Intro to Neural Networks

Lisbon Machine Learning School

18 June 2018

What’s in this tutorial

- We will learn about
  - What is a neural network: historical perspective
  - What can neural networks model
  - What do they actually learn

Instructor

- Bhiksha Raj

  Professor,
  Language Technologies Institute

  (Also: MLD, ECE, Music Tech)
  Carnegie Mellon Univ.
- bhiksha@cs.cmu.edu

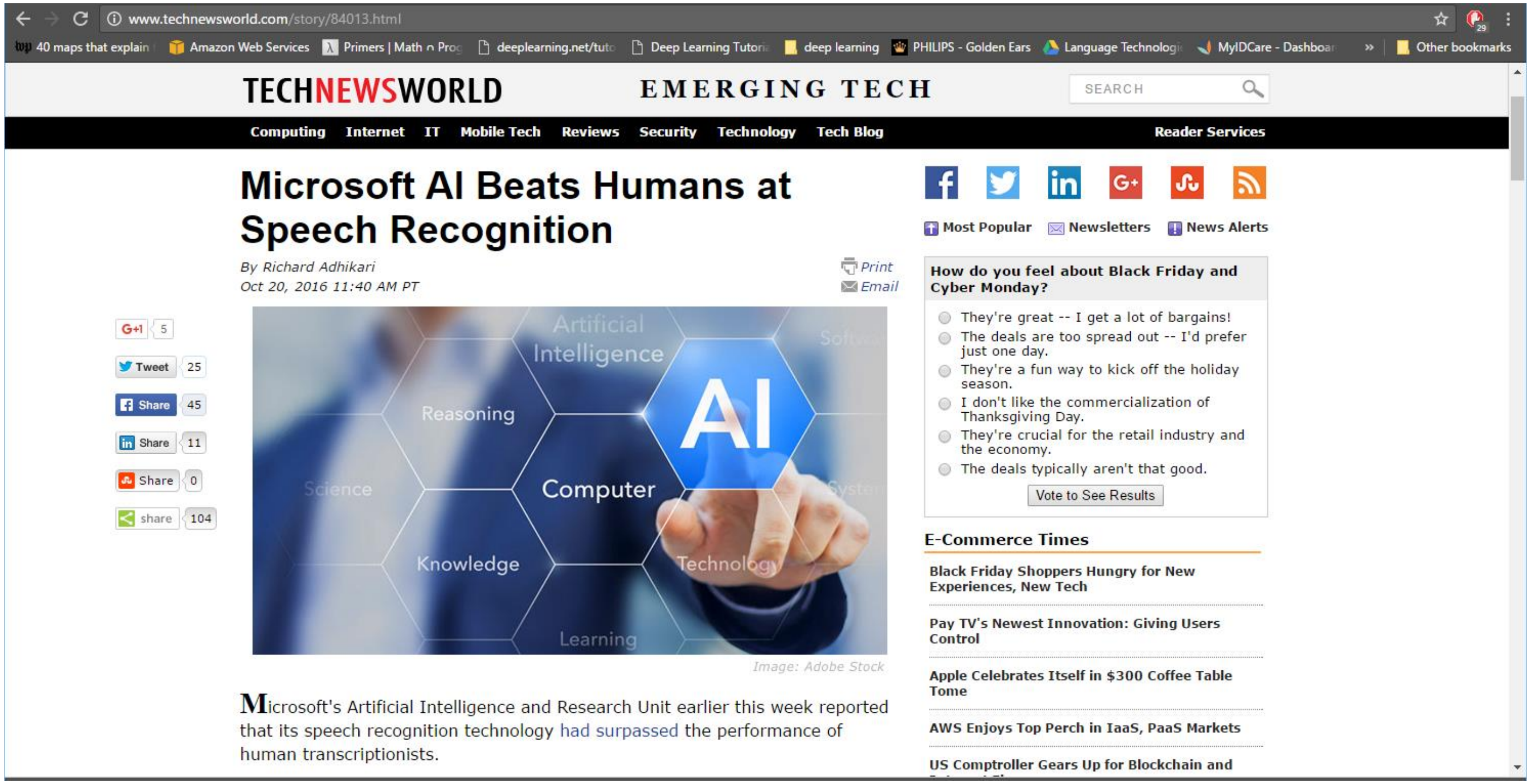
Part 1: What is a neural network

Neural Networks are taking over!

- Neural networks have become one of the major thrust areas recently in various pattern recognition, prediction, and analysis problems

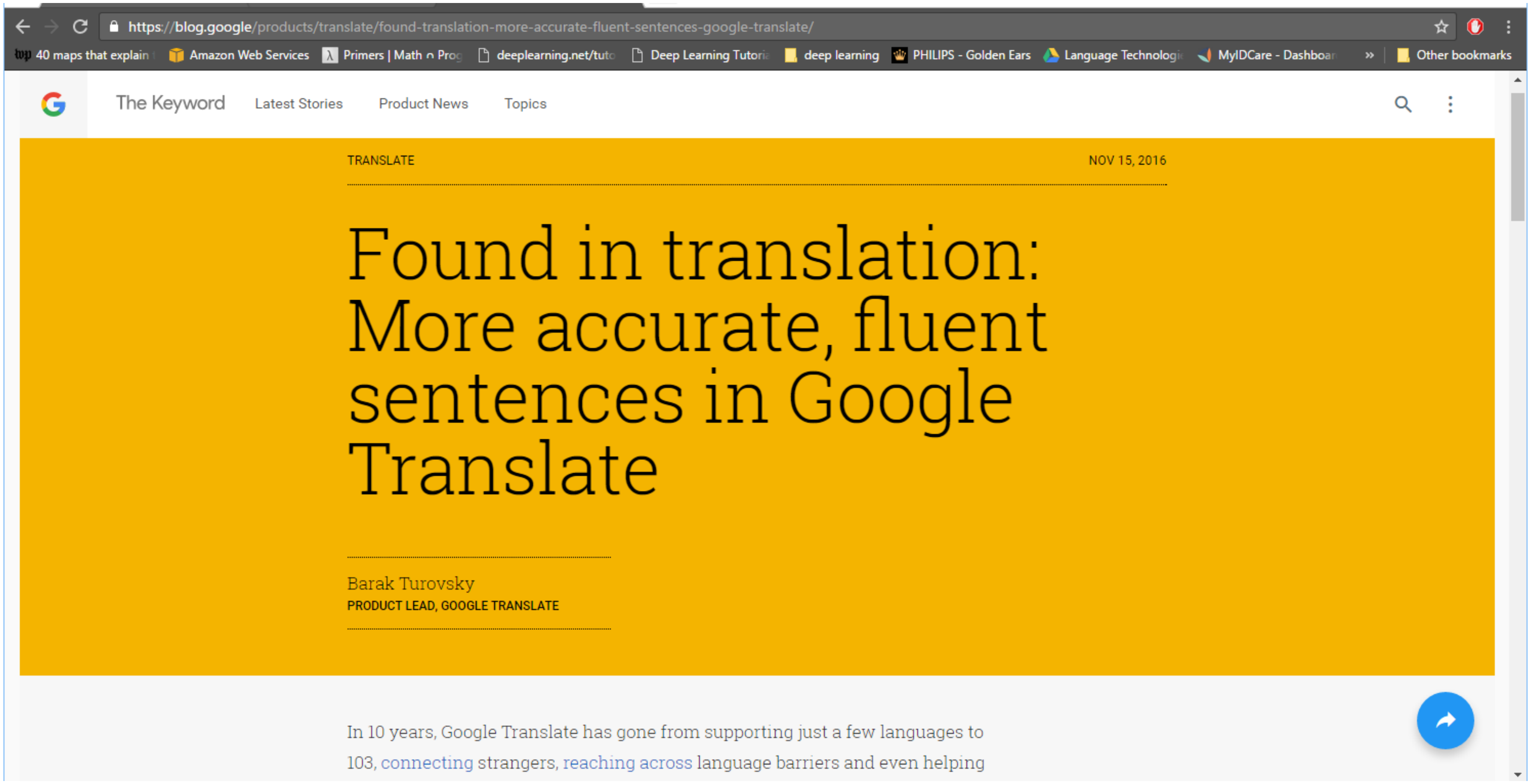
- In many problems they have established the state of the art
  - Often exceeding previous benchmarks by large margins

Recent success with neural networks



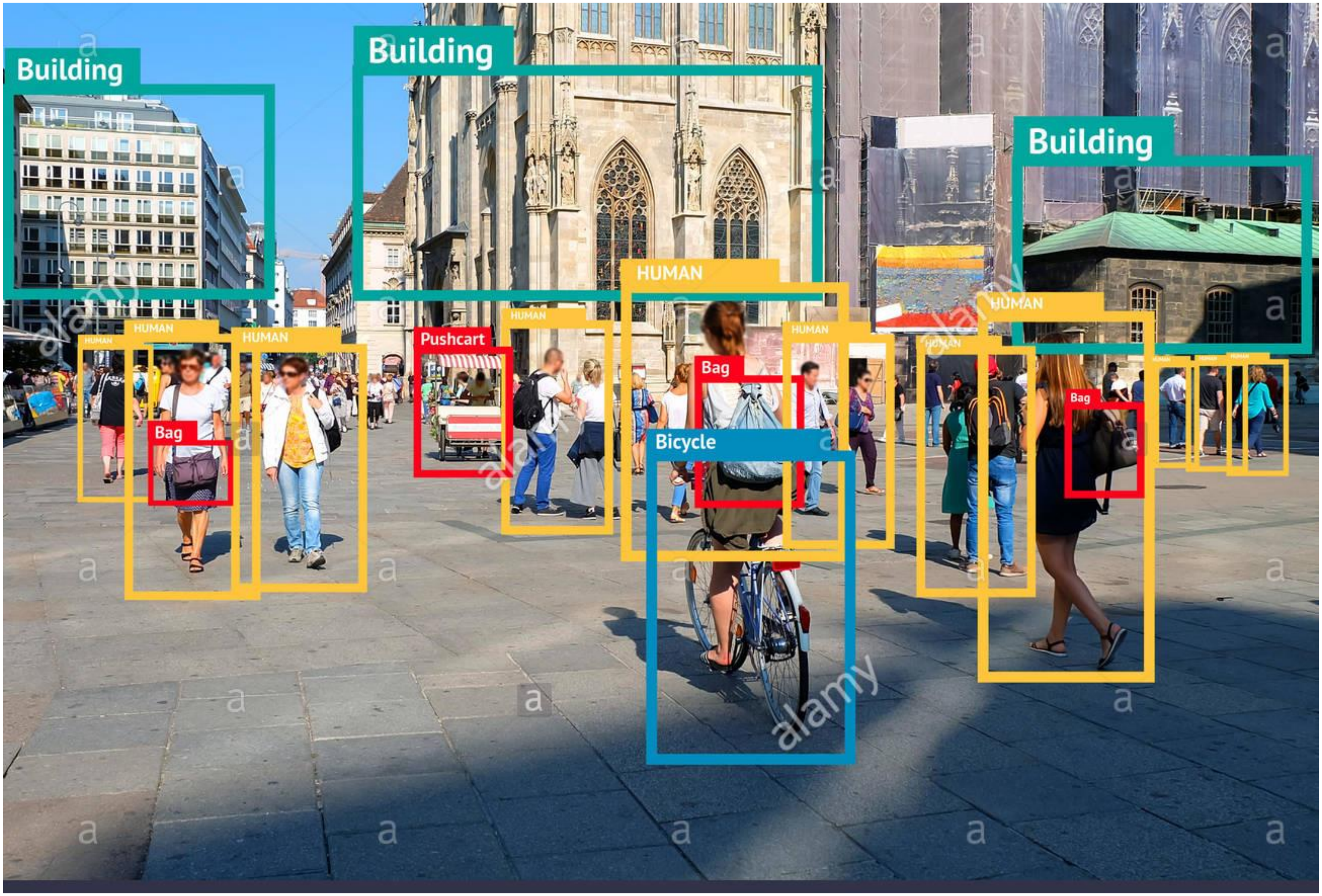
- Some recent successes with neural networks

Recent success with neural networks

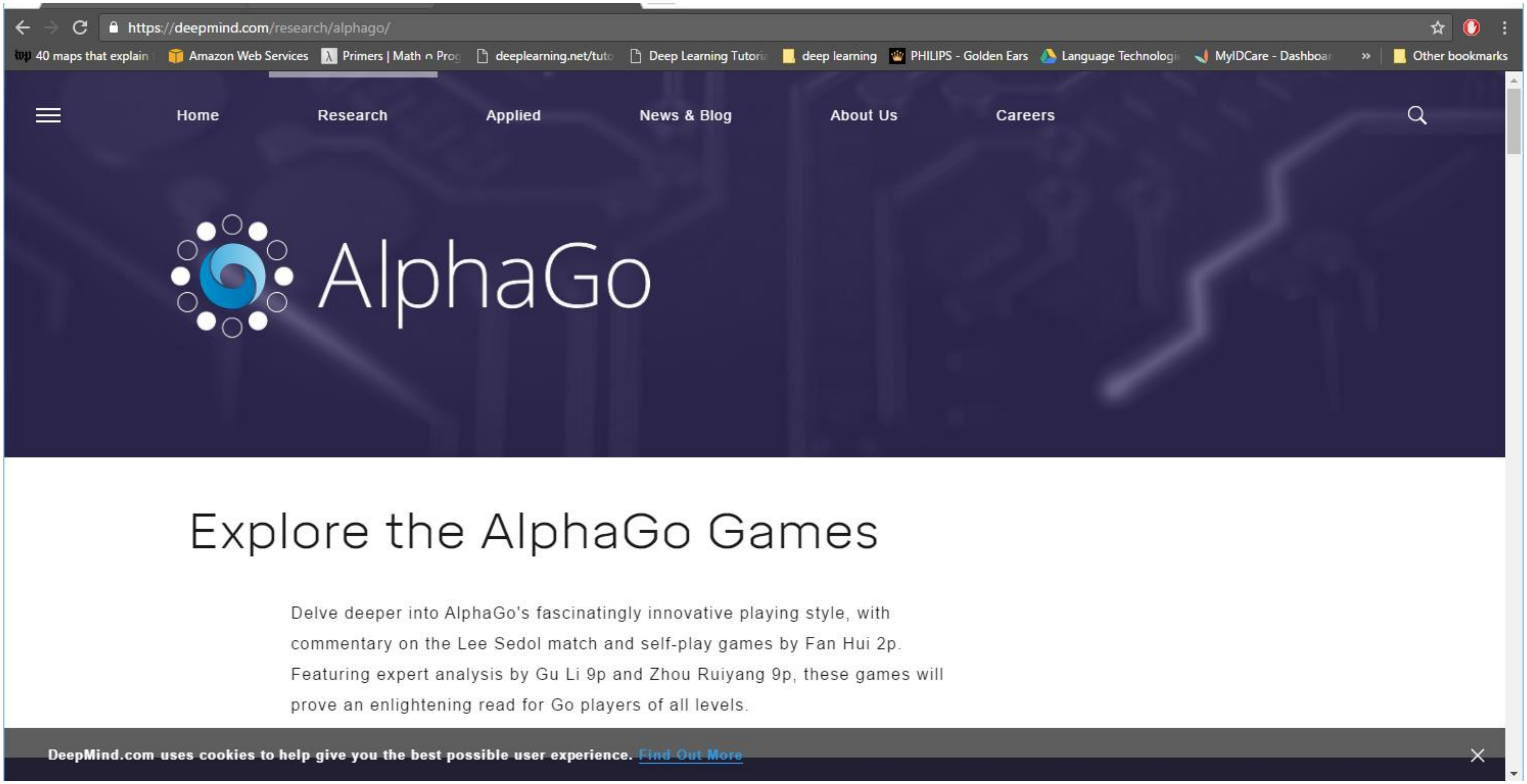


- Some recent successes with neural networks

Recent success with neural networks

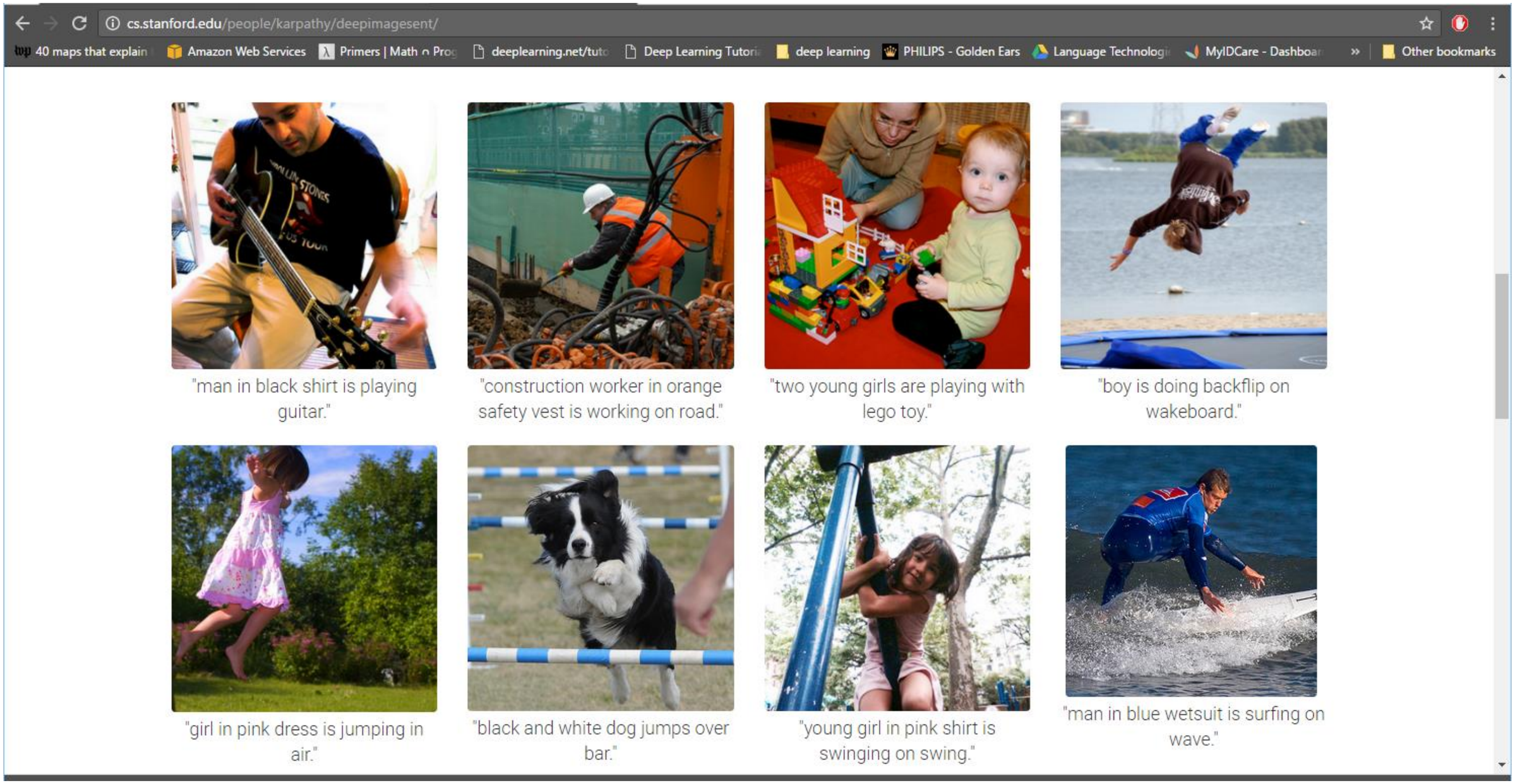


Recent success with neural networks



- Some recent successes with neural networks

Recent success with neural networks



- Captions generated entirely by a neural network

Successes with neural networks

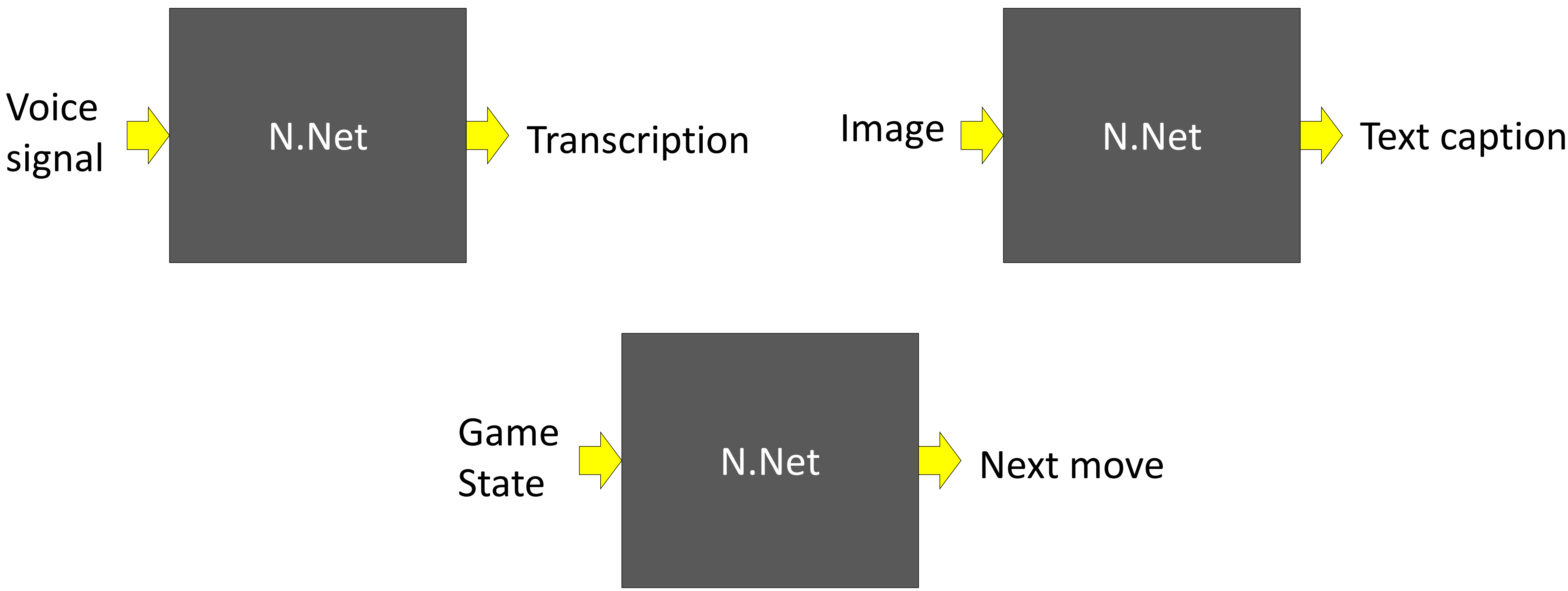
- And a variety of other problems:
  - Image analysis
  - Natural language processing
  - Speech processing
  - Even predicting stock markets!

Neural nets and the employment market

This guy didn't know about neural networks (a.k.a deep learning)

This guy learned about neural networks (a.k.a deep learning)

So what are neural networks??

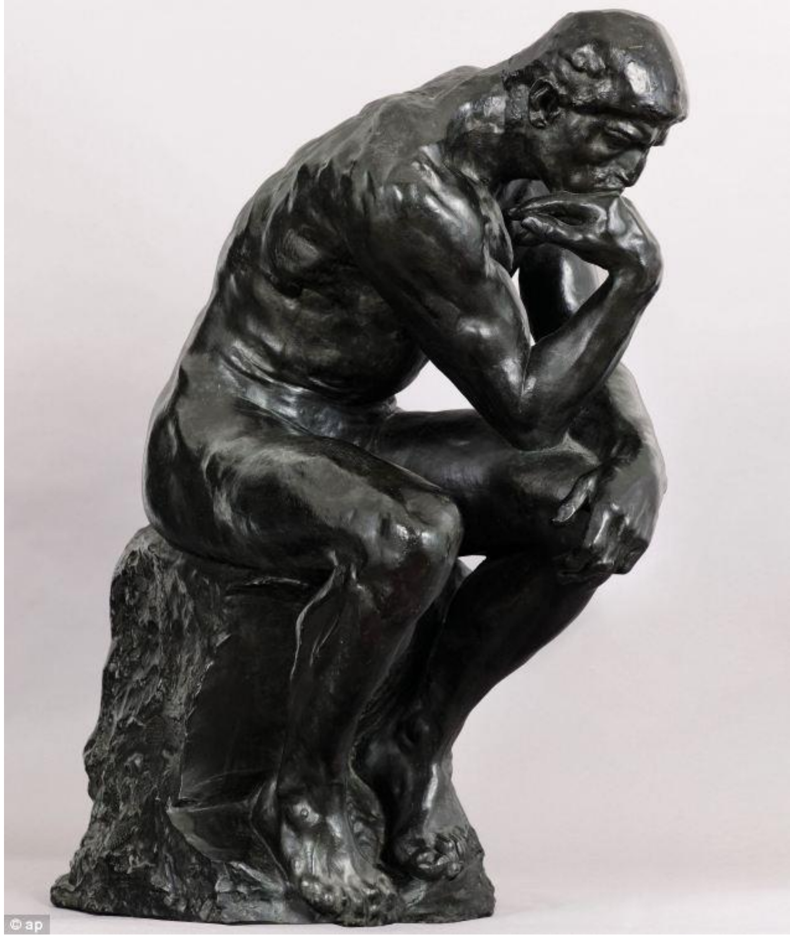


- What are these boxes?

So what are neural networks??

- It begins with this..

So what are neural networks??



"The Thinker!"
by Augustin Rodin

- Or even earlier.. with this..

The magical capacity of humans

- Humans can
  - Learn
  - Solve problems
  - Recognize patterns
  - Create
  - Cogitate
  - …

Dante!

- Worthy of emulation
- But how do humans “work“?

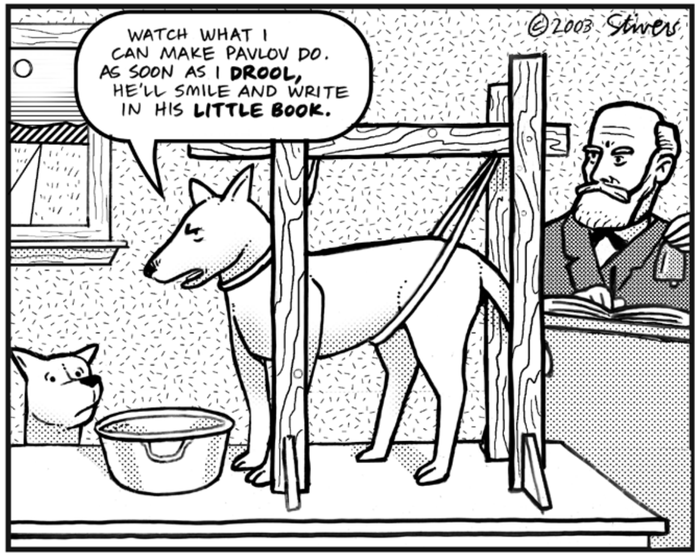
Cognition and the brain..

- “If the brain was simple enough to be understood - we would be too simple to understand it!”
  - Marvin Minsky

Early Models of Human Cognition

- Associationism
  - Humans learn through association
- 400BC-1900AD: Plato, David Hume, Ivan Pavlov..

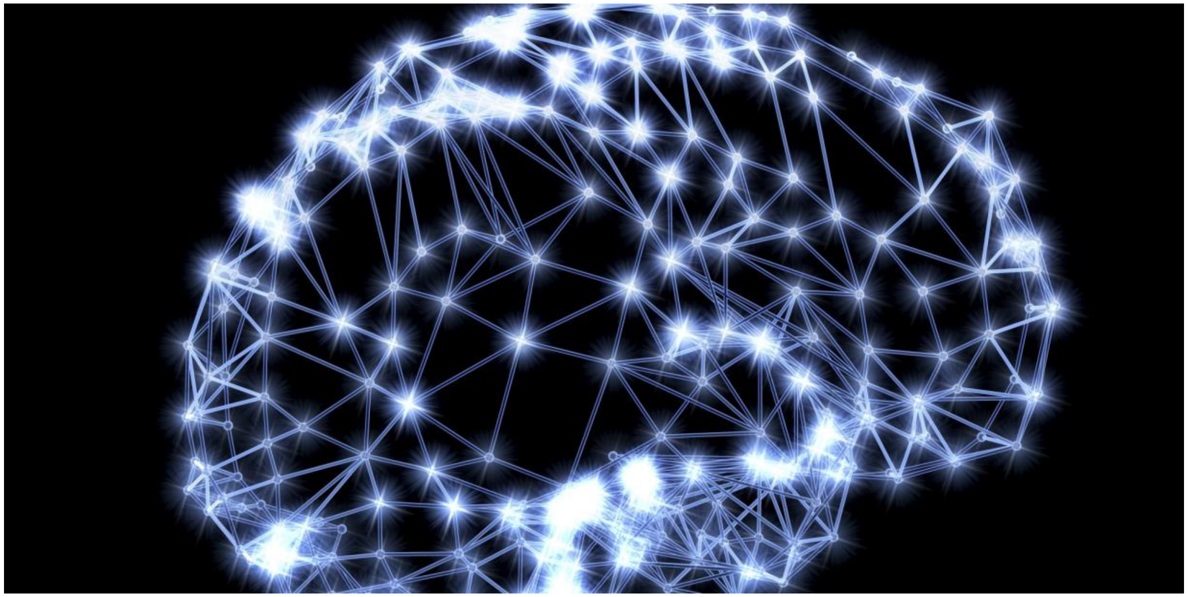
What are "Associations"



- Lightning is generally followed by thunder
  - Ergo – "hey here's a bolt of lightning, we're going to hear thunder"
  - Ergo – "We just heard thunder; did someone get hit by lightning"?

- Association!

Observation: ***The Brain***

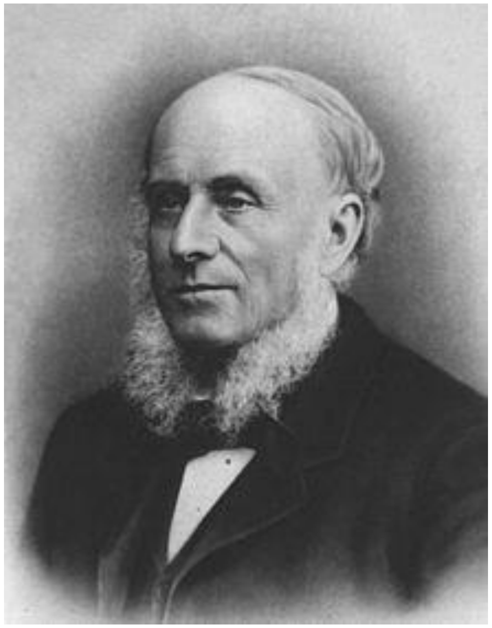


- Mid 1800s: The brain is a mass of interconnected neurons

Brain: Interconnected Neurons

- Many neurons connect *in* to each neuron
- Each neuron connects *out* to many neurons

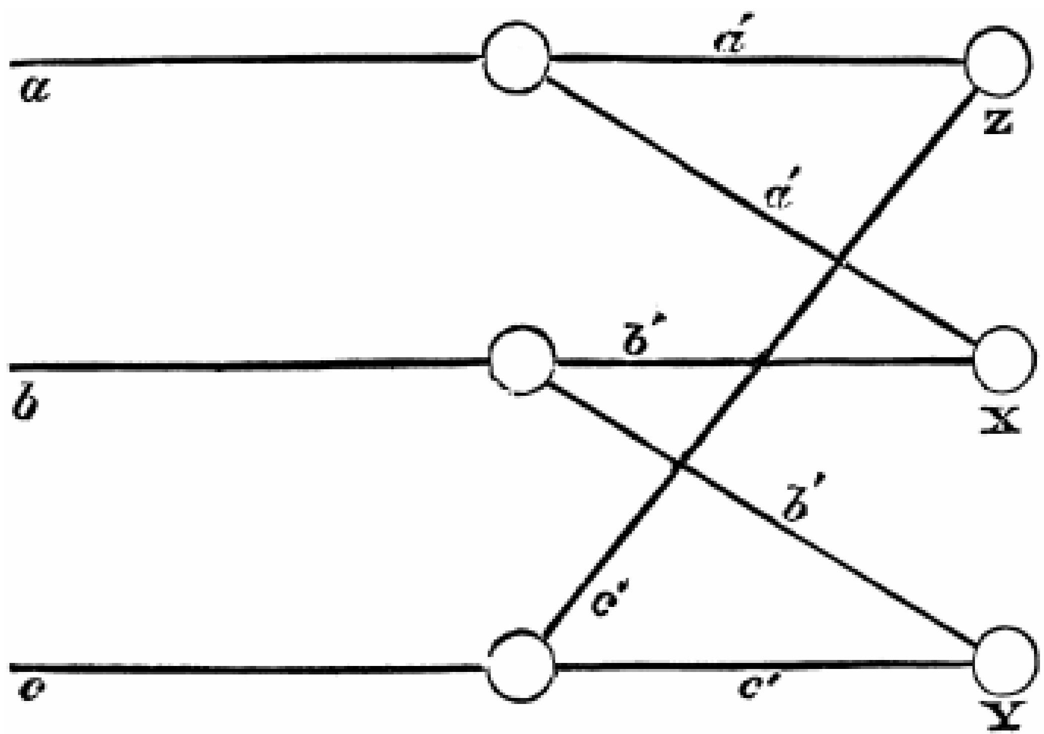
Enter ***Connectionism***



- Alexander Bain, philosopher, mathematician, logician, linguist, professor
- 1873: The information is in the *connections*
  - *The mind and body* (1873)

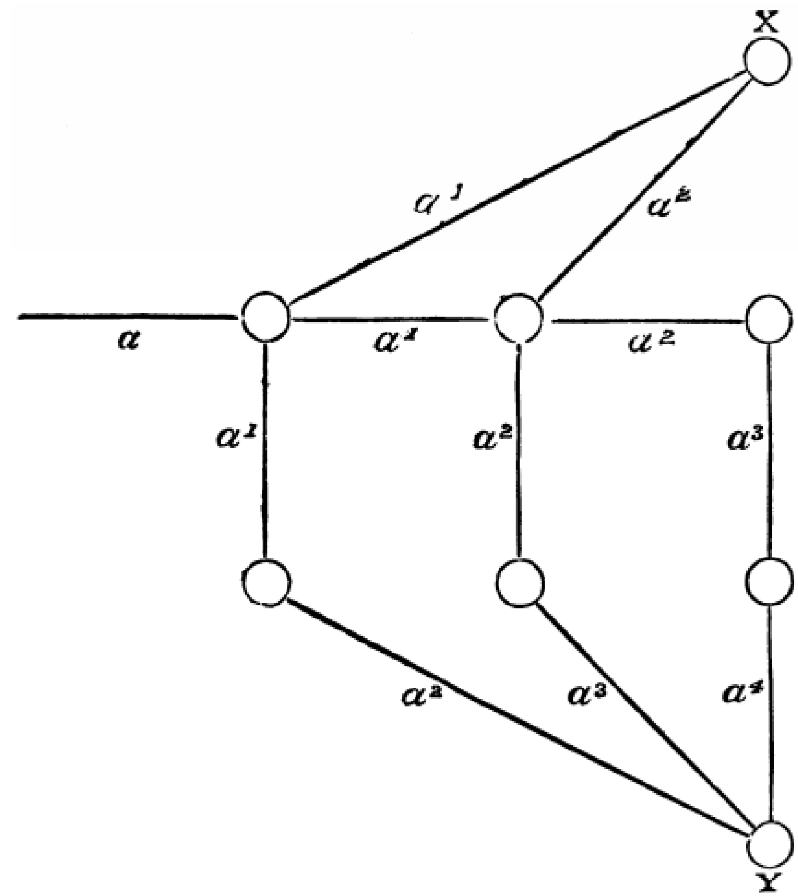
Bain's Idea : **Neural Groupings**

- Neurons excite and stimulate each other
- Different combinations of inputs can result in different outputs



Bain’s Idea : **Neural Groupings**

- Different intensities of activation of A lead to the differences in when X and Y are activated



Bain’s Idea 2: **Making Memories**

- “when two impressions concur, or closely succeed one another, the nerve currents find some bridge or place of continuity, better or worse, according to the abundance of nerve matter available for the transition.”

- Predicts “Hebbian” learning (half a century before Hebb!)

Bain's Doubts

- ***"The fundamental cause of the trouble is that in the modern world the stupid are cocksure while the intelligent are full of doubt."***
  - Bertrand Russell

- In 1873, Bain postulated that there must be one million neurons and 5 billion connections relating to 200,000 "acquisitions"

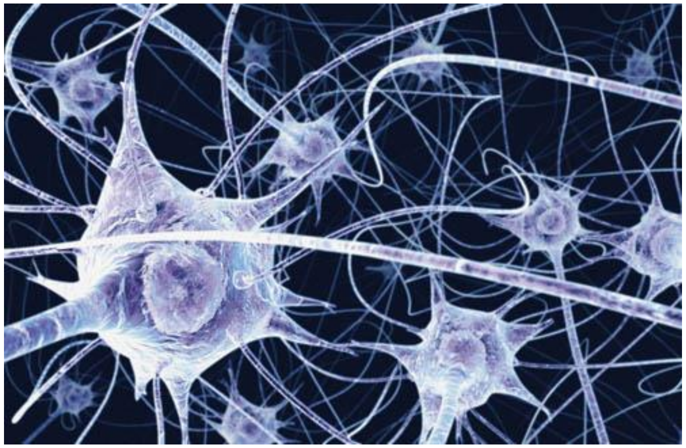
- In 1883, Bain was concerned that he hadn't taken into account the number of "partially formed associations" and the number of neurons responsible for recall/learning

- By the end of his life (1903), recanted all his ideas!
  - Too complex; the brain would need too many neurons and connections

Connectionism lives on..

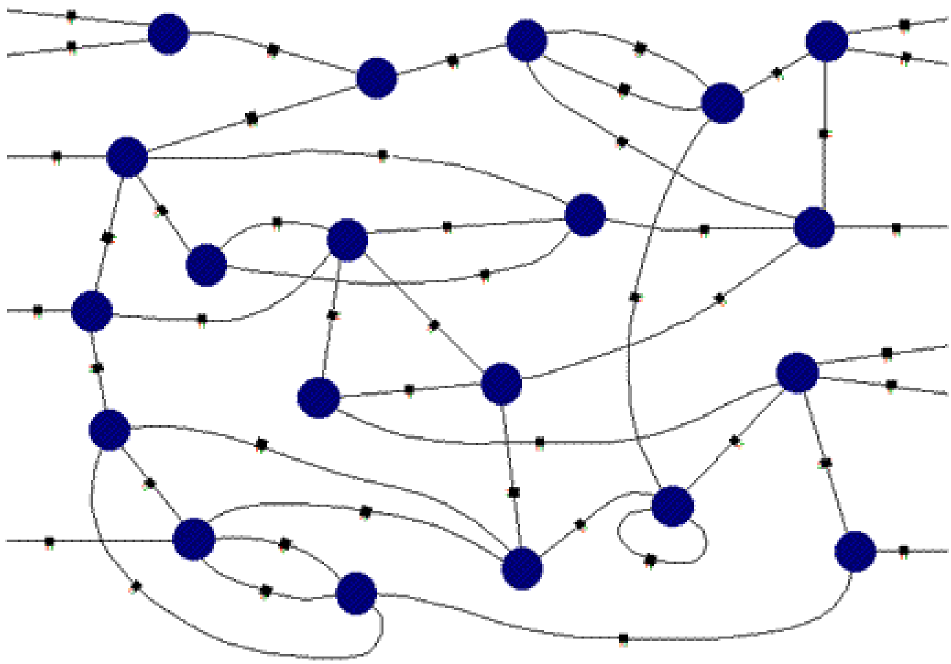
- The human brain is a connectionist machine
  - Bain, A. (1873). *Mind and body. The theories of their relation.* London: Henry King.
  - Ferrier, D. (1876). *The Functions of the Brain.* London: Smith, Elder and Co

- Neurons connect to other neurons. The processing/capacity of the brain is a function of these connections



- Connectionist machines emulate this structure

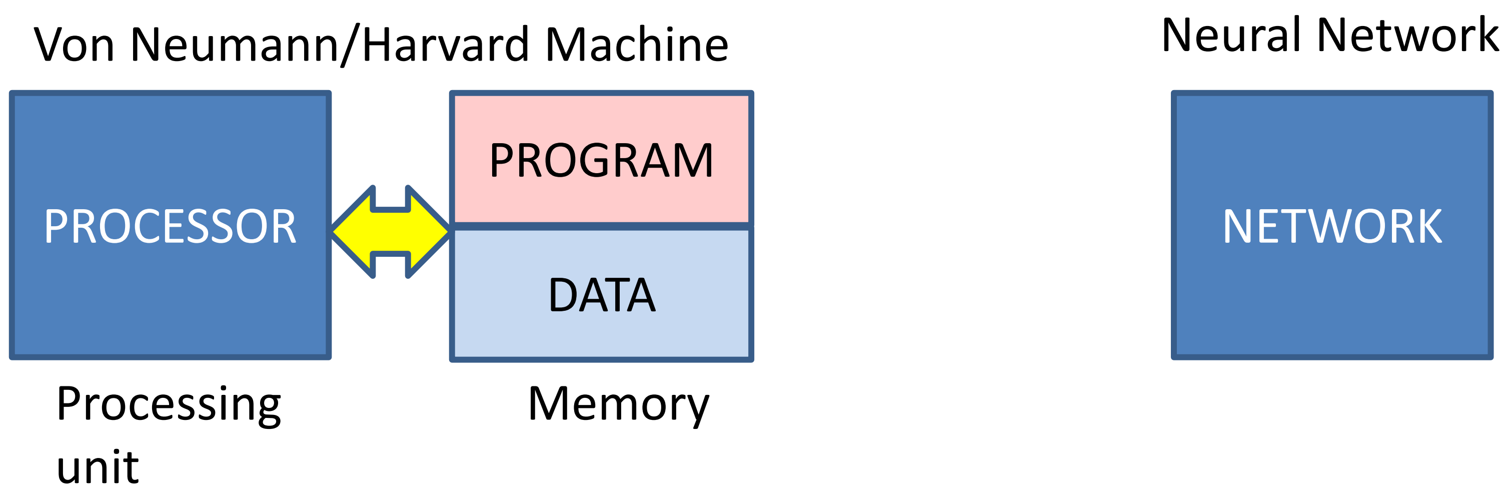
Connectionist Machines



- Network of processing elements
- **All world knowledge is stored in the *connections* between the elements**

Connectionist Machines

- Neural networks are *connectionist* machines
  - As opposed to Von Neumann Machines

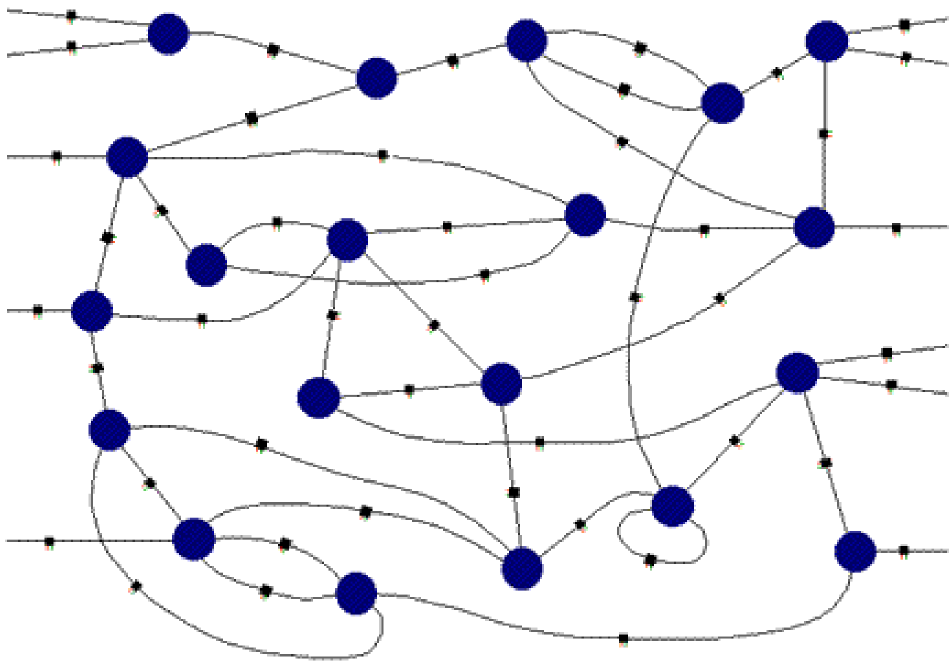


- The machine has many non-linear processing units
  - The program is the connections between these units
    - Connections may also define memory

Recap

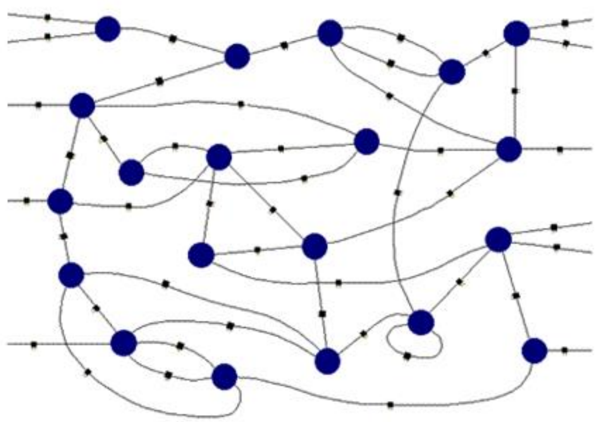
- Neural network based AI has taken over most AI tasks
- Neural networks originally began as computational models of the brain
  - Or more generally, models of cognition
- The earliest model of cognition was *associationism*
- The more recent model of the brain is *connectionist*
  - Neurons connect to neurons
  - The workings of the brain are encoded in these connections
- Current neural network models are *connectionist machines*

Connectionist Machines



- Network of processing elements
- All world knowledge is stored in the *connections* between the elements

Connectionist Machines

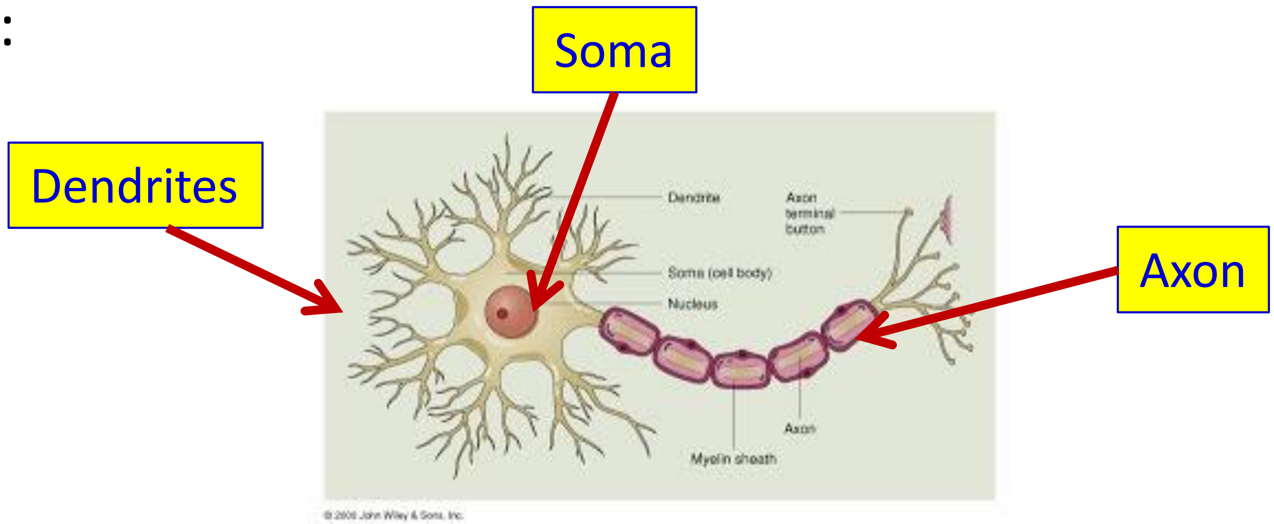


- Connectionist machines are networks of units..

- We need a model for the *units*

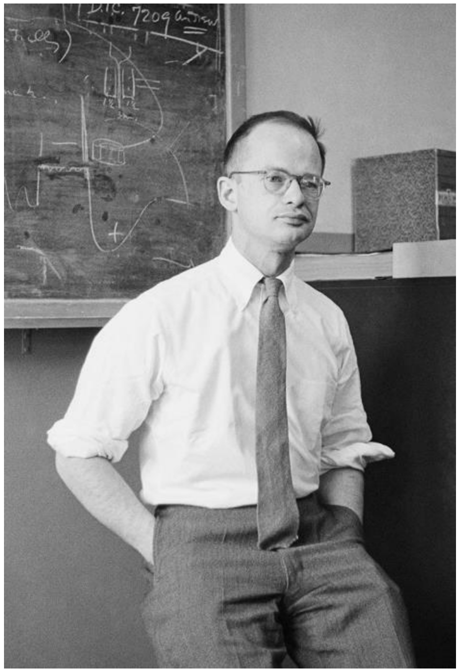
Modelling the brain

- What are the units?
- A neuron:



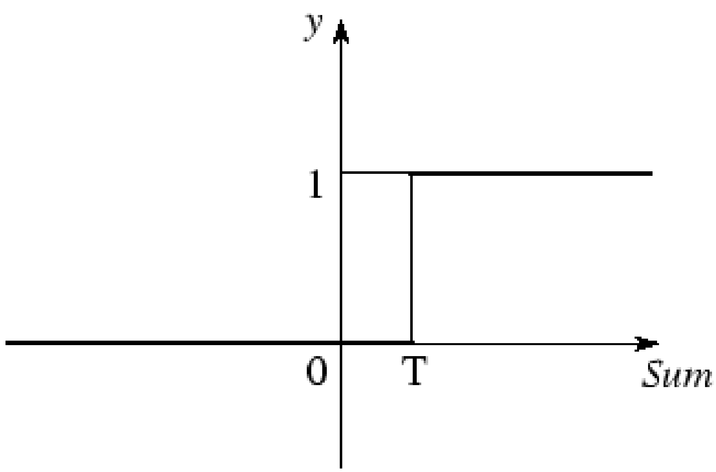
- Signals come in through the dendrites into the Soma
- A signal goes out via the axon to other neurons
  - Only one axon per neuron
- Factoid that may only interest me: Neurons do not undergo cell division

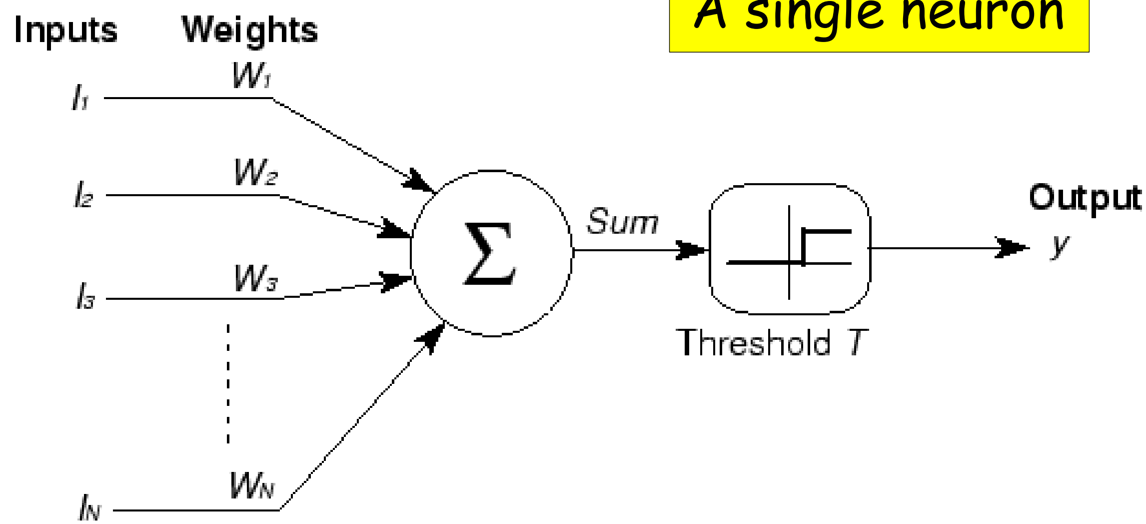
McCullough and Pitts



- The Doctor and the Hobo..
  - Warren McCulloch: Neurophysician
  - Walter Pitts: Homeless wannabe logician who arrived at his door

The McCulloch and Pitts model





A single neuron

- A mathematical model of a neuron
  - McCulloch, W.S. & Pitts, W.H. (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, 5:115-137, 1943
    - Pitts was only 20 years old at this time
  - Threshold Logic

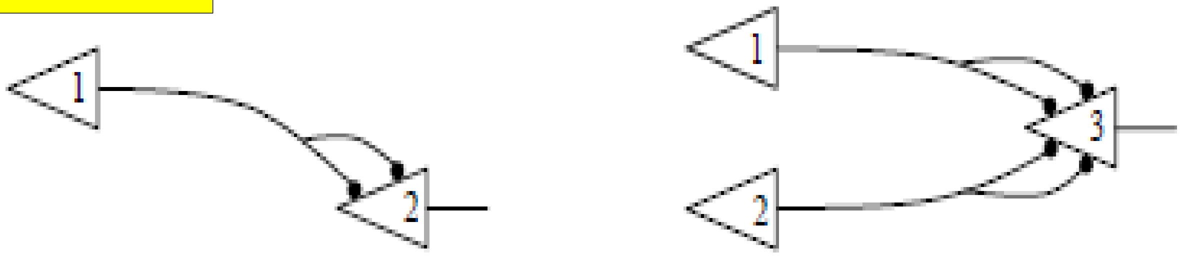
Synaptic Model

- *Excitatory synapse:* Transmits weighted input to the neuron

- *Inhibitory synapse:* Any signal from an inhibitory synapse forces output to zero
  - The activity of any inhibitory synapse absolutely prevents excitation of the neuron at that time.
    - Regardless of other inputs

Simple "networks"
of neurons can perform
Boolean operations

Boolean Gates



$N_2(t)) \leftrightarrow N_1(t\text{-}1)$
net for temporal predecessor

$N_3(t) \leftrightarrow N_1(t\text{-}1) \vee N_2(t\text{-}1)$
net for disjunction

$N_3(t) \leftrightarrow N_1(t\text{-}1) \& N_2(t\text{-}1)$
net for conjunction

$N_3(t) \leftrightarrow N_1(t\text{-}1) \& \sim N_2(t\text{-}1)$
net for conjunction and negation

Figure 1. Diagrams of McCulloch and Pitts nets. In order to send an output pulse, each neuron must receive two excitory inputs and no inhibitory inputs. Lines ending in a dot represent excitatory connections; lines ending in a hoop represent inhibitory connections.

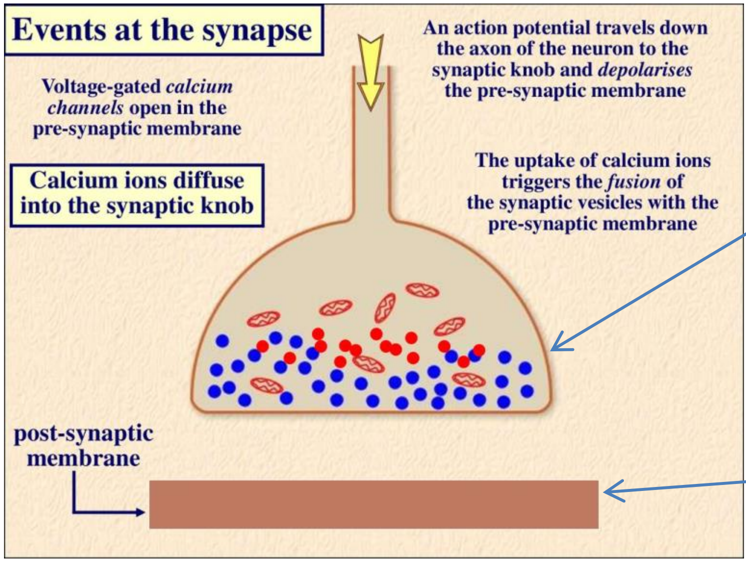
Criticisms

- Several..
  - Claimed their machine could emulate a Turing machine
- Didn’t provide a learning mechanism..

Donald Hebb

- "Organization of behavior", 1949
- A learning mechanism:
  - Neurons that fire together wire together

Hebbian Learning



Axonal connection from neuron X

Dendrite of neuron Y

- If neuron x_i repeatedly triggers neuron y, the synaptic knob connecting x_i to y gets larger
- In a mathematical model:

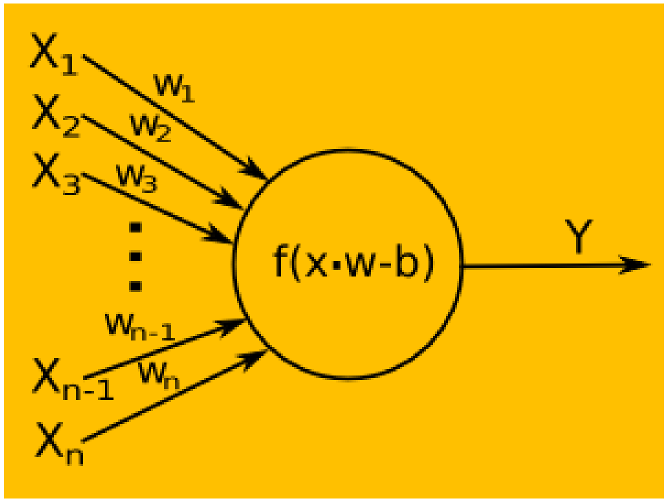
$$w_i = w_i + \eta x_i y$$

  - Weight of i^{th} neuron's input to output neuron y
- This simple formula is actually the basis of many learning algorithms in ML

A better model

- Frank Rosenblatt
  - Psychologist, Logician
  - Inventor of the solution to everything, aka the Perceptron (1958)

Simplified mathematical model

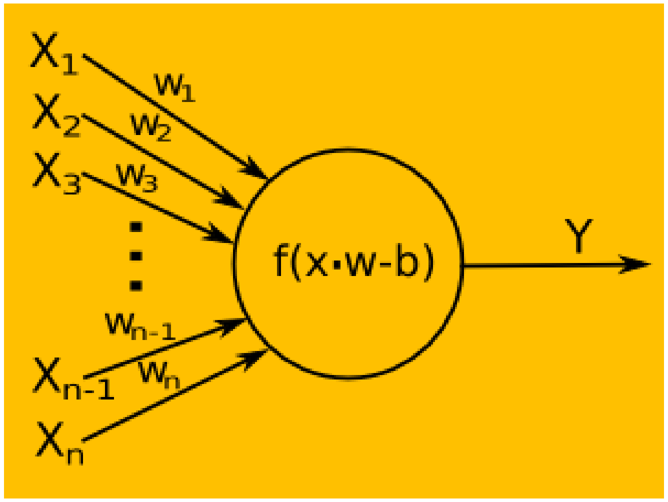


- Number of inputs combine linearly
  - Threshold logic: Fire if combined input exceeds threshold

$$Y = \begin{cases} 1 \;\; if \; \sum_i w_i x_i + b > 0 \\ 0 \qquad\qquad\qquad else \end{cases}$$

His "Simple" Perceptron

- Originally assumed could represent *any* Boolean circuit and perform any logic
  - *"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence,"* New York Times (8 July) 1958
  - *"Frankenstein Monster Designed by Navy That Thinks,"* Tulsa, Oklahoma Times 1958



Also provided a learning algorithm

$$\mathbf{w} = \mathbf{w} + \eta(d(\mathbf{x}) - y(\mathbf{x}))\mathbf{x}$$

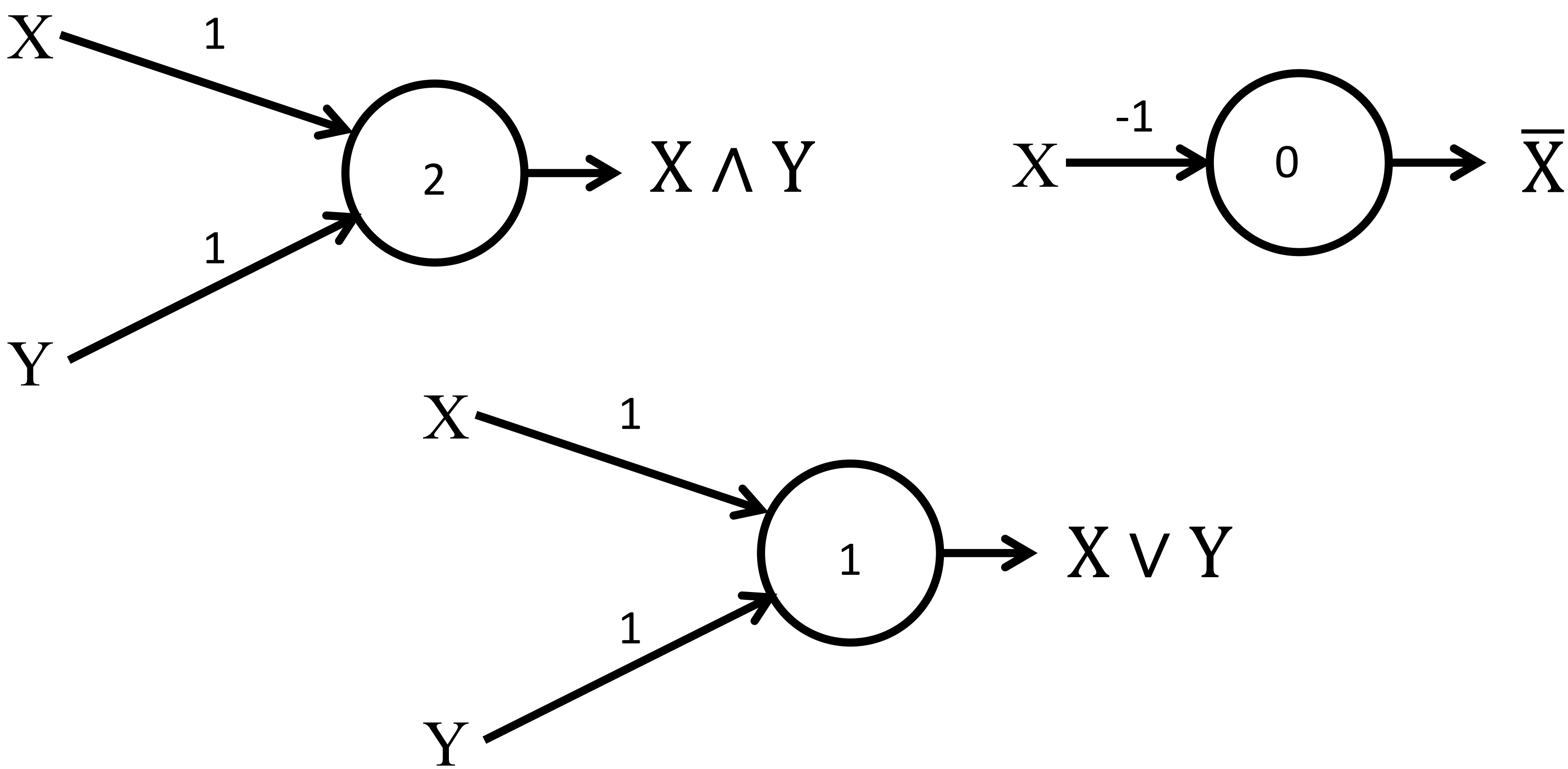
Sequential Learning:

$d(x)$ is the desired output in response to input x

$y(x)$ is the actual output in response to x

- Boolean tasks
- Update the weights whenever the perceptron output is wrong
- Proved convergence

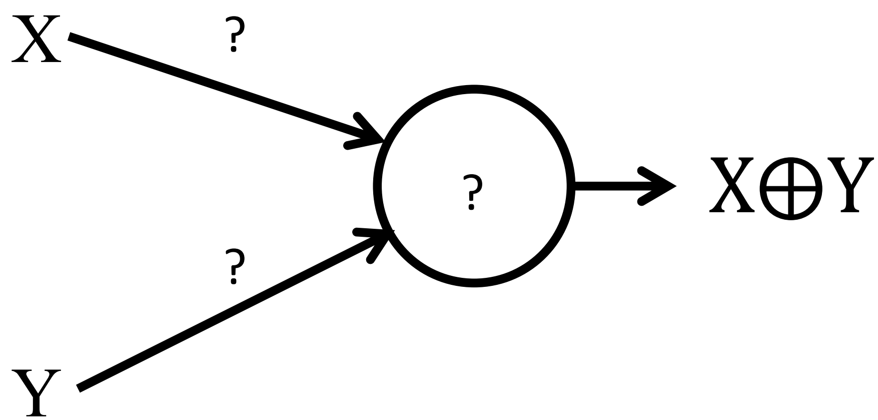
Perceptron



- Easily shown to mimic any Boolean gate
- But…

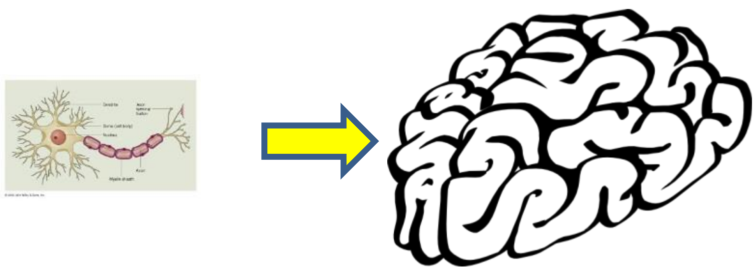
Perceptron

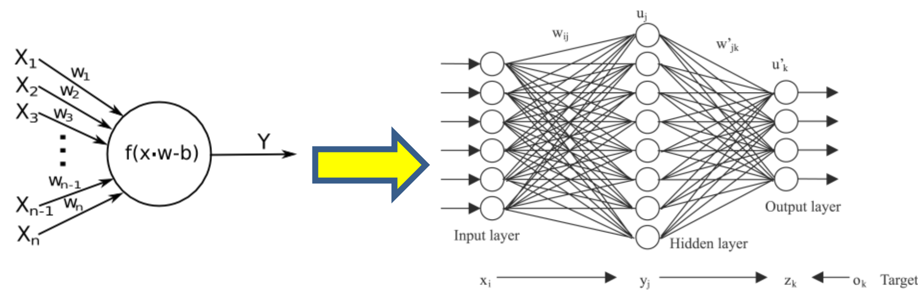
No solution for XOR!
Not universal!



