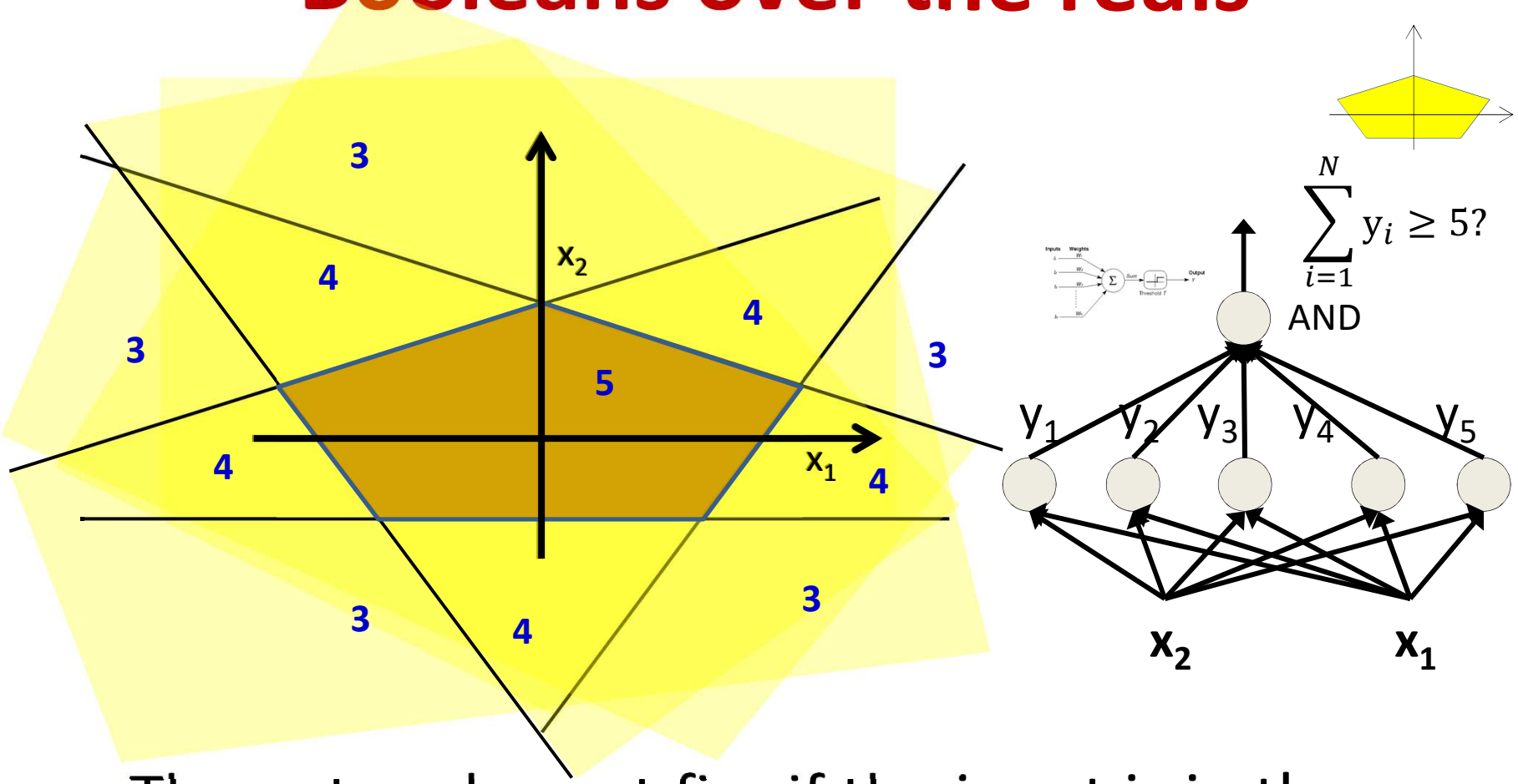
# Booleans over the reals



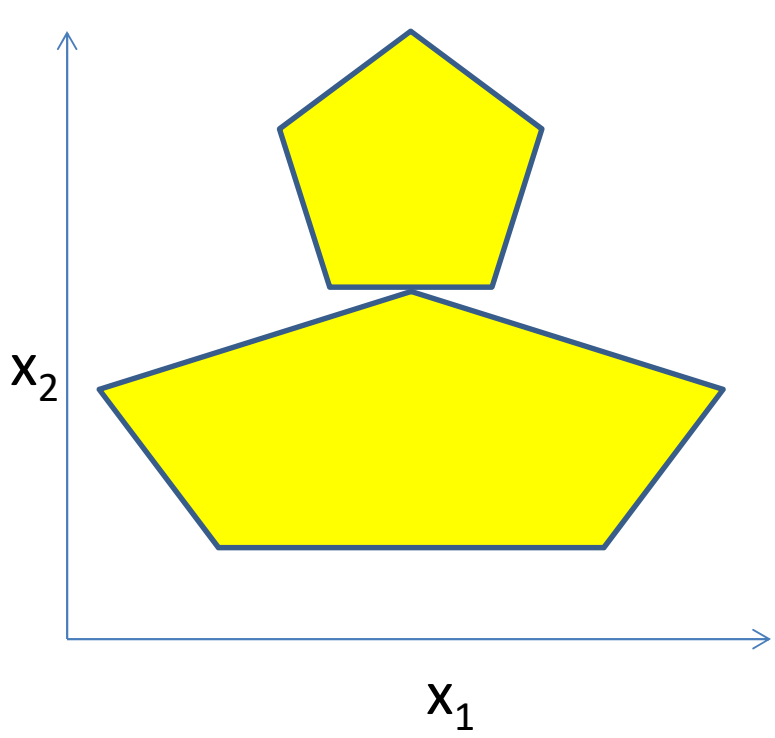

- The network must fire if the input is in the coloured area

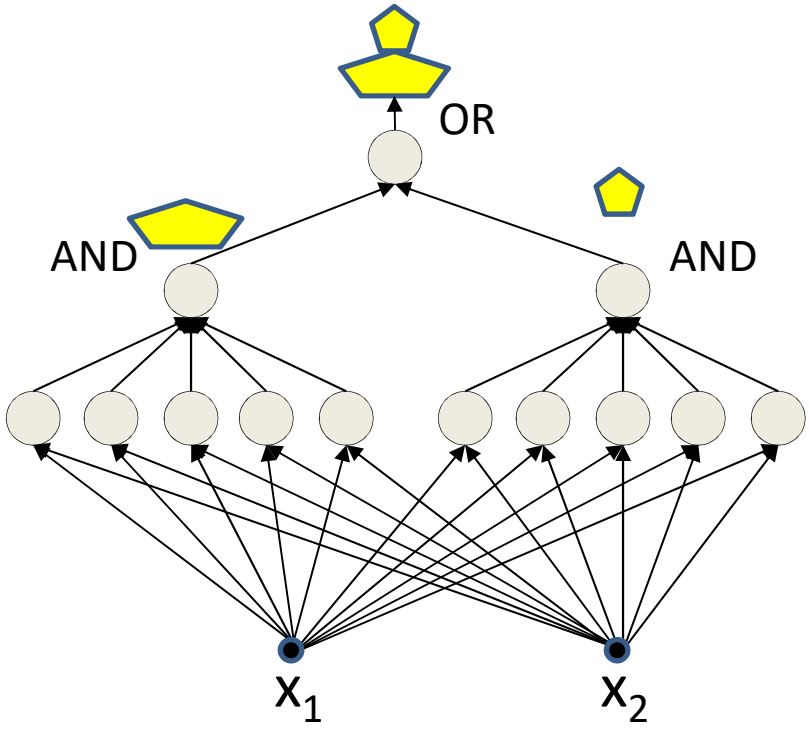
# Booleans over the reals




- The network must fire if the input is in the coloured area

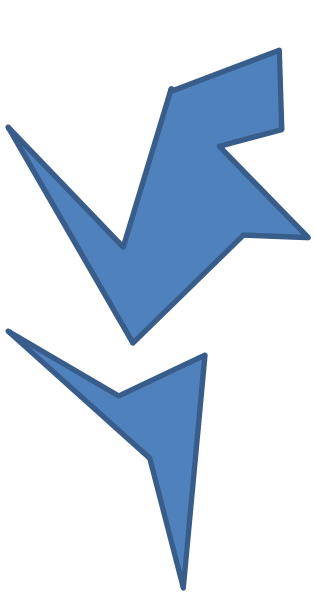
# More complex decision boundaries

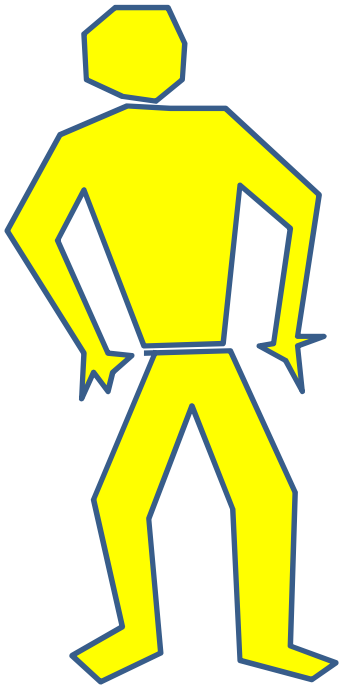



- Network to fire if the input is in the yellow area
  - "OR" two polygons
  - A third layer is required

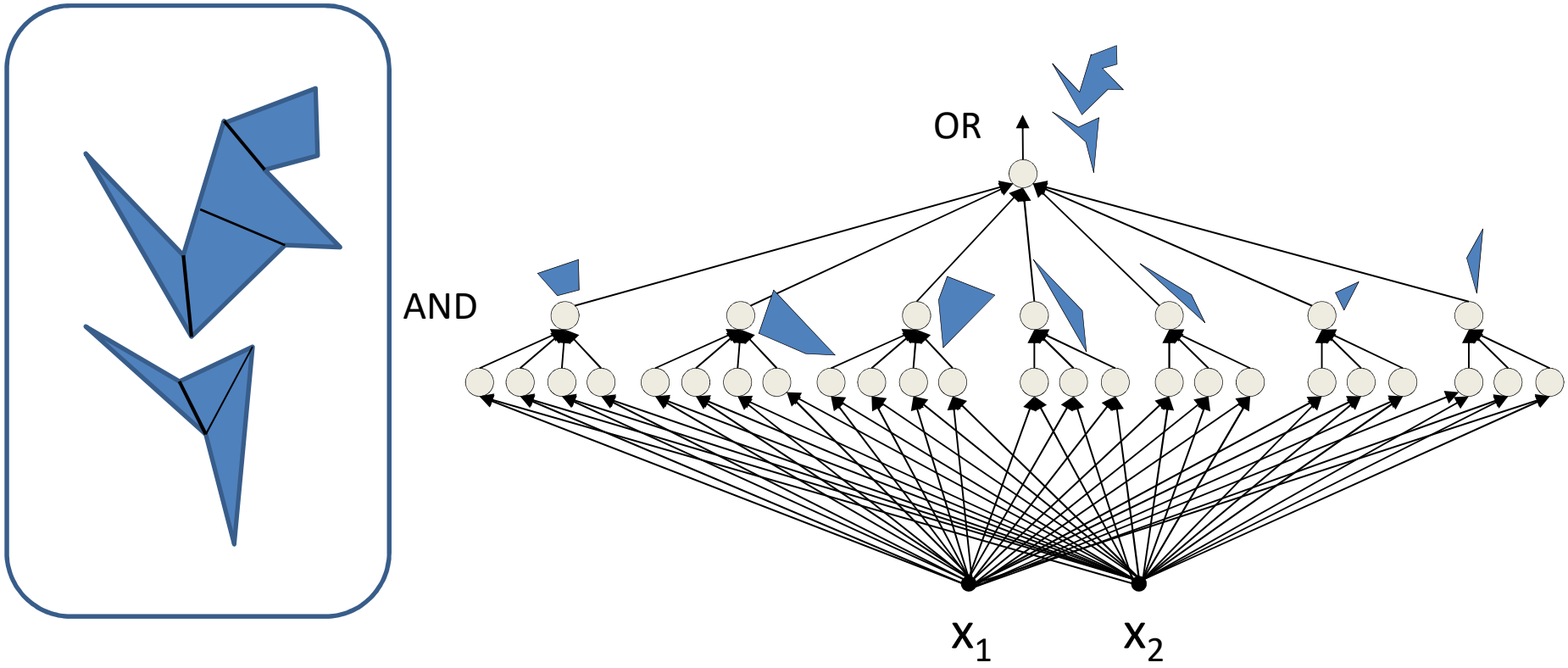
# Complex decision boundaries





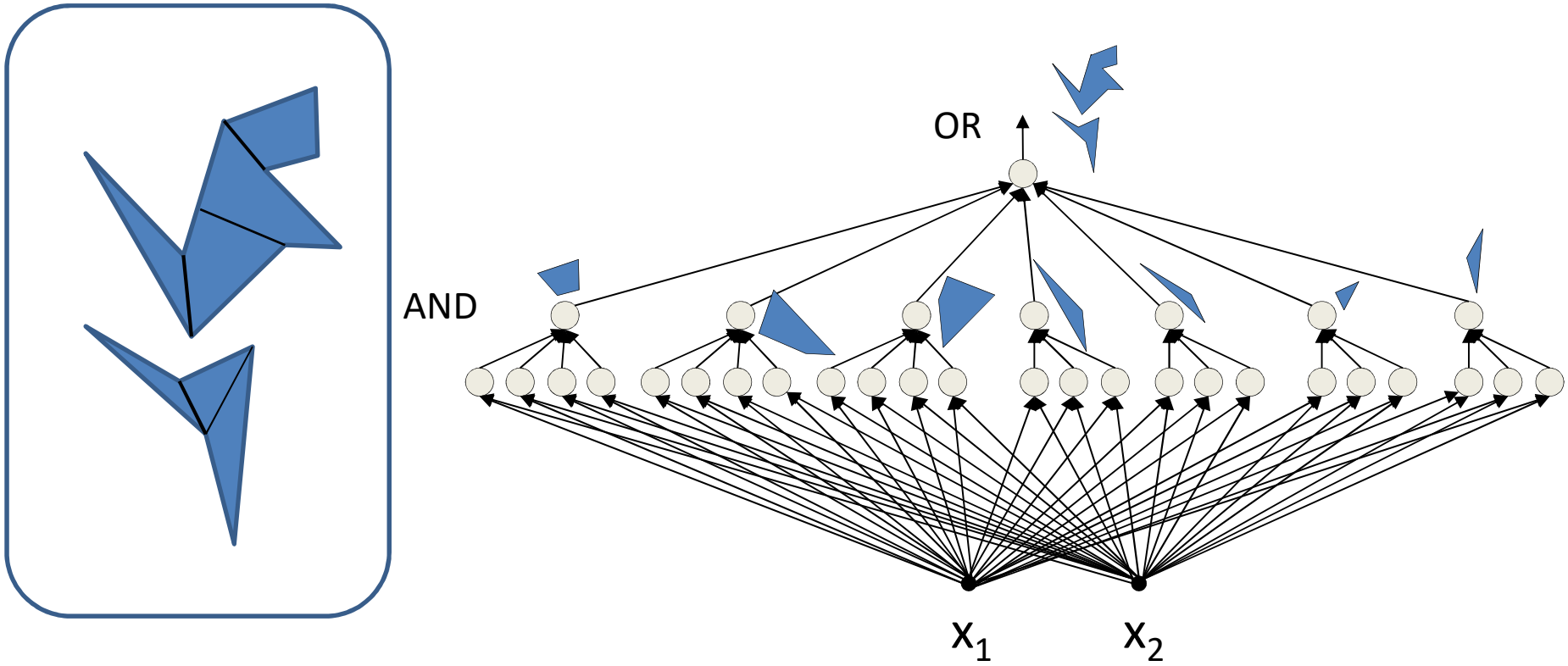
- Can compose *arbitrarily* complex decision boundaries

