

Multi-Layer Networks

M. Soleymani
Deep Learning

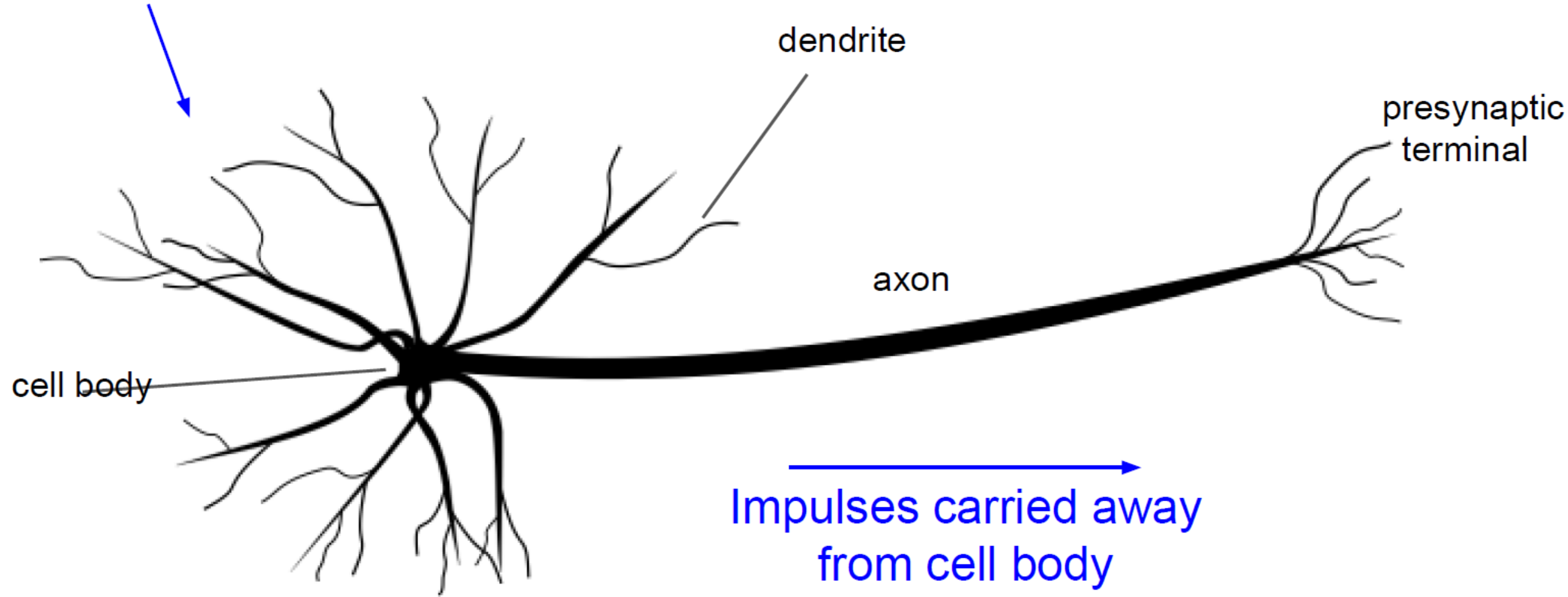
Sharif University of Technology
Spring 2020

Most slides have been adapted from
Bhiksha Raj, 11-785, CMU 2019
and Fei Fei Li lectures, cs231n, Stanford 2017,
and some from Hinton, “NN for Machine Learning”, coursera, 2015.

Reasons to study neural computation

- Neuroscience: To understand how the brain actually works.
 - Its very big and very complicated and made of stuff that dies when you poke it around. So we need to use computer simulations.
- AI: To solve practical problems by using novel learning algorithms inspired by the brain
 - Learning algorithms can be very useful even if they are not how the brain actually works.

Impulses carried toward cell body



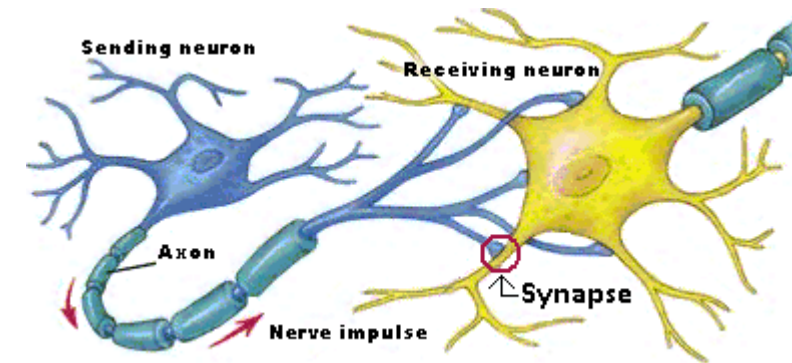
[This image](#) by Felipe Perucho is licensed under [CC-BY 3.0](#)

A typical cortical neuron

- Gross physical structure:
 - There is one axon that branches
 - There is a dendritic tree that collects input from other neurons.



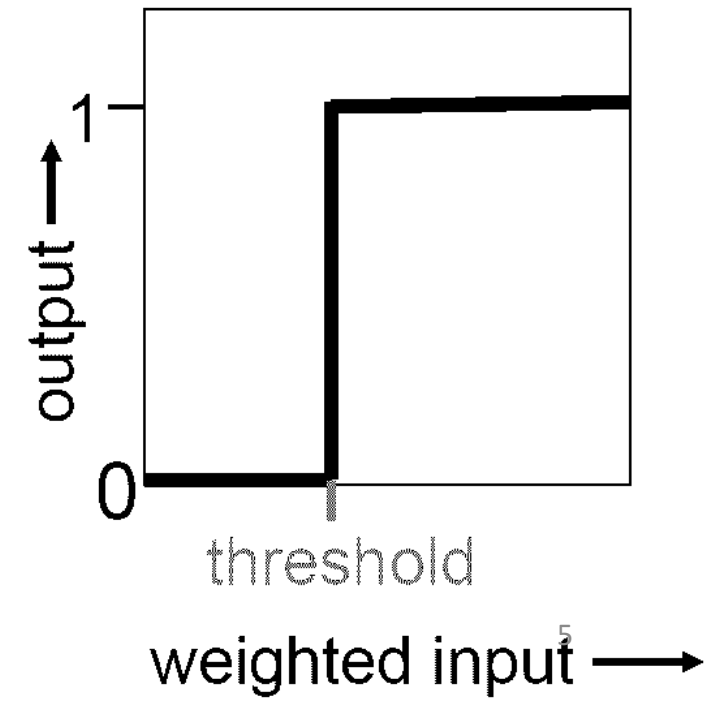
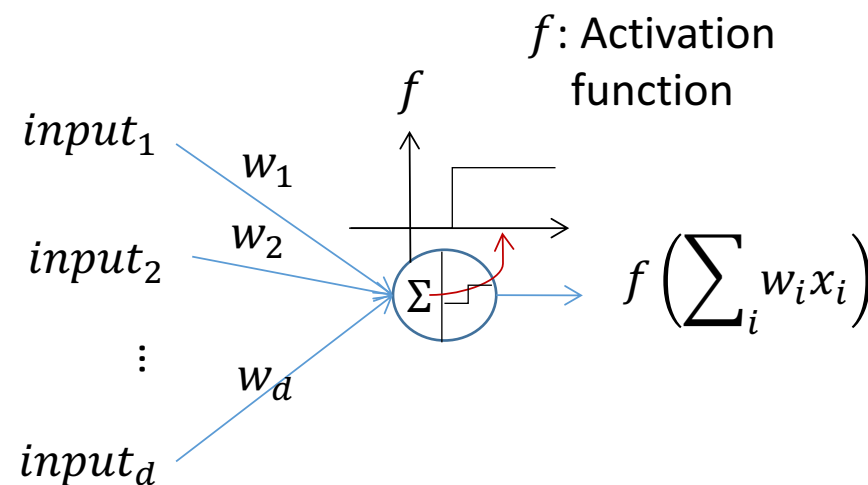
- Axons typically contact dendritic trees at synapses
 - A spike of activity in the axon causes charge to be injected into the post-synaptic neuron.



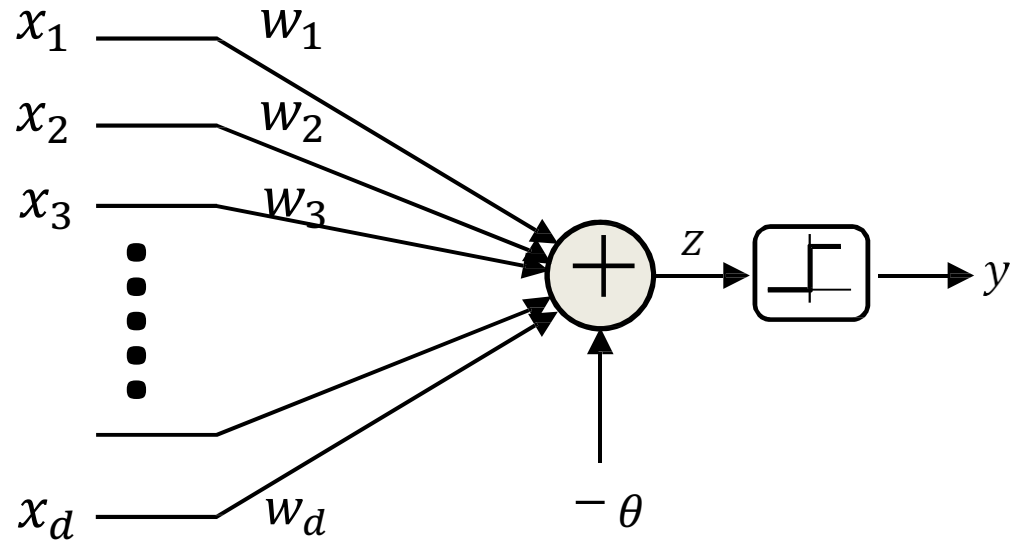
- Spike generation:
 - There is an axon hillock that generates outgoing spikes whenever enough charge has flowed in at synapses to depolarize the cell membrane.

Binary threshold neurons

- McCulloch-Pitts (1943): influenced Von Neumann.
 - First compute a weighted sum of the inputs.
 - send out a spike of activity if the weighted sum exceeds a threshold.
 - McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition!



A better figure



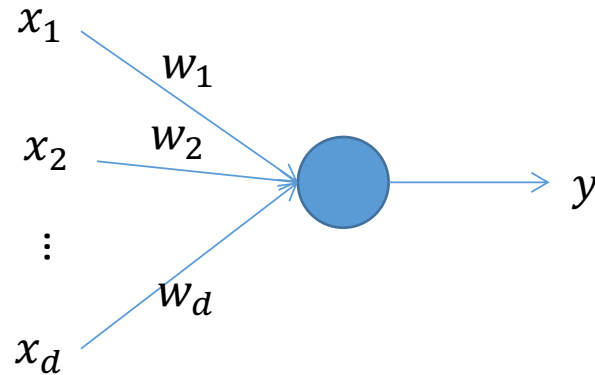
$$z = \sum_i w_i x_i - \theta$$

$$z = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{else} \end{cases}$$

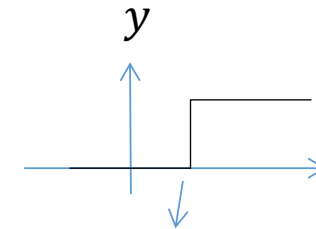
- A threshold unit
 - “Fires” if the weighted sum of inputs and the “bias” T is positive

McCulloch-Pitts neuron: binary threshold

Equivalent to

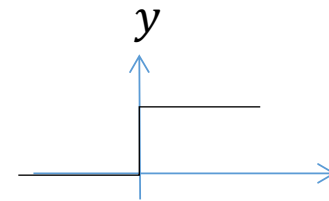
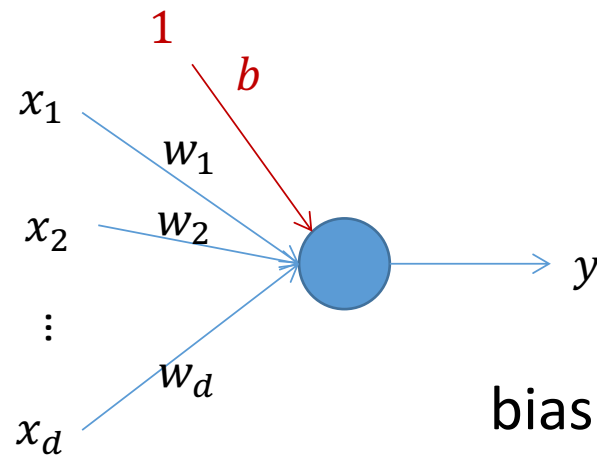


binary McCulloch-Pitts neuron



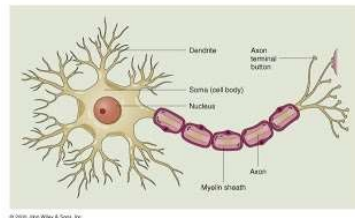
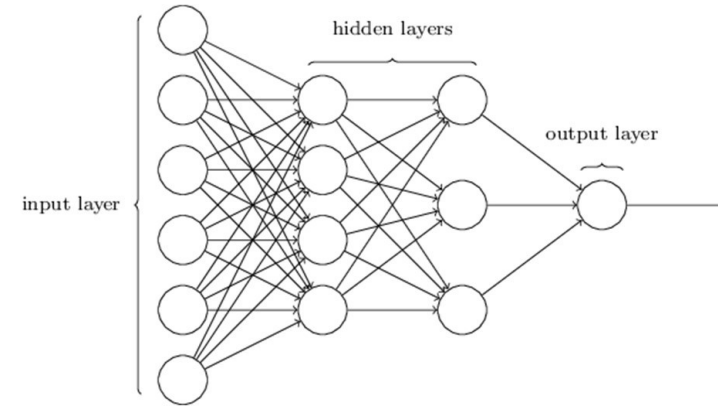
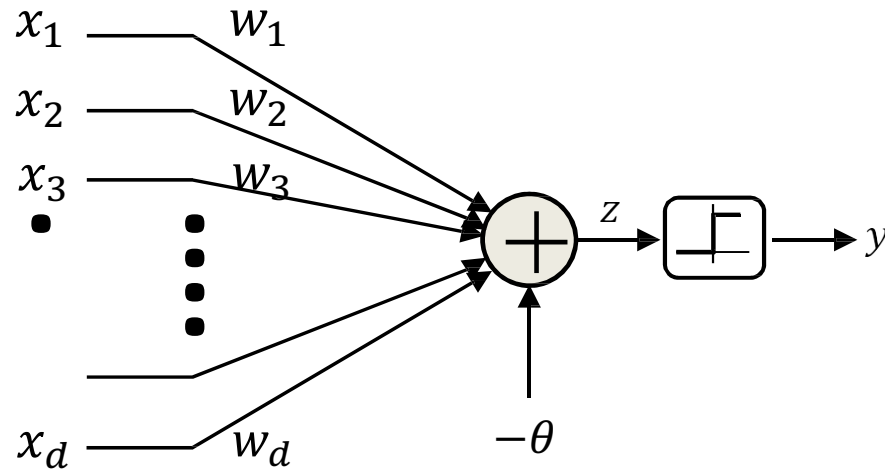
θ : activation threshold

$$y = \begin{cases} 1, & z \geq \theta \\ 0, & z < \theta \end{cases}$$



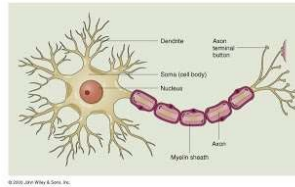
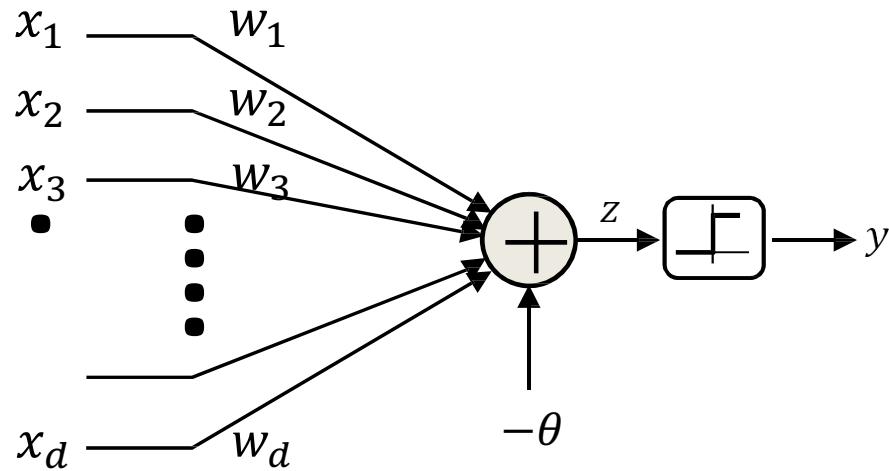
bias: $b = -\theta$

Neural nets and the brain



- Neural nets are composed of networks of computational models of neurons called perceptrons

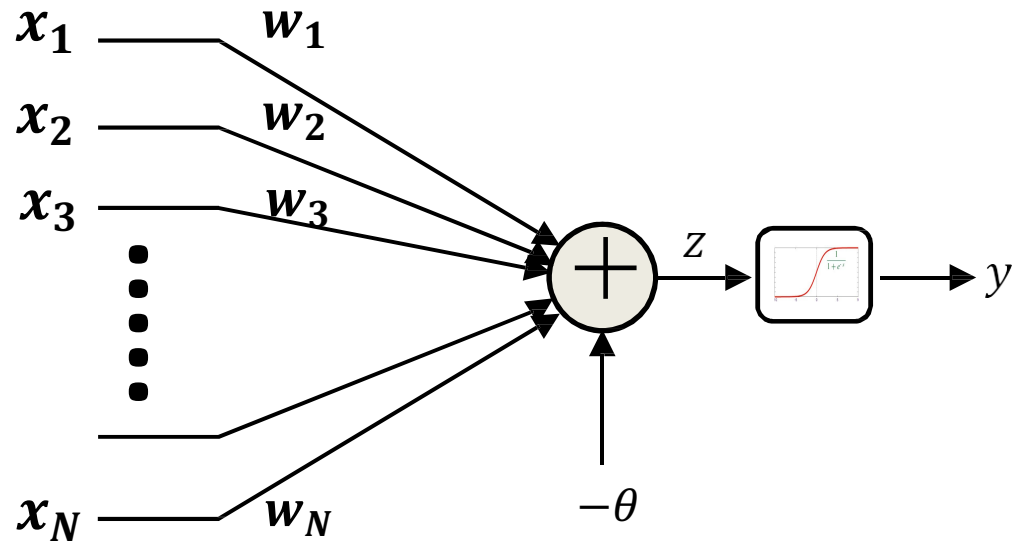
The perceptron



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

- A threshold unit
 - “Fires” if the weighted sum of inputs exceeds a threshold
 - Electrical engineers will call this a threshold gate
 - A basic unit of Boolean circuits

The “soft” perceptron (logistic)



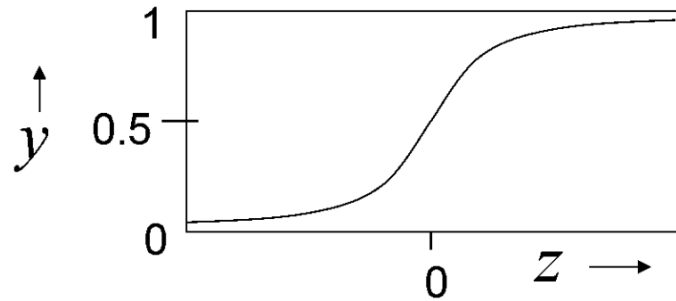
$$z = \sum_i w_i x_i - \theta$$

$$y = \frac{1}{1 + \exp(-z)}$$

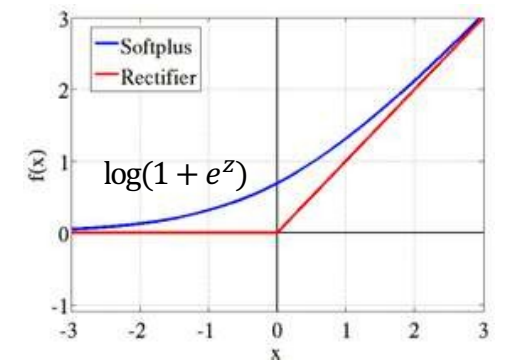
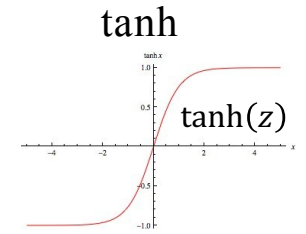
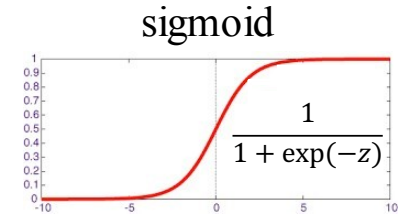
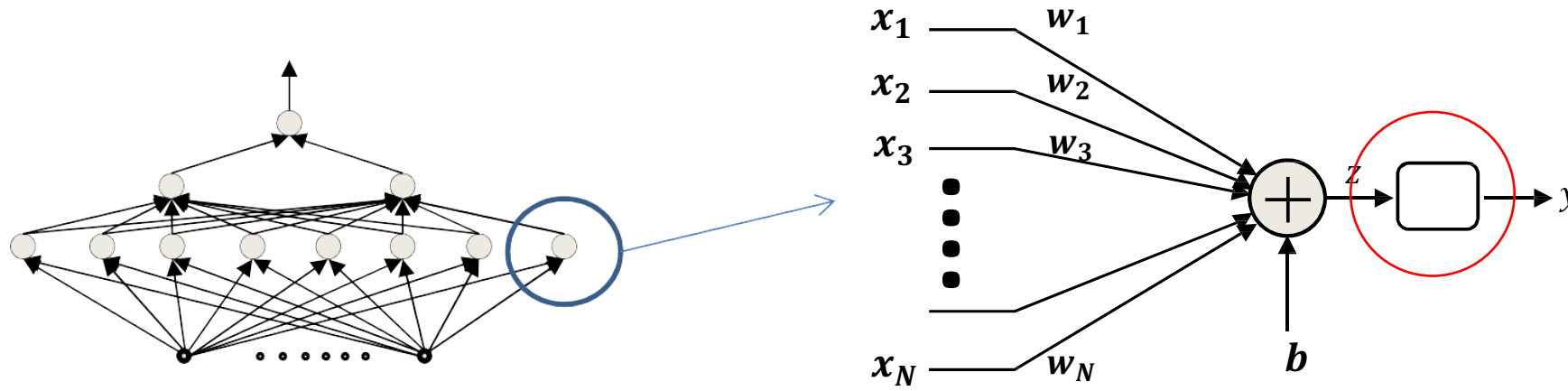
- A “squashing” function instead of a threshold at the output
 - The **sigmoid** “activation” replaces the threshold
 - **Activation**: acts on the weighted combination of inputs (and threshold)

Sigmoid neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
- Typically they use the logistic function
 - They have nice derivatives.



Other “activations”



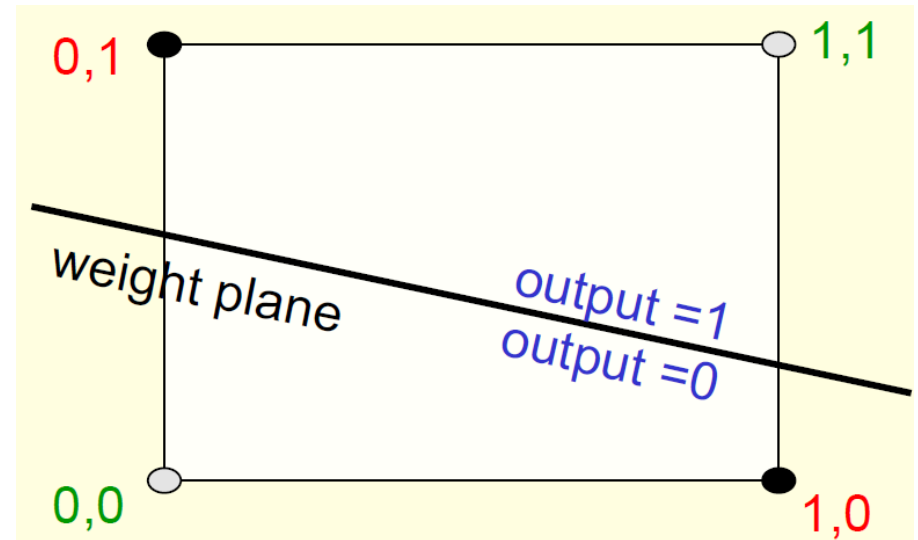
- Does not always have to be a squashing function
 - We will hear more about activations later
- We will continue to assume a “threshold” activation in this lecture

The history of perceptrons

- They were popularised by Frank Rosenblatt in the early 1960's.
 - They appeared to have a very powerful learning algorithm.
 - Lots of grand claims were made for what they could learn to do.
- In 1969, Minsky and Papert published a book called “Perceptrons” that analyzed what they could do and showed their limitations.
 - Many people thought these limitations applied to all neural network models.

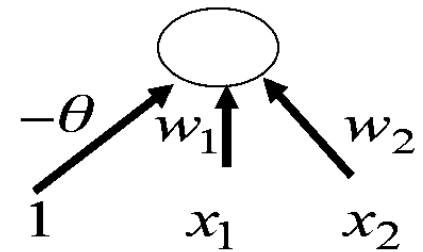
What binary threshold neurons cannot do

- A binary threshold output unit cannot even tell if two single bit features are the same!
- A geometric view of what binary threshold neurons cannot do
- The positive and negative cases cannot be separated by a plane



What binary threshold neurons cannot do

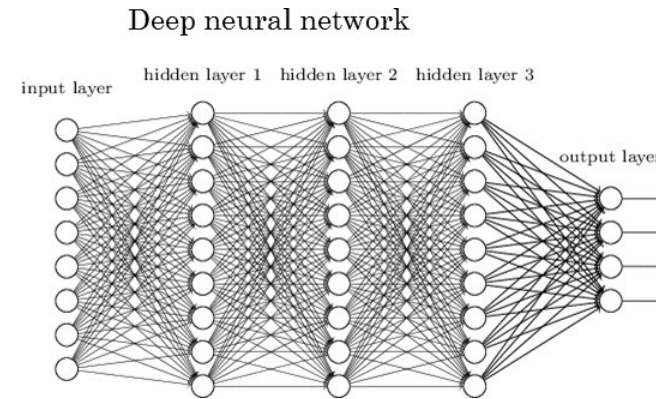
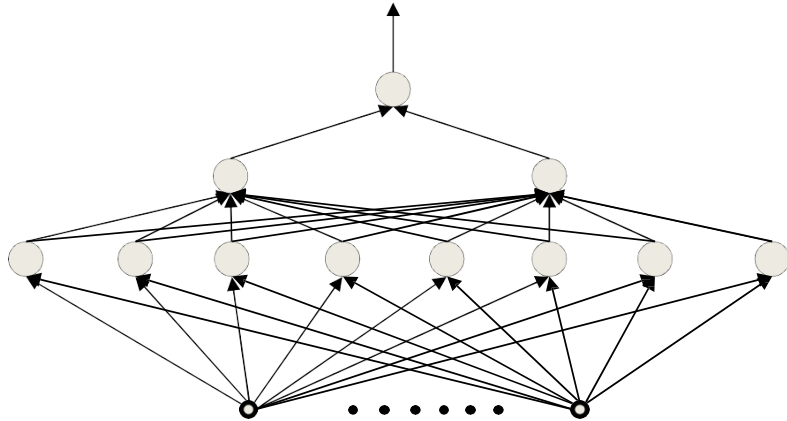
- Positive cases (same): $(1,1) \rightarrow 1$; $(0,0) \rightarrow 1$
- Negative cases (different): $(1,0) \rightarrow 0$; $(0,1) \rightarrow 0$
- The four input-output pairs give four inequalities that are impossible to satisfy:
 - $w_1 + w_2 \geq \theta$
 - $0 \geq \theta$
 - $w_1 < \theta$
 - $w_2 < \theta$



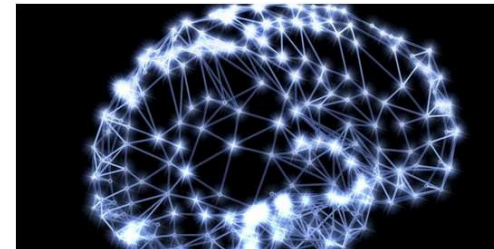
Networks with hidden units

- Networks without hidden units are very limited in the input-output mappings they can learn to model.
 - More layers of linear units do not help. Its still linear.
 - Fixed output non-linearities are not enough.
- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?