- Minsky and Papert, 1968

A single neuron is not enough

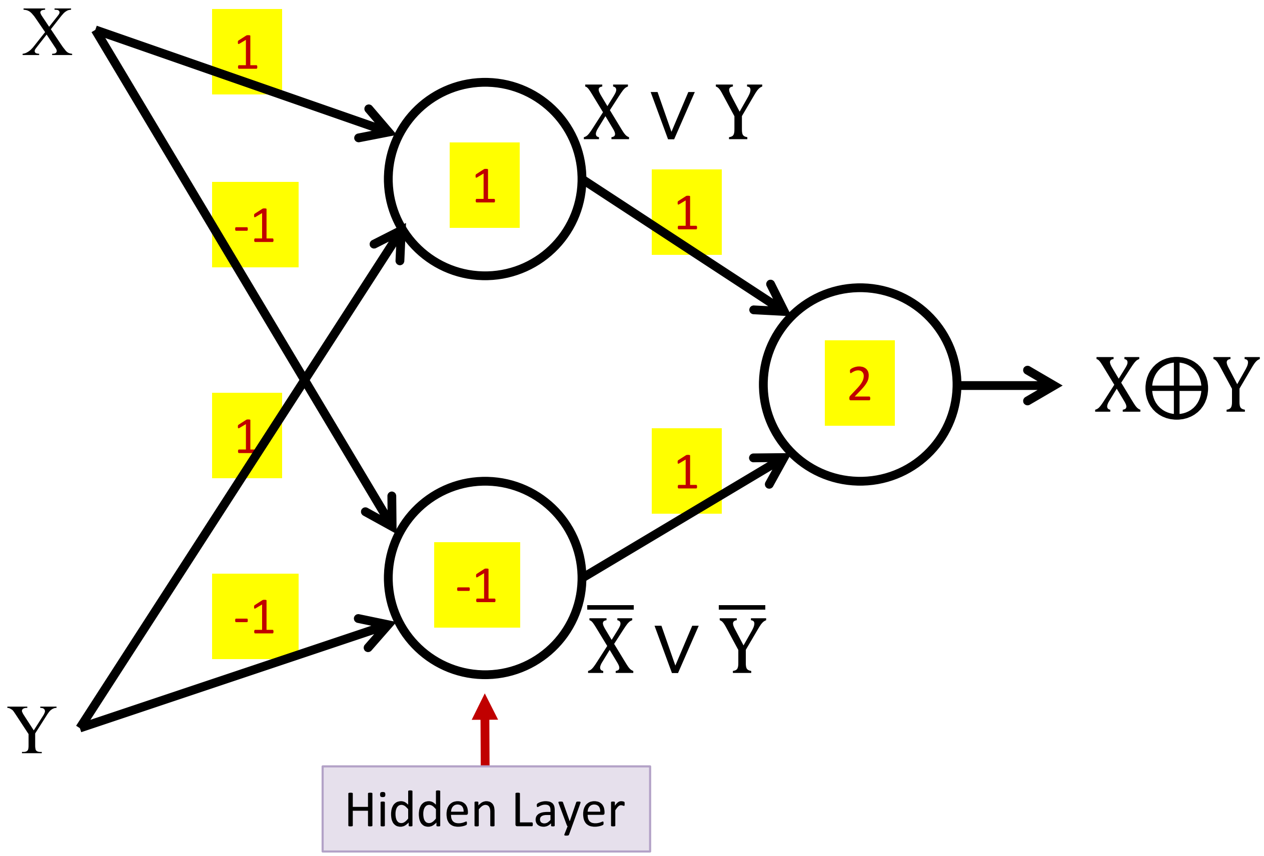




- Individual elements are weak computational elements
  - Marvin Minsky and Seymour Papert, 1969, *Perceptrons: An Introduction to Computational Geometry*

- *Networked* elements are required

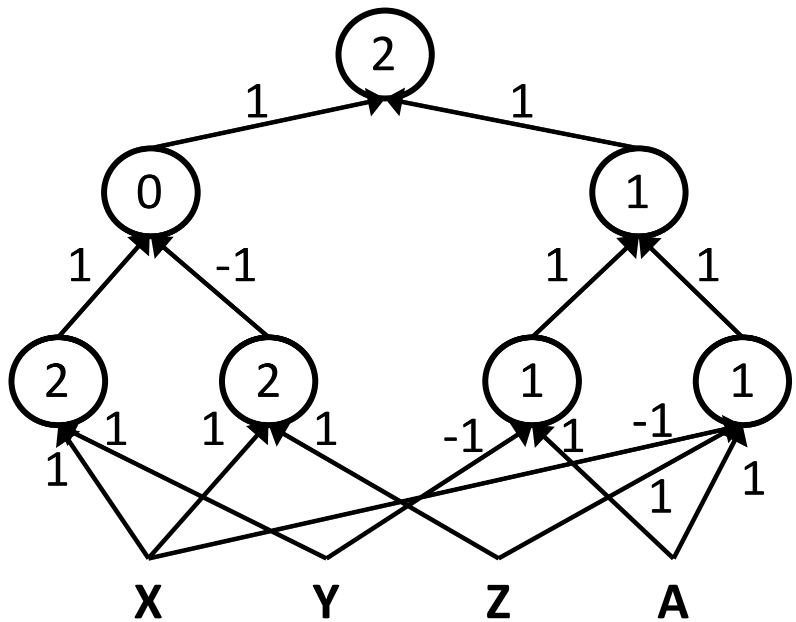
Multi-layer Perceptron!



- XOR
  - The first layer is a “hidden” layer
  - Also originally suggested by Minsky and Papert, 1968

A more generic model

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

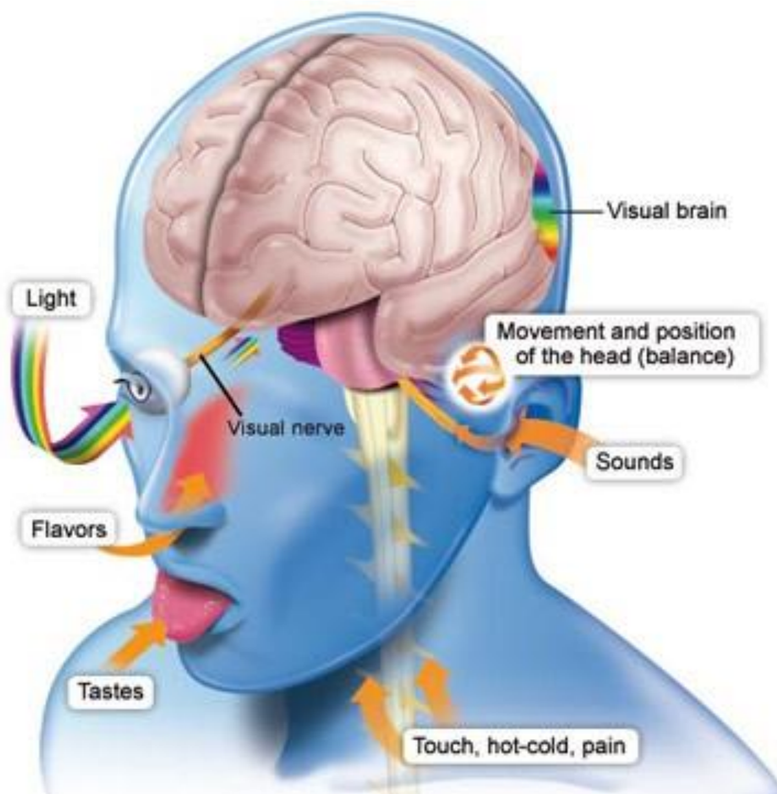


- A "multi-layer" perceptron
- Can compose arbitrarily complicated Boolean functions!
  - More on this in the next part

Story so far

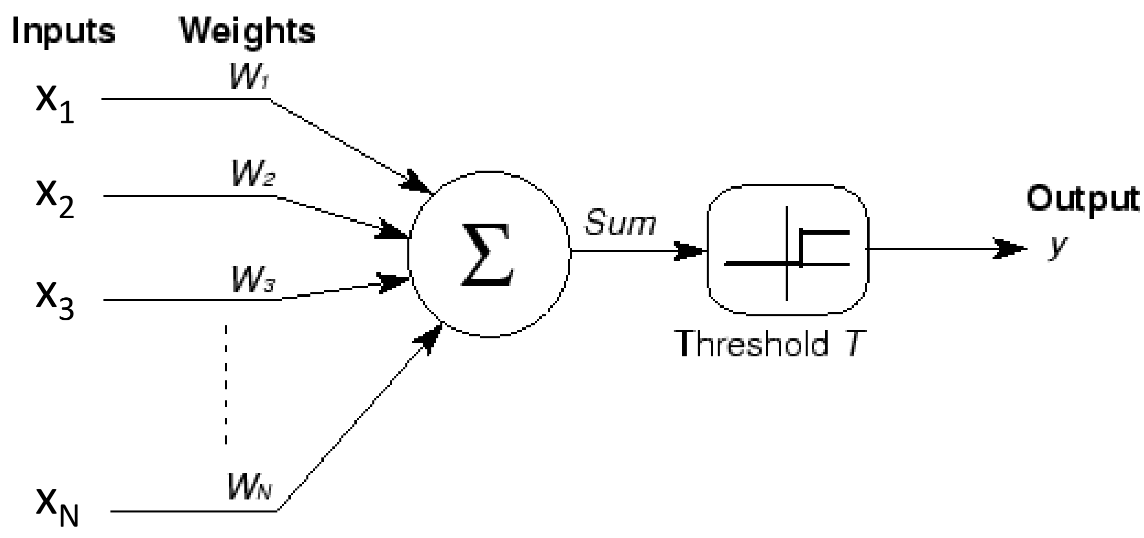
- Neural networks began as computational models of the brain
- Neural network models are *connectionist machines*
  - The comprise networks of neural units
- McCullough and Pitt model: Neurons as Boolean threshold units
  - Models the brain as performing propositional logic
  - But no learning rule
- Hebb's learning rule: Neurons that fire together wire together
  - Unstable
- Rosenblatt's perceptron : A variant of the McCulloch and Pitt neuron with a provably convergent learning rule
  - But individual perceptrons are limited in their capacity (Minsky and Papert)
- Multi-layer perceptrons can model arbitrarily complex Boolean functions

But our brain is not Boolean



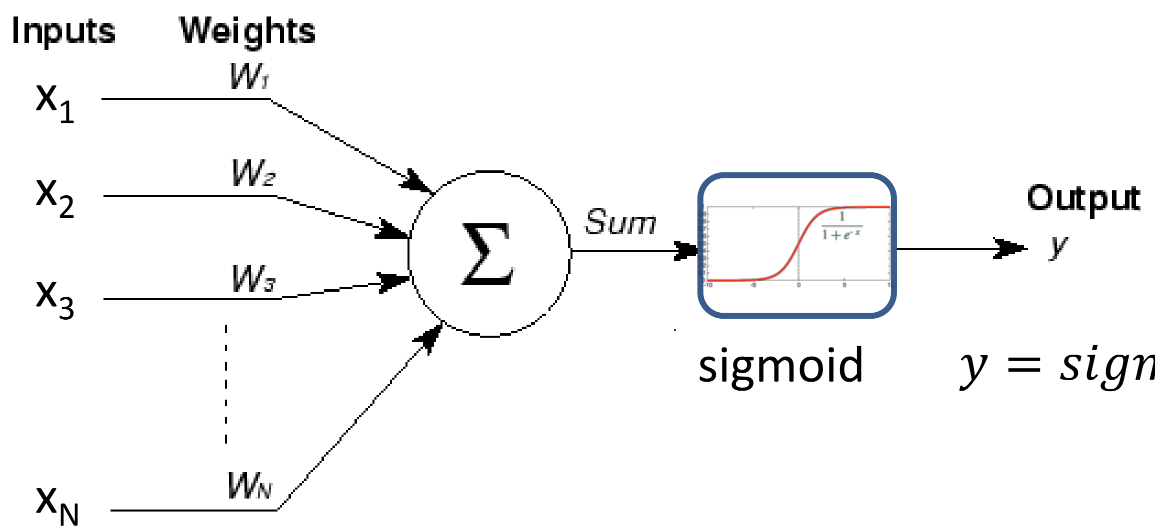
- We have real inputs
- We make non-Boolean inferences/predictions

The perceptron with *real* inputs



- $x_1 \ldots x_N$ are real valued
- $W_1 \ldots W_N$ are real valued
- Unit "fires" if weighted input exceeds a threshold

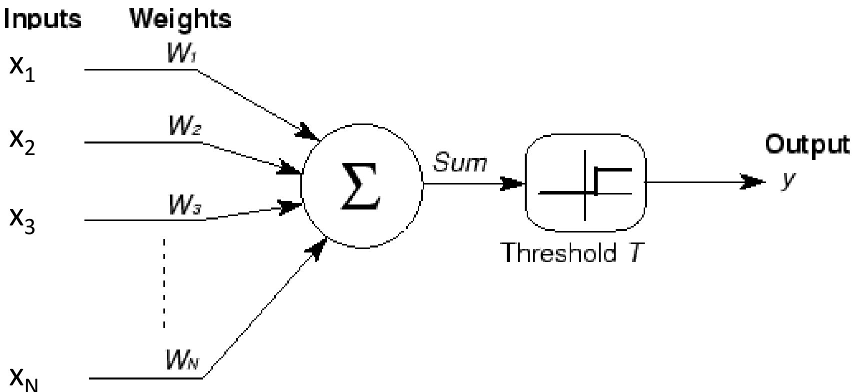
The perceptron with *real* inputs and a real *output*



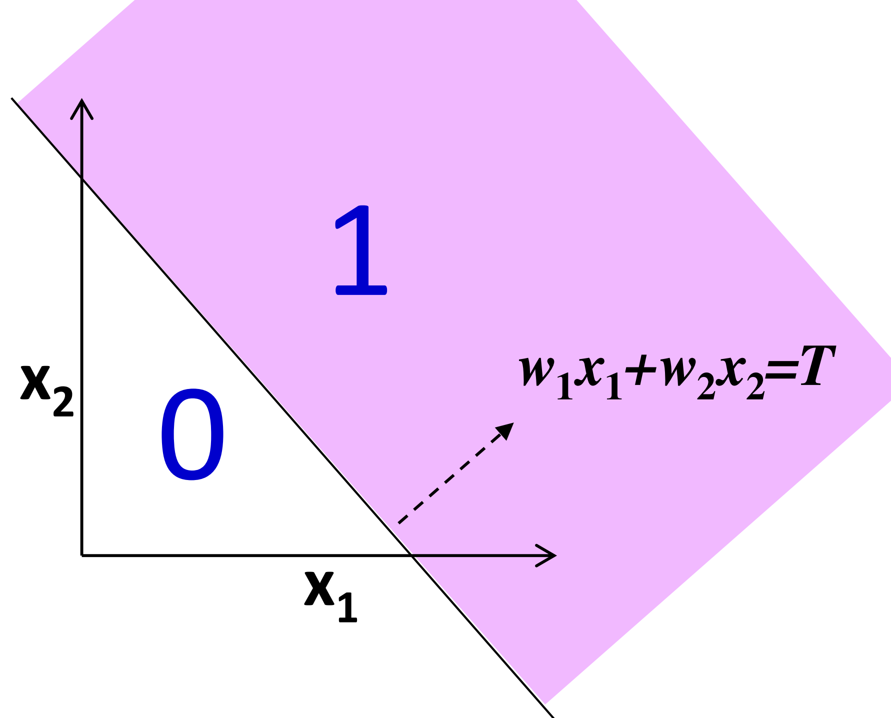
$$y = sigmoid\left(\sum_i w_i x_i\right)$$

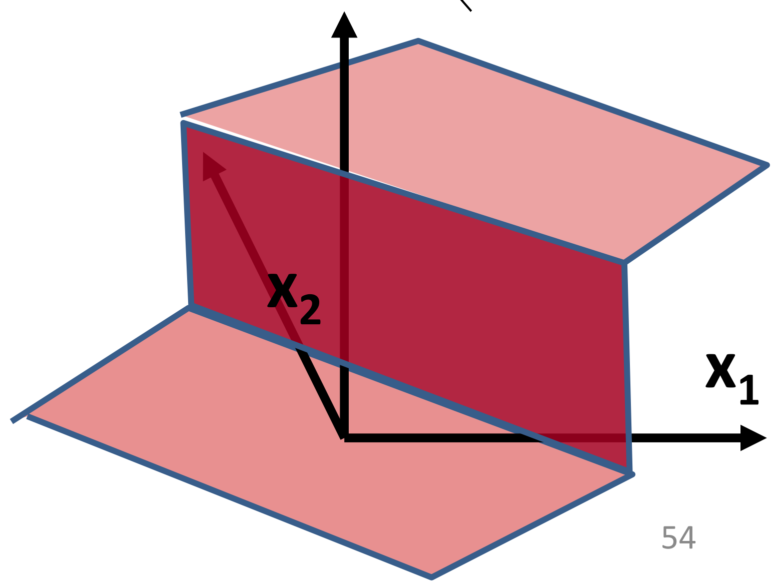
- $x_1 \ldots x_N$ are real valued
- $W_1 \ldots W_N$ are real valued
- The output y can also be real valued
  - Sometimes viewed as the "probability" of firing
  - *Is useful to continue assuming Boolean outputs though*

A Perceptron on Reals



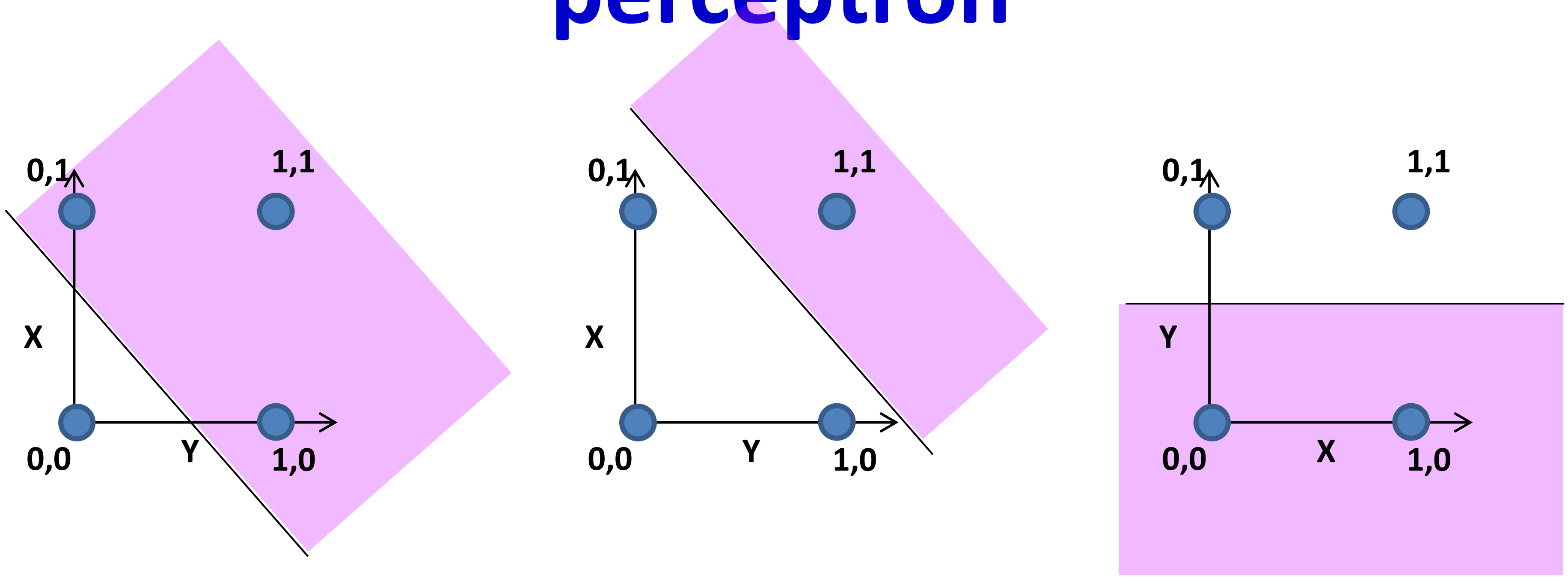
$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$





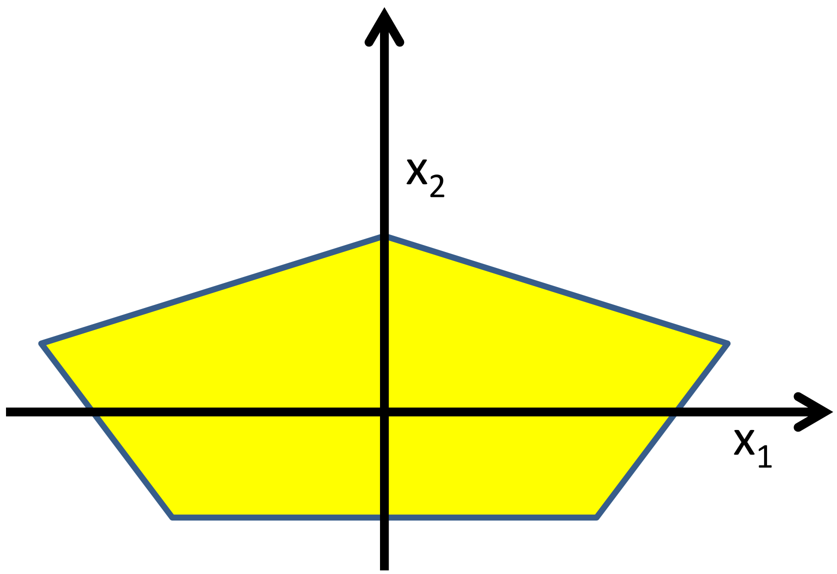
- A perceptron operates on *real*-valued vectors
  - **This is a *linear classifier***

Boolean functions with a real perceptron



- Boolean perceptrons are also linear classifiers
  - Purple regions have output 1 in the figures
  - What are these functions
  - Why can we not compose an XOR?

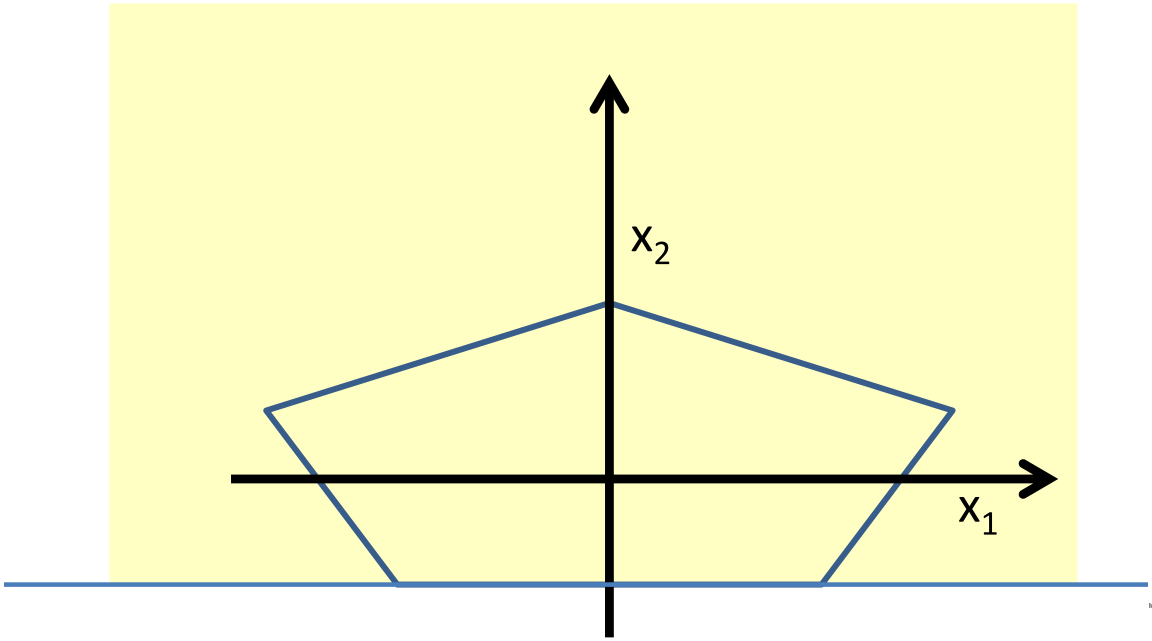
Composing complicated "decision" boundaries

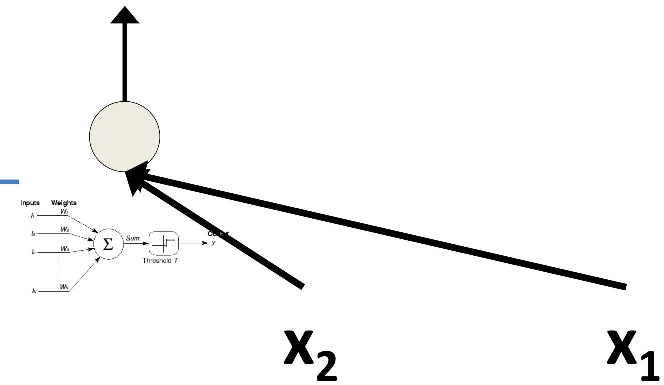


Can now be composed into "networks" to compute arbitrary classification "boundaries"

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

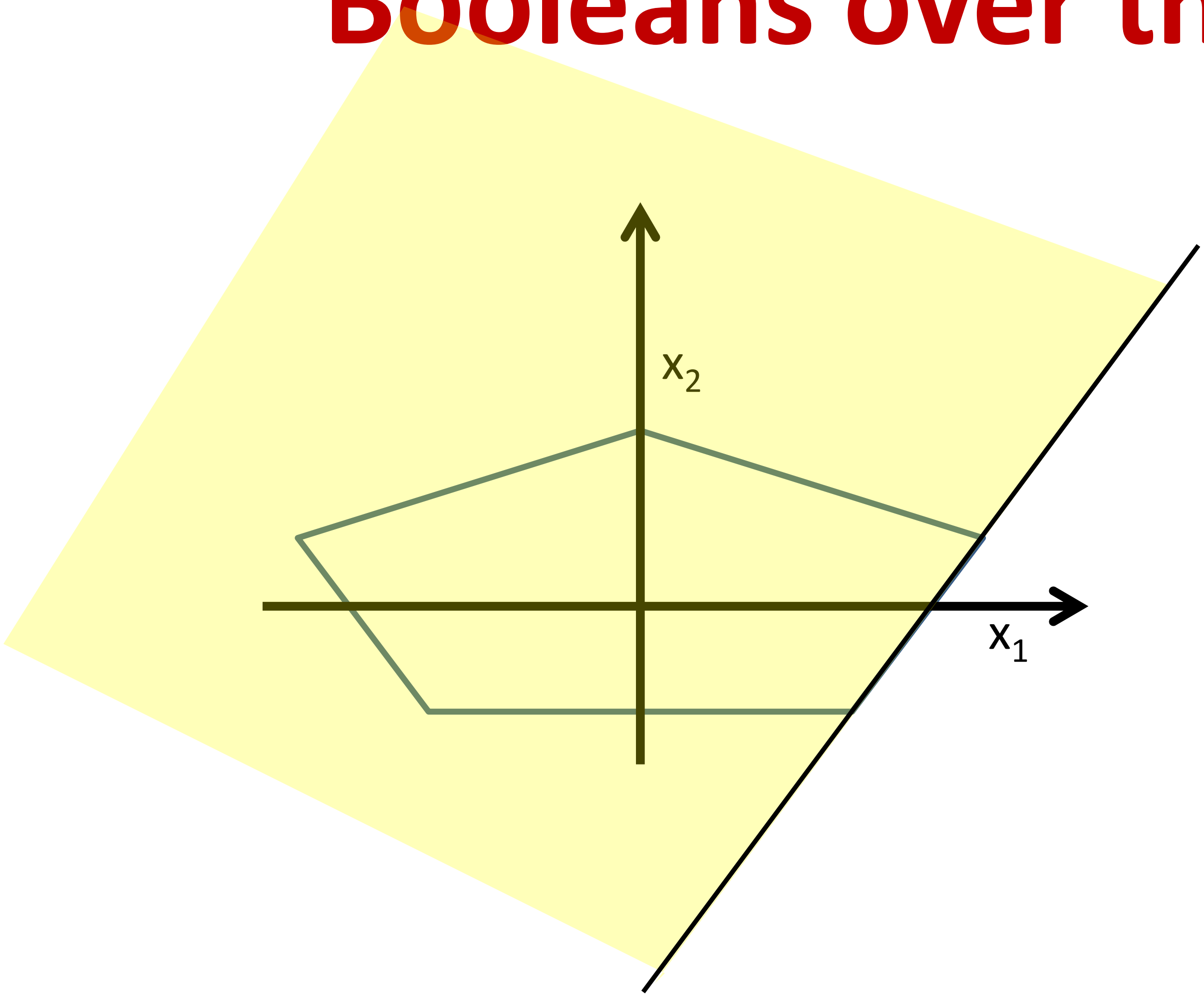
Booleans over the reals

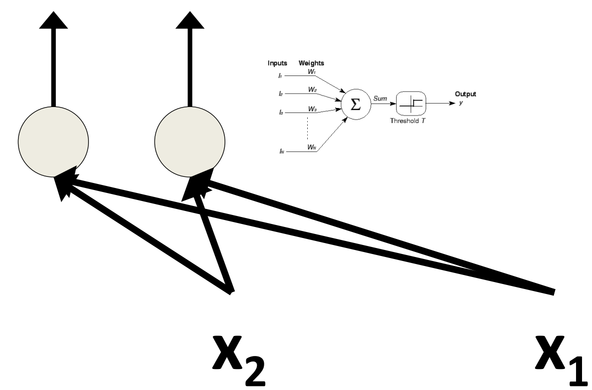




- The network must fire if the input is in the coloured area

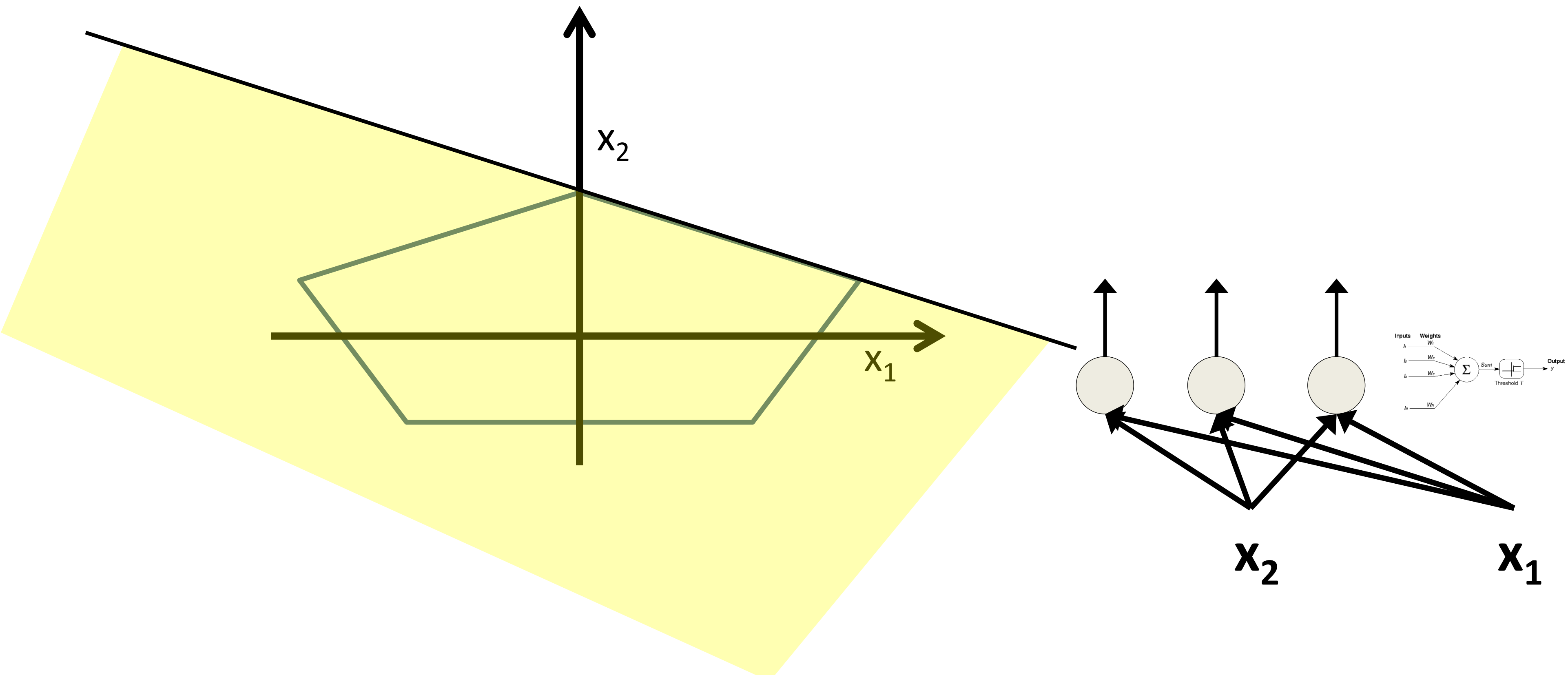
Booleans over the reals





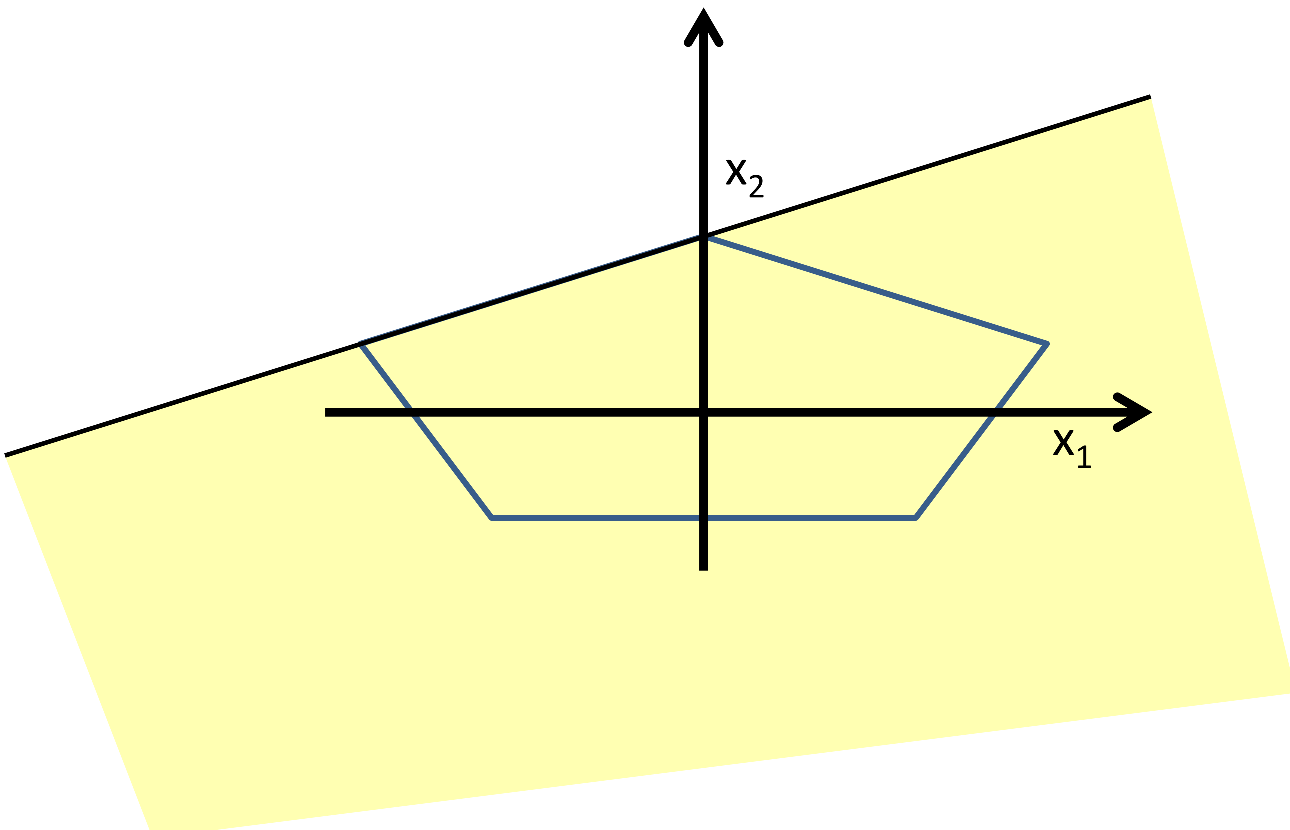
- The network must fire if the input is in the coloured area

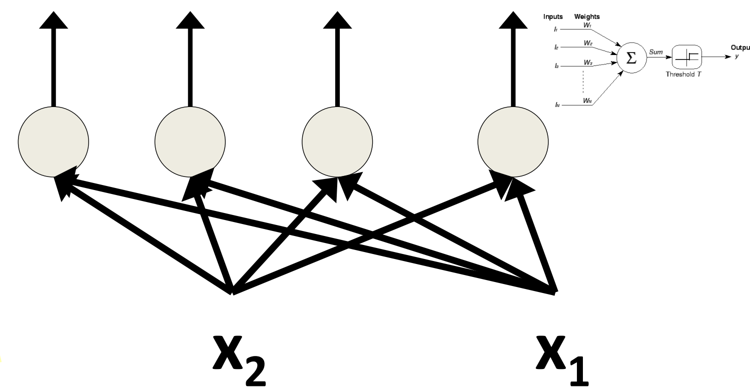
Booleans over the reals



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

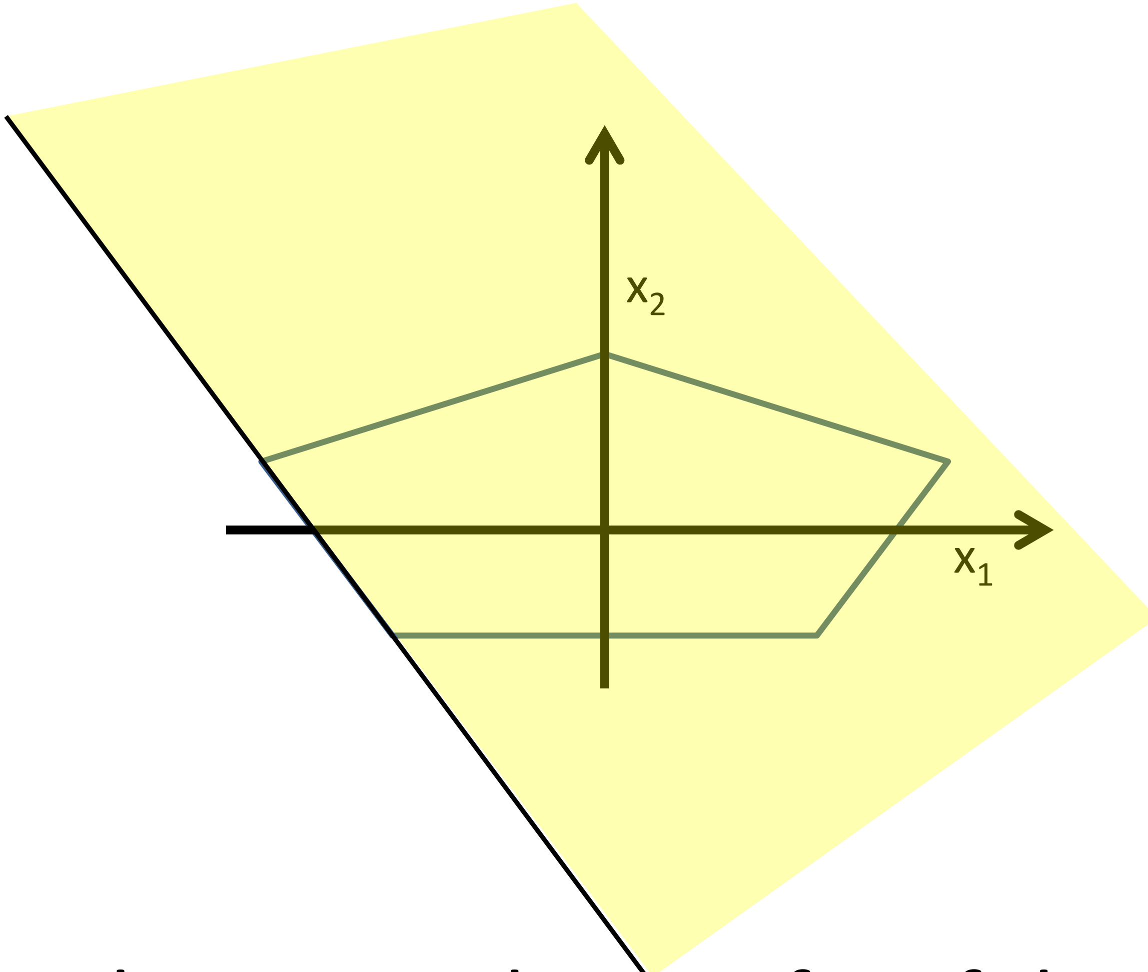
Booleans over the reals

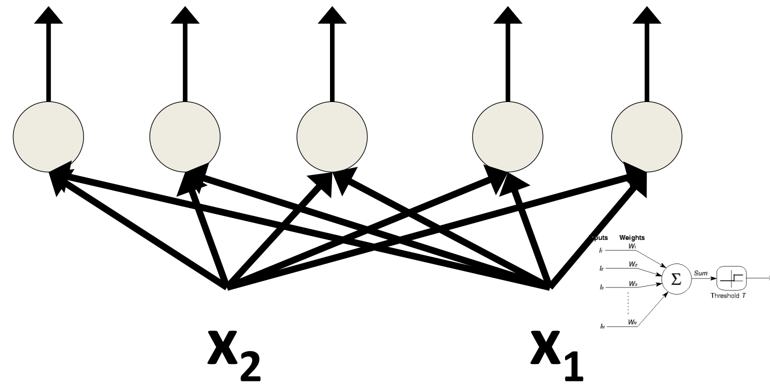




- The network must fire if the input is in the coloured area

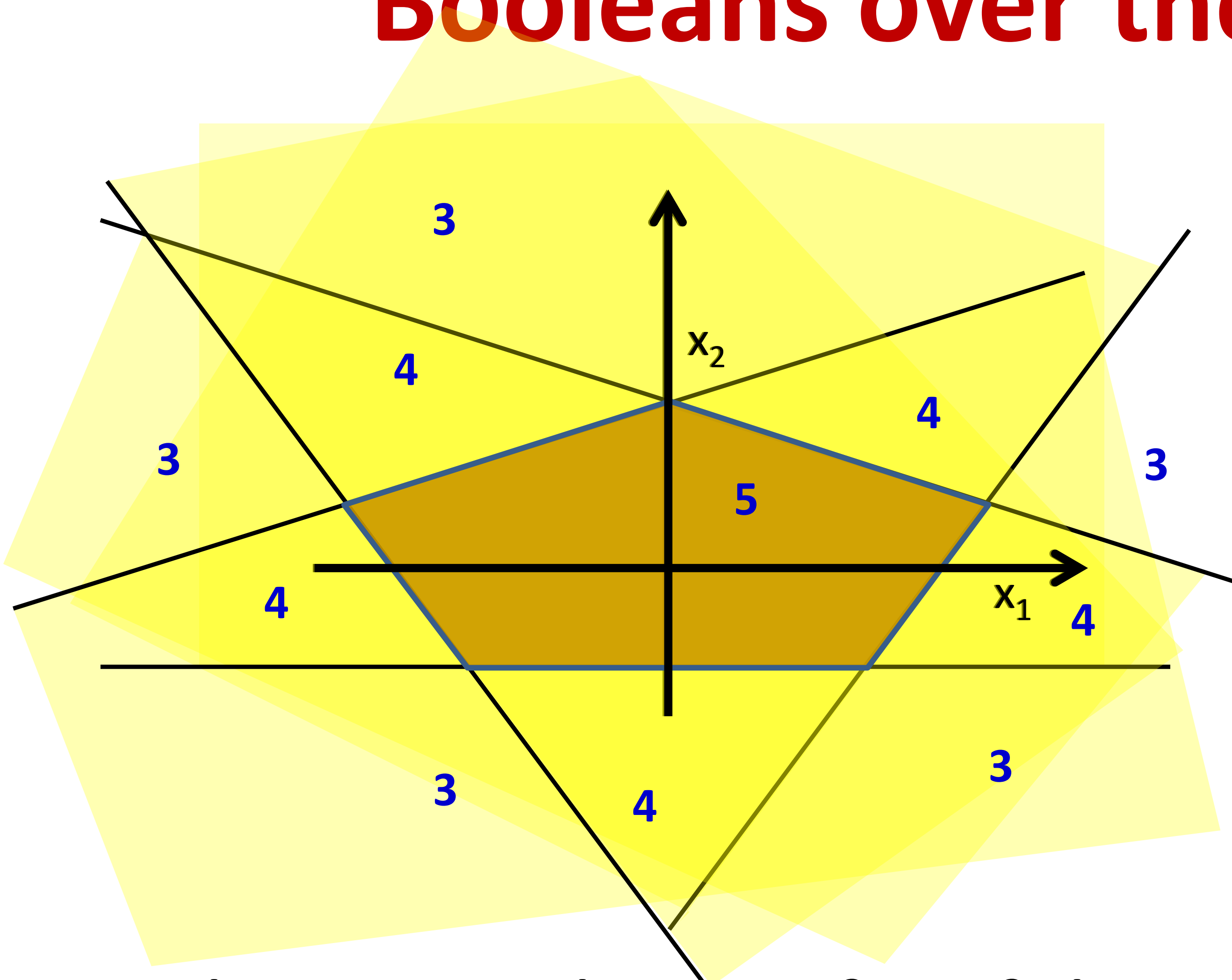
Booleans over the reals

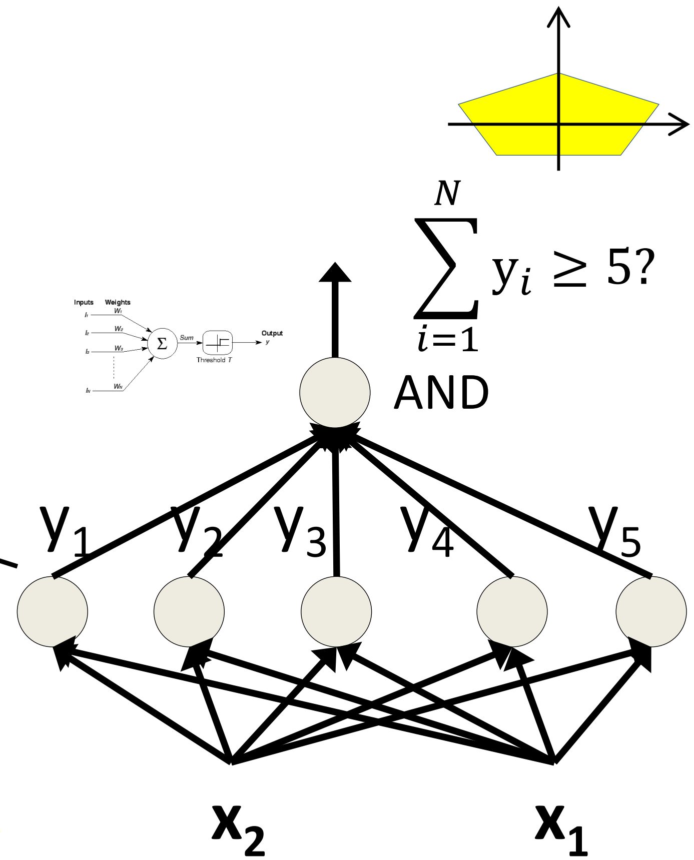




- The network must fire if the input is in the coloured area

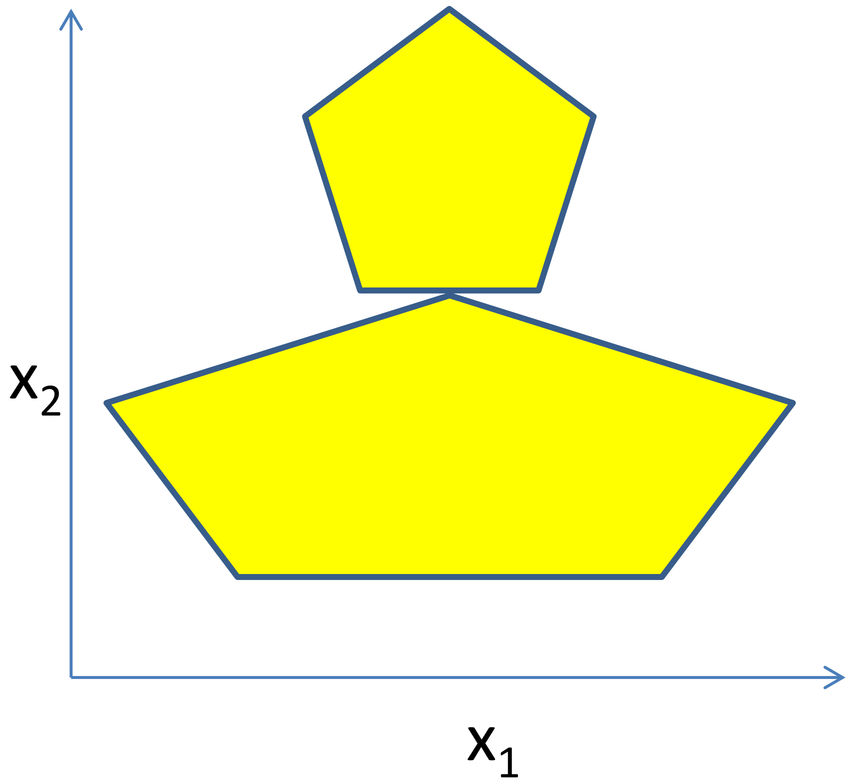
Booleans over the reals

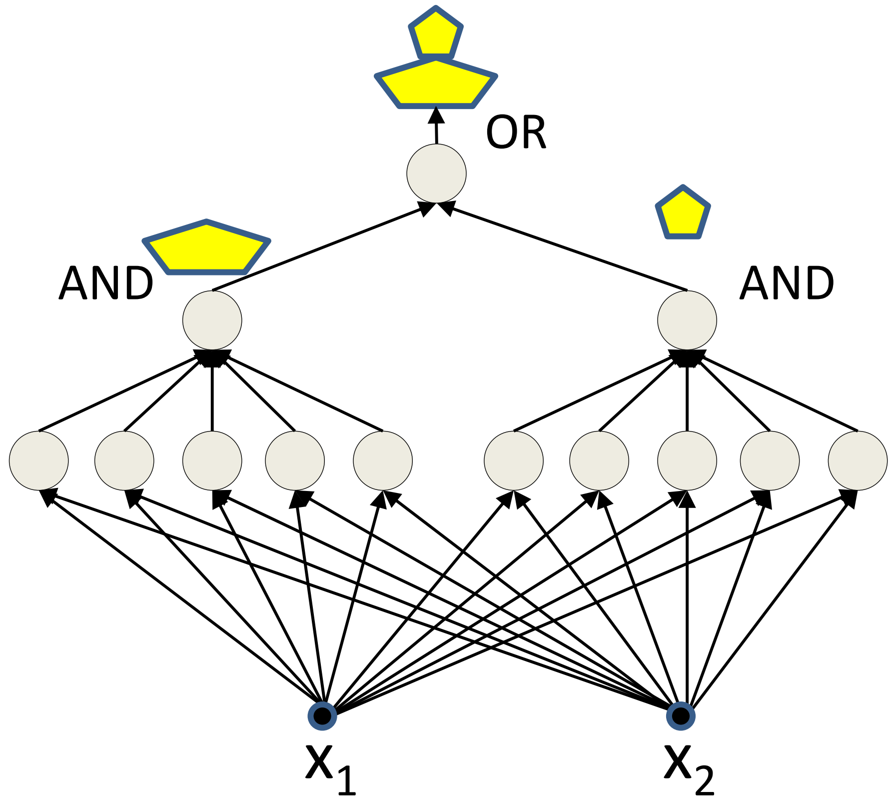




- The network must fire if the input is in the coloured area

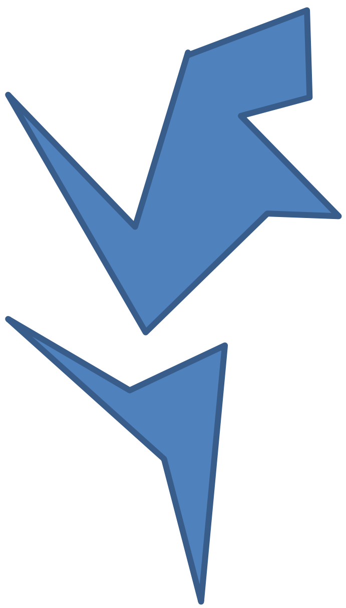
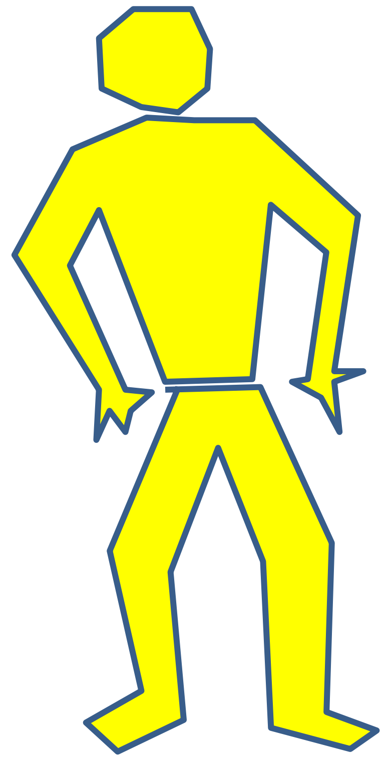
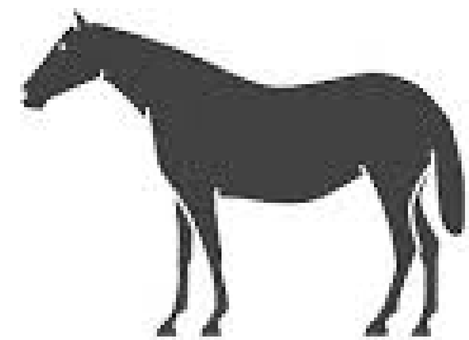
More complex decision boundaries





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

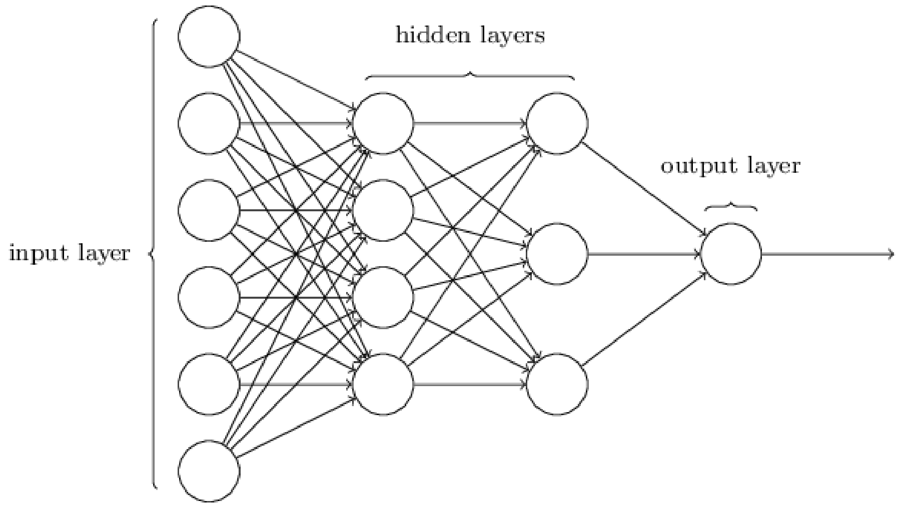
Complex decision boundaries

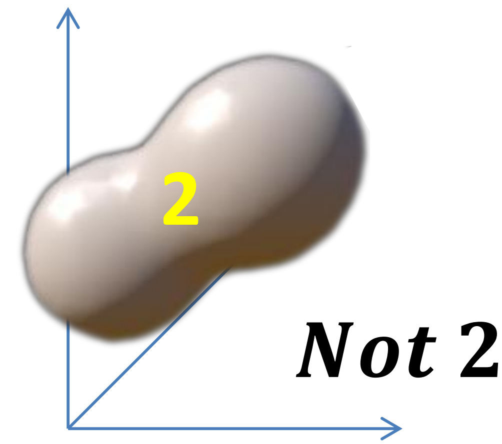




- Can compose very complex decision boundaries
  - How complex exactly? More on this in the next part

Complex decision boundaries

784 dimensions
(MNIST)





784 dimensions

- Classification problems: finding decision boundaries in high-dimensional space

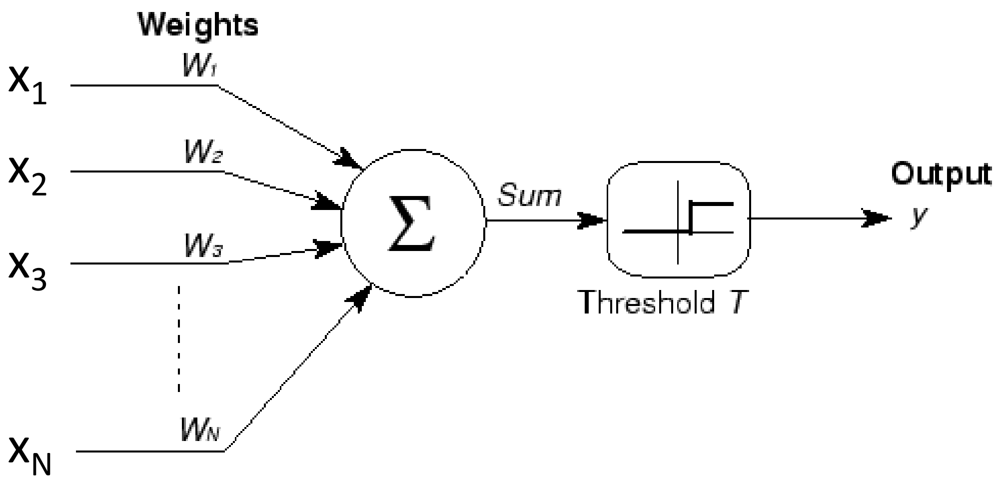
Story so far

- **MLPs are connectionist computational models**
  – Individual perceptrons are computational equivalent of neurons
  – The MLP is a layered composition of many perceptrons
- **MLPs can model Boolean functions**
  – Individual perceptrons can act as Boolean gates
  – Networks of perceptrons are Boolean functions
- **MLPs are Boolean *machines***
  – They represent Boolean functions over linear boundaries
  – They can represent arbitrary decision boundaries
  – They can be used to *classify* data

So what does the perceptron really model?

- Is there a “semantic” interpretation?

Lets look at the weights

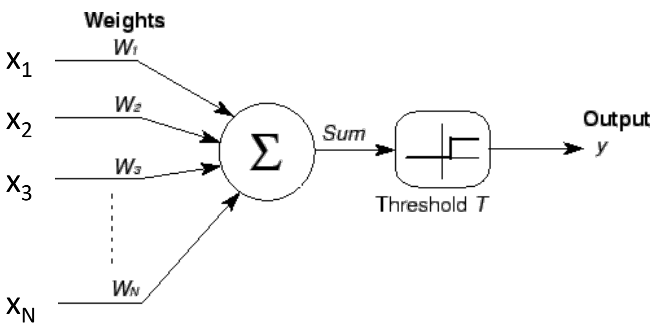


$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

$$y = \begin{cases} 1 \; if \; \mathbf{x}^T\mathbf{w} \geq T \\ 0 \; else \end{cases}$$

- What do the *weights* tell us?
  - The neuron fires if the inner product between the weights and the inputs exceeds a threshold

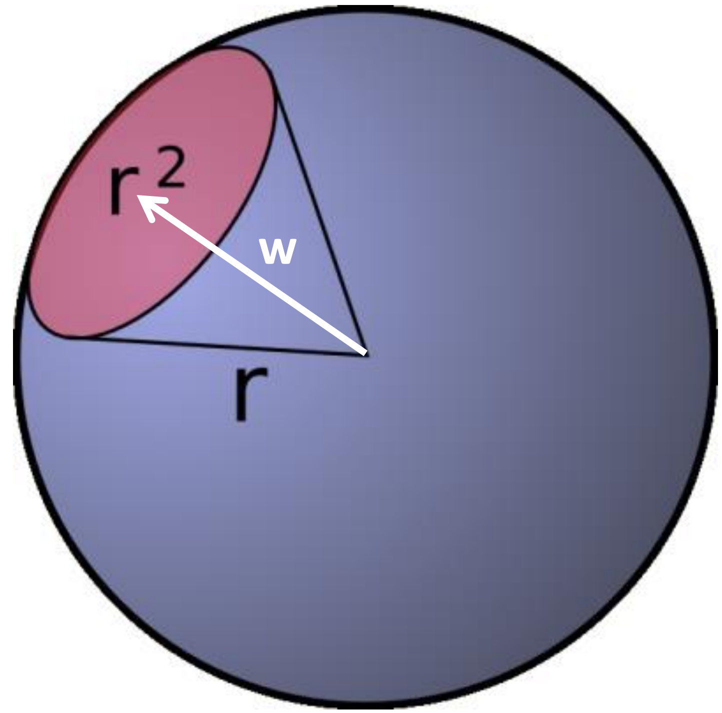
The weight as a “template”



$$X^T W > T$$

$$\cos\theta > \frac{T}{|X|}$$

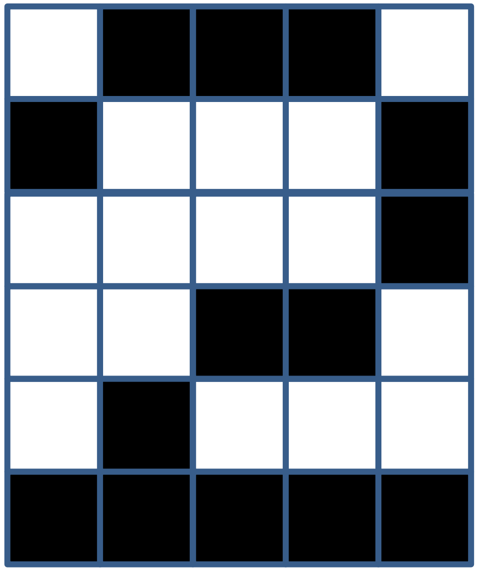
$$\theta < \cos^{-1}\left(\frac{T}{|X|}\right)$$



- The perceptron fires if the input is within a specified angle of the weight
- Neuron fires if the input vector is close enough to the weight vector.
  - If the input pattern matches the weight pattern closely enough

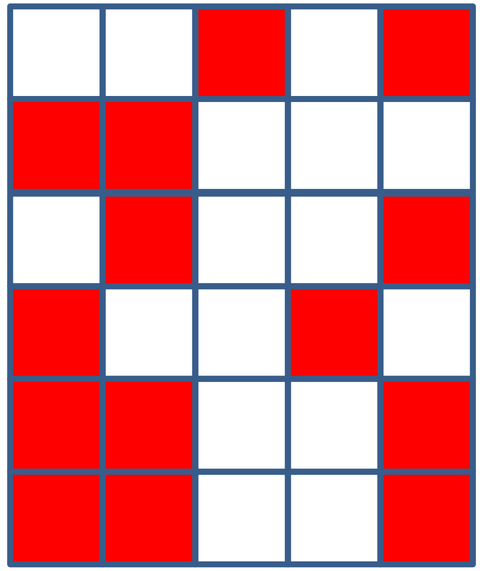
The weight as a template

W



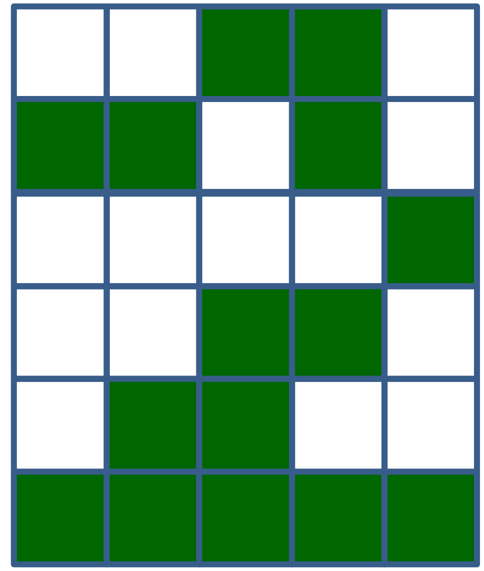
$$y = \begin{cases} 1 \; if \; \sum_i w_i \mathrm{x}_i \geq T \\ 0 \; else \end{cases}$$

X



Correlation = 0.57

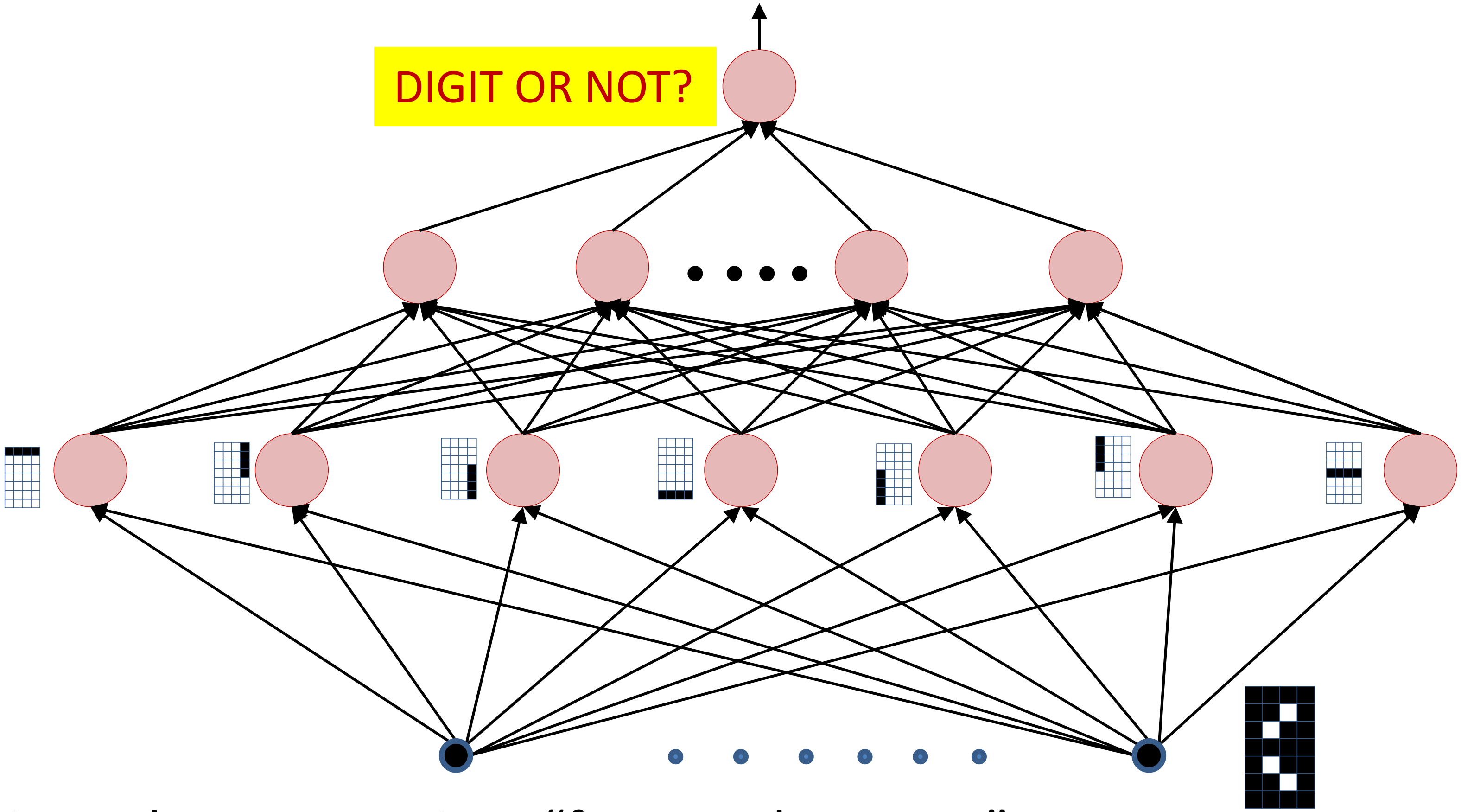
X



Correlation = 0.82

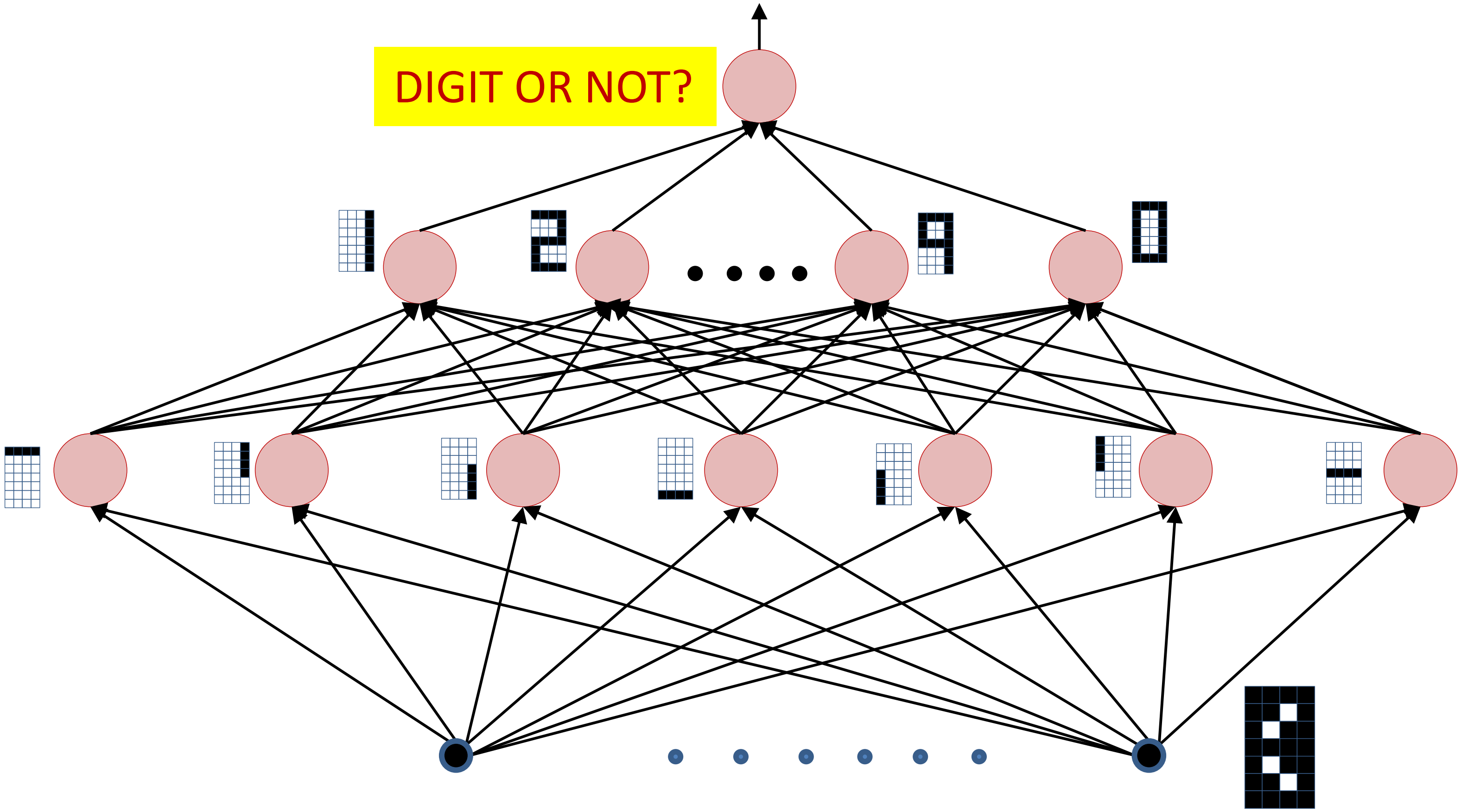
- If the *correlation* between the weight pattern and the inputs exceeds a threshold, fire
- The perceptron is a *correlation filter!*