# Complex decision boundaries



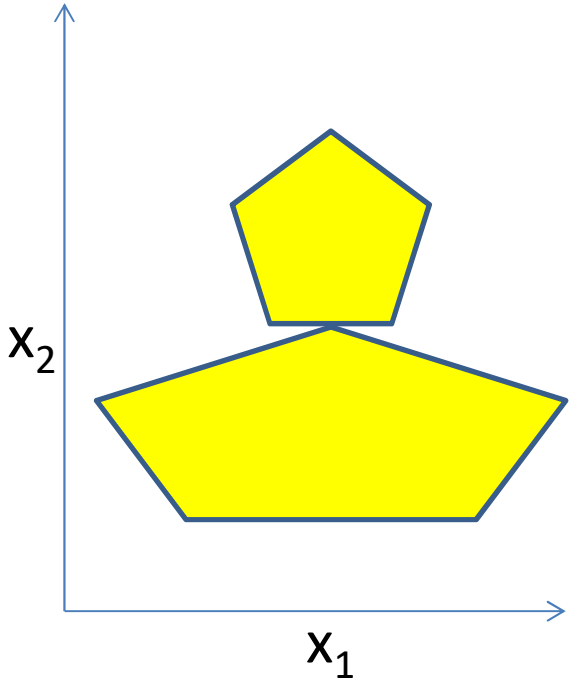

- Can compose *arbitrarily* complex decision boundaries

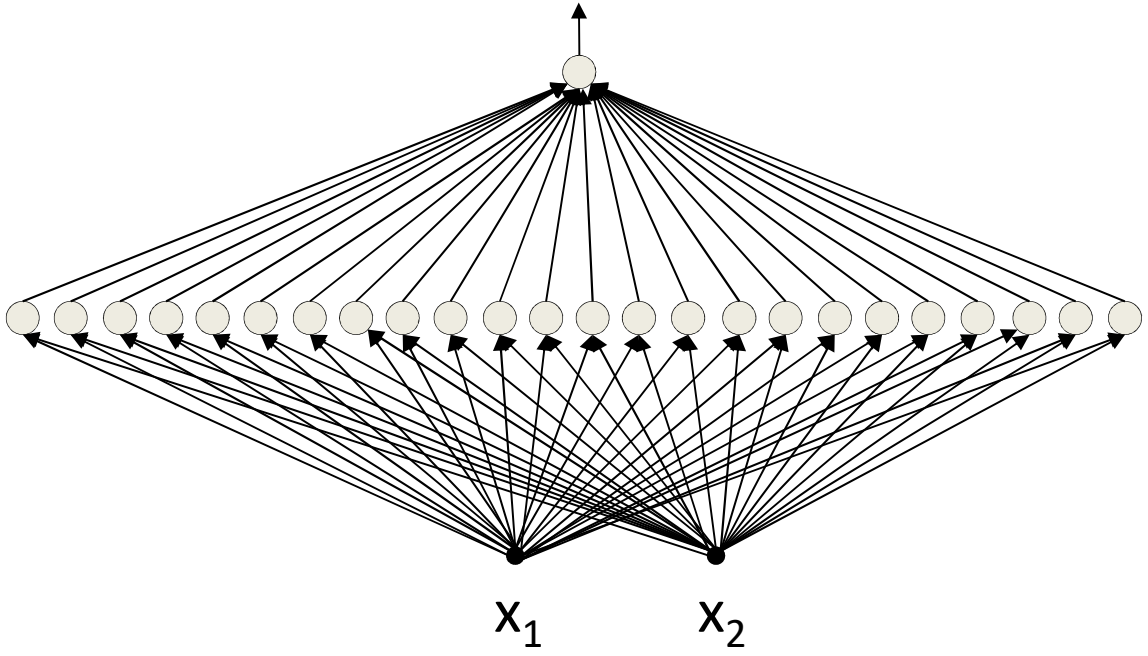
# Complex decision boundaries




- Can compose *arbitrarily* complex decision boundaries
  - With *only one hidden layer!*
  - **How**?

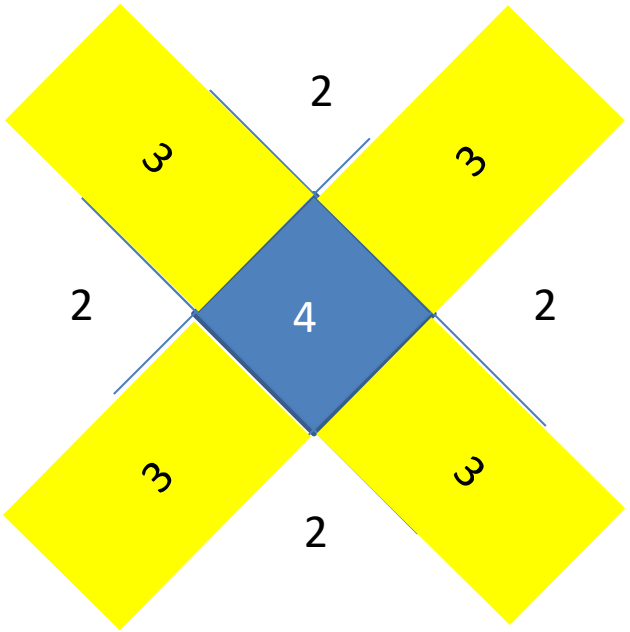
# Exercise: compose this with one hidden layer

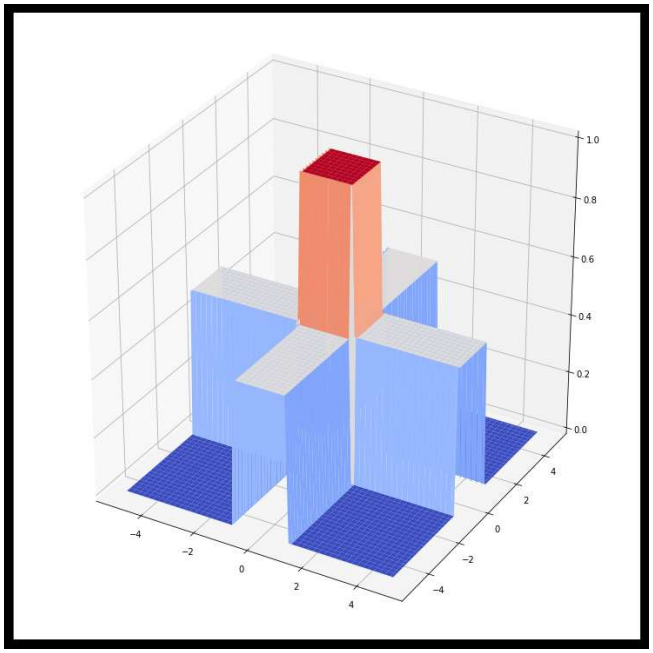


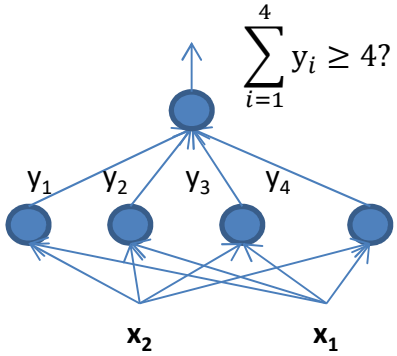

- How would you compose the decision boundary to the left with only *one* hidden layer?