The multi-layer perceptron

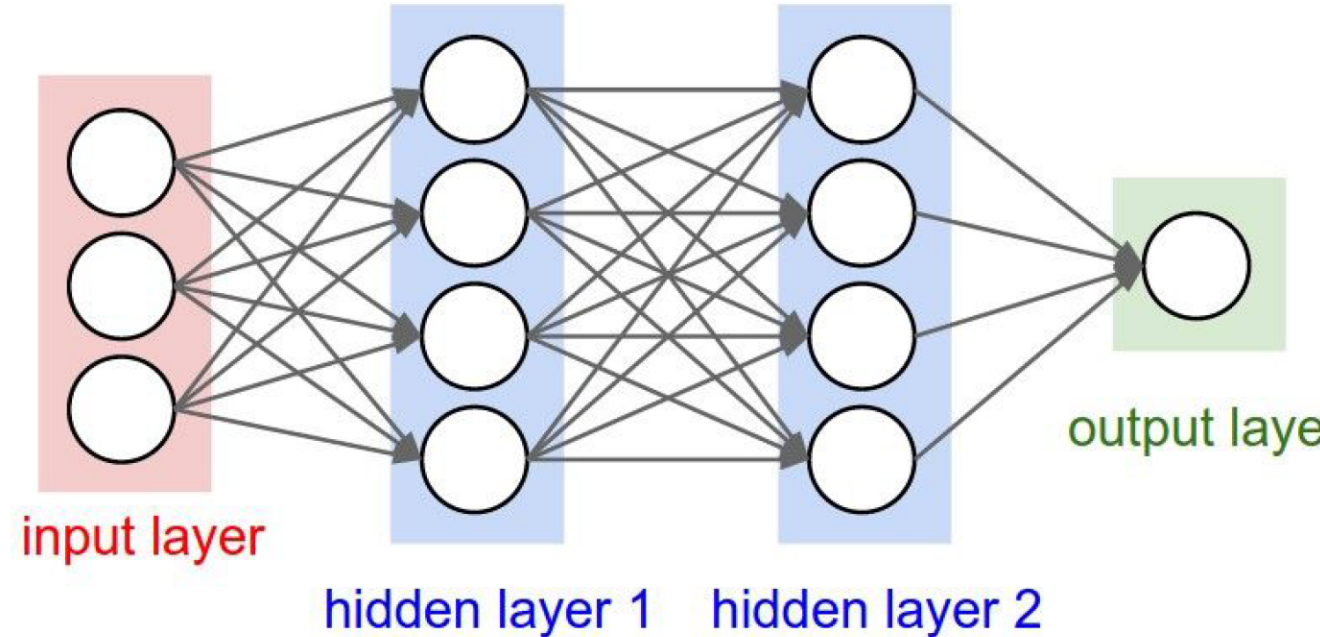
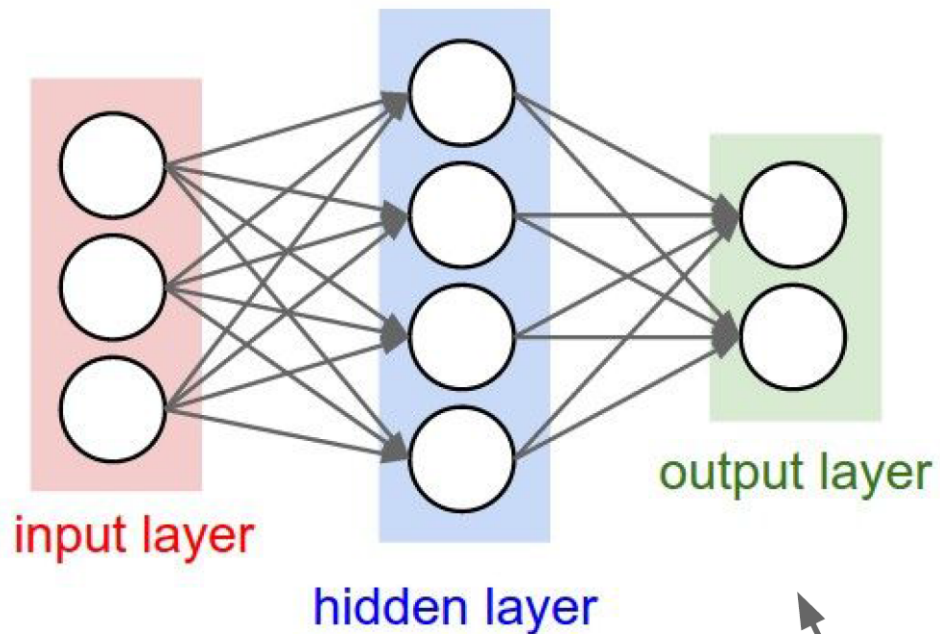


- A network of perceptrons
 - Generally “layered”



Feed-forward neural networks

- Also called **Multi-Layer Perceptron (MLP)**



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

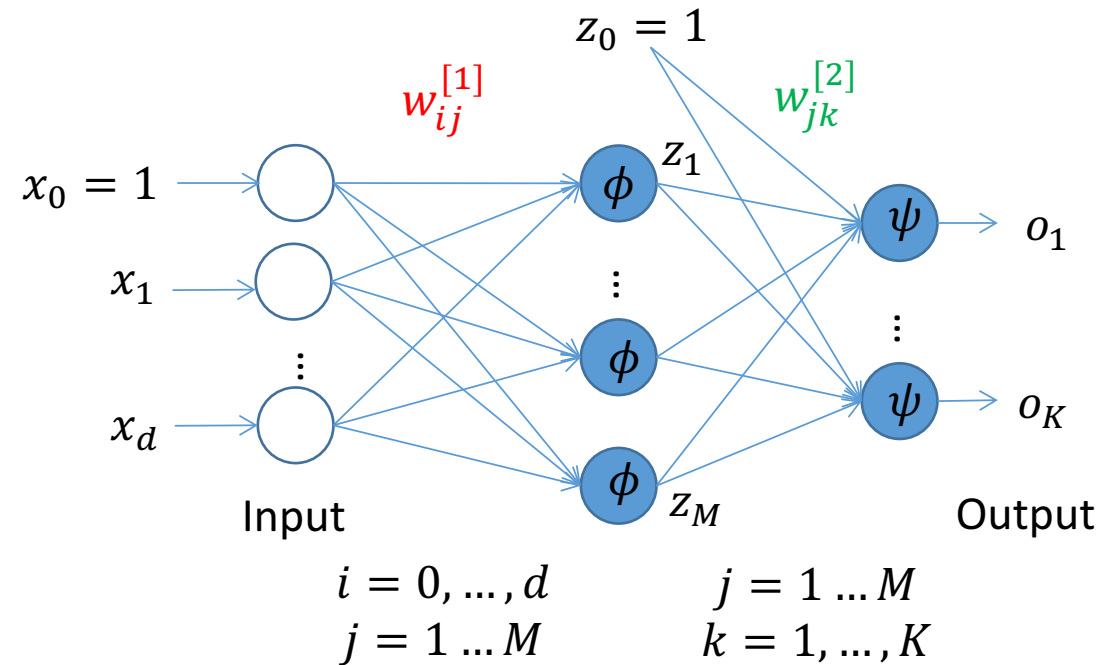
“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

MLP with single hidden layer

- Two-layer MLP (Number of layers of adaptive weights is counted)

$$o_k(\mathbf{x}) = \psi \left(\sum_{j=0}^M w_{jk}^{[2]} z_j \right) \Rightarrow o_k(\mathbf{x}) = \psi \left(\sum_{j=0}^M w_{jk}^{[2]} \underbrace{\phi \left(\sum_{i=0}^d w_{ij}^{[1]} x_i \right)}_{z_j} \right)$$



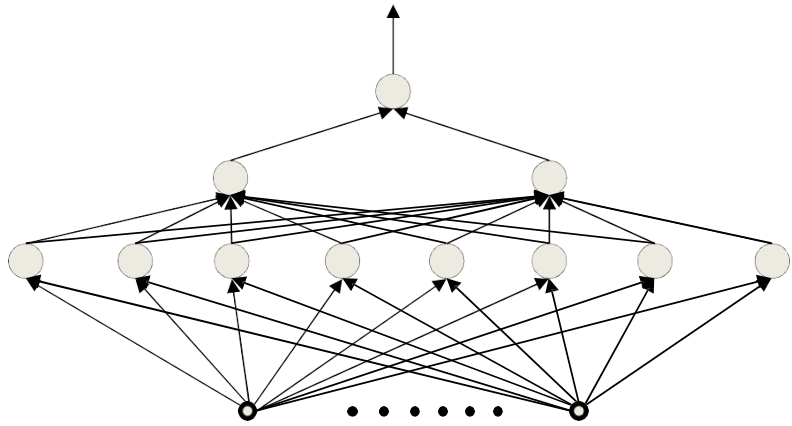
Beyond linear models

(Before) Linear score function: $f = \mathbf{W}x$

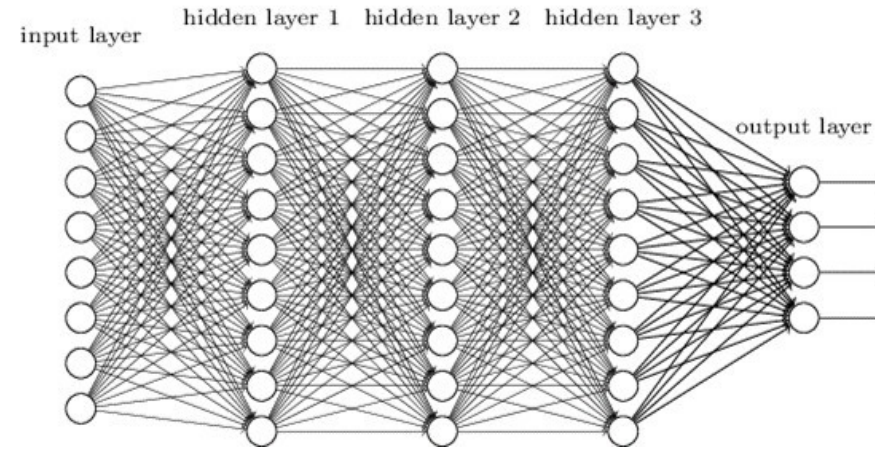
(Now) 2-layer Neural Network
or 3-layer Neural Network $f = \mathbf{W}_2\phi(\mathbf{W}_1x)$

$$f = \mathbf{W}_3\phi(\mathbf{W}_2\phi(\mathbf{W}_1x))$$

Defining “depth”



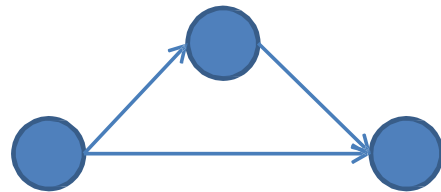
Deep neural network



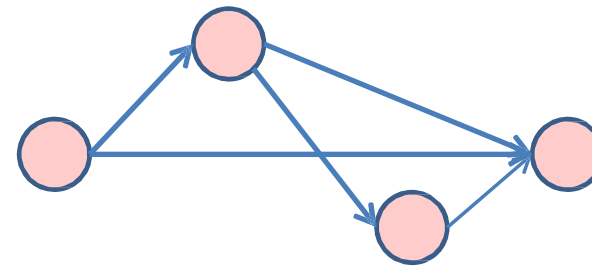
- What is a “deep” network

Deep Structures

- In any directed network of computational elements with input source nodes and output sink nodes, “depth” is the length of the longest path from a source to a sink



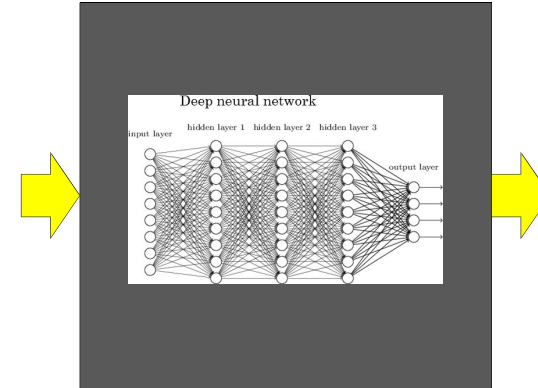
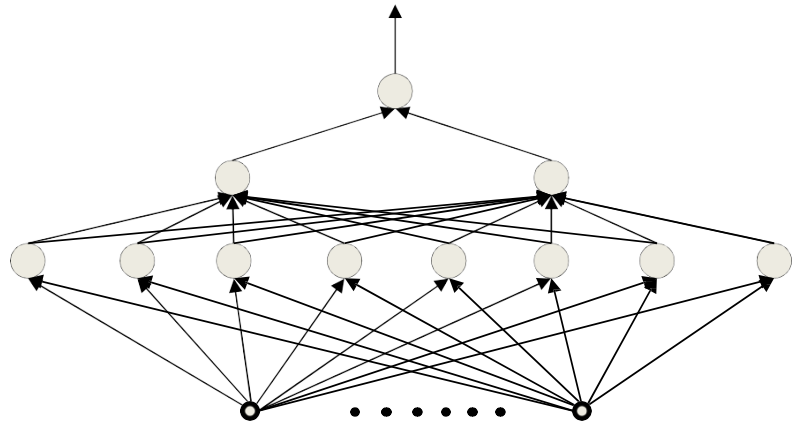
- Left: Depth = 2.



- Right: Depth = 3

- “Deep” \Leftrightarrow Depth > 2

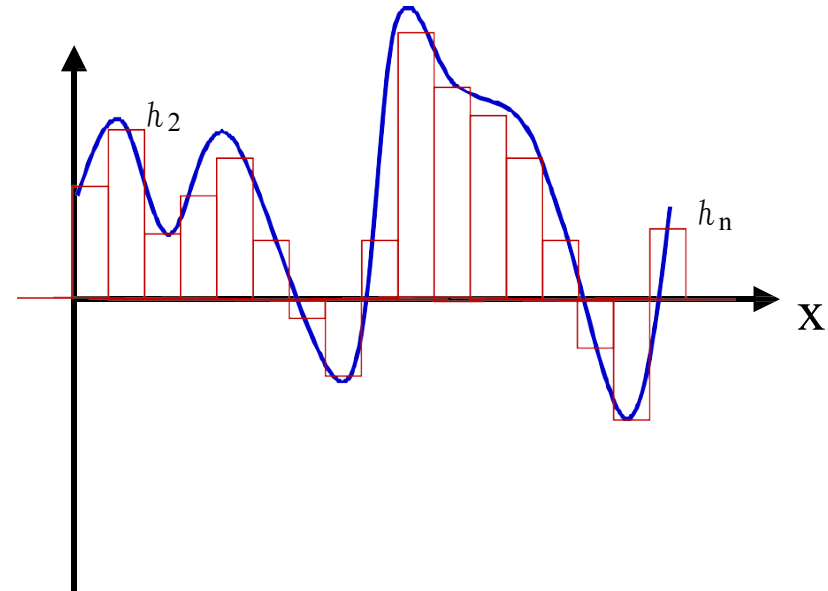
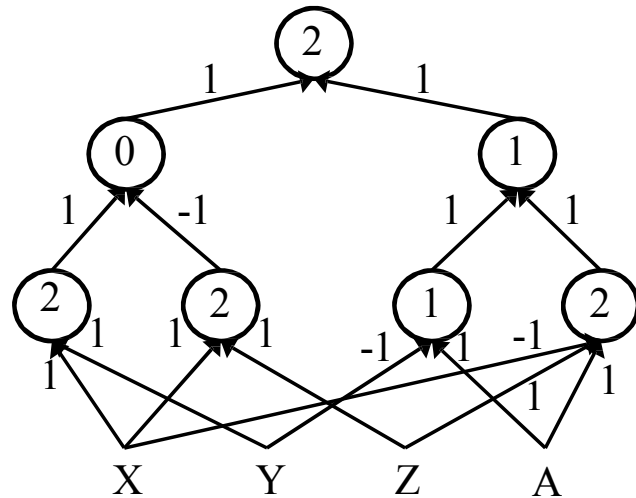
The multi-layer perceptron



- Inputs are real or Boolean stimuli
- Outputs are real or Boolean values
 - Can have multiple outputs for a single input
- What can this network compute?
 - What kinds of input/output relationships can it model?

MLPs approximate functions

$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | (\bar{X} \& \bar{Z}))$$



- MLP s can compose Boolean functions
- MLPs can compose real-valued functions
- What are the limitations?

MLP

- Multi-layer Perceptrons as universal Boolean functions
 - The need for depth
- MLPs as universal classifiers
 - The need for depth
- MLPs as universal approximators
- A discussion of optimal depth and width

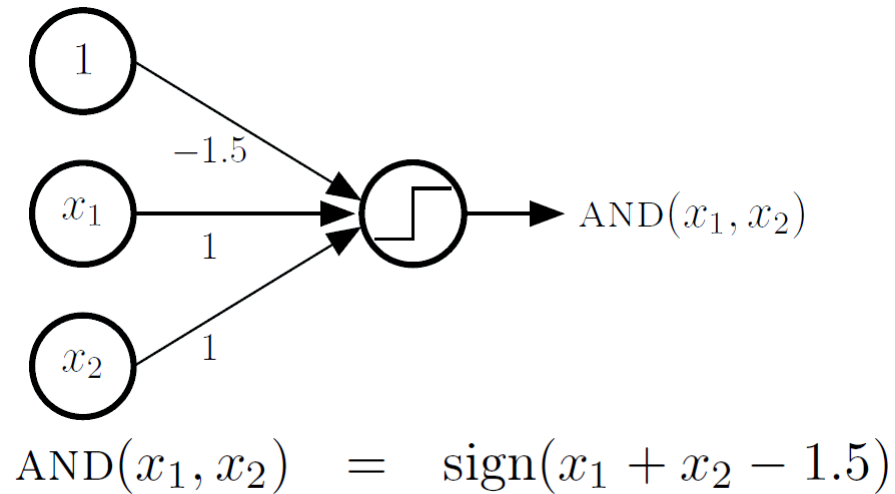
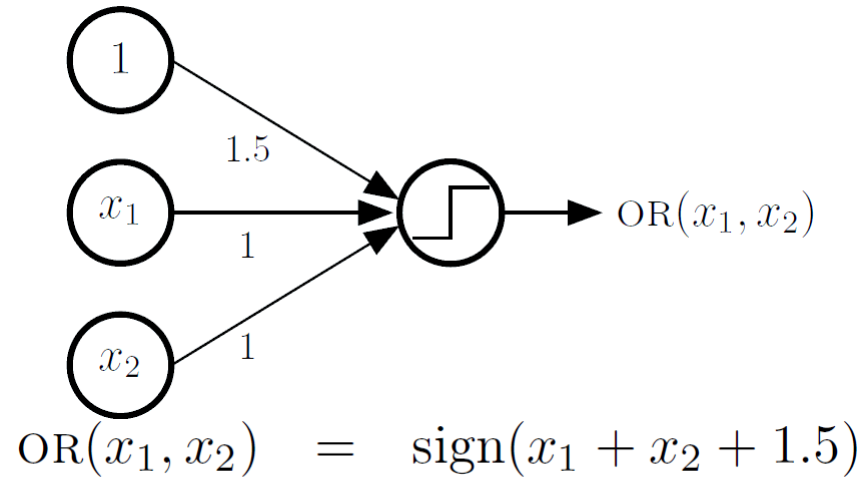
MLP

- Multi-layer Perceptrons as universal Boolean functions
 - The need for depth
- MLPs as universal classifiers
 - The need for depth
- MLPs as universal approximators
- A discussion of optimal depth and width

Multi-layer Perceptrons as universal Boolean functions

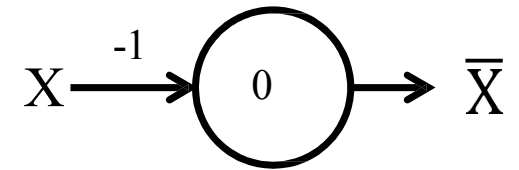
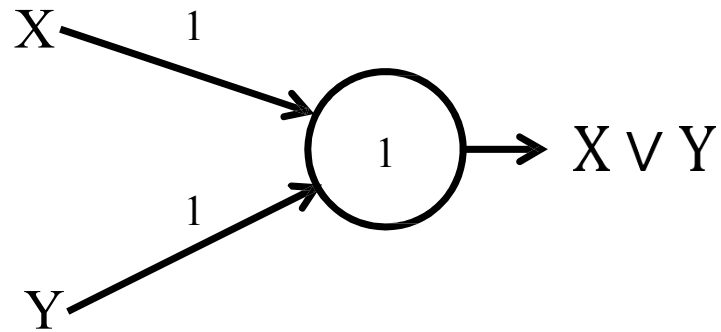
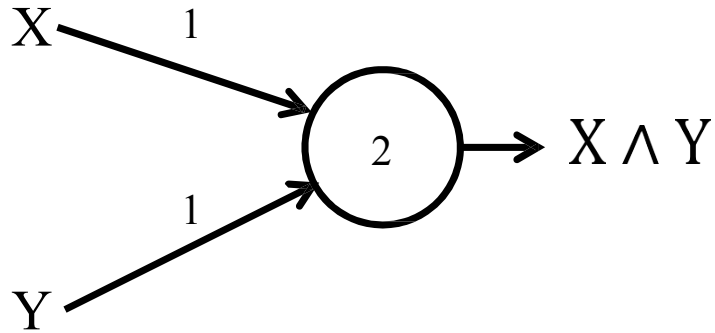
AND & OR networks

- For -1 and 1 inputs:



The perceptron as a Boolean gate

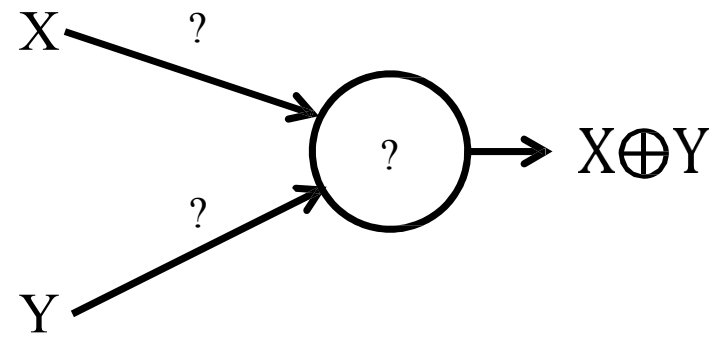
- For 0 and 1 inputs:



Values in the circles are thresholds
Values on edges are weights

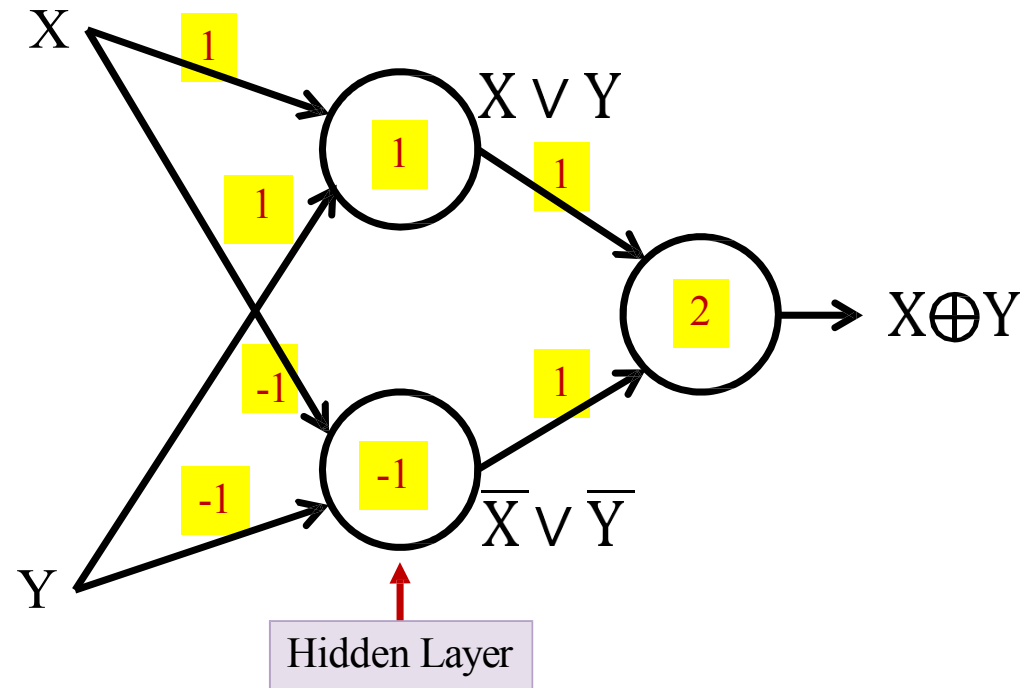
- A perceptron can model any simple binary Boolean gate

The perceptron is not enough



- Cannot compute an XOR

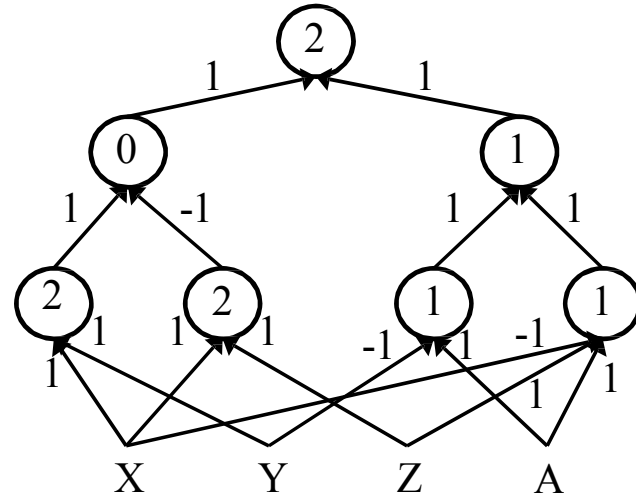
Multi-layer perceptron XOR



- An XOR takes three neurons

Multi-layer perceptron

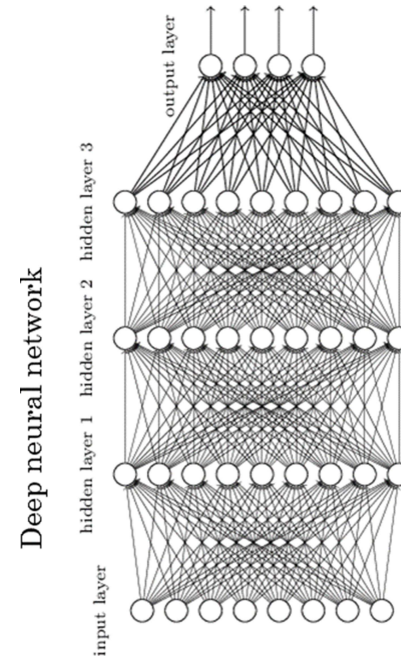
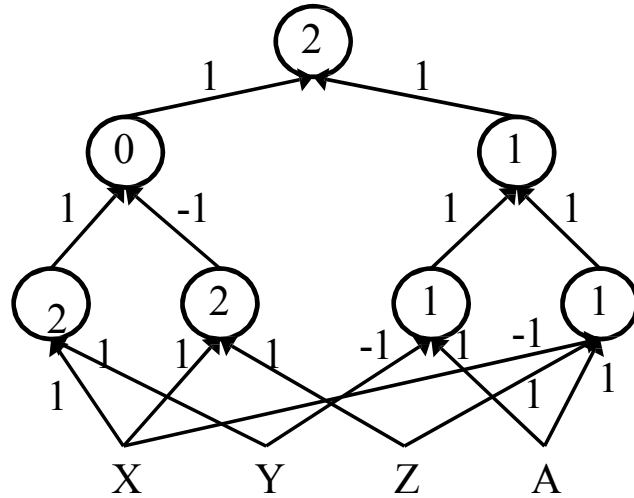
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | \overline{(X \& Z)})$$



- MLPs can compute more complex Boolean functions
- MLPs can compute any Boolean function
 - Since they can emulate individual gates
- MLPs are universal Boolean functions

MLP as Boolean Functions

$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \& Y) | \overline{(X \& Z)})$$



- MLPs are universal Boolean functions
 - Any function over any number of inputs and any number of outputs
- But **how many “layers”** will they need?