The MLP as a Boolean function over feature detectors



- The input layer comprises "feature detectors"
  - Detect if certain patterns have occurred in the input
- The network is a Boolean function over the feature detectors
- I.e. it is important for the *first* layer to capture relevant patterns

The MLP as a cascade of feature detectors



- The network is a cascade of feature detectors
  - Higher level neurons compose complex templates from features represented by lower-level neurons

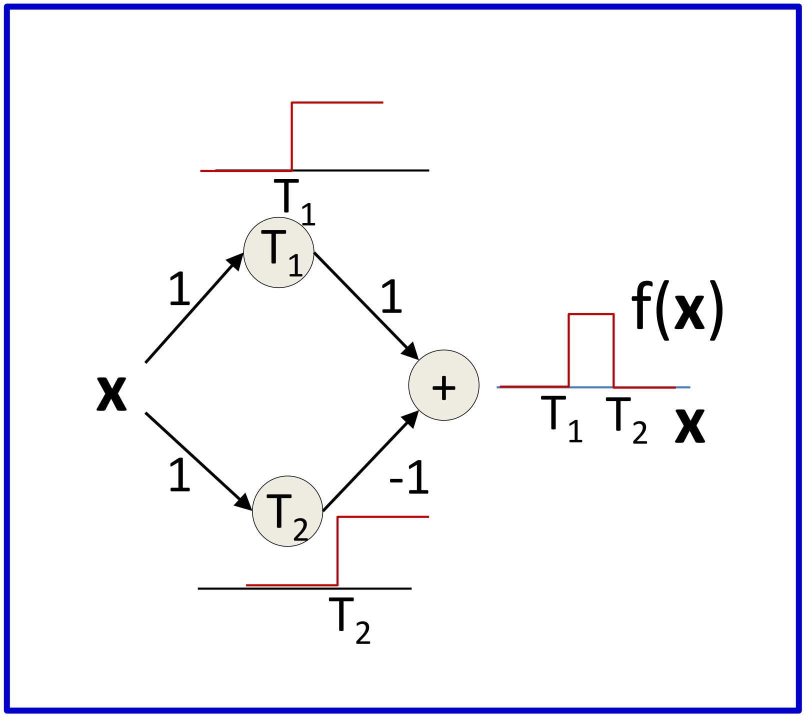
Story so far

- **Multi-layer perceptrons are connectionist computational models**
- **MLPs are Boolean *machines***
  - They can model Boolean functions
  - They can represent arbitrary decision boundaries over real inputs

- **Perceptrons are *correlation filters***
  - They detect patterns in the input
- **MLPs are Boolean formulae over patterns detected by perceptrons**
  - Higher-level perceptrons may also be viewed as feature detectors

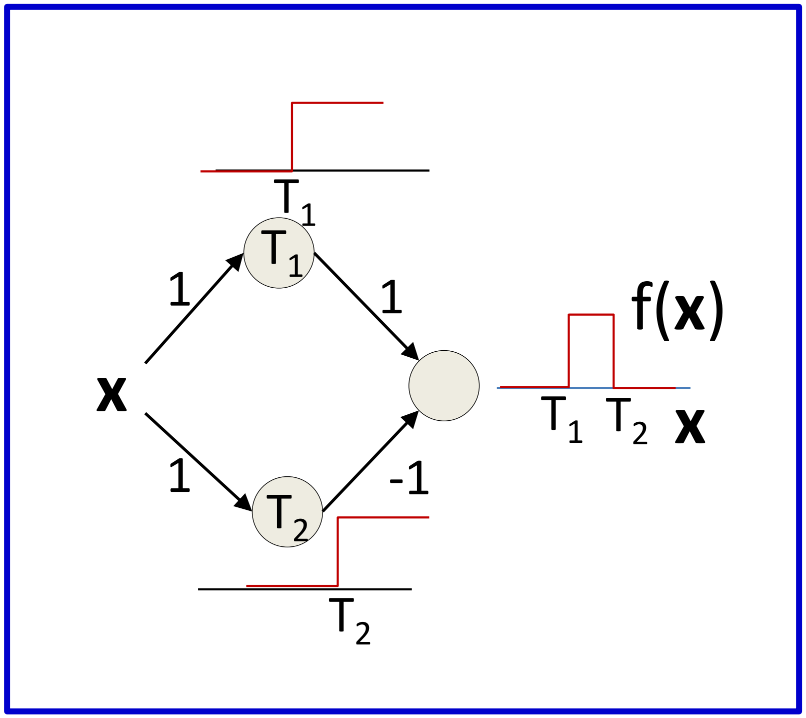
- **Extra: MLP in classification**
  - The network will fire if the combination of the detected basic features matches an "acceptable" pattern for a desired class of signal
    - E.g. Appropriate combinations of (Nose, Eyes, Eyebrows, Cheek, Chin) → Face

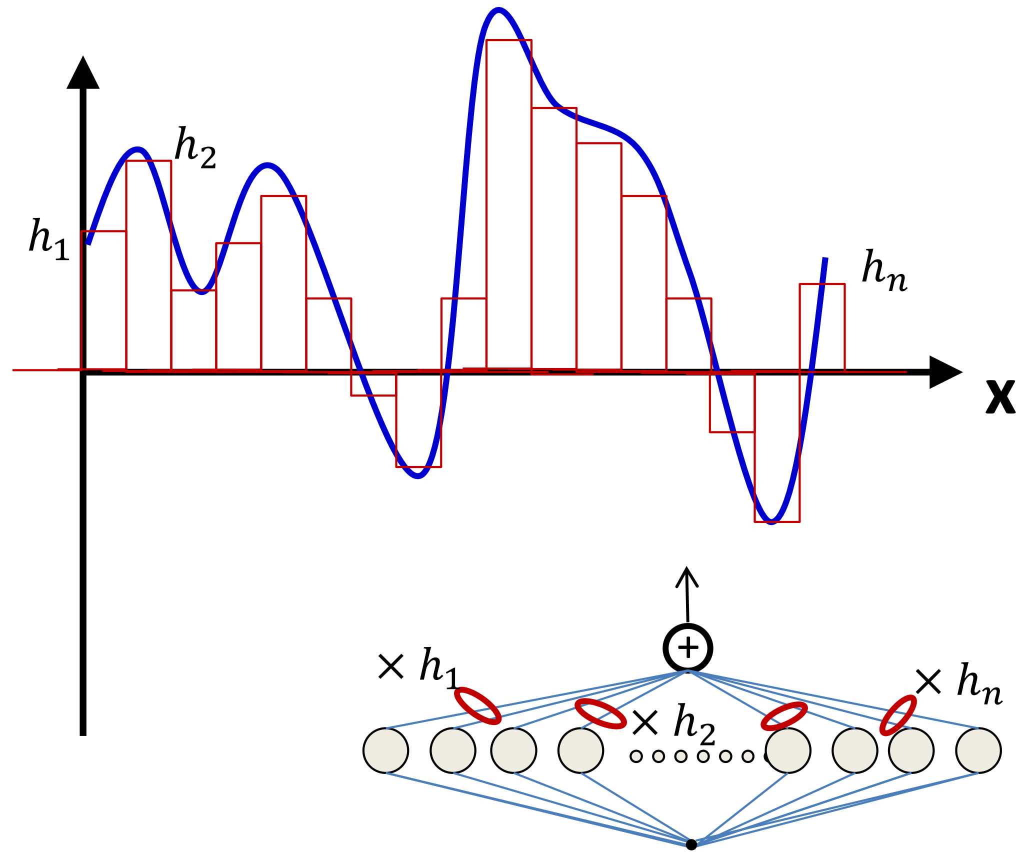
MLP as a continuous-valued regression



- 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
- This generalizes to functions of any number of inputs (next part)

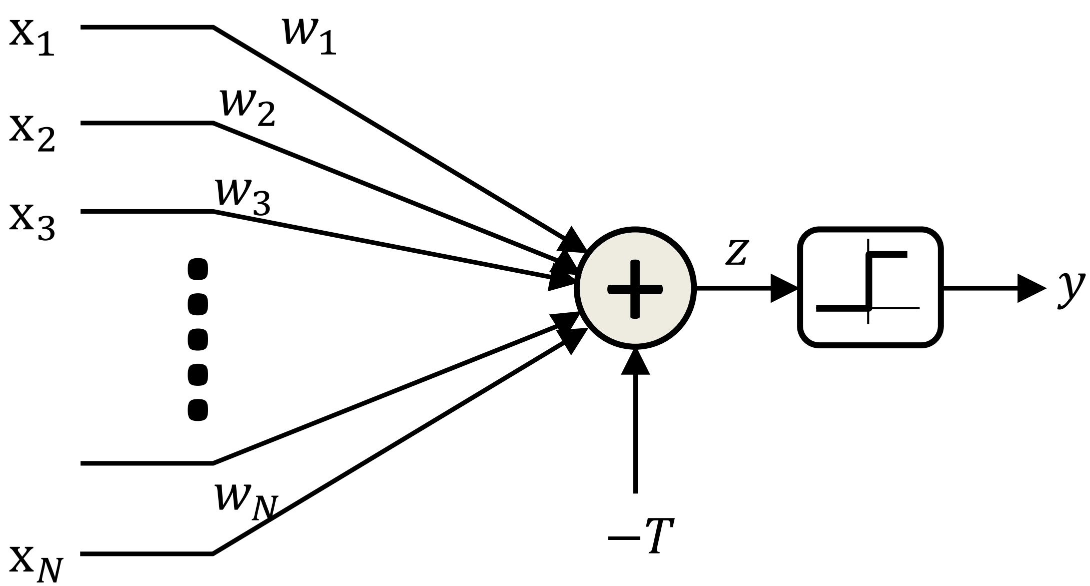
Story so far

- **Multi-layer perceptrons are connectionist computational models**
- **MLPs are *classification engines***
  – They can identify classes in the data
  – Individual perceptrons are feature detectors
  – The network will fire if the combination of the detected basic features matches an "acceptable" pattern for a desired class of signal

- **MLP can also model continuous valued functions**

Neural Networks:
Part 2: What can a network represent

Recap: The perceptron

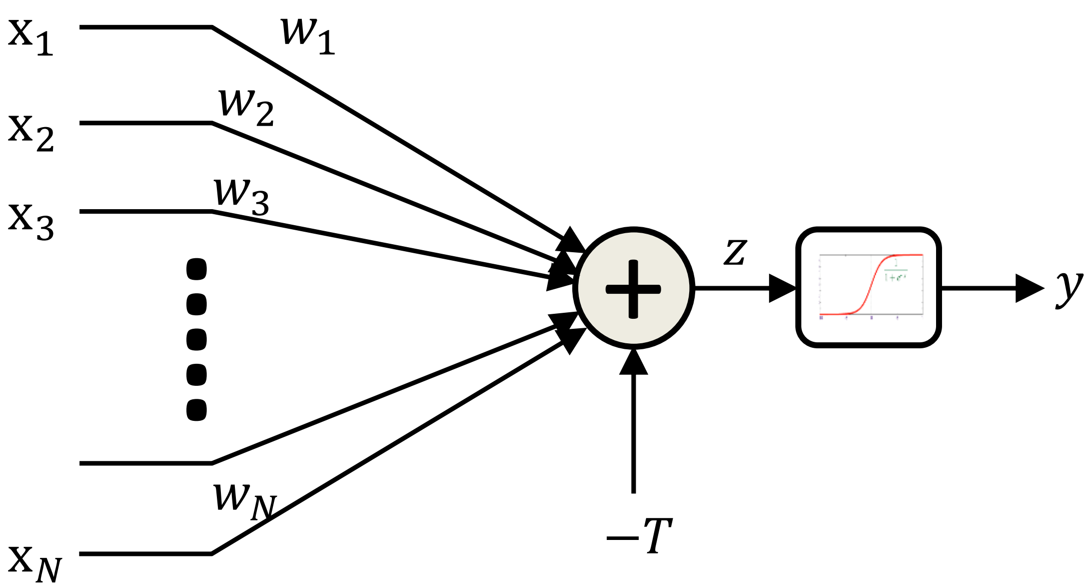


$$z = \sum_i w_i x_i - T$$

$$y = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; else \end{cases}$$

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

The "soft" perceptron

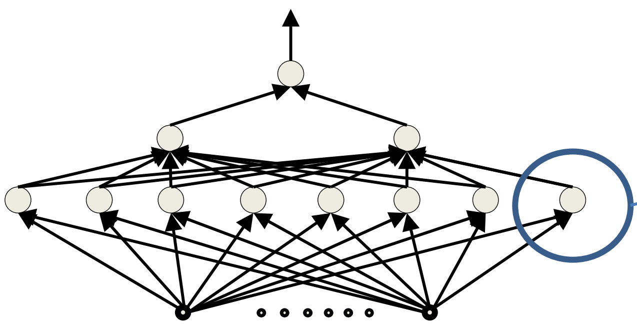


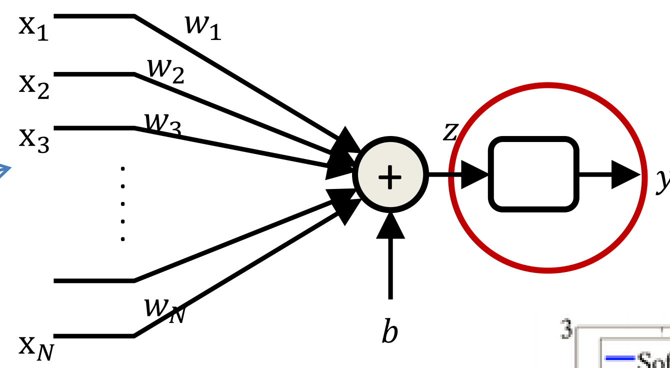
$$z = \sum_i w_i x_i - T$$

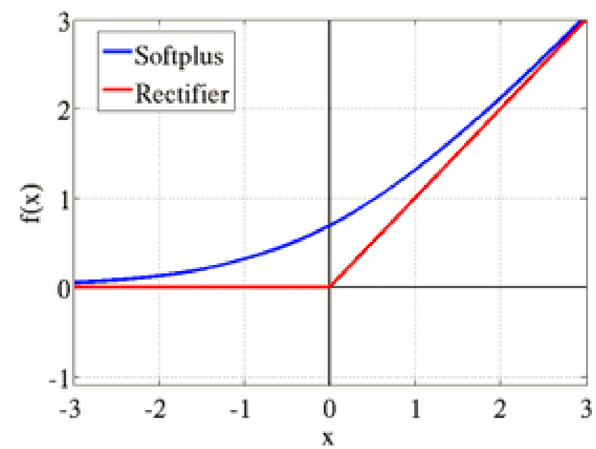
$$y = \frac{1}{1 + exp(-z)}$$

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

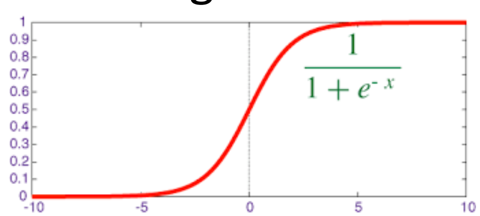
Other "activations"



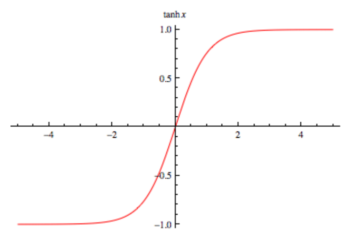




sigmoid

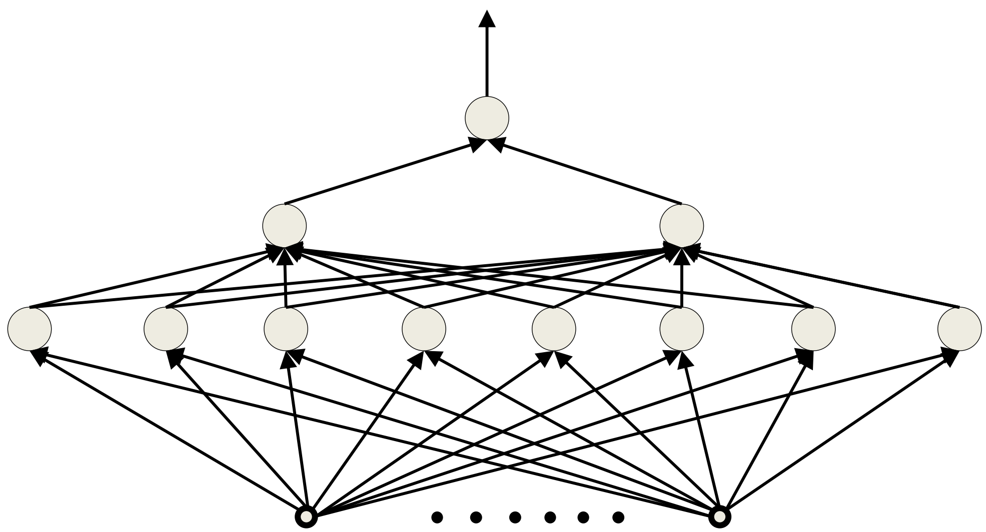


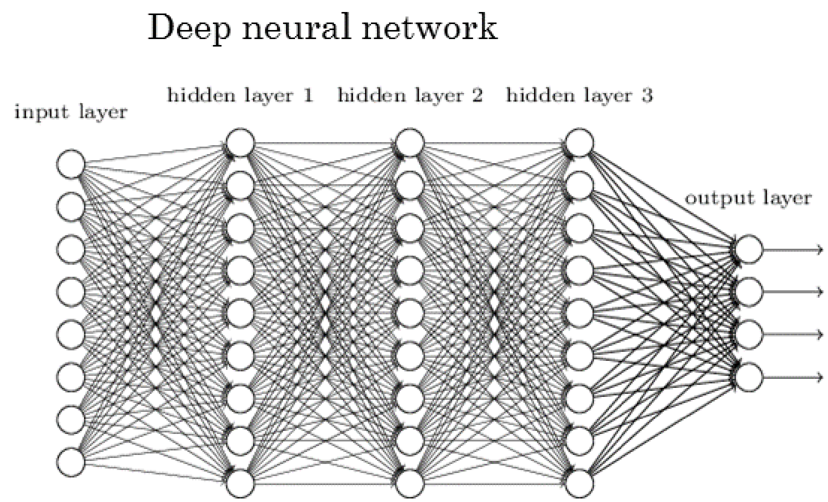
tanh



- Does not always have to be a squashing function
- We will continue to assume a "threshold" activation in this lecture

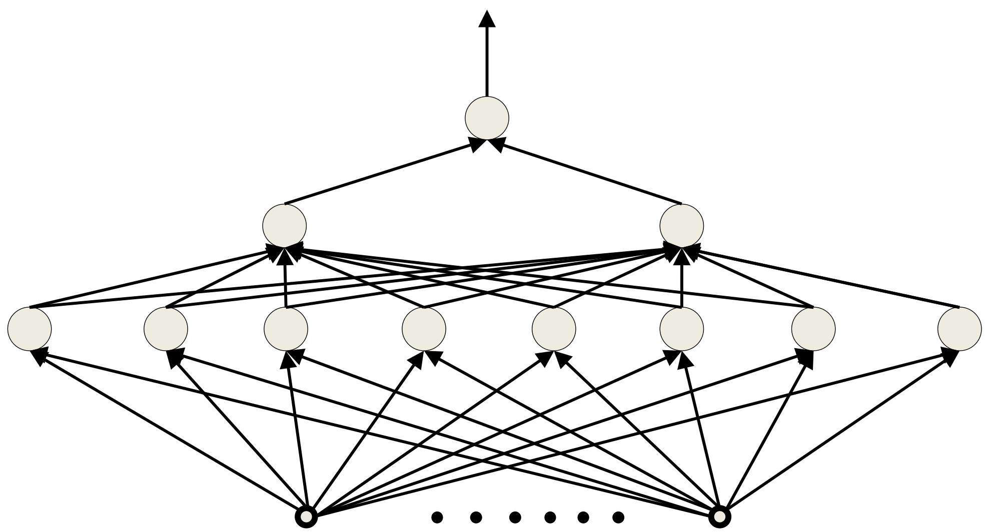
Recap: the multi-layer perceptron

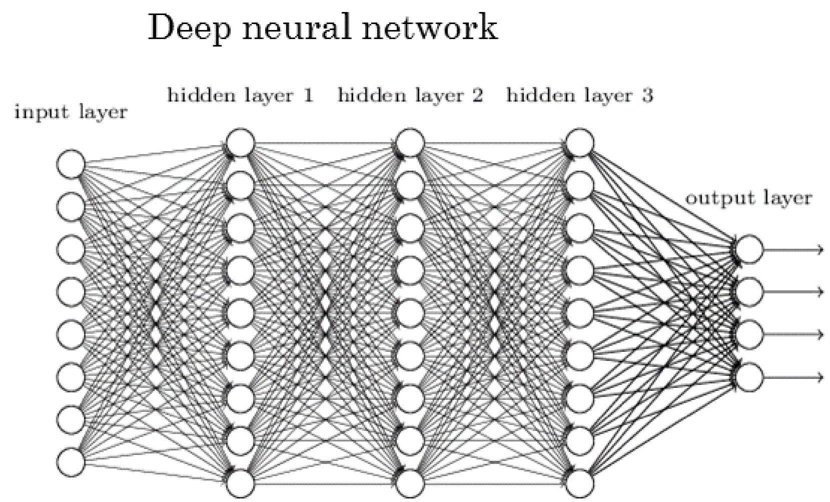




- A network of perceptrons
  - Generally "layered"

Aside: Note on “depth”

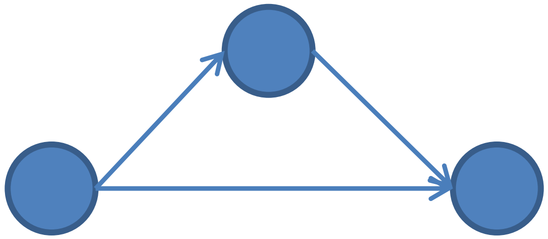


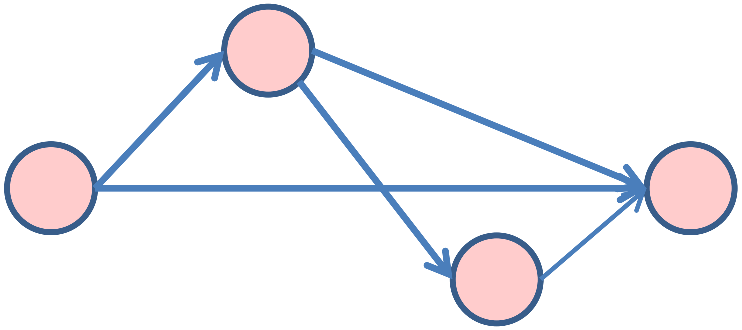


- What is a “deep” network

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

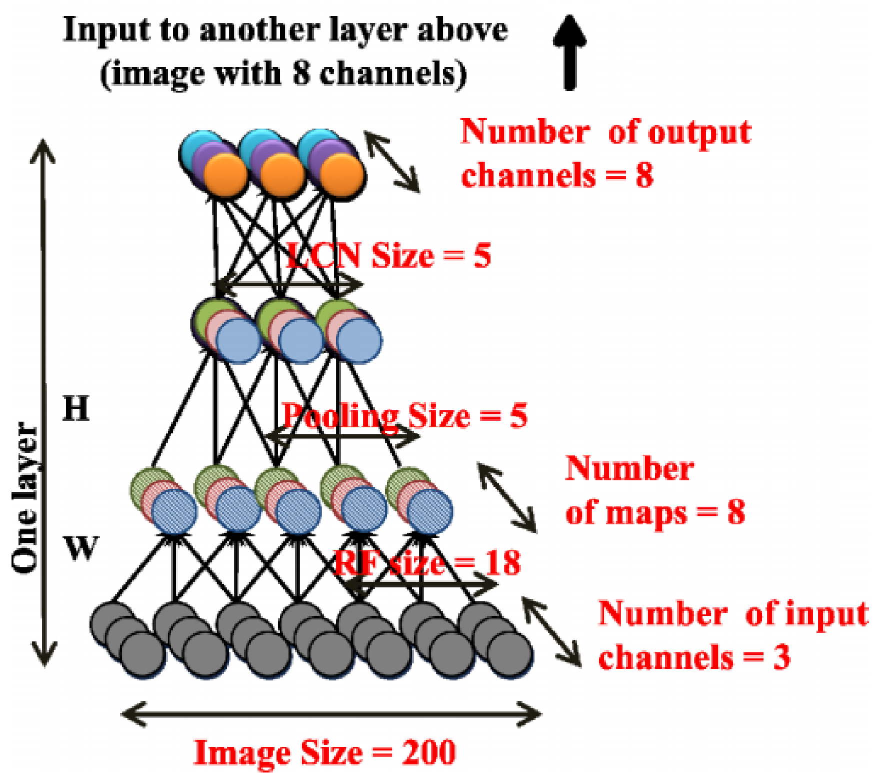




- Left: Depth = 2.   Right: Depth = 3

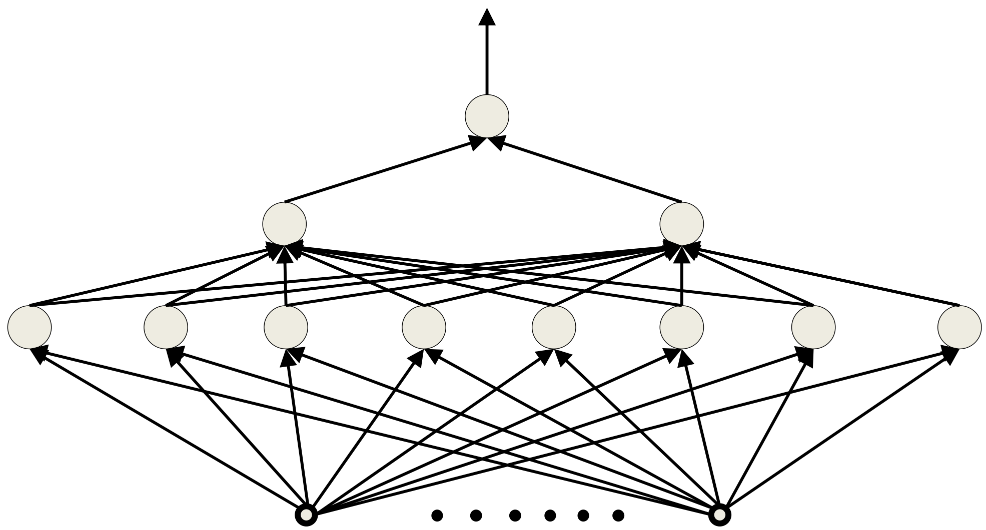
Deep Structures

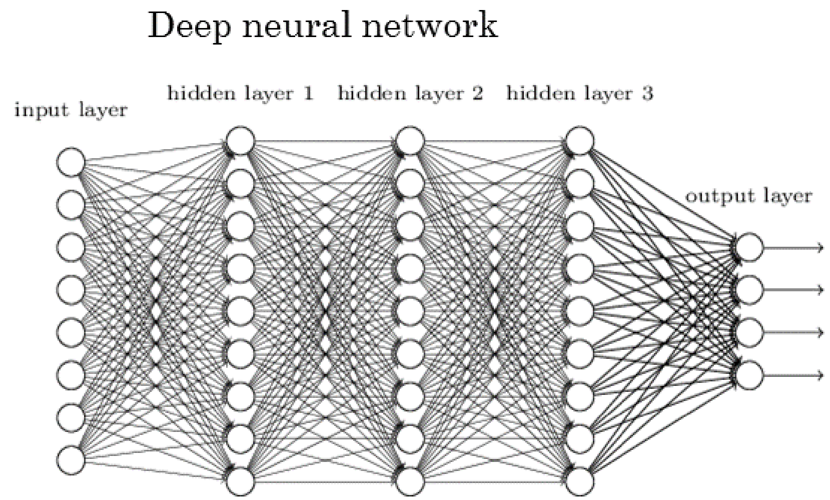
- *Layered* deep structure



- "Deep" → Depth > 2

The multi-layer perceptron

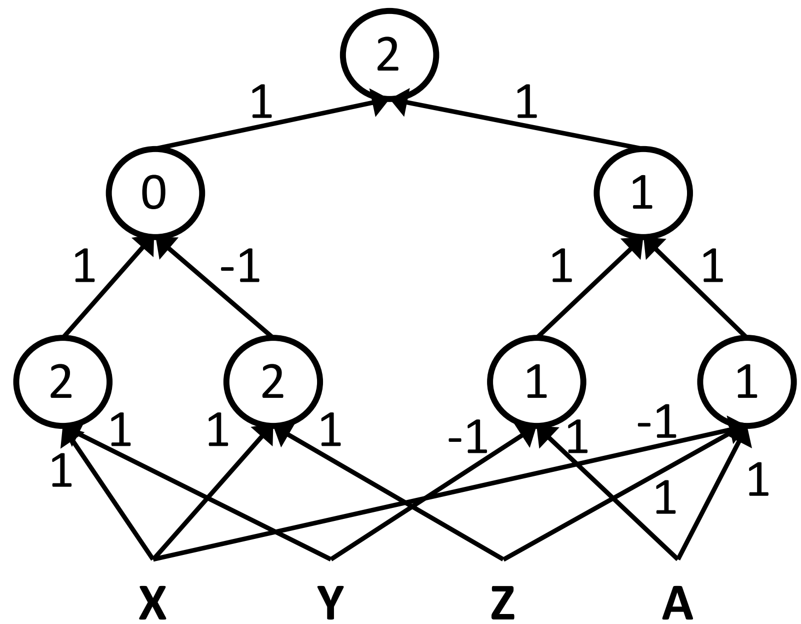


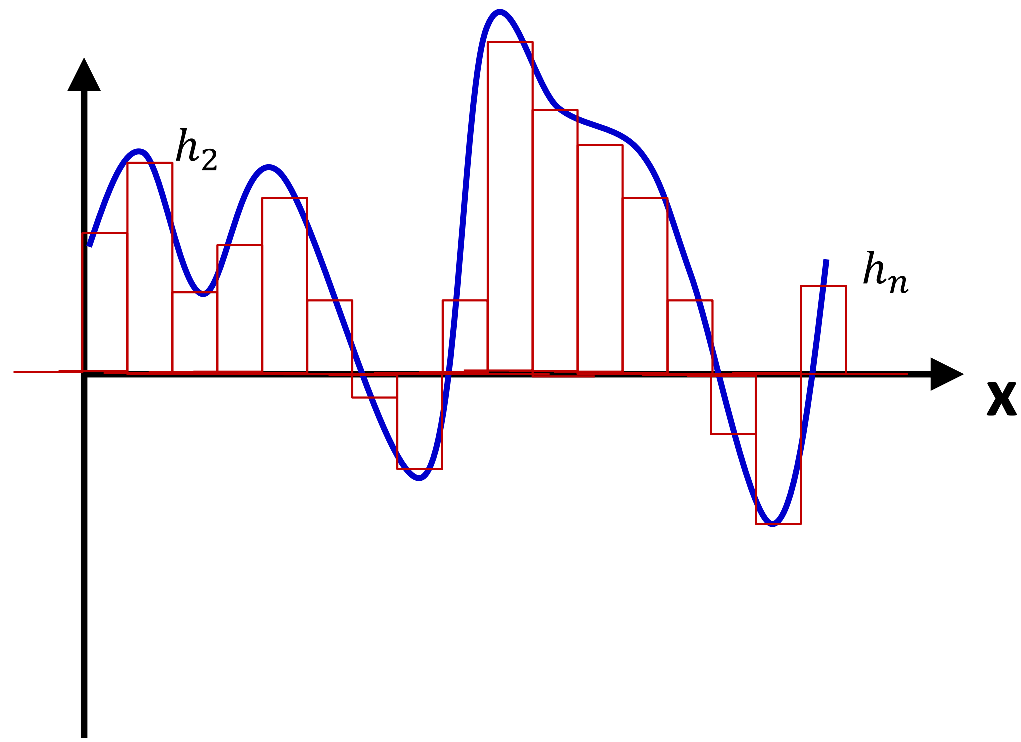


- Inputs are real or Boolean stimuli
- Outputs are real or Boolean values
  - Can have multiple outputs for a single input
- **What can this network compute?**
  - **What kinds of input/output relationships can it model?**

MLPs approximate functions

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



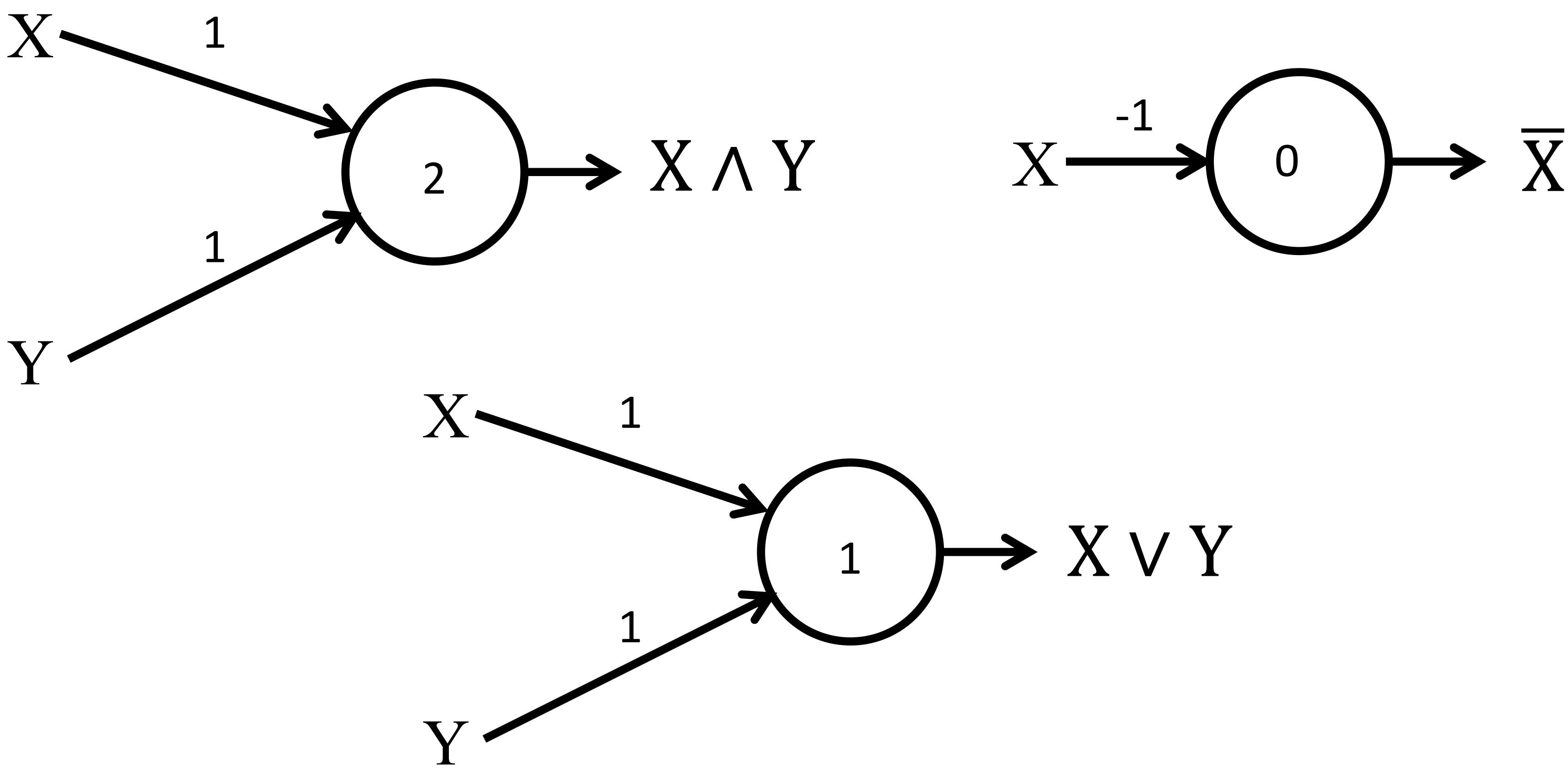


- MLPs can compose Boolean functions
- MLPs can compose real-valued functions
- What are the limitations?

The MLP as a Boolean function

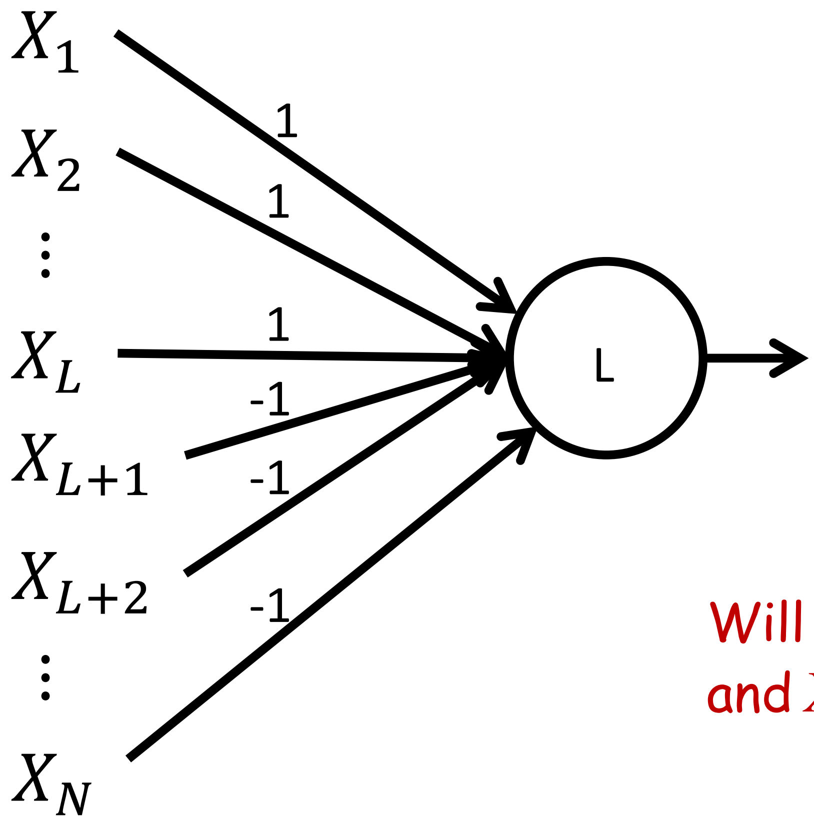
- How well do MLPs model Boolean functions?

The perceptron as a Boolean gate



- A perceptron can model any simple binary Boolean gate

Perceptron as a Boolean gate

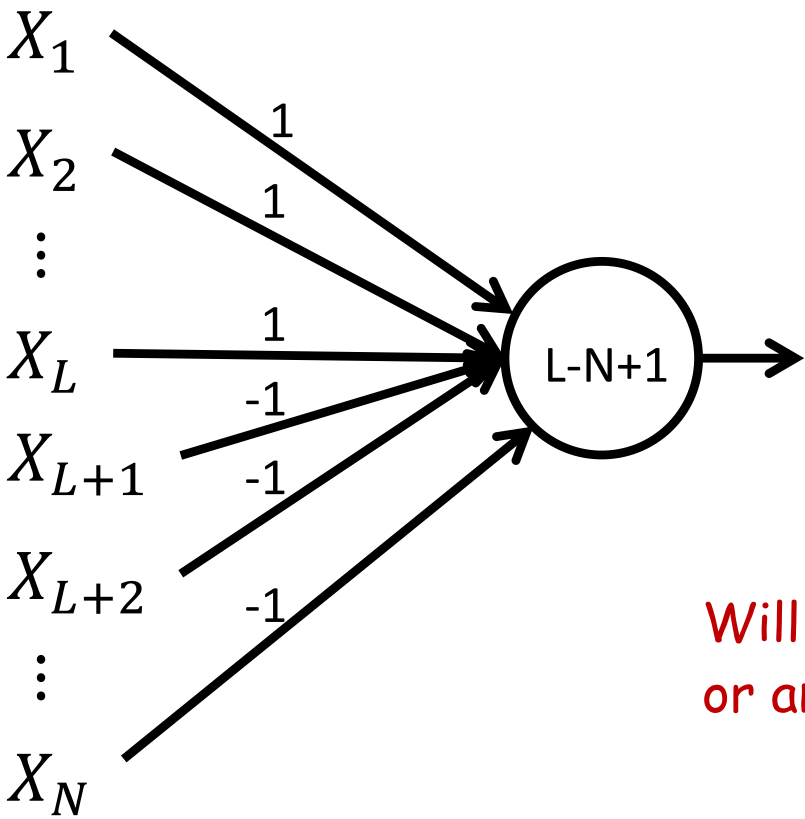


$$\left(\bigwedge_{i=1}^{L} X_i\right) \wedge \left(\bigwedge_{i=L+1}^{N} \bar{X}_i\right)$$

Will fire only if X_1 .. X_L are all 1 and X_{L+1} .. X_N are all 0

- The universal AND gate
  - AND any number of inputs
    - Any subset of who may be negated

Perceptron as a Boolean gate

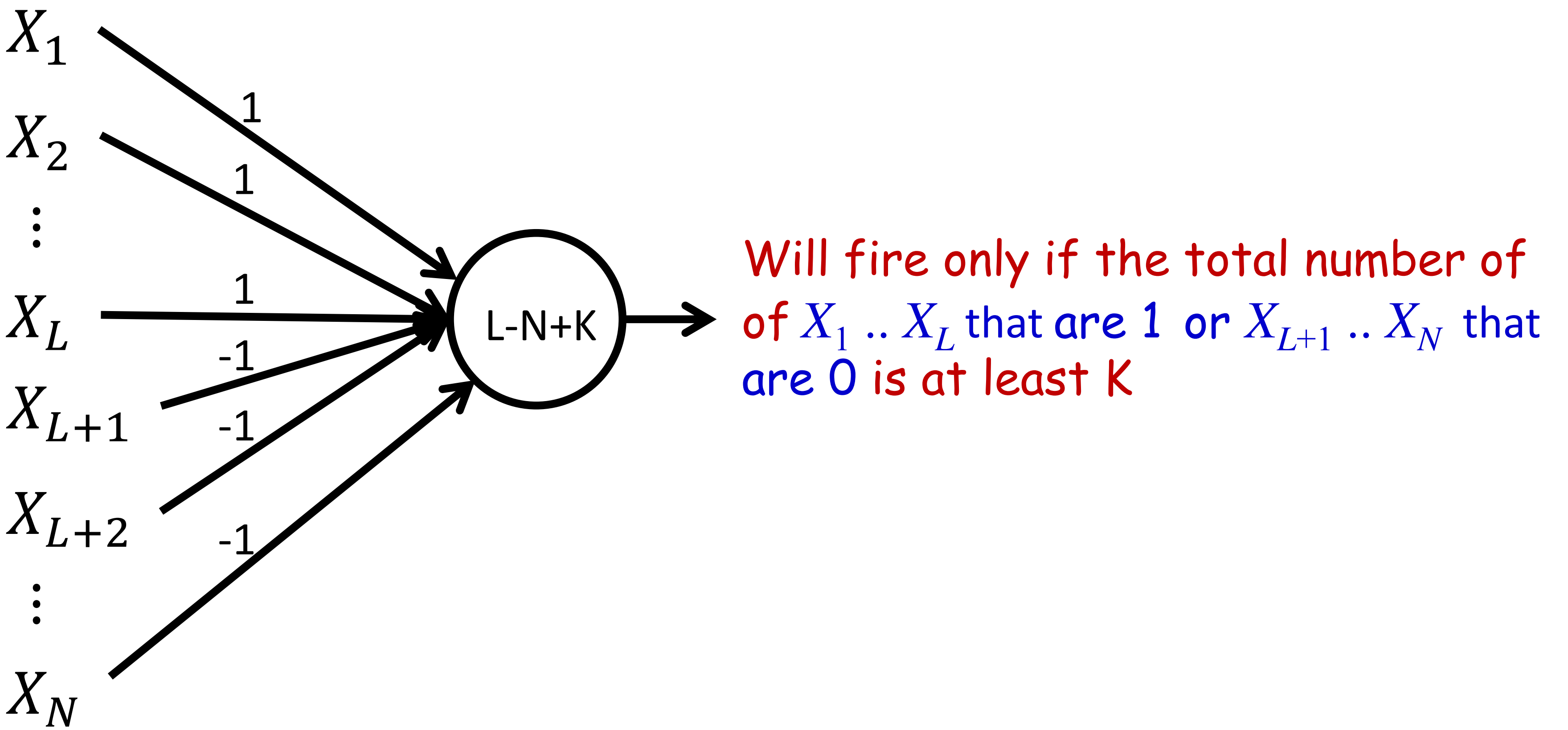


$$\left(\bigvee_{i=1}^{L} X_i\right) \vee \left(\bigvee_{i=L+1}^{N} \bar{X}_i\right)$$

Will fire only if any of X_1 .. X_L are 1 or any of X_{L+1} .. X_N are 0

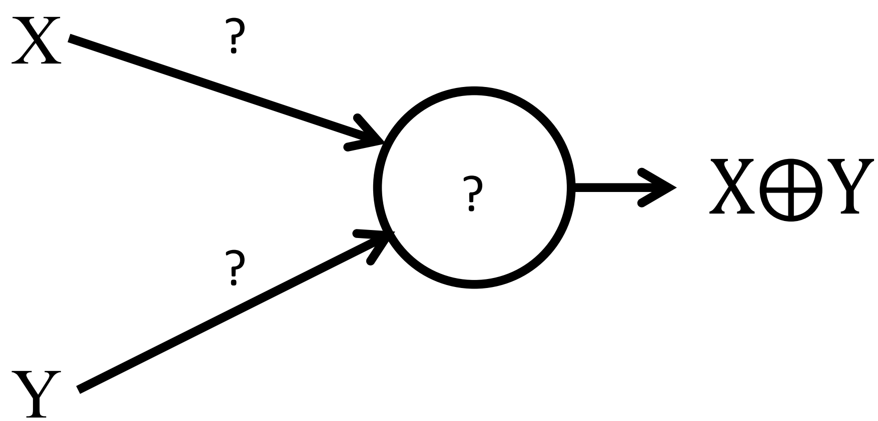
- The universal OR gate
  - OR any number of inputs
    - Any subset of who may be negated

Perceptron as a Boolean Gate



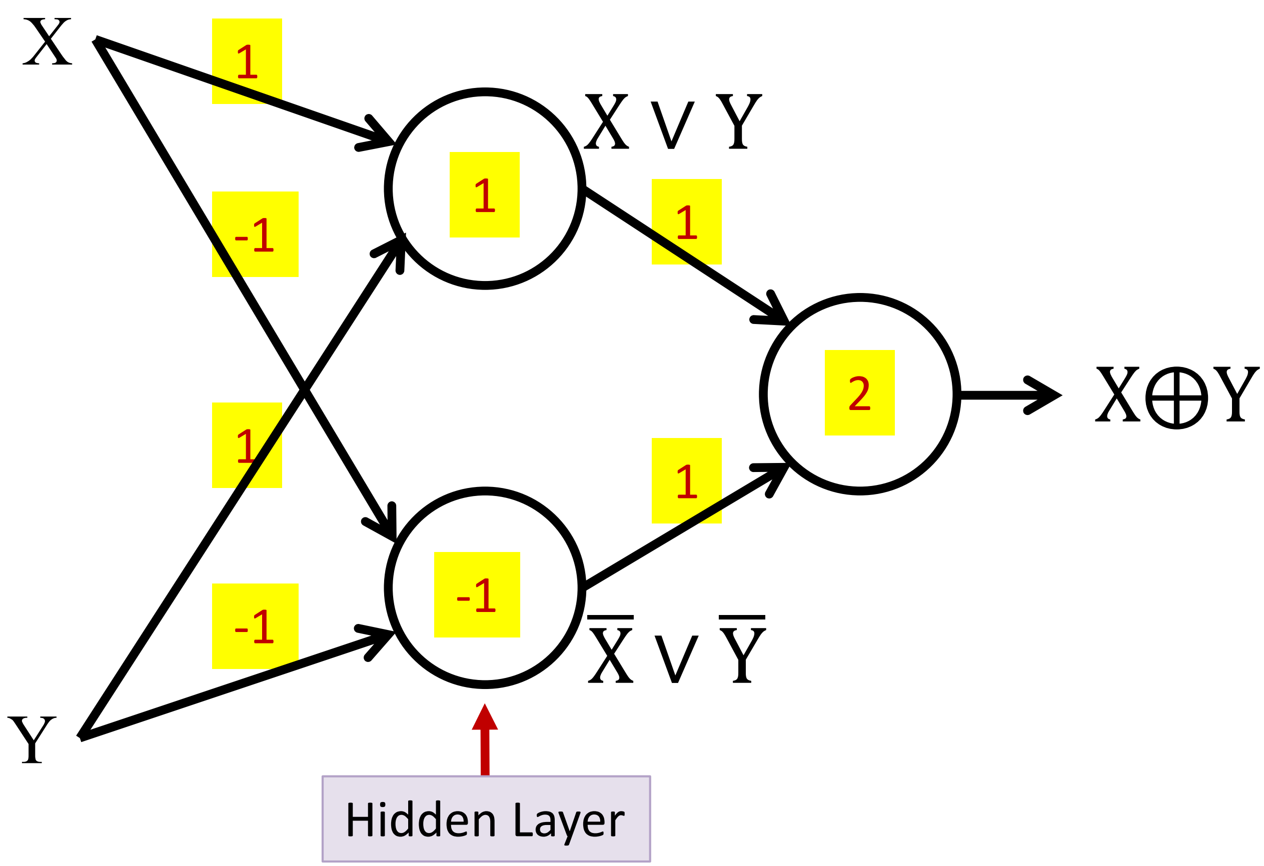
- Universal OR:
  - Fire if any K-*subset* of inputs is "ON"

The perceptron is not enough



- Cannot compute an XOR

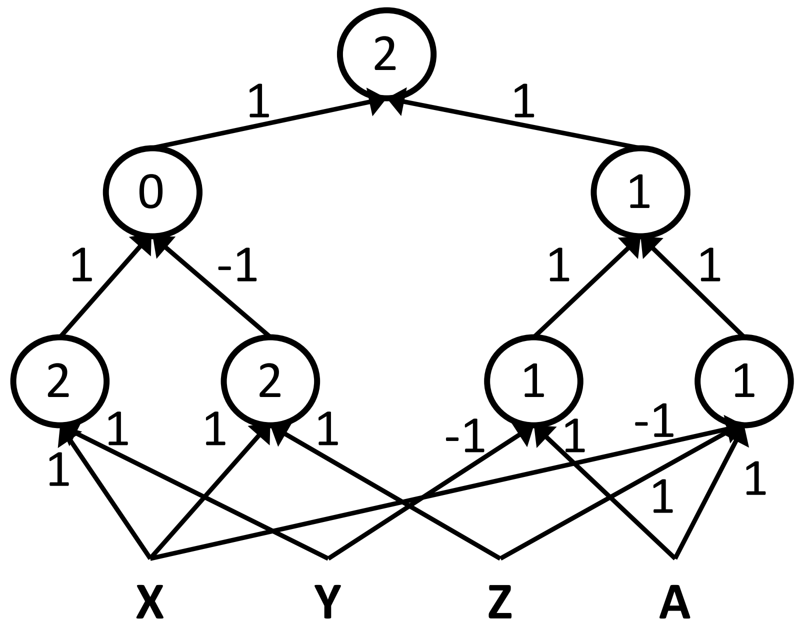
Multi-layer perceptron



- MLPs can compute the XOR

Multi-layer perceptron

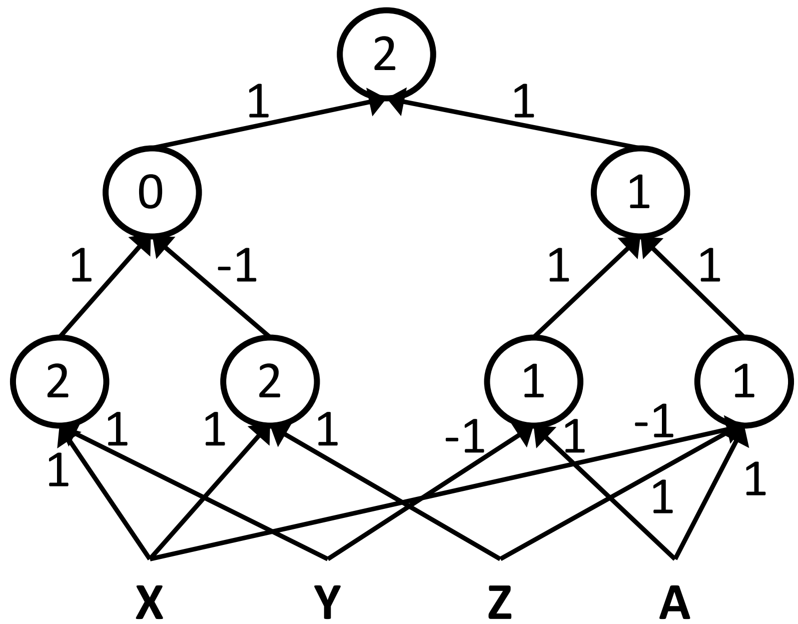
$$((A \& \bar{X} \& Z) | (A \& \bar{Y})) \& ((X \,\&\, Y) | \overline{(X \& Z)})$$

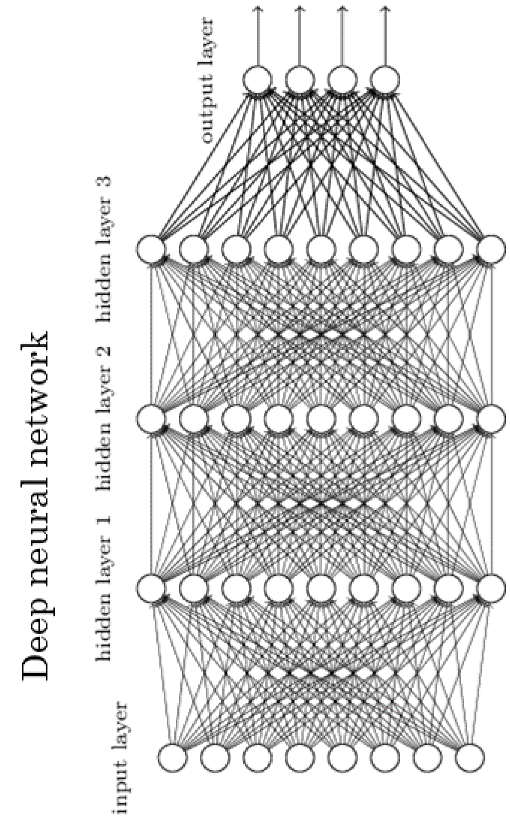


- MLPs can compute more complex Boolean functions
- MLPs can compute *any* Boolean function
  - Since they can emulate individual gates
- **MLPs are *universal Boolean functions***

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

- 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

Truth table shows *all* input combinations for which output is 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

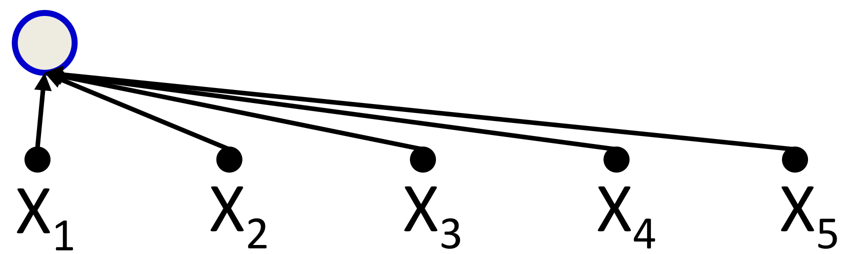
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

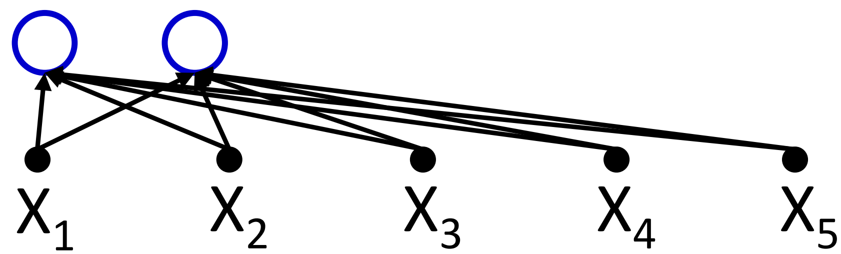
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

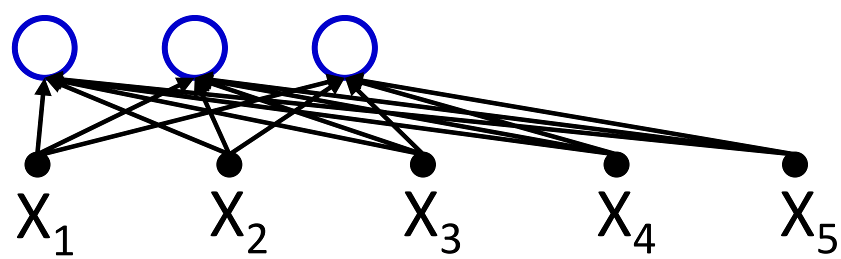
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

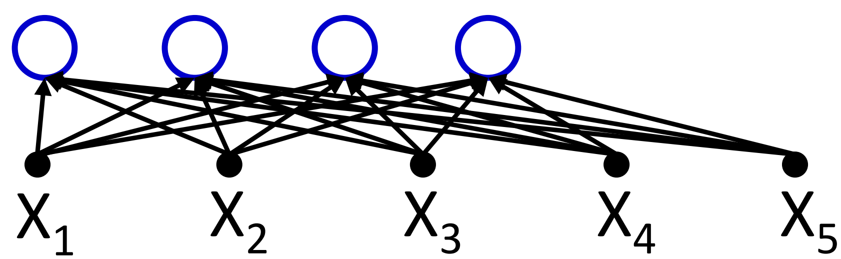
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

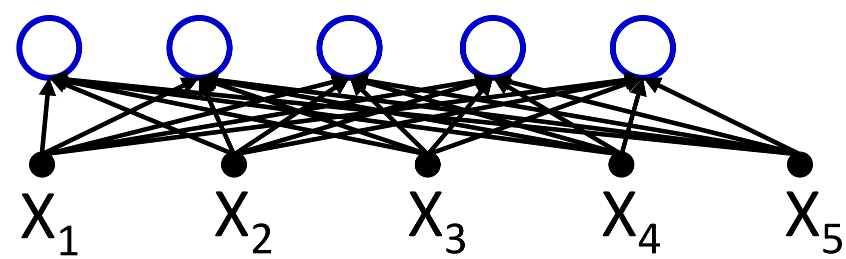
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

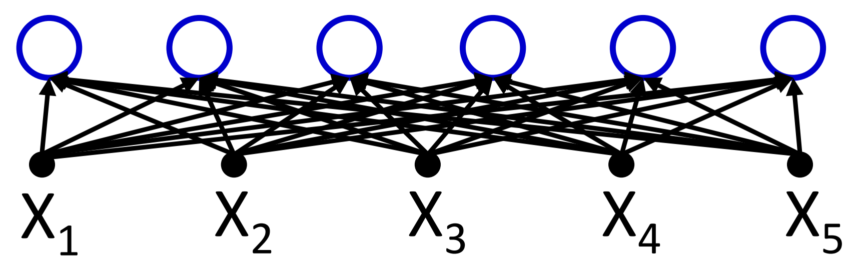
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

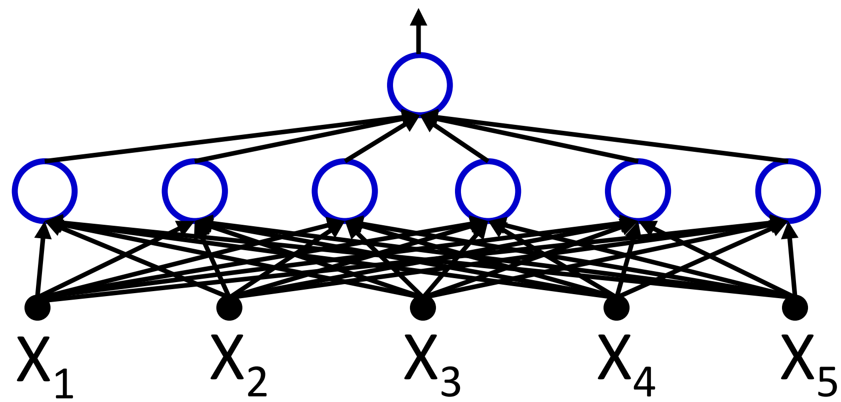
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 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

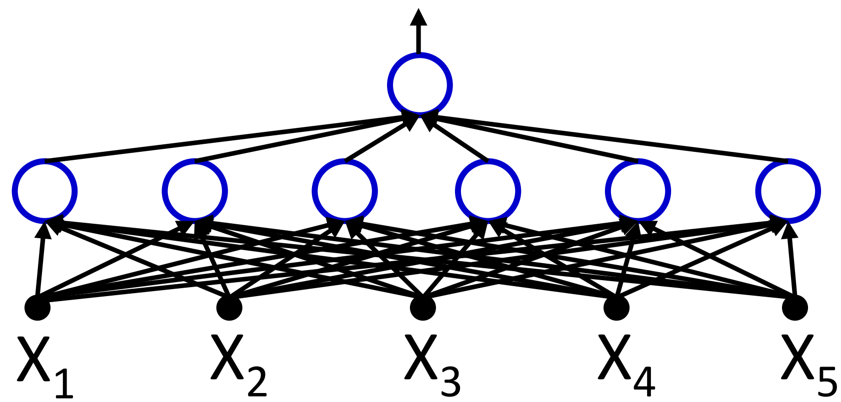
How many layers for a Boolean MLP?

Truth Table

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

Truth table shows *all* input combinations for which output is 1

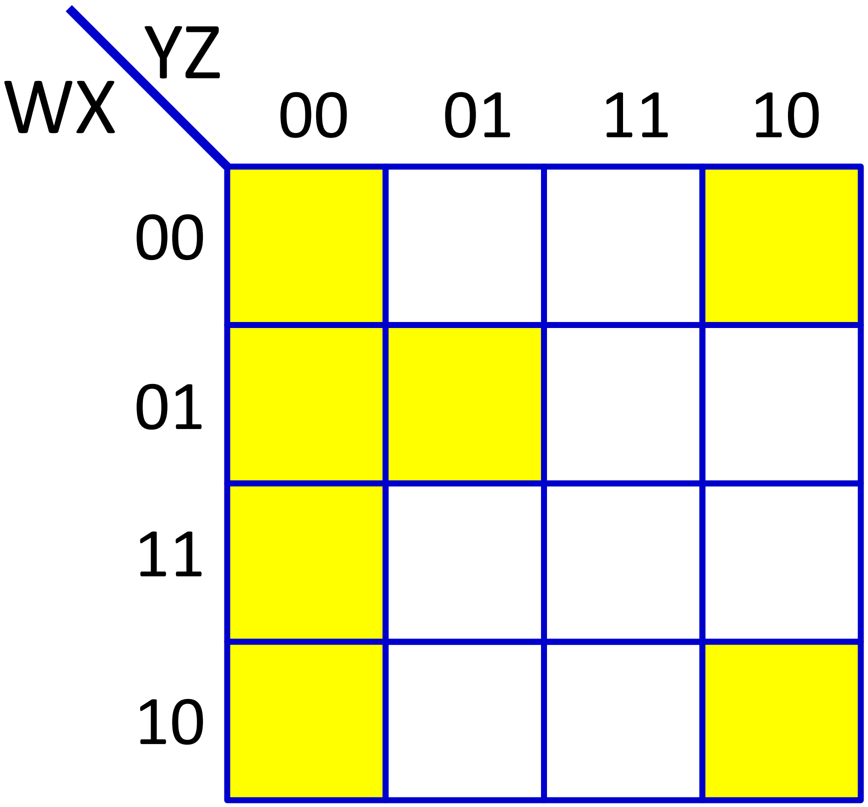
$$Y = \bar{X}_1\bar{X}_2X_3X_4\bar{X}_5 + \bar{X}_1X_2\bar{X}_3X_4X_5 + \bar{X}_1X_2X_3\bar{X}_4\bar{X}_5 + X_1\bar{X}_2\bar{X}_3\bar{X}_4X_5 + X_1\bar{X}_2X_3X_4X_5 + X_1X_2\bar{X}_3\bar{X}_4X_5$$



- *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?