# Composing a Square decision boundary

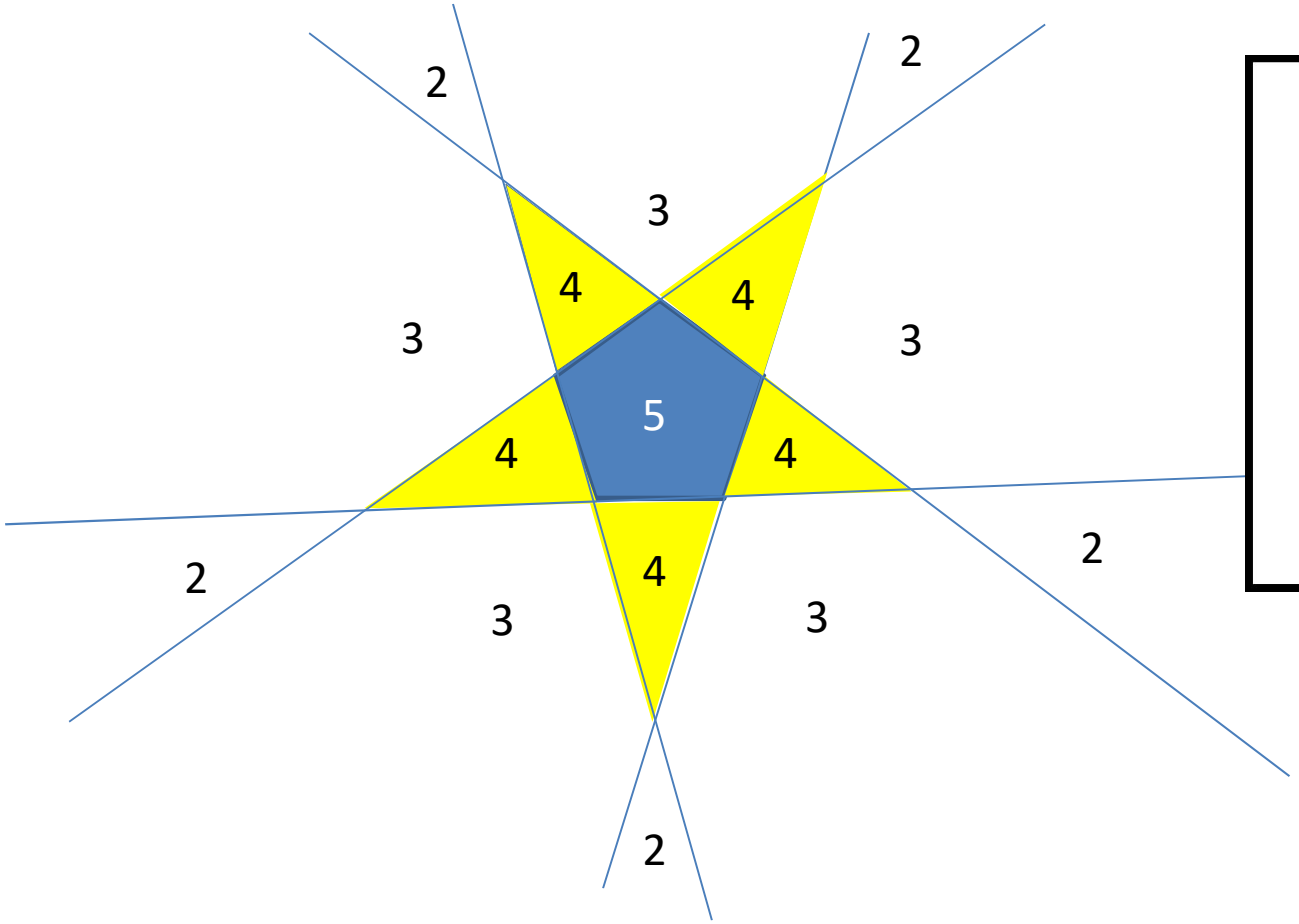


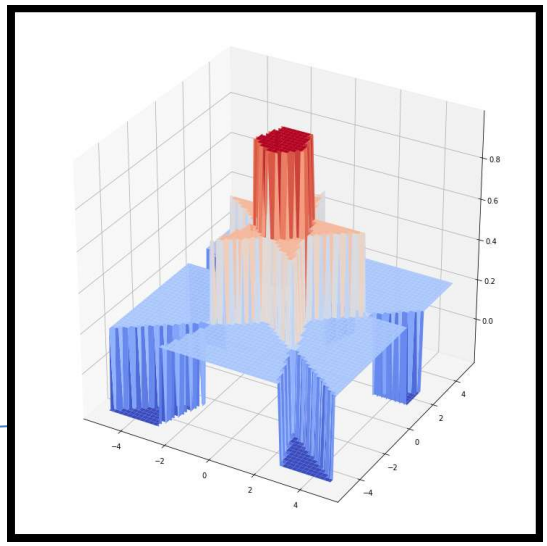



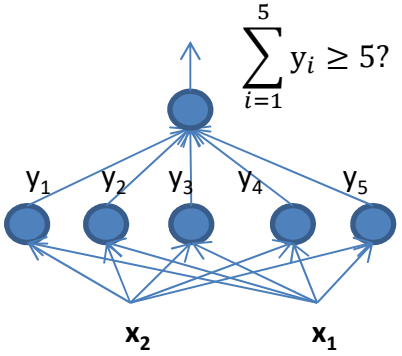
- The polygon net



# Composing a pentagon

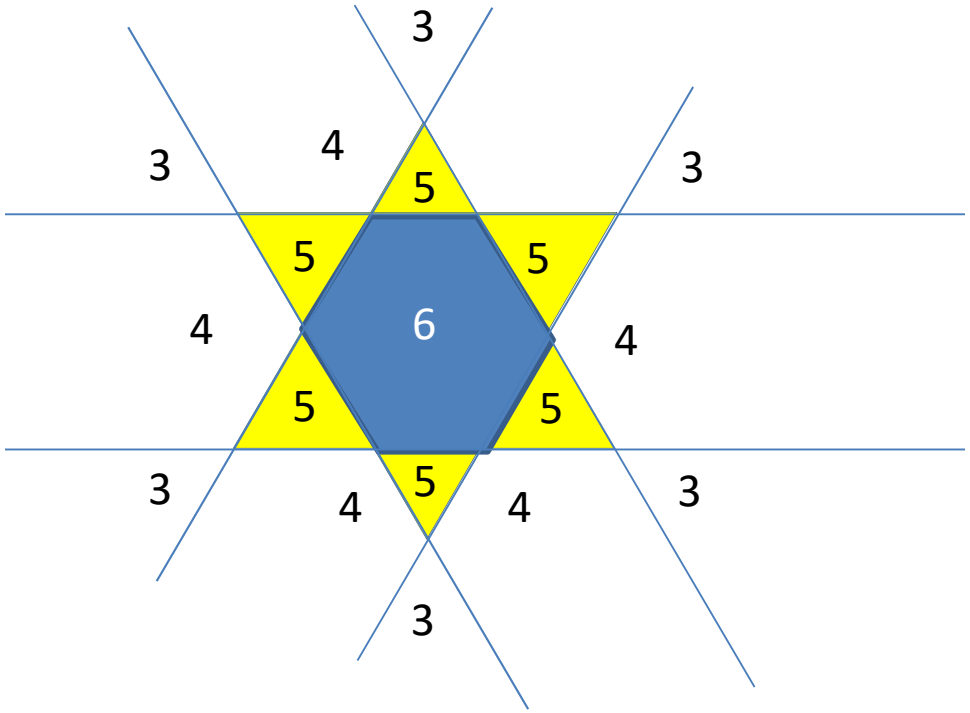


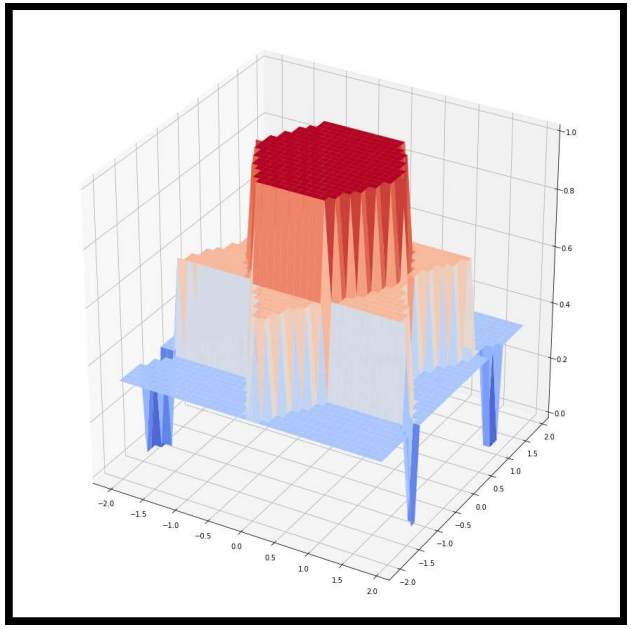





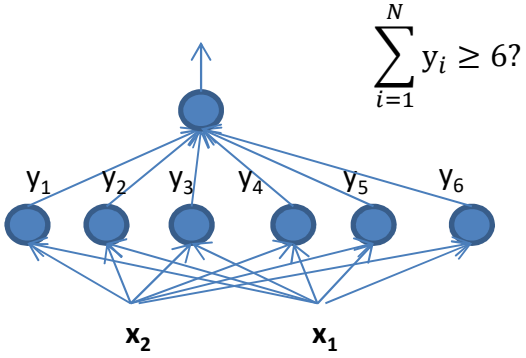

- The polygon net

# Composing a hexagon

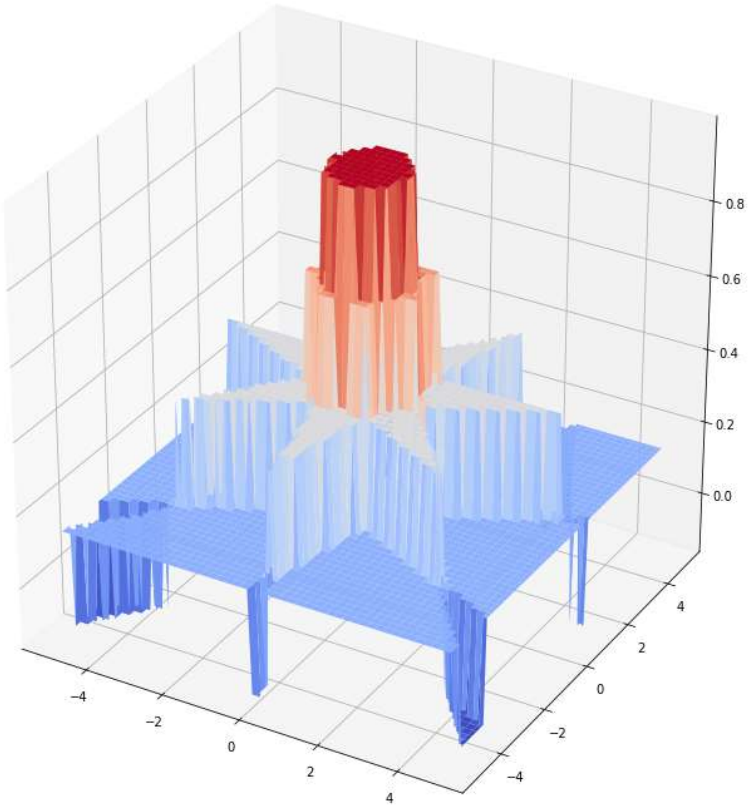


- The polygon net

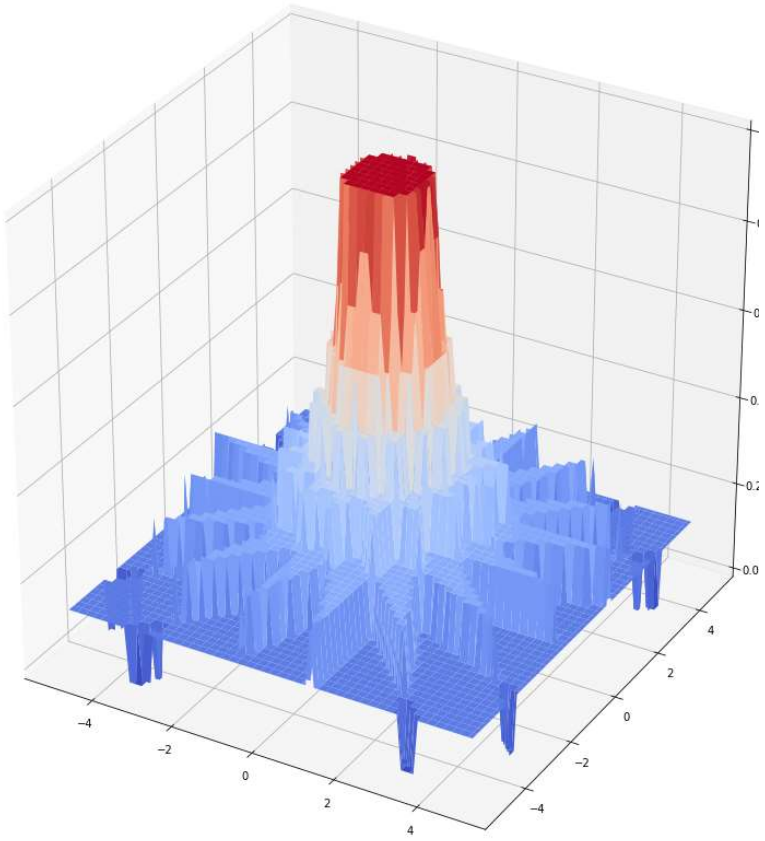


# How about a heptagon



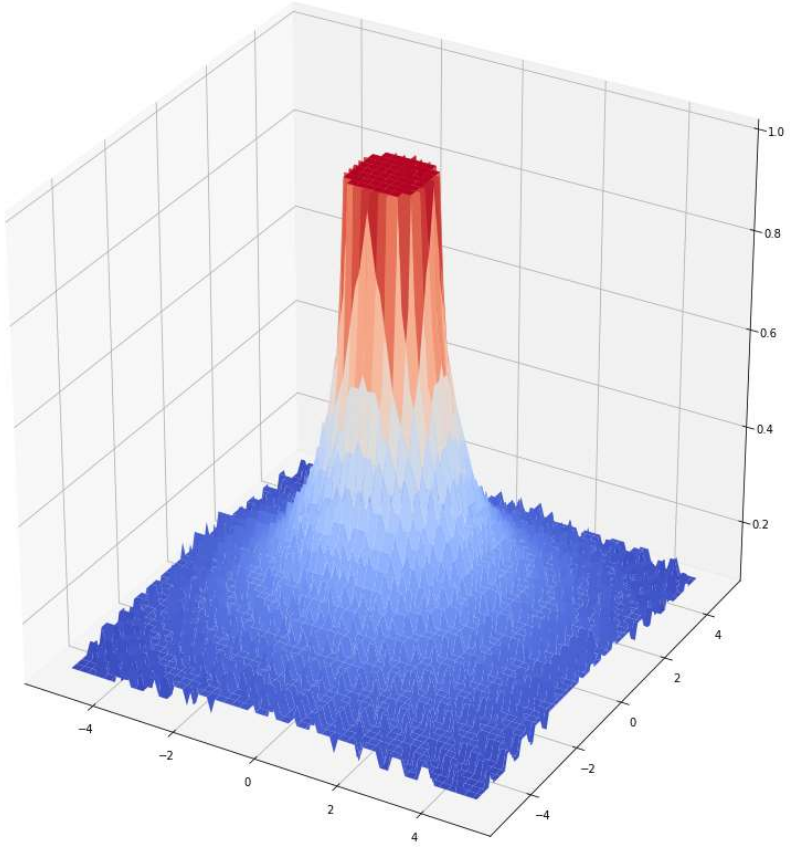

- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

# 16 sides



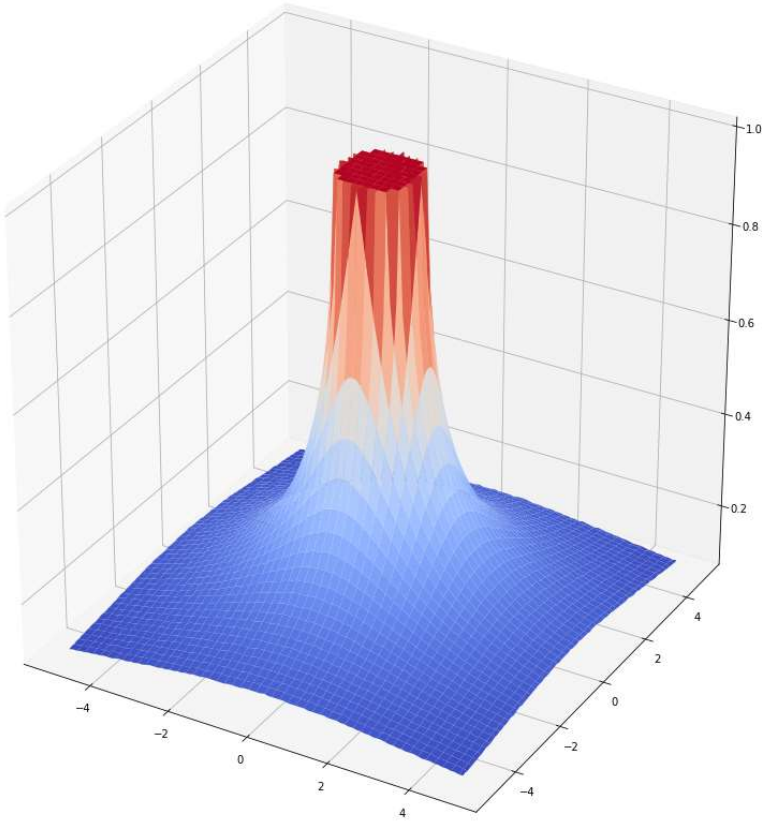

- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

# 64 sides



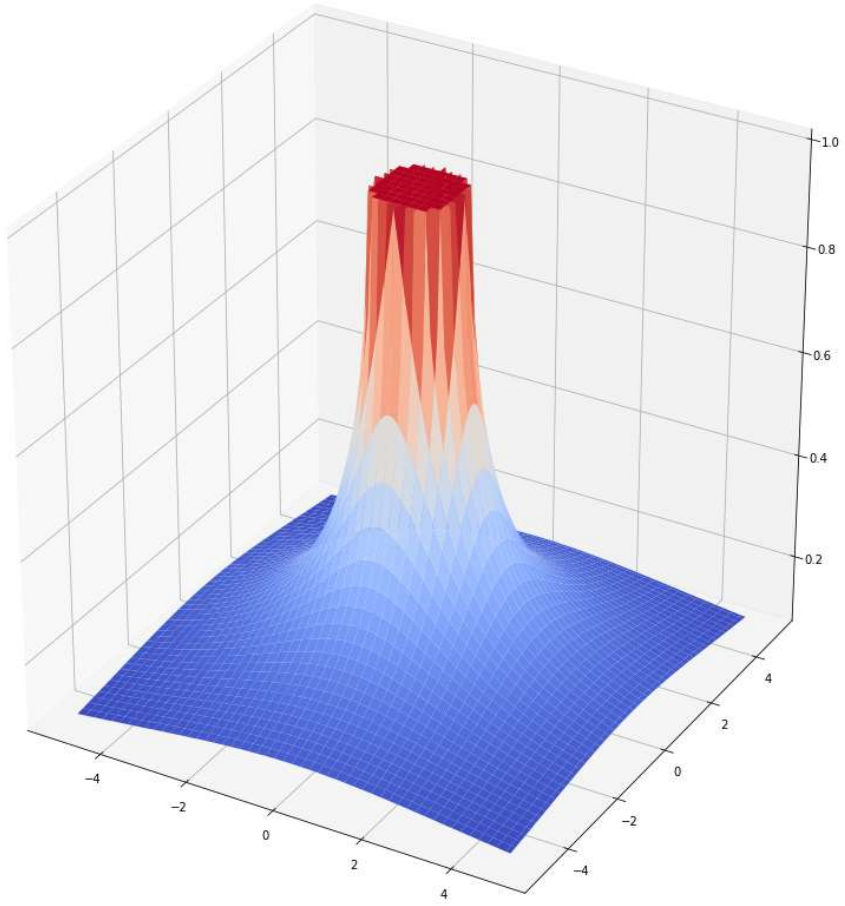

- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

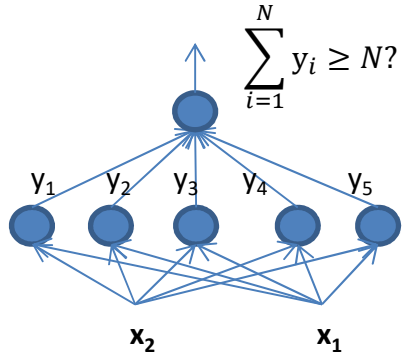
# 1000 sides




- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

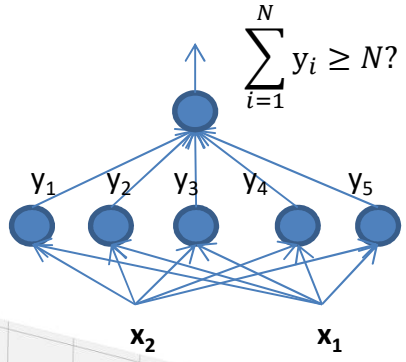
# Polygon net

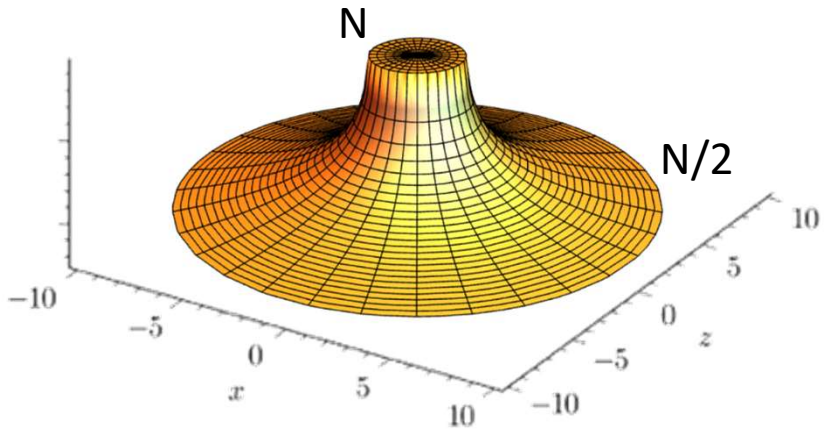


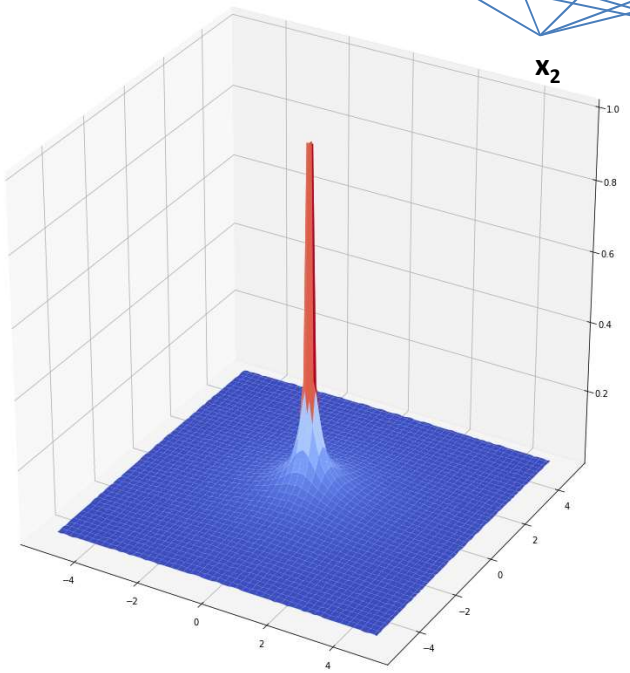




- Increasing the number of sides reduces the area outside the polygon that have N/2 < Sum < N