How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$

- Expressed in disjunctive normal form

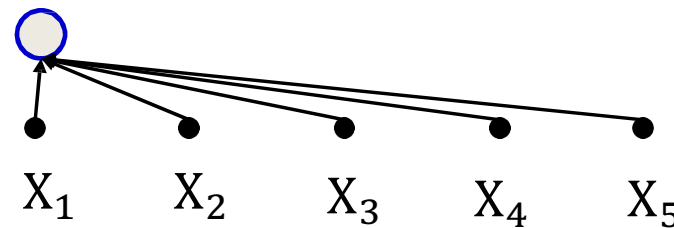
How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

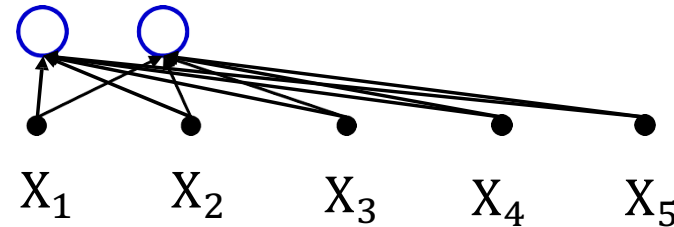
How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

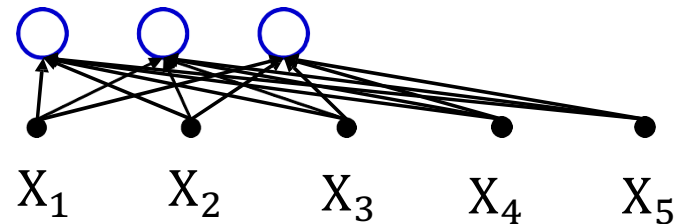
How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



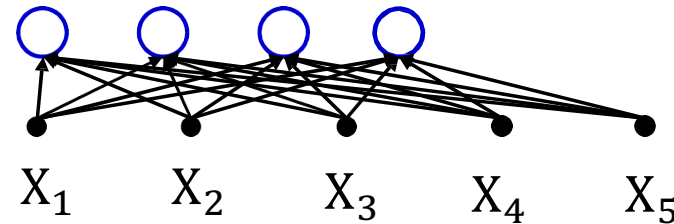
- Expressed in disjunctive normal form

How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



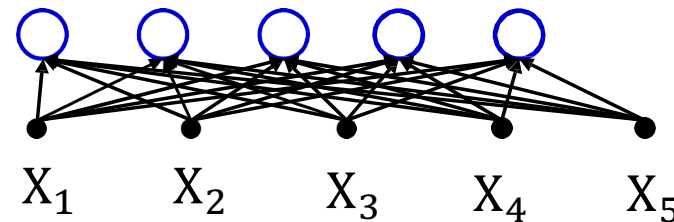
- Expressed in disjunctive normal form

How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + \boxed{X_1 \bar{X}_2 X_3 X_4 X_5} + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

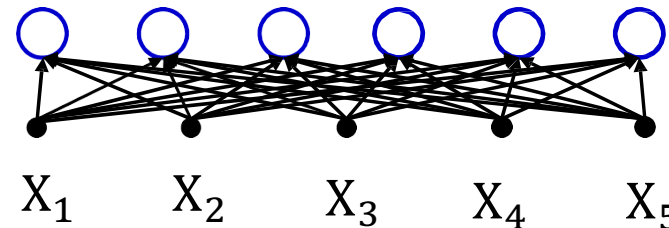
How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



- Expressed in disjunctive normal form

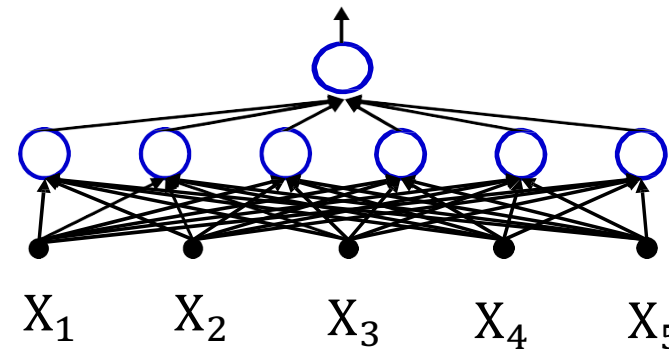
How many layers for a Boolean MLP?

Truth table shows all input combinations for which output is 1

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1 \bar{X}_2 X_3 X_4 \bar{X}_5 + \bar{X}_1 X_2 \bar{X}_3 X_4 X_5 + \bar{X}_1 X_2 X_3 \bar{X}_4 \bar{X}_5 + X_1 \bar{X}_2 \bar{X}_3 \bar{X}_4 X_5 + X_1 \bar{X}_2 X_3 X_4 X_5 + X_1 X_2 \bar{X}_3 \bar{X}_4 X_5$$



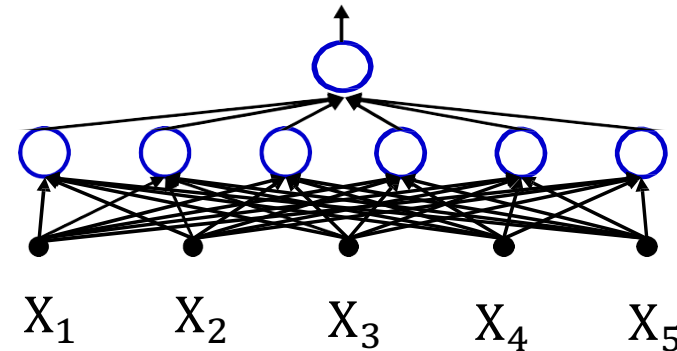
- Expressed in disjunctive normal form

How many layers for a Boolean MLP?

Truth Table

X_1	X_2	X_3	X_4	X_5	Y
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1

$$y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



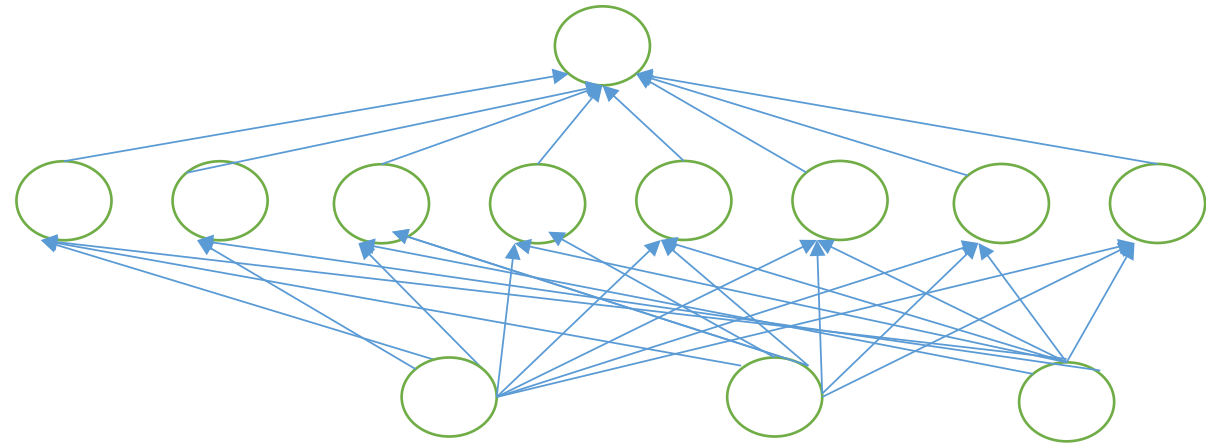
- Any truth table can be expressed in this manner!
- A one-hidden-layer MLP is a Universal Boolean Function
- But what is the largest number of perceptrons required in the single hidden layer for an N-input-variable function?

Using deep network: Parity function on N inputs

- Simple MLP with one hidden layer:

2^{N-1} Hidden units

$(N + 2)2^{N-1} + 1$ Weights and biases



Using deep network: Parity function on N inputs

- Simple MLP with one hidden layer:

2^{N-1} Hidden units

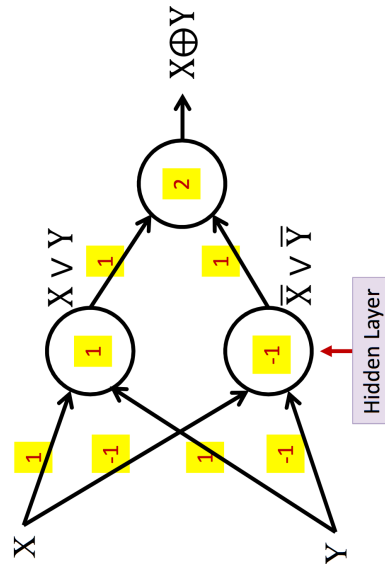
$(N + 2)2^{N-1} + 1$ Weights and biases

- $f = X_1 \oplus X_2 \oplus \dots \oplus X_N$

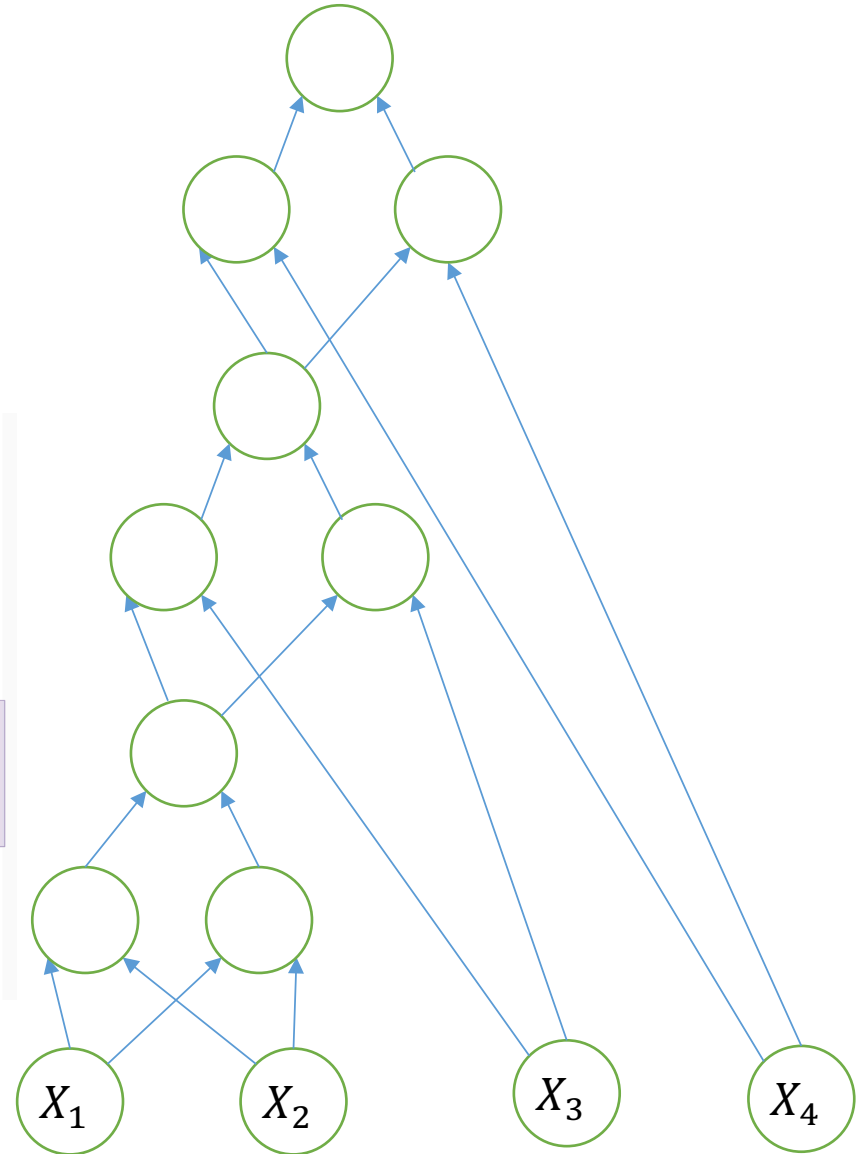
$3(N - 1)$ Nodes

$9(N - 1)$ Weights and biases

Depth is linear in N



The actual number of parameters in a network is the number that really matters in software or hardware implementations



Network size: summary

- An MLP is a universal Boolean function
- But can represent a given function only if
 - It is sufficiently wide
 - It is sufficiently deep
 - Depth can be traded off for (sometimes) exponential growth of the width of the network
- Optimal width and depth depend on the number of variables and the complexity of the Boolean function
 - Complexity: minimal number of terms in DNF formula to represent it

Summary: Wide vs. deep network

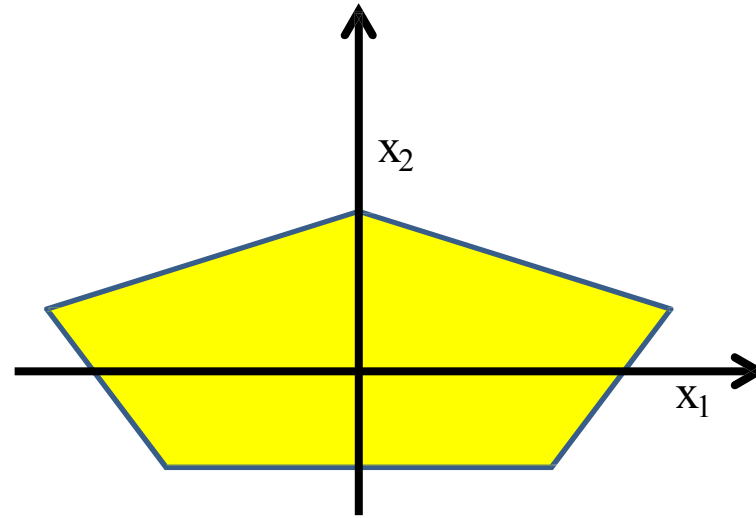
- MLP with a single hidden layer is a universal Boolean function
- Even a network with a single hidden layer is a universal Boolean machine
 - But a single-layer network may require an exponentially large number of perceptrons
- Deeper networks may require far fewer neurons than shallower networks
 - Could be exponentially smaller

MLPs as universal classifiers

MLP

- Multi-layer Perceptrons as universal Boolean functions
 - The need for depth
- MLPs as universal classifiers
 - The need for depth
- MLPs as universal approximators
- A discussion of optimal depth and width

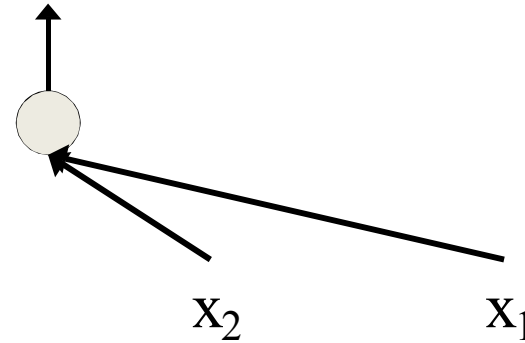
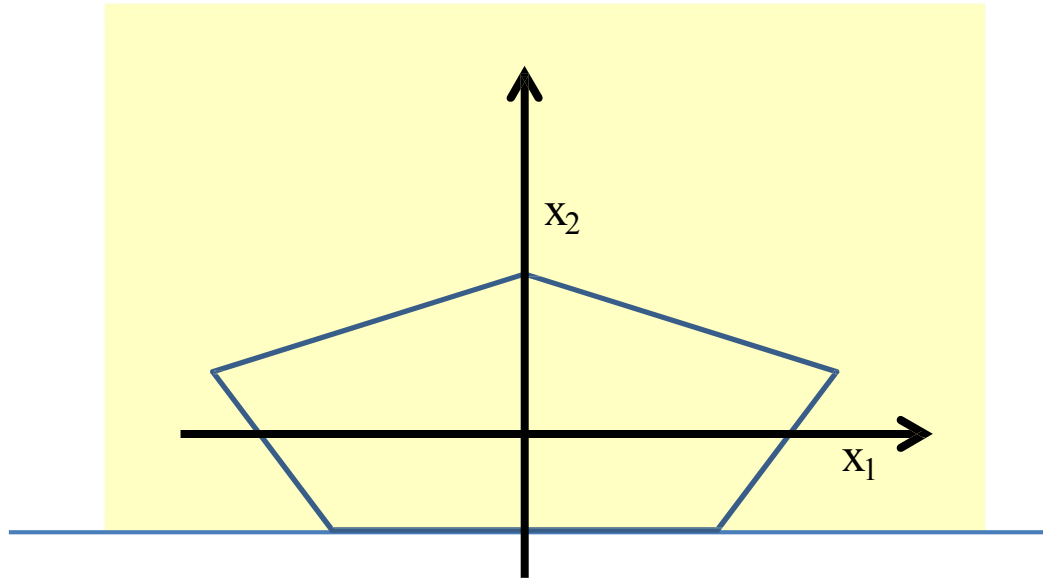
Composing complicated “decision” boundaries



Can now be composed into “networks” to compute arbitrary classification “boundaries”

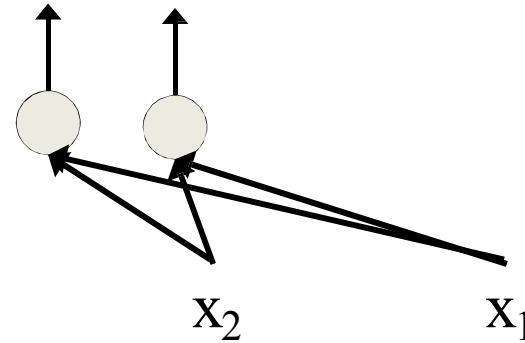
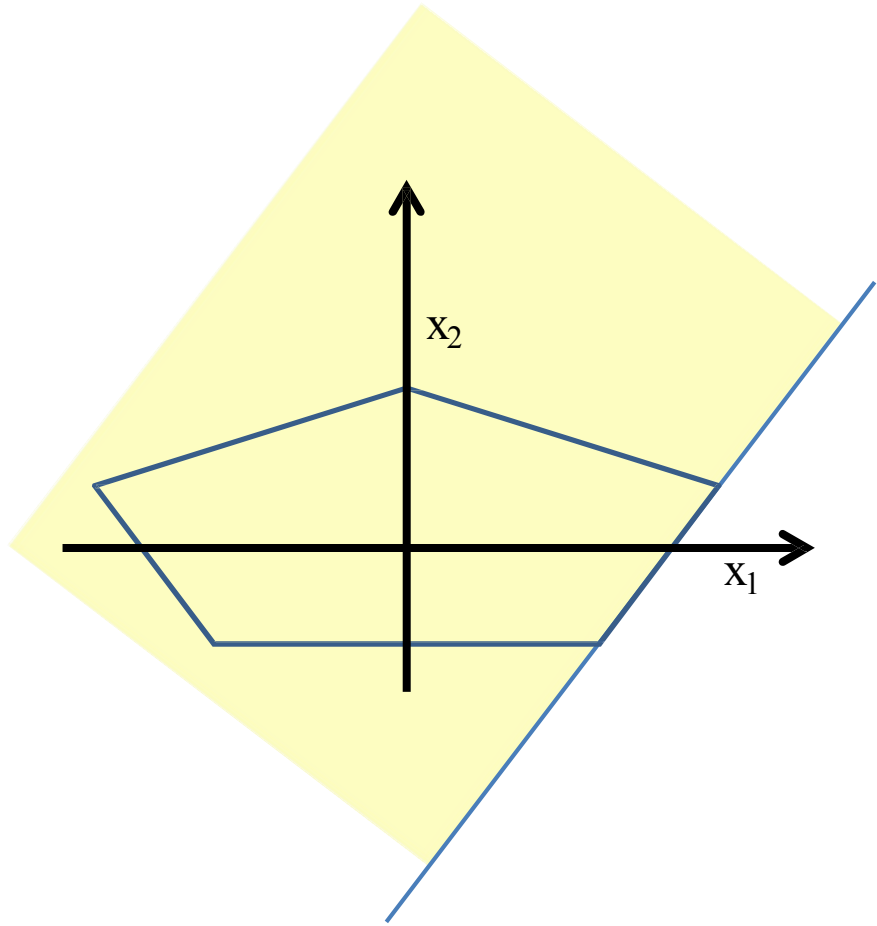
- Build a network of units with a single output that fires if the input is in the coloured area

Booleans over the reals



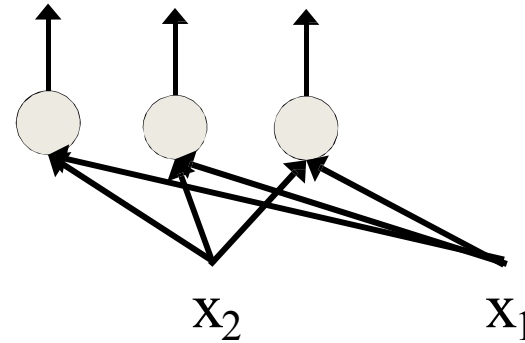
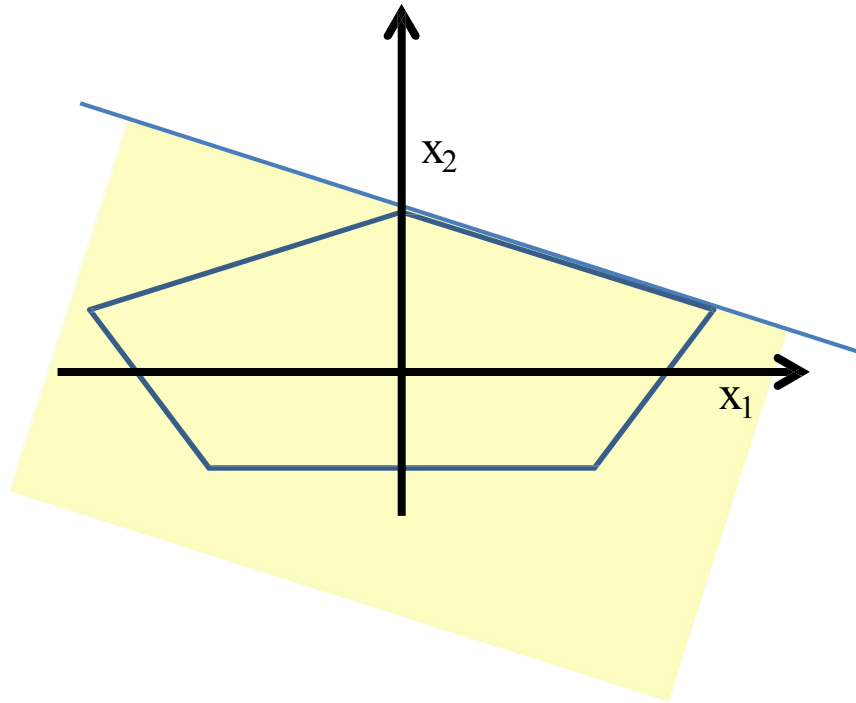
- The network must fire if the input is in the coloured area

Booleans over the reals



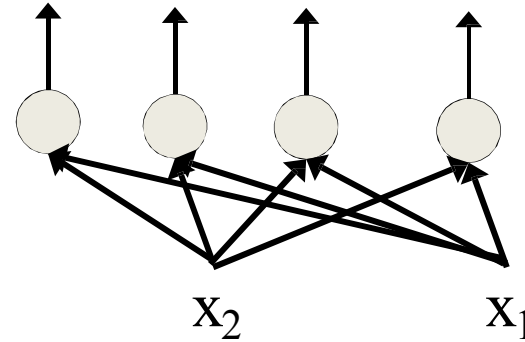
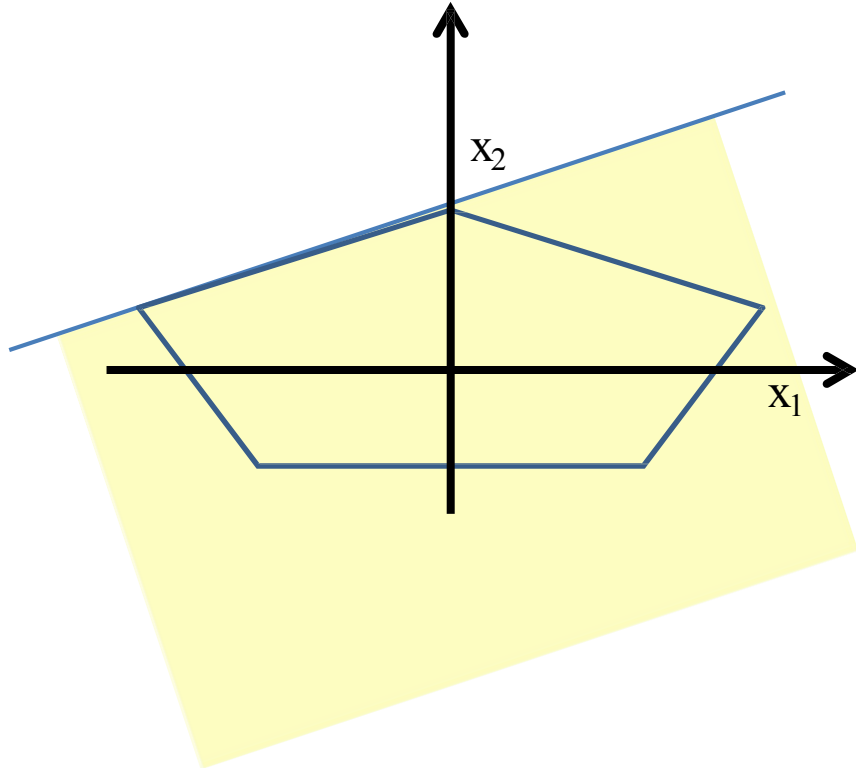
- The network must fire if the input is in the coloured area