Reducing a Boolean Function



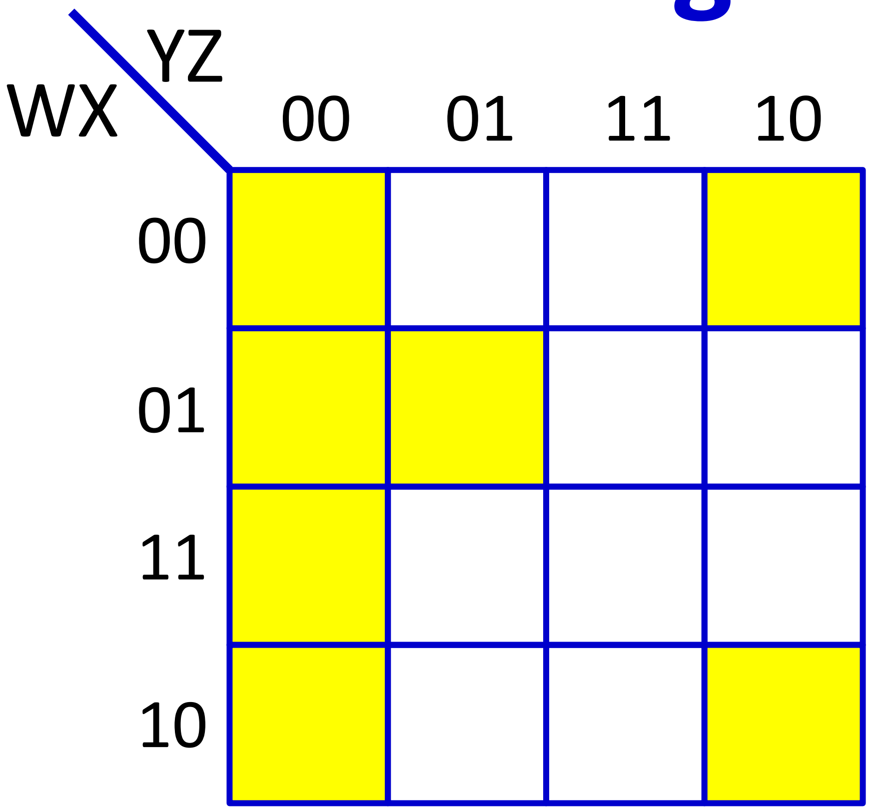
This is a "Karnaugh Map"

It represents a truth table as a grid
Filled boxes represent input combinations for which output is 1; blank boxes have output 0

Adjacent boxes can be "grouped" to reduce the complexity of the DNF formula for the table

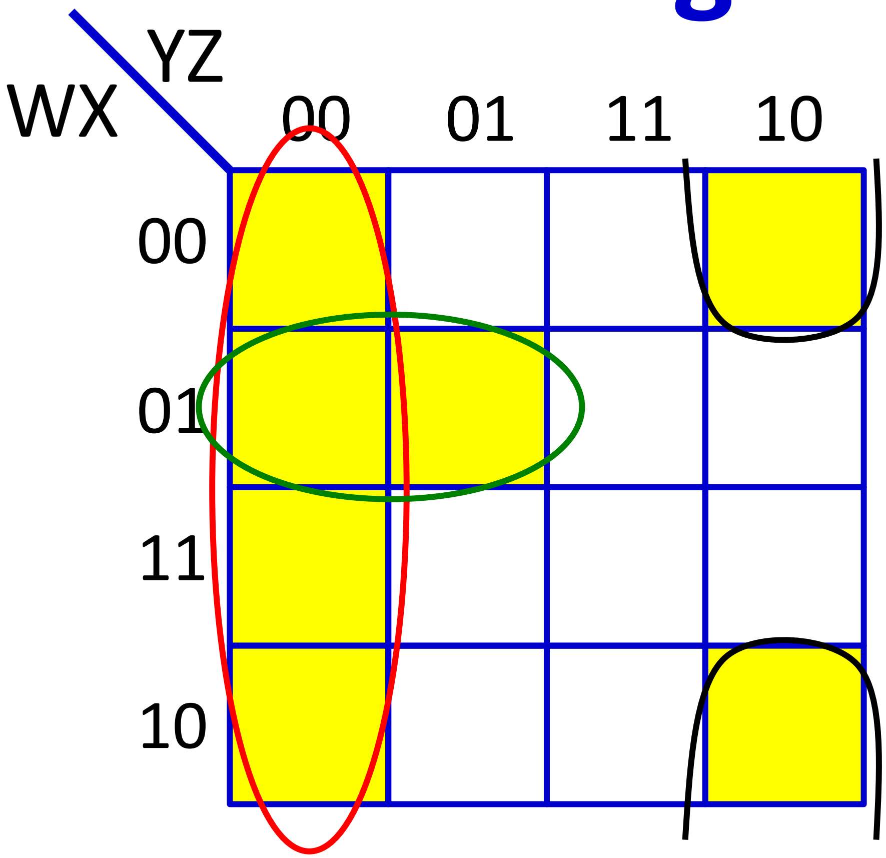
- DNF form:
  - Find groups
  - Express as reduced DNF

Reducing a Boolean Function



Basic DNF formula will require 7 terms

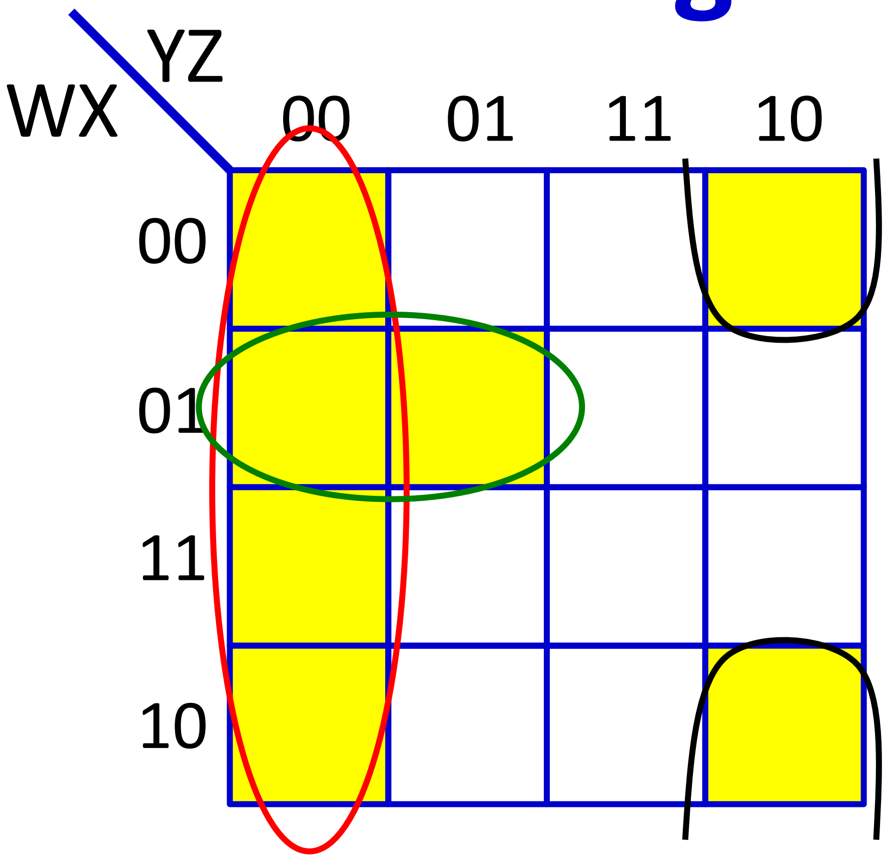
Reducing a Boolean Function



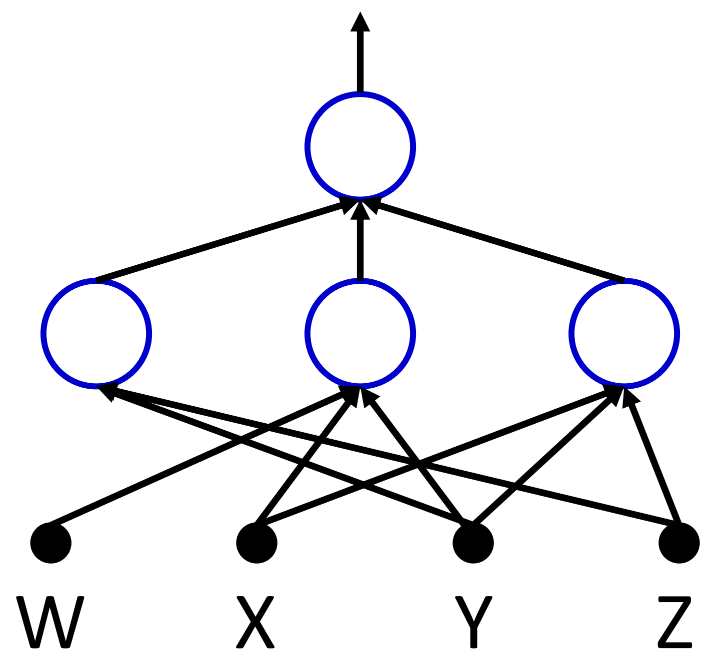
$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$

- *Reduced* DNF form:
  - Find groups
  - Express as reduced DNF

Reducing a Boolean Function



$$O = \bar{Y}\bar{Z} + \bar{W}X\bar{Y} + \bar{X}Y\bar{Z}$$



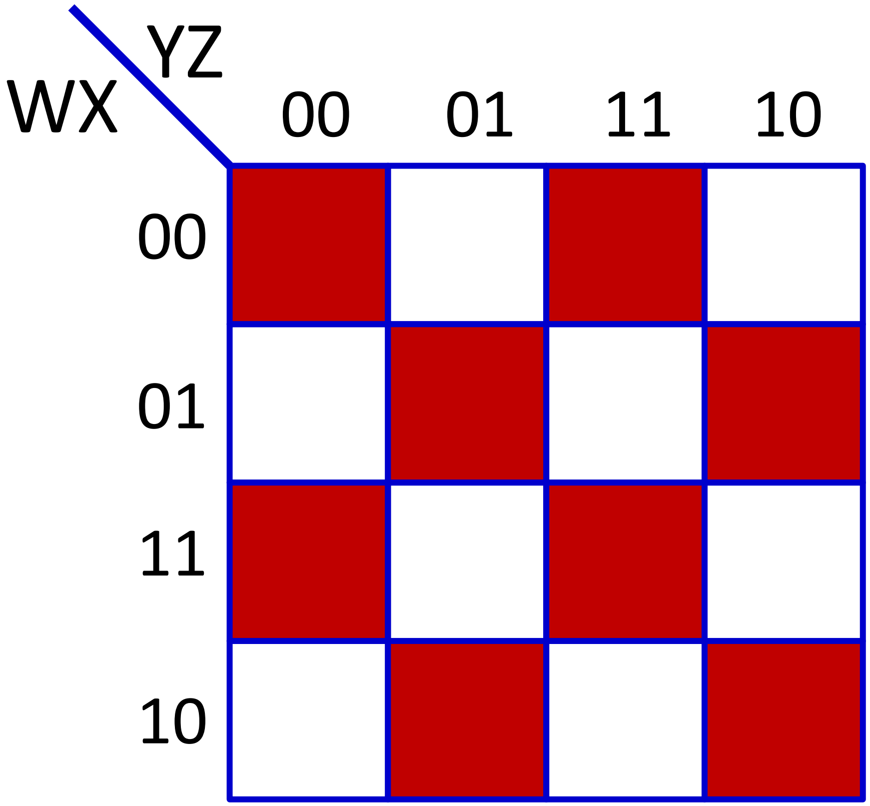
- *Reduced* DNF form:
  - Find groups
  - Express as reduced DNF

Largest irreducible DNF?

WX \ YZ	00	01	11	10
00				
01				
11				
10				

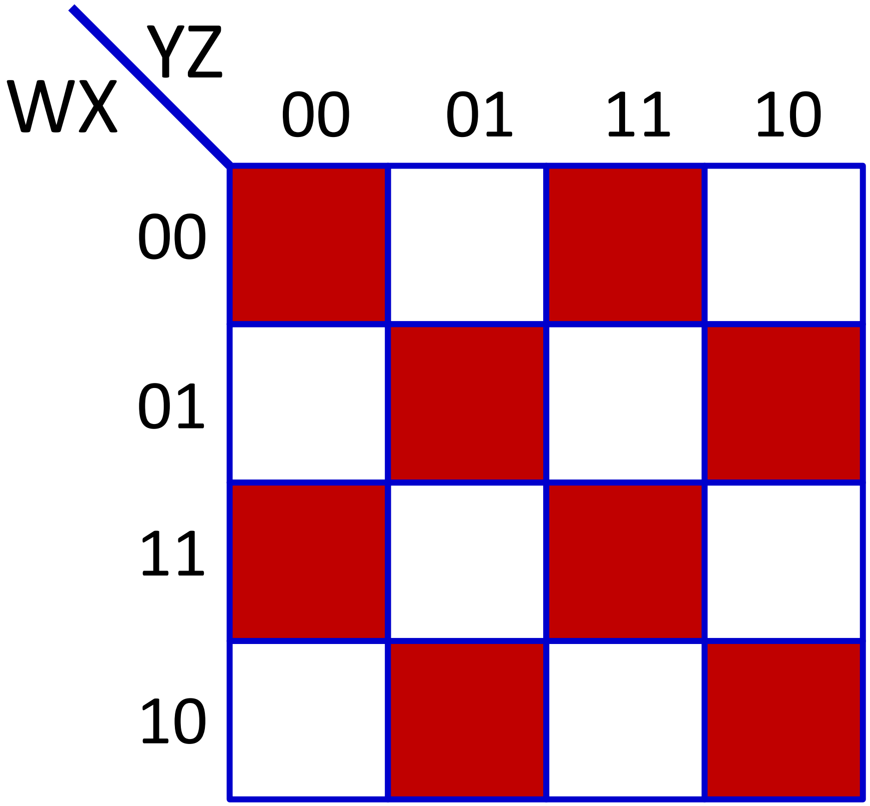
- What arrangement of ones and zeros simply cannot be reduced further?

Largest irreducible DNF?



- What arrangement of ones and zeros simply cannot be reduced further?

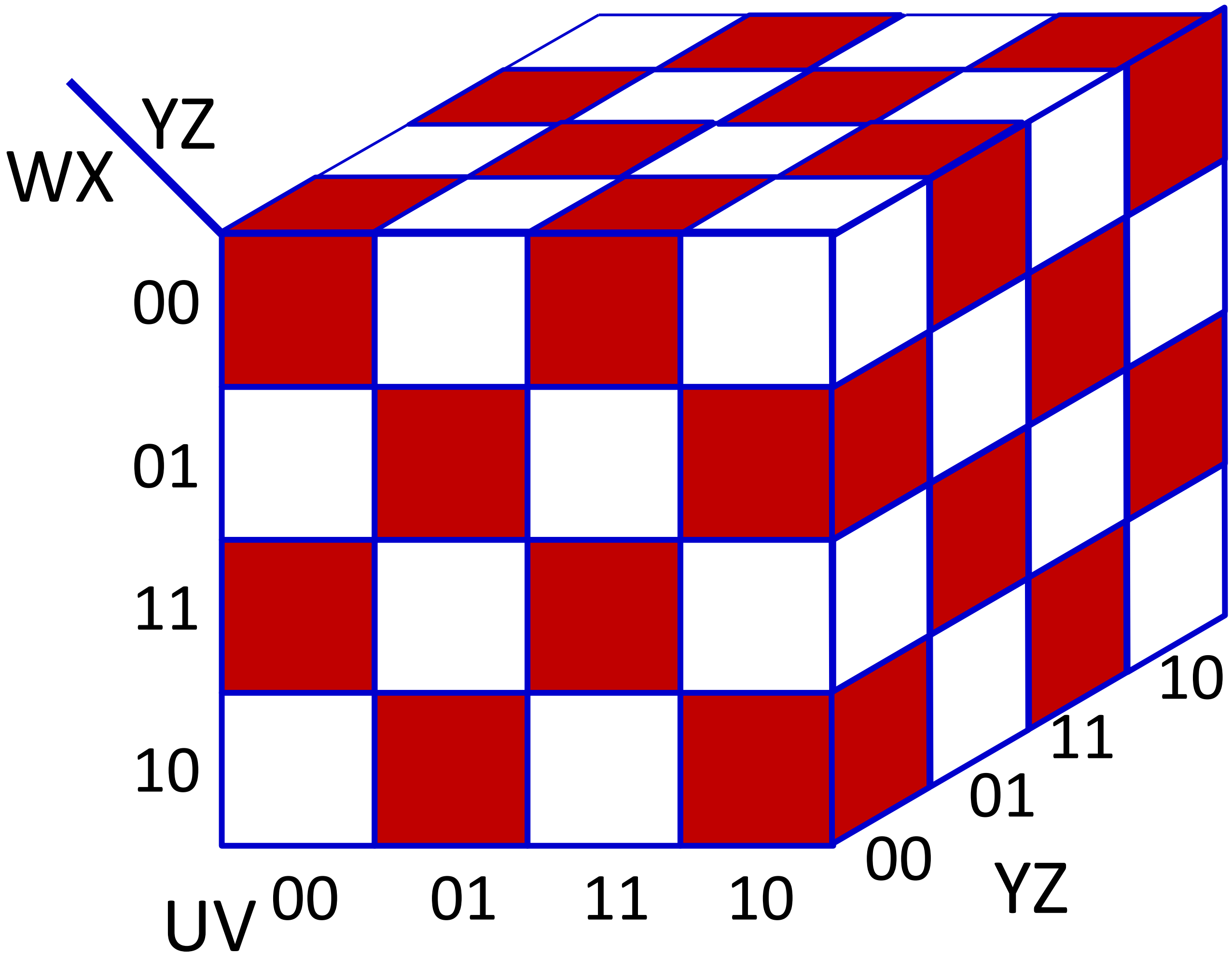
Largest irreducible DNF?



How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function?

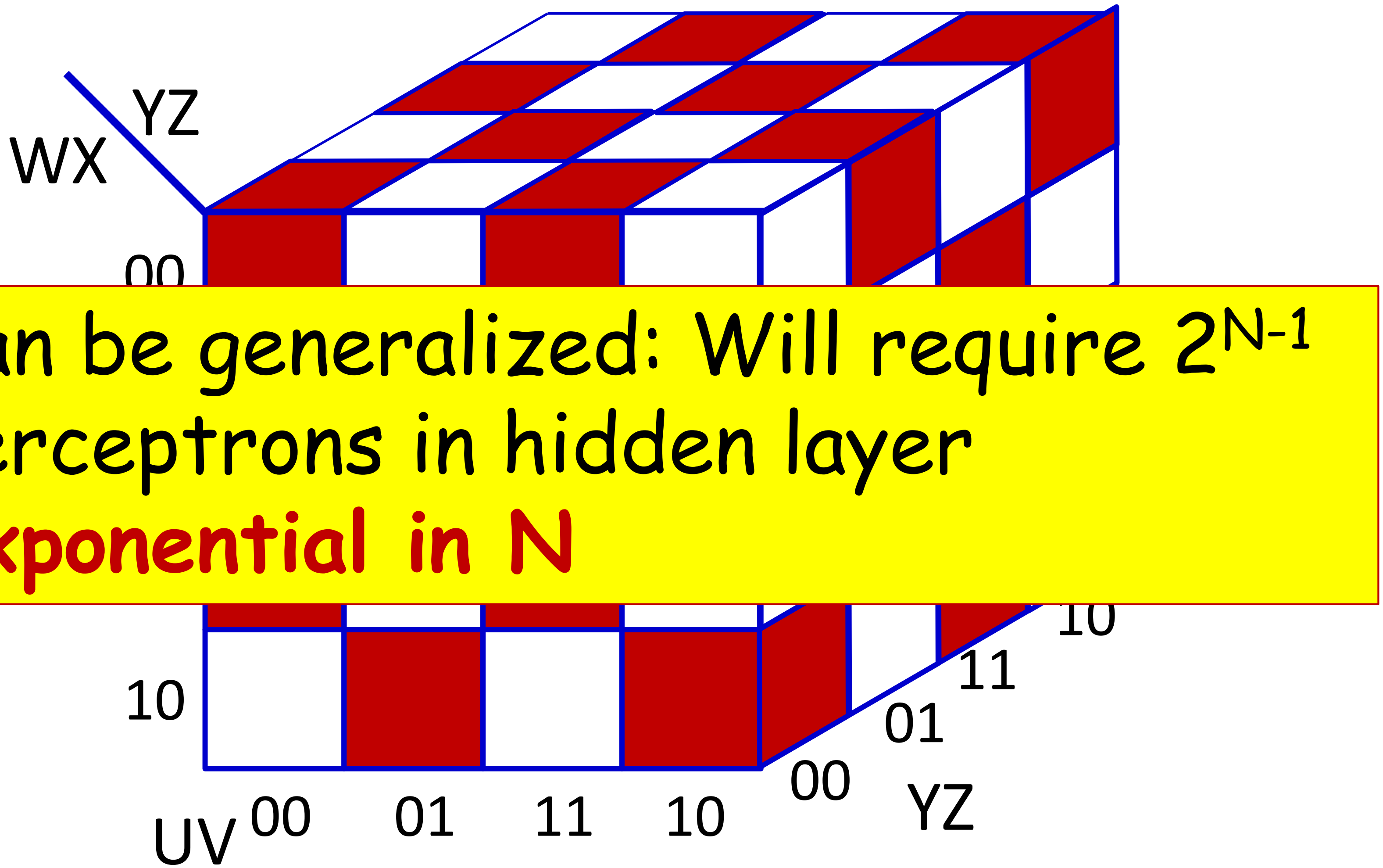
- What arrangement of ones and zeros simply cannot be reduced further?

Width of a single-layer Boolean MLP



- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function of 6 variables?

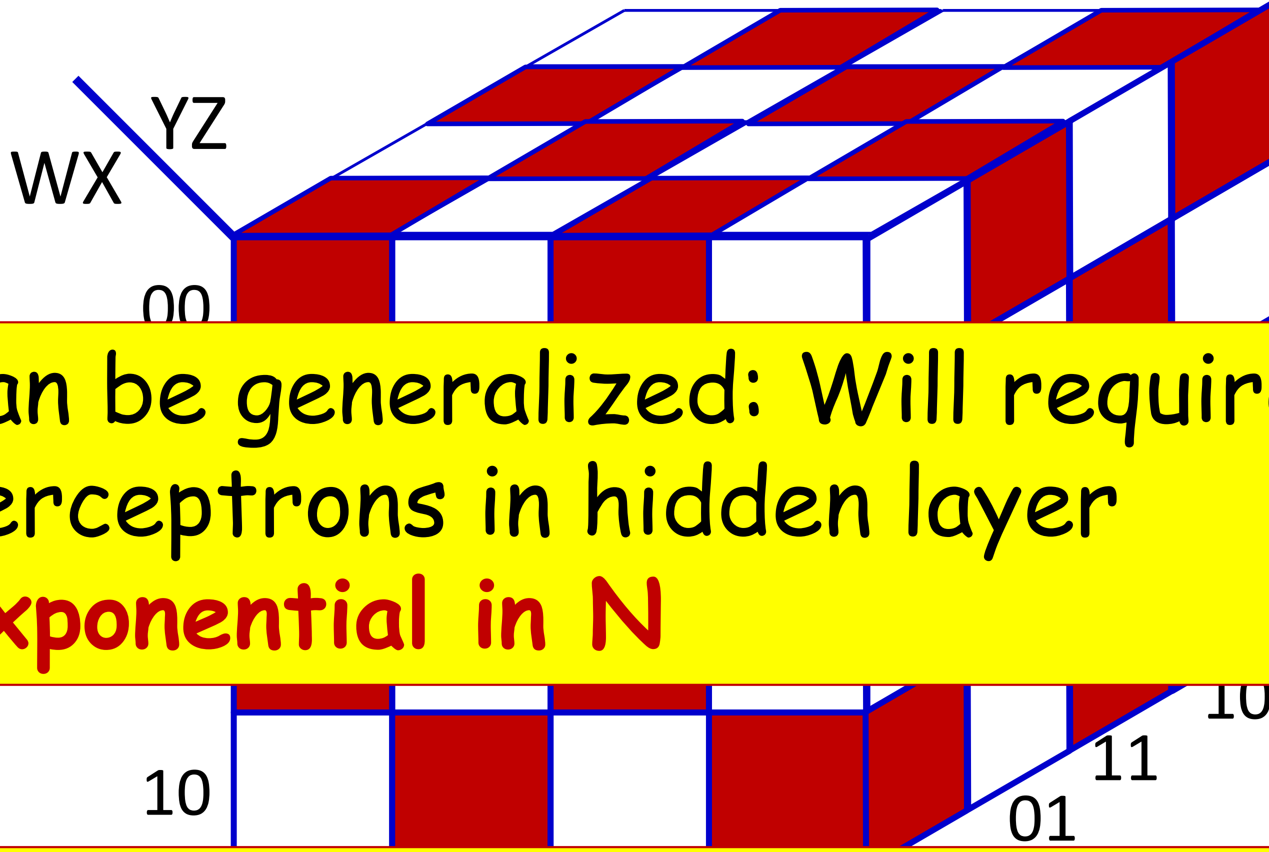
Width of a single-layer Boolean MLP



Can be generalized: Will require 2^{N-1} perceptrons in hidden layer
Exponential in N

- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function

Width of a single-layer Boolean MLP



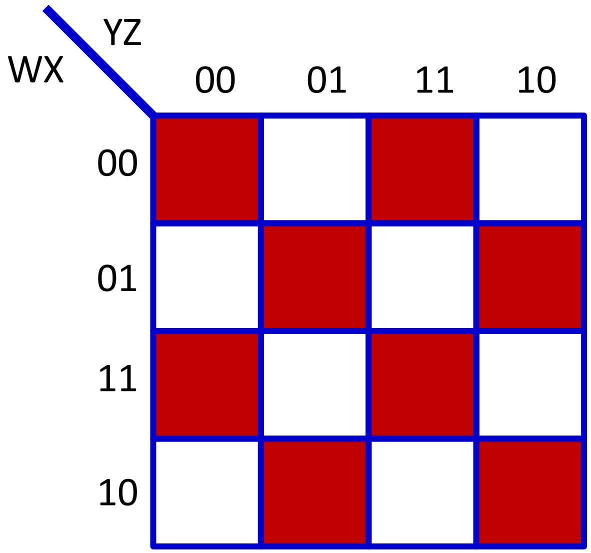
Can be generalized: Will require 2^{N-1} perceptrons in hidden layer

Exponential in N

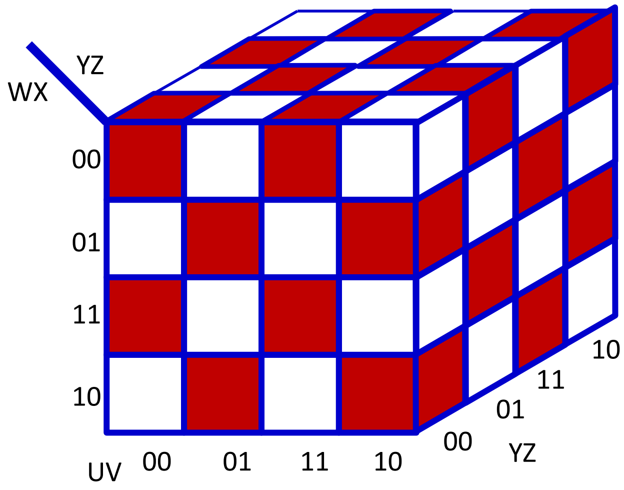
How many units if we use multiple layers?

- How many neurons in a DNF (one-hidden-layer) MLP for this Boolean function

Width of a deep MLP

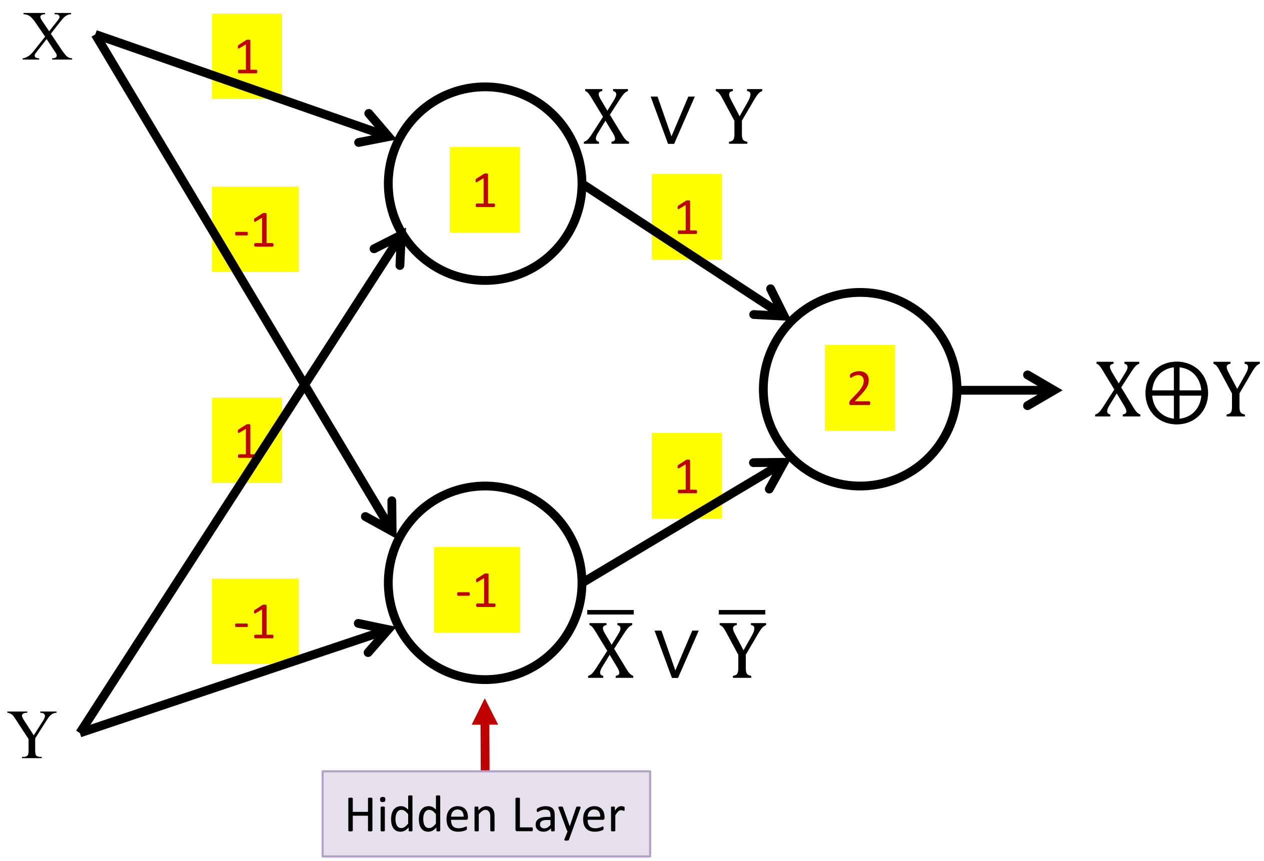


$$O = W \oplus X \oplus Y \oplus Z$$



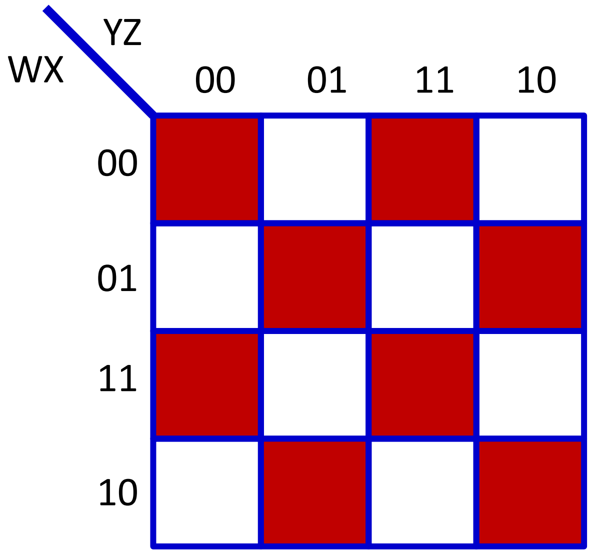
$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

Multi-layer perceptron XOR

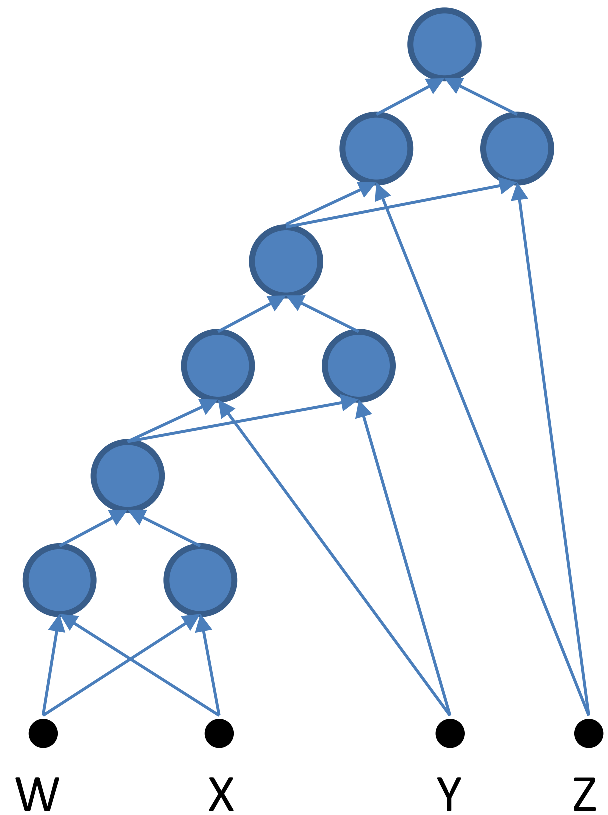


- An XOR takes three perceptrons

Width of a deep MLP



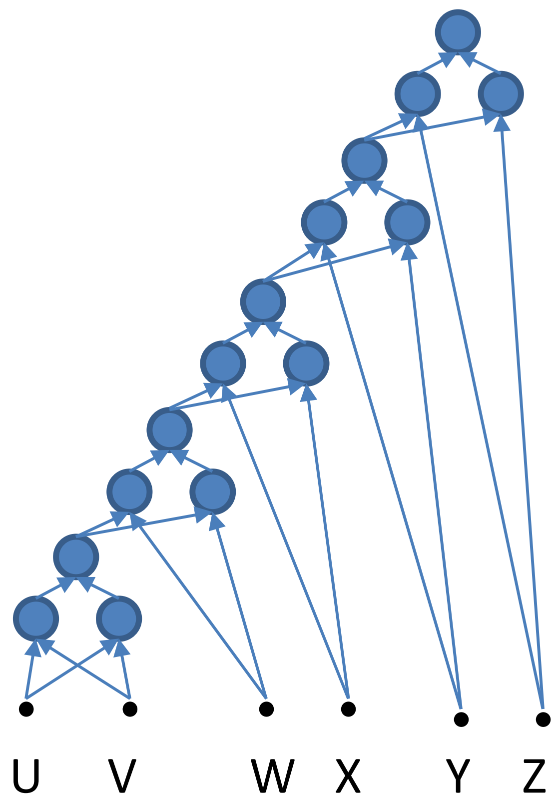
$$O = W \oplus X \oplus Y \oplus Z$$

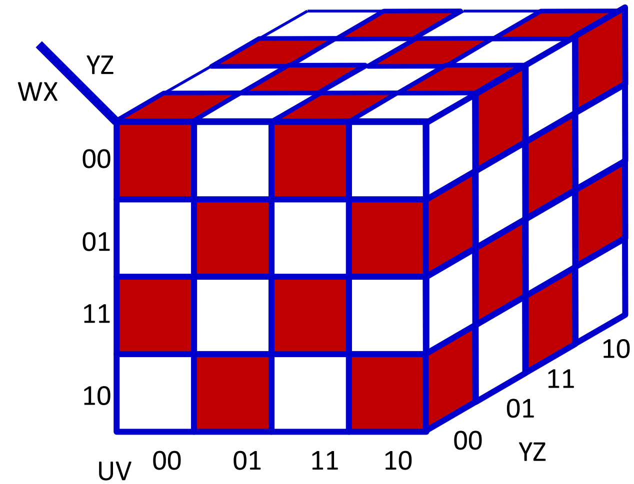


9 perceptrons

- An XOR needs 3 perceptrons
- This network will require 3x3 = 9 perceptrons

Width of a deep MLP



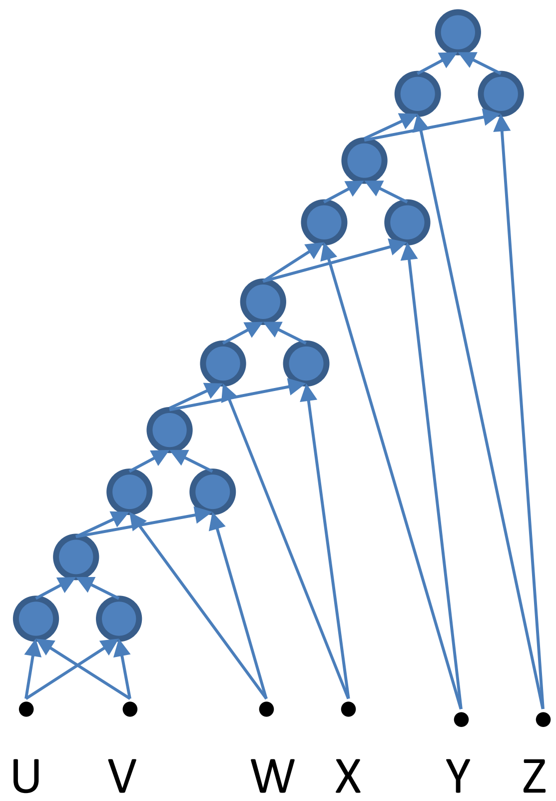


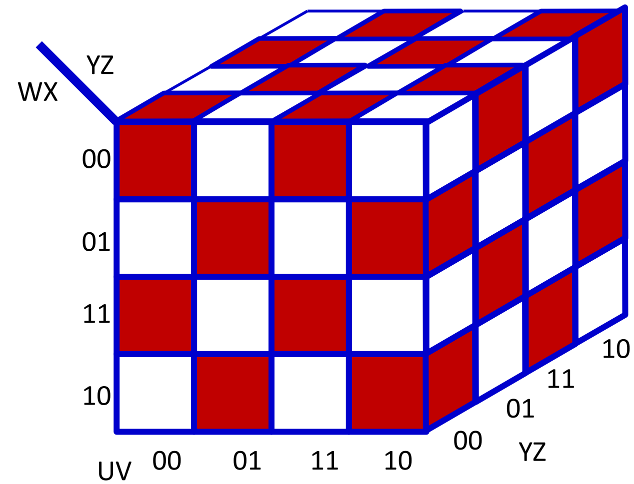
$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

15 perceptrons

- An XOR needs 3 perceptrons
- This network will require 3x5 = 15 perceptrons

Width of a deep MLP



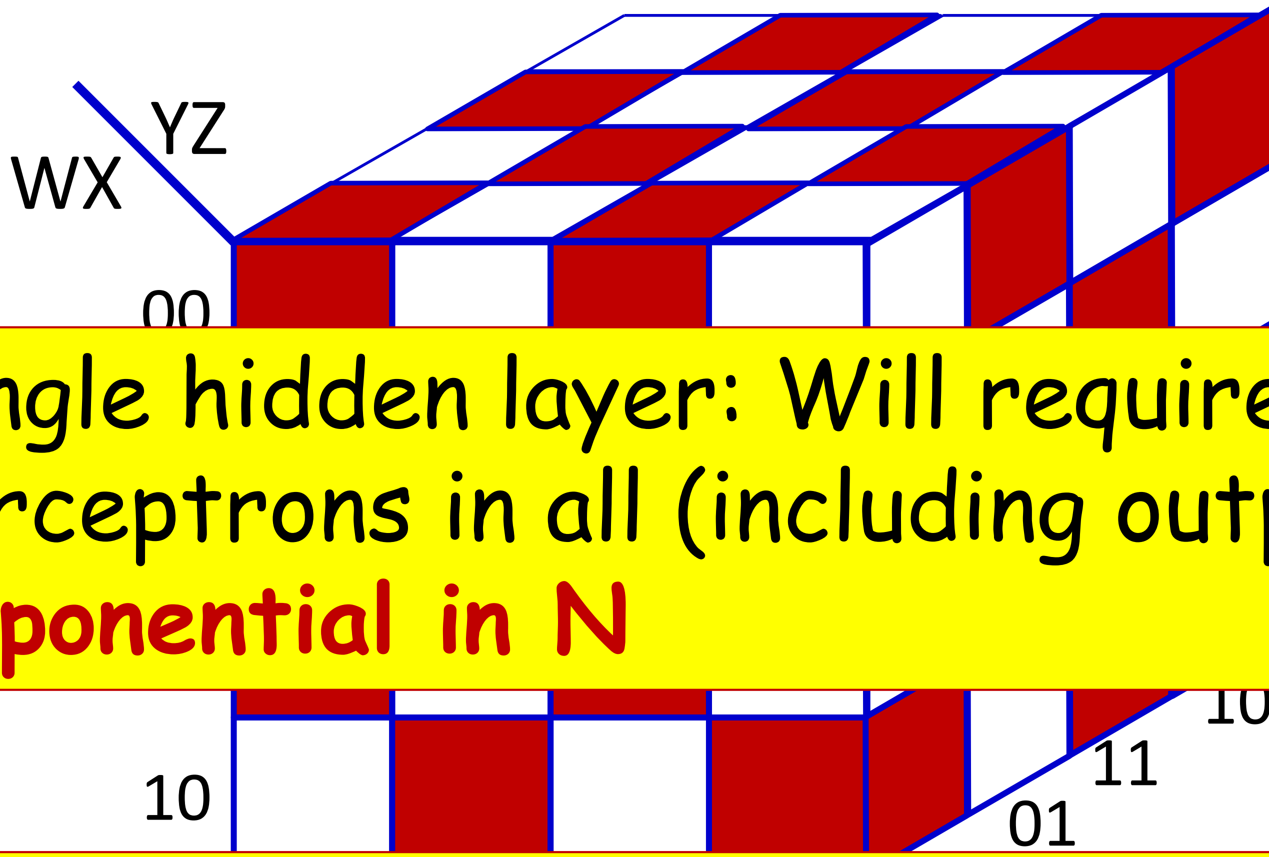


$$O = U \oplus V \oplus W \oplus X \oplus Y \oplus Z$$

More generally, the XOR of N variables will require 3(N-1) perceptrons!!

- An XOR needs 3 perceptrons
- This network will require 3x5 = 15 perceptrons

Width of a single-layer Boolean MLP



Single hidden layer: Will require $2^{N-1}+1$ perceptrons in all (including output unit)

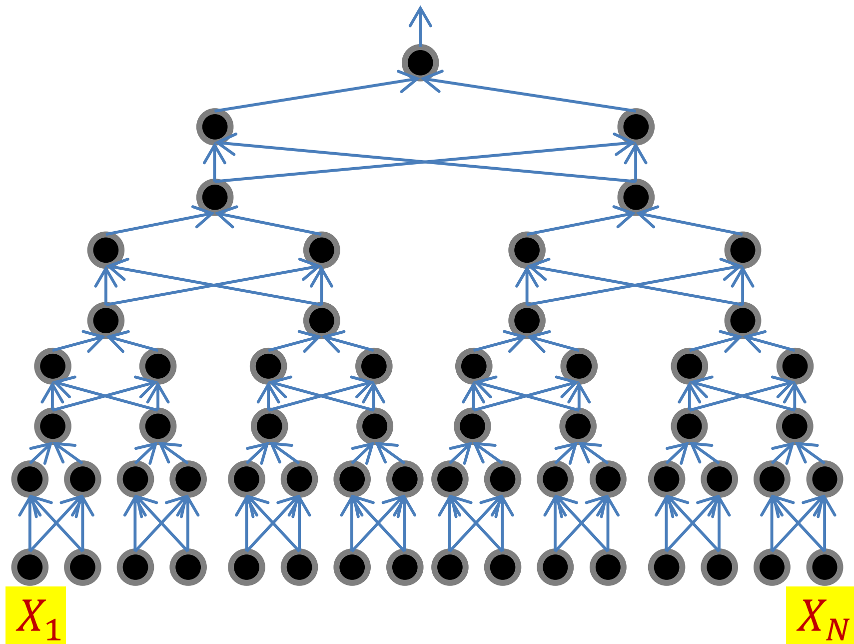
Exponential in N

Will require 3(N-1) perceptrons in a deep network

Linear in N!!!

Can be arranged in only $2\log_2(N)$ layers

A better representation

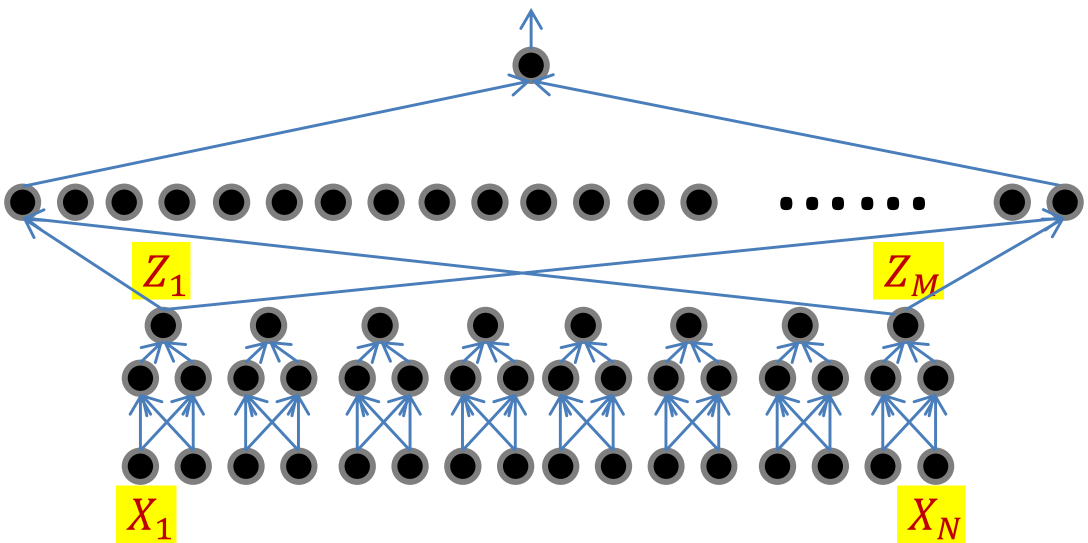


$$O = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$

- Only $2\log_2 N$ layers
  - By pairing terms
  - 2 layers per XOR

$$O = (((((X_1 \oplus X_2) \oplus (X_1 \oplus X_2)) \oplus ((X_5 \oplus X_6) \oplus (X_7 \oplus X_8))) \oplus (((\ldots$$

The challenge of depth



$$O = X_1 \oplus X_2 \oplus \cdots \oplus X_N$$
$$= Z_1 \oplus Z_2 \oplus \cdots \oplus Z_M$$

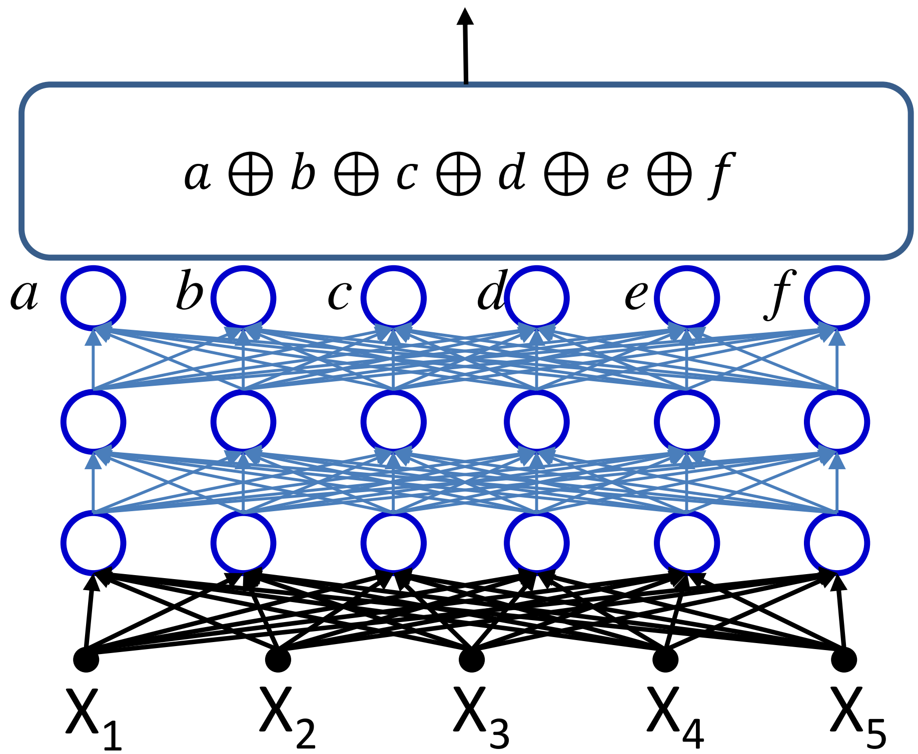
- Using only K hidden layers will require $O(2^{(N-K/2)})$ neurons in the Kth layer
  – Because the output can be shown to be the XOR of all the outputs of the K-1th hidden layer
  – **I.e. reducing the number of layers below the minimum will result in an exponentially sized network to express the function fully**
  – **A network with fewer than the required number of neurons *cannot* model the function**

Recap: The need for depth

- *Deep* Boolean MLPs that scale *linearly* with the number of inputs …
- … can become exponentially large if recast using only one layer

- It gets worse..

The need for depth



- The wide function can happen at any layer
- Having a few extra layers can greatly reduce network size

Depth vs Size in Boolean Circuits

- The XOR is really a parity problem

- Any *Boolean* circuit of depth d using AND,OR and NOT gates with unbounded fan-in must have size $2^{n^{1/d}}$
  - Parity, Circuits, and the Polynomial-Time Hierarchy, M. Furst, J. B. Saxe, and M. Sipser, Mathematical Systems Theory 1984
  - Alternately stated: $parity \notin AC^0$
    - Set of constant-depth polynomial size circuits of unbounded fan-in elements

Caveat: Not all Boolean functions..

- Not all Boolean circuits have such clear depth-vs-size tradeoff

- Shannon's theorem: For $n > 2$, there is Boolean function of n variables that requires at least $2^n/n$ gates
  - More correctly, for large n,almost all n-input Boolean functions need more than $2^n/n$ gates

- Note: If all Boolean functions over n inputs could be computed using a circuit of size that is polynomial in n, P = NP!

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

Story so far

- Multi-layer perceptrons are *Universal Boolean Machines*
- 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 to express the same function
  - Could be *exponentially* smaller

Caveat

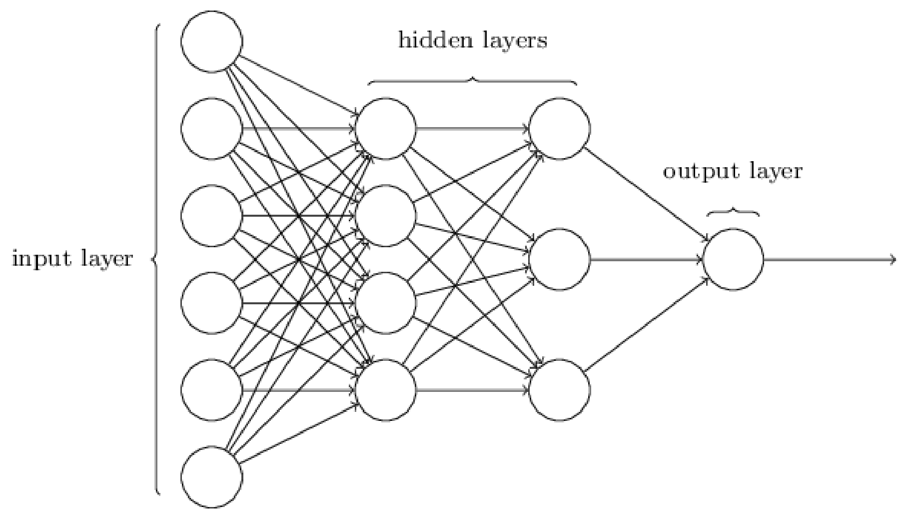
- Used a simple "Boolean circuit" analogy for explanation
- We actually have *threshold circuit* (TC) not, just a Boolean circuit (AC)
  - Specifically composed of threshold gates
    - More versatile than Boolean gates
      - E.g. "at least K inputs are 1" is a single TC gate, but an exponential size AC
      - For fixed depth, $Boolean\ circuits \subset threshold\ circuits$ (strict subset)
  - A depth-2 TC parity circuit can be composed with $\mathcal{O}(n^2)$ weights
    - But a network of depth $\log(n)$ requires only $\mathcal{O}(n)$ weights
  - But more generally, for large n, for most Boolean functions, a threshold circuit that is polynomial in n at optimal depth d becomes exponentially large at $d - 1$

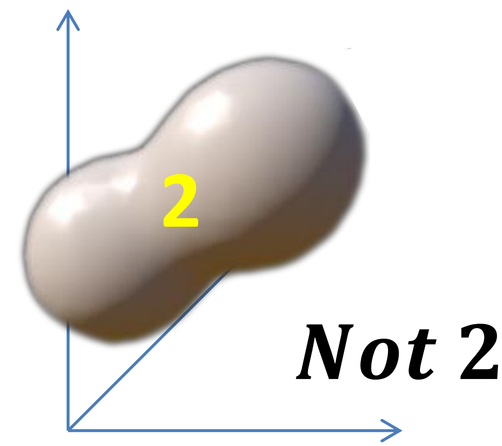
- Other formal analyses typically view neural networks as *arithmetic circuits*
  - Circuits which compute polynomials over any field

- So lets consider functions over the field of reals

The MLP as a classifier

784 dimensions
(MNIST)

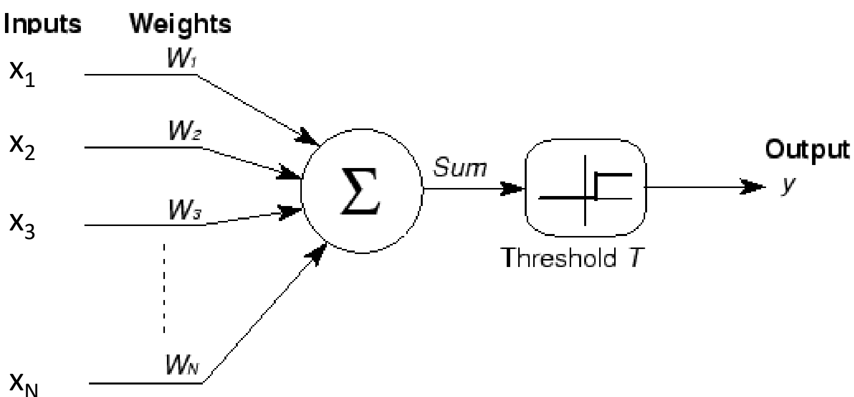




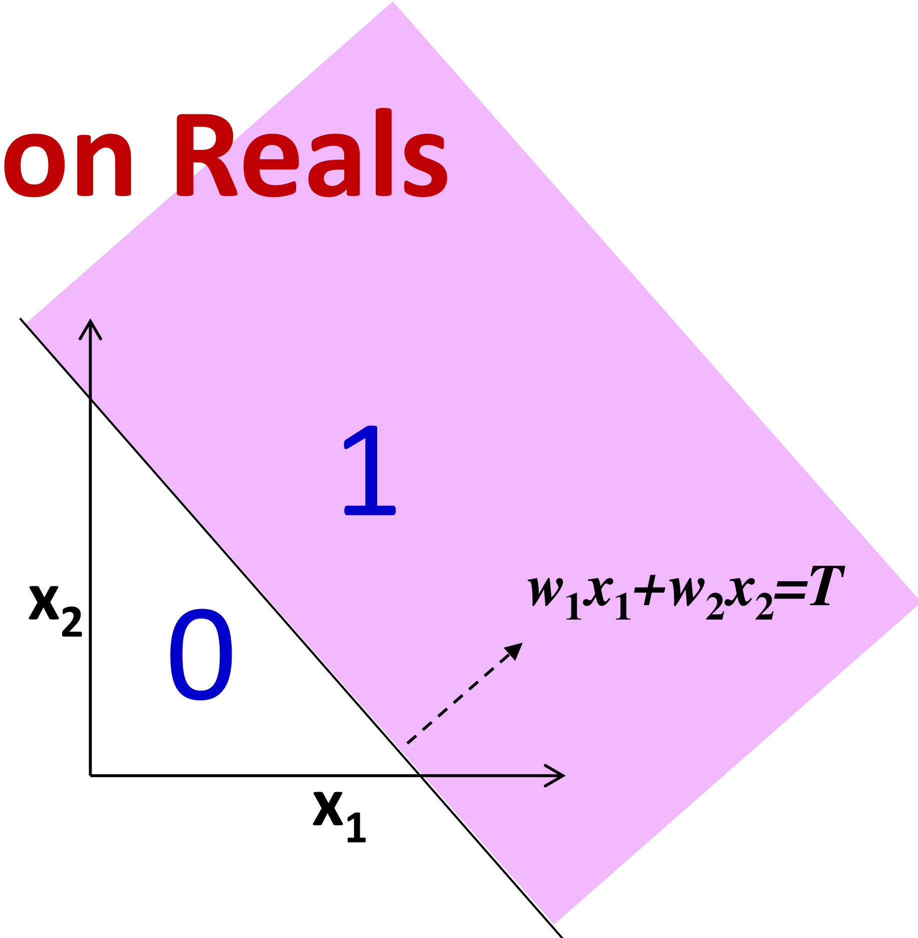
784 dimensions

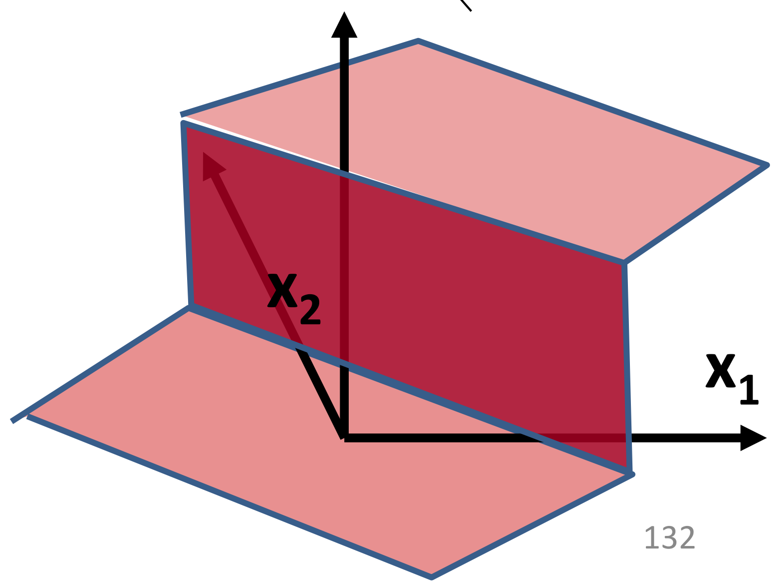
- MLP as a function over real inputs
- MLP as a function that finds a complex “decision boundary” over a space of *reals*

A Perceptron on Reals



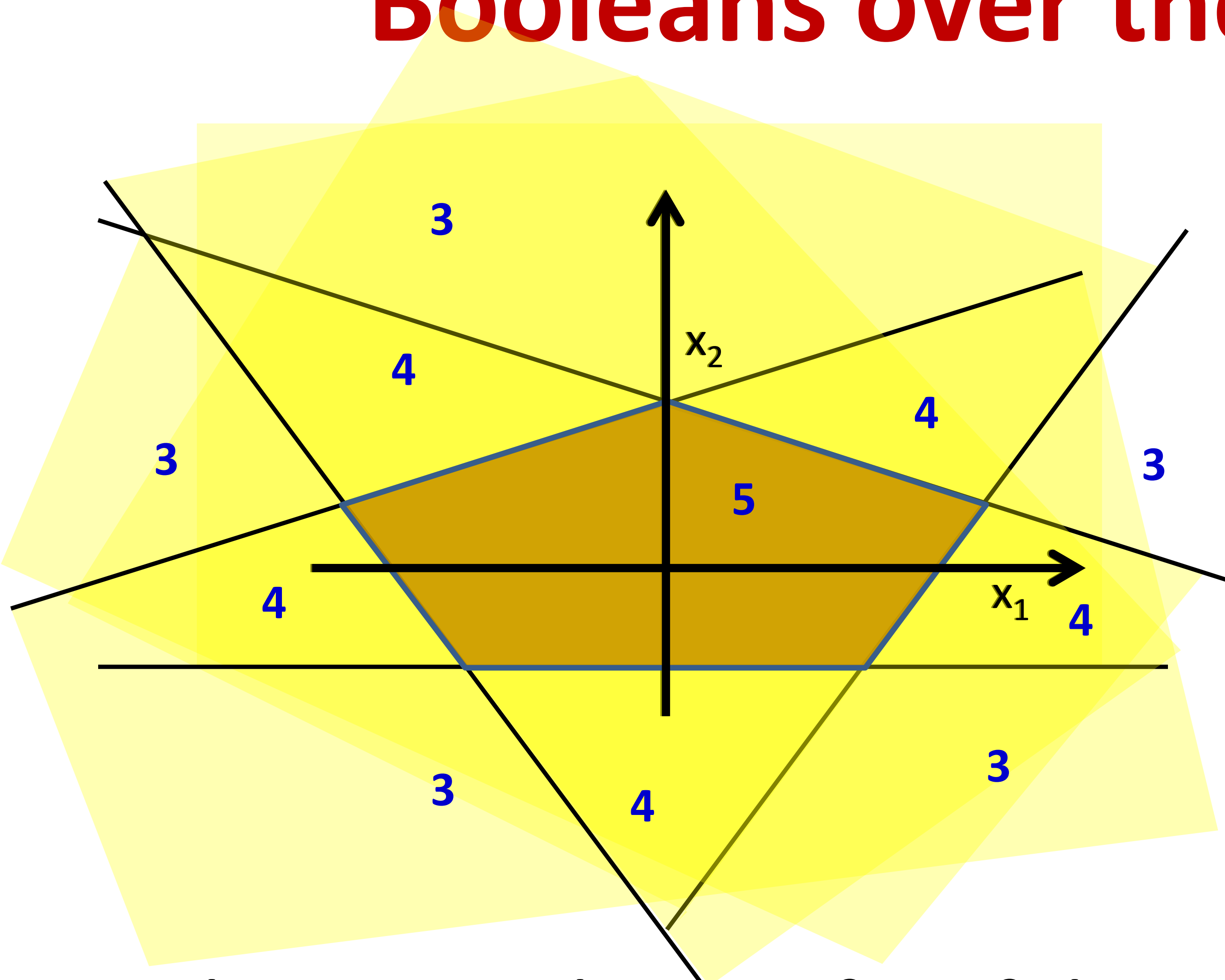
$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$

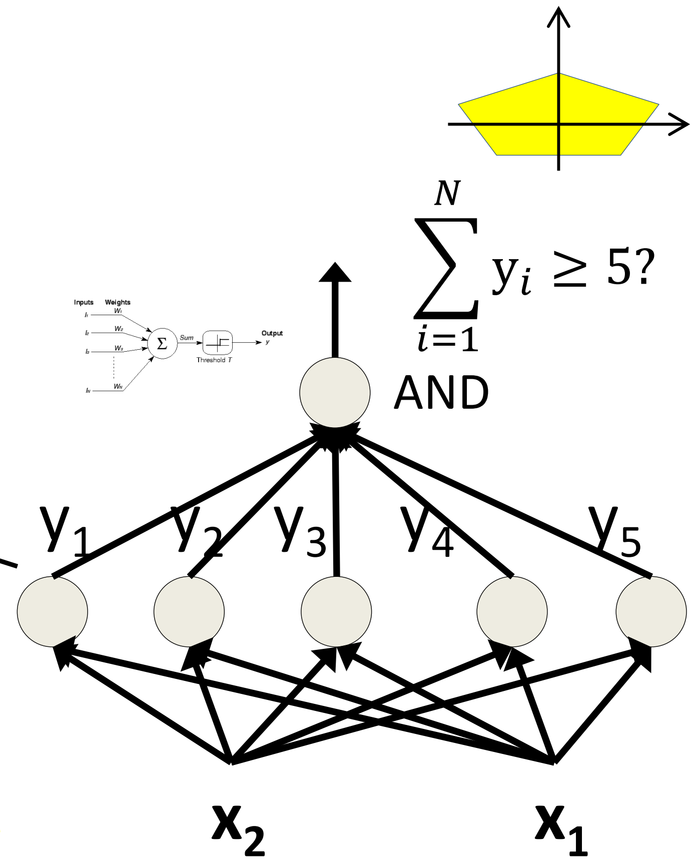




- A perceptron operates on *real*-valued vectors
  - **This is a *linear classifier***