# In the limit

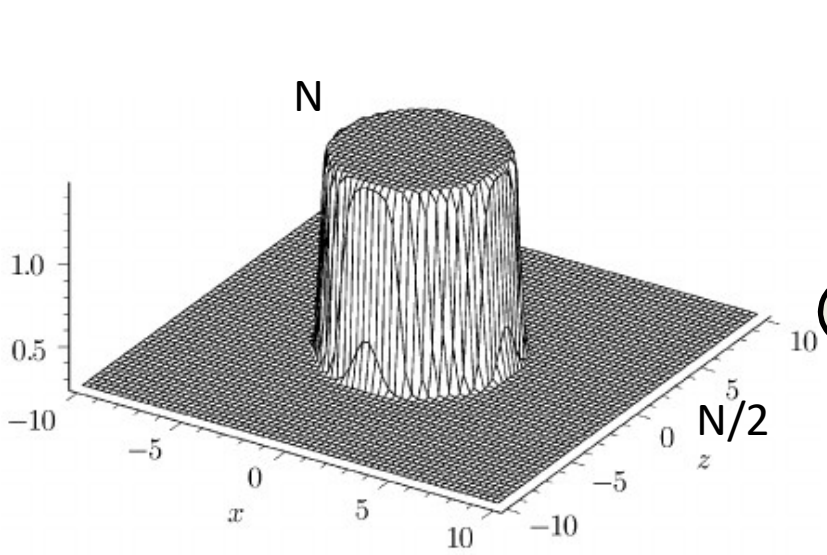


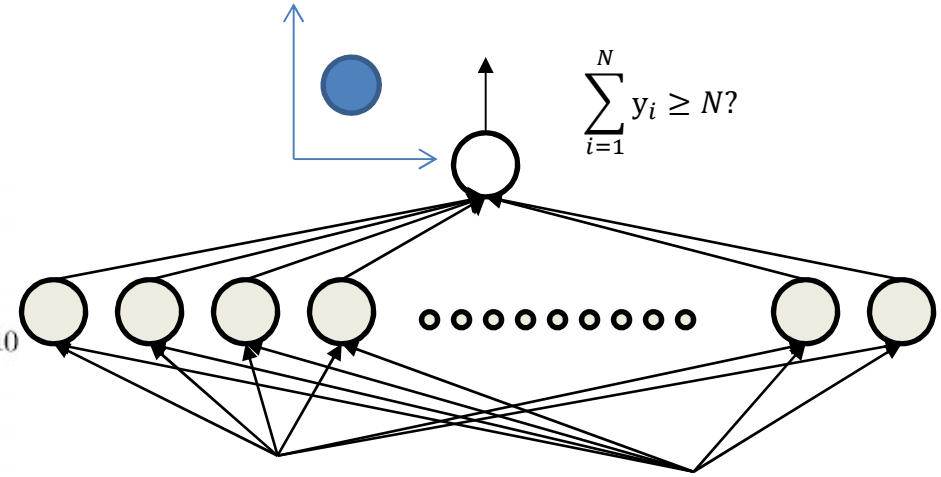



- $\sum_i y_i = N\left(1 - \frac{1}{\pi} arccos\left(min\left(1, \frac{radius}{|\mathbf{x} - cent\quad|}\right)\right)\right)$
- For small radius, it's a near perfect cylinder
  - N in the cylinder, N/2 outside

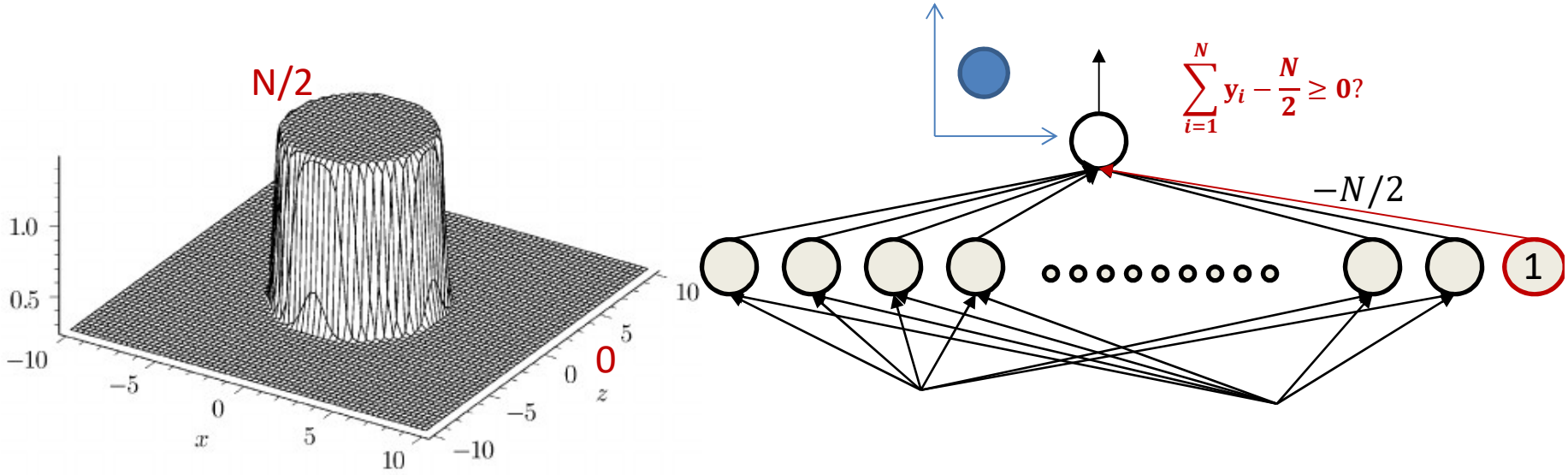
# Composing a circle



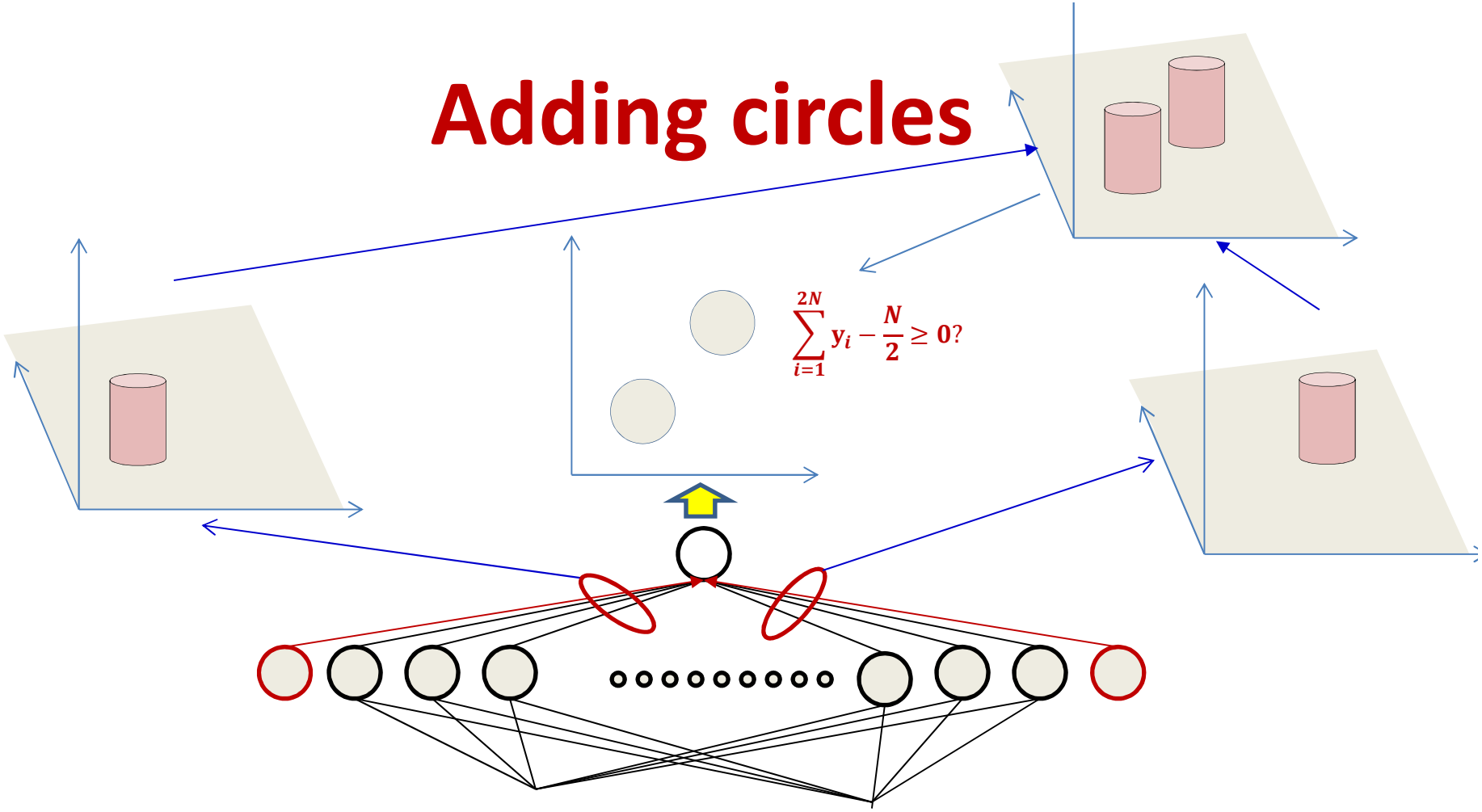

- The circle net
  - Very large number of neurons
  - *Sum is N inside the circle, N/2 outside almost everywhere*
  - Circle can be at any location

# Composing a circle



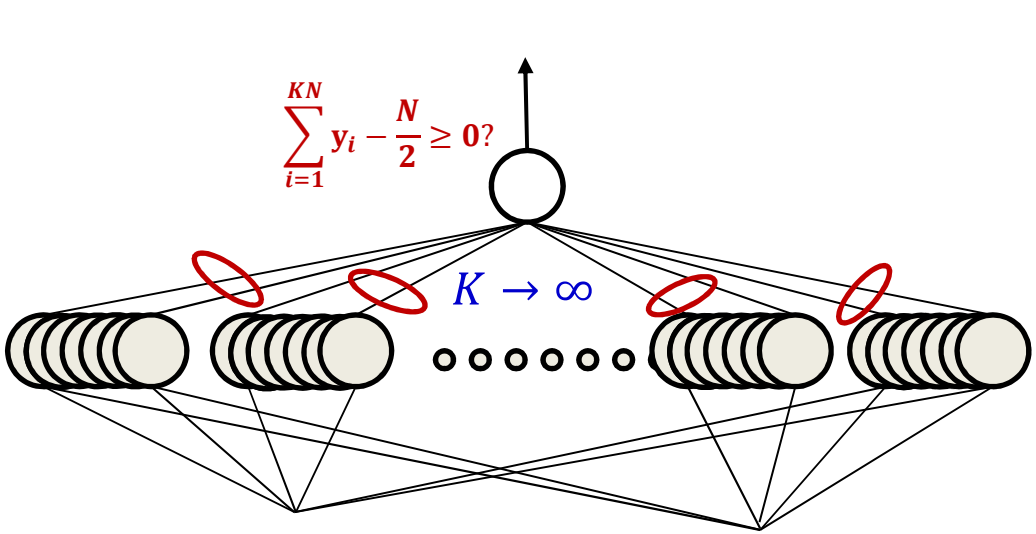

- The circle net
  - Very large number of neurons
  - *Sum is N/2 inside the circle, 0 outside almost everywhere*
  - Circle can be at any location

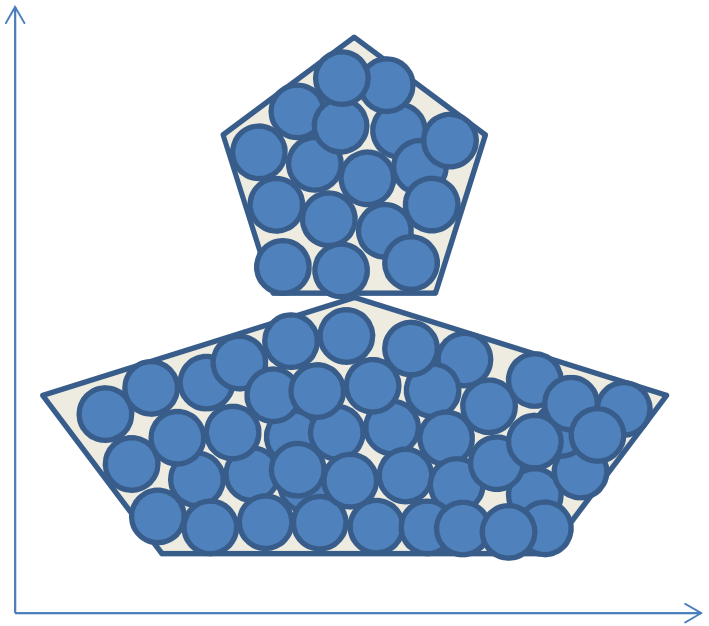
# Adding circles




- The “sum” of two circles sub nets is exactly N/2 inside either circle, and 0 almost everywhere outside

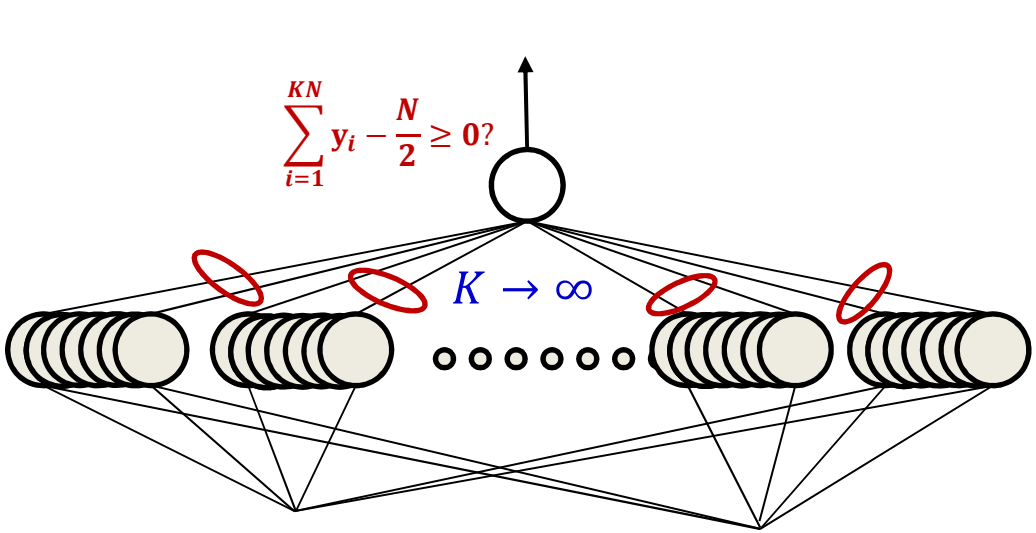
# Composing an arbitrary figure

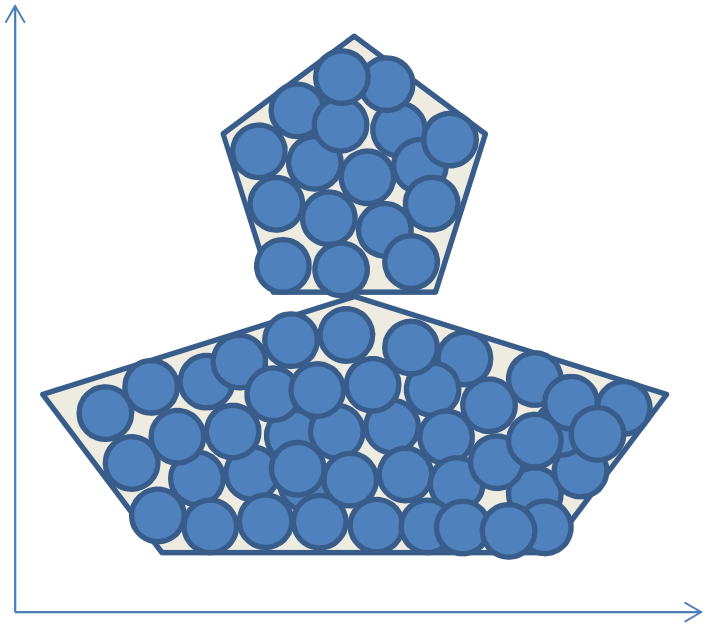


- Just fit in an arbitrary number of circles
  - More accurate approximation with greater number of smaller circles
  - Can achieve arbitrary precision