Booleans over the reals



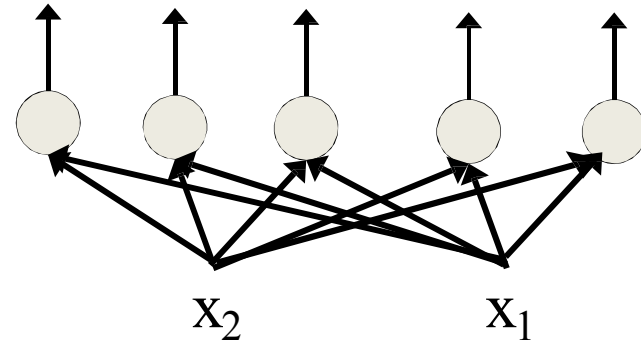
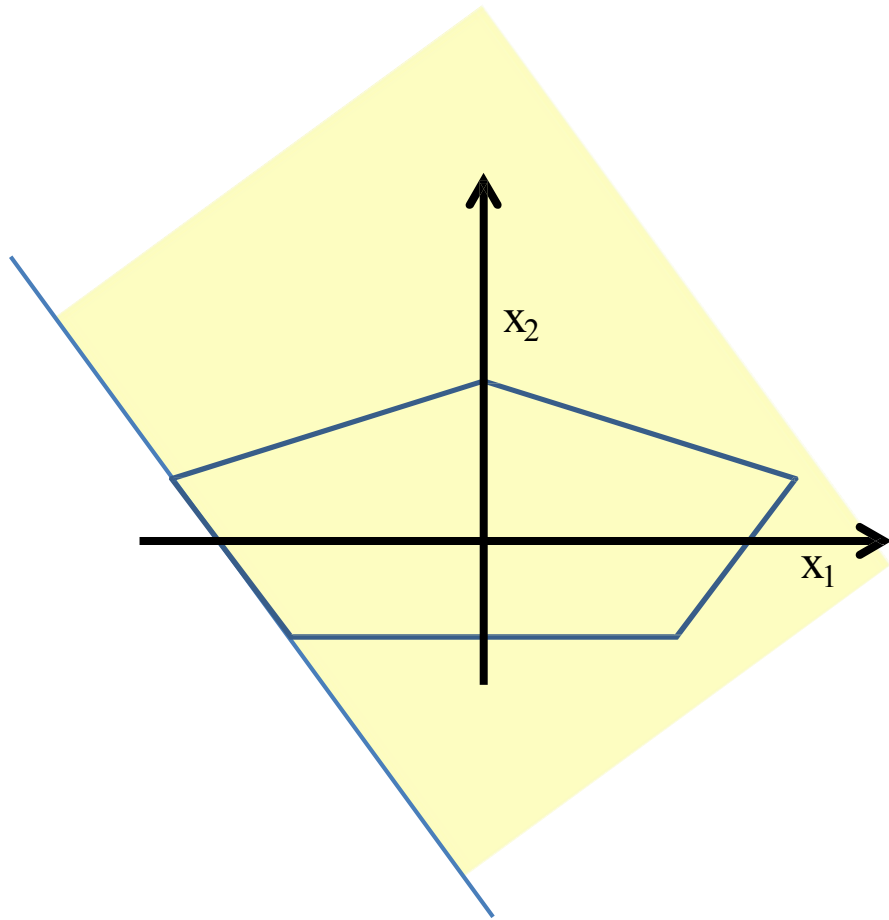
- The network must fire if the input is in the coloured area

Booleans over the reals



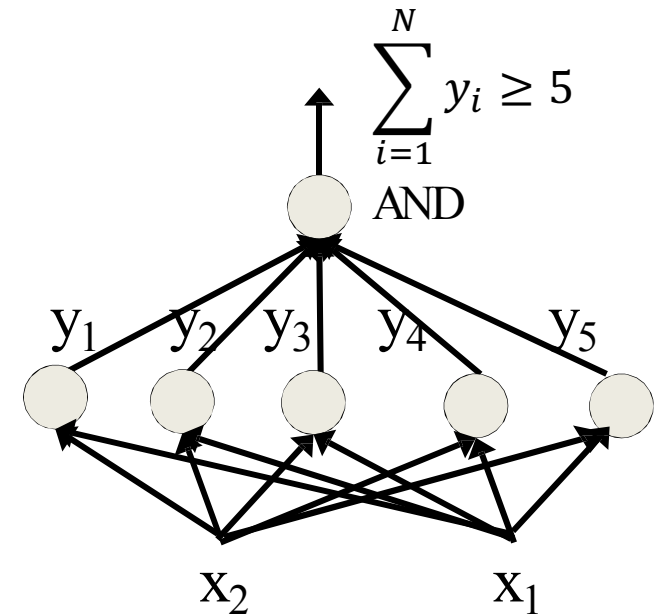
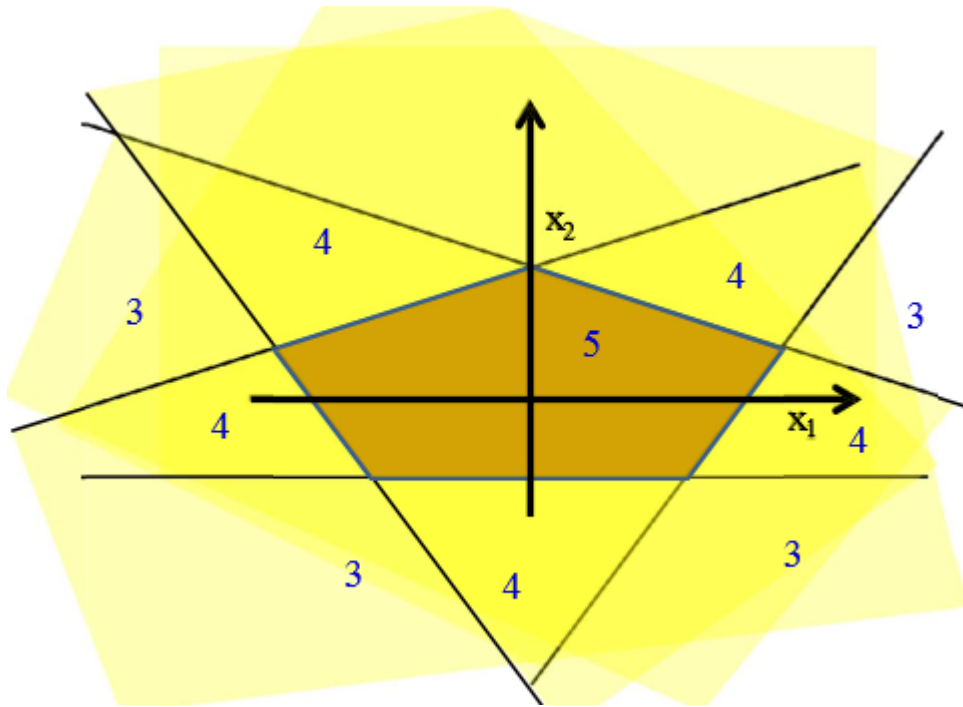
- The network must fire if the input is in the coloured area

Booleans over the reals



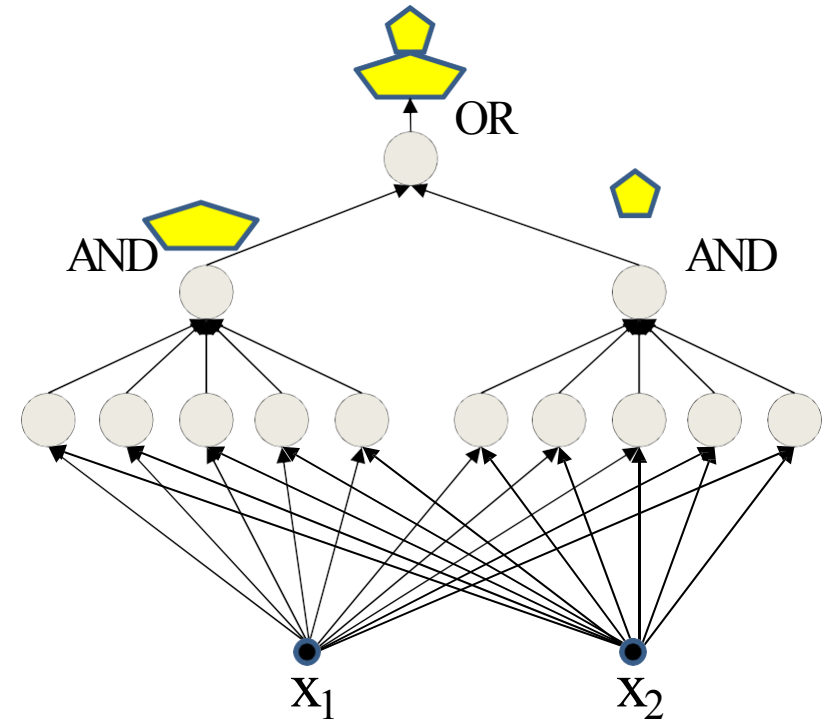
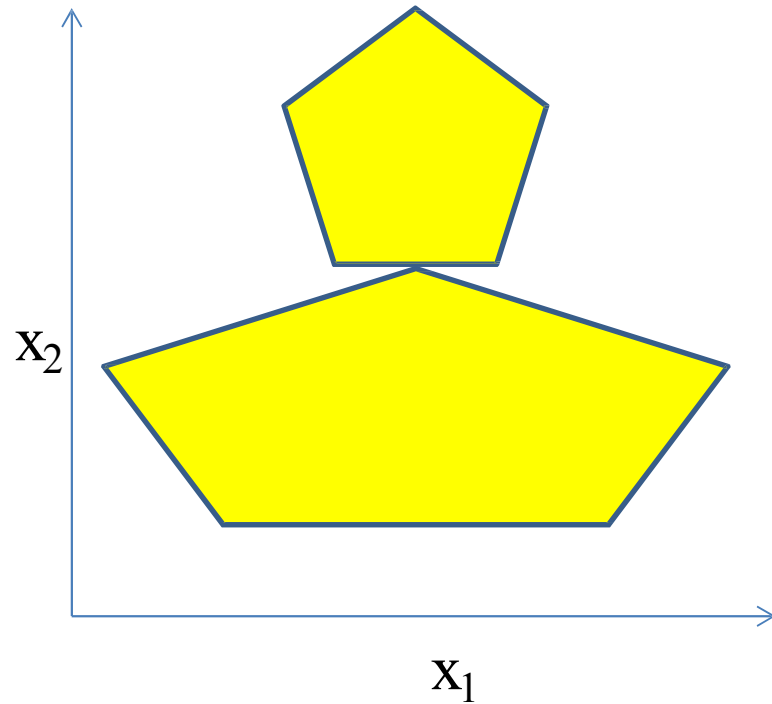
- The network must fire if the input is in the coloured area

Booleans over the reals



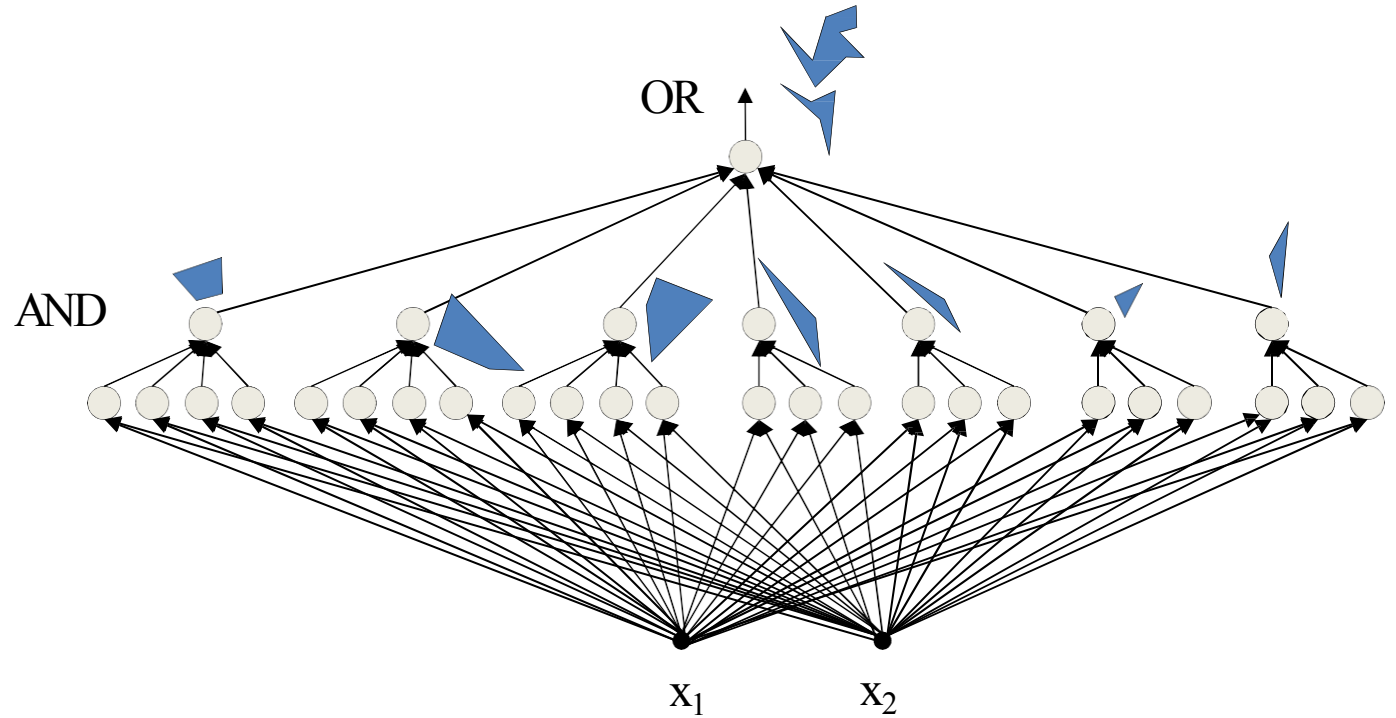
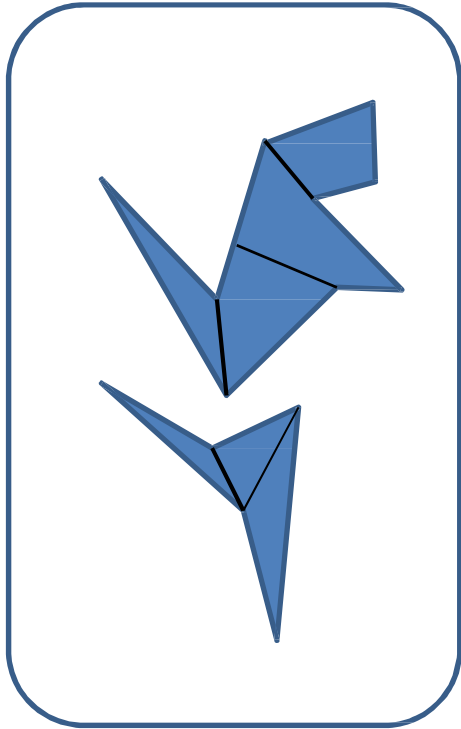
- The network must fire if the input is in the coloured area

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
 - With only one hidden layer!
 - How?

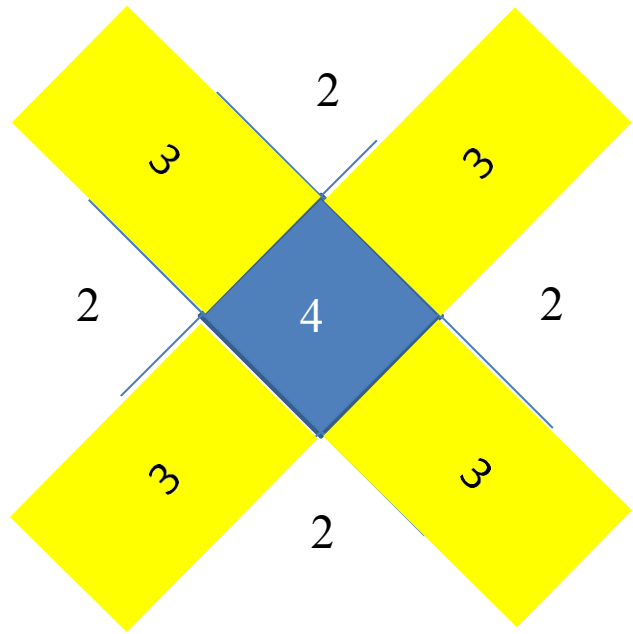
MLP with Different Number of Layers

MLP with unit step activation function

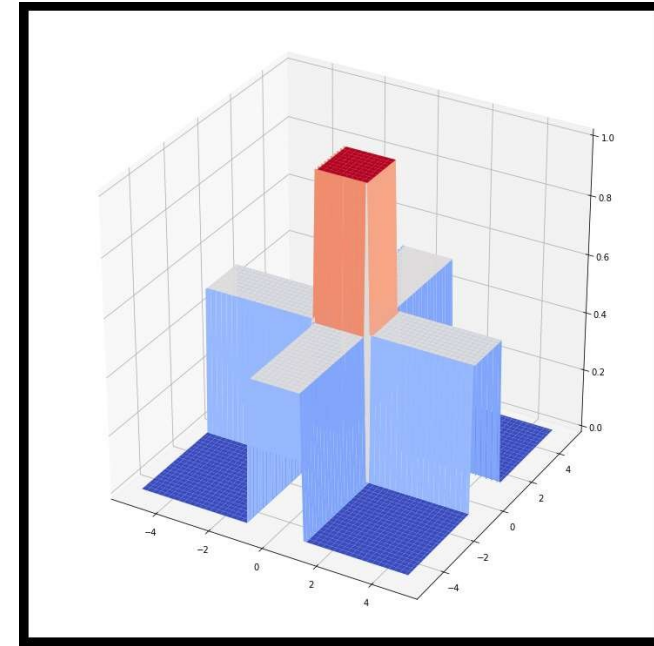
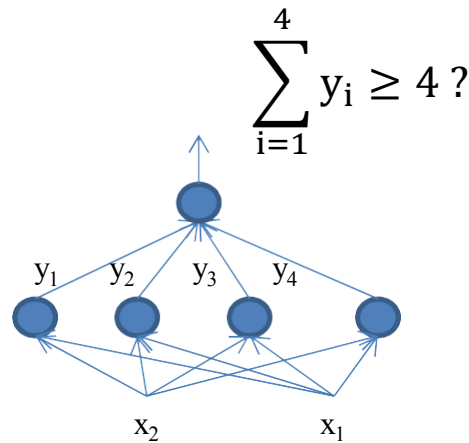
Decision region found by an output unit.

Structure	Type of Decision Regions	Interpretation	Example of region
Single Layer (no hidden layer)	Half space	Region found by a hyper-plane	
Two Layer (one hidden layer)	Polyhedral (open or closed) region	Intersection of half spaces	
Three Layer (two hidden layers)	Arbitrary regions	Union of polyhedrals	

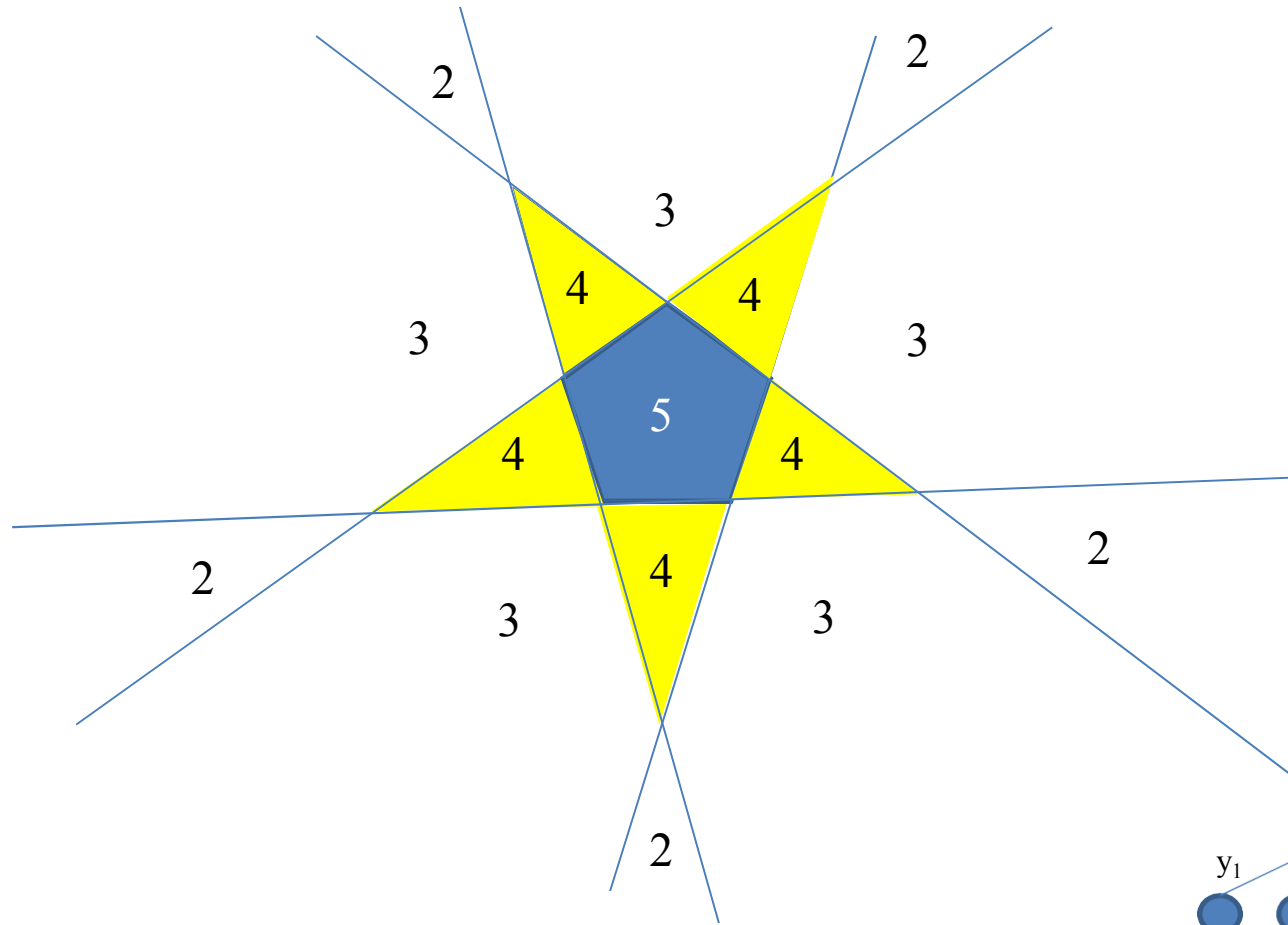
Composing a square decision boundary



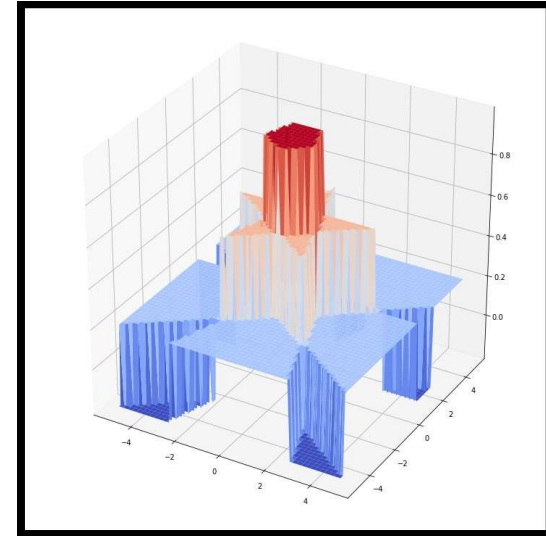
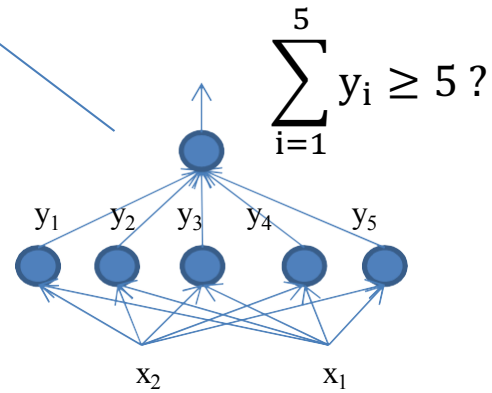
- The polygon net



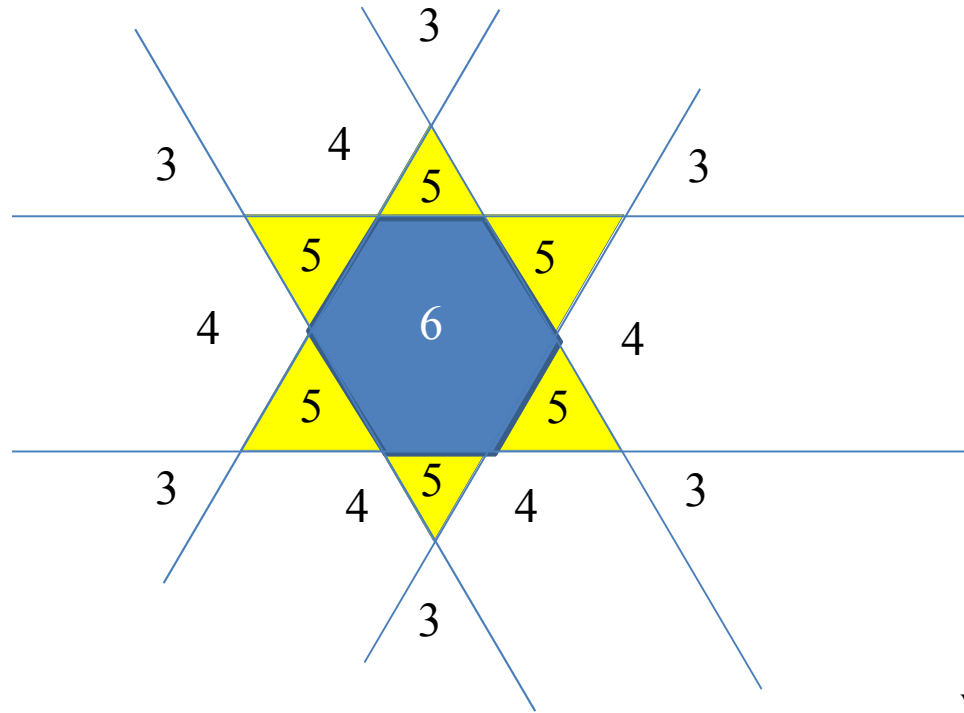
Composing a square decision boundary



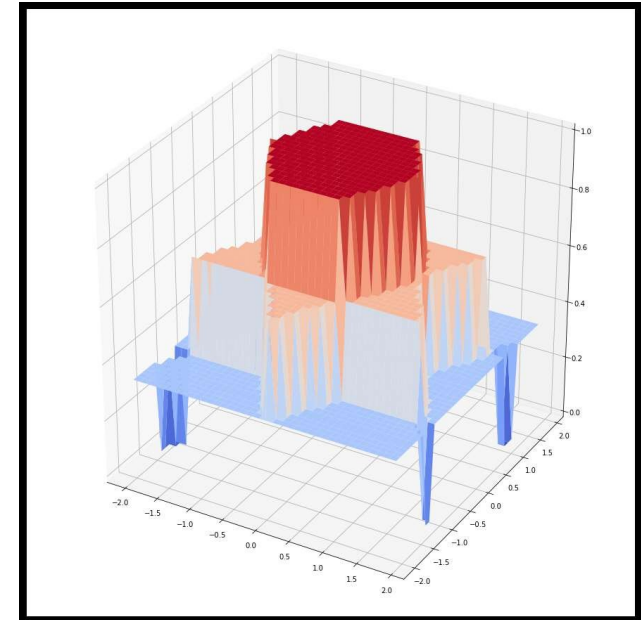
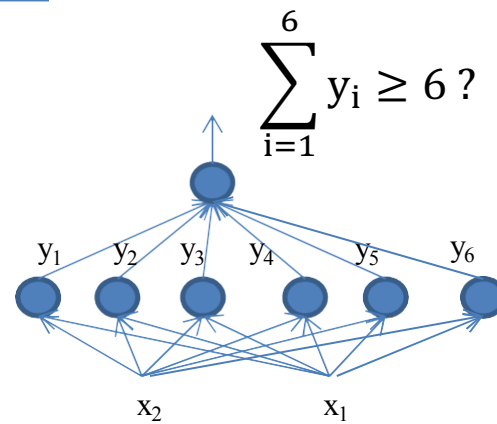
- The polygon net