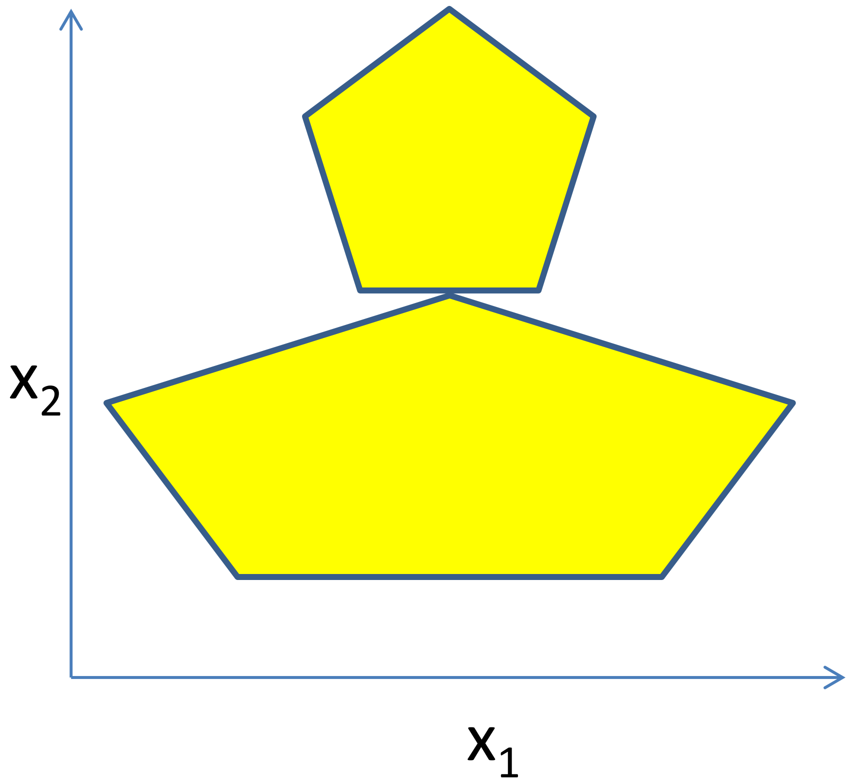
Booleans over the reals

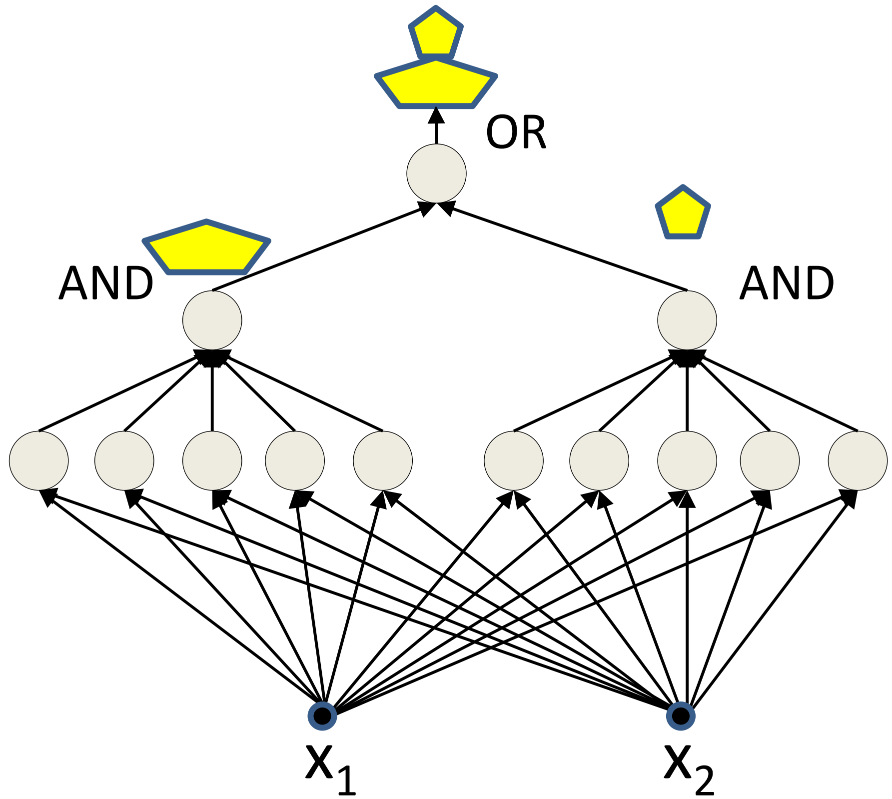




- The network must fire if the input is in the coloured area

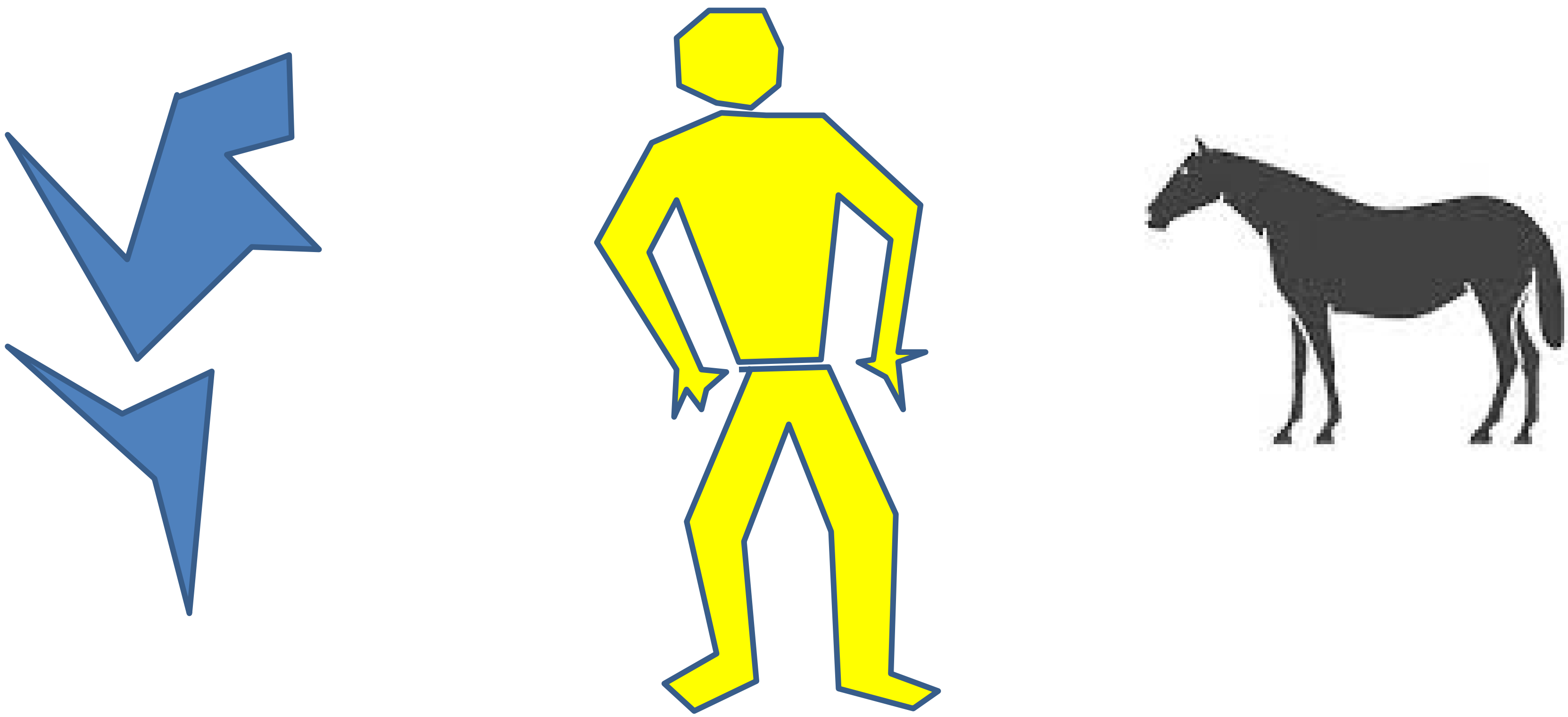
More complex decision boundaries





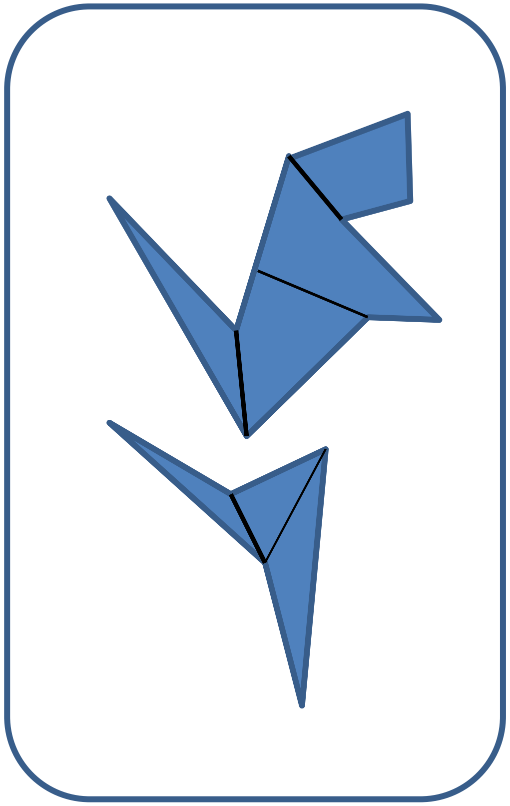
- Network to fire if the input is in the yellow area
  - "OR" two polygons
  - A third layer is required

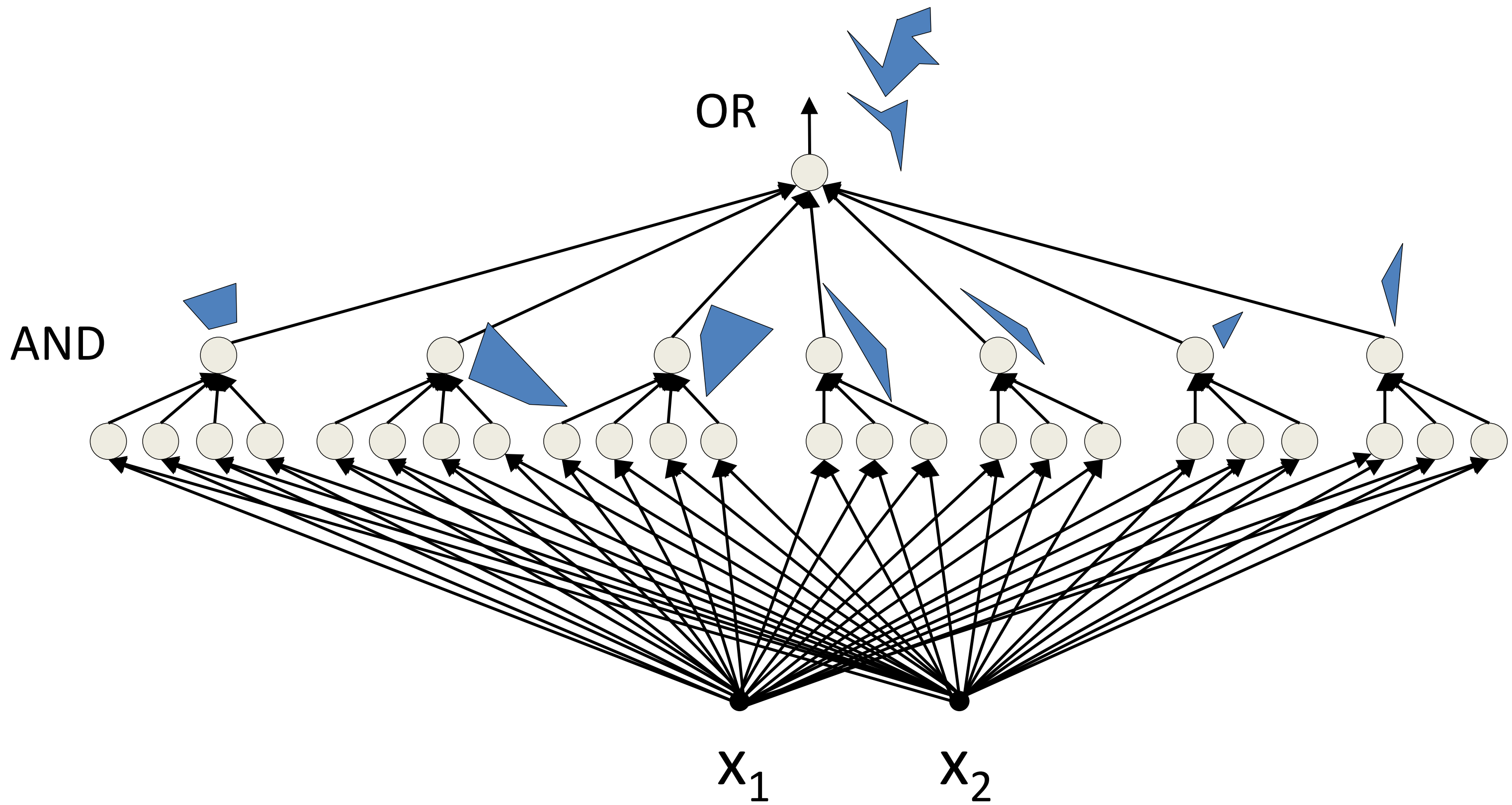
Complex decision boundaries



- Can compose *arbitrarily* complex decision boundaries

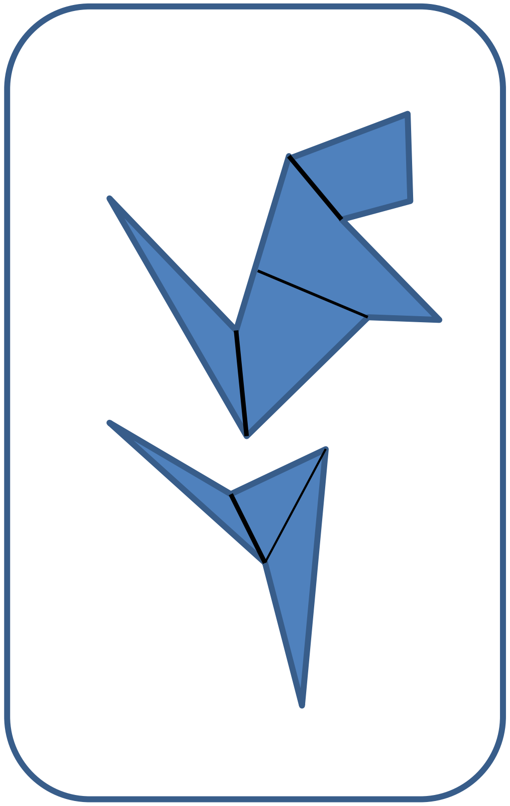
Complex decision boundaries

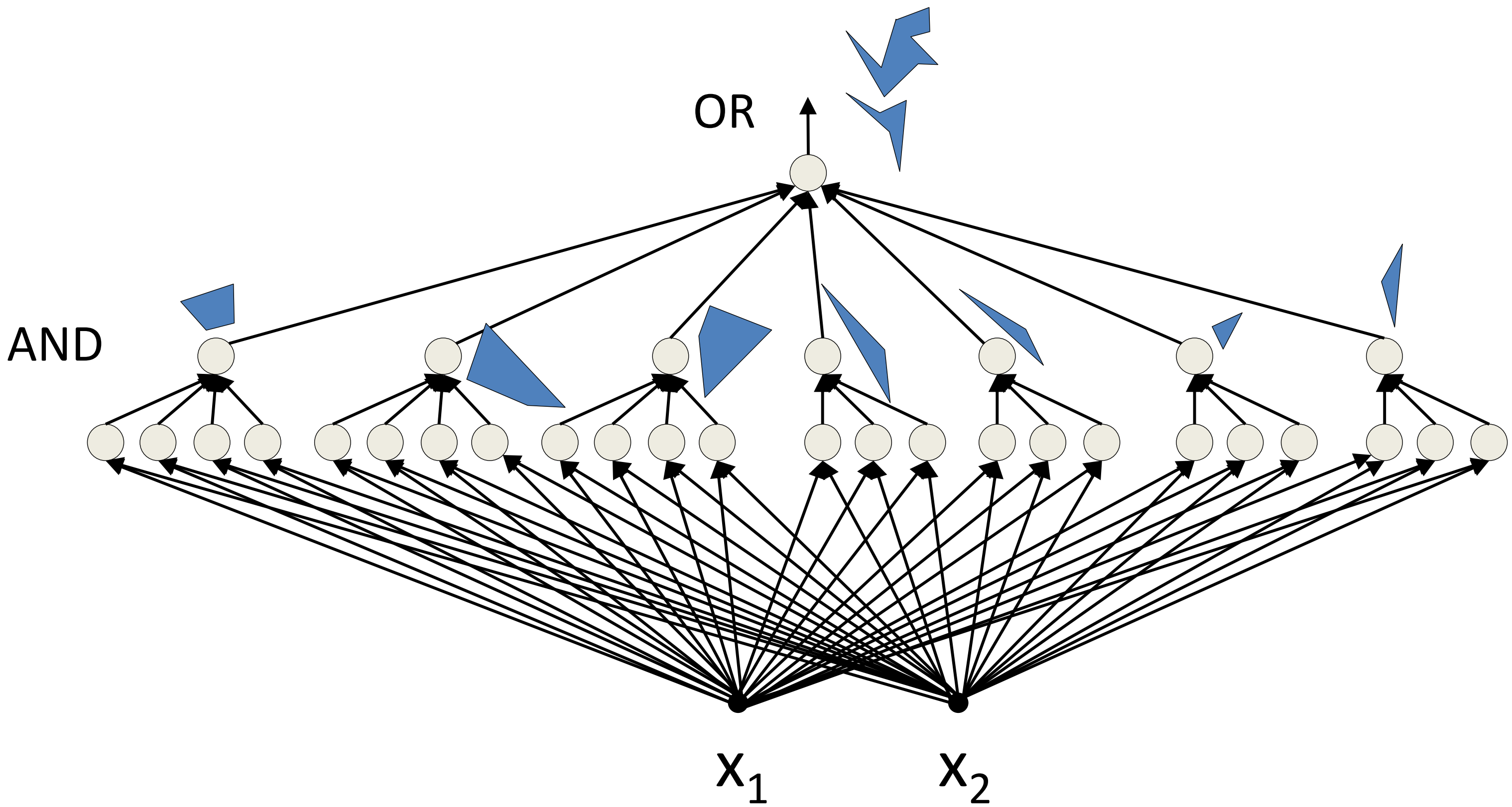




- Can compose *arbitrarily* complex decision boundaries

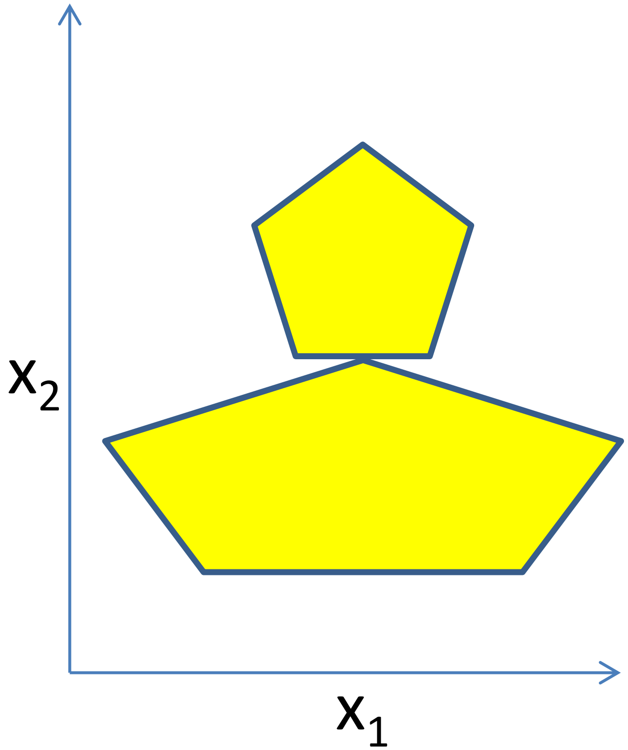
Complex decision boundaries

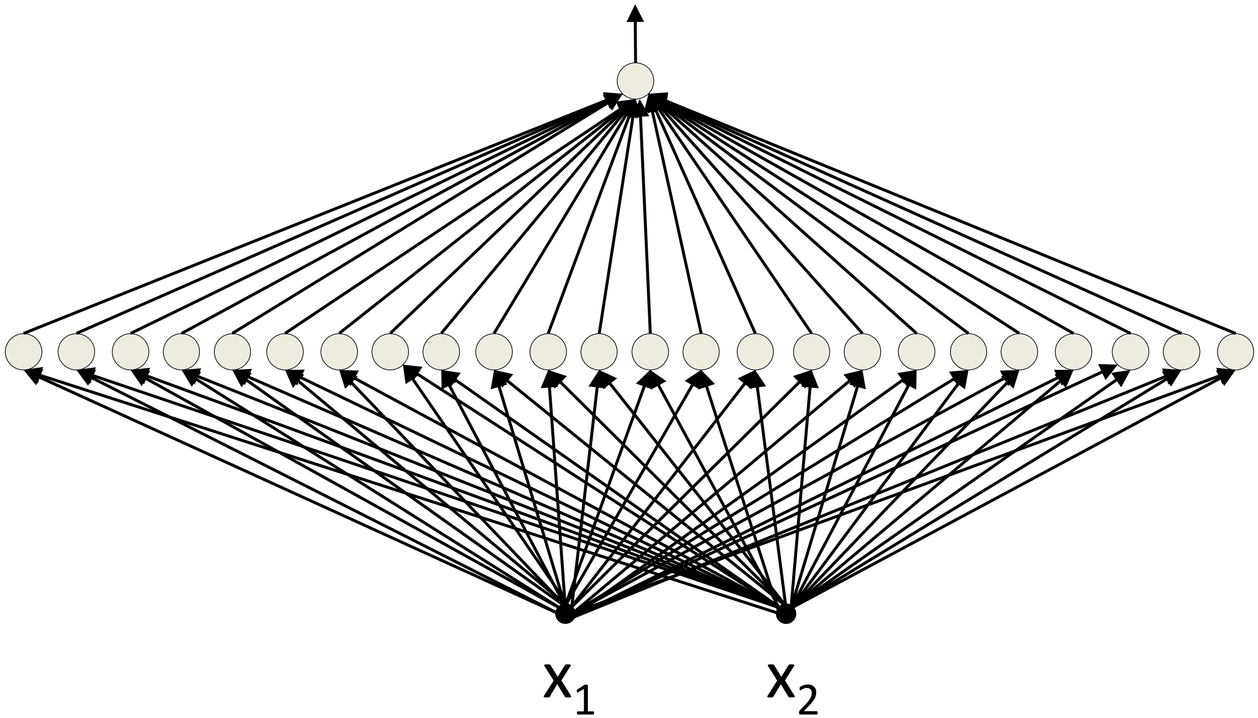




- Can compose *arbitrarily* complex decision boundaries
  - With *only one hidden layer!*
  - **How**?

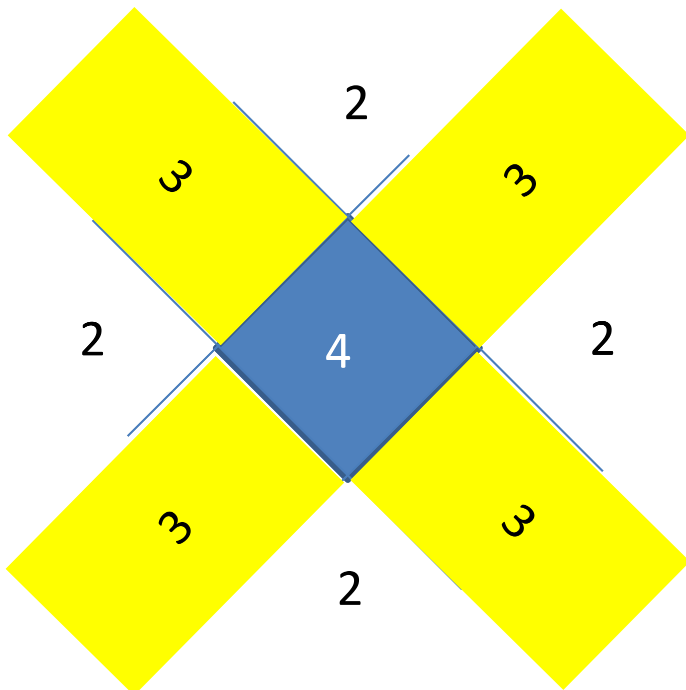
Exercise: compose this with one hidden layer

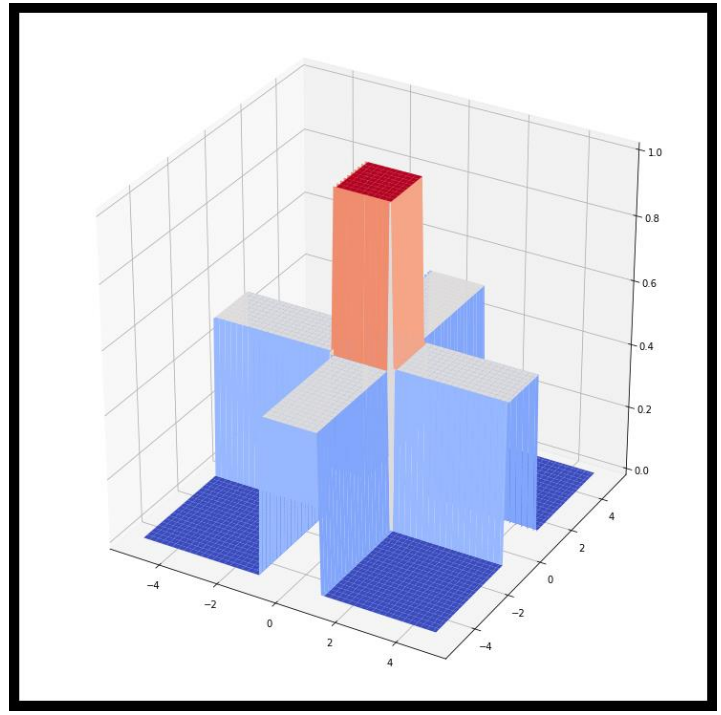




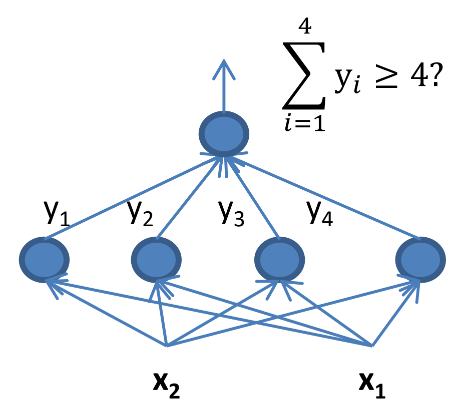
- How would you compose the decision boundary to the left with only *one* hidden layer?

Composing a Square decision boundary

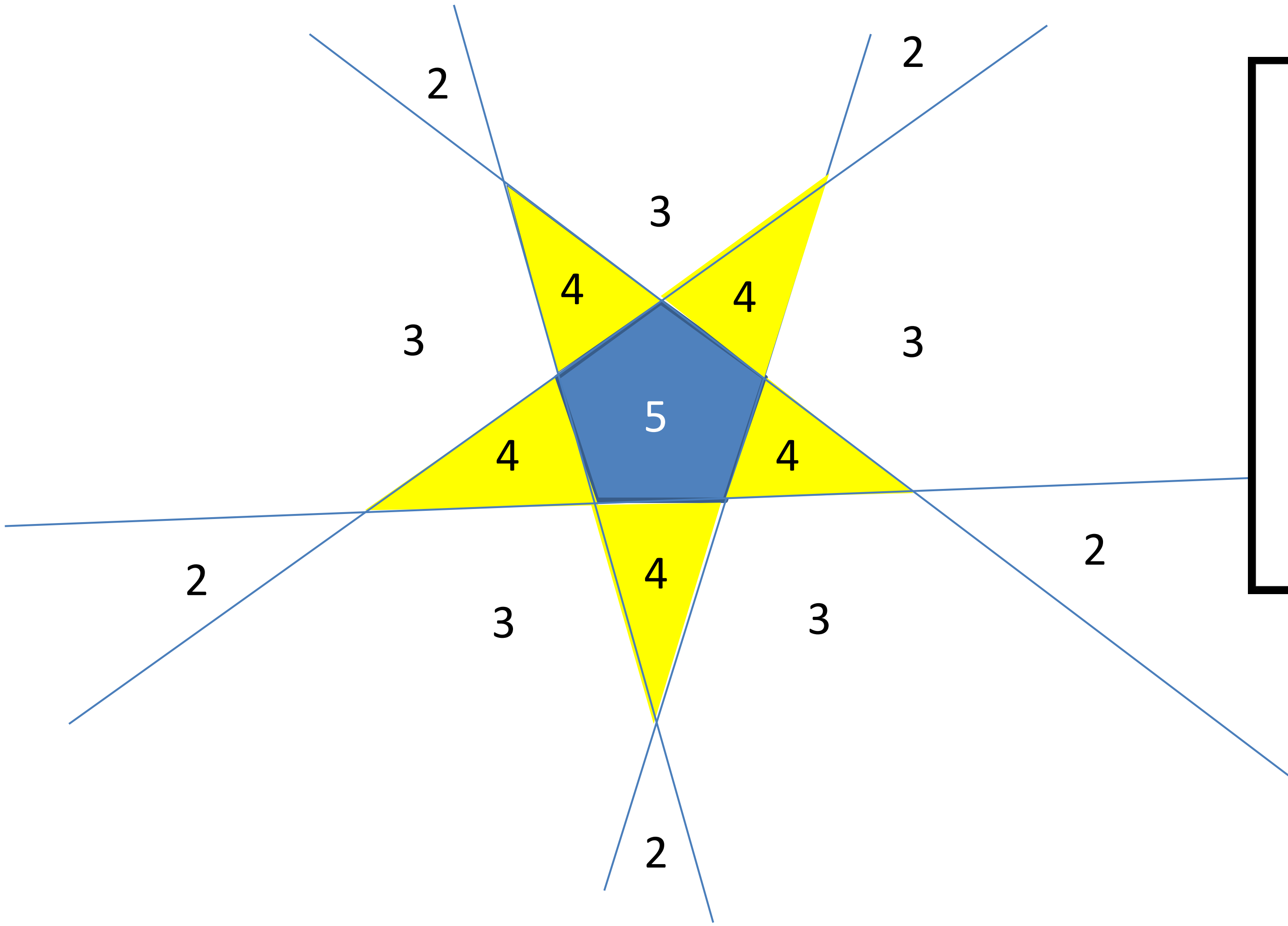


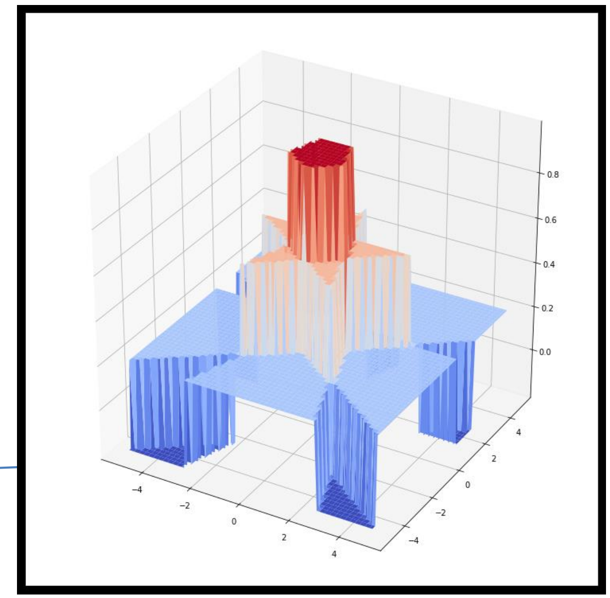


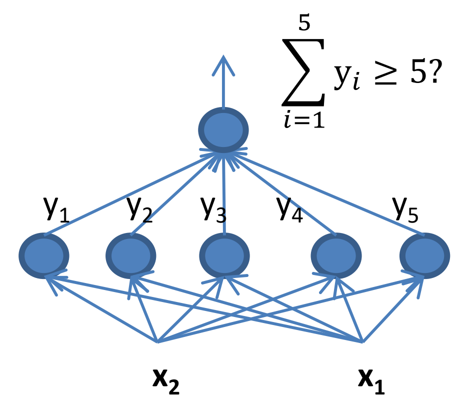
- The polygon net



Composing a pentagon

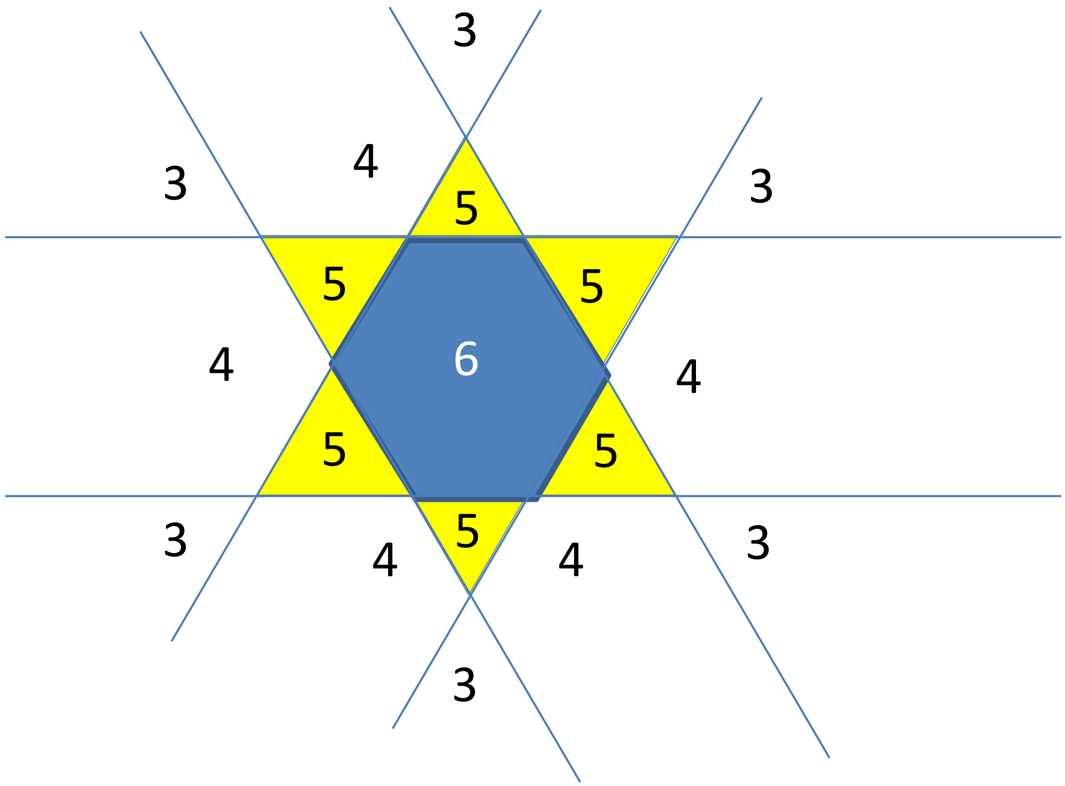


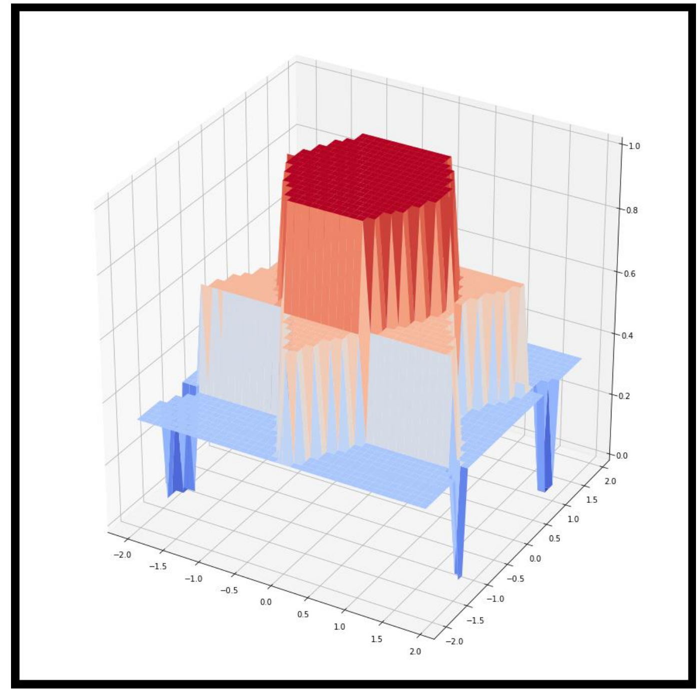




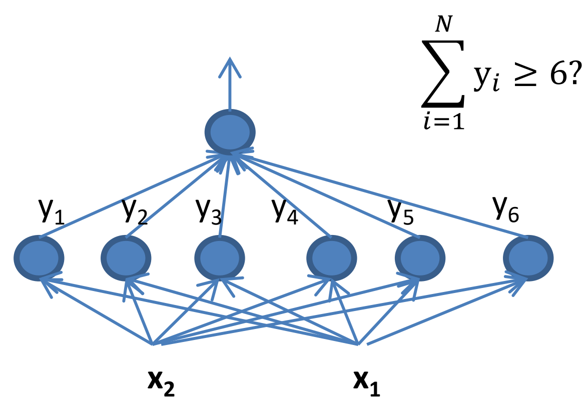
- The polygon net

Composing a hexagon

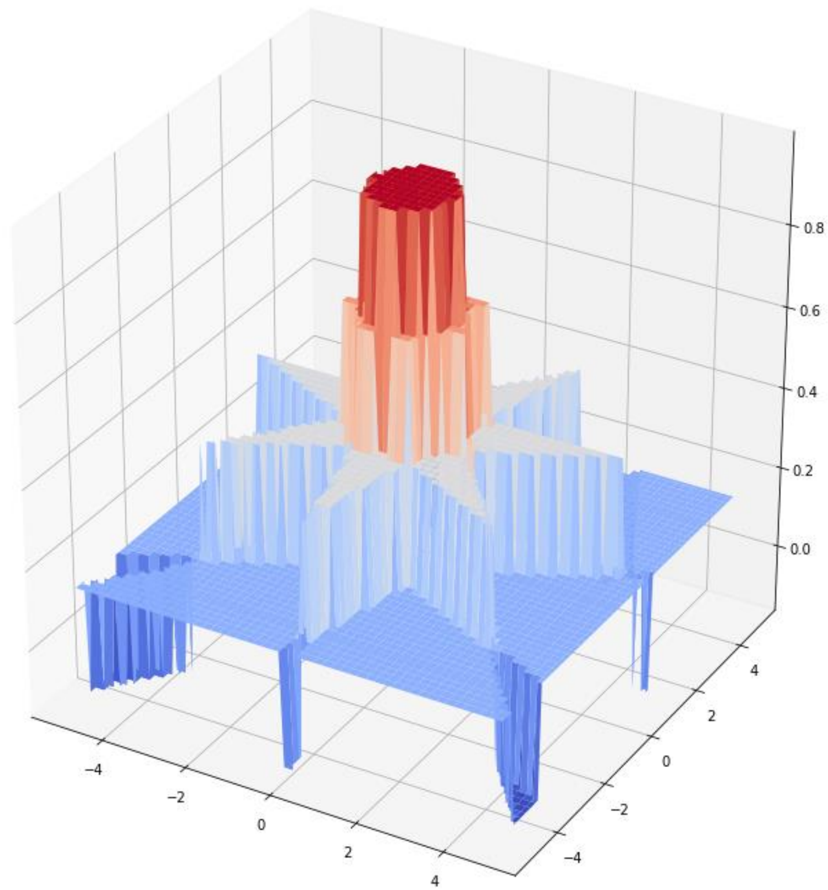




- The polygon net

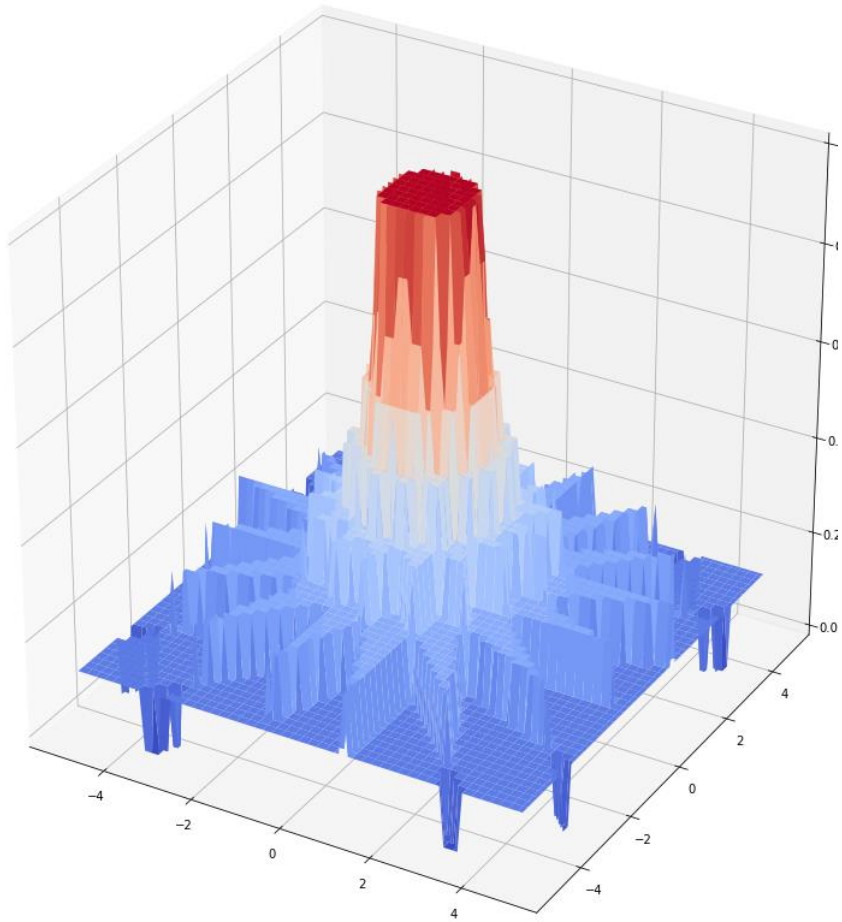


How about a heptagon



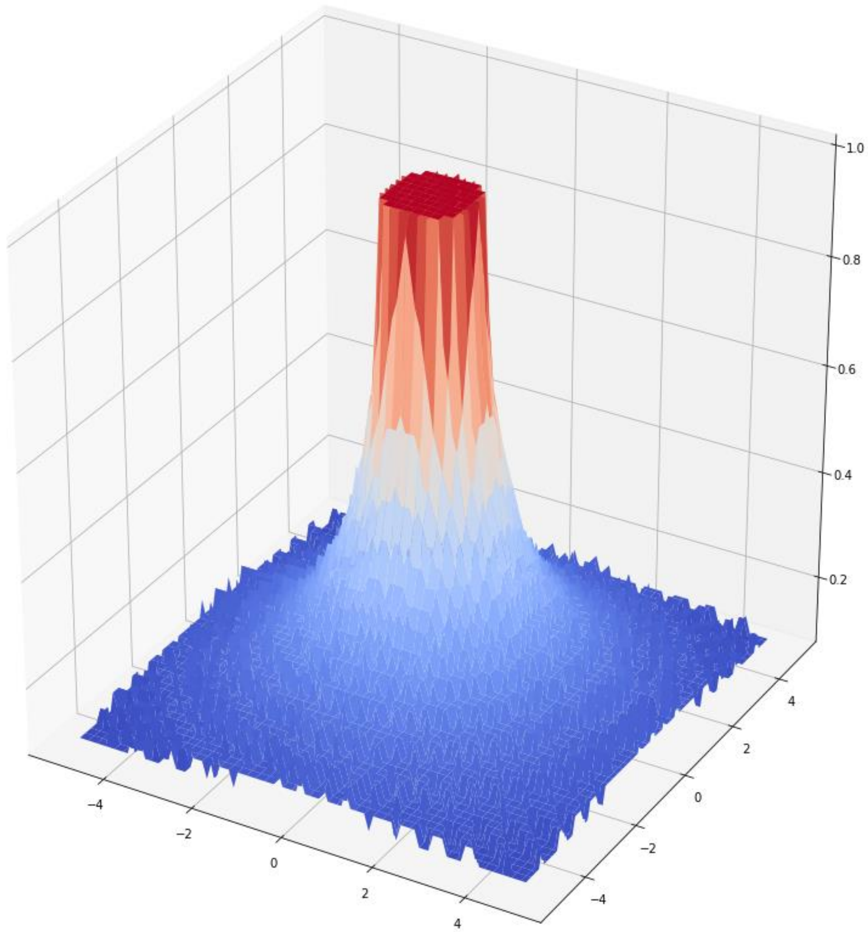
- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

16 sides



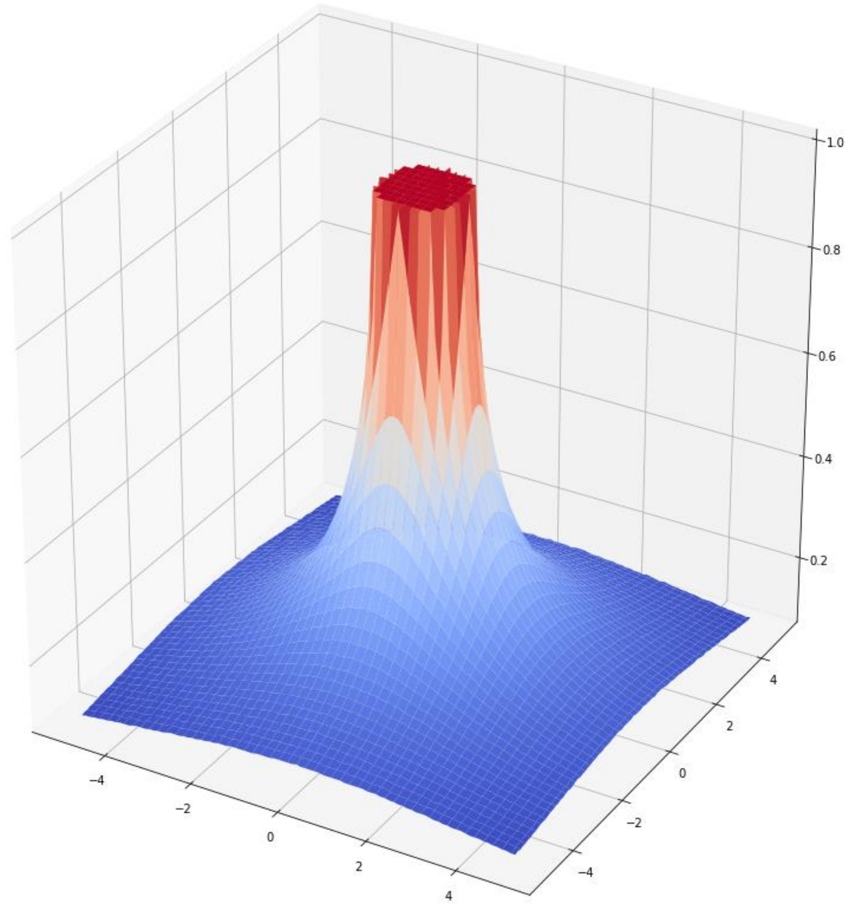
- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

64 sides



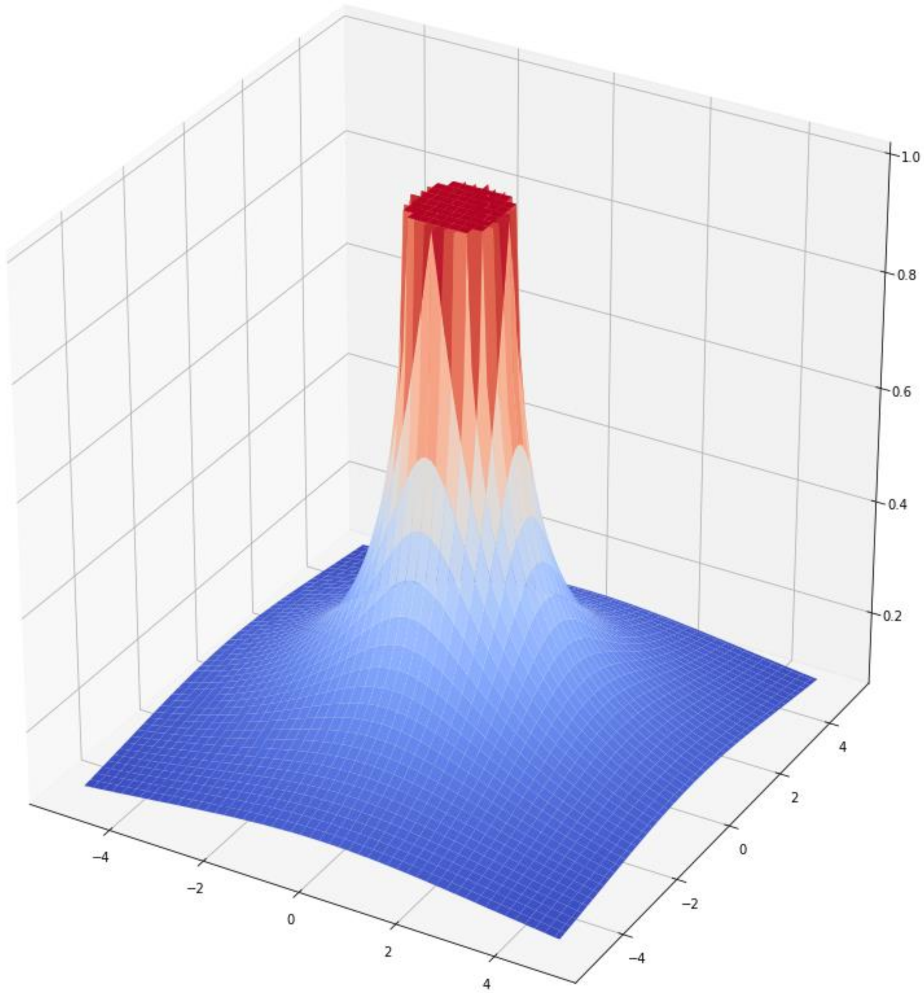
- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

1000 sides

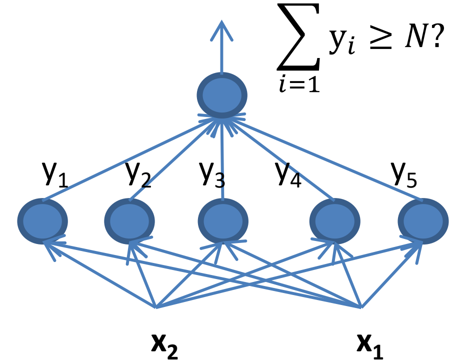


- What are the sums in the different regions?
  - A pattern emerges as we consider N > 6..

Polygon net

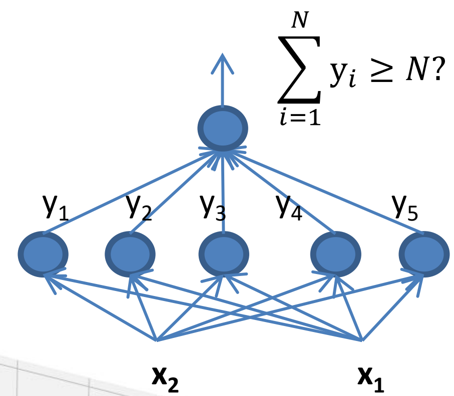


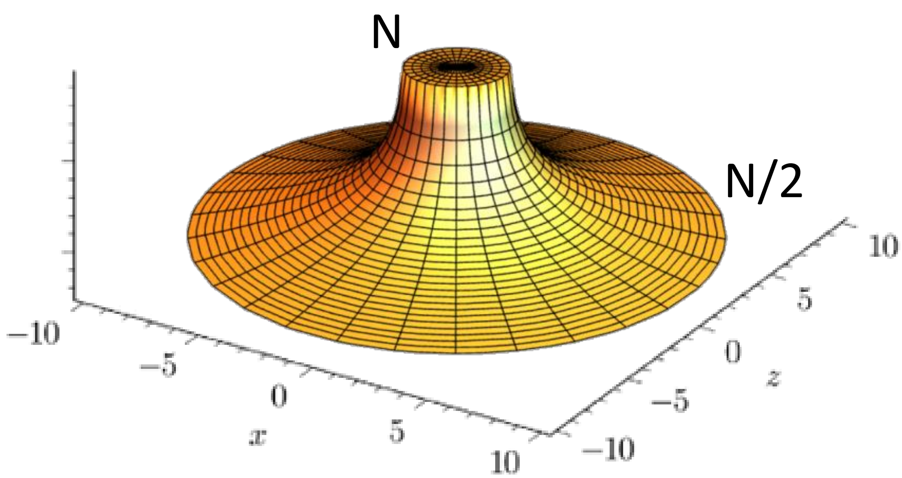
$$\sum_{i=1}^{N} y_i \geq N?$$

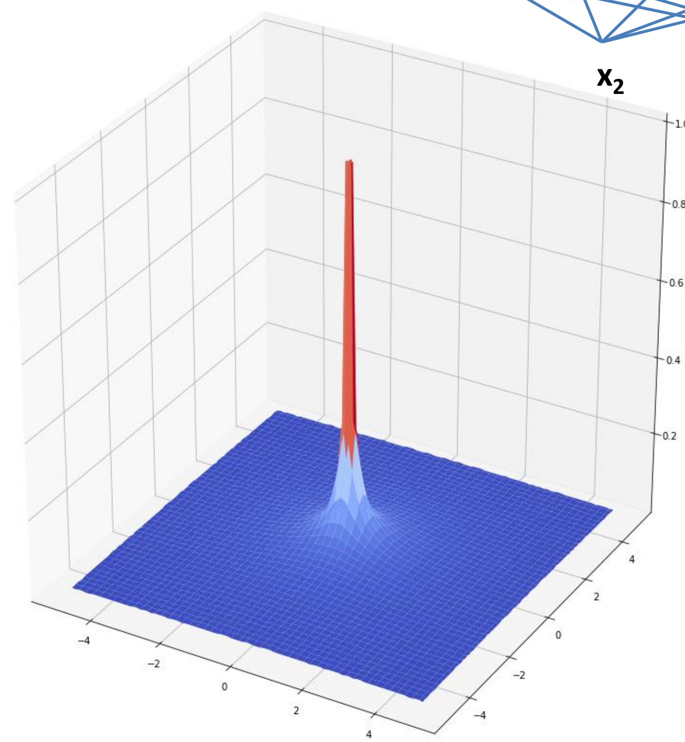


- Increasing the number of sides reduces the area outside the polygon that have N/2 < Sum < N

In the limit

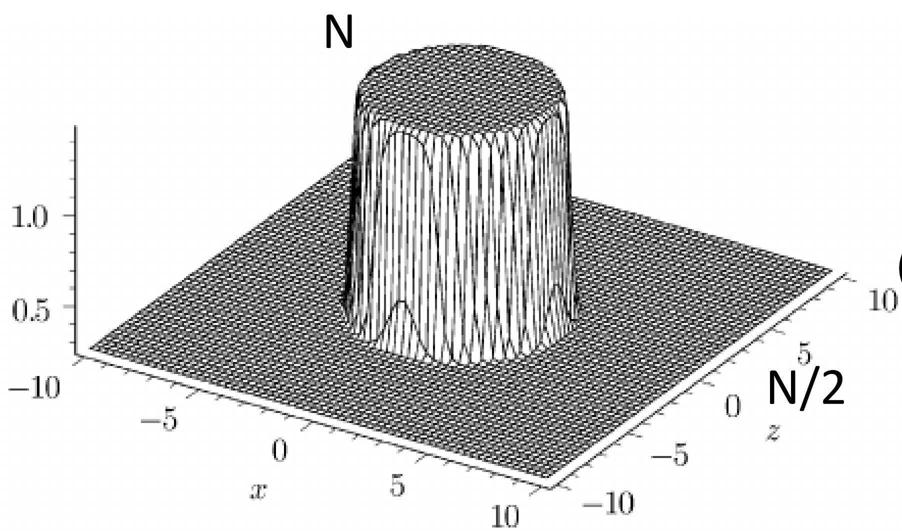


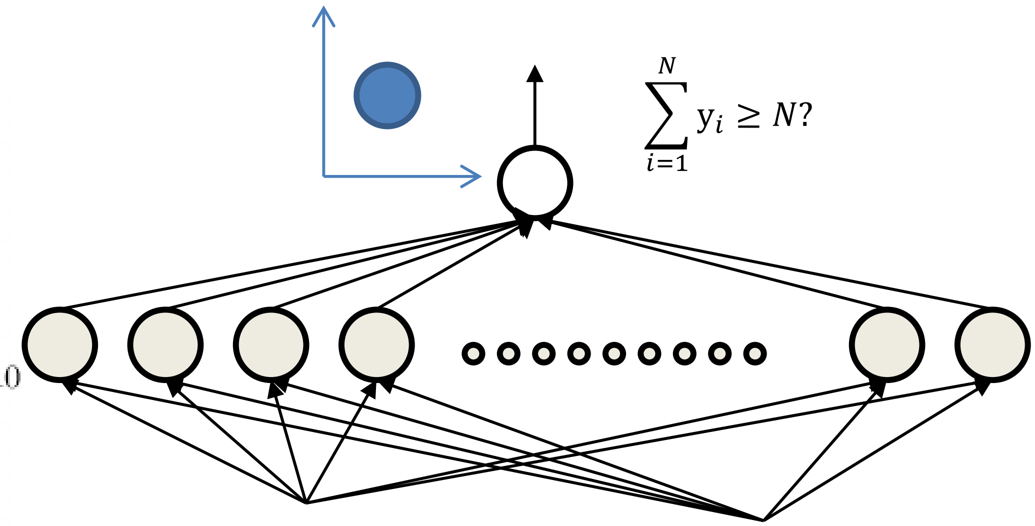




- $\sum_i y_i = N\left(1 - \frac{1}{\pi} arccos\left(min\left(1, \frac{radius}{|\mathbf{x} - center|}\right)\right)\right)$
- For small radius, it's a near perfect cylinder
  - N in the cylinder, N/2 outside

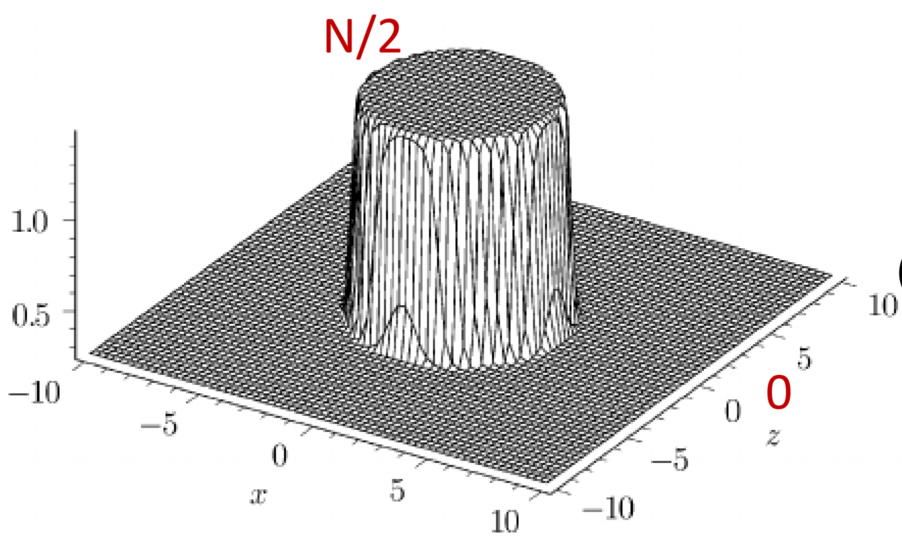
Composing a circle

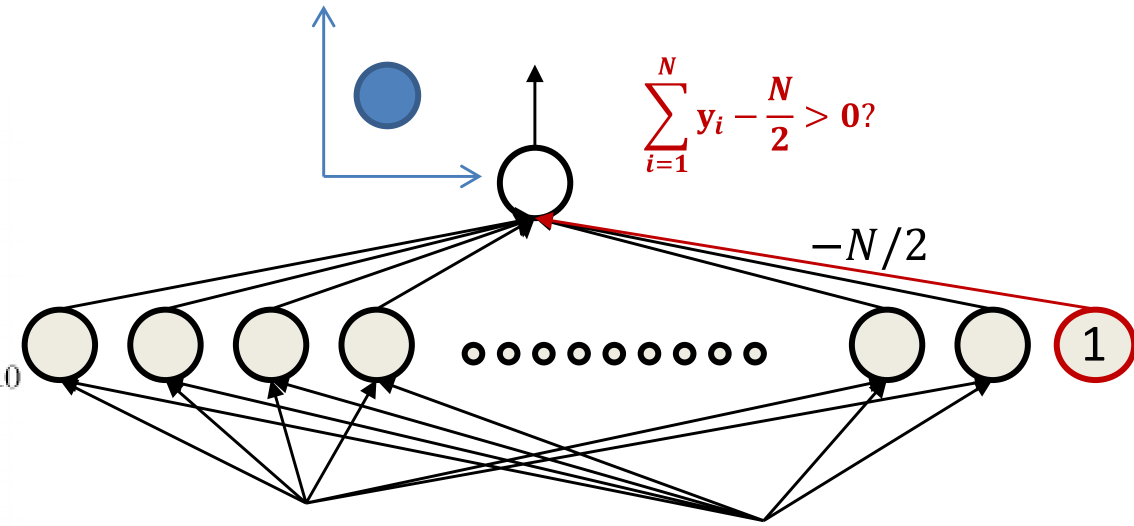




- The circle net
  - Very large number of neurons
  - *Sum is N inside the circle, N/2 outside everywhere*
  - Circle can be of arbitrary diameter, at any location

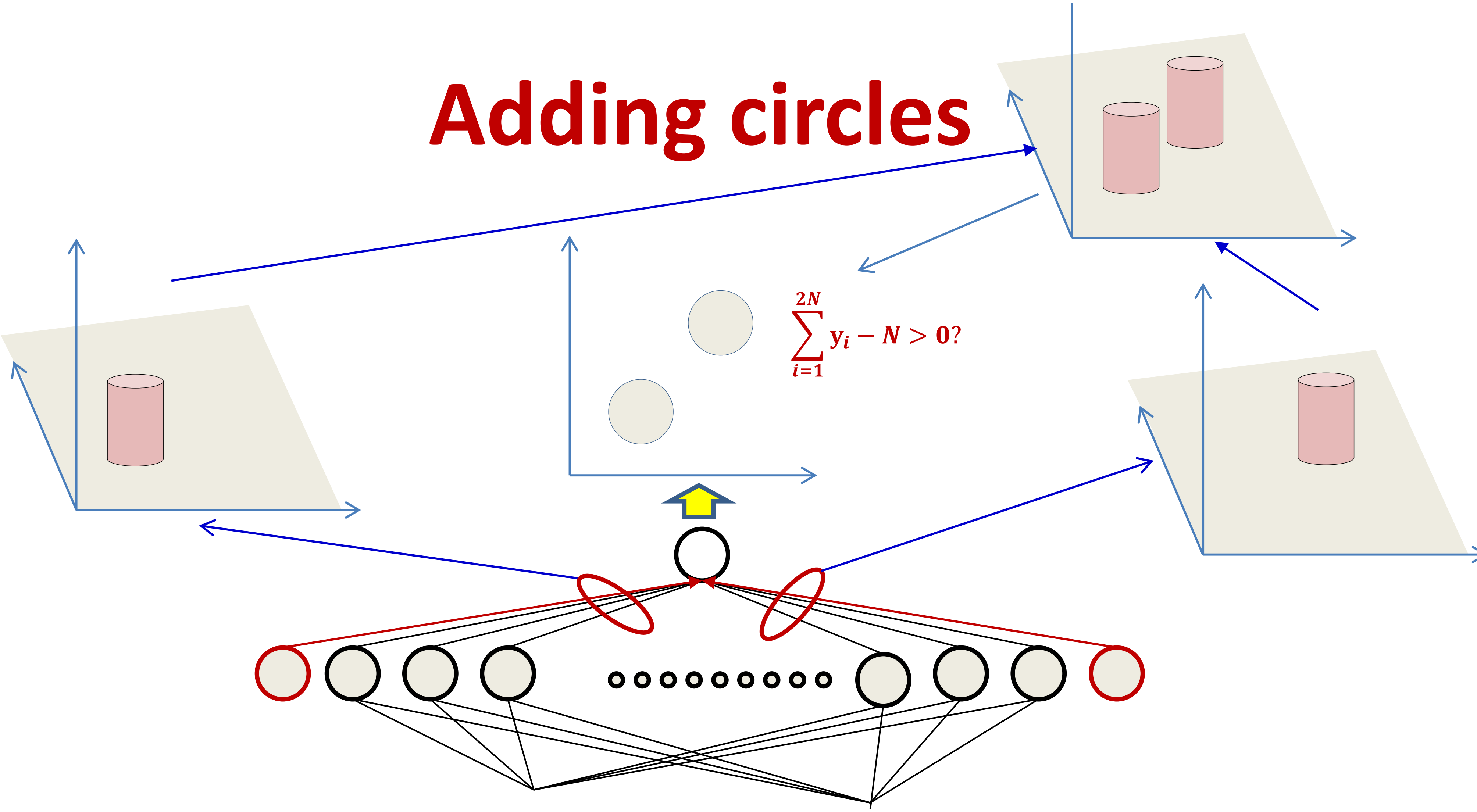
Composing a circle





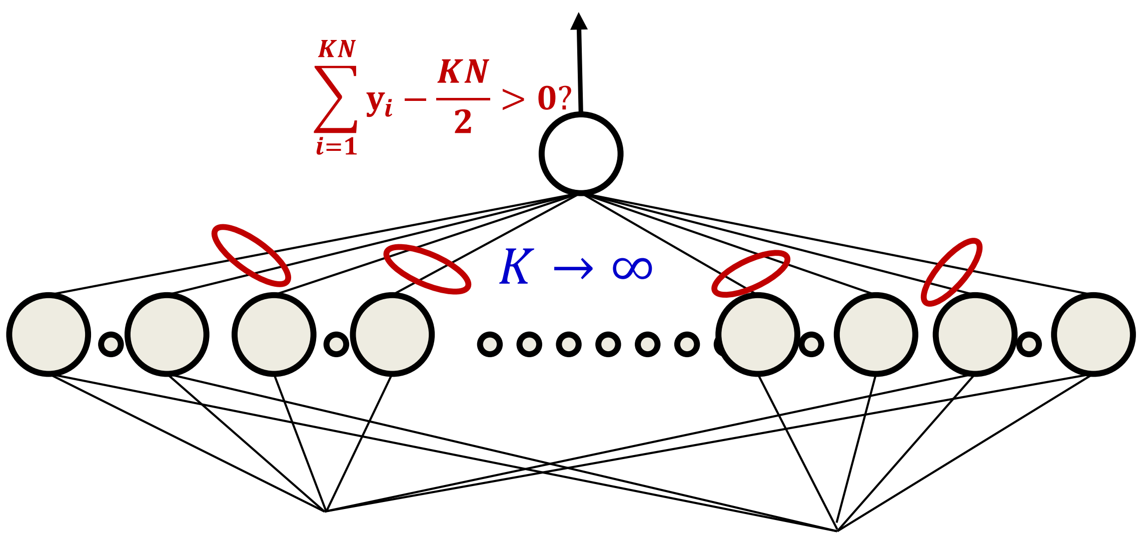
- The circle net
  - Very large number of neurons
  - *Sum is N/2 inside the circle, 0 outside everywhere*
  - Circle can be of arbitrary diameter, at any location

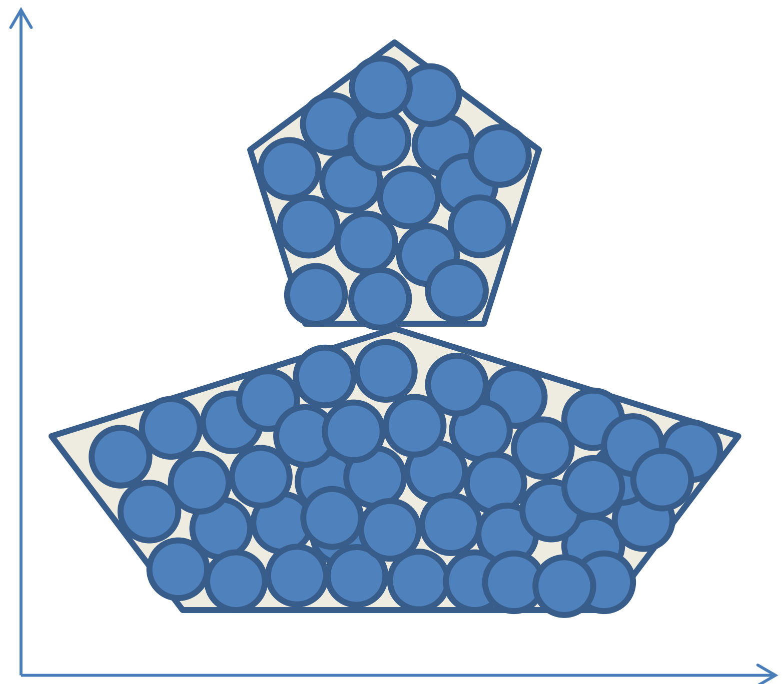
Adding circles



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

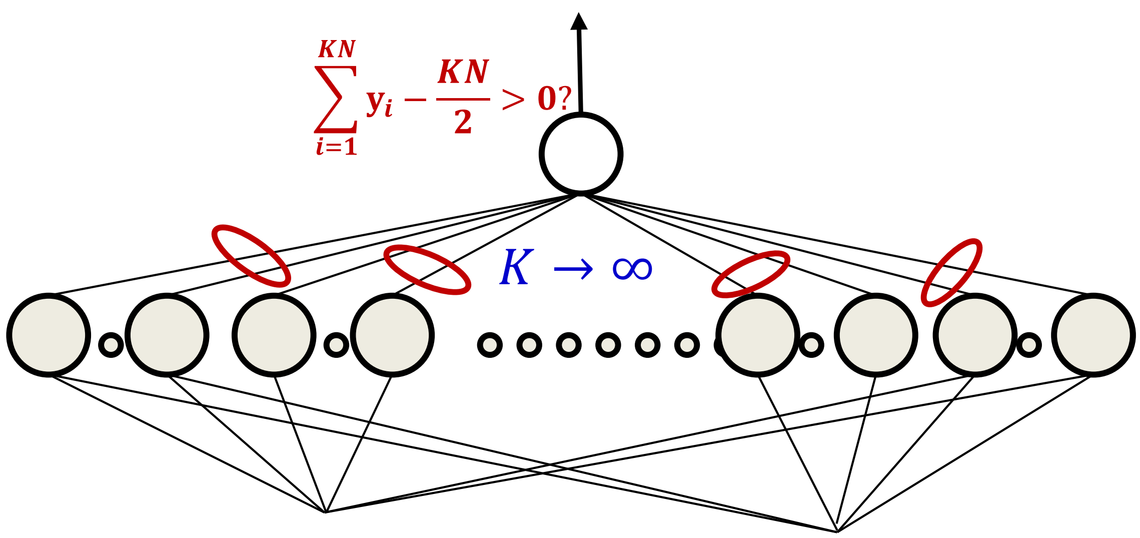
Composing an arbitrary figure

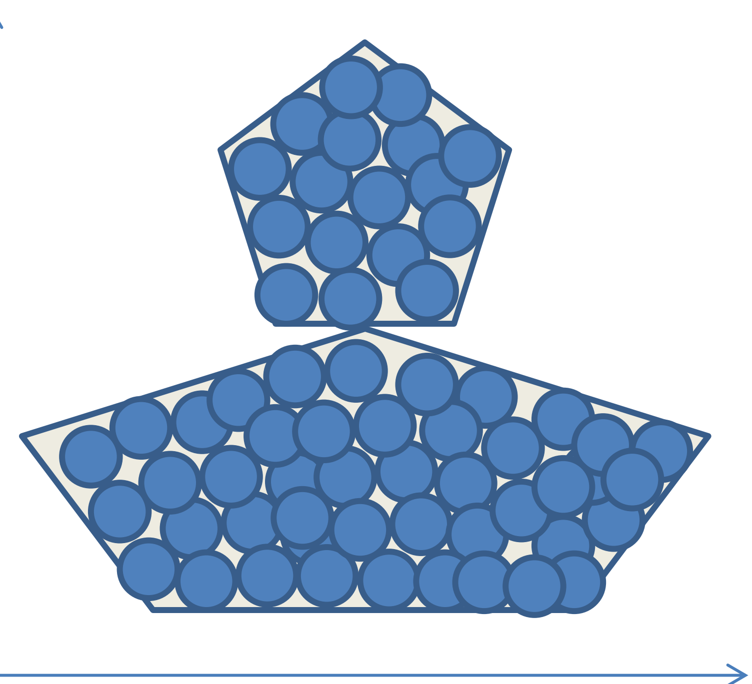




- Just fit in an arbitrary number of circles
  - More accurate approximation with greater number of smaller circles
  - Can achieve arbitrary precision

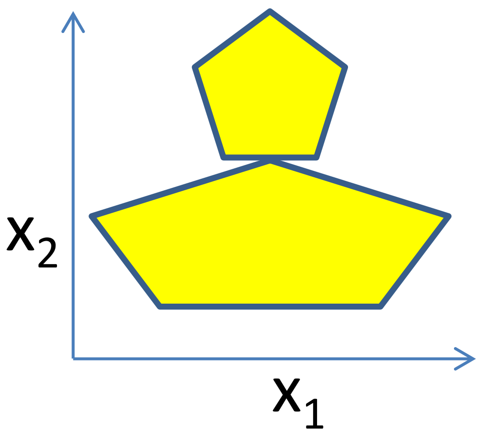
MLP: Universal classifier

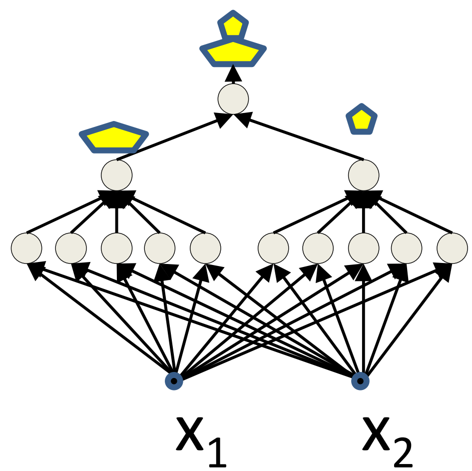


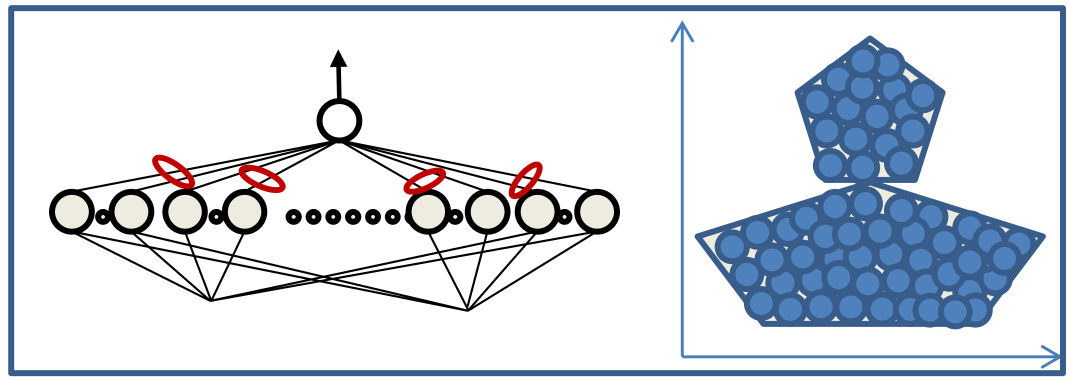


- MLPs can capture *any* classification boundary
- A *one-layer MLP* can model any classification boundary
- *MLPs are universal classifiers*

Depth and the universal classifier







- Deeper networks can require far fewer neurons

Optimal depth..