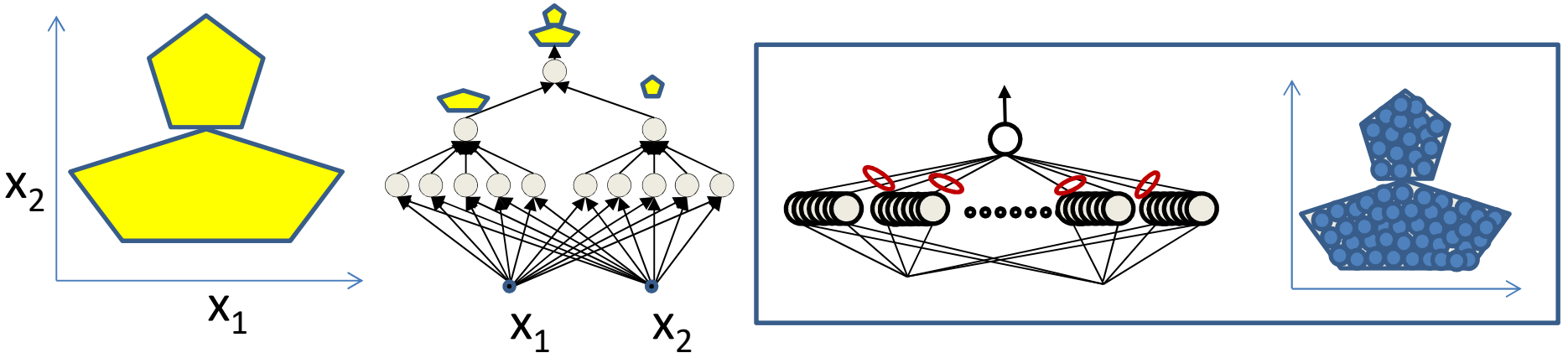
# MLP: Universal classifier



- MLPs can capture *any* classification boundary
- A *one-layer MLP* can model any classification boundary
- *MLPs are universal classifiers*

# Depth and the universal classifier

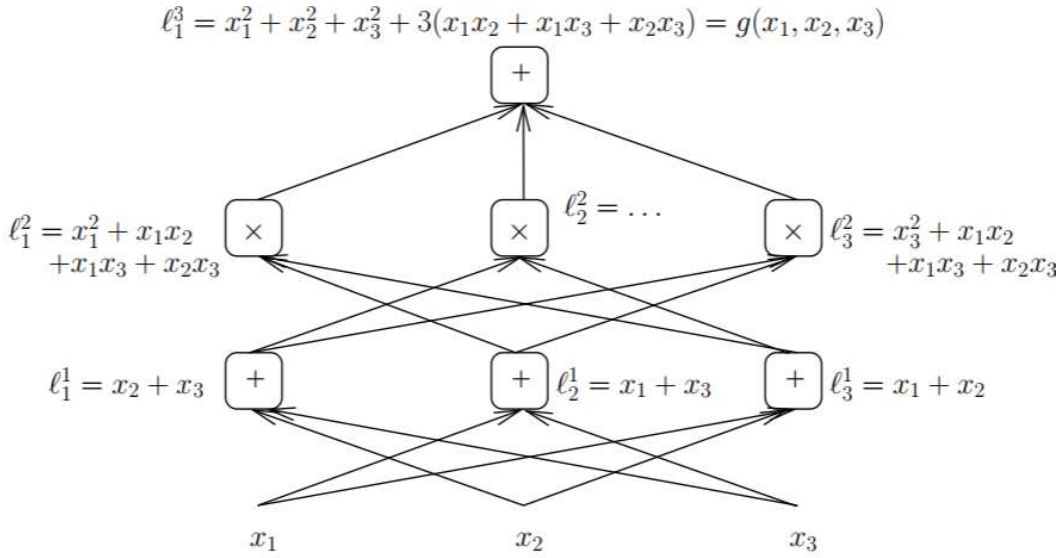



- Deeper networks can require far fewer neurons

# Optimal depth..

- Formal analyses typically view these as category of *arithmetic circuits*
  - Compute polynomials over any field
    - Valiant et. al: A polynomial of degree n requires a network of depth $log^2(n)$
      - Cannot be computed with shallower networks
      - The majority of functions are very high (possibly ∞) order polynomials
    - Bengio et. al: Shows a similar result for sum-product networks
      - But only considers two-input units
      - Generalized by Mhaskar et al. to all functions that can be expressed as a binary tree
  - Depth/Size analyses of arithmetic circuits still a research problem

# Special case: Sum-product nets




- "Shallow vs deep sum-product networks," Oliver Dellaleau and Yoshua Bengio
  - For networks where layers alternately perform either sums or products, a deep network may require an exponentially fewer number of layers than a shallow one

# Depth in sum-product networks

Theorem 5

*A certain class of functions $\mathcal{F}$ of $n$ inputs can be represented using a deep network with $\mathcal{O}(n)$ units, whereas it would require $\mathcal{O}(2^{\sqrt{n}})$ units for a shallow network.*

Theorem 6

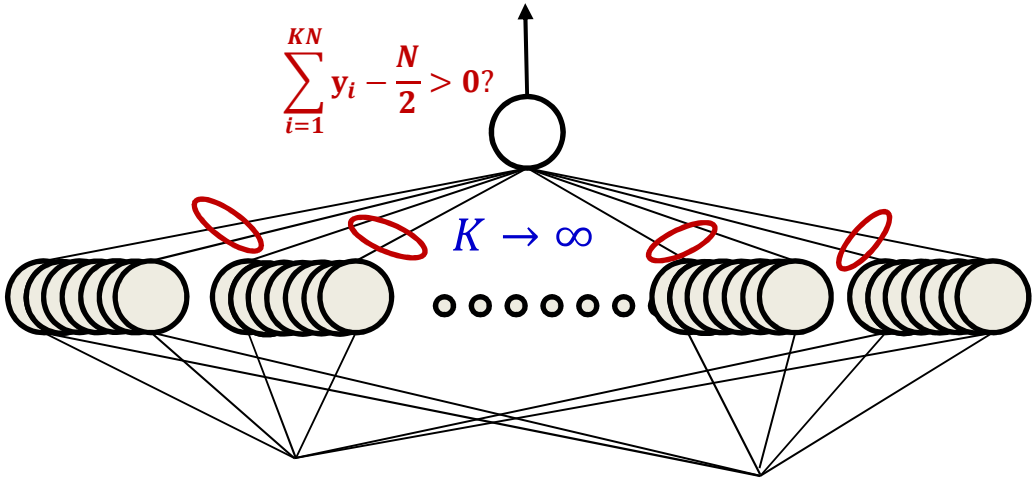
*For a certain class of functions $\mathcal{G}$ of $n$ inputs, the deep sum-product network with depth $k$ can be represented with $\mathcal{O}(nk)$ units, whereas it would require $\mathcal{O}((n-1)^k)$ units for a shallow network.*