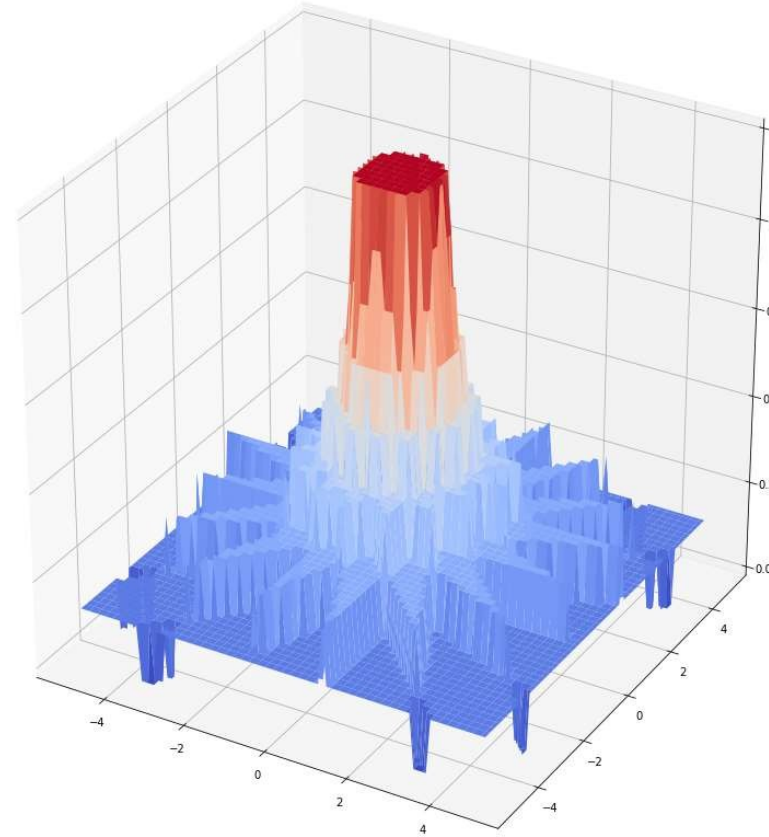
Composing a pentagon



- The polygon net

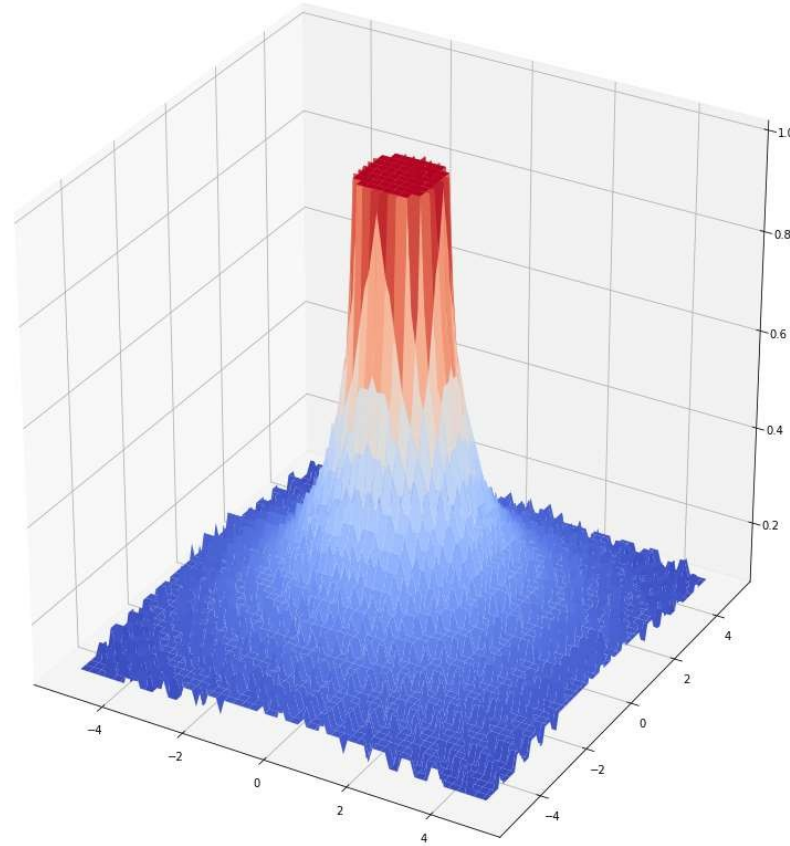


16 sides



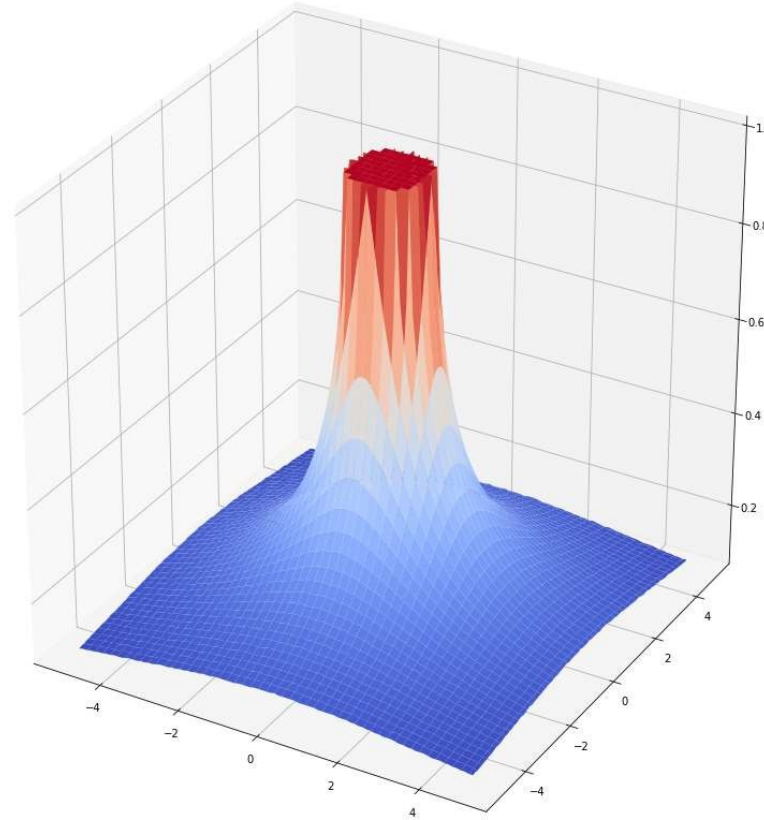
- What are the sums in the different regions?

64 sides



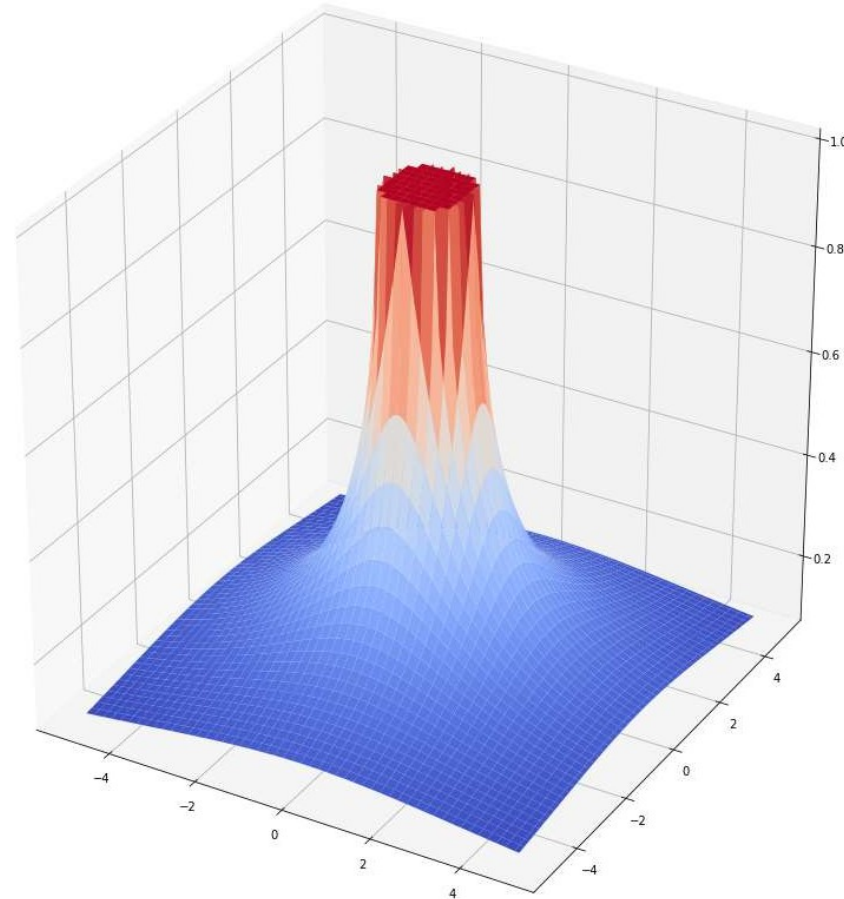
- What are the sums in the different regions?

1000 sides



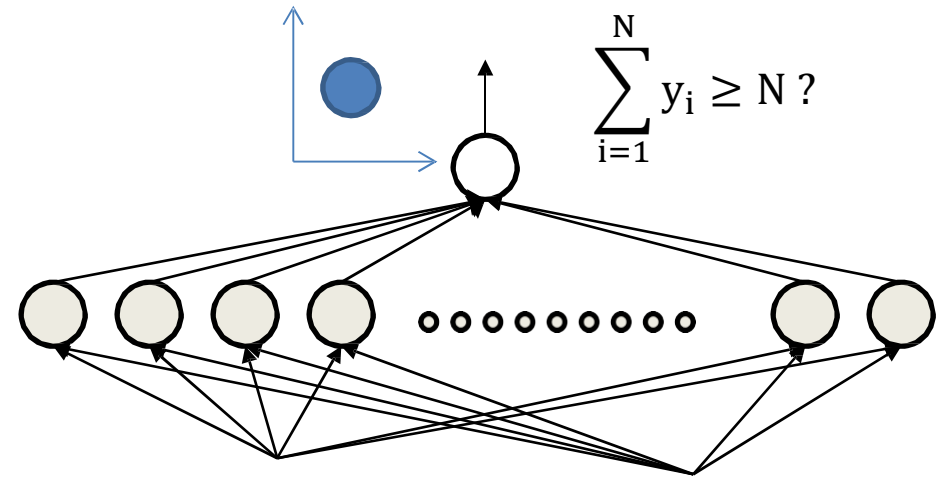
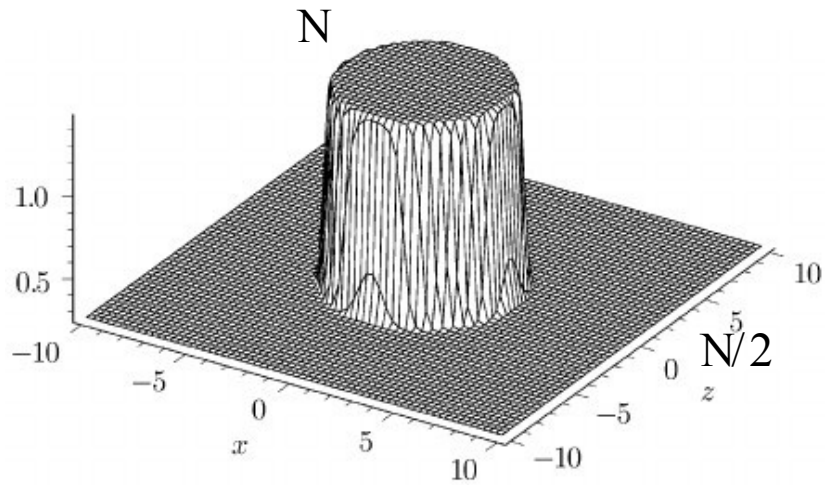
- What are the sums in the different regions?

Polygon net



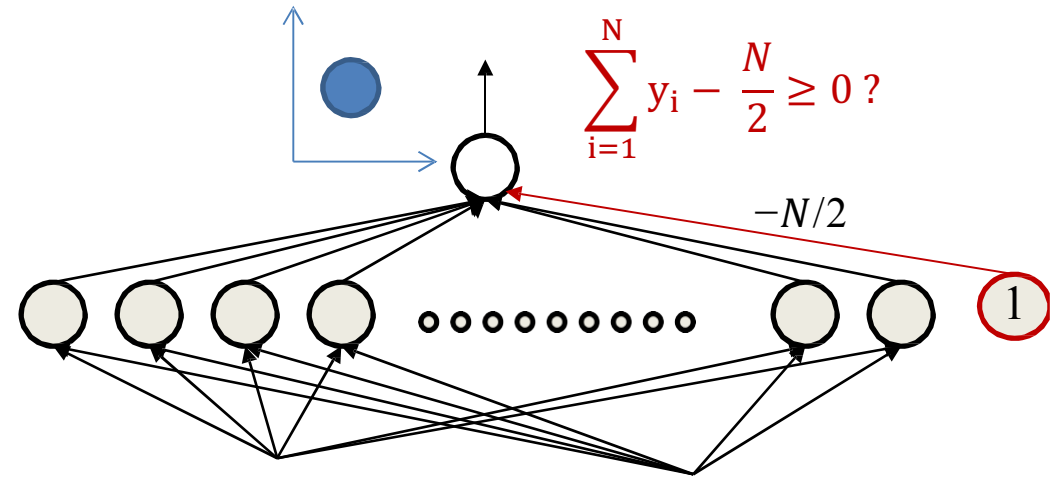
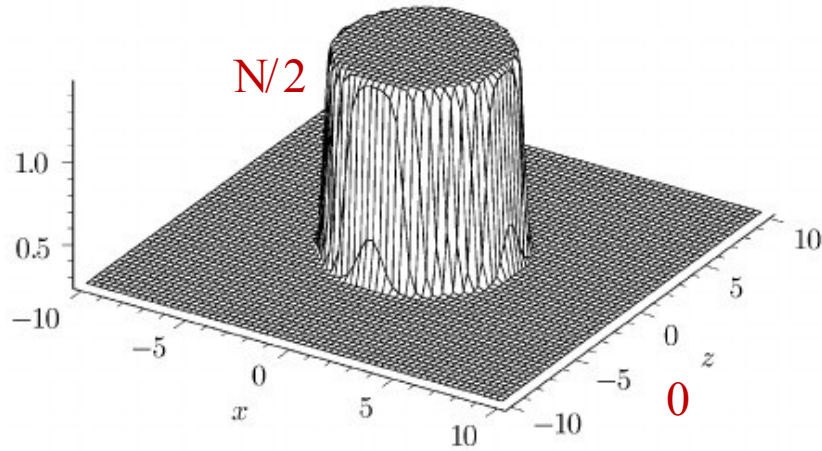
- Increasing the number of sides reduces the area outside the polygon that have $N/2 < \text{Sum} < N$

Composing a circle



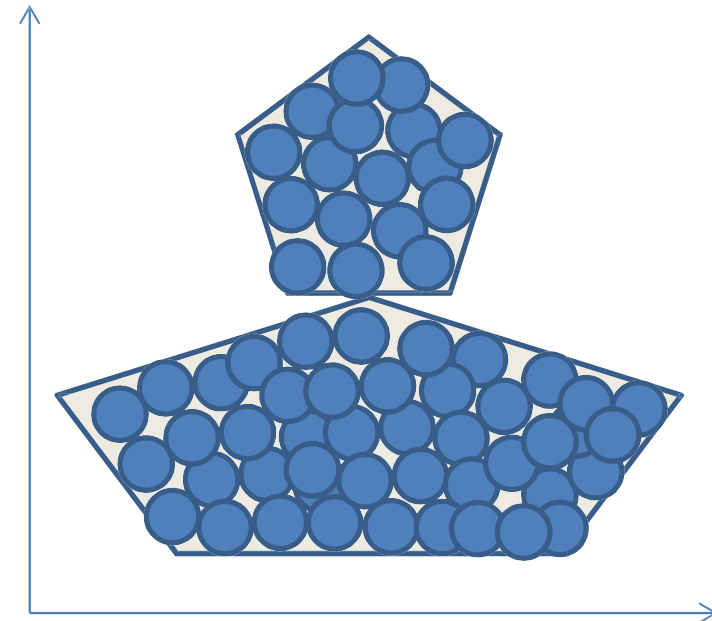
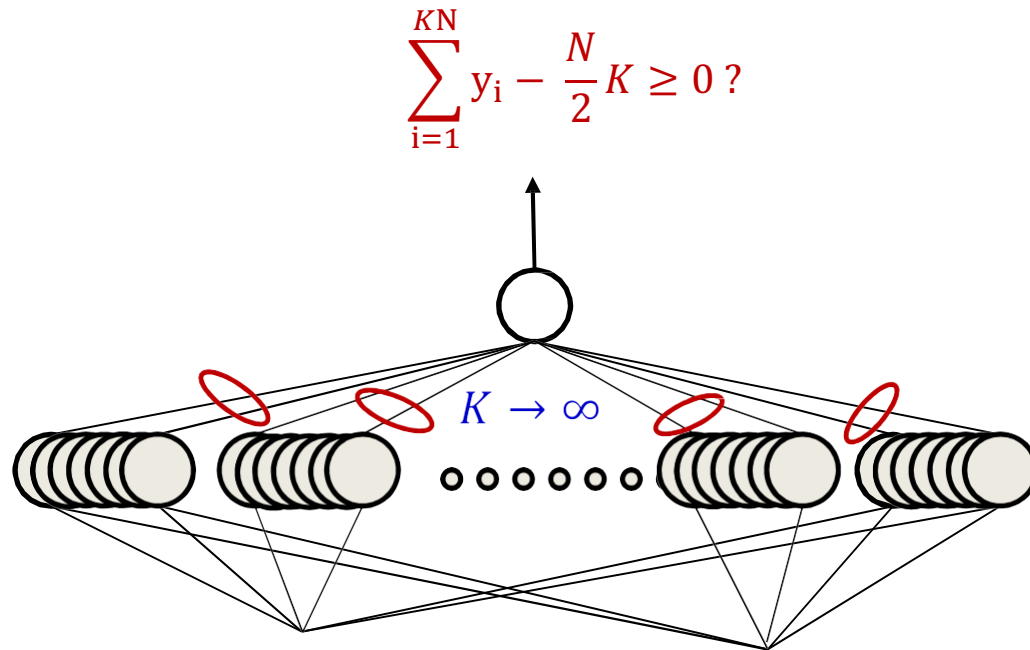
- The circle net
 - Very large number of neurons
 - Sum is **N inside** the circle, **N/2 outside almost everywhere**
 - Circle can be at any location

Composing a circle



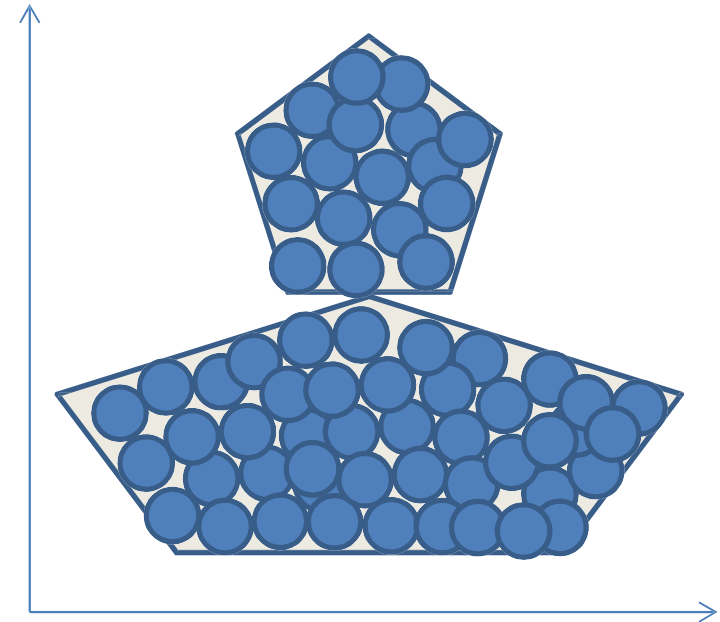
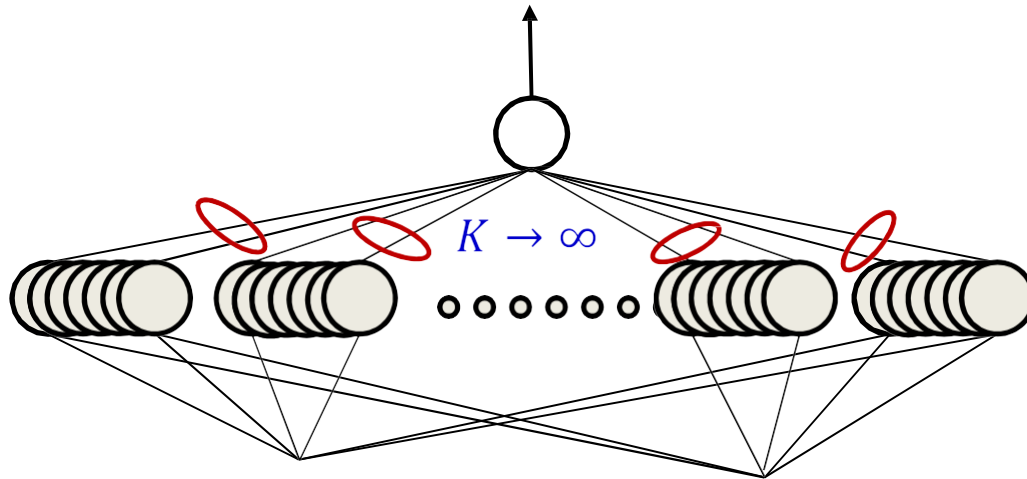
- The circle net
 - Very large number of neurons
 - Sum is $N/2$ inside the circle, 0 outside almost everywhere
 - Circle can be at any location

Composing an arbitrary figure



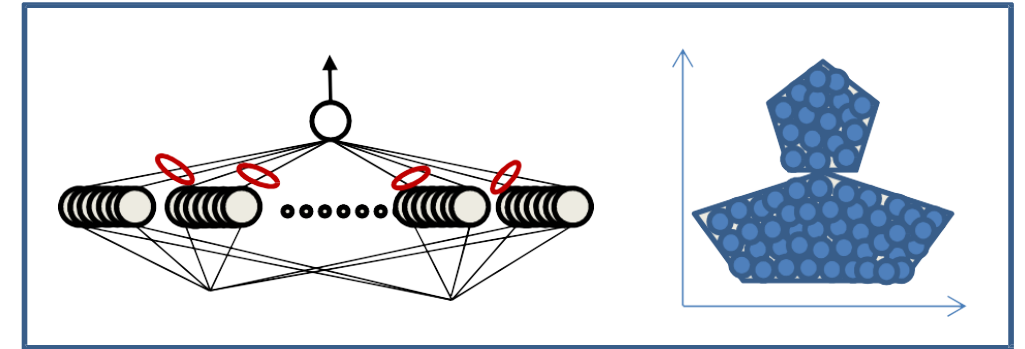
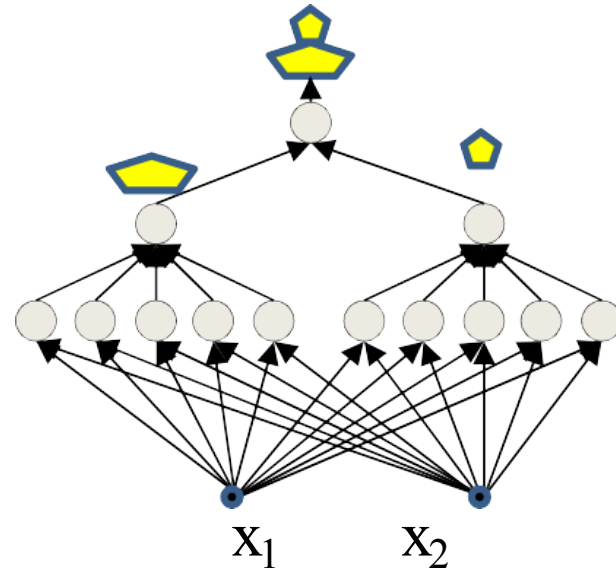
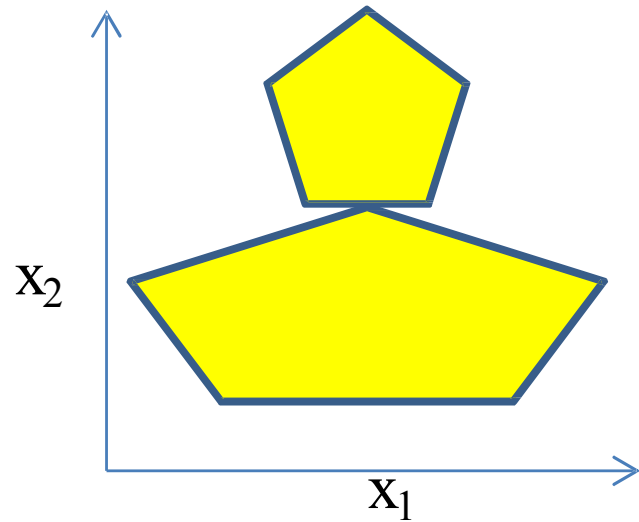
- Just fit in an arbitrary number of circles
 - More accurate approximation with greater number of smaller circles
 - Can achieve arbitrary precision