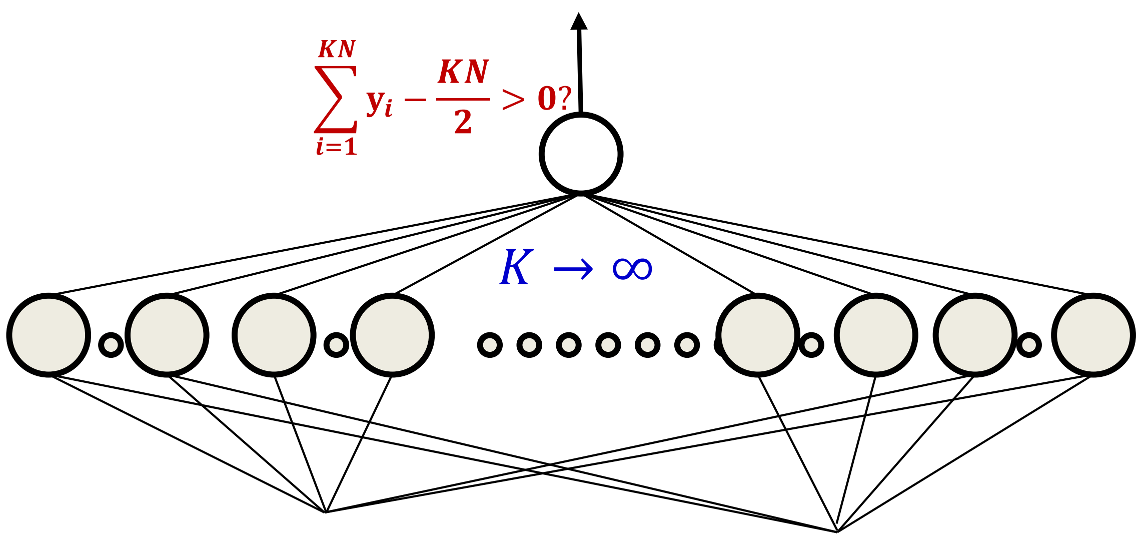
- Formal analyses typically view these as a category of *arithmetic circuits*
  - Compute polynomials over any field
    - Valiant et. al: A polynomial of degree n requires a network of depth $log^2(n)$
      - Cannot be computed with shallower networks
      - Nearly all functions are very high or even infinite-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

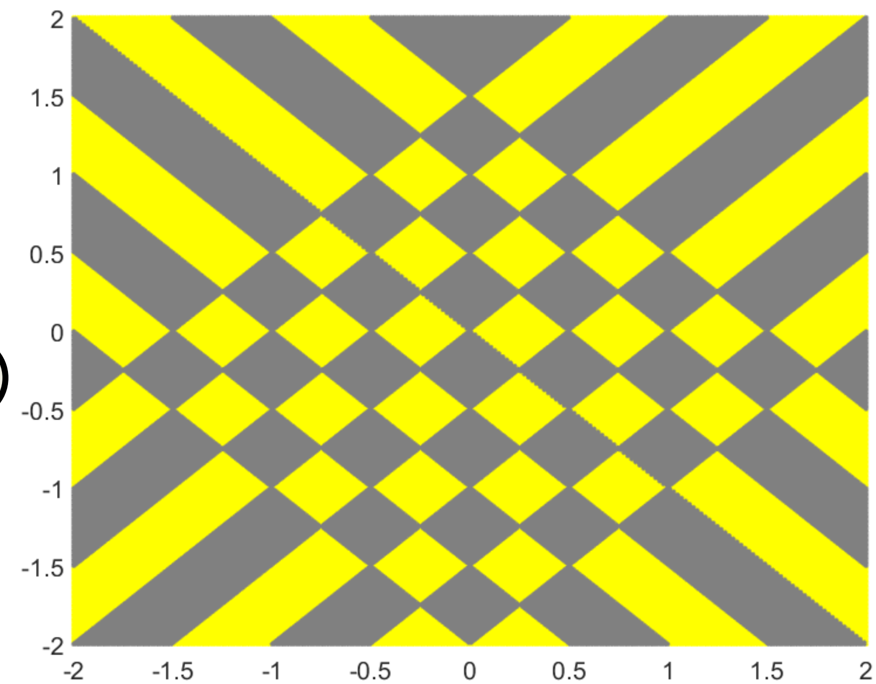
Optimal depth in *generic* nets

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

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

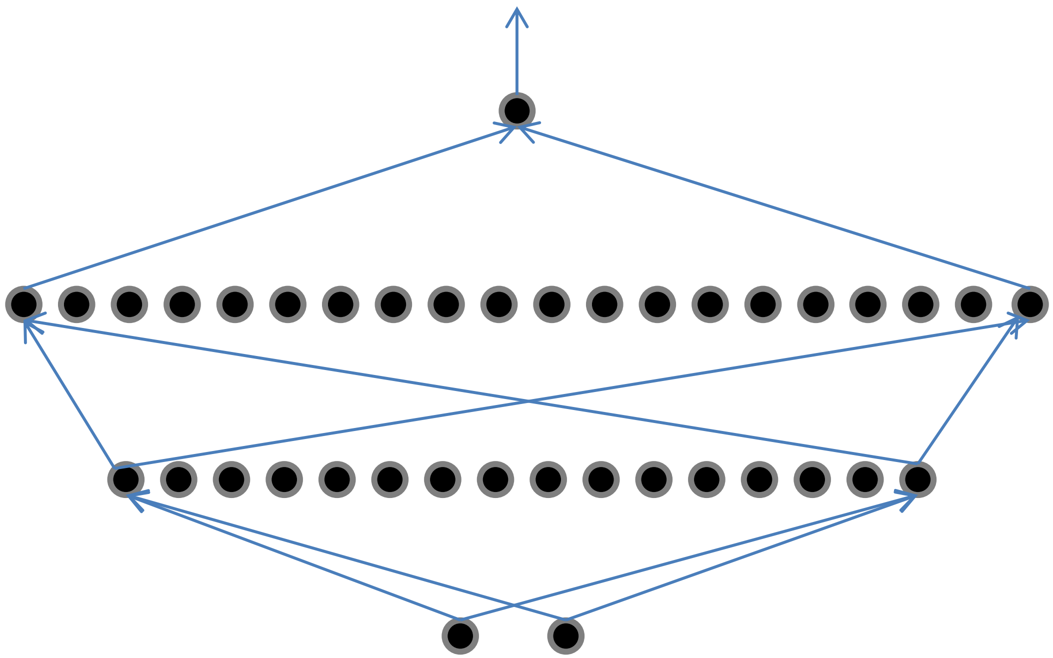
Optimal depth

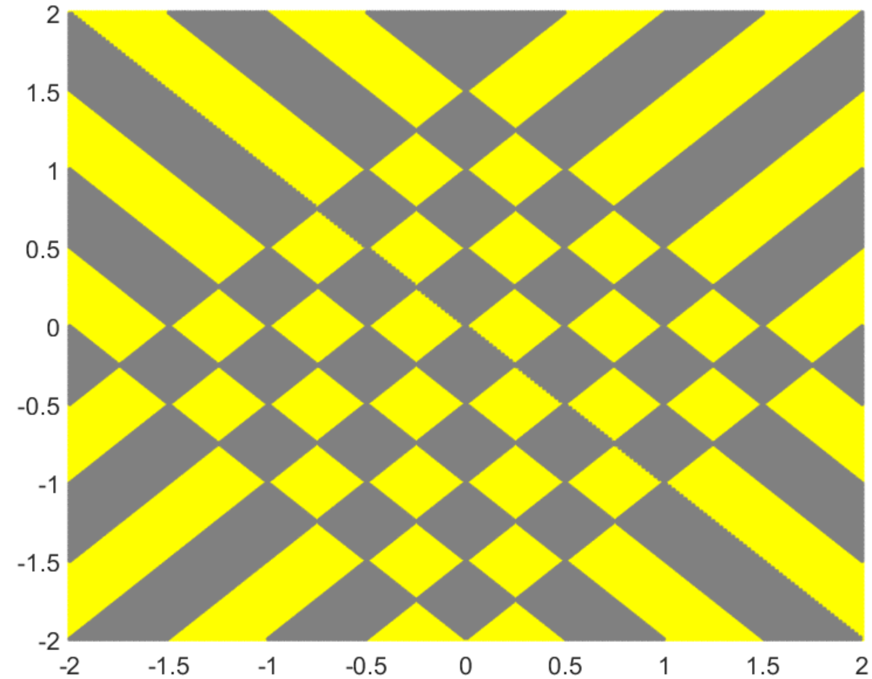




- A one-hidden-layer neural network will required infinite hidden neurons

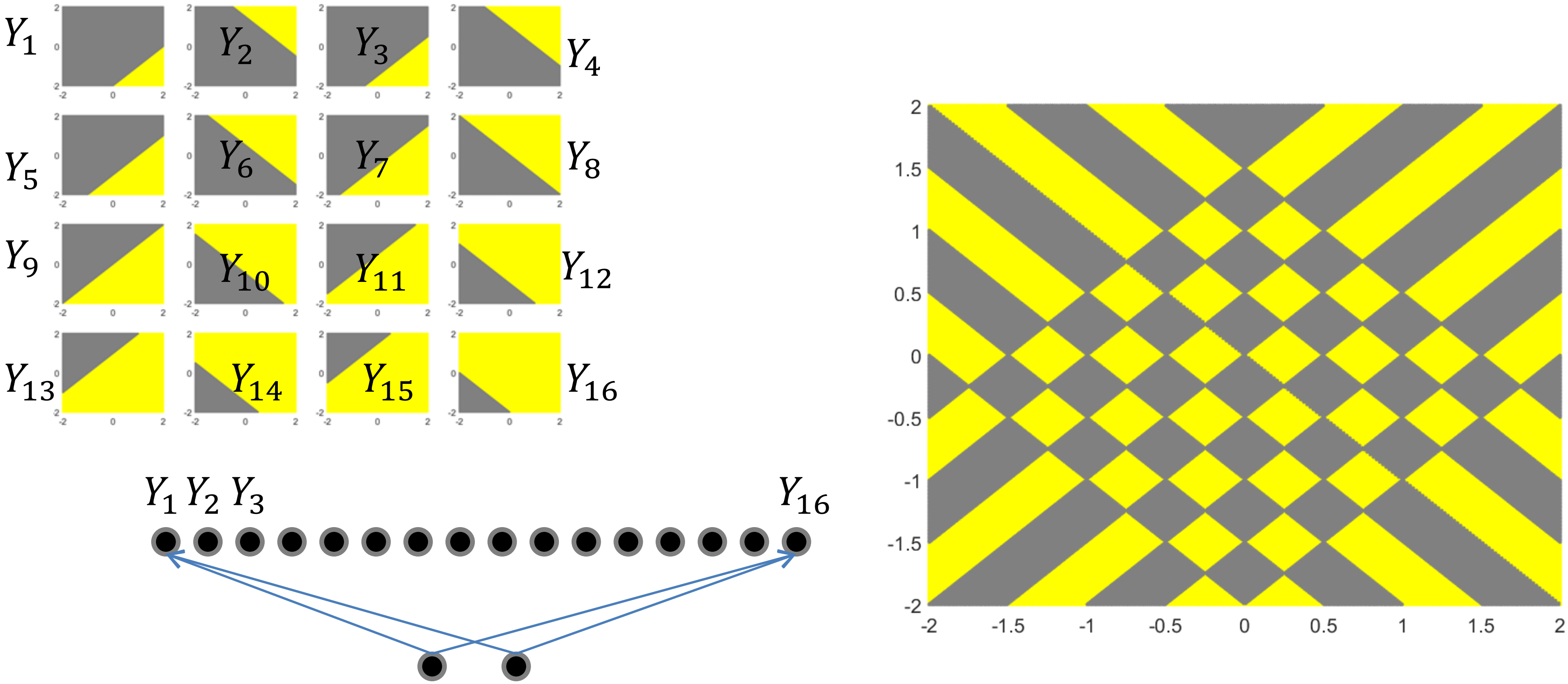
Optimal depth





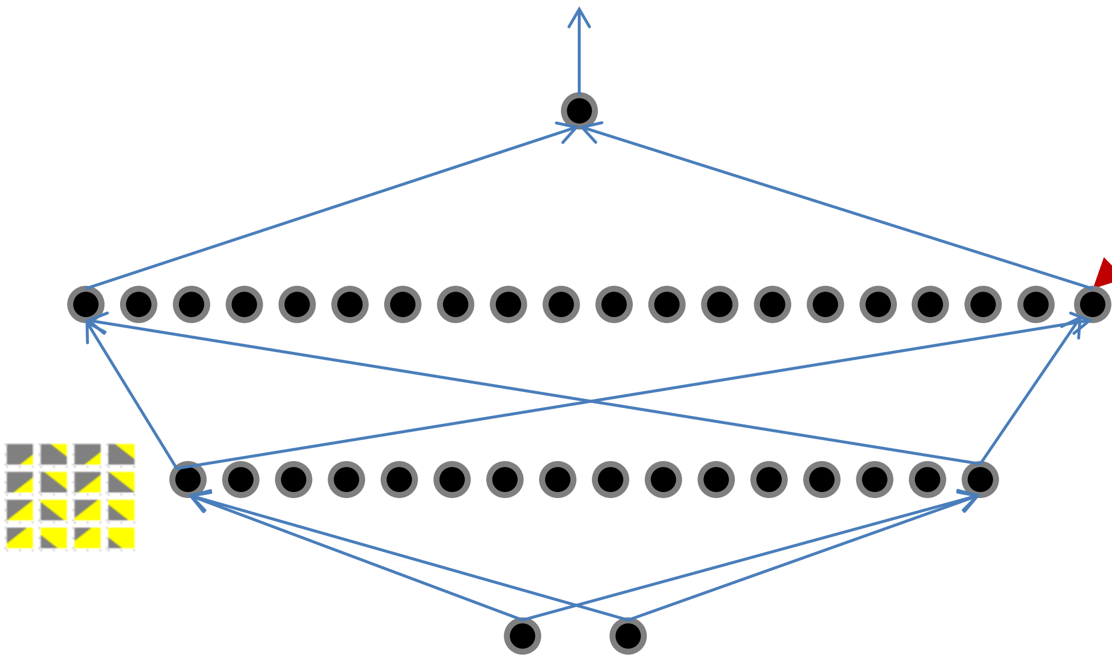
- Two layer network: 56 hidden neurons

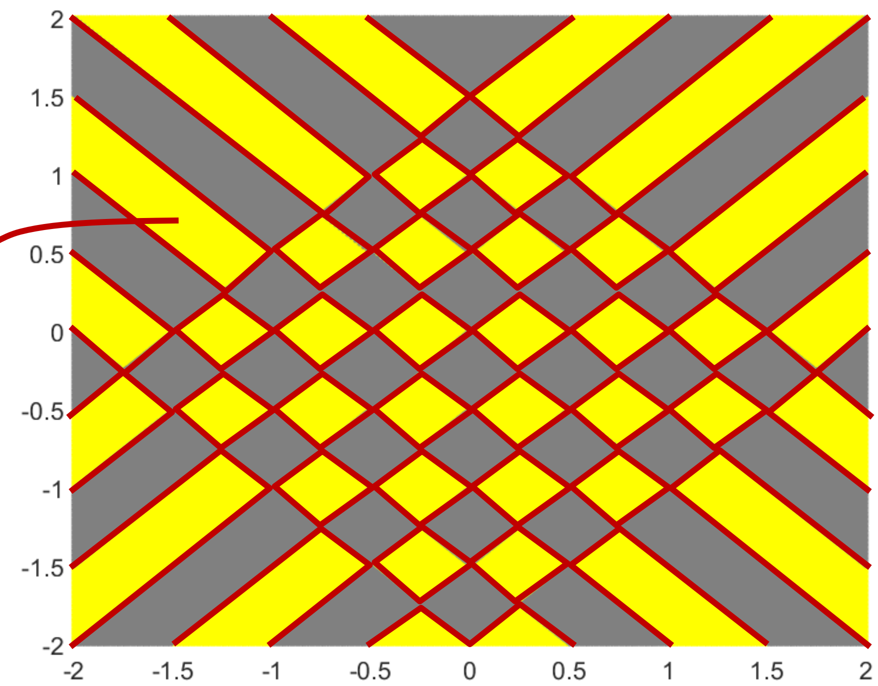
Optimal depth



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

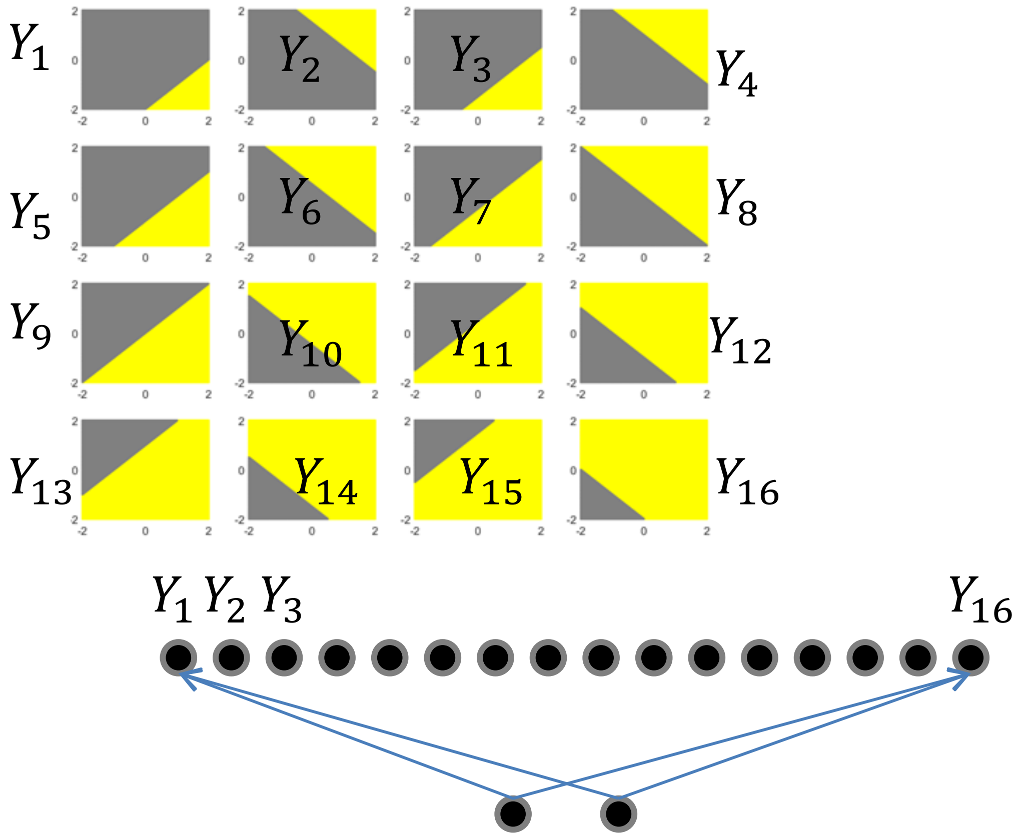
Optimal depth

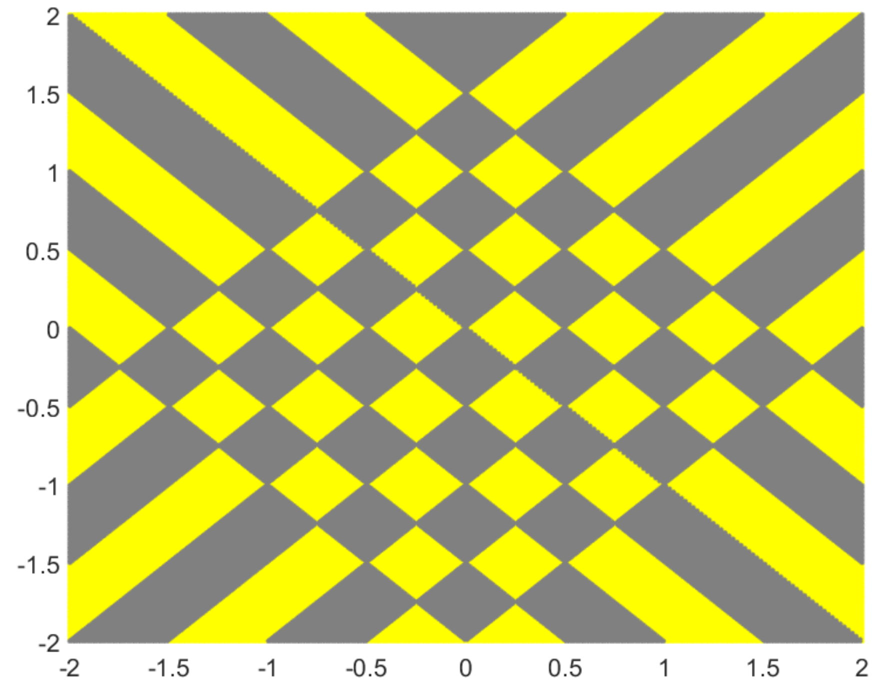




- Two-layer network: 56 hidden neurons
  - 16 in hidden layer 1
  - 40 in hidden layer 2
  - 57 total neurons, including output neuron

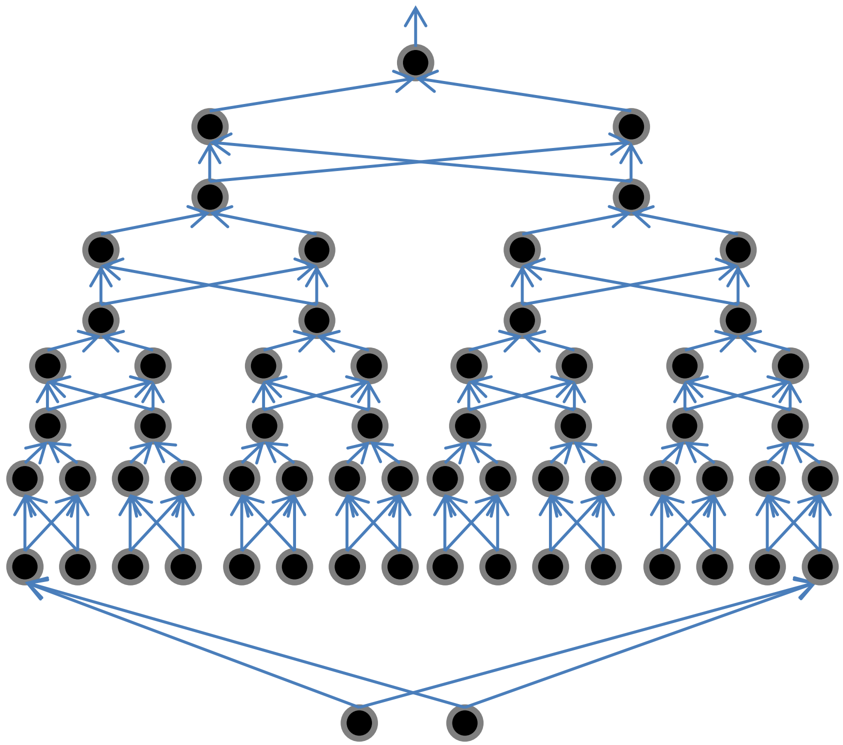
Optimal depth

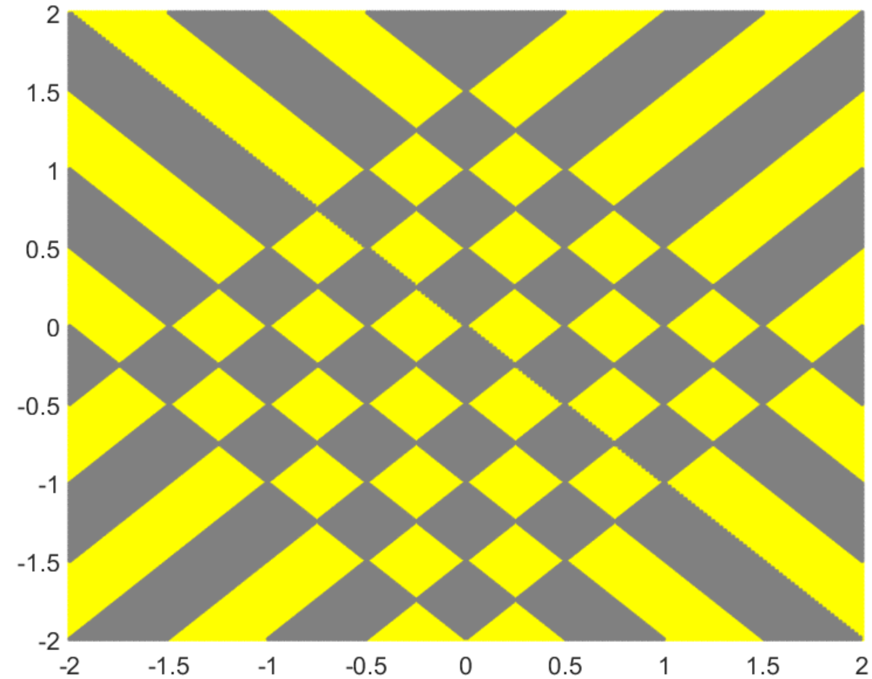




- But this is just $Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{16}$

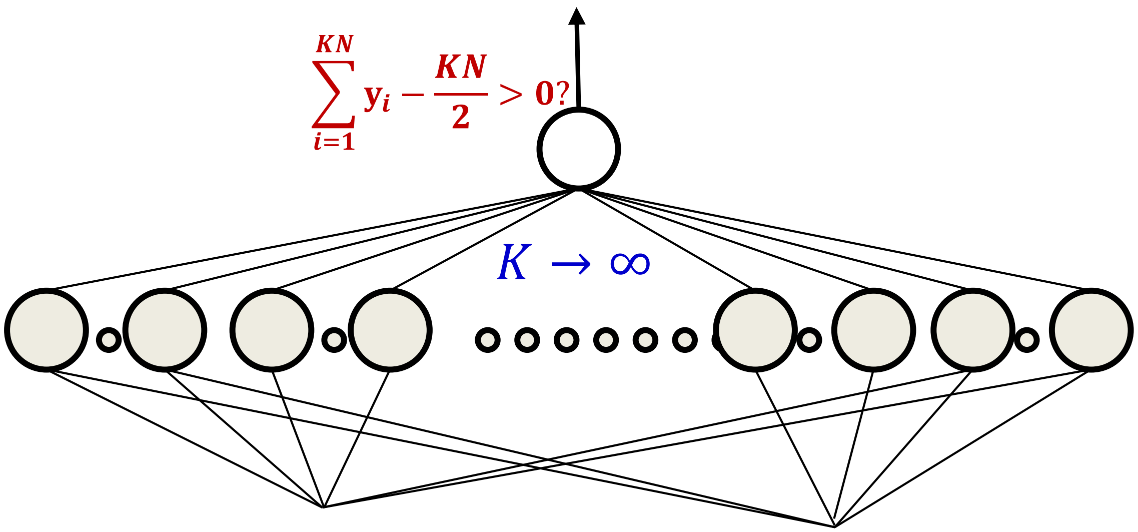
Optimal depth

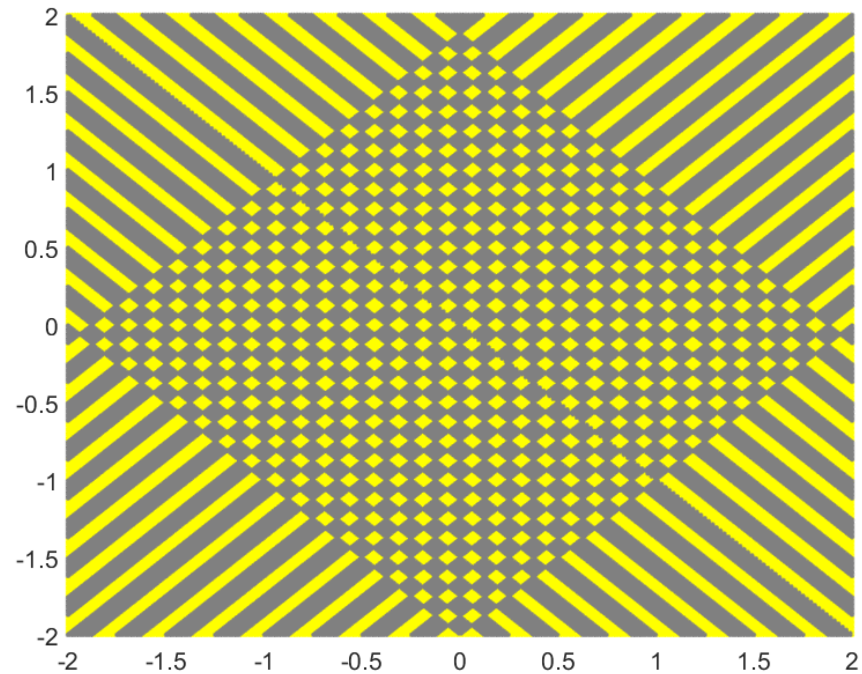




- But this is just $Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{16}$
  - The XOR net will require 16 + 15x3 = 61 neurons
    - Greater than the 2-layer network with only 52 neurons

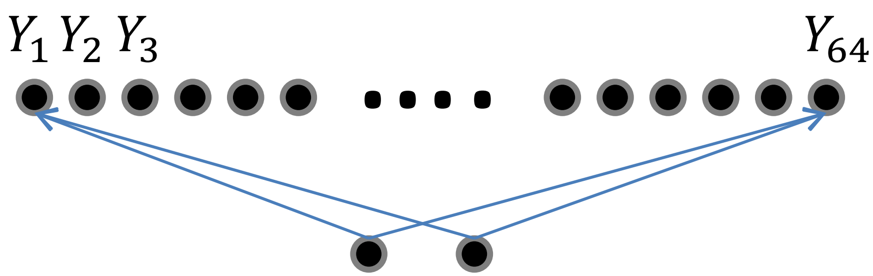
Optimal depth

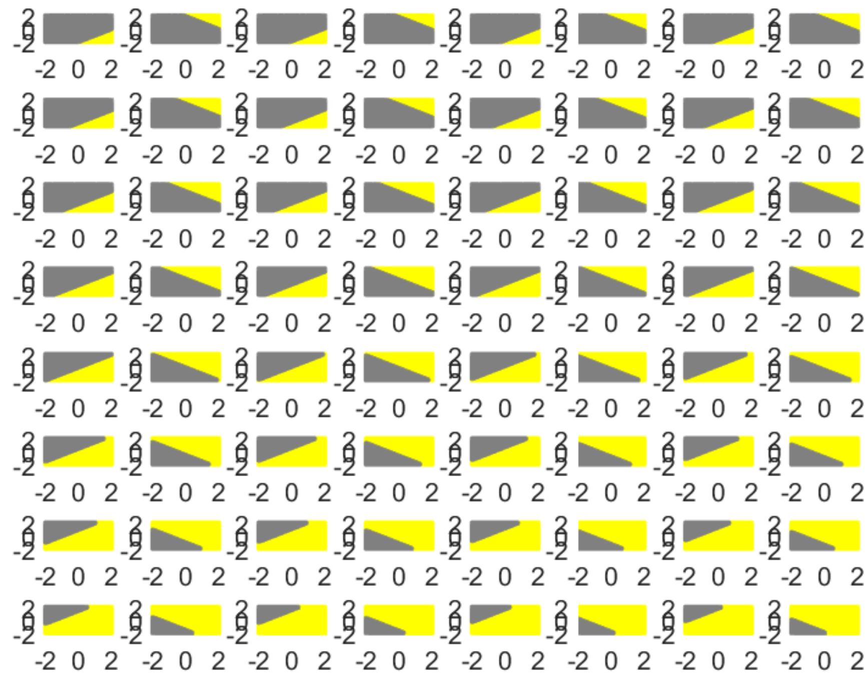




- A one-hidden-layer neural network will required infinite hidden neurons

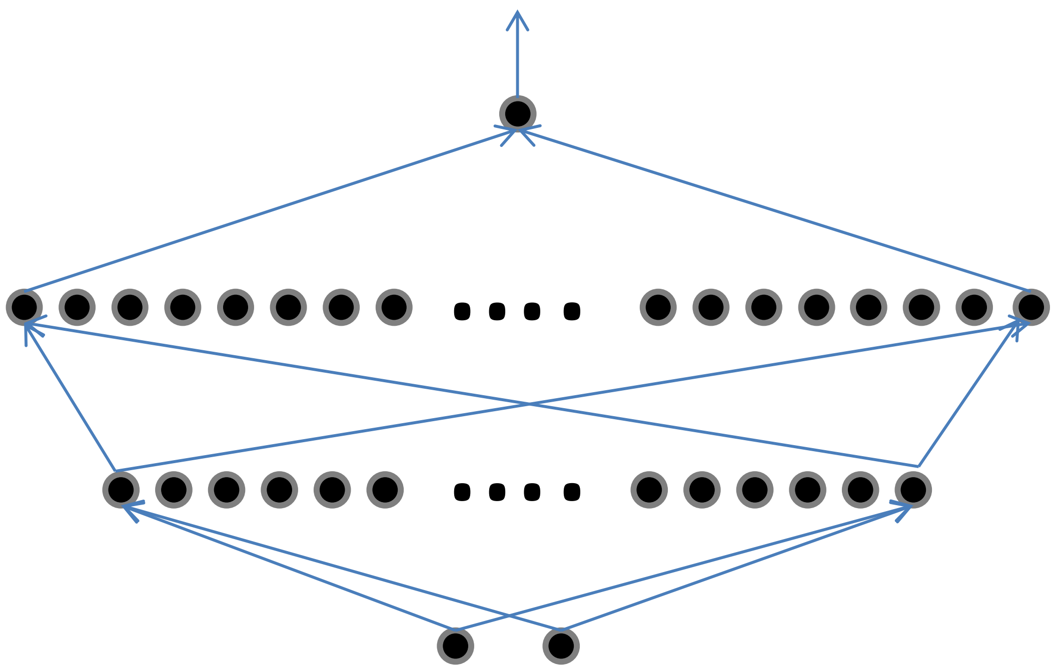
Actual linear units

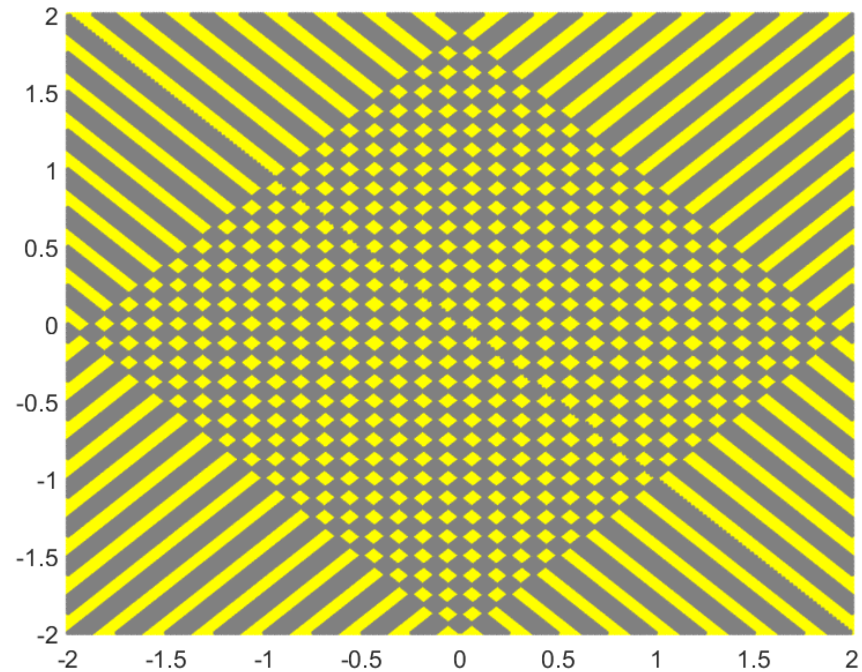




- 64 basic linear feature detectors

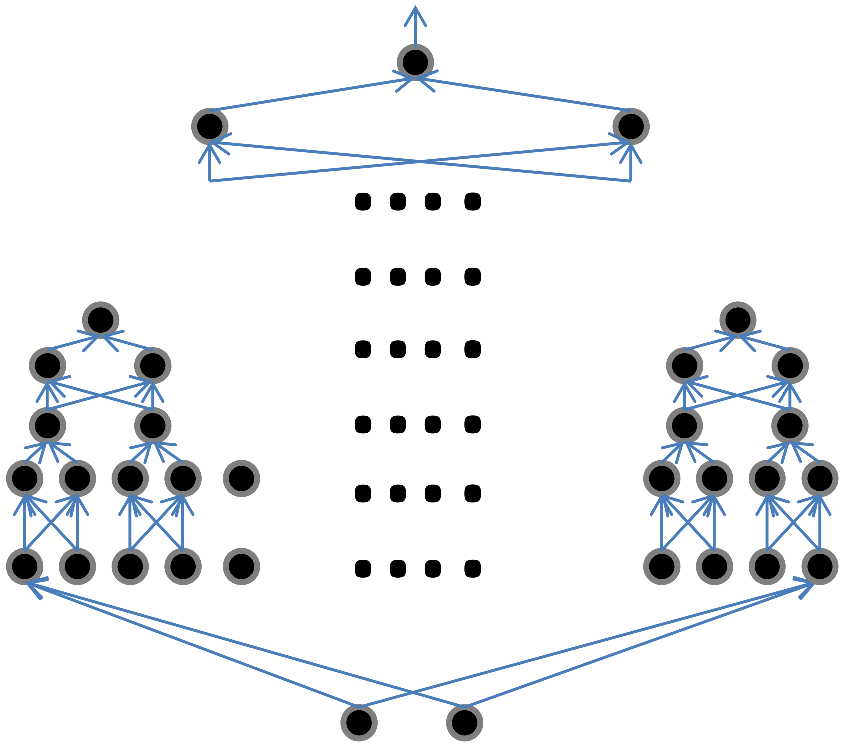
Optimal depth

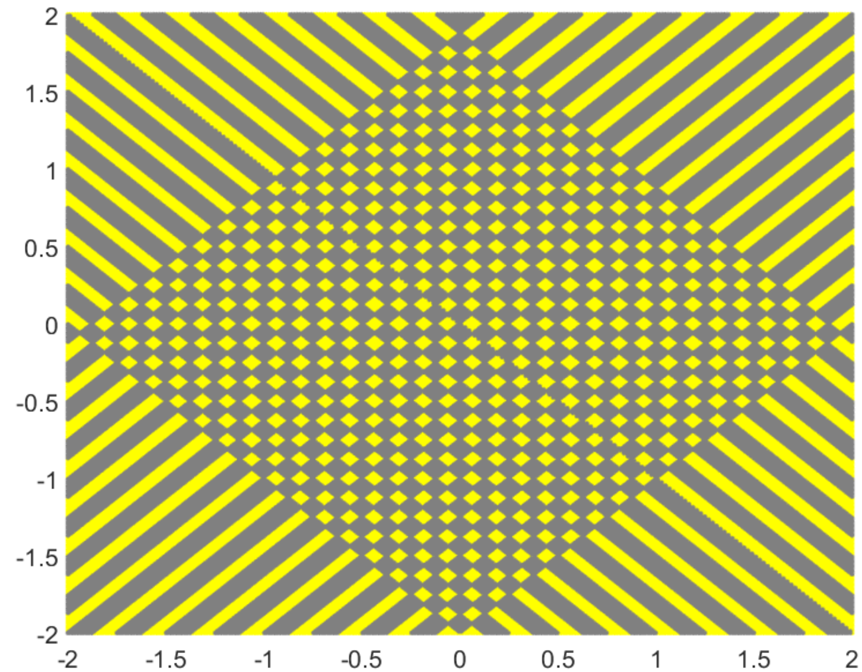




- Two hidden layers: 608 hidden neurons
  - 64 in layer 1
  - 544 in layer 2
- 609 total neurons (including output neuron)

Optimal depth

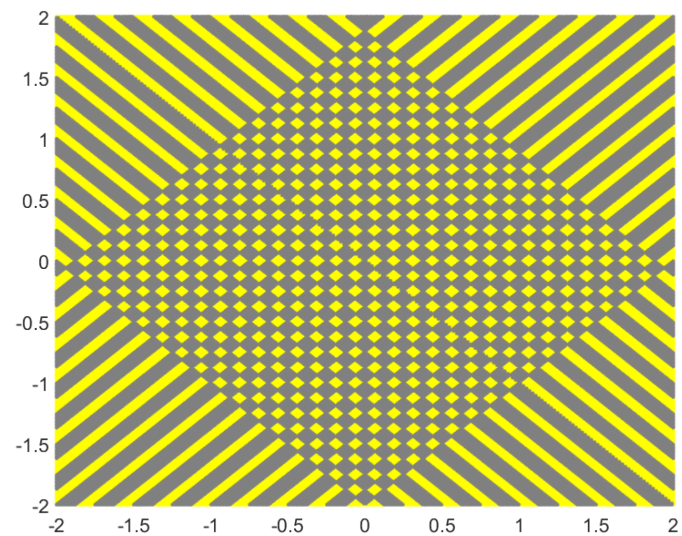




- XOR network (12 hidden layers): 253 neurons
- The difference in size between the deeper optimal (XOR) net and shallower nets increases with increasing pattern complexity

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)

Depth: Summary

- The number of neurons required in a shallow network is
  - Polynomial in the number of basic patterns
  - Exponential in the dimensionality of the input
  - (this is the worst case)

Story so far

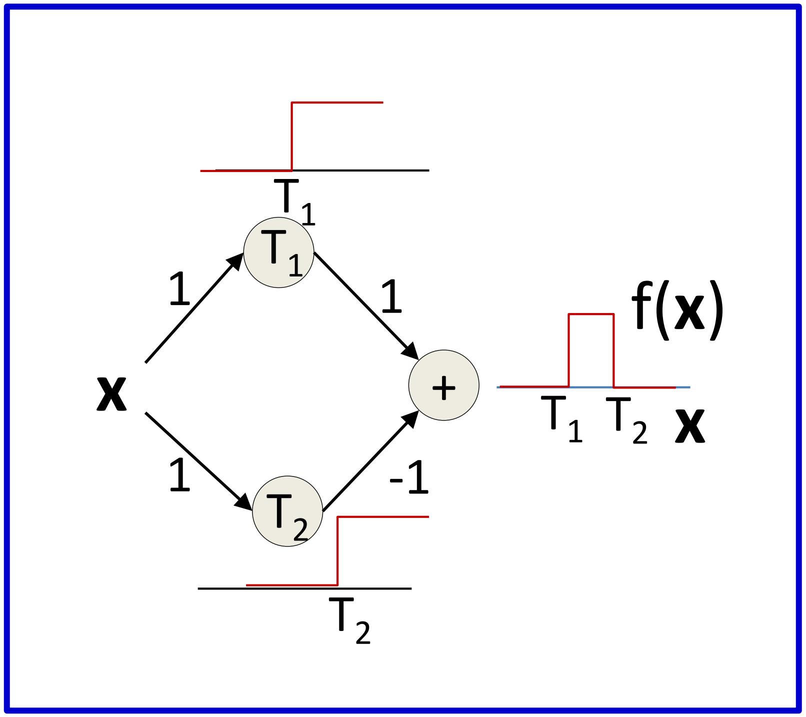
- Multi-layer perceptrons are *Universal Boolean Machines*
  – Even a network with a *single* hidden layer is a universal Boolean machine

- 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

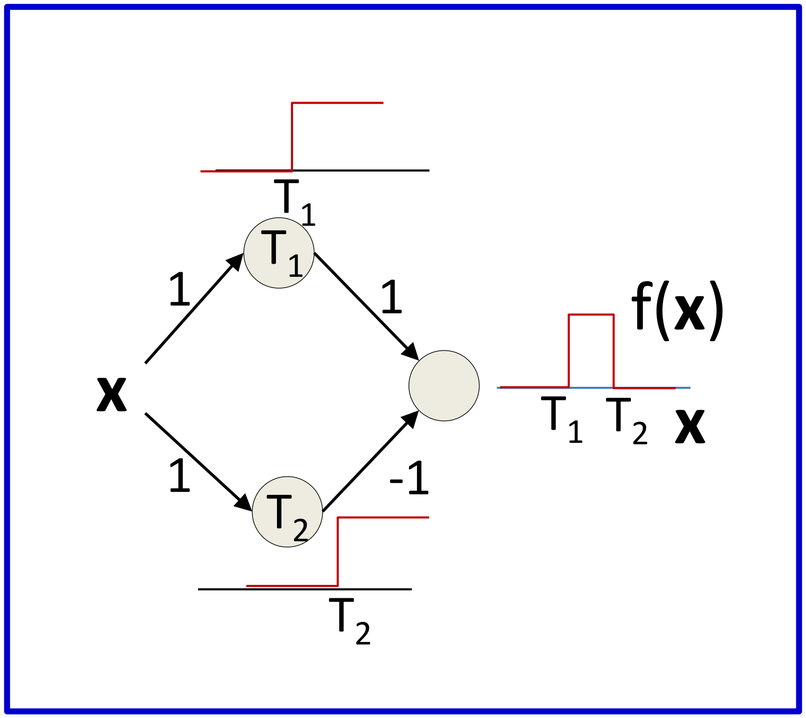
- Deeper networks may require exponentially fewer neurons than shallower networks to express the same function
  – Could be *exponentially* smaller
  – Deeper networks are more *expressive*

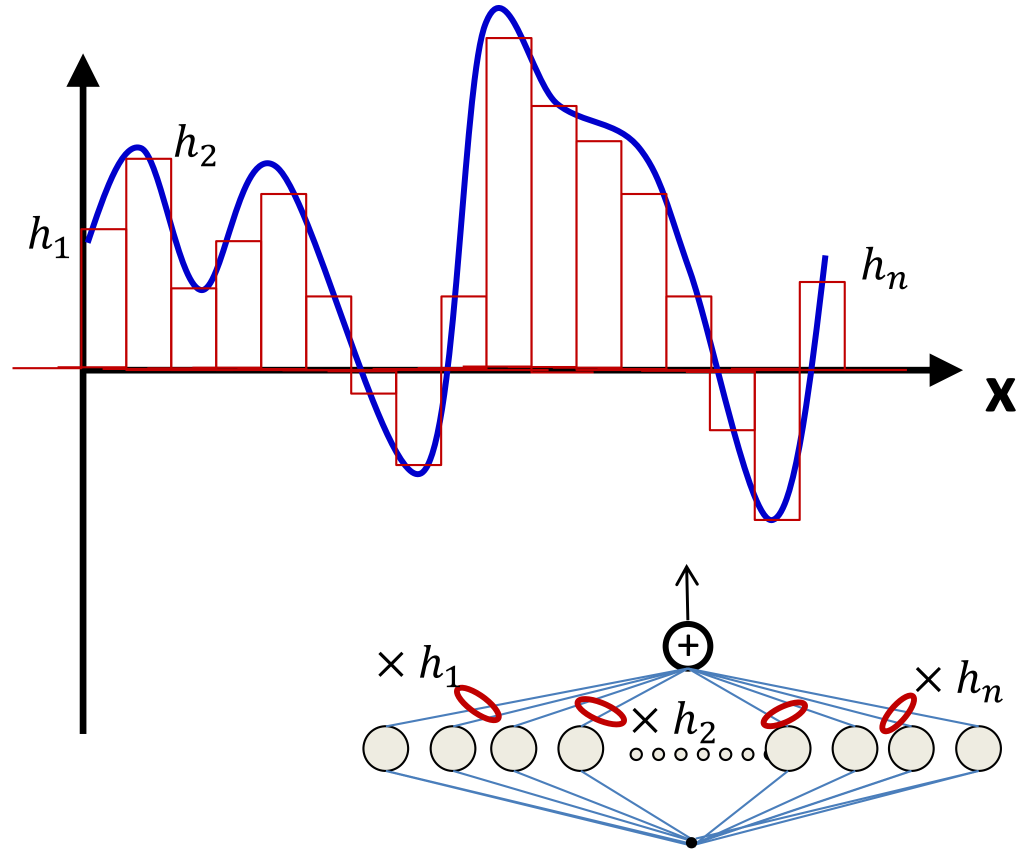
MLP as a continuous-valued regression



- 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

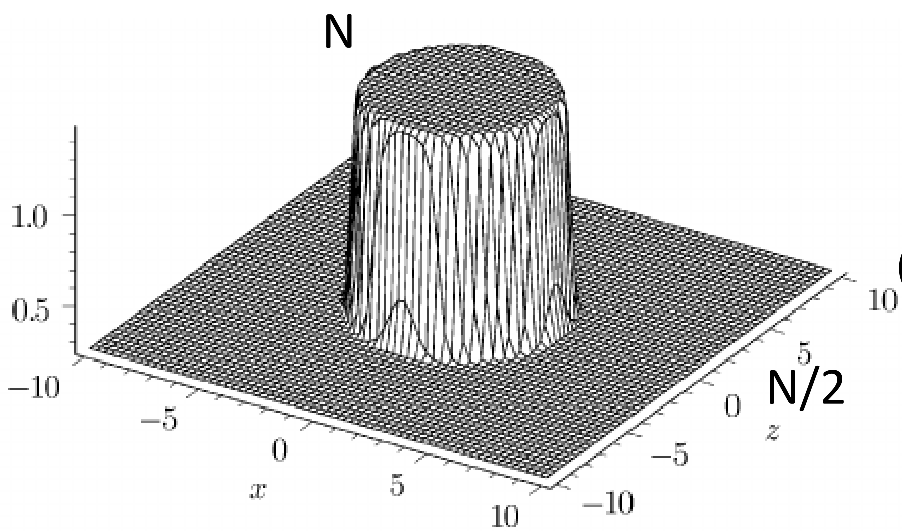
MLP as a continuous-valued regression

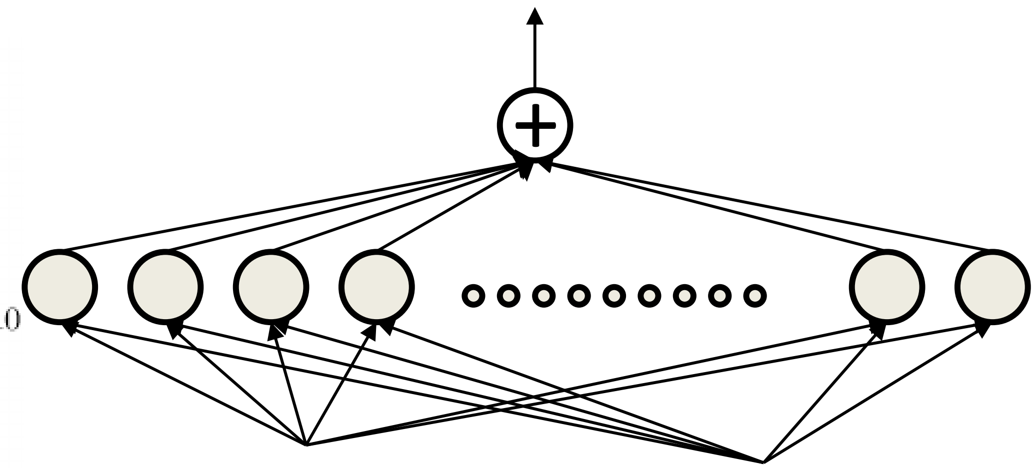




- A simple 3-unit MLP can generate a "square pulse" over an input
- **An MLP with many units can model an arbitrary function over an input**
  - To arbitrary precision
    - Simply make the individual pulses narrower
- ***A one-layer MLP can model an arbitrary function of a single input***

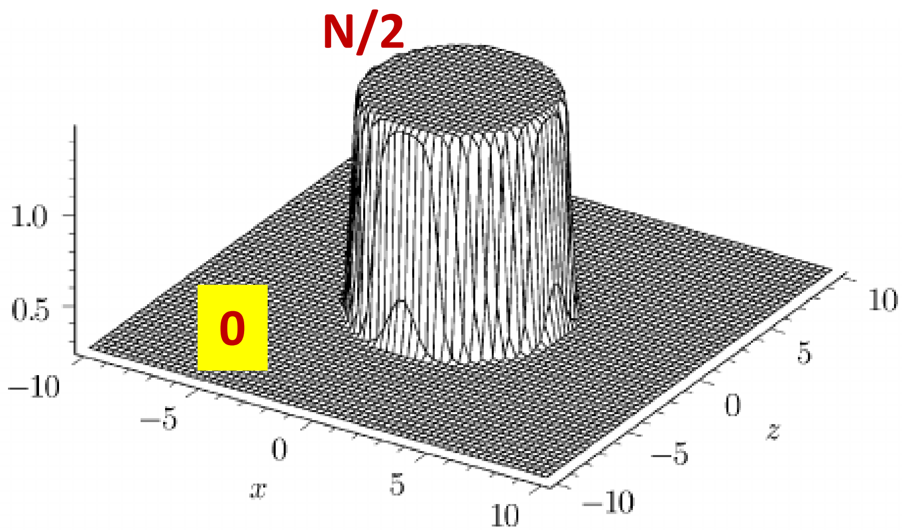
For higher-dimensional functions

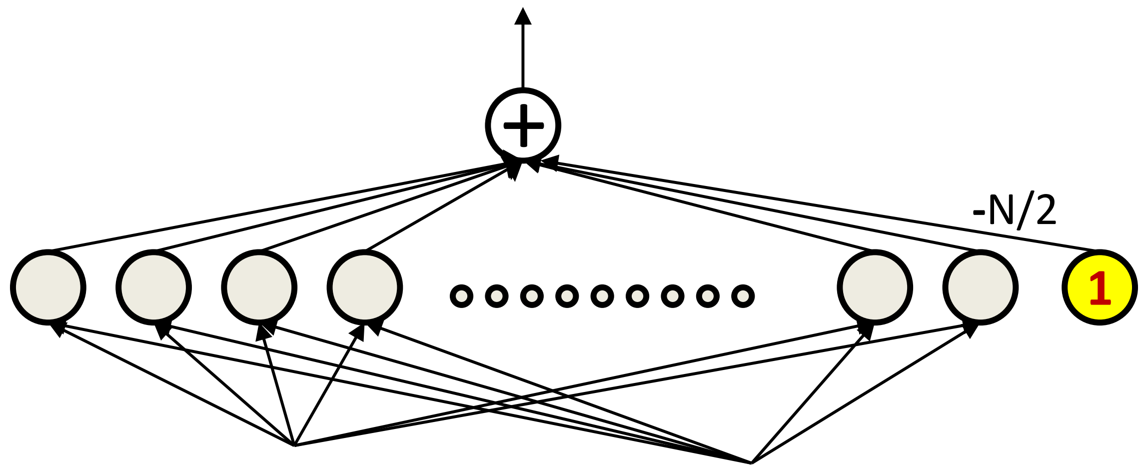




- An MLP can compose a cylinder
  - N in the circle, N/2 outside

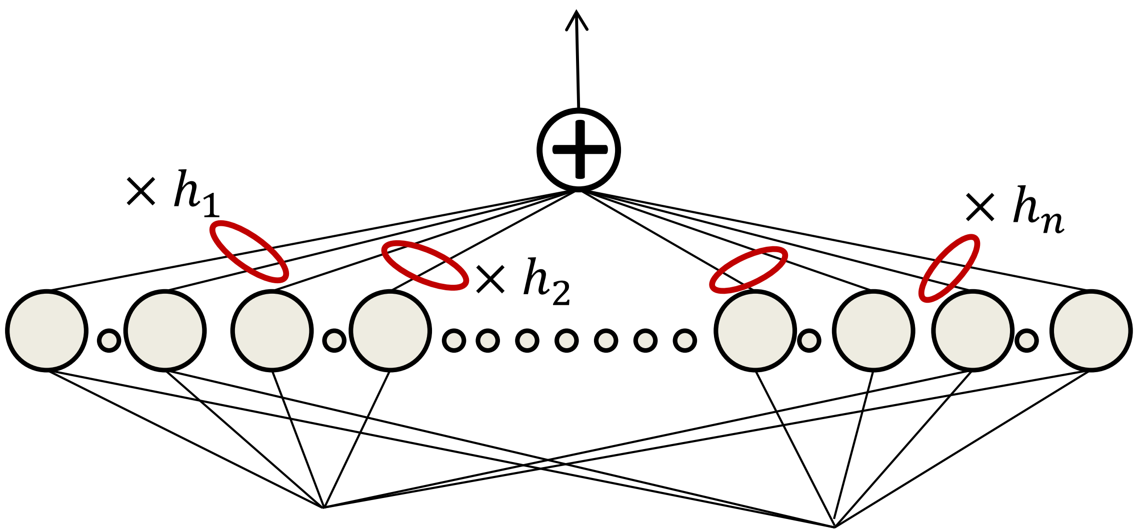
A "true" cylinder

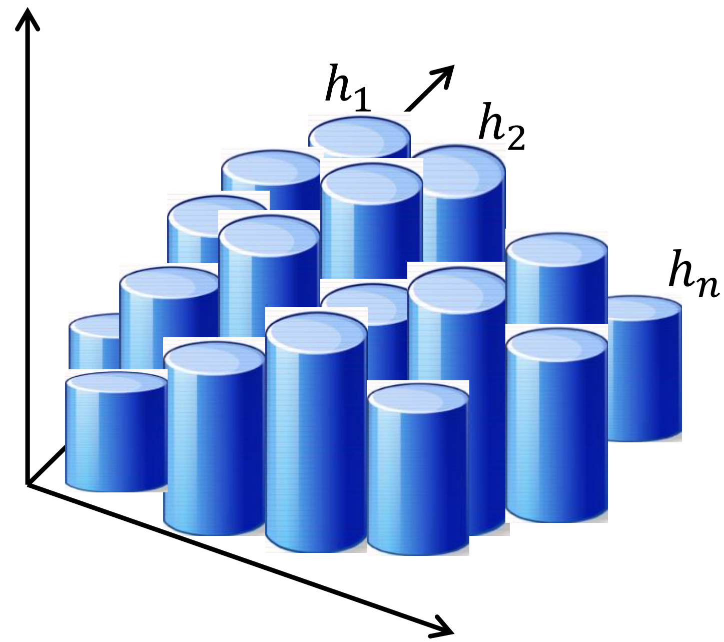




- An MLP can compose a *true* (almost) cylinder
  - N/2 in the circle, 0 outside
  - By adding a "bias"
  - We will encounter bias terms again
    - They are standard components of perceptrons

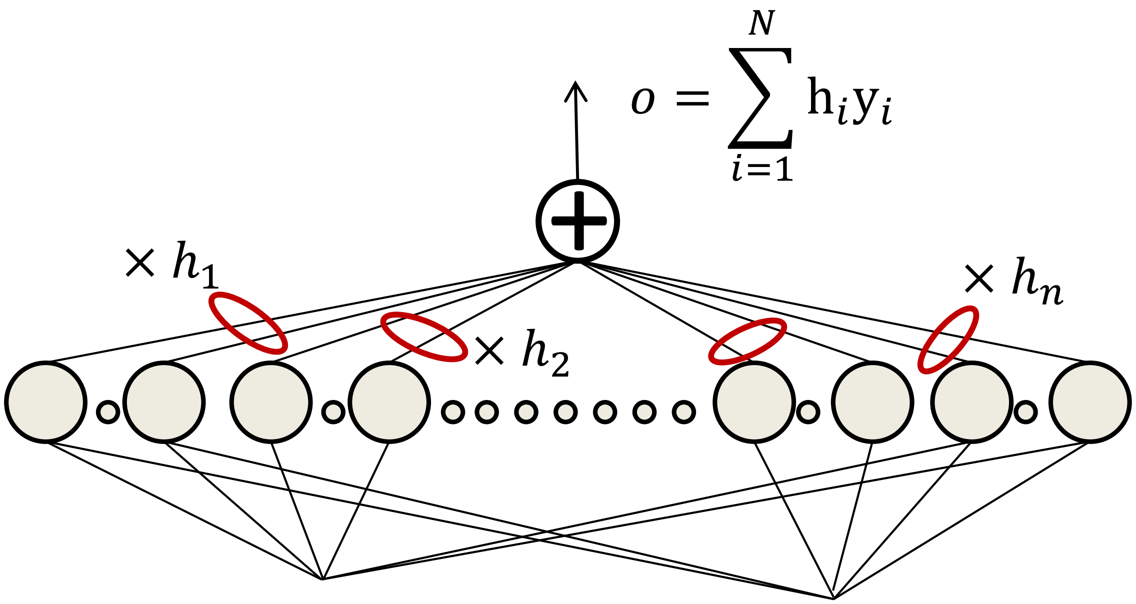
MLP as a continuous-valued function

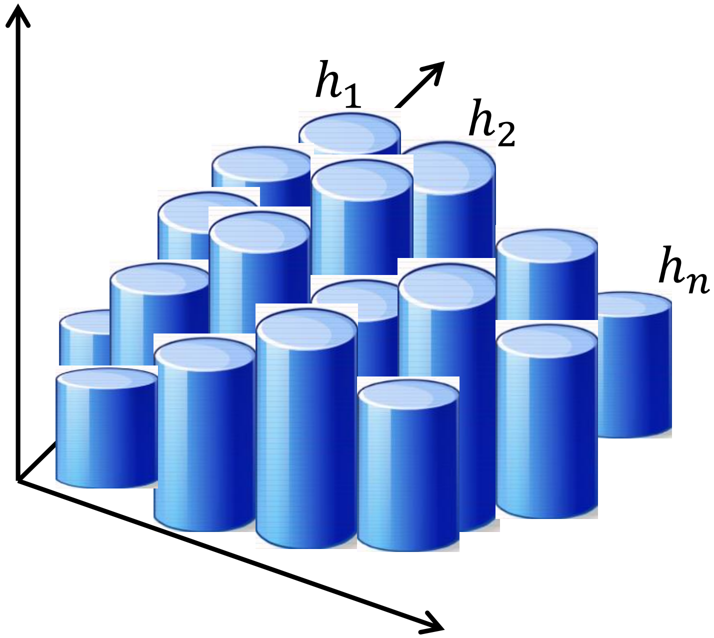




- MLPs can actually compose arbitrary functions
  - Even with only one layer
    - As sums of scaled and shifted cylinders
  - To arbitrary precision
    - By making the cylinders thinner
  - **The MLP is a universal approximator!**

Caution: MLPs with additive output units are universal approximators





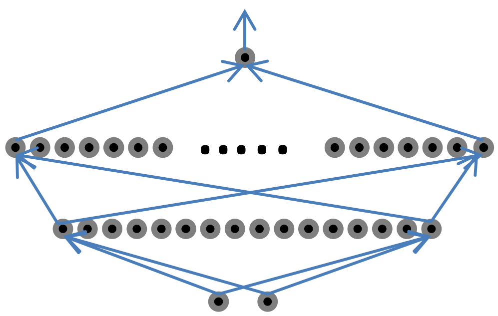
- MLPs can actually compose arbitrary functions
- But explanation so far only holds if the output unit only performs summation
  - i.e. does not have an additional "activation"

The issue of depth

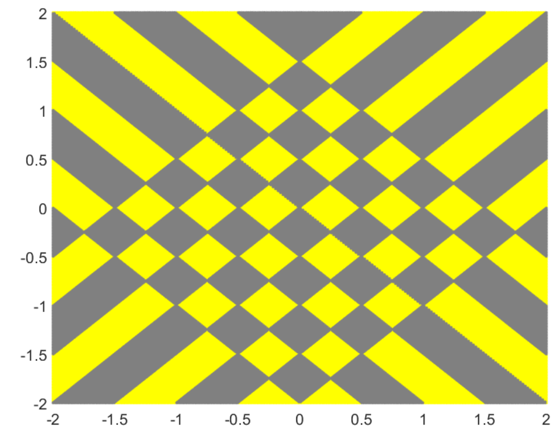
- Previous discussion showed that a *single-layer* MLP is a universal function approximator
  - Can approximate any function to arbitrary precision
  - But may require infinite neurons in the layer

- 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

Sufficiency of architecture

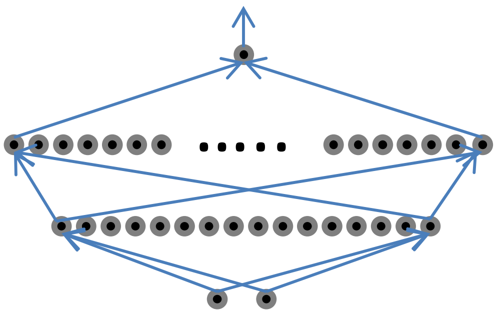


A network with 16 or more neurons in the first layer is capable of representing the figure to the right perfectly

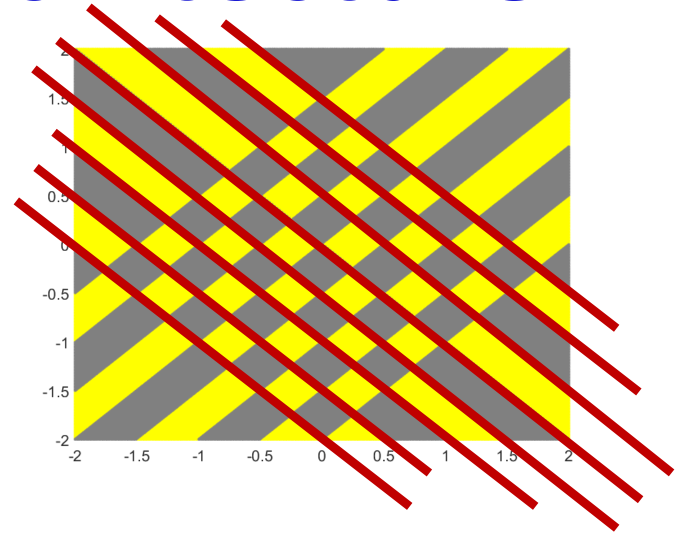


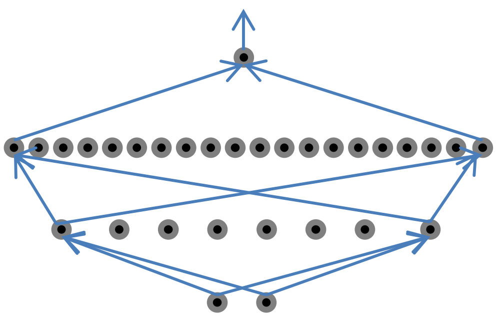
- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture



A network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly





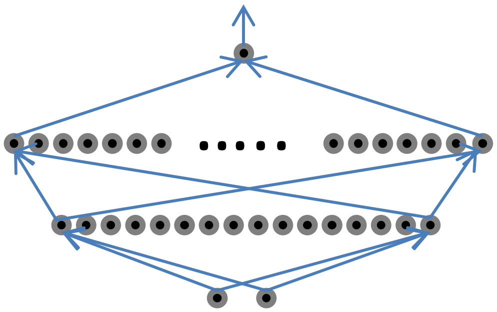
A network with less than 16 neurons in the first layer cannot represent this pattern exactly

- ❖ With caveats..