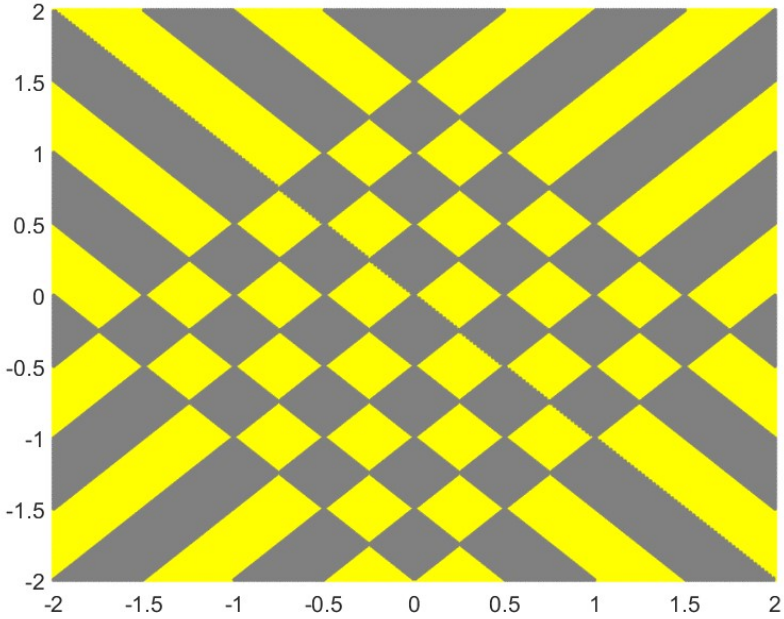
# Optimal depth in *generic* nets

- We look at a different pattern:
  - "worst case" decision boundaries

- For *threshold-activation* networks
  - Generalizes to other nets

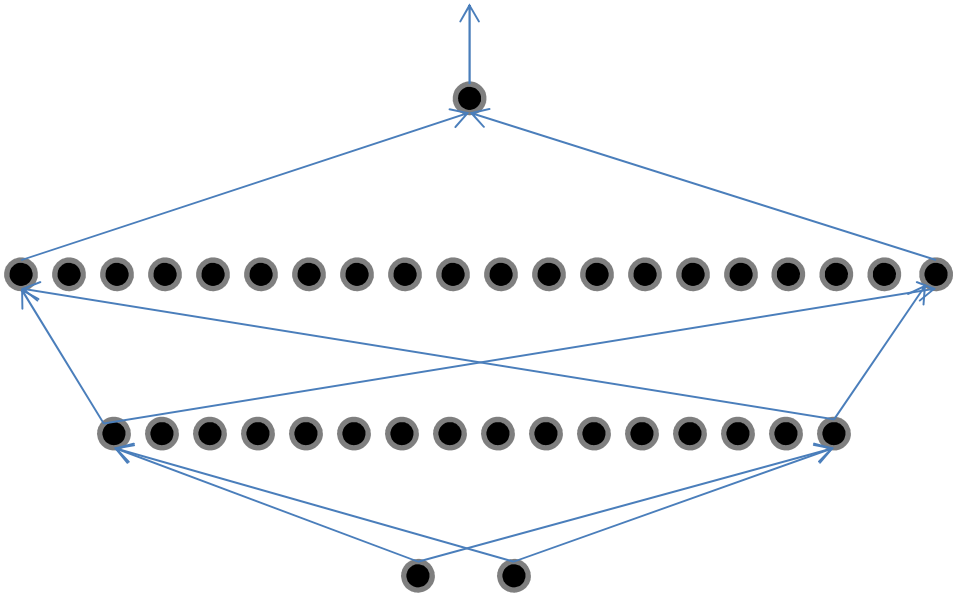
# Optimal depth

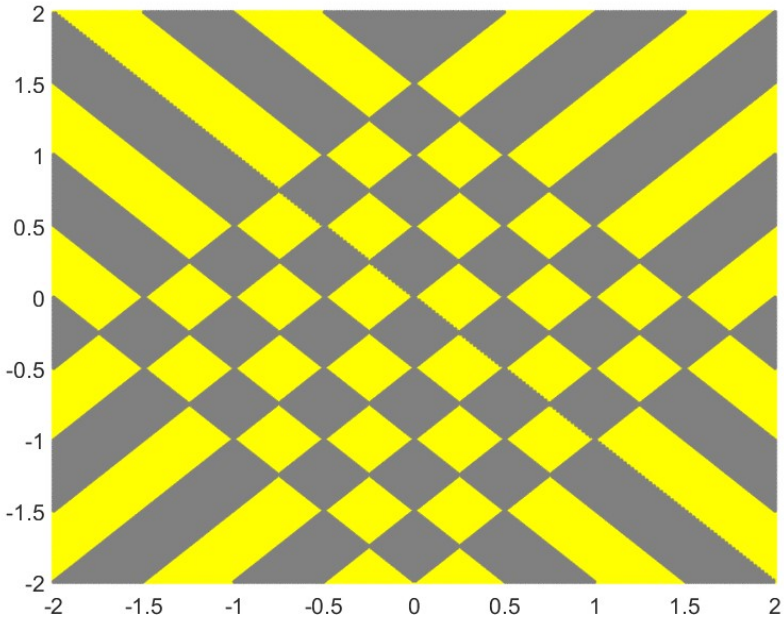



- A naïve one-hidden-layer neural network will required infinite hidden neurons

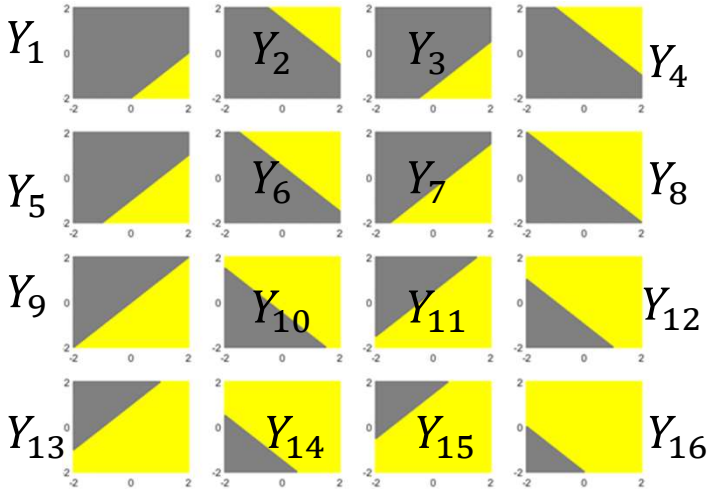
# Optimal depth

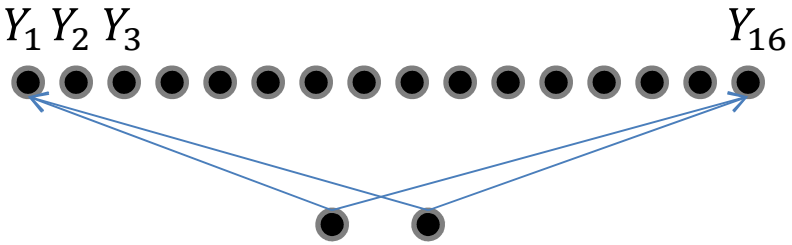


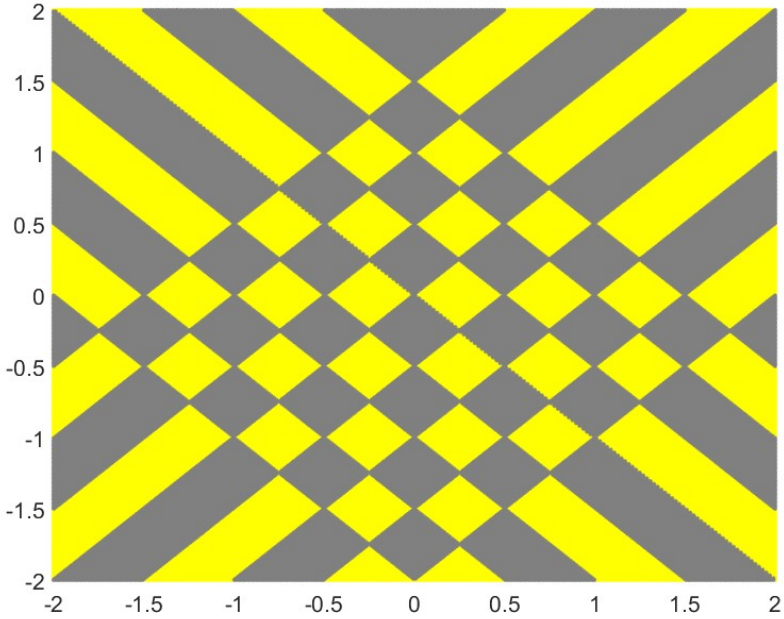



- Two hidden-layer network: 56 hidden neurons

# Optimal depth

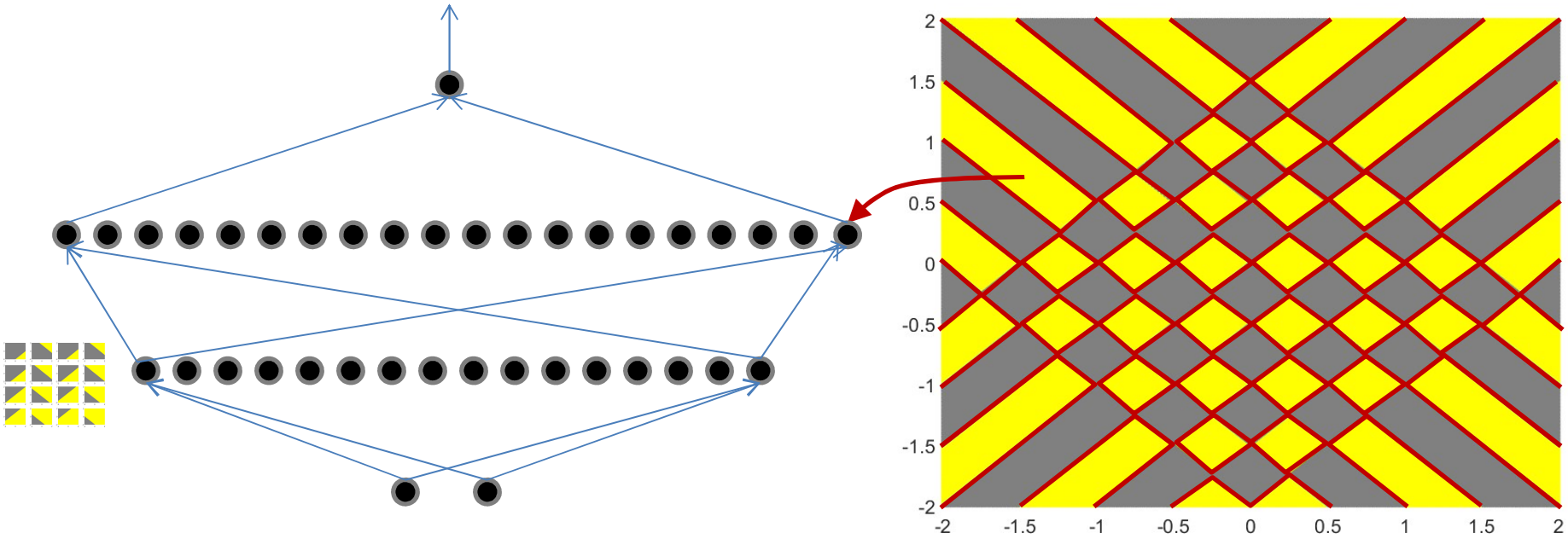


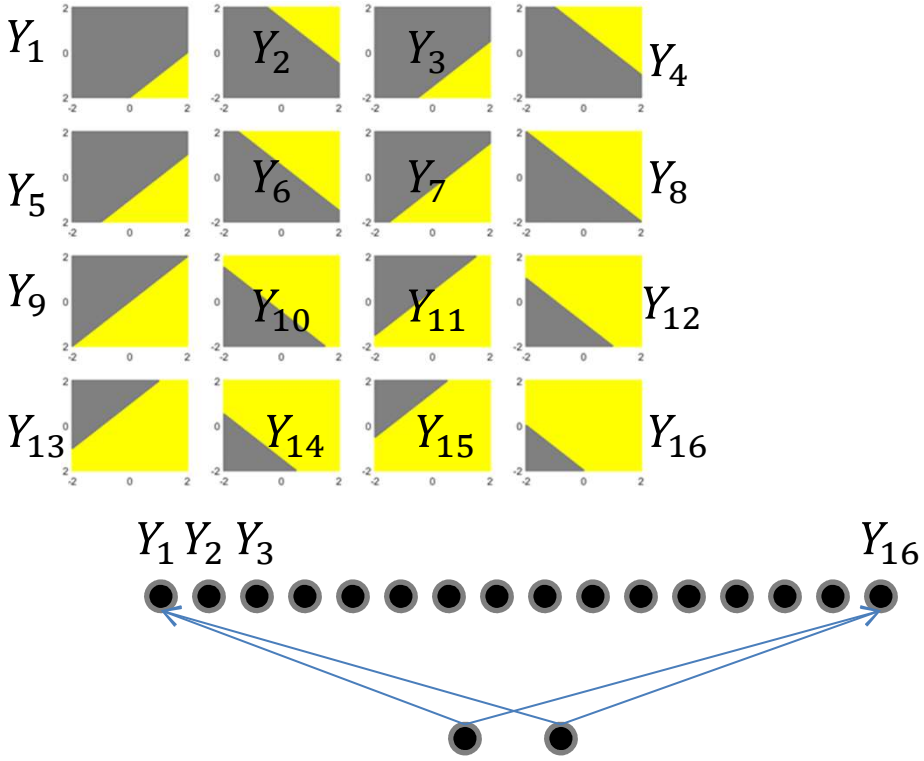

- Two layer network: 56 hidden neurons
  - 16 neurons in hidden layer 1

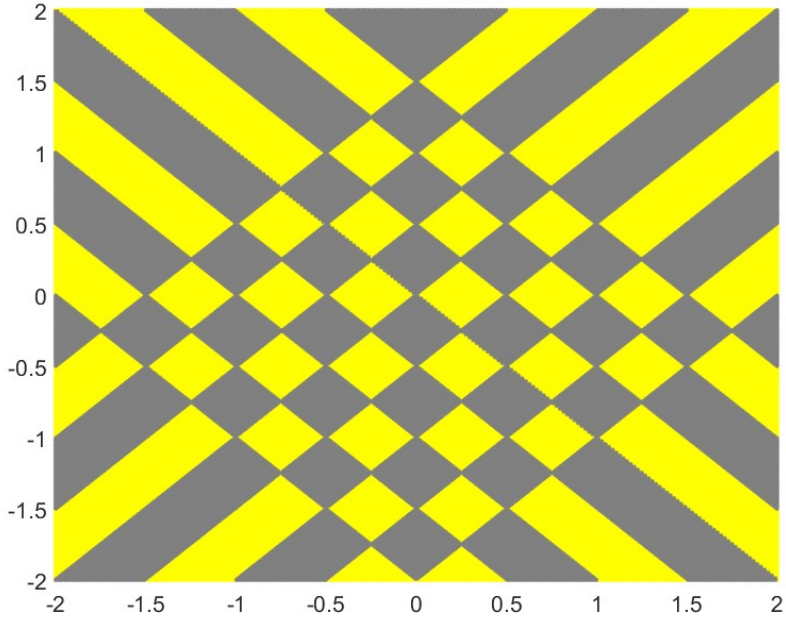
# Optimal depth




- Two-layer network: 56 hidden neurons
  - 16 in hidden layer 1
  - 40 in hidden layer 2
  - 57 total neurons, including output neuron

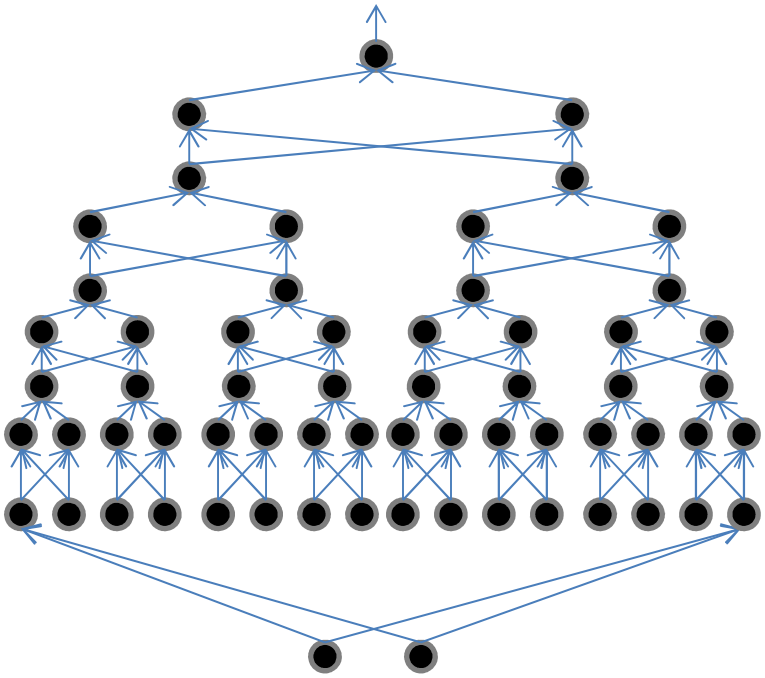
# Optimal depth

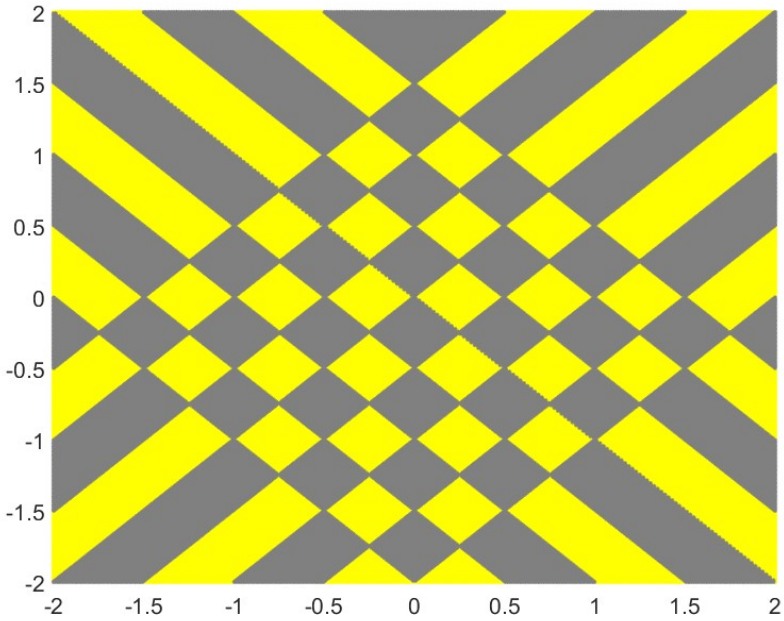



- But this is just $Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{16}$

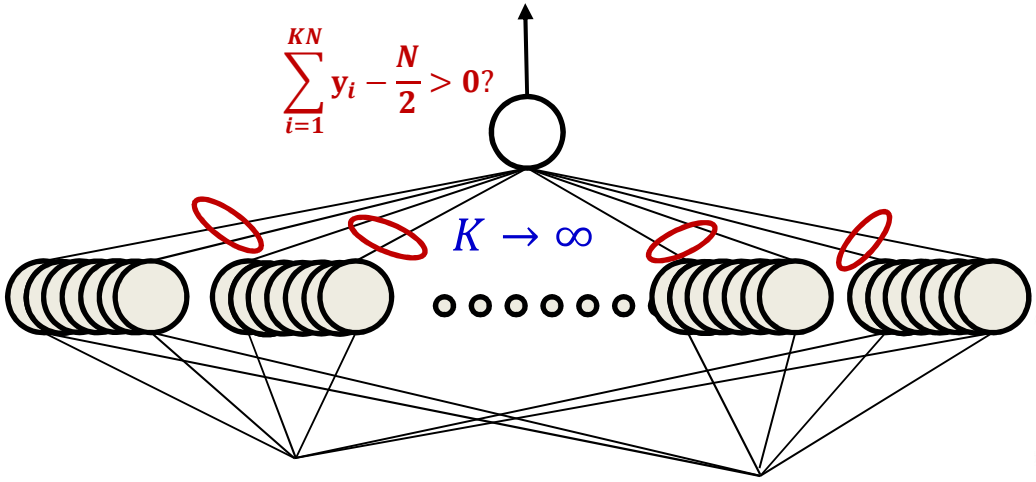
# Optimal depth

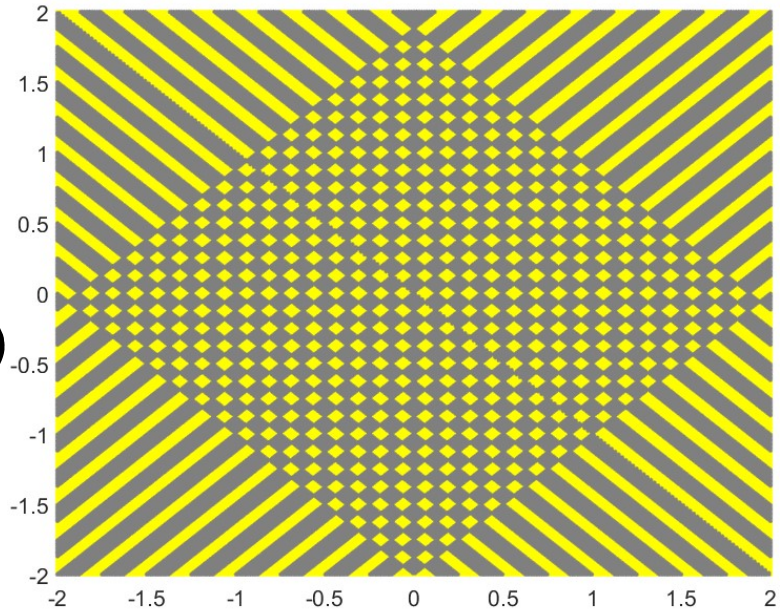





- But this is just $Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{16}$
  - The XOR net will require 16 + 15x3 = 61 neurons
    - 46 neurons if we use a two-gate XOR