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

Summary

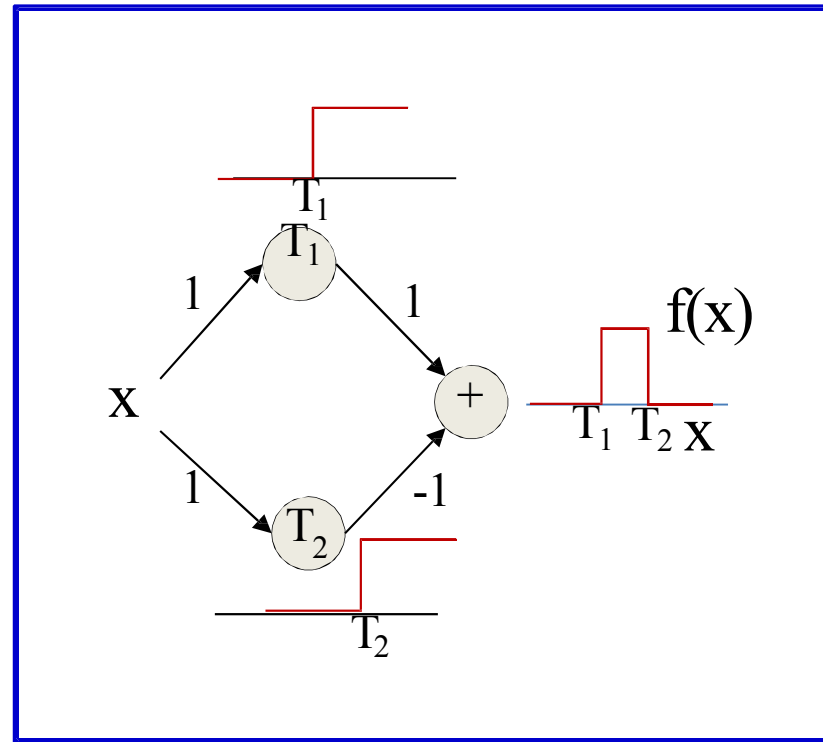
- Multi-layer perceptrons are Universal Boolean Machines
 - Even a network with a single hidden layer is a universal Boolean machine
- Multi-layer perceptrons are Universal Classification Functions
 - Even a network with a single hidden layer is a universal classifier
- But a single-layer network may require an exponentially large number of units 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

MLPs as universal approximators

MLP

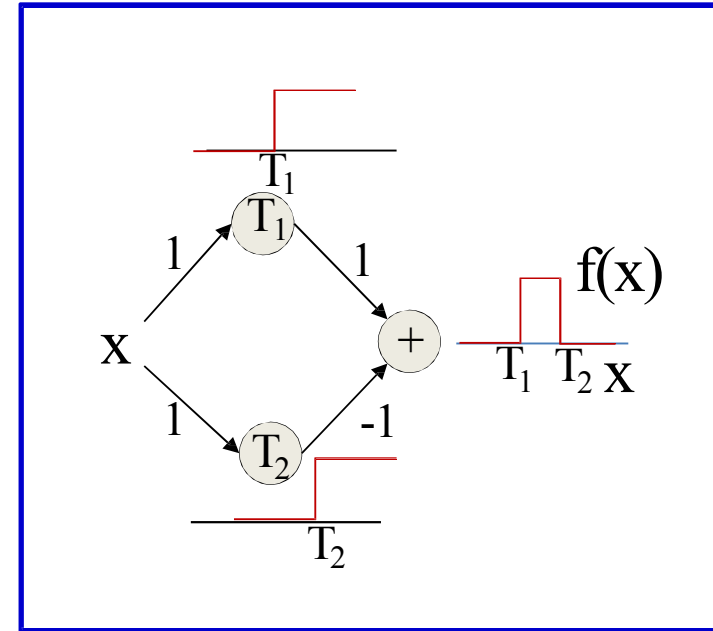
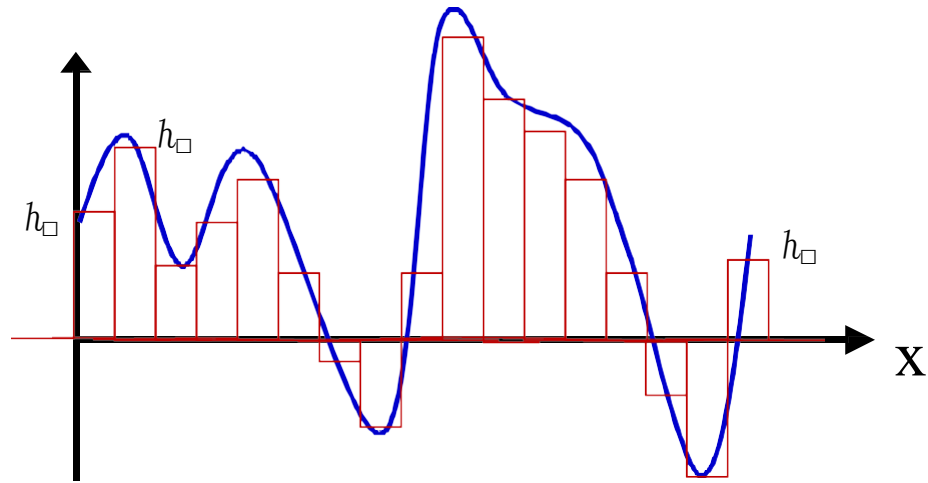
- Multi-layer Perceptrons as universal Boolean functions
 - The need for depth
- MLPs as universal classifiers
 - The need for depth
- MLPs as universal approximators
- A discussion of optimal depth and width

MLP as a continuous-valued regression



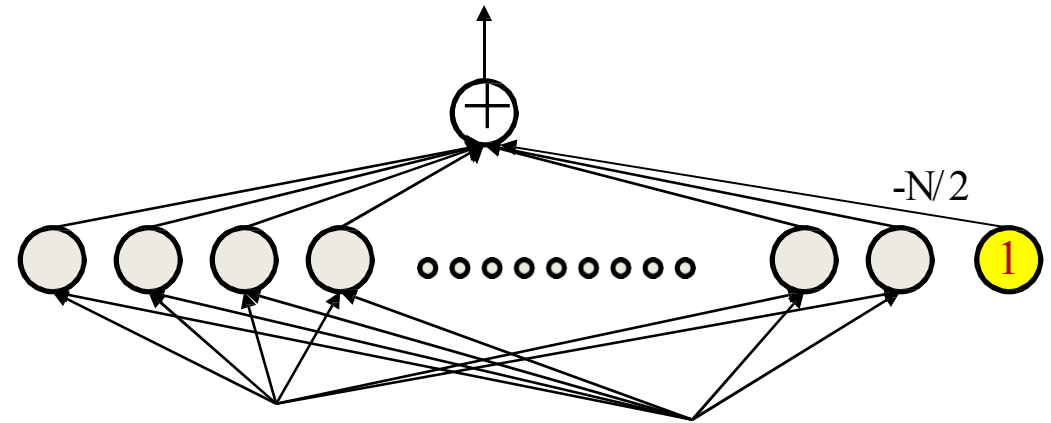
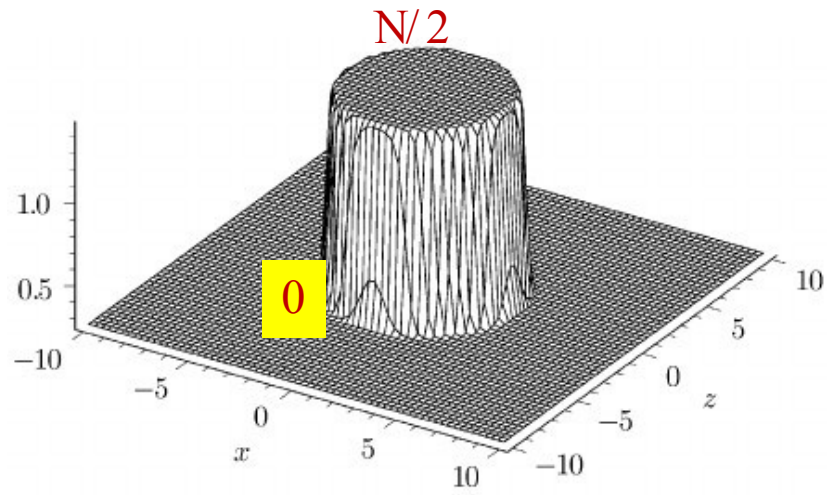
- A simple 3-unit MLP with a “summing” output unit can generate a “square pulse” over an input
 - Output is 1 only if the input lies between T_1 and T_2
 - T_1 and T_2 can be arbitrarily specified

MLP as a continuous-valued regression



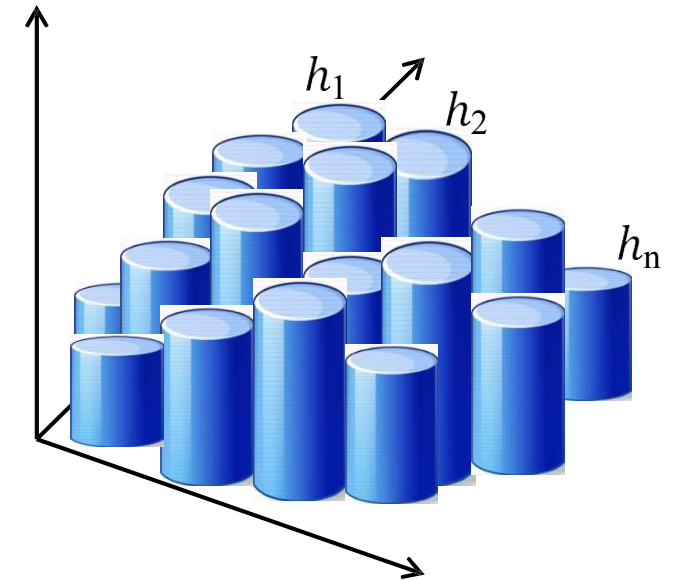
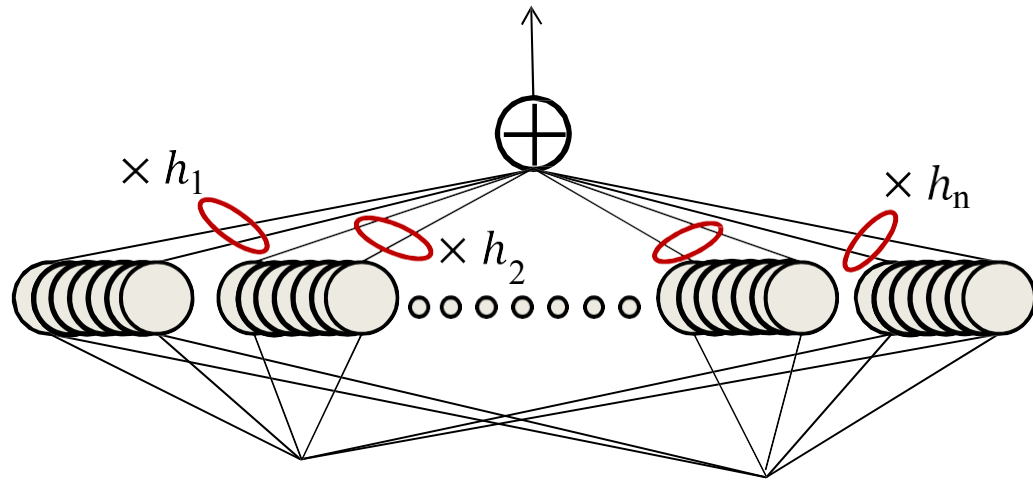
- A simple 3-unit MLP can generate a “square pulse” over an input
- An MLP with many units can model an arbitrary function over an input
 - To arbitrary precision
 - Simply make the individual pulses narrower
- A one-layer MLP can model an arbitrary function of a single input

For higher dimensions



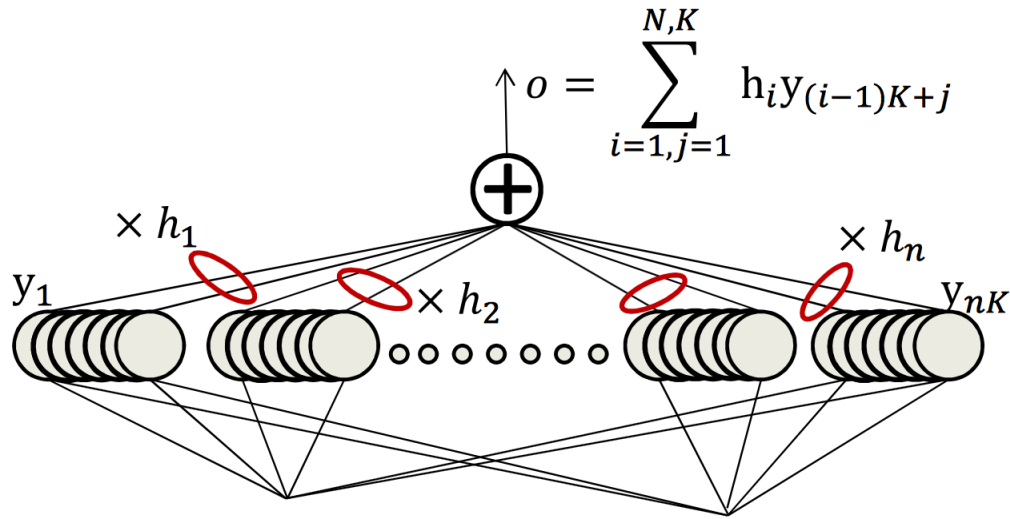
- An MLP can compose a cylinder
 - $\frac{N}{2}$ in the circle, 0 outside

MLP as a continuous-valued function

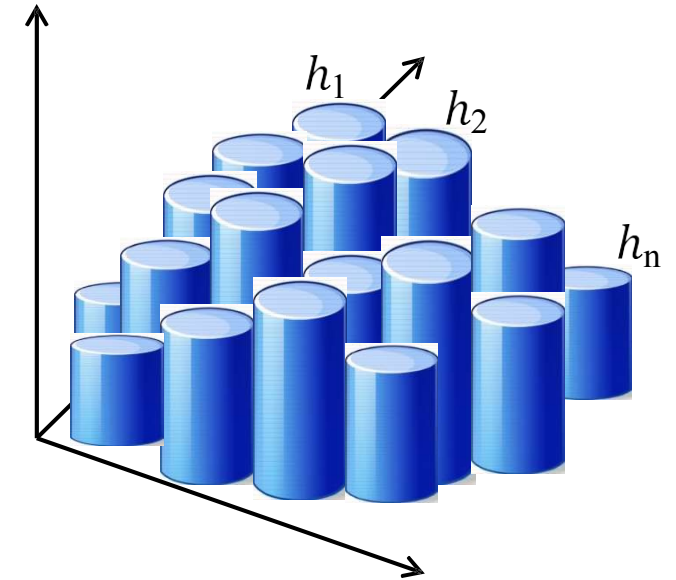


- MLPs can actually compose arbitrary functions in any number of dimensions!
 - Even with only one layer
 - As sums of scaled and shifted cylinders
 - To arbitrary precision
 - By making the cylinders thinner
 - The MLP is a universal approximator!

Caution: MLPs with additive output units are universal approximators

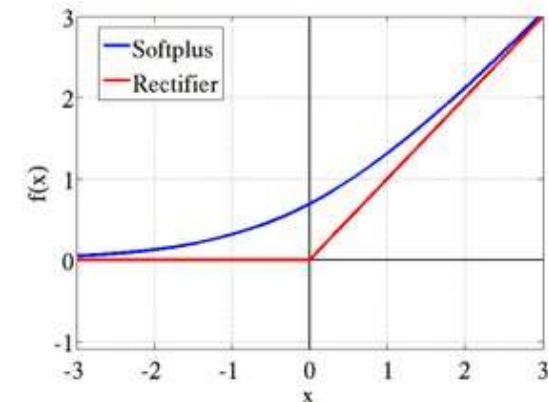
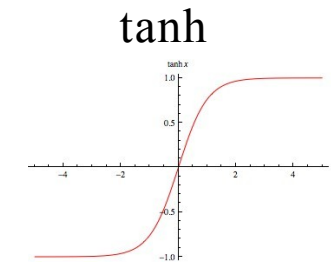
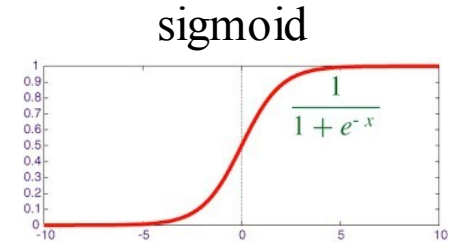
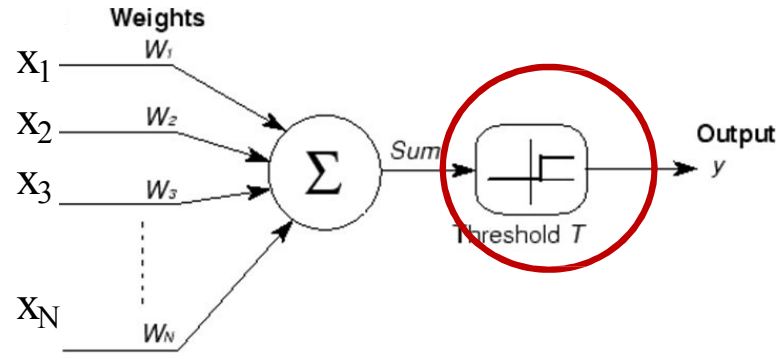
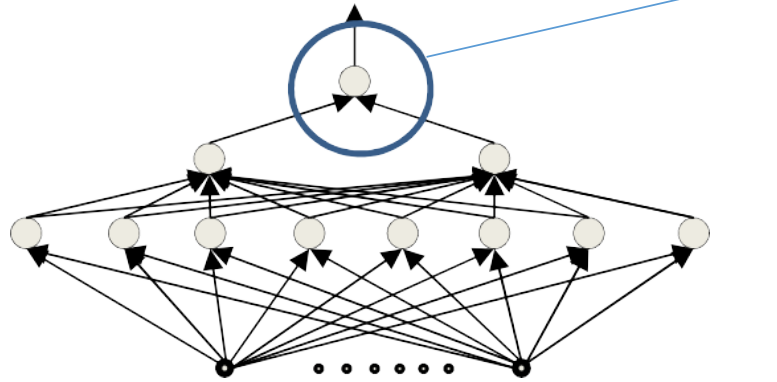


$$o = \sum_{i=1}^N h_i y_i$$



- MLPs can actually compose arbitrary functions in any number of dimensions!
- 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



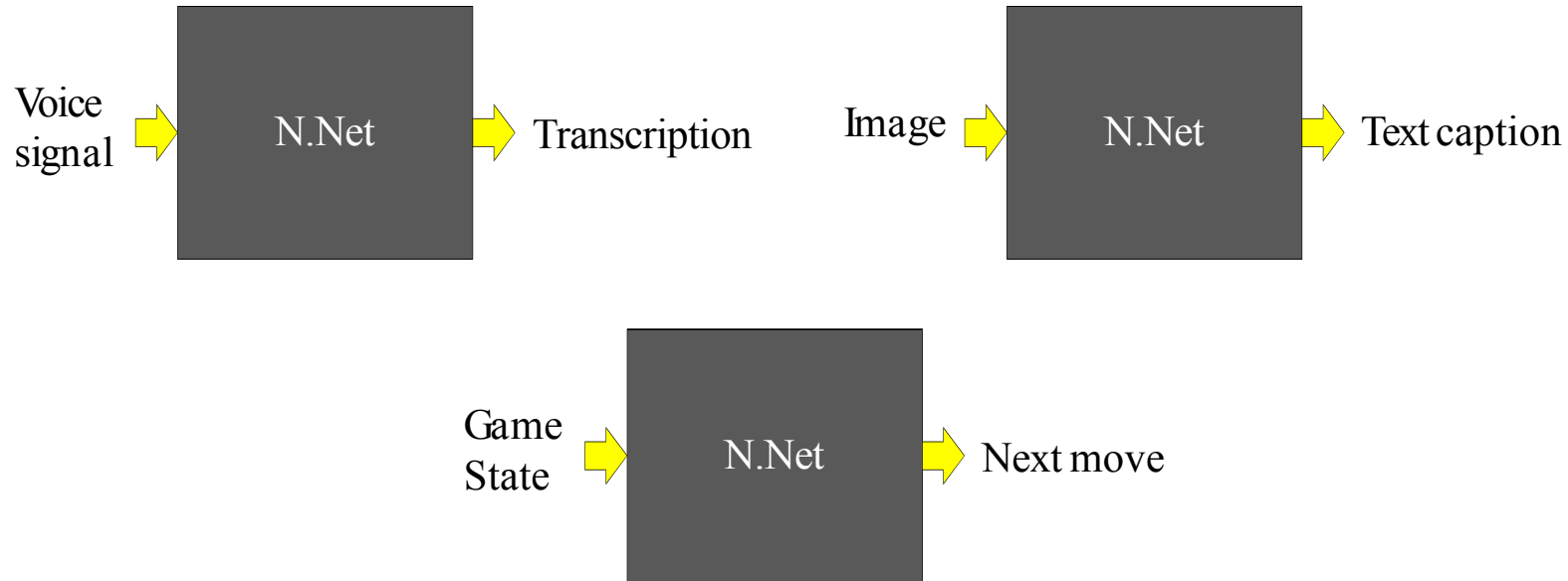
- Output neuron may have actual “activation”
 - Threshold, sigmoid, tanh, softplus, rectifier, etc.
- What is the property of such networks?

Summary

- MLPs are universal Boolean function
- MLPs are universal classifiers
- MLPs are universal function approximators

- A single-layer MLP can approximate anything to arbitrary precision
 - But could be exponentially or even infinitely wide in its inputs size
- Deeper MLPs can achieve the same precision with far fewer neurons
 - Deeper networks are more expressive

These boxes are functions



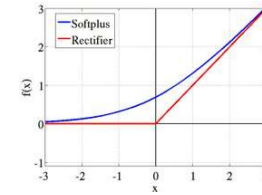
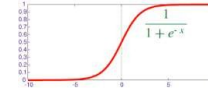
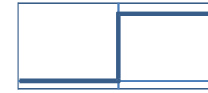
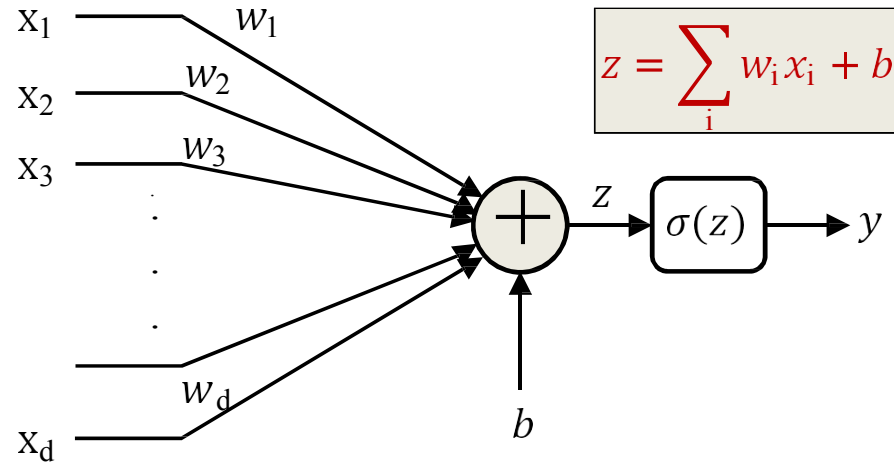
- Take an input
- Produce an output
- Can be modeled by a neuralnetwork!

Questions



- Preliminaries:
 - How do we represent the input?
 - How do we represent the output?
- How do we compose the network that performs the requisite function?

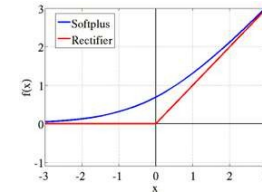
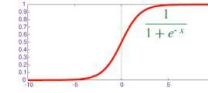
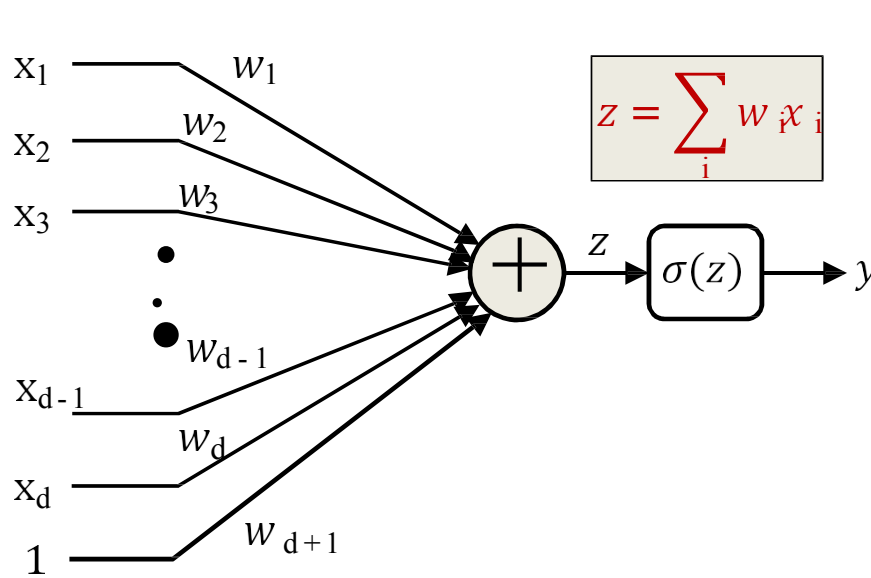
Preliminaries : The units in the networks



Activation functions $\sigma(z)$

- Units or neurons
 - General setting, inputs are real valued
 - A bias b representing a threshold to trigger the perceptron
 - Activation functions are not necessarily threshold functions

Preliminaries : Redrawing the neuron



Activation functions $\sigma(z)$

- The bias can also be viewed as the weight of another input component that is always set to 1

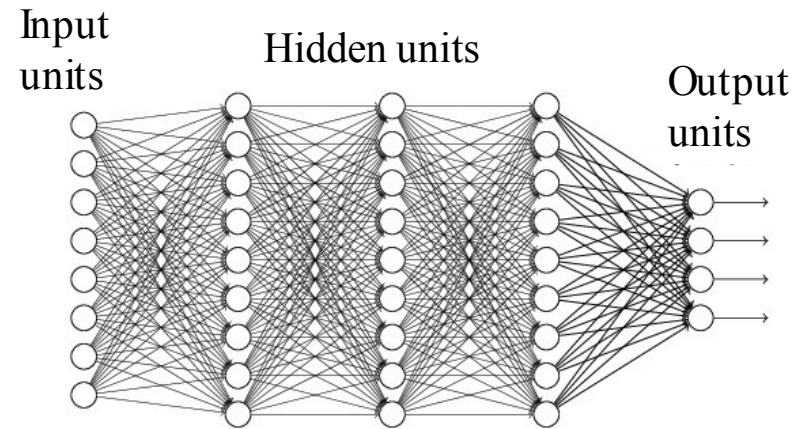
Learning problem

- Given: the architecture of the network
- Training data: A set of input-output pairs

$$(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})$$

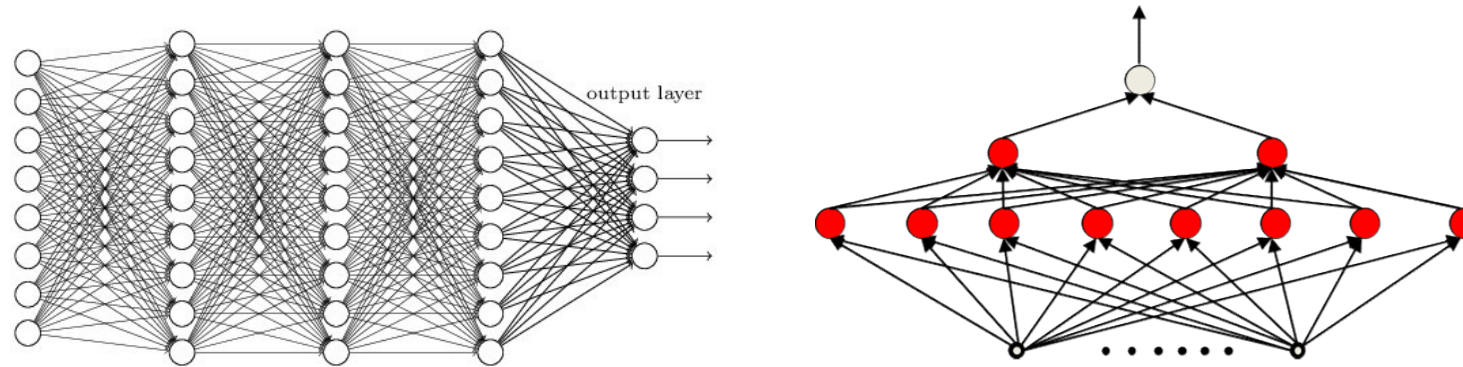
- We want to find the function f on the input space to get the output
 - We consider a neural network as a parametric function $f(\mathbf{x}; \mathbf{W})$