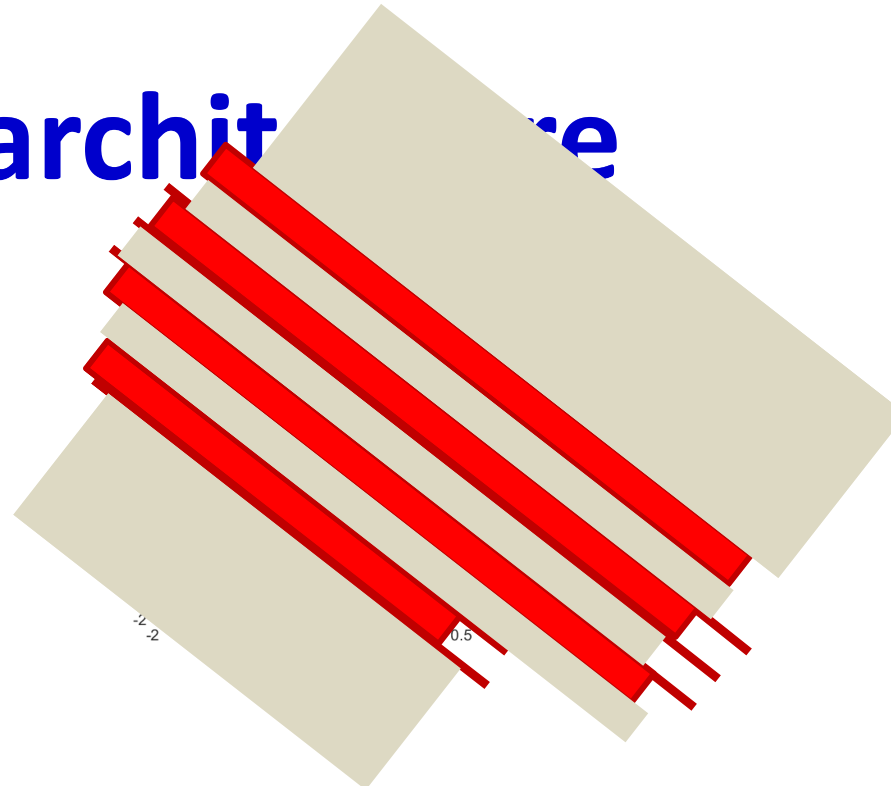
Why?

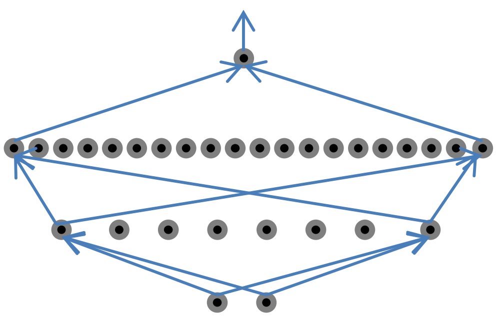
- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of archit re



A network with 16 or more neurons in the first layer is *capable of* representing the figure to the right perfectly





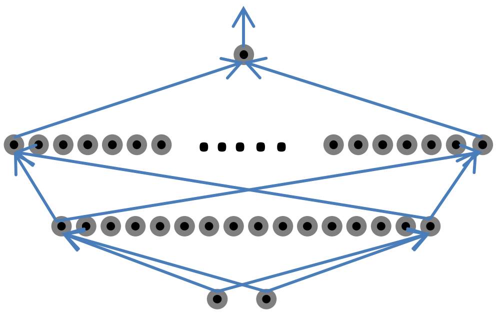
A network with less than 16 neurons in the first layer cannot represent this pattern exactly

❖ With caveats..

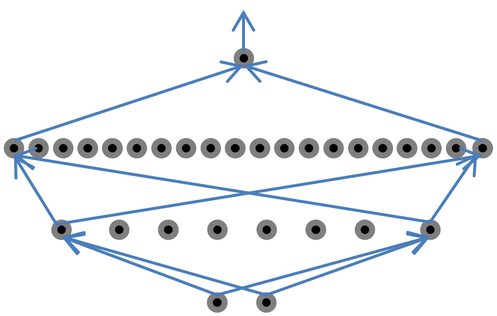
We will revisit this idea shortly

- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture

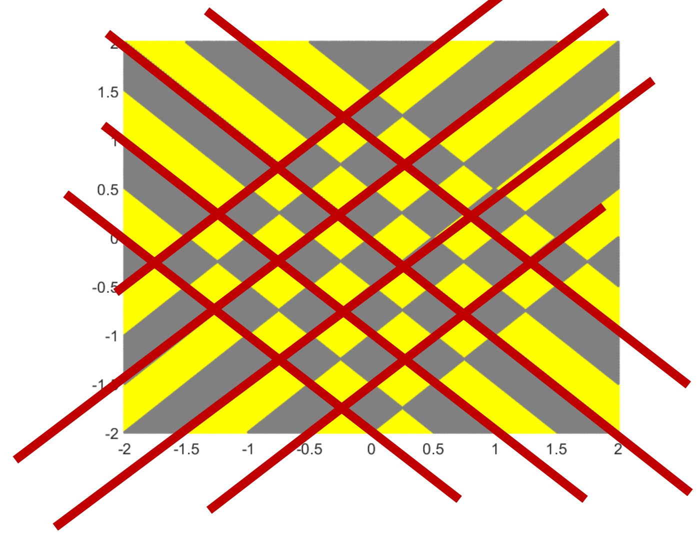


A network with 16 or more neurons in the first layer is capable of representing the figure to the right perfectly



A network with less than 16 neurons in the first layer cannot represent this pattern exactly

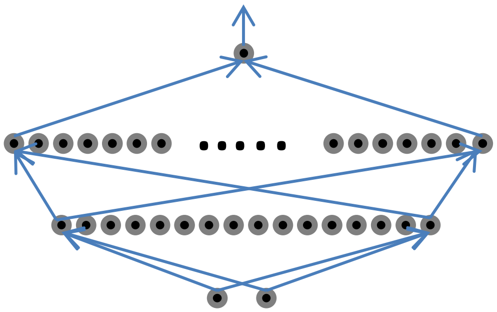
❖ With caveats..



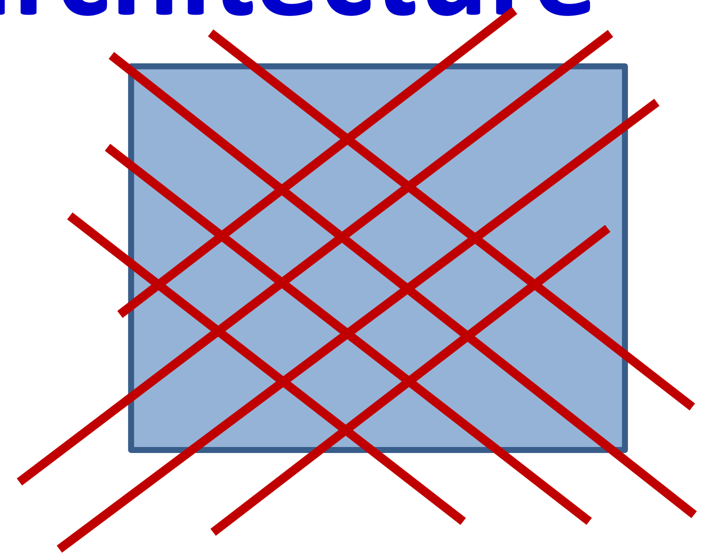
Why?

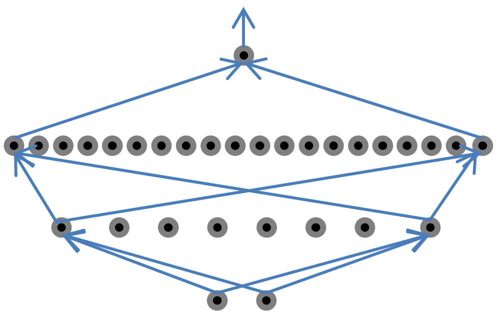
- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture



A network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly





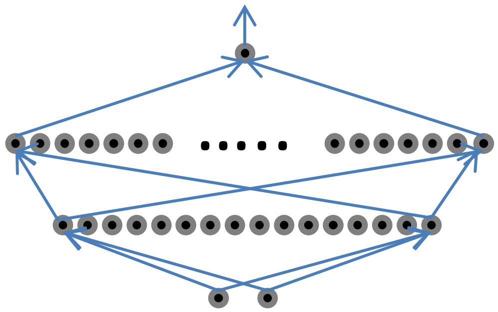
A network with less than 16 neurons in the first layer cannot represent this pattern exactly

- With caveats..

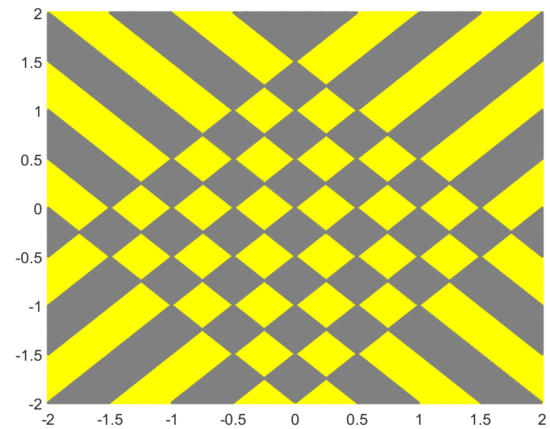
Why?

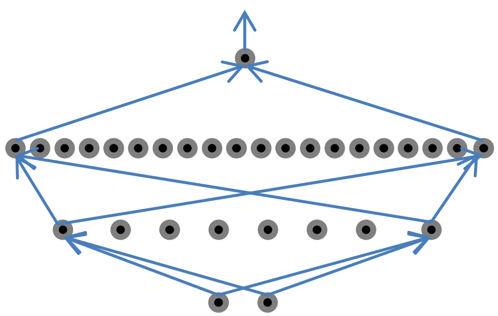
- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture



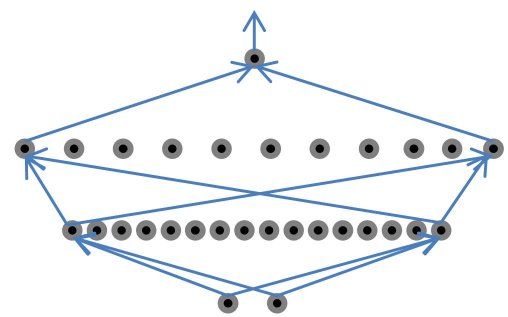
A network with 16 or more neurons in the first layer *is capable of* representing the figure to the right perfectly





A network with less than 16 neurons in the first layer cannot represent this pattern exactly

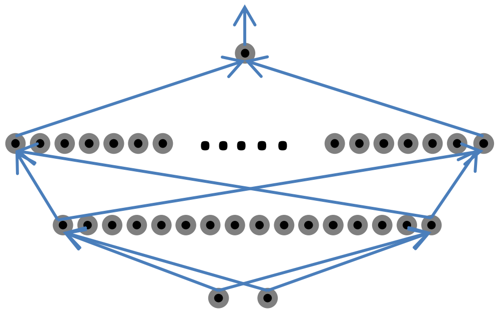
- ❖ With caveats..



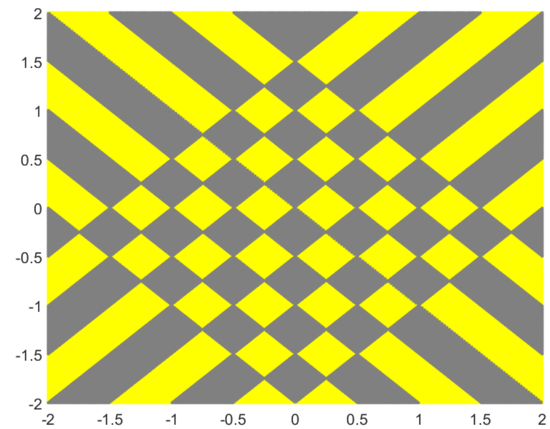
A 2-layer network with 16 neurons in the first layer cannot represent the pattern with less than 41 neurons in the second layer

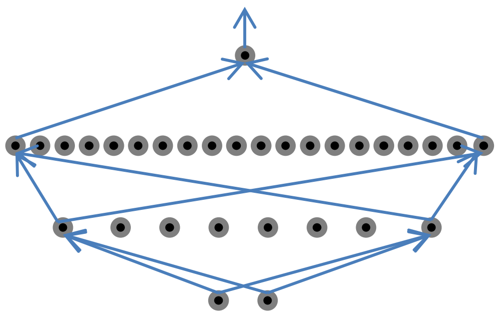
- A neural network *can* represent any function provided it has sufficient *capacity*
  - I.e. sufficiently broad and deep to represent the function
- Not all architectures can represent any function

Sufficiency of architecture



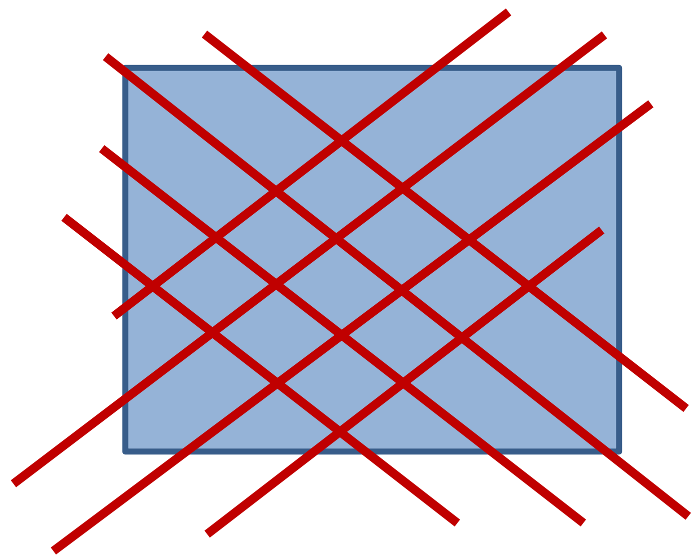
A network with 16 or more neurons in the first layer is capable of representing the figure to the right perfectly





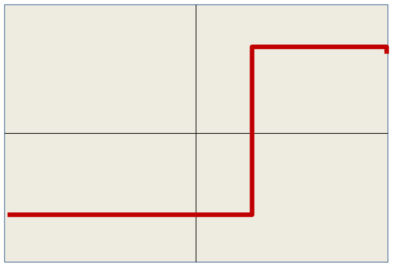
A network with less than 16 neurons in the first layer cannot represent this pattern exactly

- *With caveats..*



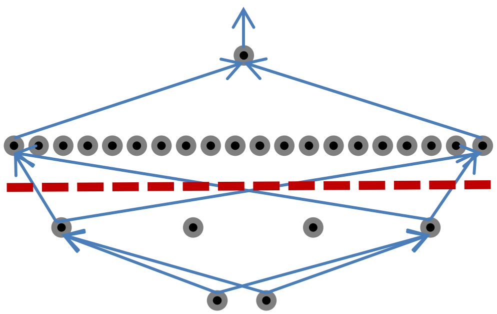
Why?

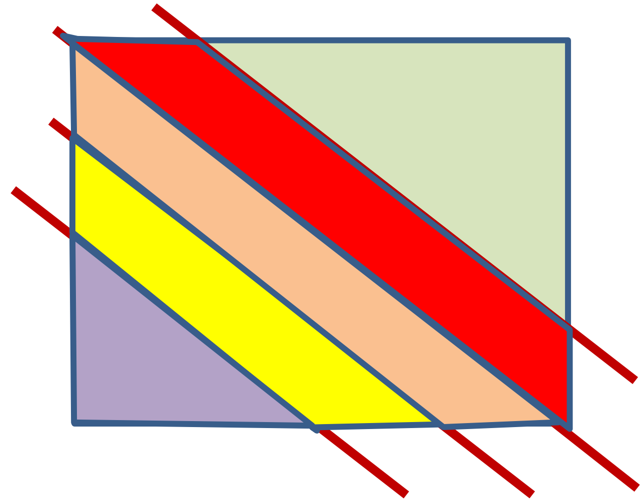
Sufficiency of architecture



This effect is because we use the threshold activation

It *gates* information in the input from later layers

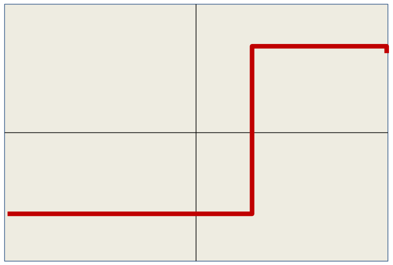




The pattern of outputs within any colored region is identical

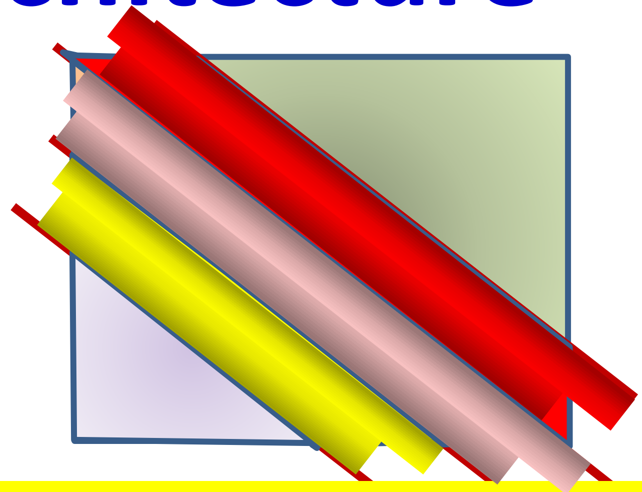
Subsequent layers do not obtain enough information to partition them

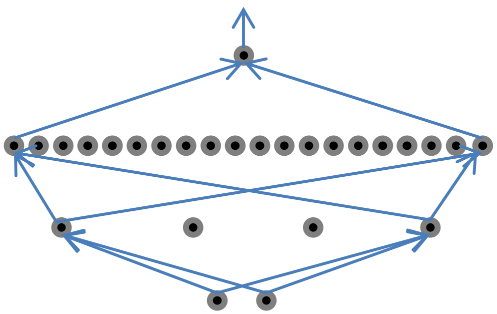
Sufficiency of architecture



This effect is because we use the threshold activation

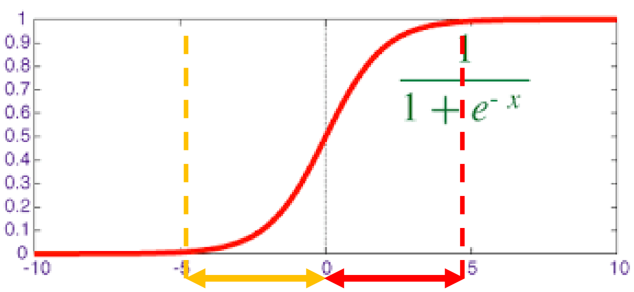
It *gates* information in the input from later layers

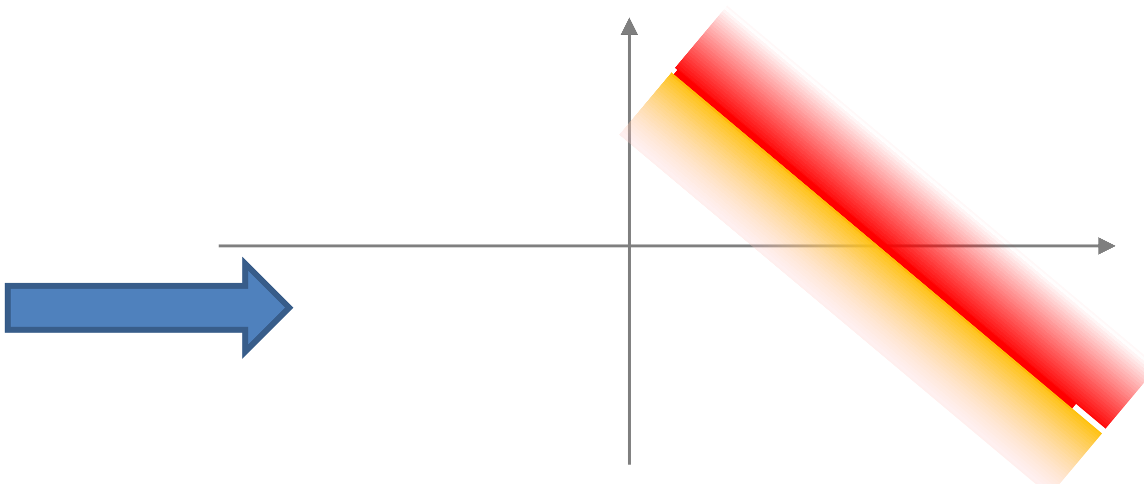




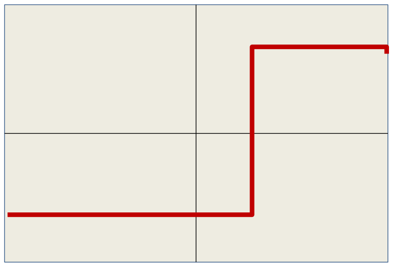
Continuous activation functions result in graded output at the layer

The gradation provides information to subsequent layers, to capture information “missed” by the lower layer (i.e. it “passes” information to subsequent layers).



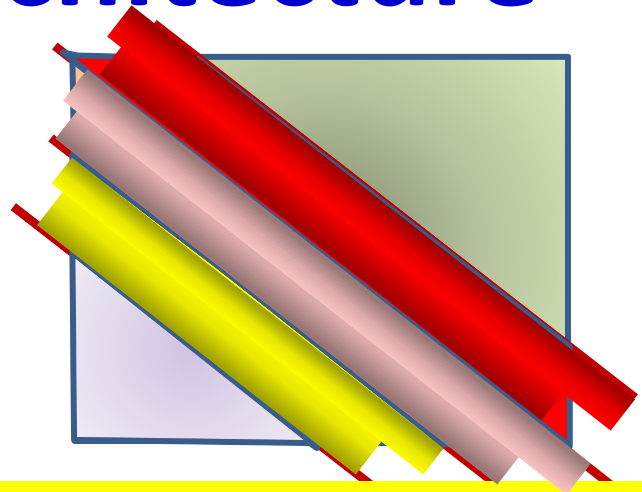


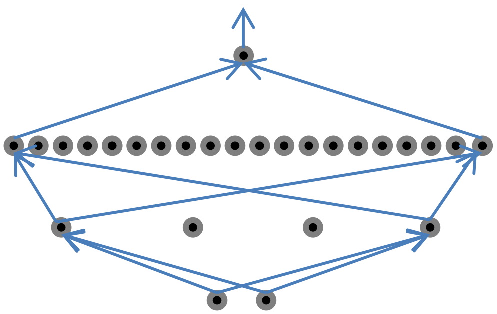
Sufficiency of architecture



This effect is because we use the threshold activation

It *gates* information in the input from later layers

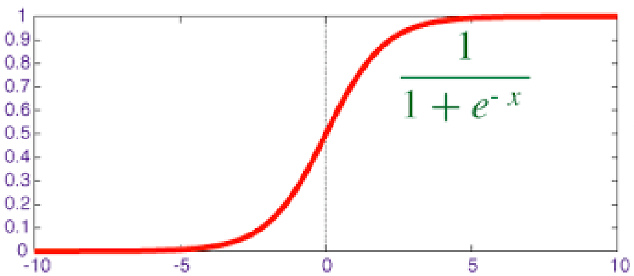


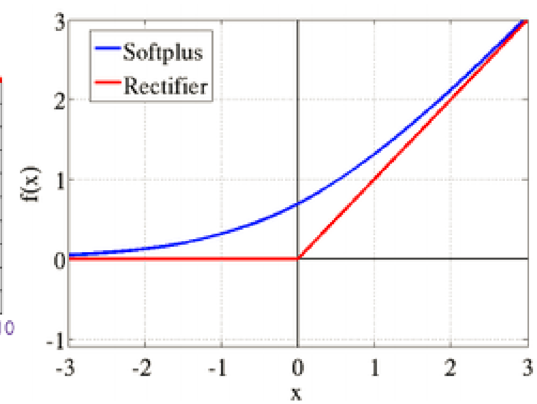


Continuous activation functions result in graded output at the layer

The gradation provides information to subsequent layers, to capture information “missed” by the lower layer (i.e. it “passes” information to subsequent layers).

Activations with more gradation (e.g. RELU) pass more information



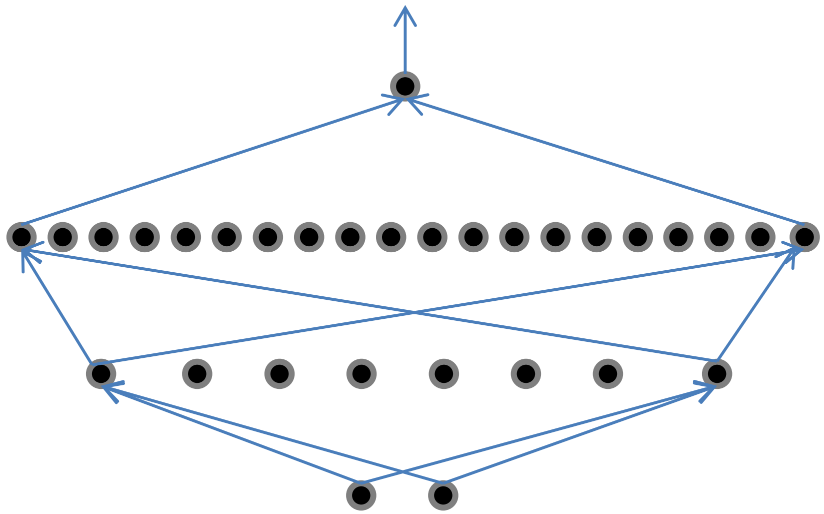


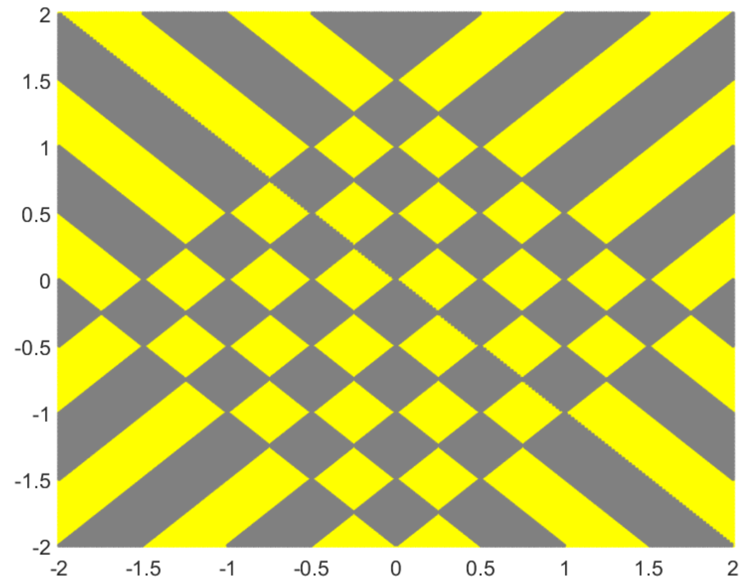
Width vs. Activations vs. Depth

- Narrow layers can still pass information to subsequent layers if the activation function is sufficiently graded

- But will require greater depth, to permit later layers to capture patterns

Sufficiency of architecture





- The *capacity* of a network has various definitions
  - *Information* or *Storage* capacity: how many patterns can it remember
  - VC dimension
    - bounded by the square of the number of weights in the network
  - From our perspective: largest number of disconnected convex regions it can represent

- A network with insufficient capacity *cannot* exactly model a function that requires a greater minimal number of convex hulls than the capacity of the network
  - But can approximate it with error

The “capacity” of a network

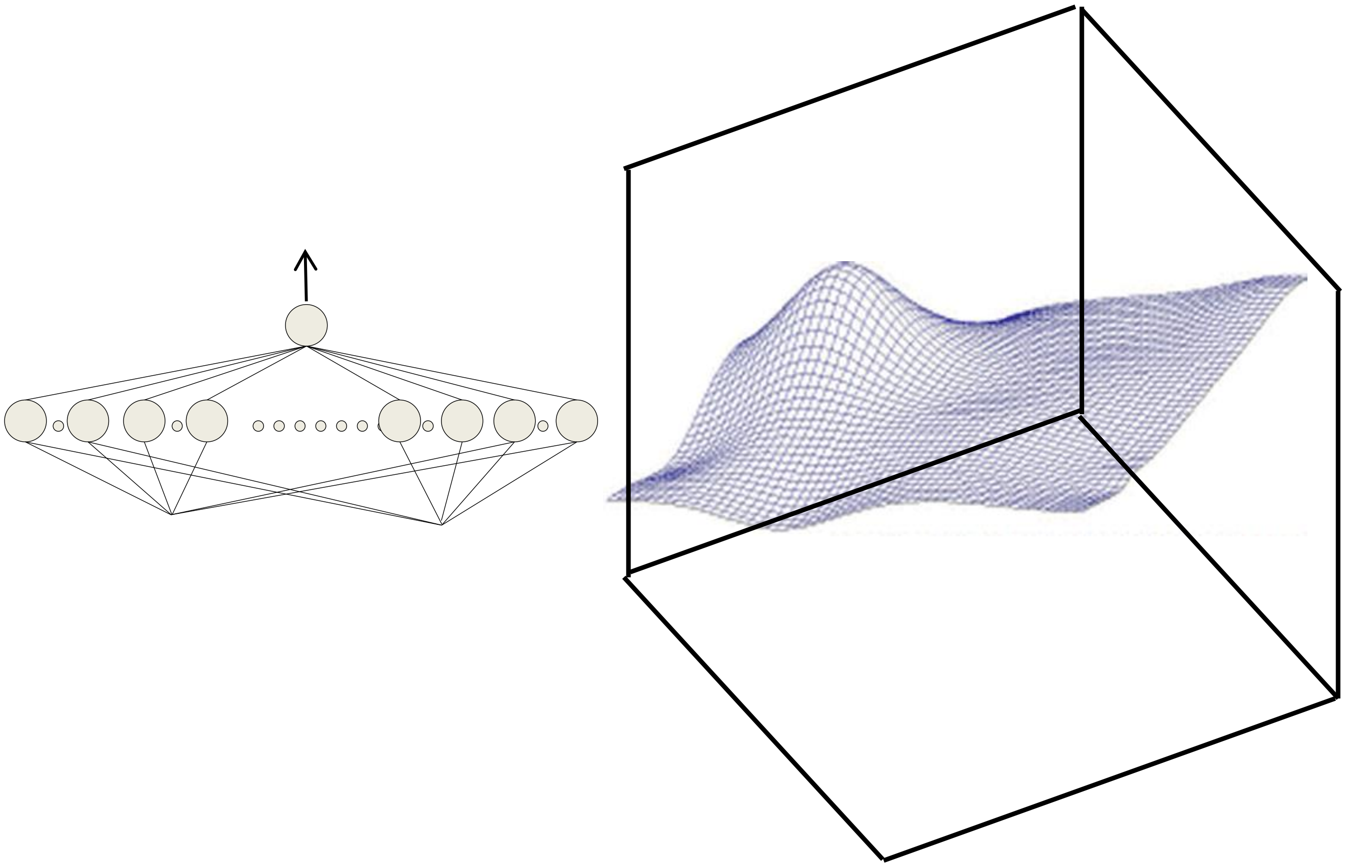
- VC dimension
- A separate lecture
  - Koiran and Sontag (1998): For “linear” or threshold units, VC dimension is proportional to the number of weights
    - For units with piecewise linear activation it is proportional to the square of the number of weights
  - Harvey, Liaw, Mehrabian “Nearly-tight VC-dimension bounds for piecewise linear neural networks” (2017):
    - For any W, L s.t. $W > CL > C^2$, there exisits a RELU network with $\leq L$ layers, $\leq W$ weights with VC dimension $\geq \frac{WL}{C}\log_2(\frac{W}{L})$
  - Friedland, Krell, “A Capacity Scaling Law for Artificial Neural Networks” (2017):
    - VC dimension of a linear/threshold net is $\mathcal{O}(MK)$, M is the overall number of hidden neurons, K is the weights per neuron

Lessons

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

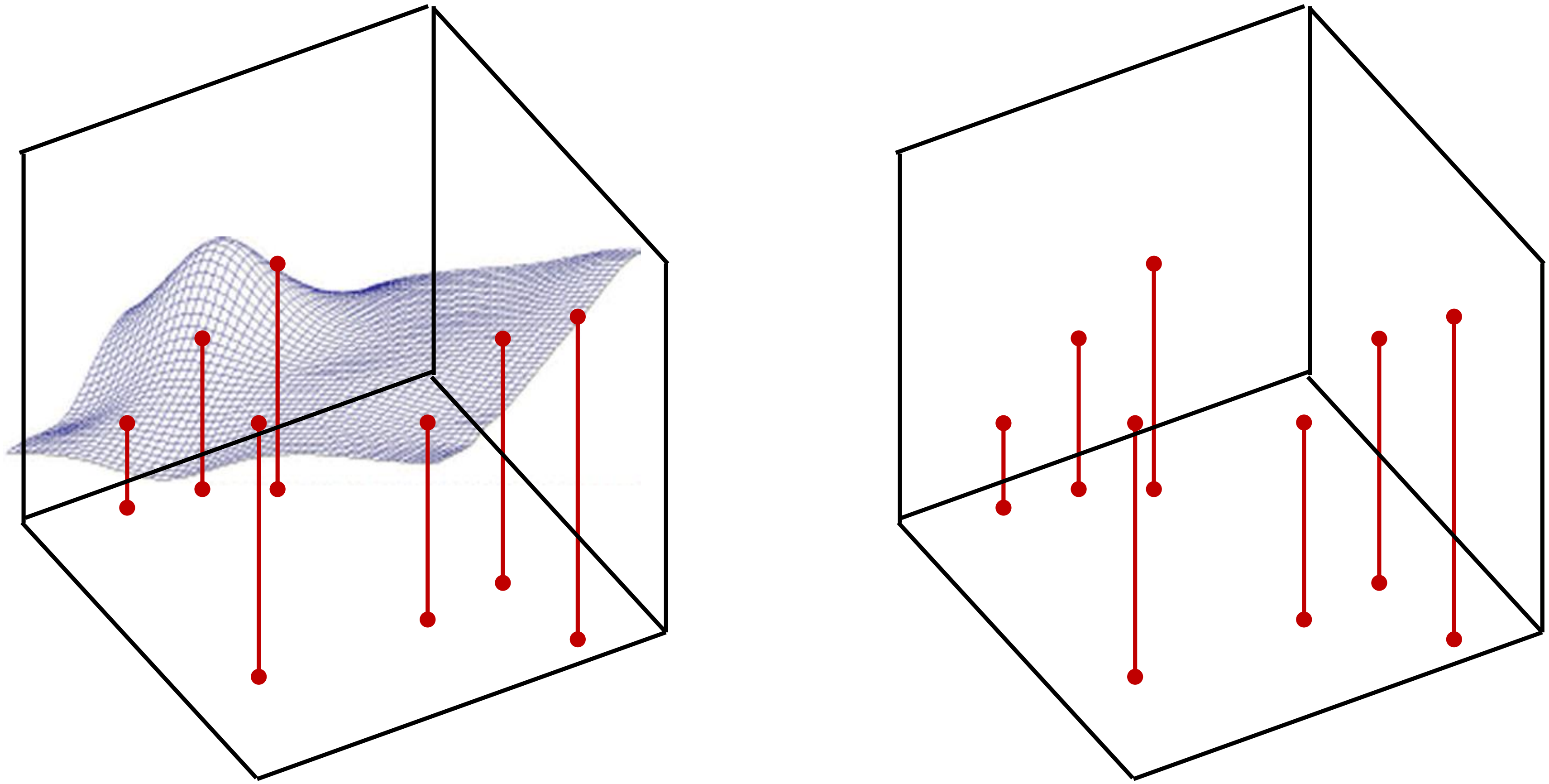
- A *single-layer* MLP can approximate anything to arbitrary precision
  - But could be exponentially or even infinitely wide in its inputs size
- Deeper MLPs can achieve the same precision with far fewer neurons
  - Deeper networks are more expressive

Learning the network



- The neural network can approximate *any* function
- But only if the function is known *a priori*

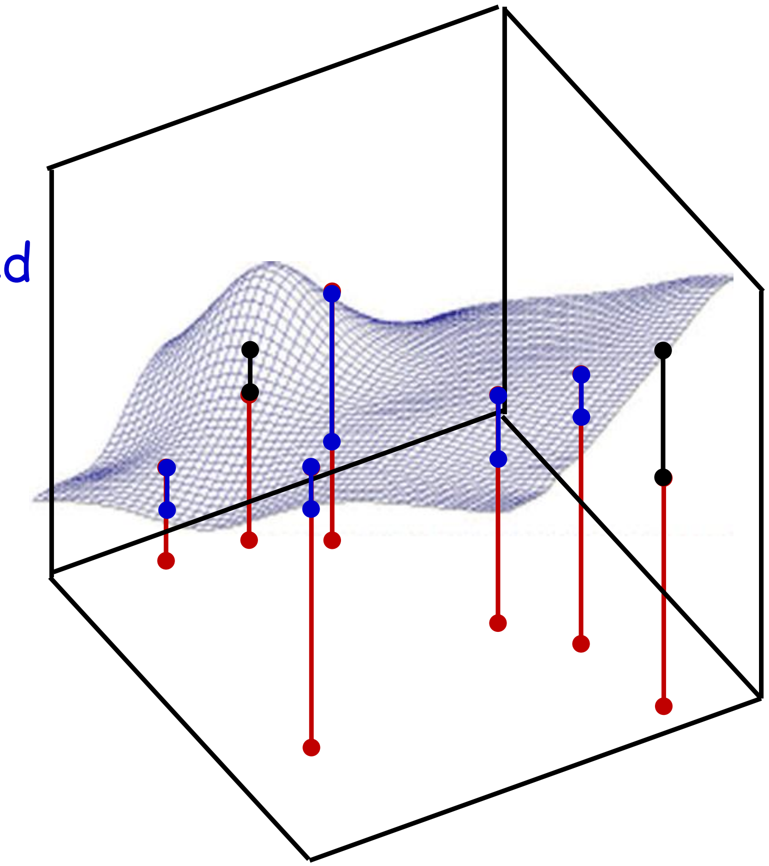
Learning the network



- In reality, we will only get a few *snapshots* of the function to learn it from
- We must learn the entire function from these “training” snapshots

General approach to training

Blue lines: error when function is *below* desired output

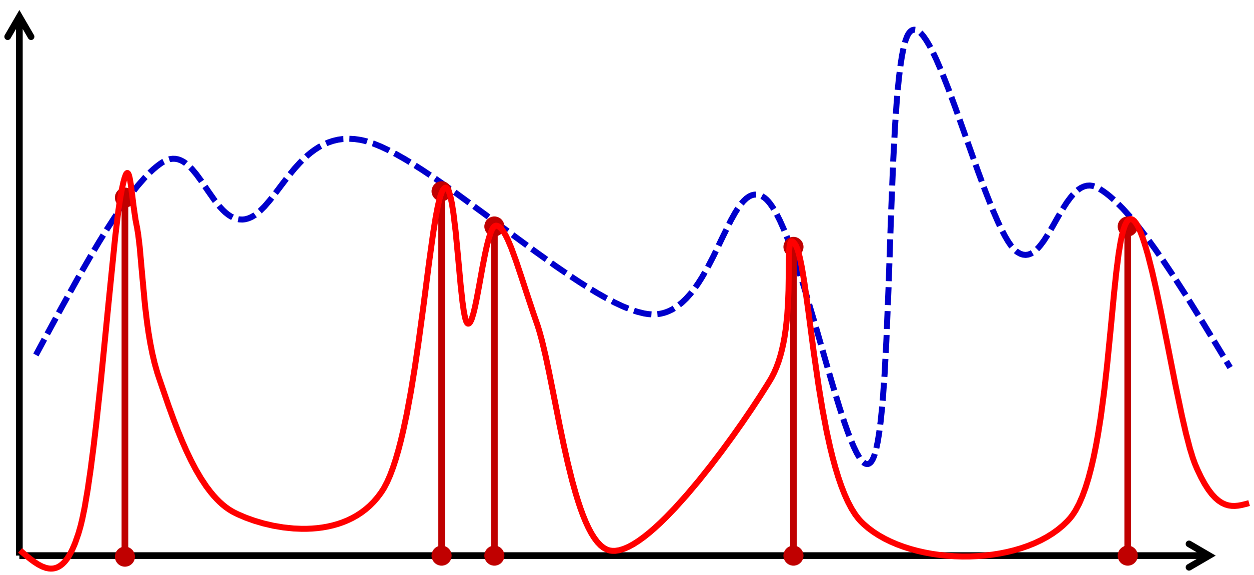


Black lines: error when function is *above* desired output

$$E = \sum_i (y_i - f(\mathbf{x}_i, \mathbf{W}))^2$$

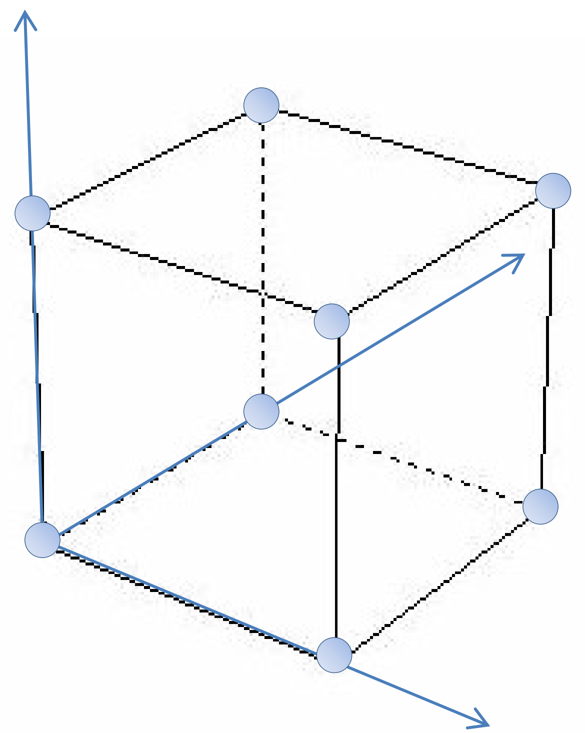
- Define an *error* between the ***actual*** network output for any parameter value and the *desired* output
  - Error typically defined as the *sum* of the squared error over individual training instances

General approach to training



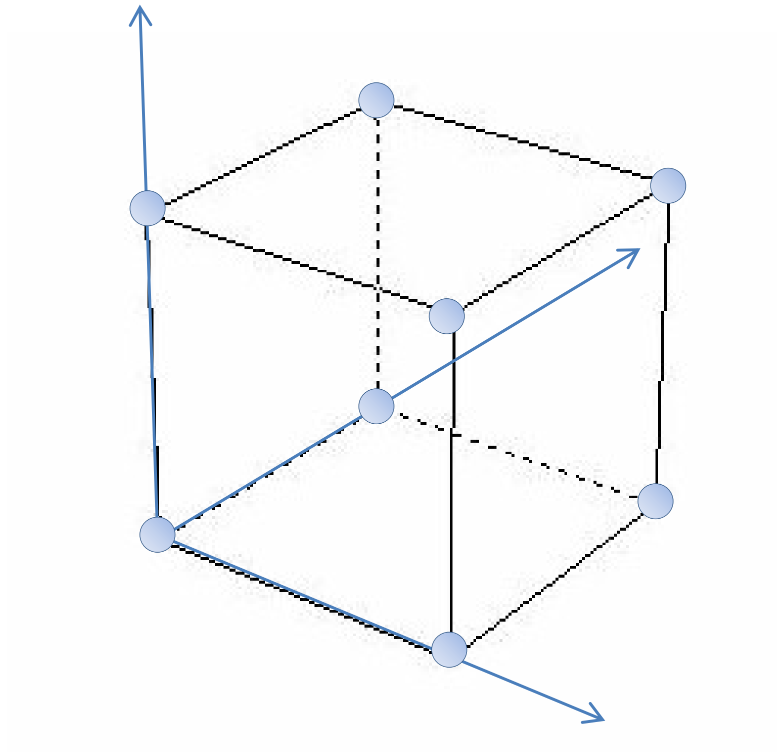
- Problem: Network may just learn the values at the inputs
  - Learn the red curve instead of the dotted blue one
    - Given only the red vertical bars as inputs
  - Need "smoothness" constraints

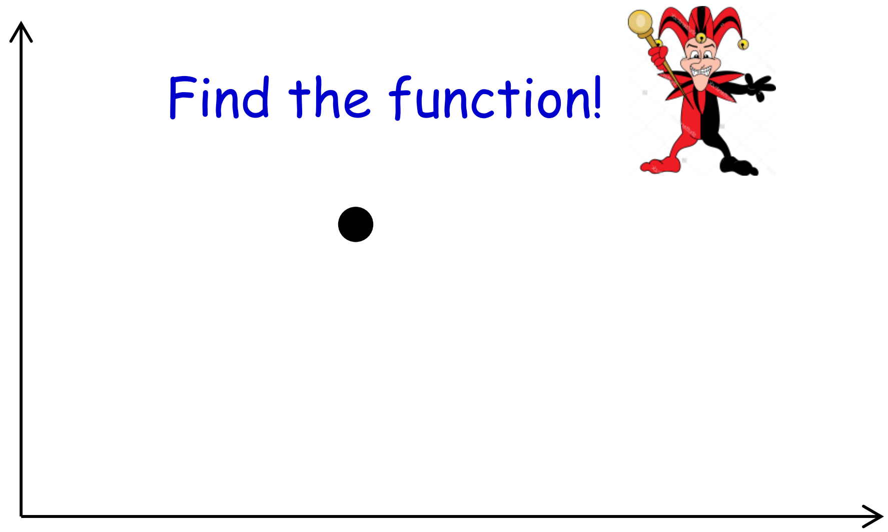
Data under-specification in learning



- Consider a binary 100-dimensional input
- There are $2^{100}=10^{30}$ possible inputs
- Complete specification of the function will require specification of 10^{30} output values
- A training set with only 10^{15} training instances will be off by a factor of 10^{15}

Data under-specification in learning



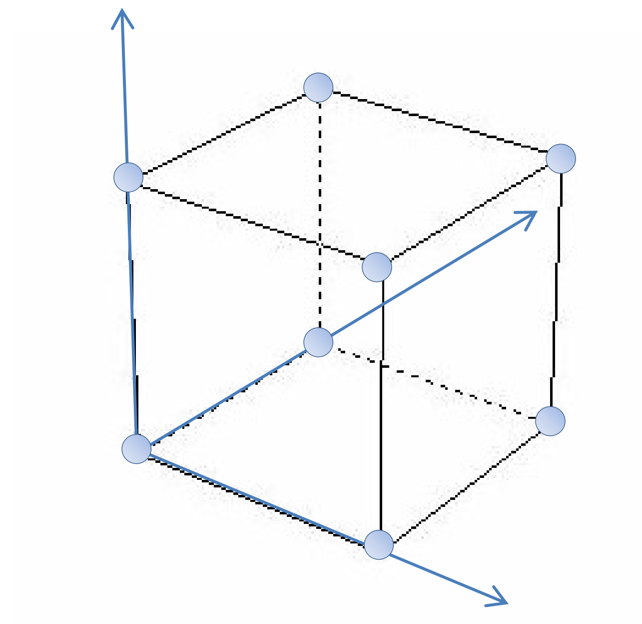


- Consider a binary 100-dimensional input
- There are $2^{100}=10^{30}$ possible inputs
- Complete specification of the function will require specification of 10^{30} output values
- A training set with only 10^{15} training instances will be off by a factor of 10^{15}

Data under-specification in learning

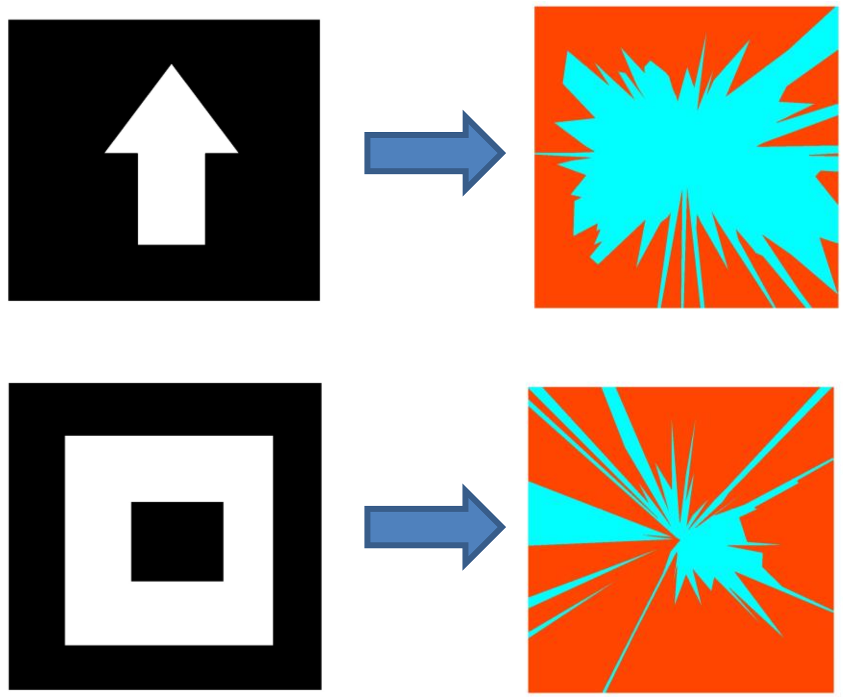
- MLPs naturally impose constraints

- MLPs are universal approximators
  - Arbitrarily increasing size can give you arbitrarily wiggly functions
  - The function will remain ill-defined on the majority of the space



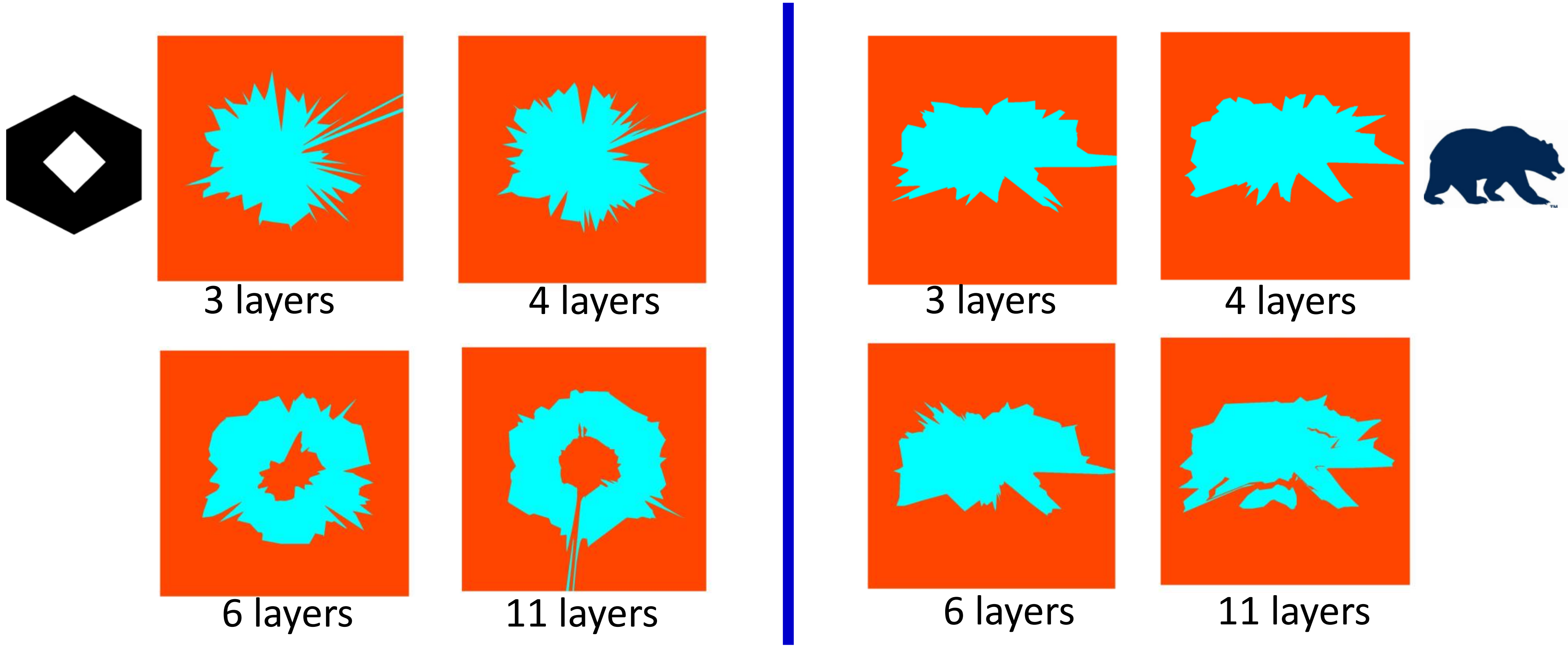
- *For a given number of parameters deeper networks impose more smoothness than shallow ones*
  - Each layer works on the already smooth surface output by the previous layer

Even when we get it all right



- Typical results (varies with initialization)
- 1000 training points
  - Many orders of magnitude more than you usually get
- All the training tricks known to mankind

But depth and training data help

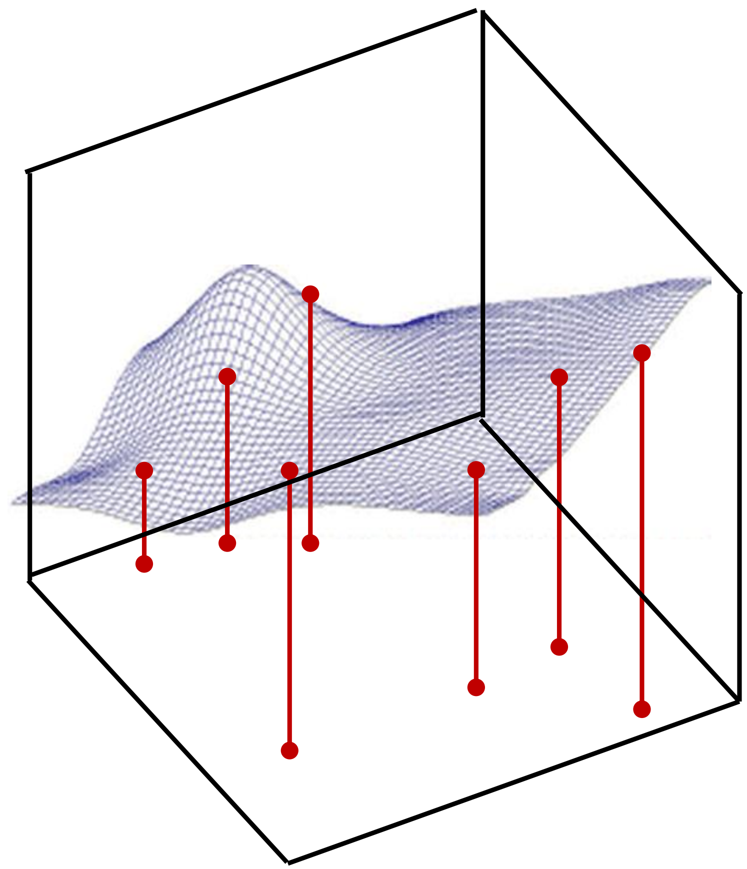


- Deeper networks seem to learn better, for the same number of total neurons
  - *Implicit smoothness constraints*
    - *As opposed to explicit constraints from more conventional classification models*
- **Similar functions not learnable using more usual pattern-recognition models!!**

10000 training instances

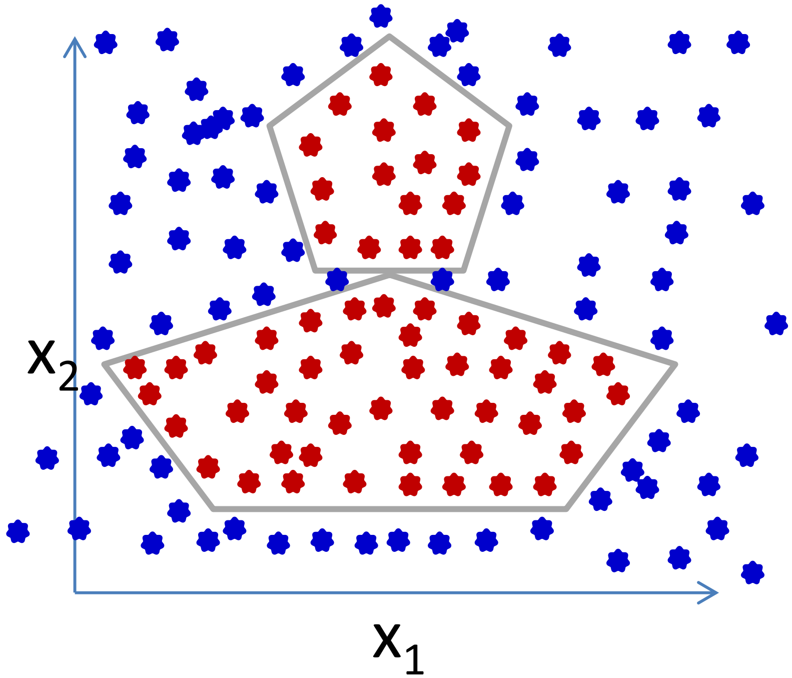
Part 3: What does the network learn?

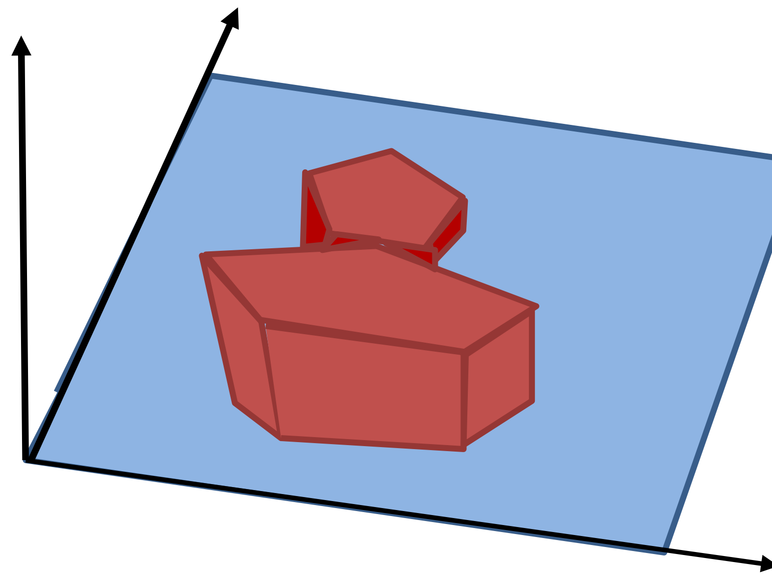
Learning in the net



- Problem: Given a collection of input-output pairs, learn the function

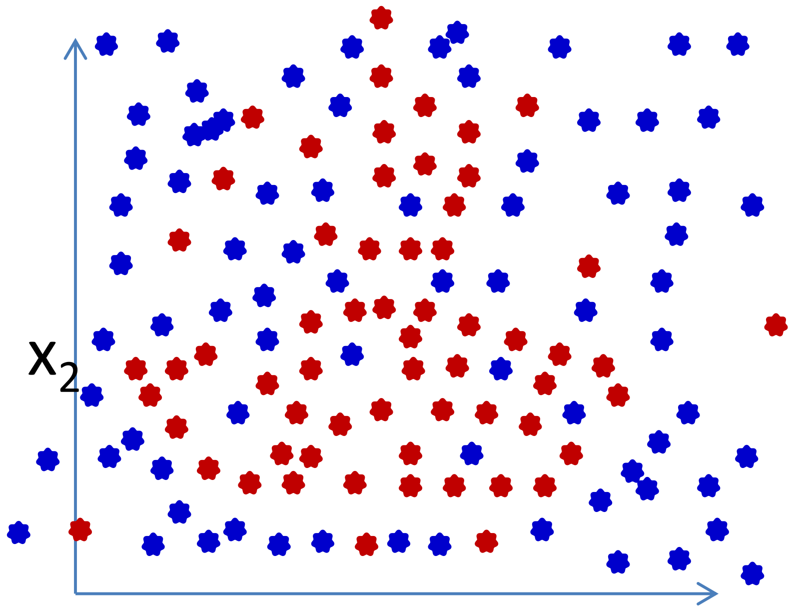
Learning for classification

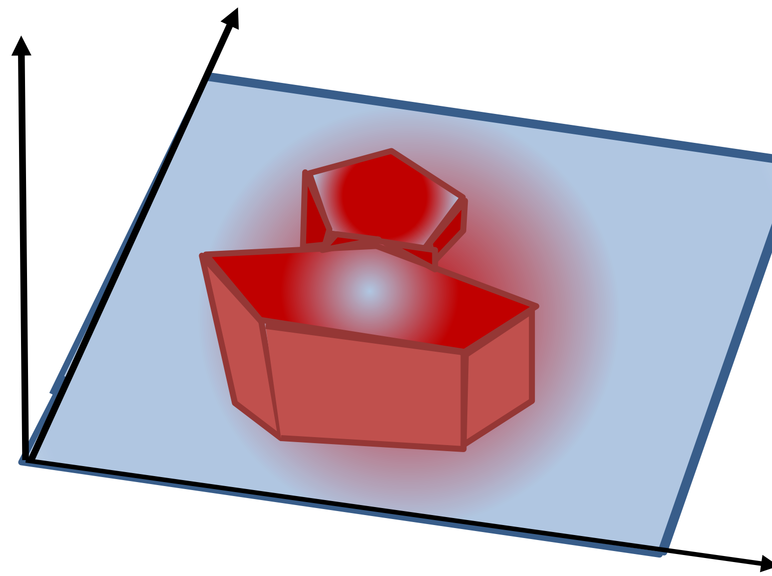




- When the net must learn to classify..
  - Learn the classification boundaries that separate the training instances

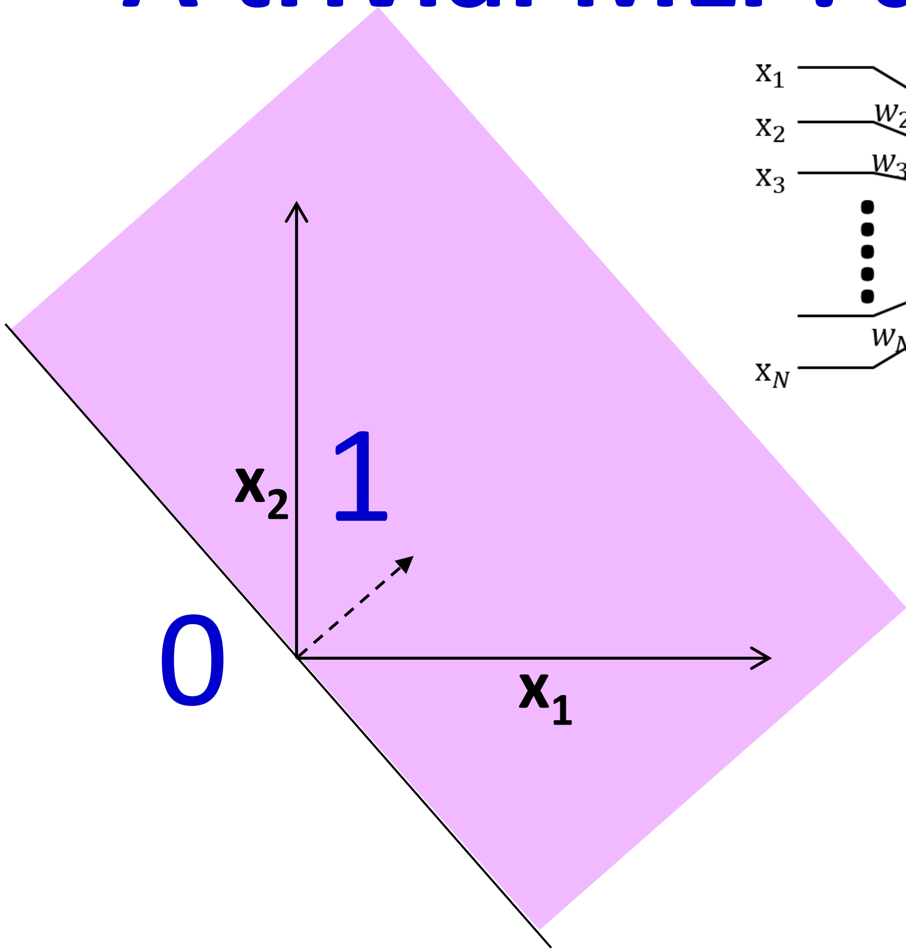
Learning for classification

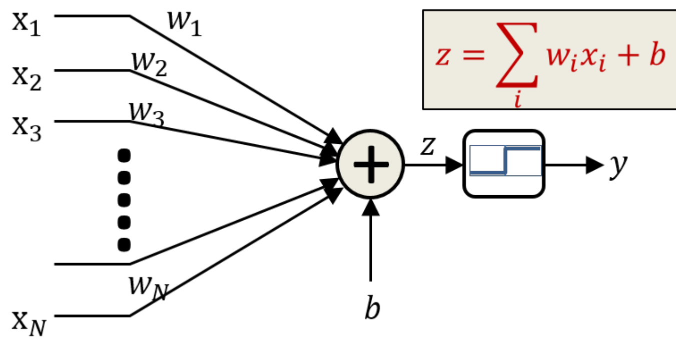


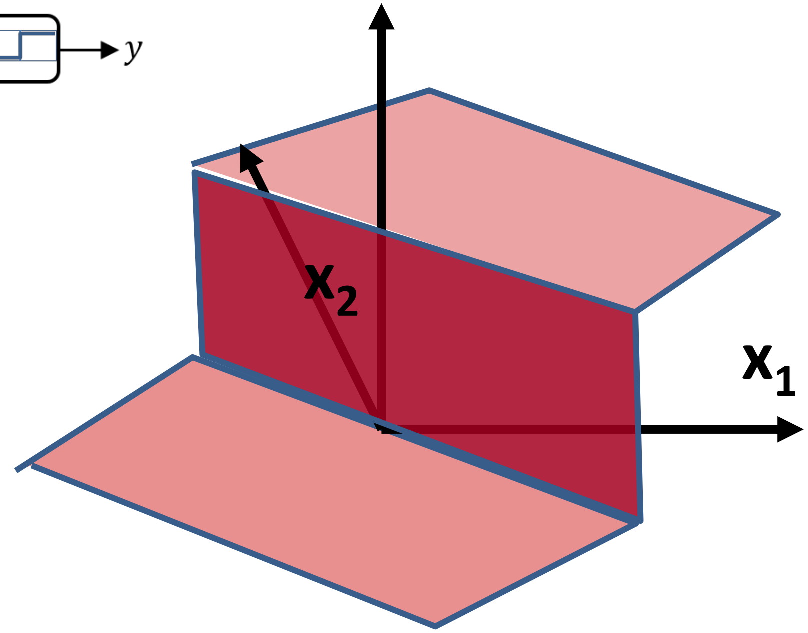


- In reality
  - In general not really cleanly separated
    - So what is the function we learn?