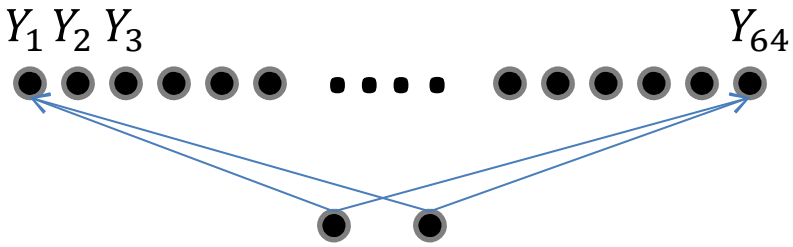
# Optimal depth

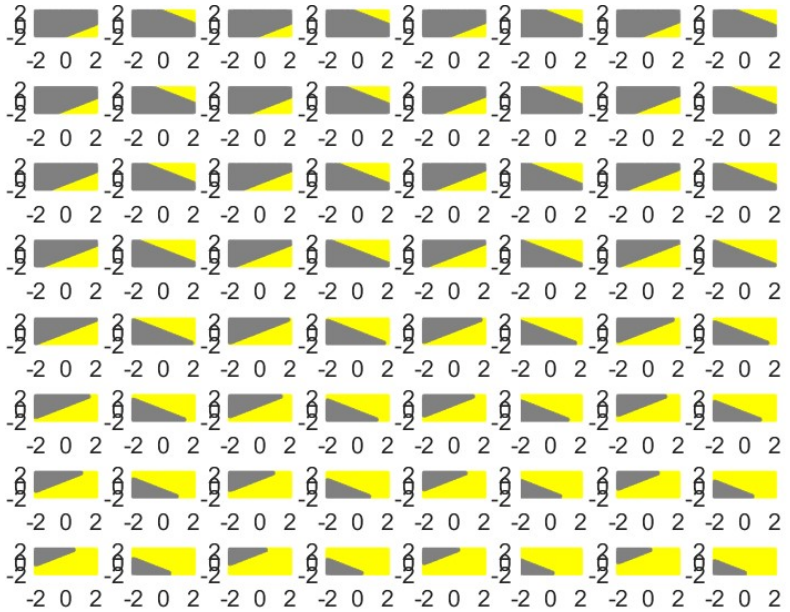



- A naïve one-hidden-layer neural network will required infinite hidden neurons

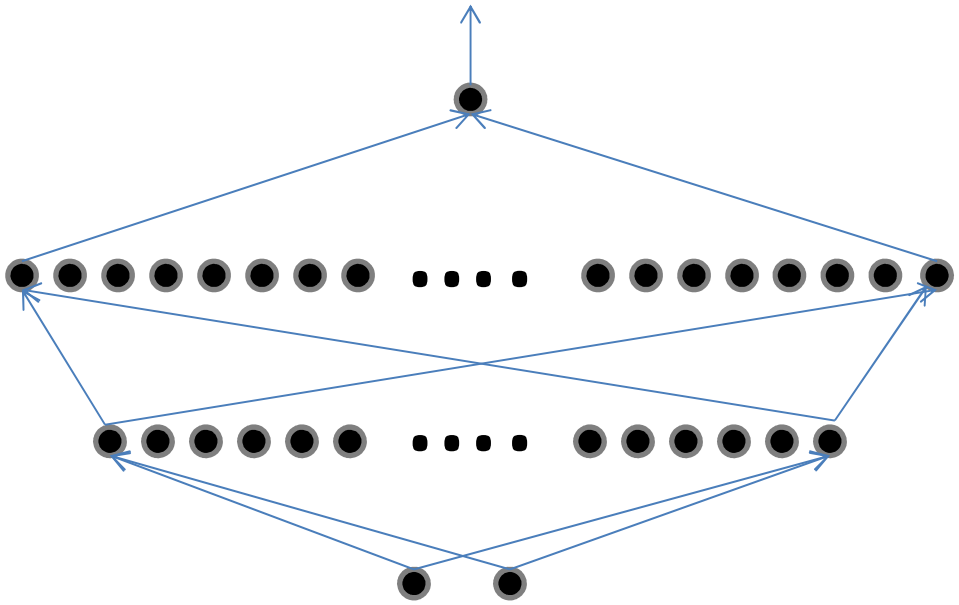
# Actual linear units

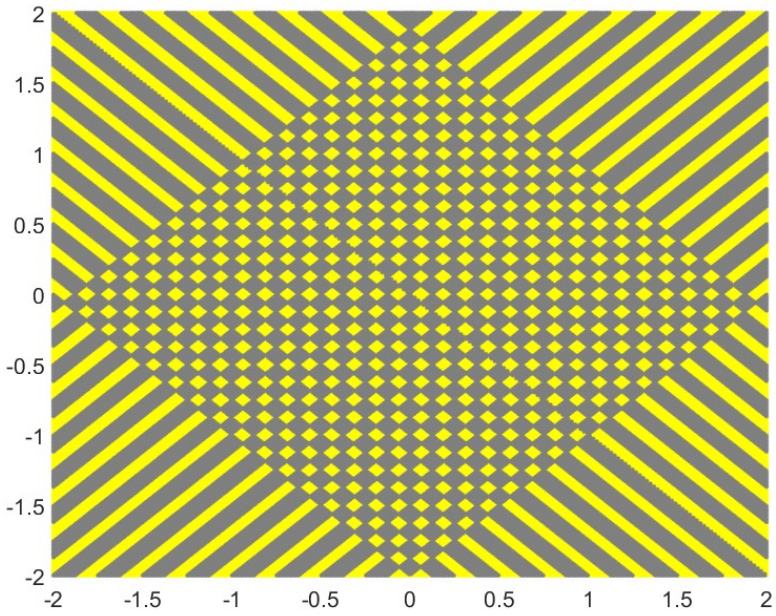


- 64 basic linear feature detectors

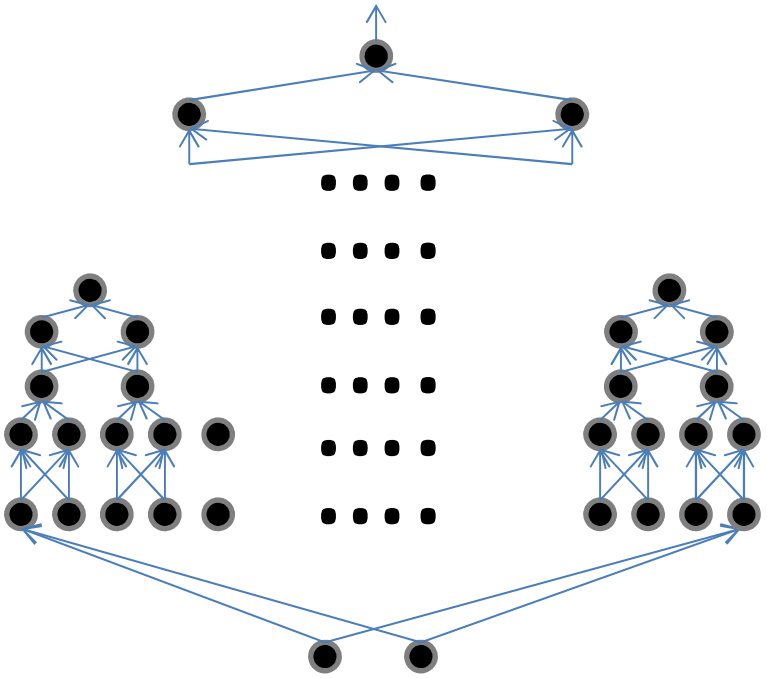
# Optimal depth

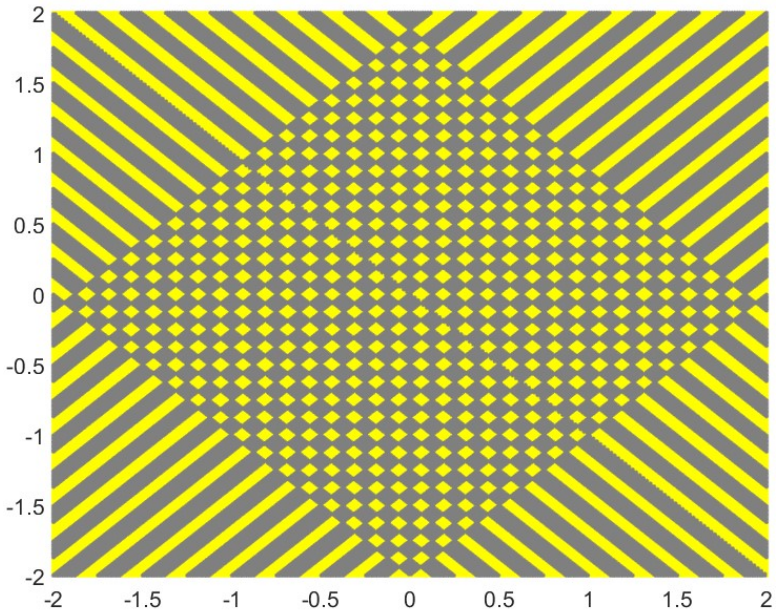





- Two hidden layers:  608 hidden neurons
  - 64 in layer 1
  - 544 in layer 2
- 609 total neurons (including output neuron)

# Optimal depth

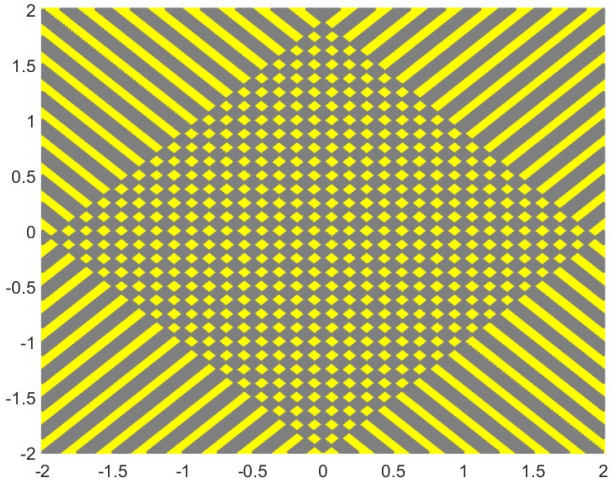





- XOR network (12 hidden layers): 253 neurons
  - 190 neurons with 2-gate XOR
- The difference in size between the deeper optimal (XOR) net and shallower nets increases with increasing pattern complexity and input dimension

# Network size?

- In this problem the 2-layer net was *quadratic* in the number of lines
  - $\lfloor (N+2)^2/8 \rfloor$ neurons in 2nd hidden layer
  - Not exponential
  - Even though the pattern is an XOR
  - Why?

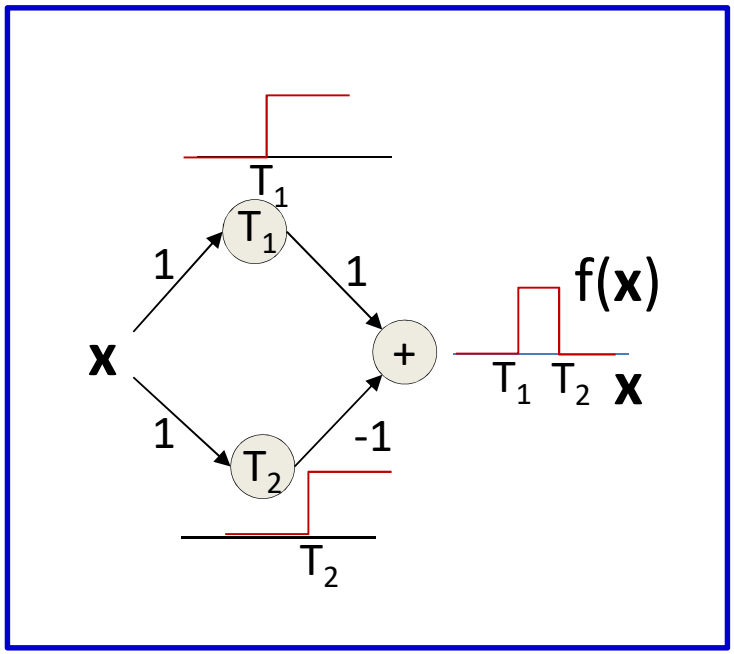



- The data are two-dimensional!
  - Only two *fully independent* features
  - The pattern is exponential in the *dimension of the input (two)!*
- For general case of $N$ mutually intersecting hyperplanes in $D$ dimensions, we will need $\mathcal{O}\left(\frac{N^D}{(D-1)!}\right)$ weights (assuming $N \gg D$).
  - Increasing input dimensions can increase the worst-case size of the shallower network exponentially, but not the XOR net
    - The size of the XOR net depends only on the number of first-level linear detectors ($N$)

# Story so far

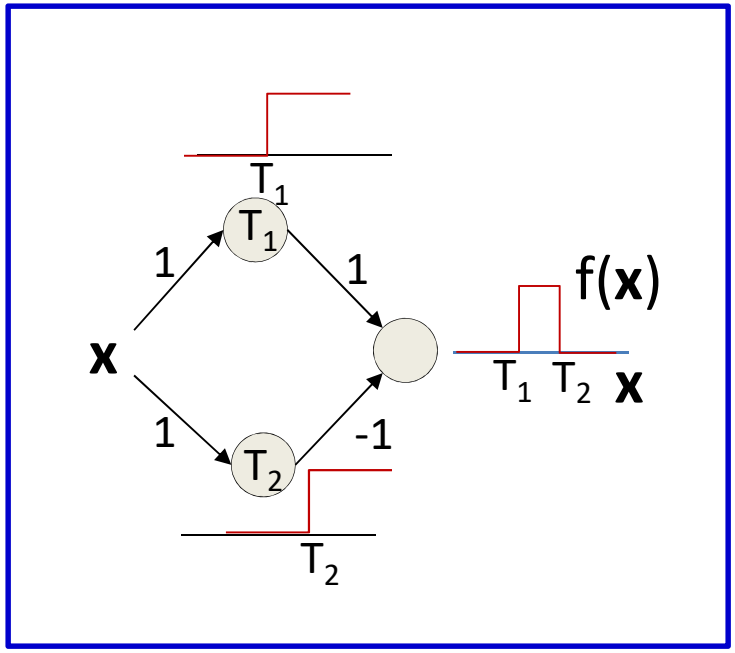
- Multi-layer perceptrons are *Universal Boolean Machines*
  - Even a network with a *single* hidden layer is a universal Boolean machine
- Multi-layer perceptrons are *Universal Classification Functions*
  - Even a network with a single hidden layer is a universal classifier
- But a single-layer network may require an exponentially large number of perceptrons than a deep one
- Deeper networks may require far fewer neurons than shallower networks to express the same function
  - Could be *exponentially* smaller
  - Deeper networks are more *expressive*

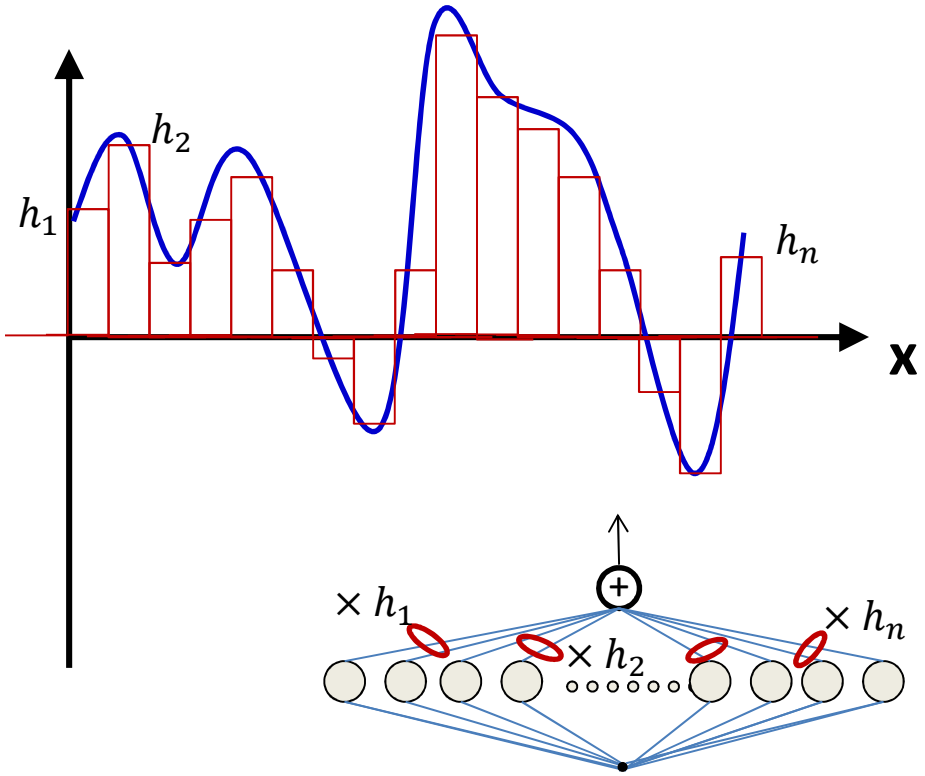
# MLP as a continuous-valued regression




- A simple 3-unit MLP with a "summing" output unit can generate a "square pulse" over an input
  - Output is 1 only if the input lies between $T_1$ and $T_2$
  - $T_1$ and $T_2$ can be arbitrarily specified