What is $f()$? Typical network



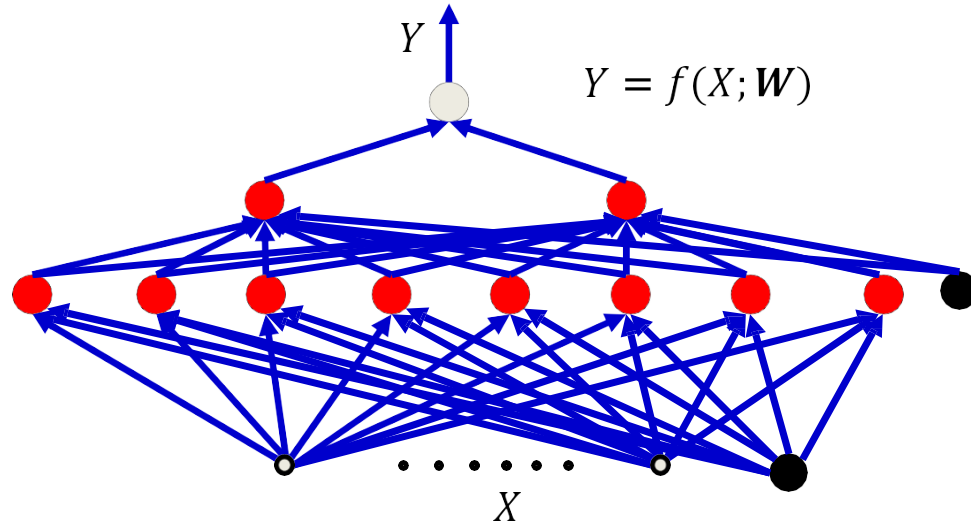
- We assume a “layered” network for simplicity
- Generic terminology
 - We will refer to the inputs as the input units
 - No neurons here – the “input units” are just the inputs
 - We refer to the outputs as the output units
 - Intermediate units are “hidden” units

First : the structure of the network



- We will assume a feed-forward network
 - No loops: Neuron outputs do not feed back to their inputs directly or indirectly
 - Loopy networks are a future topic
- **Part of the design of a network: The architecture**
 - How many layers/neurons, which neuron connects to which and how, etc.
- For now, assume the architecture of the network is capable of representing the needed function

What we learn : The parameter of the network



- Given: the architecture of the network
- **The parameters of the network:** The weights and biases
 - The weights associated with the blue arrows in the picture
- Learning the network: Determining the values of these parameters such that the network computes the desired function

Problem setup

- Given: the architecture of the network
- Training data: A set of input-output pairs
 $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})$
- We want to find the function f
 - We consider a neural network as a parametric function $f(\mathbf{x}; \mathbf{W})$
- We need a **loss function** to show how penalizes the obtained output $f(\mathbf{x}; \mathbf{W})$ when the desired output is \mathbf{y}

$$\frac{1}{N} \sum_{n=1}^N \text{loss}(f(\mathbf{x}^{(n)}; \mathbf{W}), \mathbf{y}^{(n)})$$

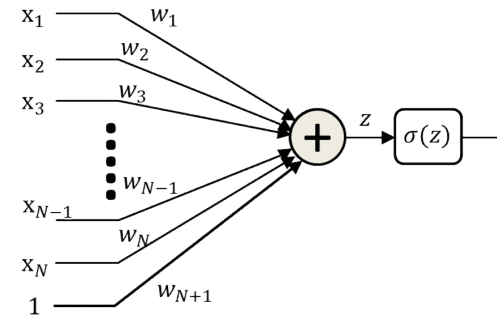
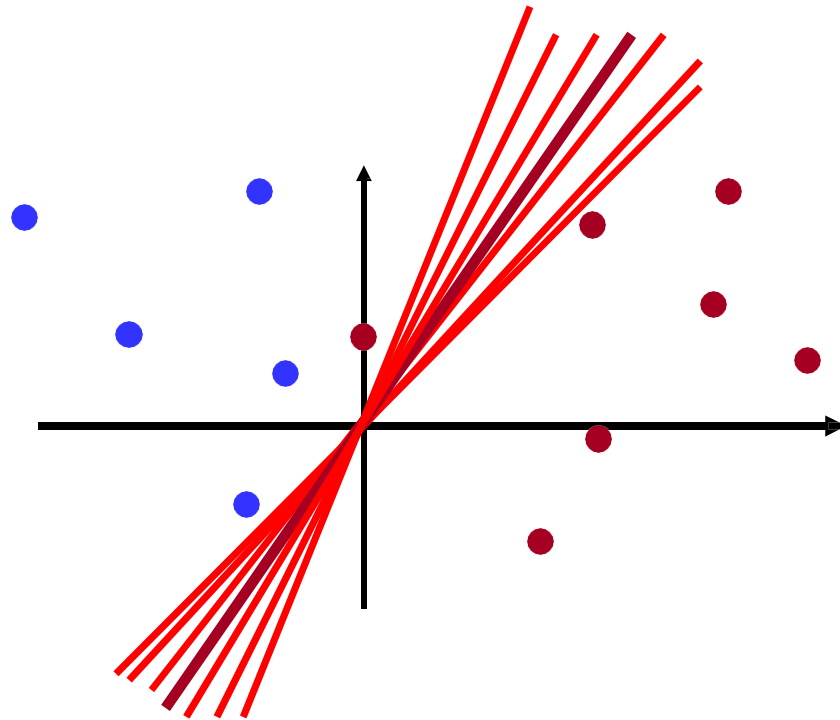
Training an MLP

- We define differentiable loss or divergence between the output of the network and the desired output for the training instances
 - And a total error, which is the average divergence over all training instances
- We optimize network parameters to minimize this error

Training an MLP: Activation function

- Learning networks of threshold-activation neurons requires solving a hard combinatorial-optimization problem
 - Because we cannot compute the influence of small changes to the parameters on the overall error
- Instead we use continuous activation functions to enables us to estimate network parameters
 - This makes the output of the network differentiable w.r.t every parameter in the network
 - The logistic activation neuron actually computes the a posteriori probability of the output given the input

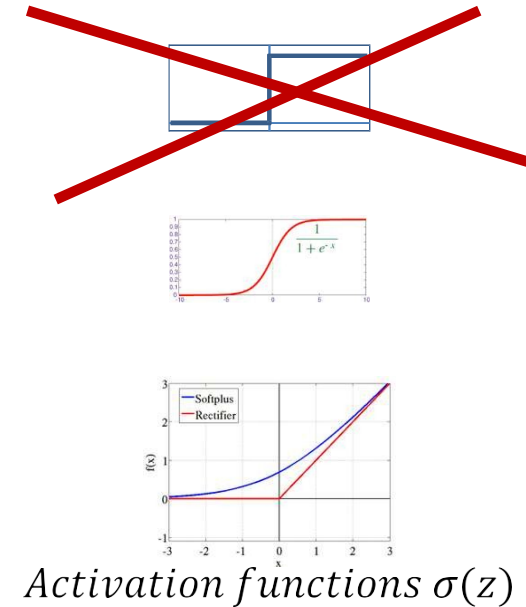
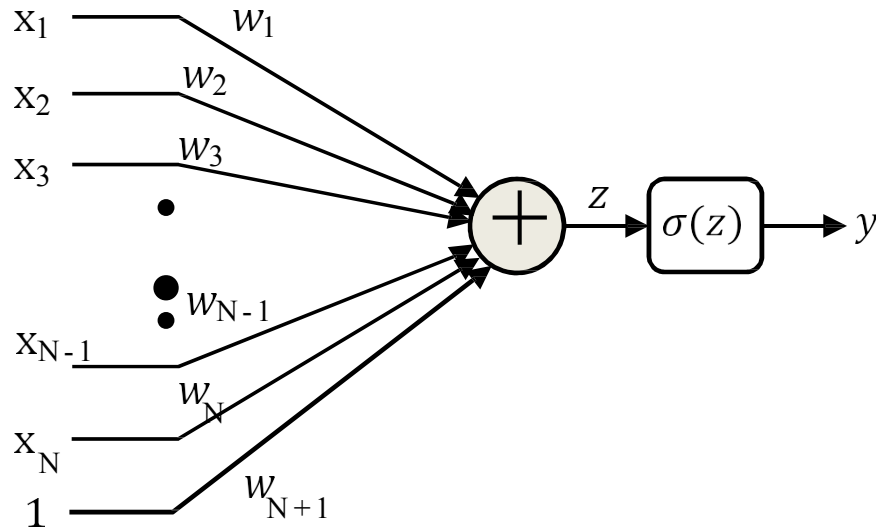
Activation function



$$\sigma(z) = \begin{array}{|c|c|} \hline & \\ \hline \end{array}$$

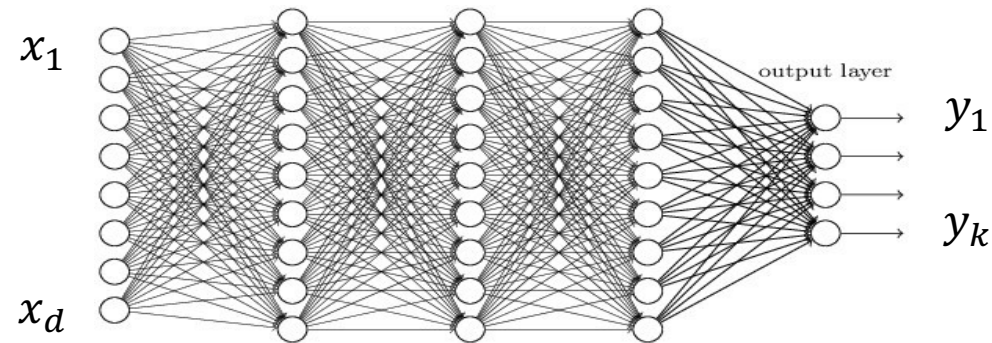
- With threshold, the neuron's output is a flat function with zero derivative everywhere, except at 0 where it is non-differentiable
 - You can vary the weights a lot without changing the error
 - There is no indication of which direction to change the weights to reduce error

Activation function



- Makes the neuron differentiable, with non-zero derivatives over much of the input space
 - Small changes in weight can result in non-negligible changes in output
 - This enables us to estimate the parameters using gradient descent techniques..

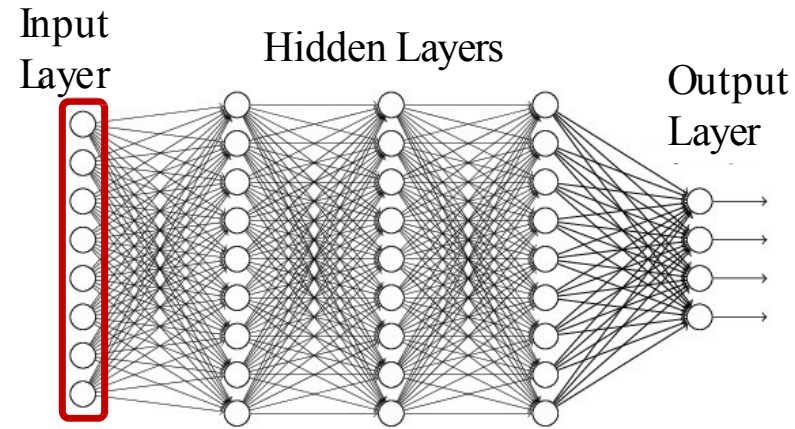
Vector notation



Given a training of input-output pairs $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)})$, $(\mathbf{x}^{(2)}, \mathbf{y}^{(2)})$, ..., $(\mathbf{x}^{(N)}, \mathbf{y}^{(N)})$

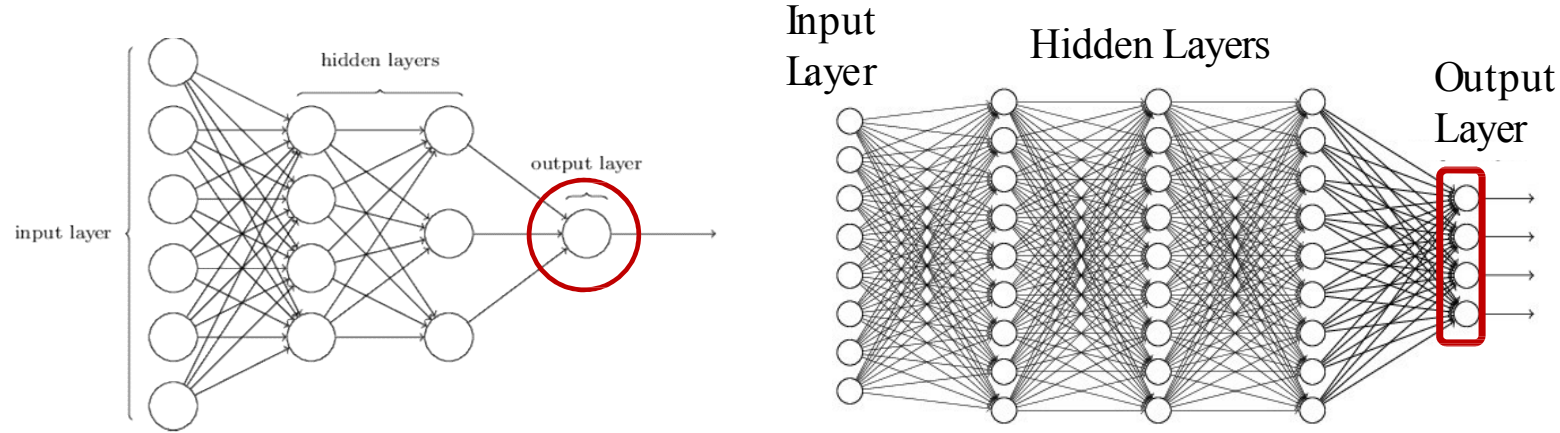
- $\mathbf{x}^{(n)} = [x_1^{(n)}, x_2^{(n)}, \dots, x_d^{(n)}]$ is the n th input vector
- $\mathbf{y}^{(n)} = [y_1^{(n)}, y_2^{(n)}, \dots, y_k^{(n)}]$ is the n th desired output
- $\mathbf{o}^{(n)} = [o_1^{(n)}, o_2^{(n)}, \dots, o_k^{(n)}]$ is the n th vector of actual outputs of the network
- We will sometimes drop the superscript when referring to a specific instance

Representing the input



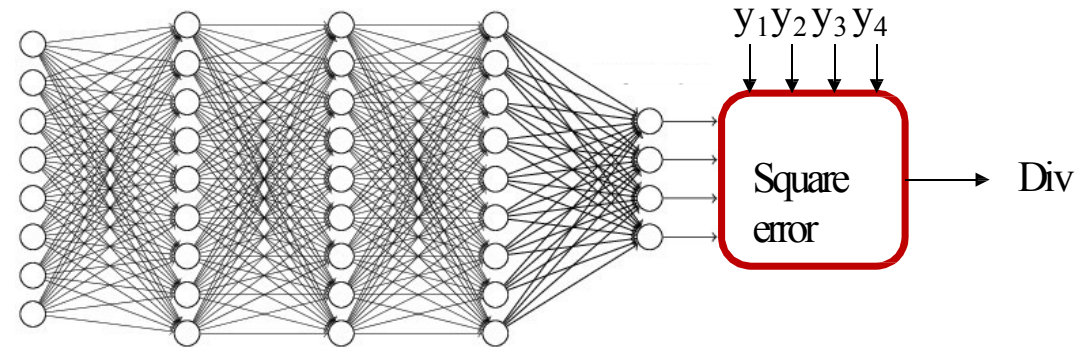
- Vectors of numbers
 - (or may even be just a scalar, if input layer is of size 1)
 - E.g. vector of pixel values
 - E.g. vector of speech features
 - E.g. real-valued vector representing text
 - We will see how this happens later in the course
 - Other real valued vectors

Representing the output



- If the desired output is real-valued, no special tricks are necessary
 - Scalar Output : single output neuron
 - o = scalar (real value)
 - Vector Output : as many output neurons as the dimension of the desired output
 - $\mathbf{o} = [o_1, o_2, \dots, o_K]$ (vector of real values)

Examples of loss functions



- For real-valued output vectors, the (scaled) L_2 divergence is popular

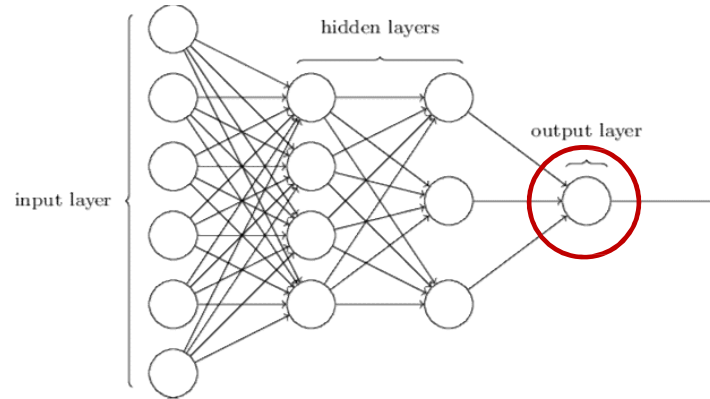
$$Error(\mathbf{y}, \mathbf{o}) = \frac{1}{2} \|\mathbf{y} - \mathbf{o}\|^2 = \frac{1}{2} \sum_k (y_k - o_k)^2$$

- Squared Euclidean distance between true and desired output
- Note: this is differentiable

$$\frac{dE(\mathbf{y}, \mathbf{o})}{do_k} = -(y_k - o_k)$$

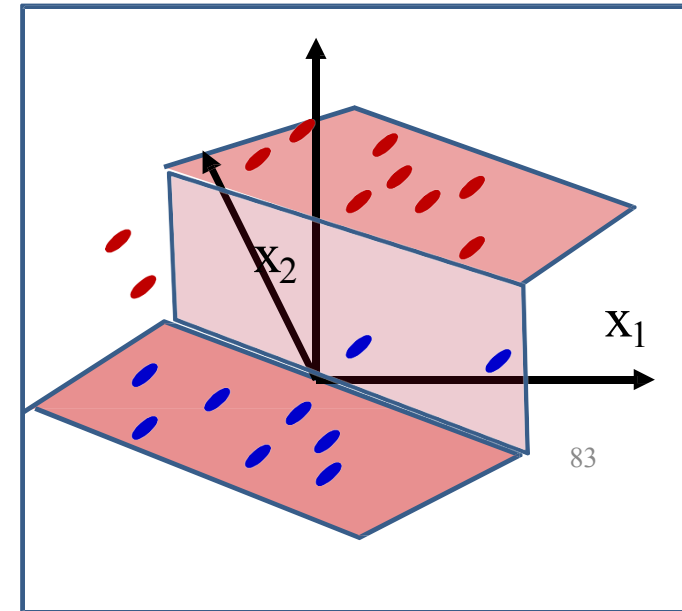
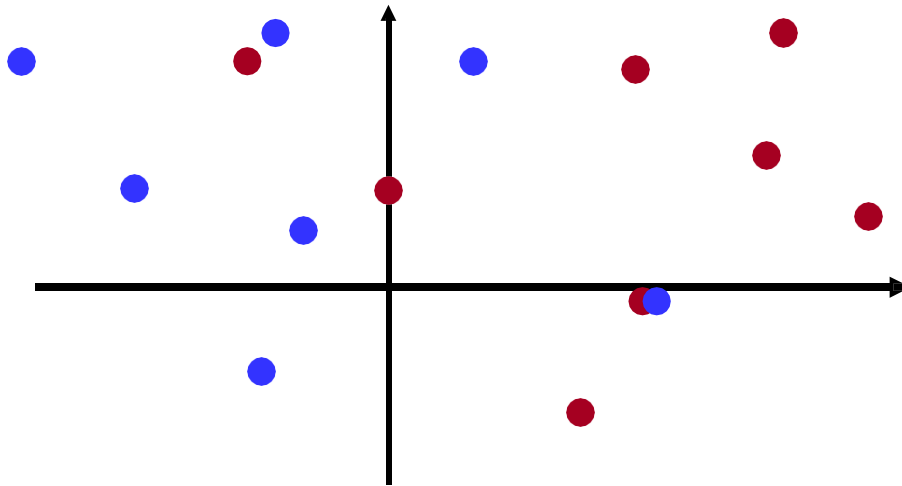
$$\nabla_{\mathbf{o}} E(\mathbf{y}, \mathbf{o}) = [o_1 - y_1, o_2 - y_2, \dots, o_K - y_K]$$

Representing the output



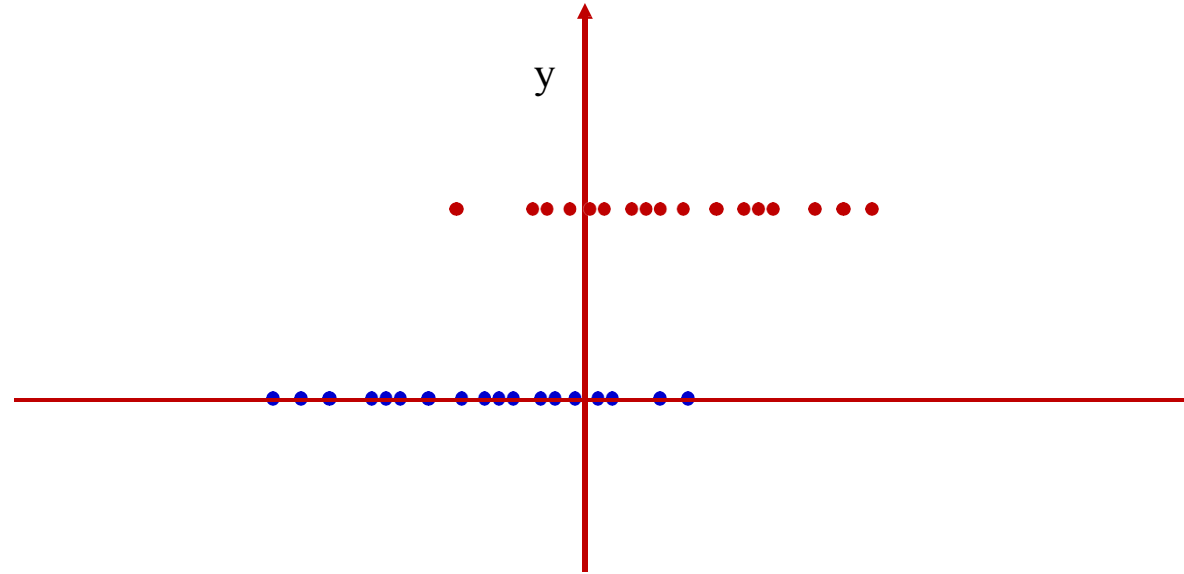
- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
 - 1 = YES it's a cat
 - 0 = NO it's not a cat.