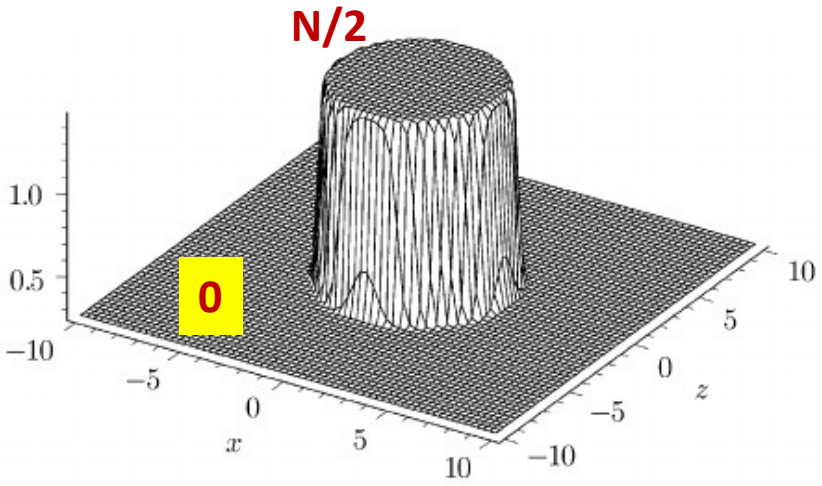
# MLP as a continuous-valued regression

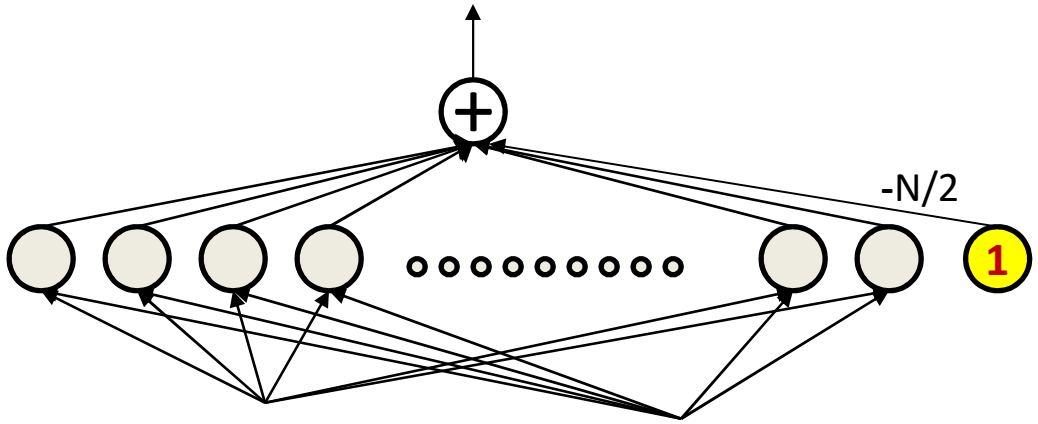



- A simple 3-unit MLP can generate a "square pulse" over an input
- **An MLP with many units can model an arbitrary function over an input**
  - To arbitrary precision
    - Simply make the individual pulses narrower
- ***A one-layer MLP can model an arbitrary function of a single input***

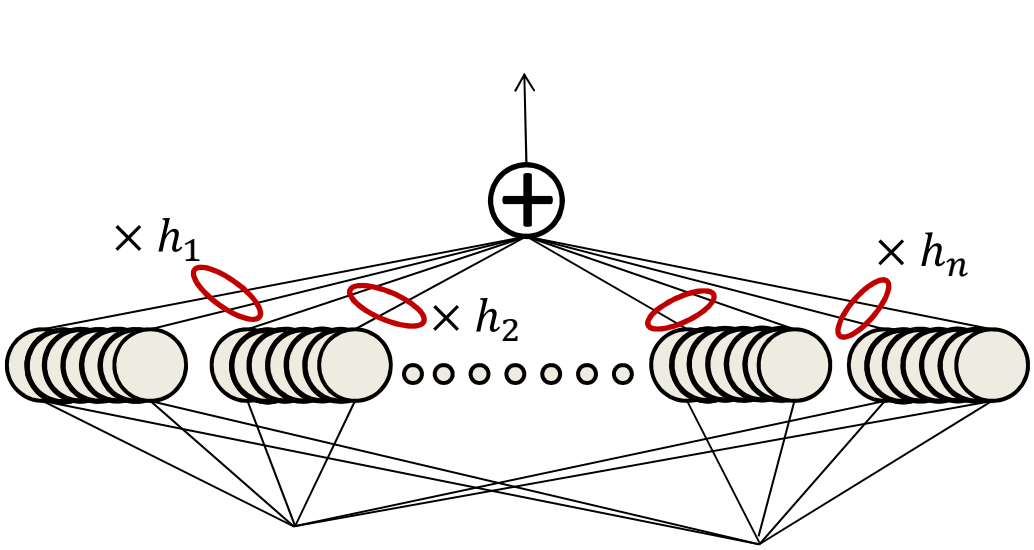
# For higher dimensions

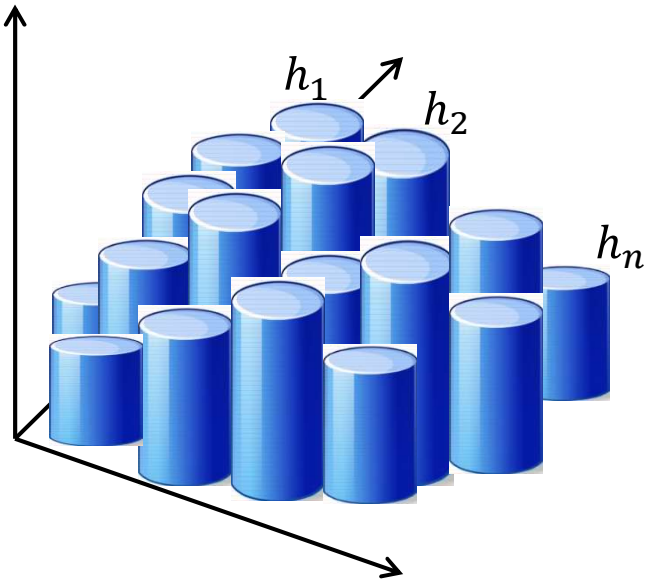



- An MLP can compose a cylinder
  - $N/2$ in the circle, 0 outside

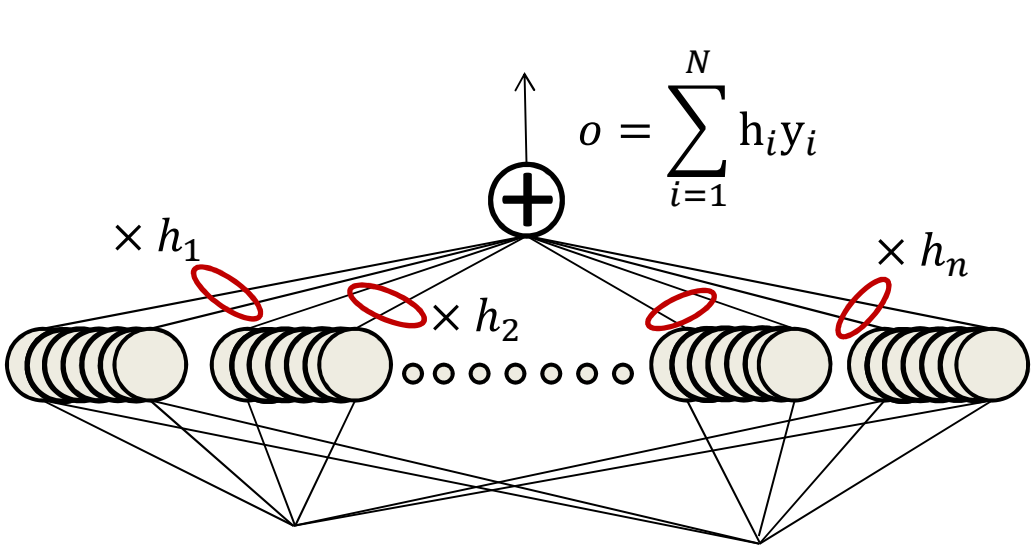
# MLP as a continuous-valued function

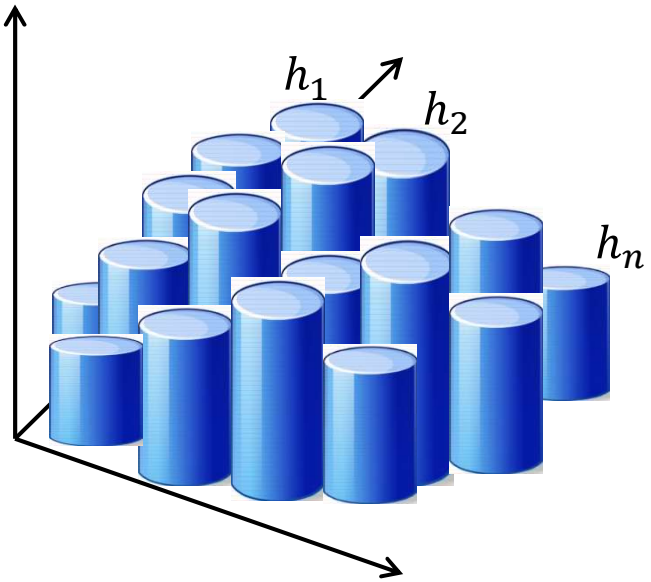



- MLPs can actually compose arbitrary functions in any number of dimensions!
  - Even with only one layer
    - As sums of scaled and shifted cylinders
  - To arbitrary precision
    - By making the cylinders thinner
  - **The MLP is a universal approximator!**

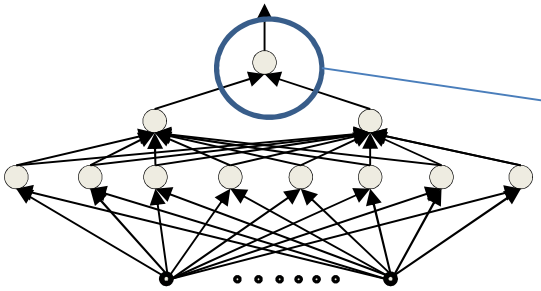
# Caution: MLPs with additive output units are universal approximators

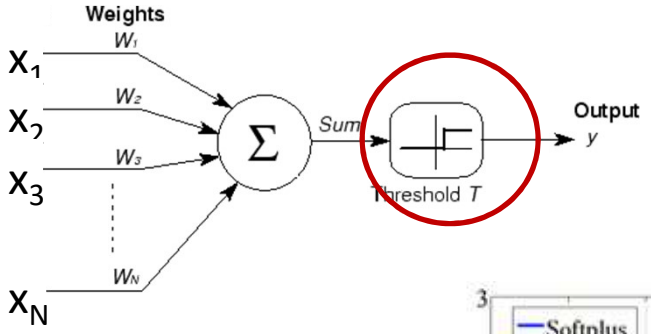


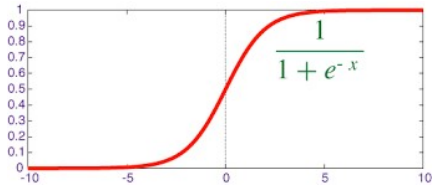

- MLPs can actually compose arbitrary functions
- But explanation so far only holds if the output unit only performs summation
  - i.e. does not have an additional “activation”

# "Proper" networks: Outputs with activations

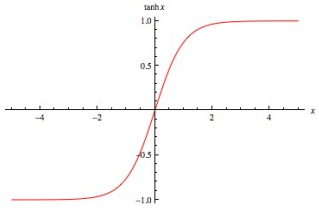


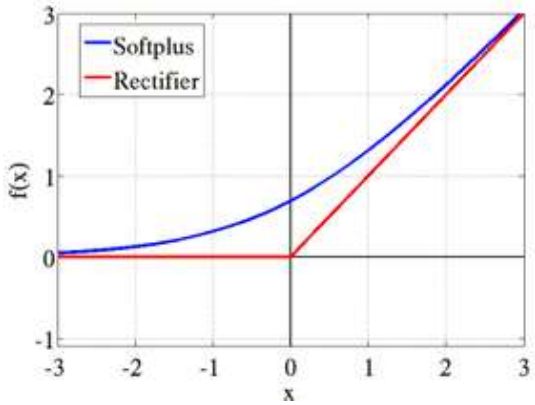



sigmoid



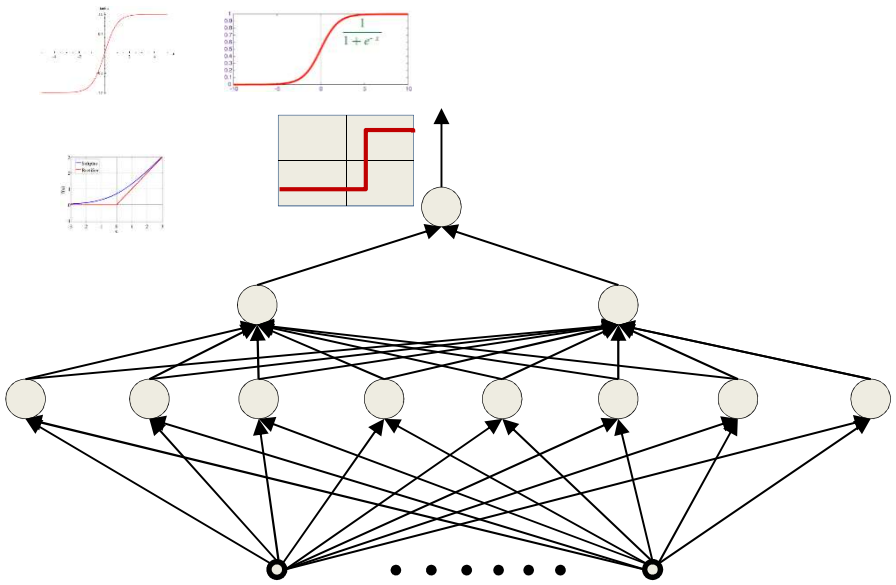

tanh






- Output neuron may have actual "activation"
  - Threshold, sigmoid, tanh, softplus, rectifier, etc.
- What is the property of such networks?

# The network as a function



$f: \{0,1\}^N \rightarrow \{0,1\}$ $Boolean$

$f: R^N \rightarrow \{0,1\}$ $Threshold$

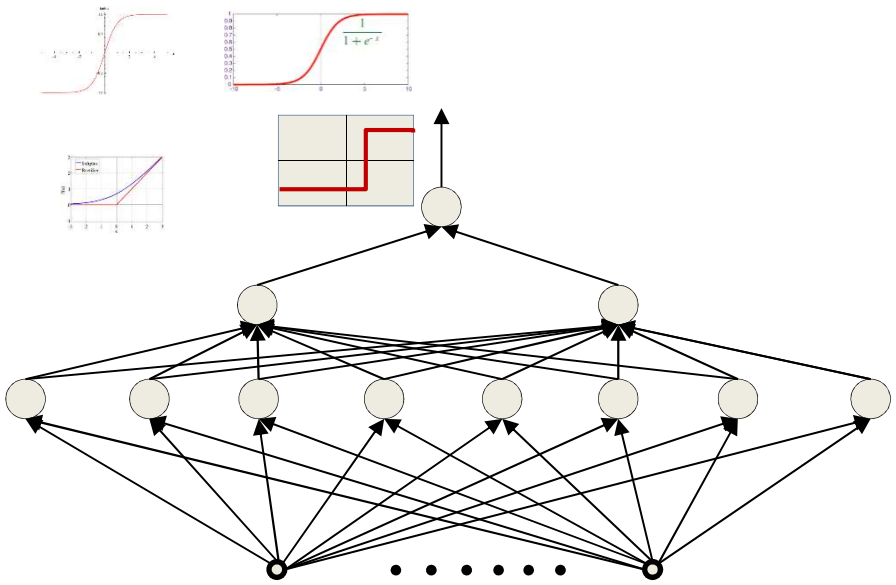
$f: R^N \rightarrow (0,1)$ $Sigmoid$

$f: R^N \rightarrow (-1,1)$ $Tanh$

$f: R^N \rightarrow (0,\infty)$ $Softrectifier, Rectifier$

- Output unit with *activation function*
  - Threshold or Sigmoid, or any other
- The network is actually a universal map from the entire domain of input values to the entire range of the output activation
  - All values the activation function of the output neuron

# The network as a function



$f: \{0,1\}^N \rightarrow \{0,1\}$ *Boolean*

$f: R^N \rightarrow \{0,1\}$ *Threshold*

$f: R^N \rightarrow (0,1)$ *Sigmoid*

$f: R^N \rightarrow (-1,1)$ *Tanh*

$f: R^N \rightarrow (0, \infty)$ *Softrectifier, Rectifier*

The MLP is a *Universal Approximator* for the entire *class* of functions (maps) it represents!

~~Output unit with activation function~~

- Threshold or Sigmoid, or any other

- The network is actually a universal map from the entire domain of input values to the entire range of the output activation
  - All values the activation function of the output neuron

# The issue of depth

- Previous discussion showed that a *single-layer* MLP is a universal function approximator
  - Can approximate any function to arbitrary precision
  - But may require infinite neurons in the layer

- More generally, deeper networks will require far fewer neurons for the same approximation error
  - The network is a generic map
    - The same principles that apply for Boolean networks apply here
  - Can be exponentially fewer than the 1-layer network