Non-linearly separable data



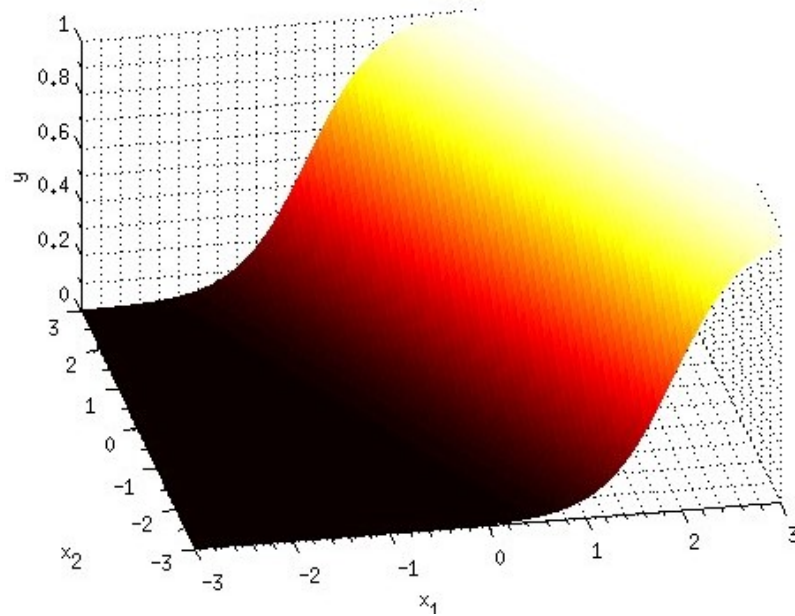
- Two-dimensional example
 - Blue dots (on the floor) on the “red” side
 - Red dots (suspended at $Y=1$) on the “blue” side
 - No line will cleanly separate the two colors

Non-linearly separable data: 1-D example

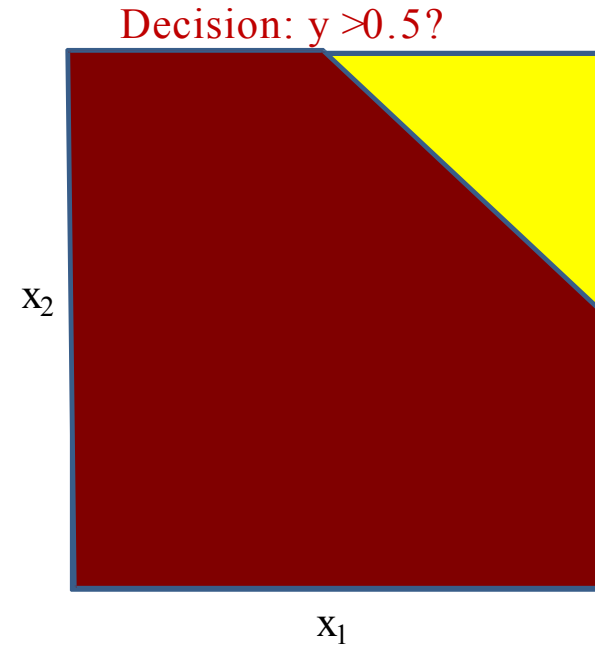


- One-dimensional example for visualization
 - All (red) dots at $Y=1$ represent instances of class $Y=1$
 - All (blue) dots at $Y=0$ are from class $Y=0$
 - The data are not linearly separable
 - No threshold will cleanly separate red and blue dots

Logistic regression



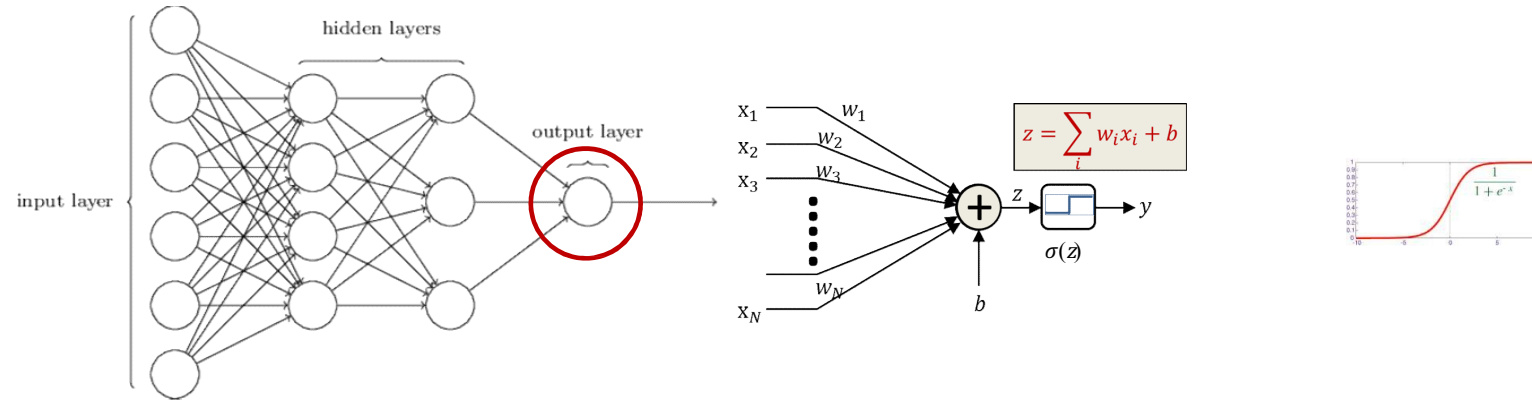
When X is a 2-D variable



$$P(Y = 1|X) = \frac{1}{1 + \exp(-\sum_i w_i x_i - b)}$$

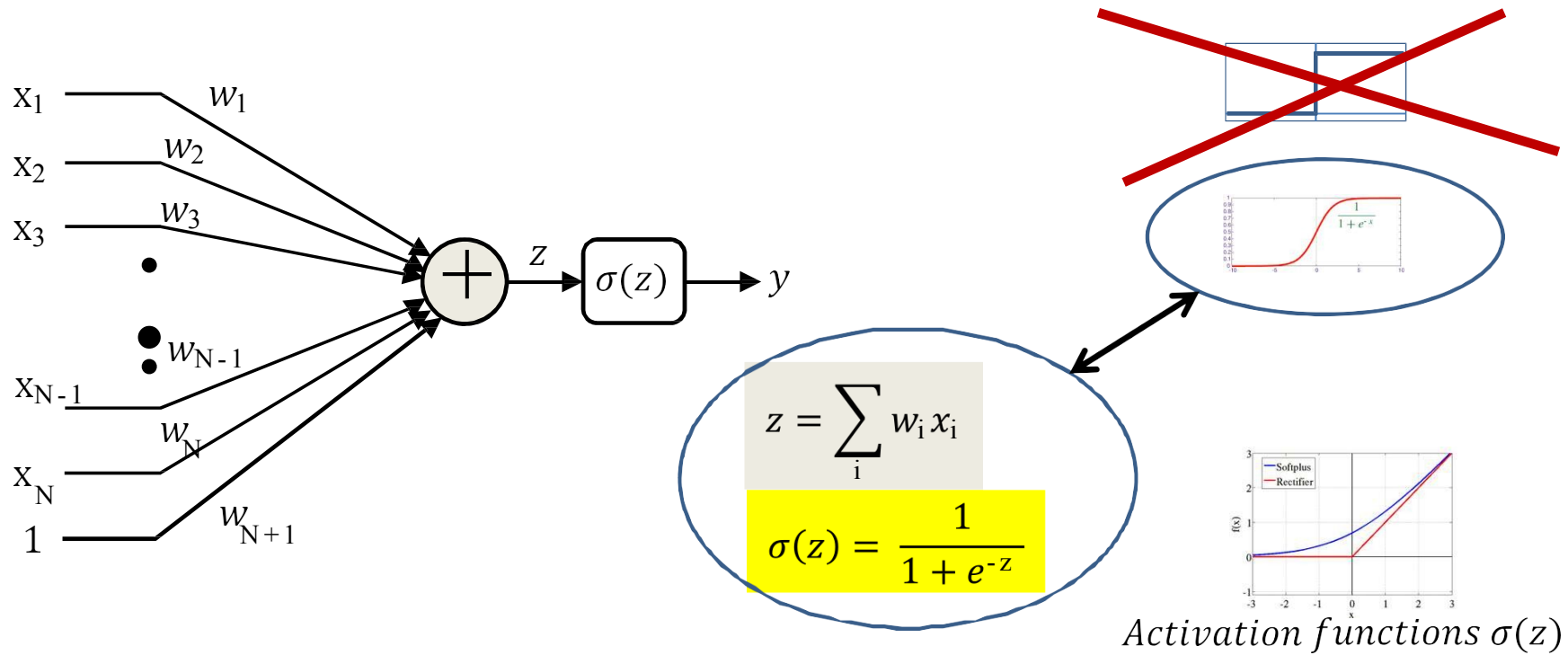
- This a neuron with a sigmoid activation
 - It actually computes the probability that the input belongs to class 1

Representing the output



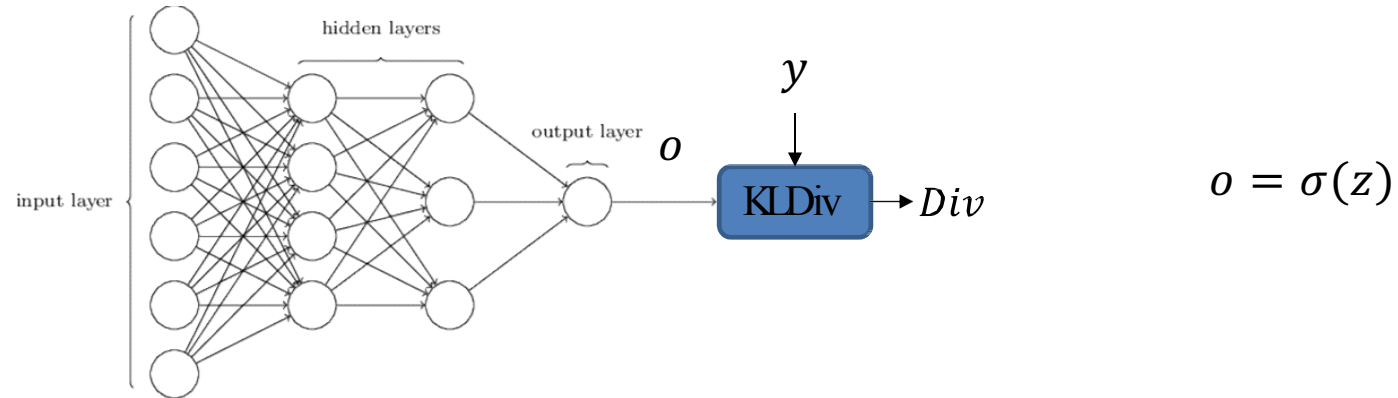
- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
- Output activation: Typically a sigmoid
 - Viewed as the probability $P(Y = 1|\mathbf{x})$ of class value 1
 - Indicating the fact that for actual data, in general a feature vector may occur for both classes, but with different probabilities
 - Is differentiable

Differentiable Activation



- This particular one has a nice interpretation

For binary classifier: Logistic regression



- For binary classifier with scalar output $o \in (0,1)$, y is 0/1, the cross entropy between the probability distribution $[o, 1 - o]$ and the ideal output probability $[y, 1 - y]$ is popular

$$L(y, o) = -y \log o - (1 - y) \log(1 - o)$$

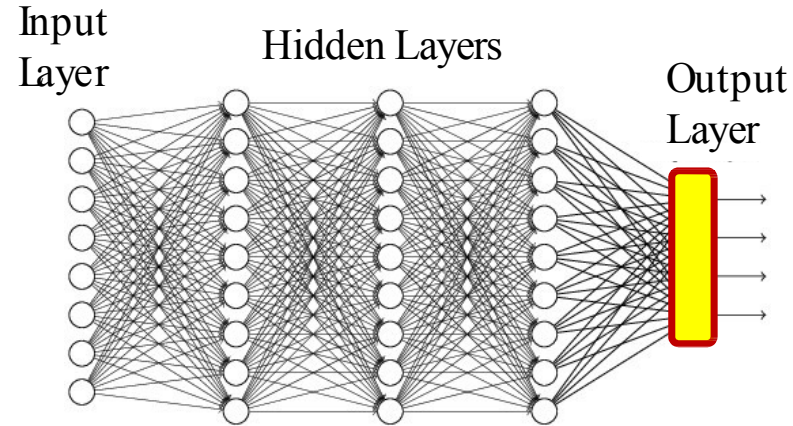
- Derivative

$$\frac{dL(y, o)}{do} = \begin{cases} -\frac{1}{o} & \text{if } y = 1 \\ \frac{1}{1 - o} & \text{if } y = 0 \end{cases}$$

Multi-class output: One-hot representations

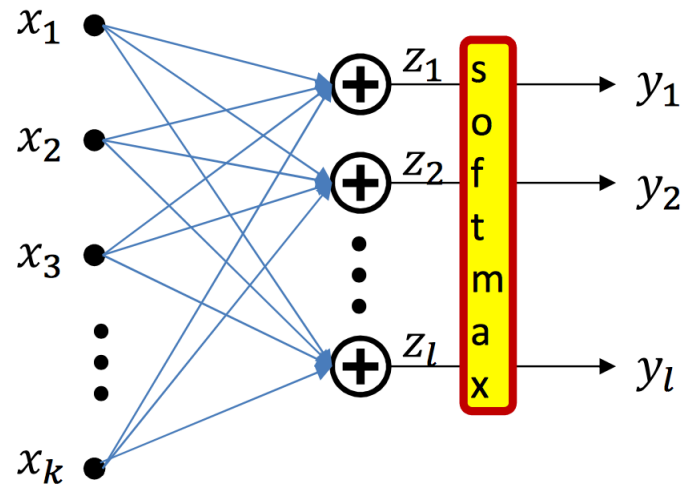
- Consider a network that must distinguish if an input is a cat, a dog, a camel, a hat, or a flower
- For inputs of each of the five classes the desired output is:
 - Cat : $[1\ 0\ 0\ 0\ 0]^T$
 - dog : $[0\ 1\ 0\ 0\ 0]^T$
 - camel : $[0\ 0\ 1\ 0\ 0]^T$
 - hat : $[0\ 0\ 0\ 1\ 0]^T$
 - flower : $[0\ 0\ 0\ 0\ 1]^T$
- For input of any class, we will have a five-dimensional vector output with four zeros and a single 1 at the position of the class
- This is a one hot vector

Multi-class networks



- For a multi-class classifier with N classes, the one-hot representation will have N binary outputs
 - An N -dimensional binary vector
- The neural network's output too must ideally be binary ($N-1$ zeros and a single 1 in the right place)
- More realistically, it will be a probability vector
 - N probability values that sum to 1.

Vector activation example: Softmax



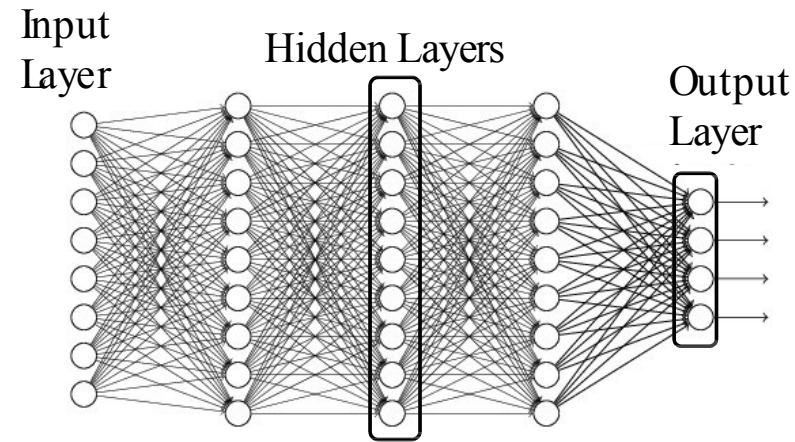
- Example: Softmax **vector** activation

$$z_i = \sum_j w_{ji} x_j + b_i$$

Parameters are weights w_{ji} and bias b_i

$$o_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Vector Activations

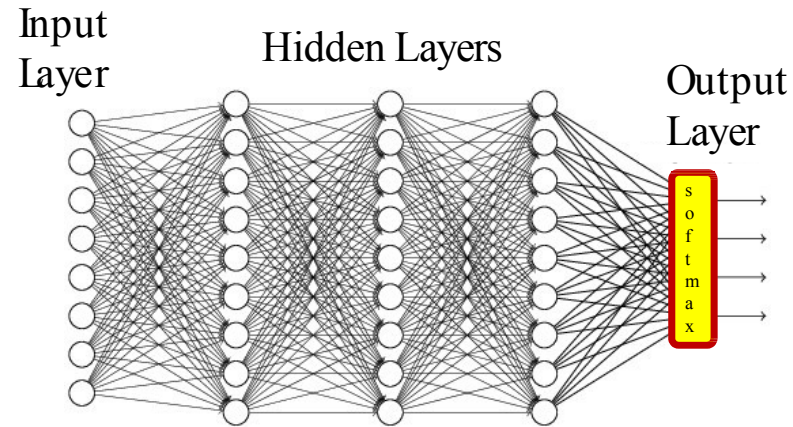


- We can also have neuron that have multiple couple outputs

$$[y_1, y_2, \dots, y_l] = f(x_1, x_2, \dots, x_k; \mathbf{W})$$

- Function $f(\cdot)$ operates on set of inputs to produce set of outputs
- Modifying a single parameter in \mathbf{W} will affect all outputs

Multi-class classification: Output



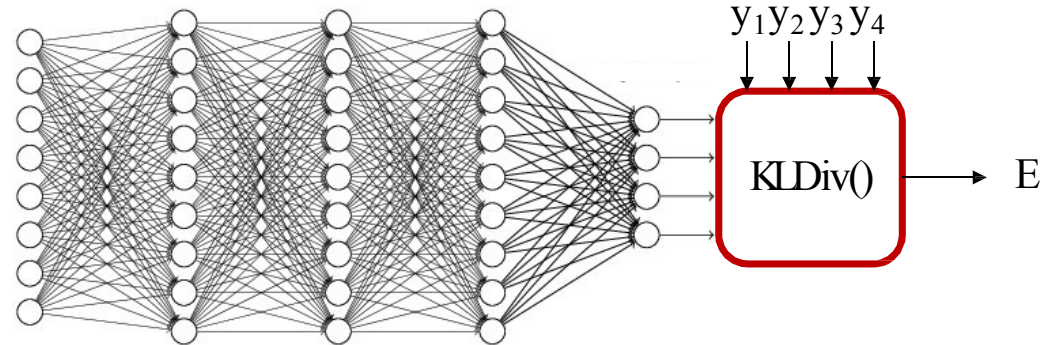
- Softmax vector activation is often used at the output of multi-class classifier nets

$$z_i = \sum_j w_{ji}^{(l)} a_j^{(n-1)}$$

$$o_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- This can be viewed as the probability $o_i = P(\text{class} = i | \mathbf{x})$

For multi-class classification



- Desired output \mathbf{y} is one hot vector $[0 \ 0 \ \dots \ 1 \ \dots \ 0 \ 0 \ 0]$ with the 1 in the c -th position (for class c)
- Actual output will be probability distribution $[o_1, o_2, \dots, o_l]$
- The cross-entropy between the desired one-hot output and actual output

$$L(\mathbf{y}, \mathbf{o}) = - \sum_i y_i \log o_i = -\log o_c$$

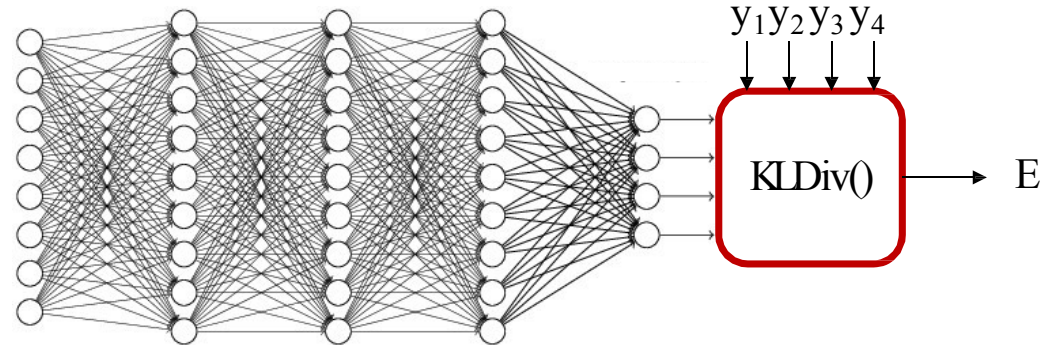
- Derivative

$$\frac{dL(\mathbf{y}, \mathbf{o})}{do_i} = \begin{cases} -\frac{1}{o_c} & \text{for the } c \text{ th component} \\ 0 & \text{for remaining component} \end{cases}$$

The slope is negative w.r.t. o_c
Indicates **increasing** o_c will **reduce** divergence

$$\nabla_{\mathbf{o}} L(\mathbf{y}, \mathbf{o}) = [0 \ 0 \ \dots \ \frac{-1}{o_c} \ \dots \ 0 \ 0]$$

For multi-class classification



- Desired output y is one hot vector $[0 \ 0 \ \dots \ 1 \ \dots \ 0 \ 0 \ 0]$ with the 1 in the c -th position (for class c)
- Actual output will be probability distribution $[o_1, o_2, \dots, o_l]$
- The cross-entropy between the desired one-hot output and actual output

$$L(\mathbf{y}, \mathbf{o}) = - \sum_i y_i \log o_i = -\log o_c$$

- Derivative

$$\frac{dL(\mathbf{y}, \mathbf{o})}{do_i} = \begin{cases} -\frac{1}{o_c} & \text{for the } c \text{ th component} \\ 0 & \text{for remaining component} \end{cases}$$

The slope is negative w.r.t. o_c

Indicates **increasing** o_c will **reduce** divergence

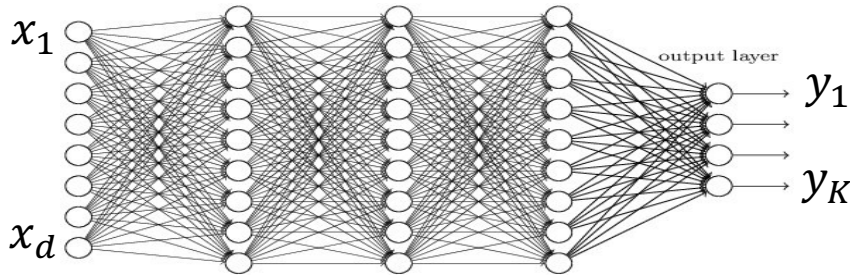
Note: when $\mathbf{y} = \mathbf{o}$ the derivative is not 0

$$\nabla_{\mathbf{o}} L(\mathbf{y}, \mathbf{o}) = [0 \ 0 \ \dots \ \frac{-1}{o_c} \ \dots \ 0 \ 0]$$

Choosing cost function: Examples

▶ Regression problem

– SSE



$$E = \sum_{n=1}^N E_n$$

$$E_n = \frac{1}{2} \left(o^{(n)} - y^{(n)} \right)^2$$

One dimensional output

$$E_n = \frac{1}{2} \left\| \mathbf{o}^{(n)} - \mathbf{y}^{(n)} \right\|^2 = \sum_{k=1}^K \left(o_k^{(n)} - y_k^{(n)} \right)^2$$

Multi-dimensional output

▶ Classification problem

– Cross-entropy

- Binary classification
- Multi-class classification

$$loss_n = -y^{(n)} \log o^{(n)} - (1 - y^{(n)}) \log(1 - o^{(n)}) \quad o = \frac{1}{1 + e^z}$$

Output layer uses sigmoid activation function

$$loss_n = -\log o_{y^{(n)}}$$

Output is found by a softmax layer $o_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$

Problem setup

- Given: the architecture of the network
- Training data: A set of input-output pairs
 $(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \dots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})$
- We need a **loss function** to show how penalizes the obtained output $o = f(\mathbf{x}; \mathbf{W})$ when the desired output is \mathbf{y}

$$E(\mathbf{W}) = \sum_{n=1}^N \text{loss}(\mathbf{o}^{(n)}, \mathbf{y}^{(n)}) = \frac{1}{N} \sum_{n=1}^N \text{loss}(f(\mathbf{x}^{(n)}; \mathbf{W}), \mathbf{y}^{(n)})$$

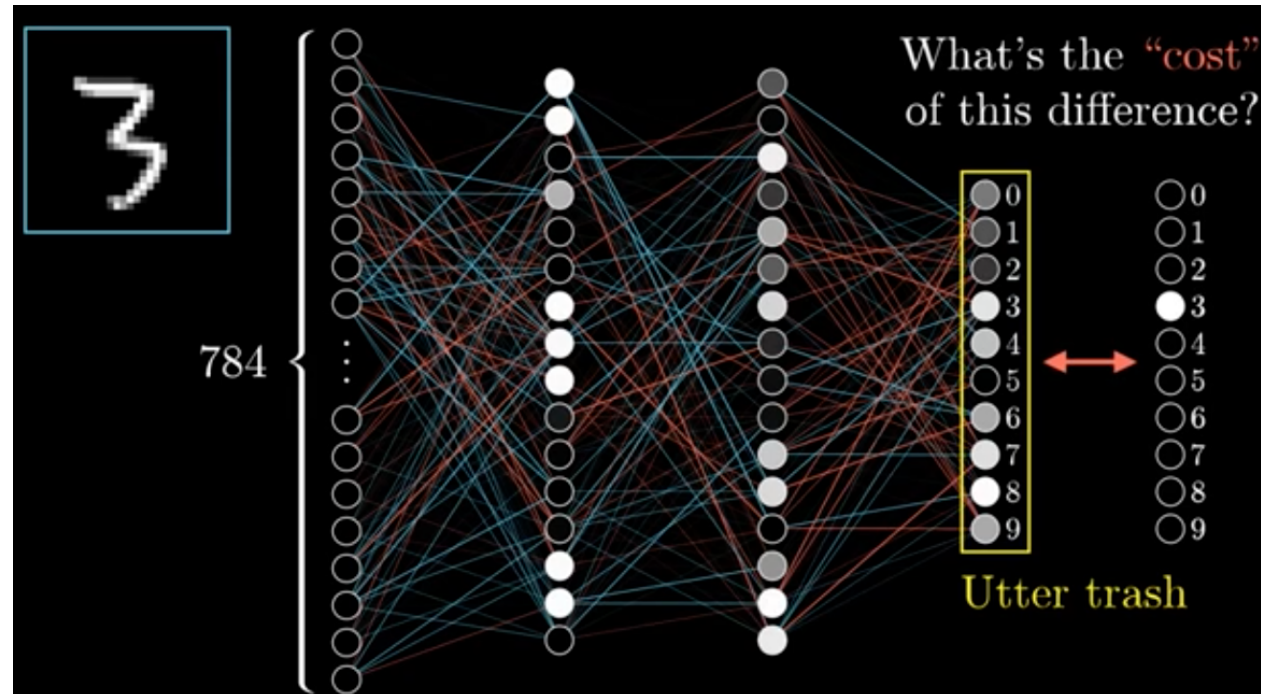
- Minimize E w.r.t. \mathbf{W} that contains $\{w_{i,j}^{[k]}, b_j^{[k]}\}$

How to adjust weights for multi layer networks?

- We need multiple layers of adaptive, non-linear hidden units. But how can we train such nets?
 - We need an efficient way of adapting all the weights, not just the last layer.
 - Learning the weights going into hidden units is equivalent to learning features.
 - This is difficult because nobody is telling us directly what the hidden units should do.

Find the weights by optimizing the cost

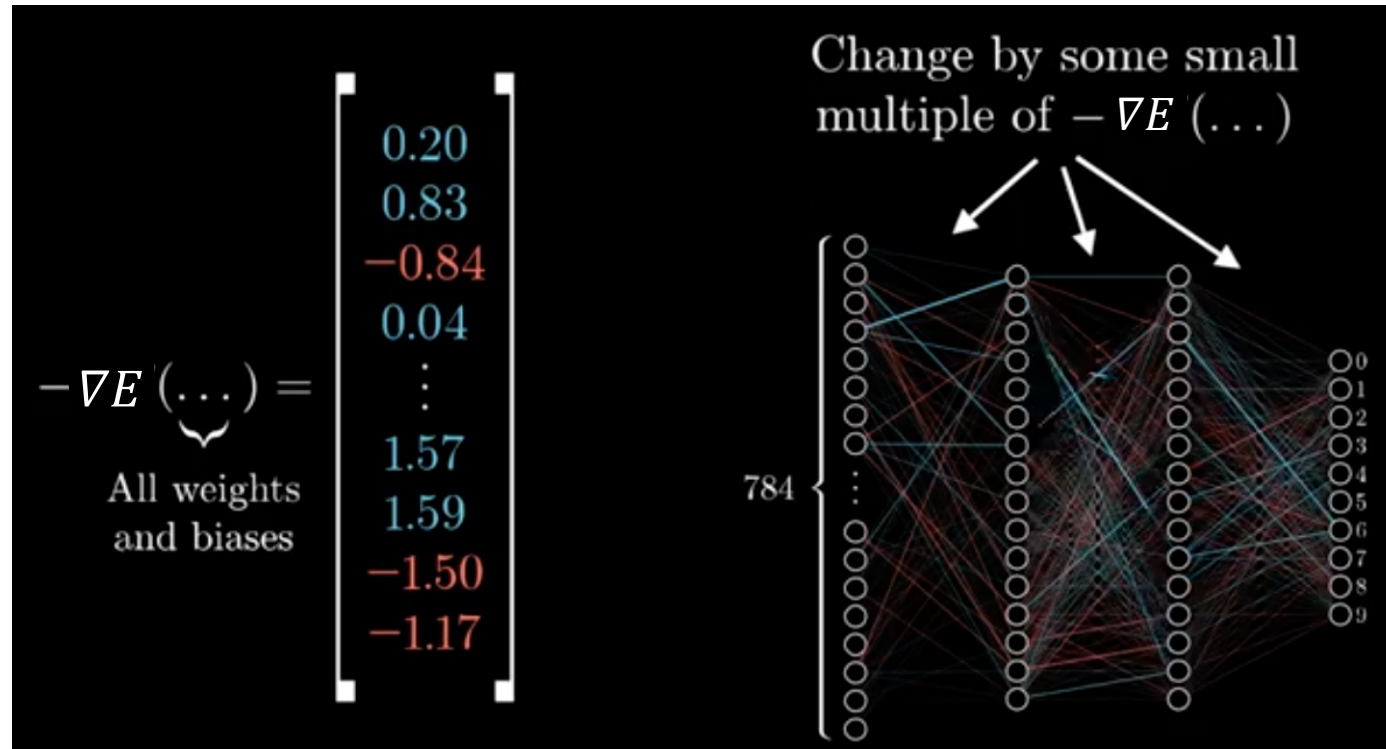
- Start from random weights and then adjust them iteratively to get lower cost.
- Update the weights according to the gradient of the cost function



Source: <http://3b1b.co>

How does the network learn?

- Which changes to the weights do improve the most?



- The magnitude of each element shows how sensitive the cost is to that weight or bias.

Training multi-layer networks

- Back-propagation
 - Training algorithm that is used to adjust weights in multi-layer networks (based on the training data)
 - The back-propagation algorithm is based on gradient descent
 - Use chain rule and dynamic programming to efficiently compute gradients