A trivial MLP: a single perceptron

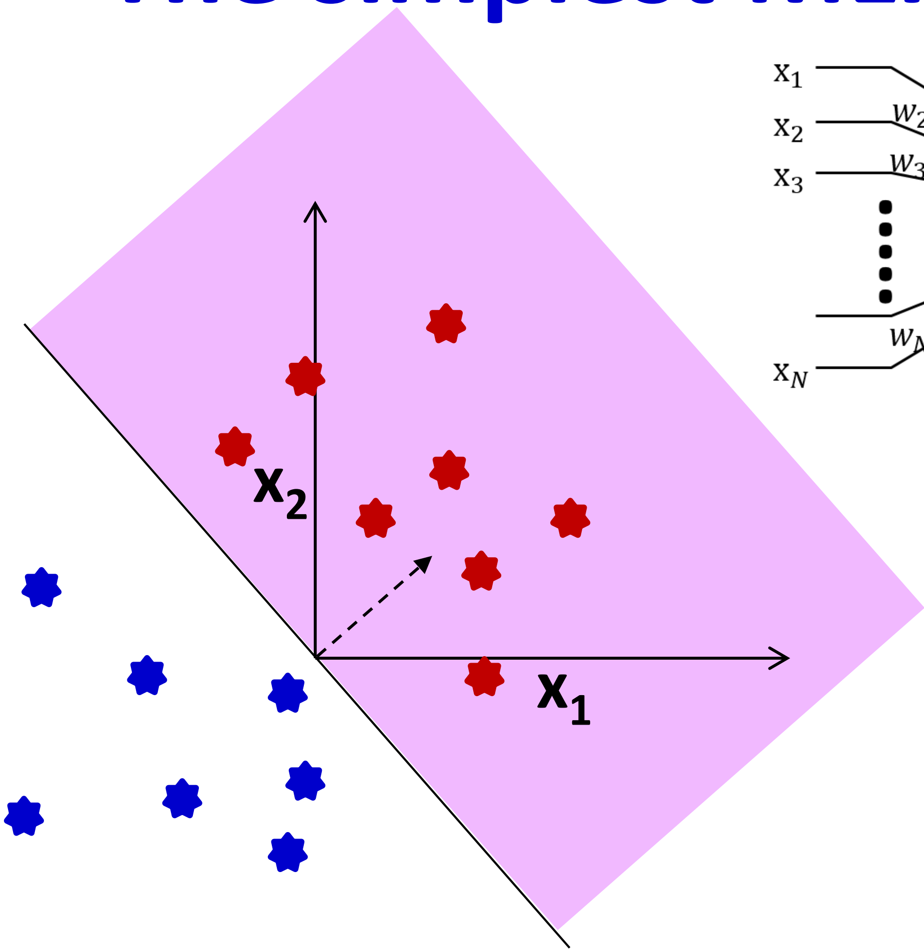


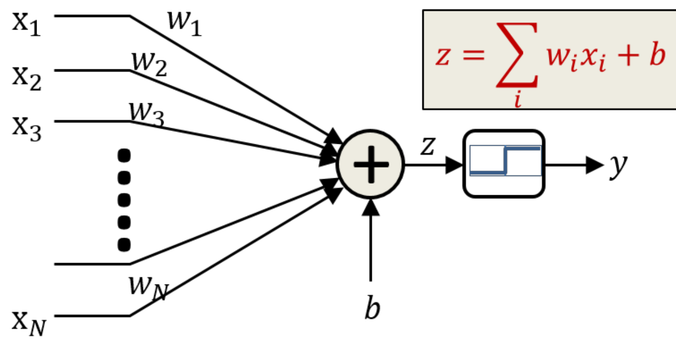


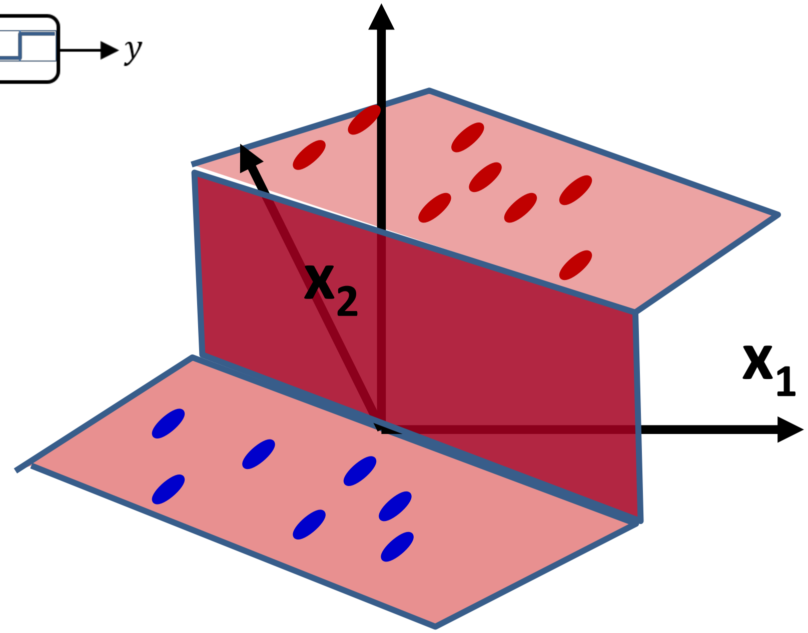


- Learn this function
  - A step function across a hyperplane

The simplest MLP: a single perceptron

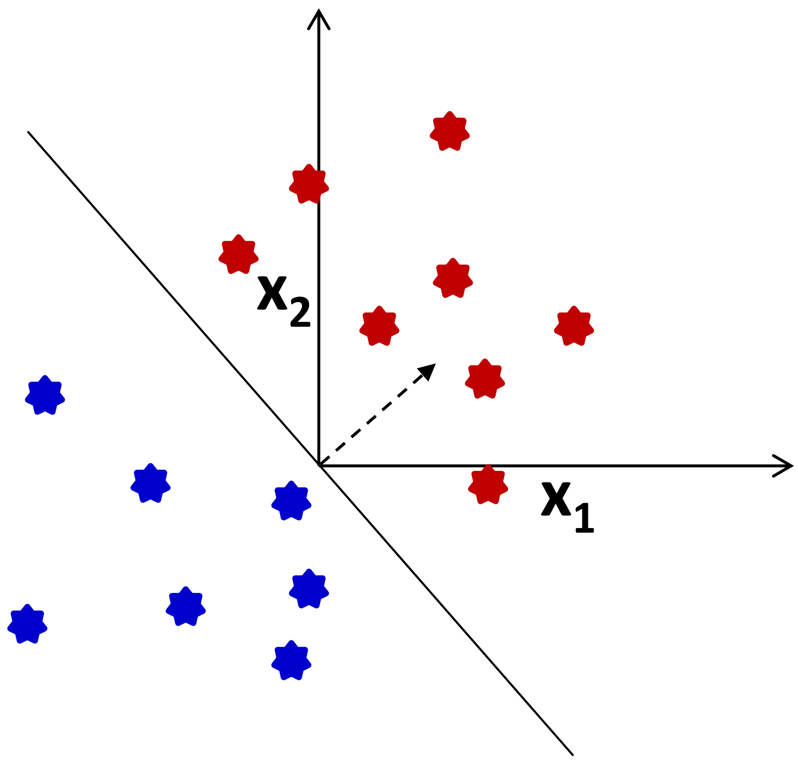


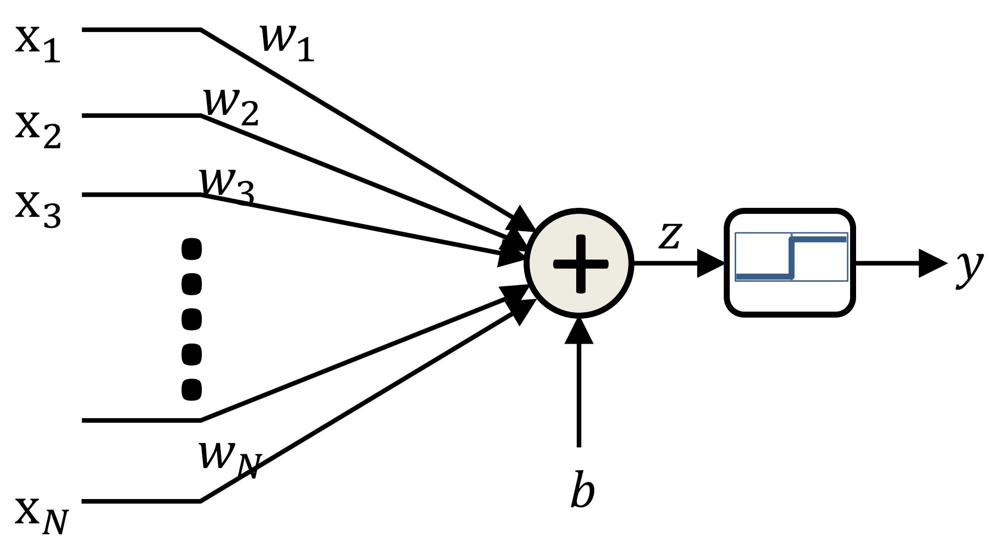




- Learn this function
  - A step function across a hyperplane
  - Given only samples form it

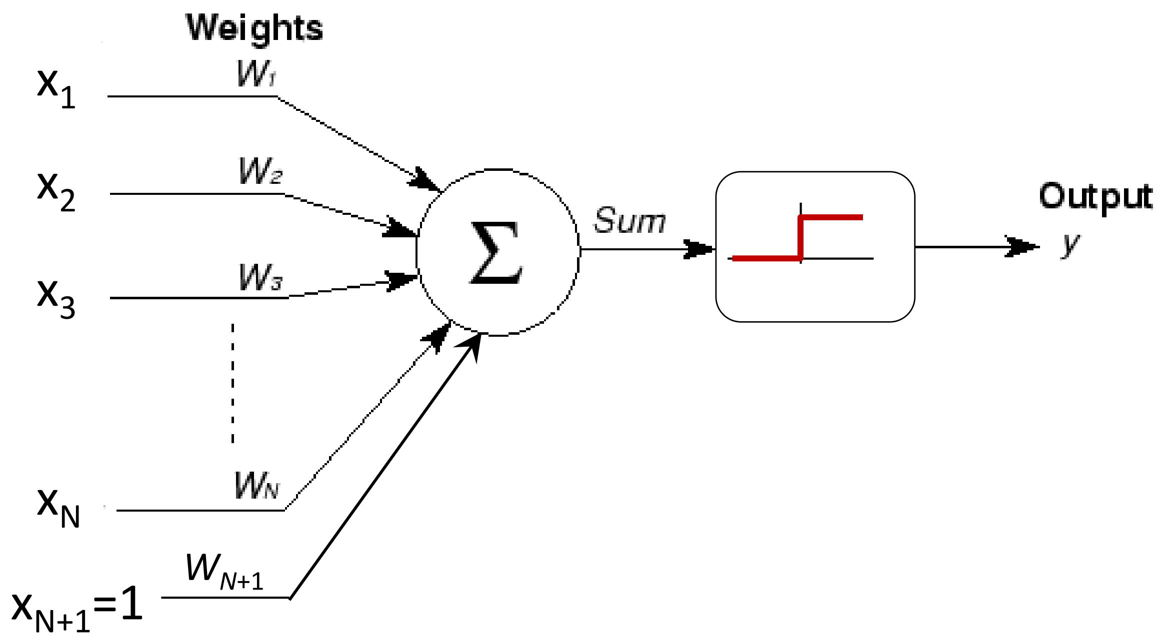
Learning the perceptron





- Given a number of input output pairs, learn the weights and bias
  - $y = \begin{cases} 1 \ \ if \ \ \sum_{i=1}^{N} w_i X_i - b \geq 0 \\ \quad 0 \ \ otherwise \end{cases}$
  - Learn $W = [w_1 .. w_N]$ and b, given several (X, y) pairs

Restating the perceptron

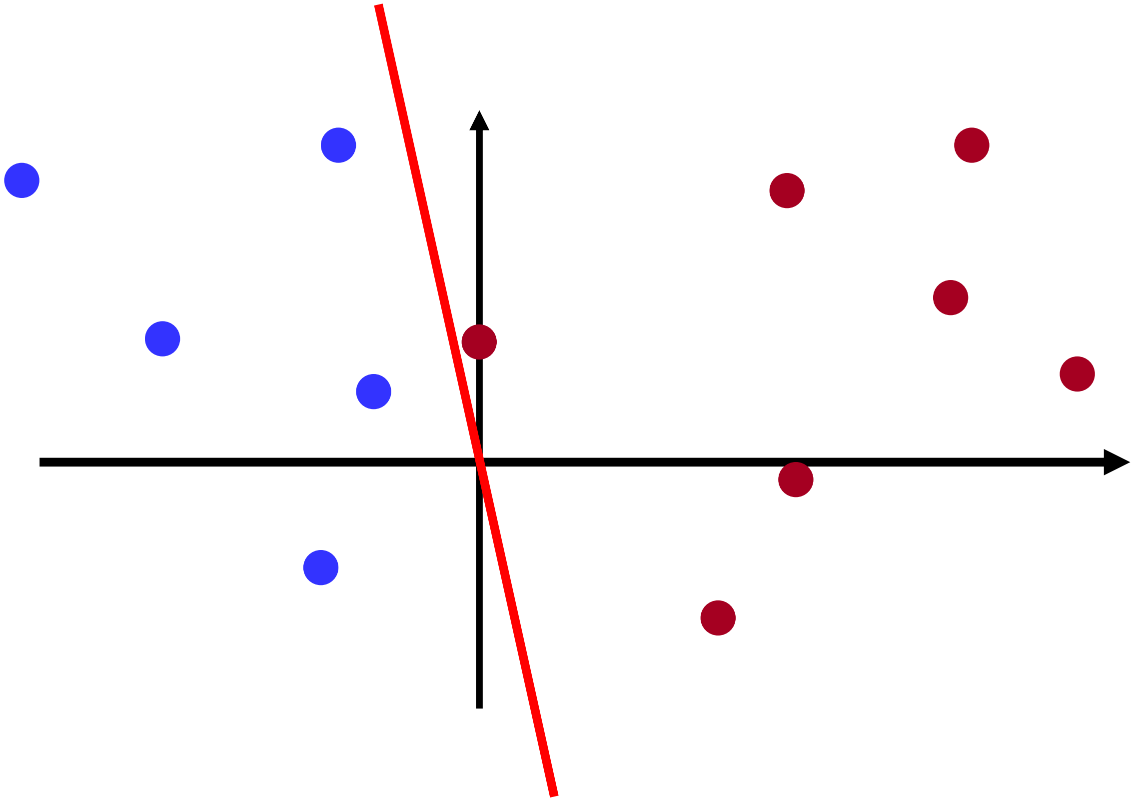


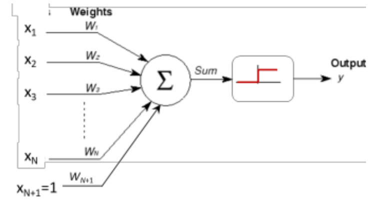
- Restating the perceptron equation by adding another dimension to X

$$y = \begin{cases} 1 \;\; if \;\; \sum_{i=1}^{N+1} w_i X_i \geq 0 \\ 0 \;\; otherwise \end{cases}$$

where $X_{N+1} = 1$

The Perceptron Problem





- Find the hyperplane $\sum_{i=1}^{N+1} w_i X_i = 0$ that perfectly separates the two groups of points

A simple learner: Perceptron Algorithm

- Given N training instances $(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)$
  - $Y_i = +1$ or -1 (instances are either positive or negative)

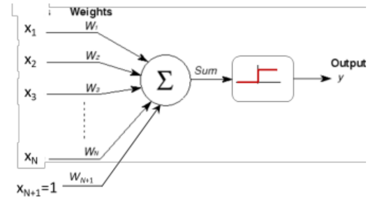
- Cycle through the training instances
- Only update W on misclassified instances
- If instance misclassified:
  - If instance is positive class

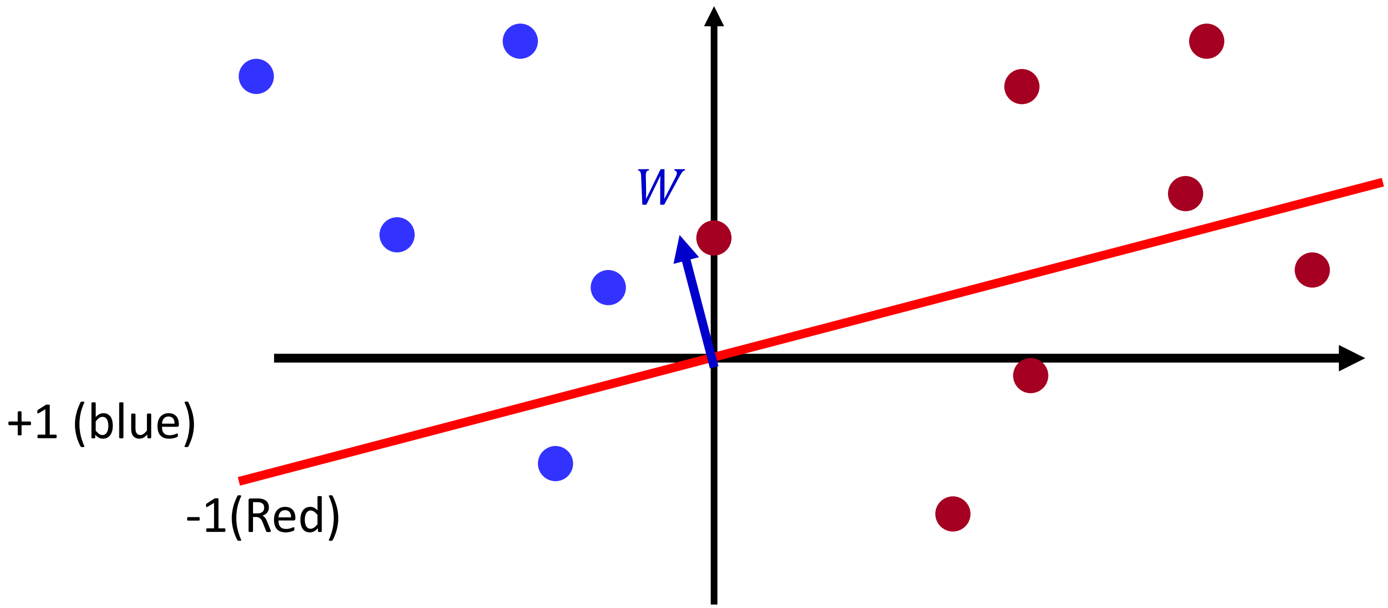
$$W = W + X_i$$

  - If instance is negative class

$$W = W - X_i$$

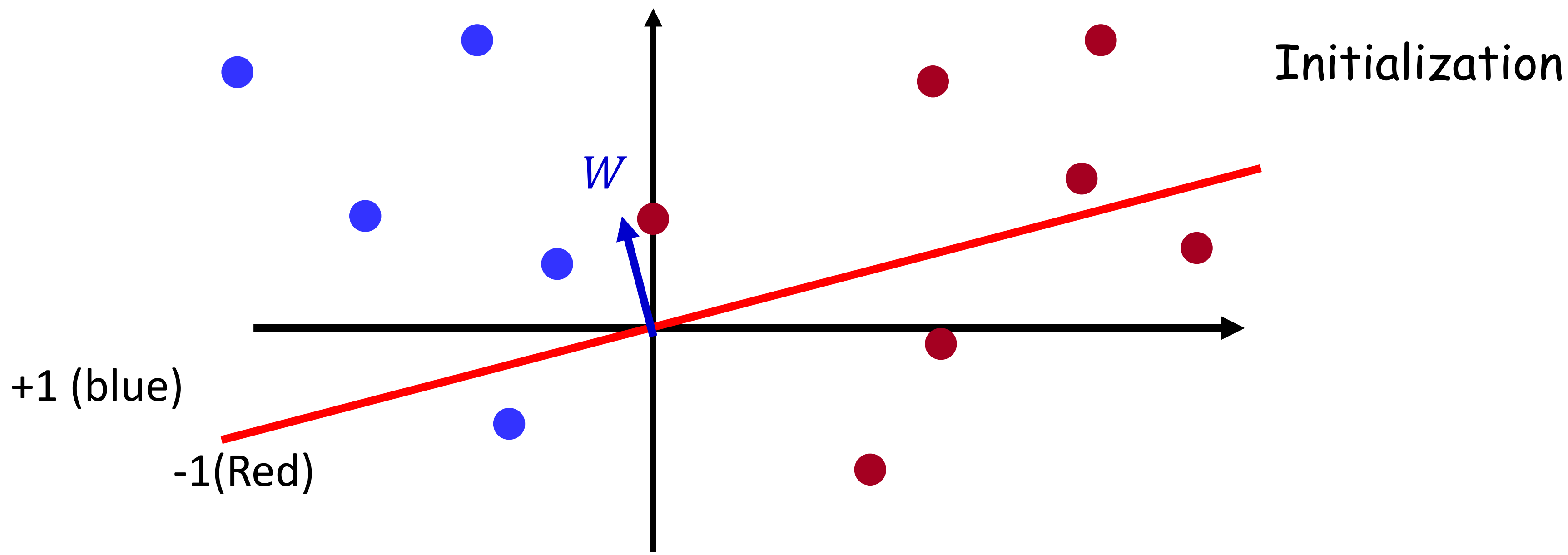
The Perceptron Algorithm



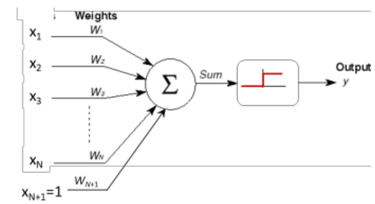


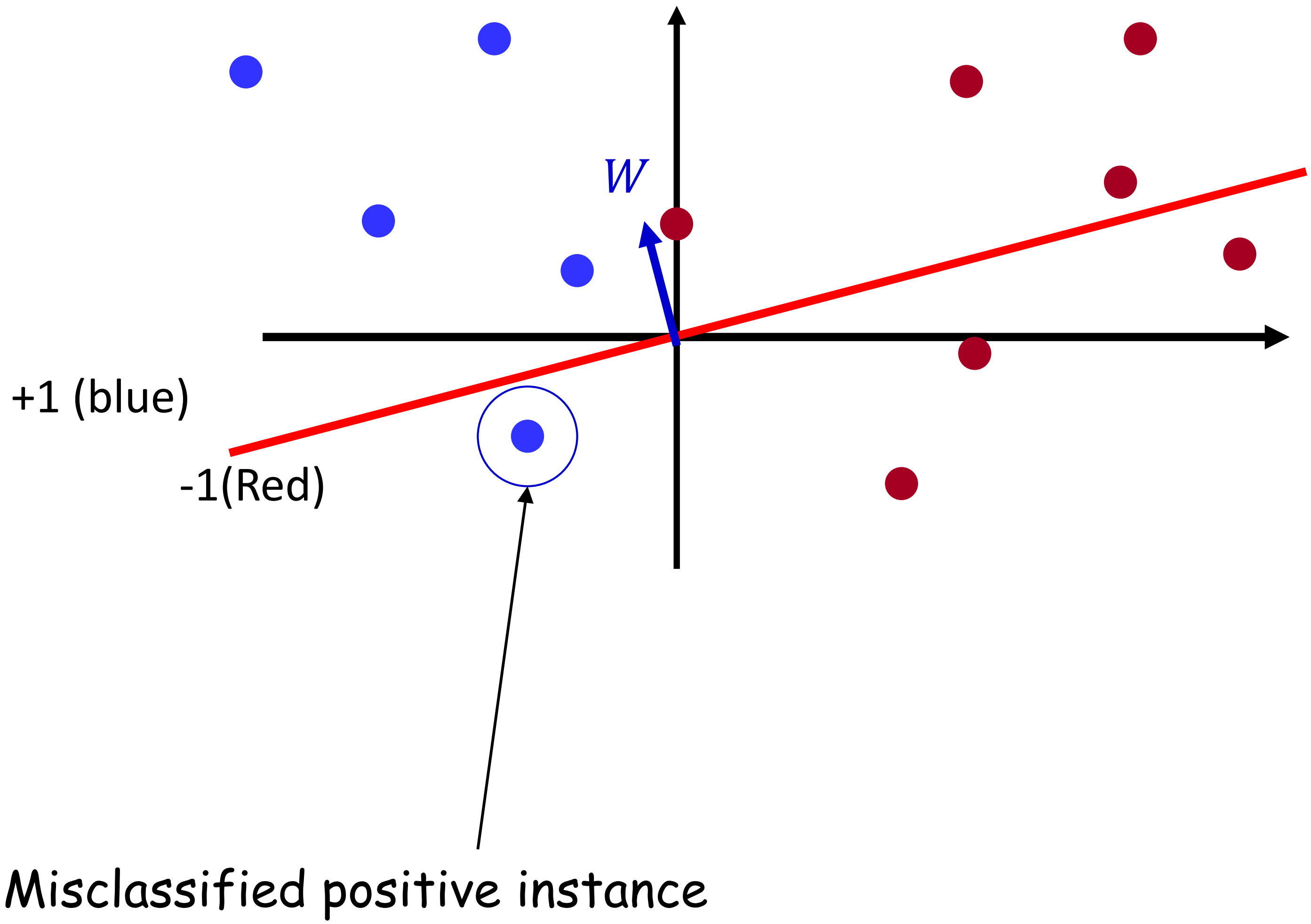
- **Initialize:** Randomly initialize the hyperplane
  - I.e. randomly initialize the normal vector W
  - Classification rule $sign(W^T X)$
  - The random initial plane will make mistakes

Perceptron Algorithm

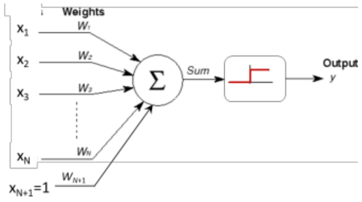


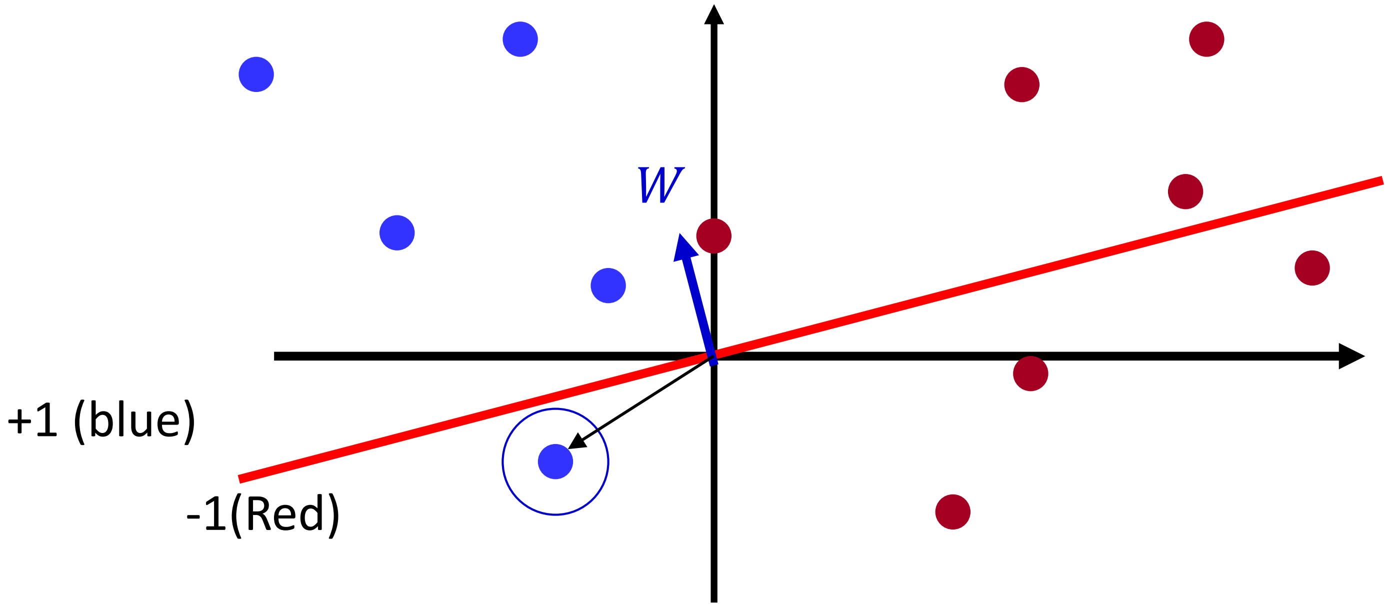
Perceptron Algorithm



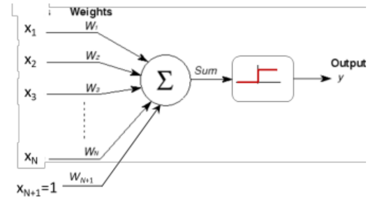


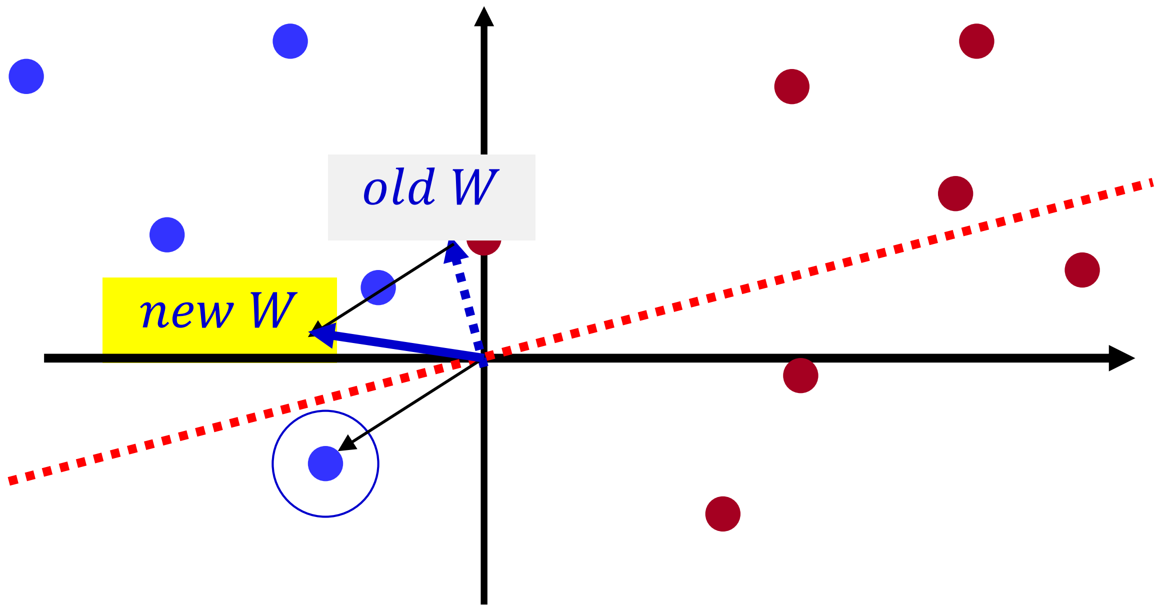
Perceptron Algorithm





Perceptron Algorithm

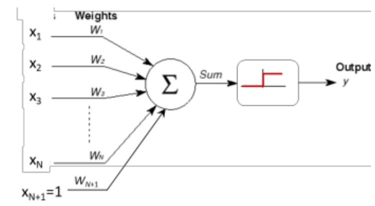


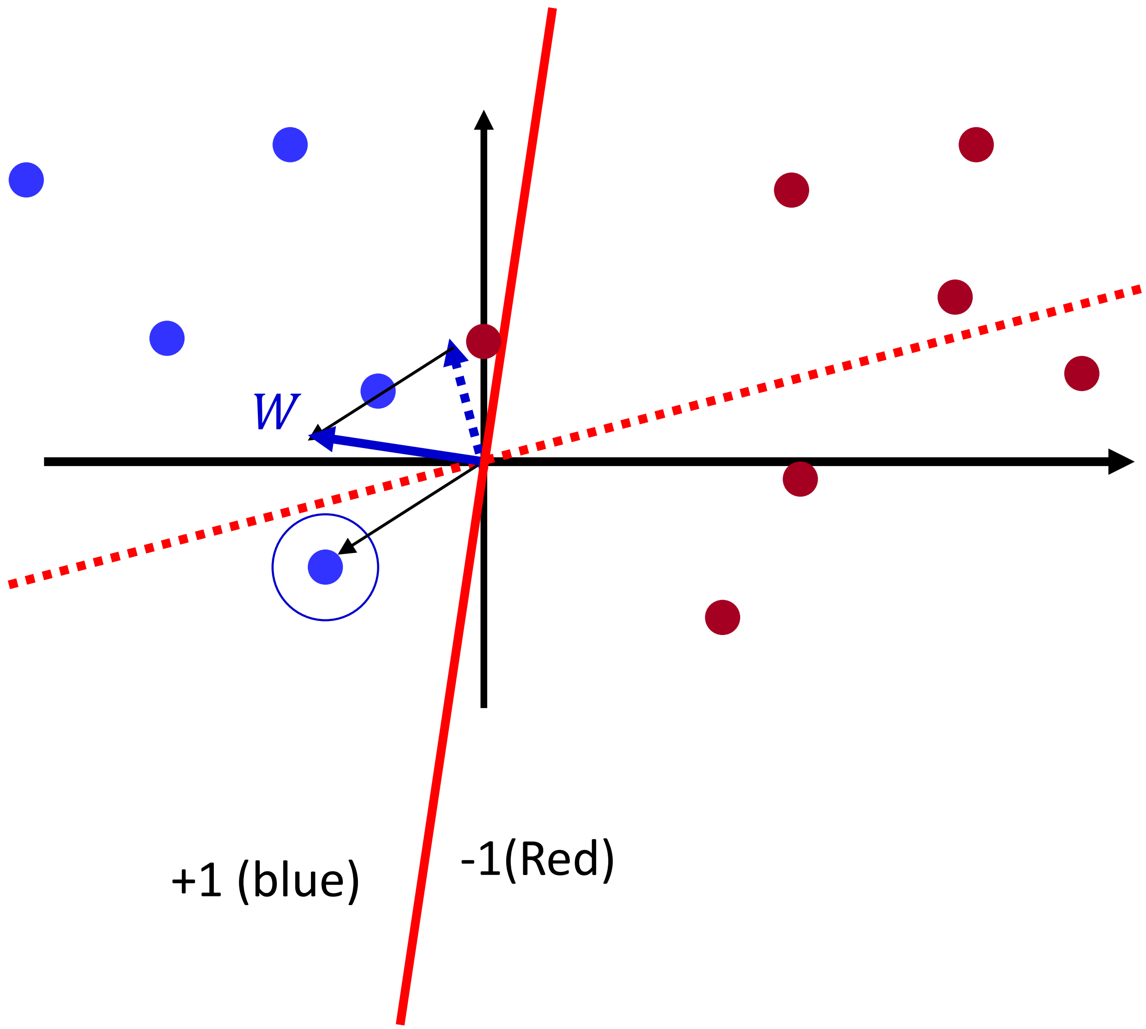


Updated weight vector

Misclassified *positive* instance, *add* it to W

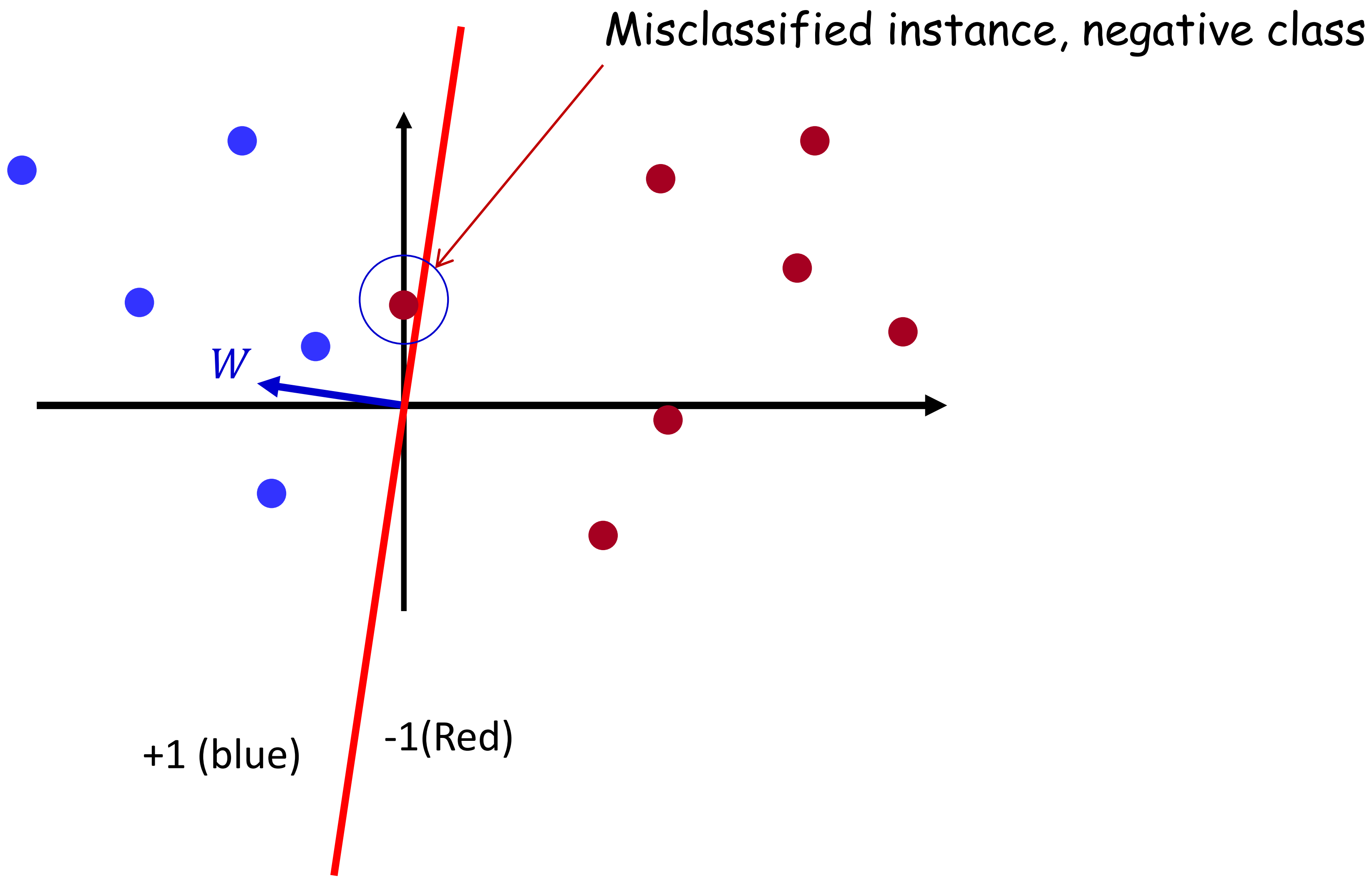
Perceptron Algorithm



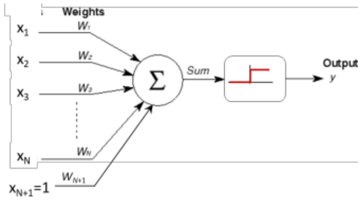


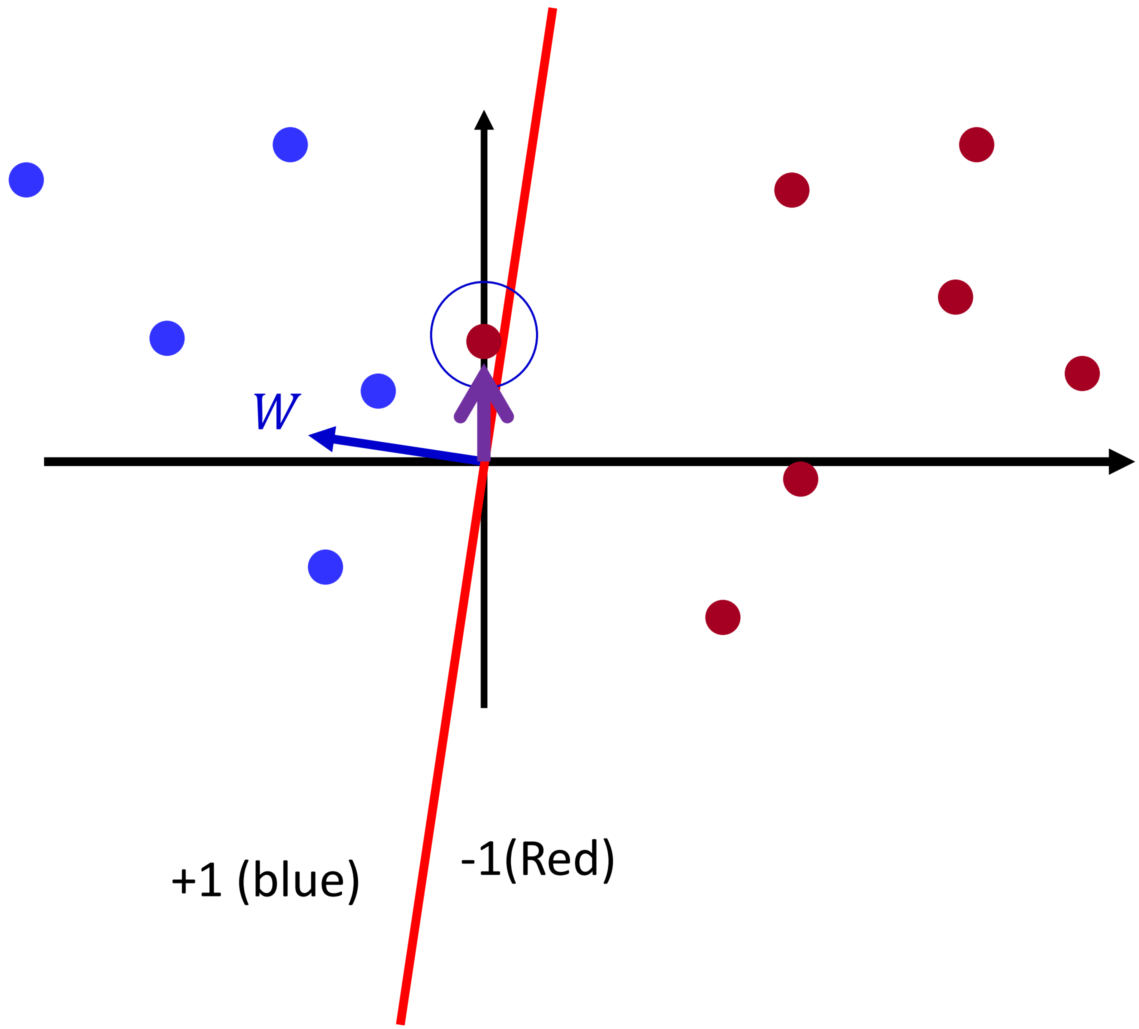
Updated hyperplane

Perceptron Algorithm

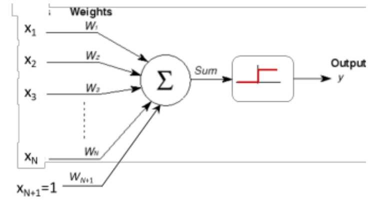


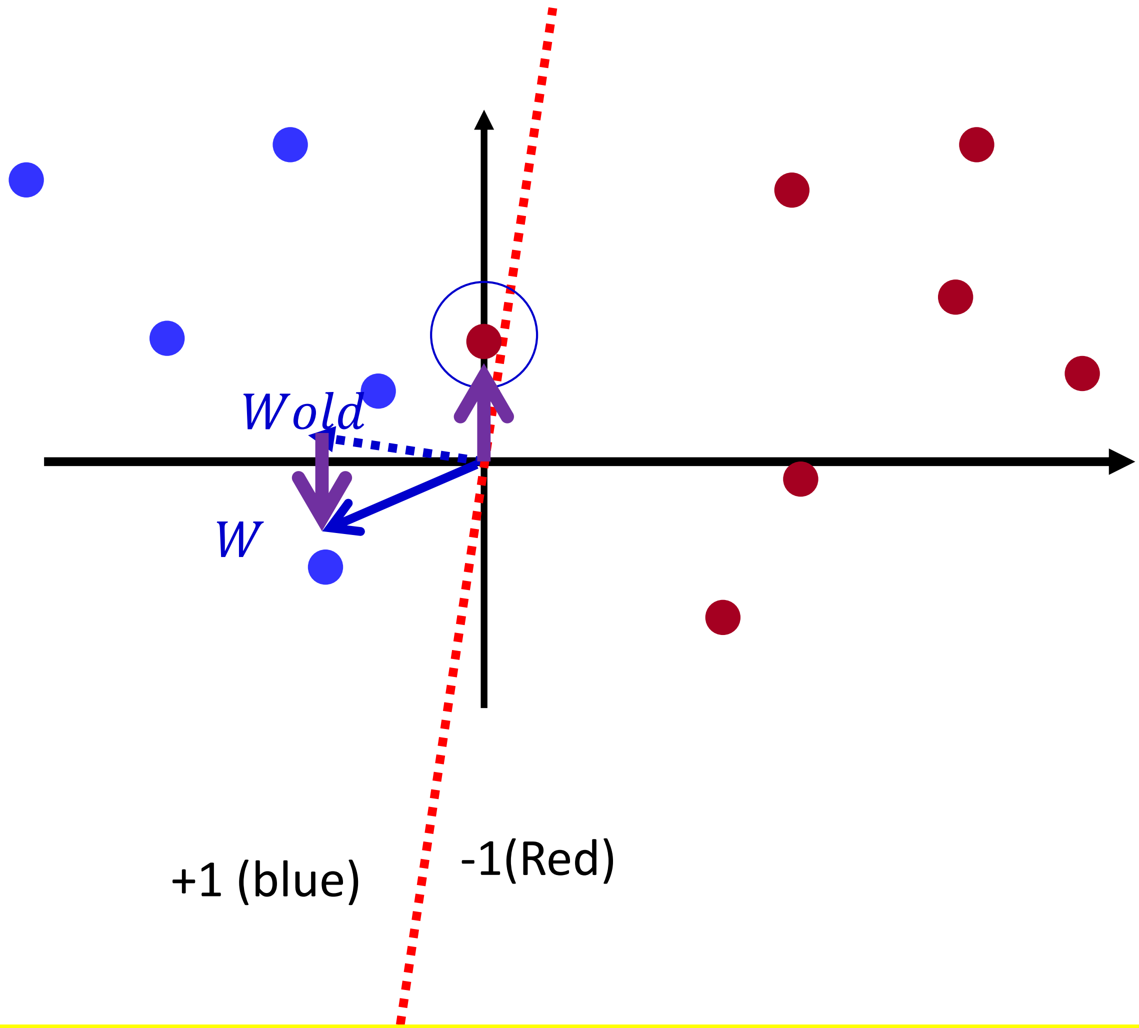
Perceptron Algorithm





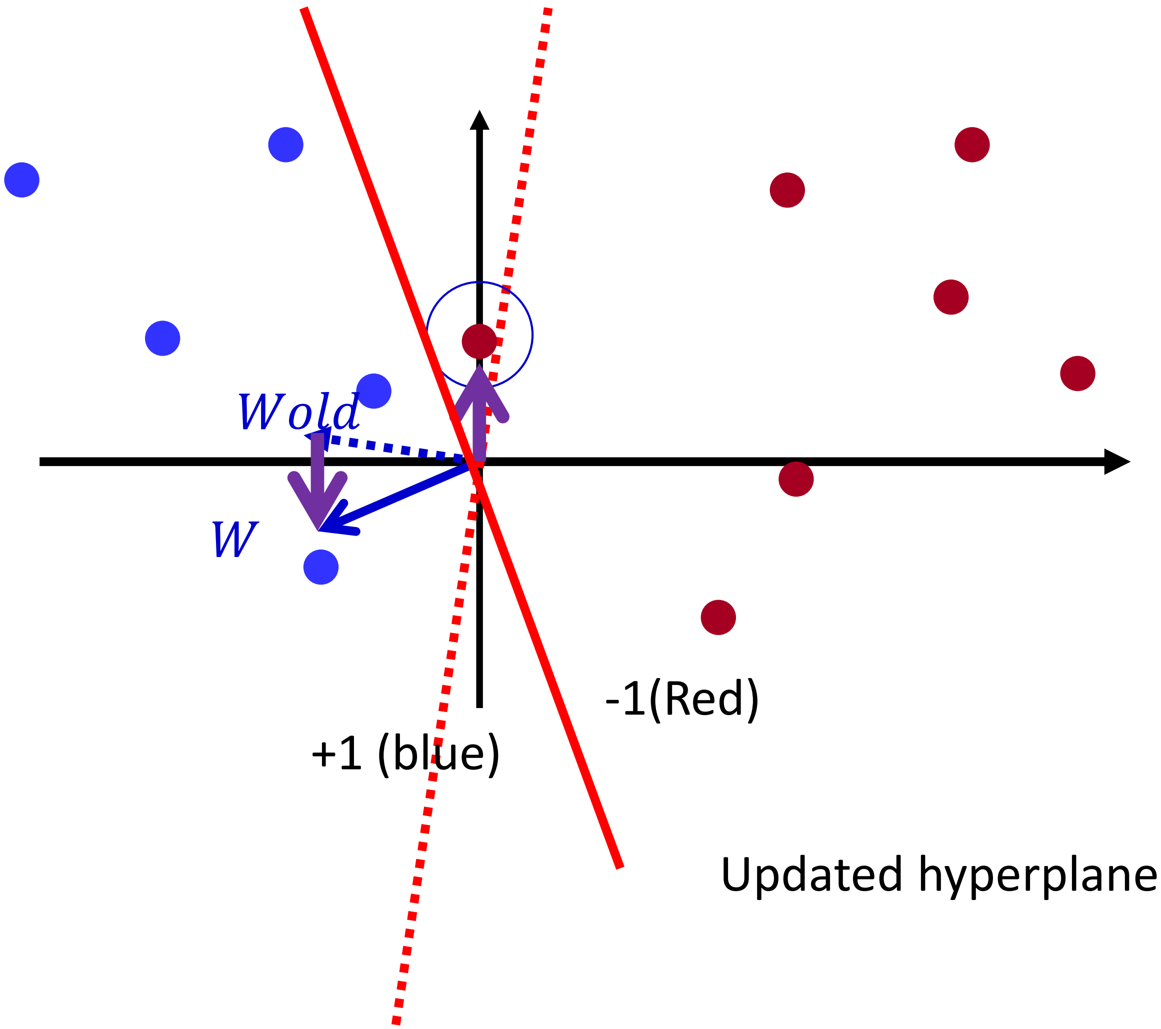
Perceptron Algorithm

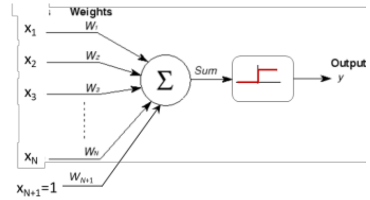




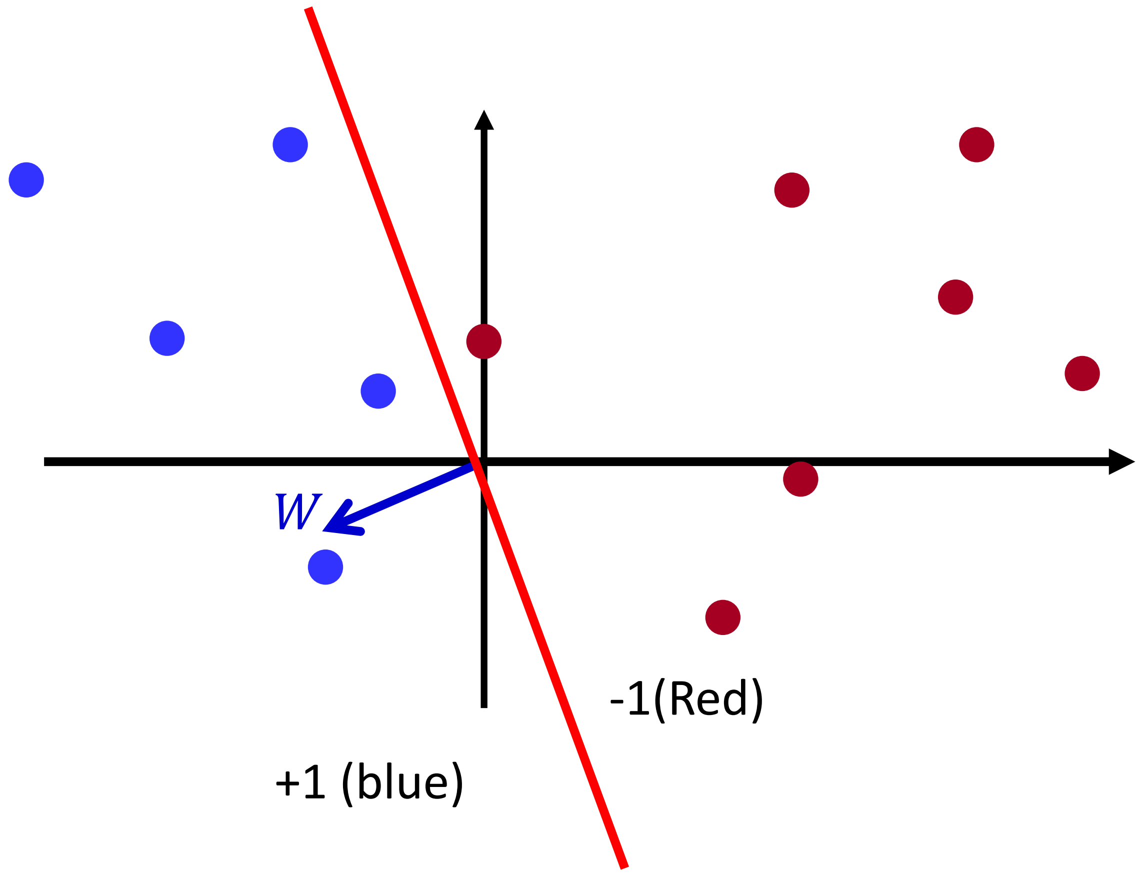
Misclassified *negative* instance, *subtract* it from W

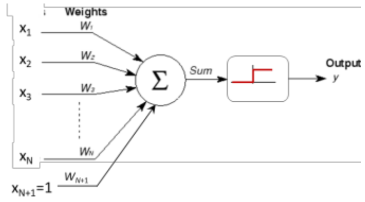
Perceptron Algorithm





Perceptron Algorithm



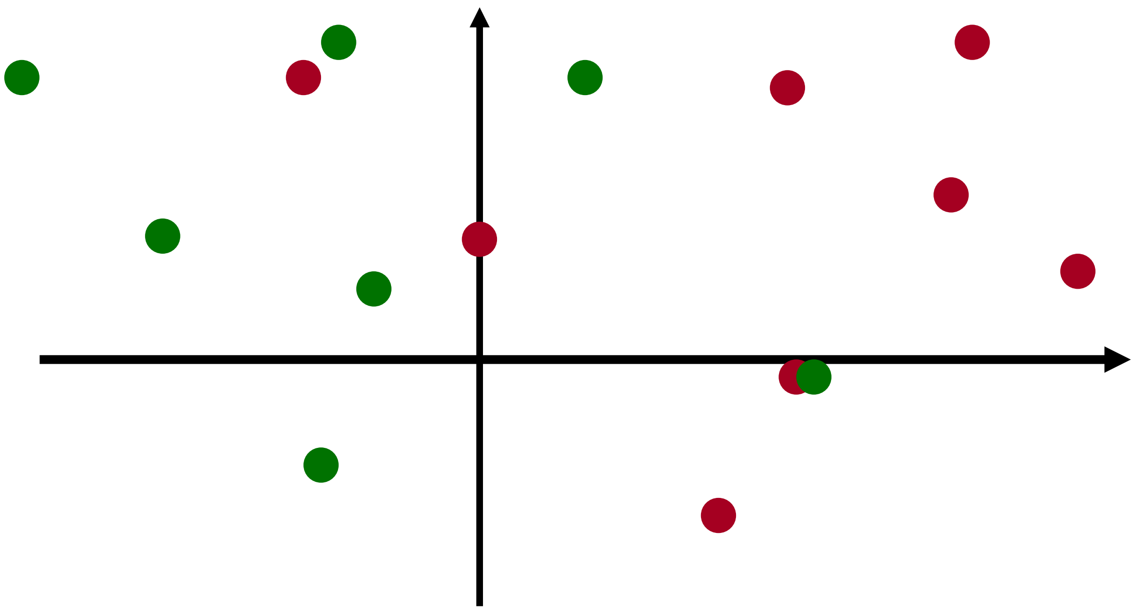


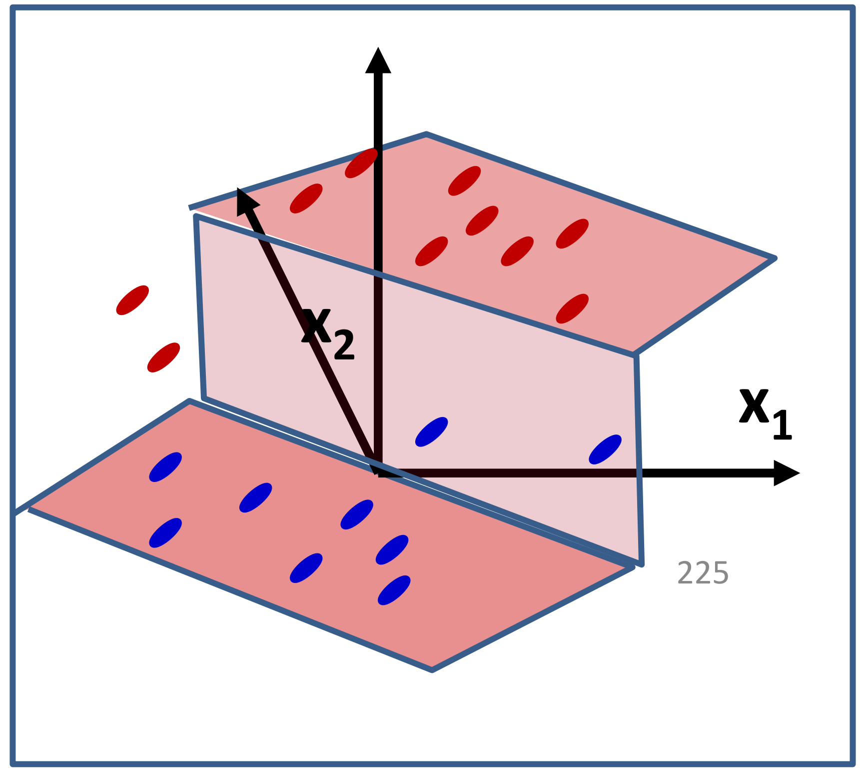
Perfect classification, no more updates

Convergence of Perceptron Algorithm

- Guaranteed to converge if classes are linearly separable
  - After no more than $\left(\frac{R}{\gamma}\right)^2$ misclassifications
    - Specifically when W is initialized to 0
  - R is length of longest training point
  - $\gamma$is the *best case* closest distance of a training point from the classifier
    - Same as the margin in an SVM
  - Intuitively – takes many increments of size γ to undo an error resulting from a step of size R

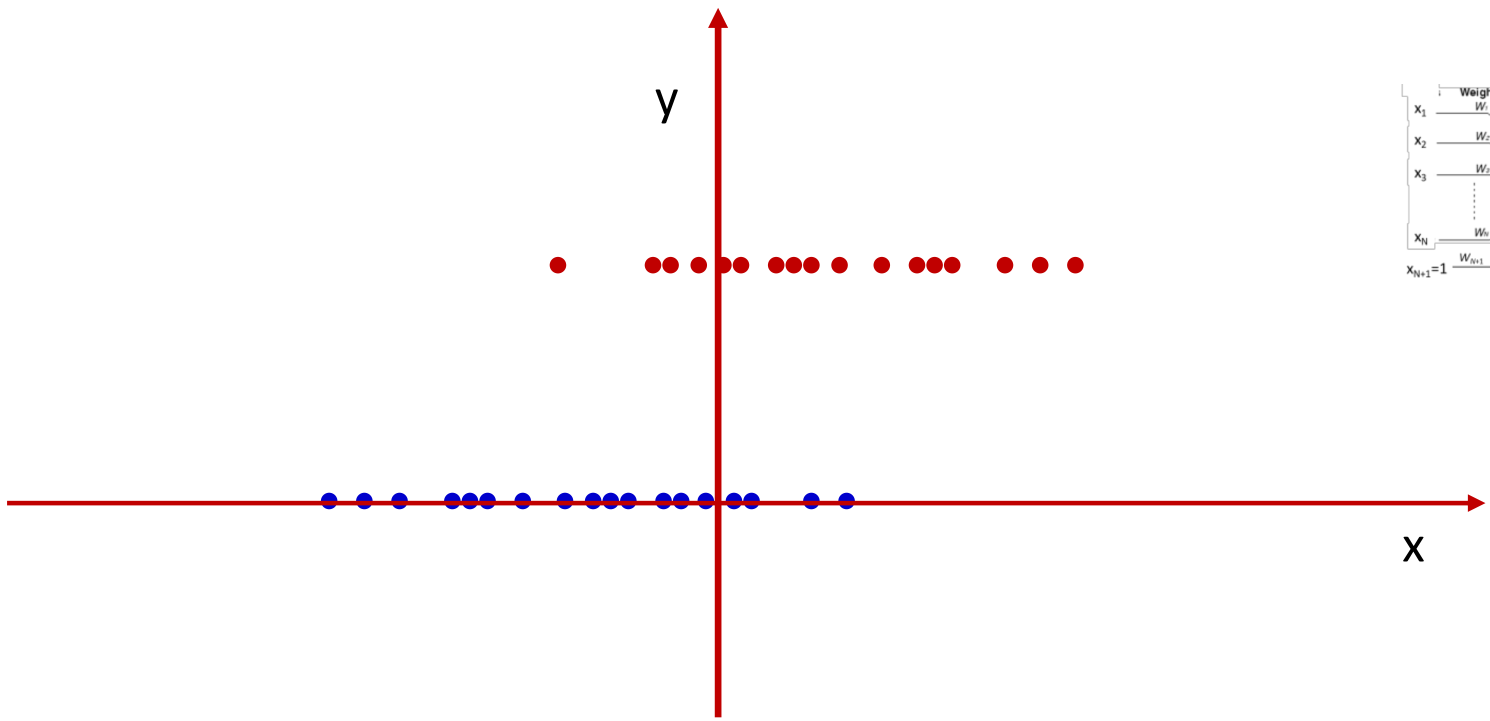
In reality: Trivial linear example

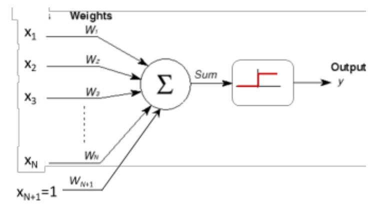




- Two-dimensional example
  - Blue dots (on the floor) on the "red" side
  - Red dots (suspended at Y=1) on the "blue" side
  - No line will cleanly separate the two colors

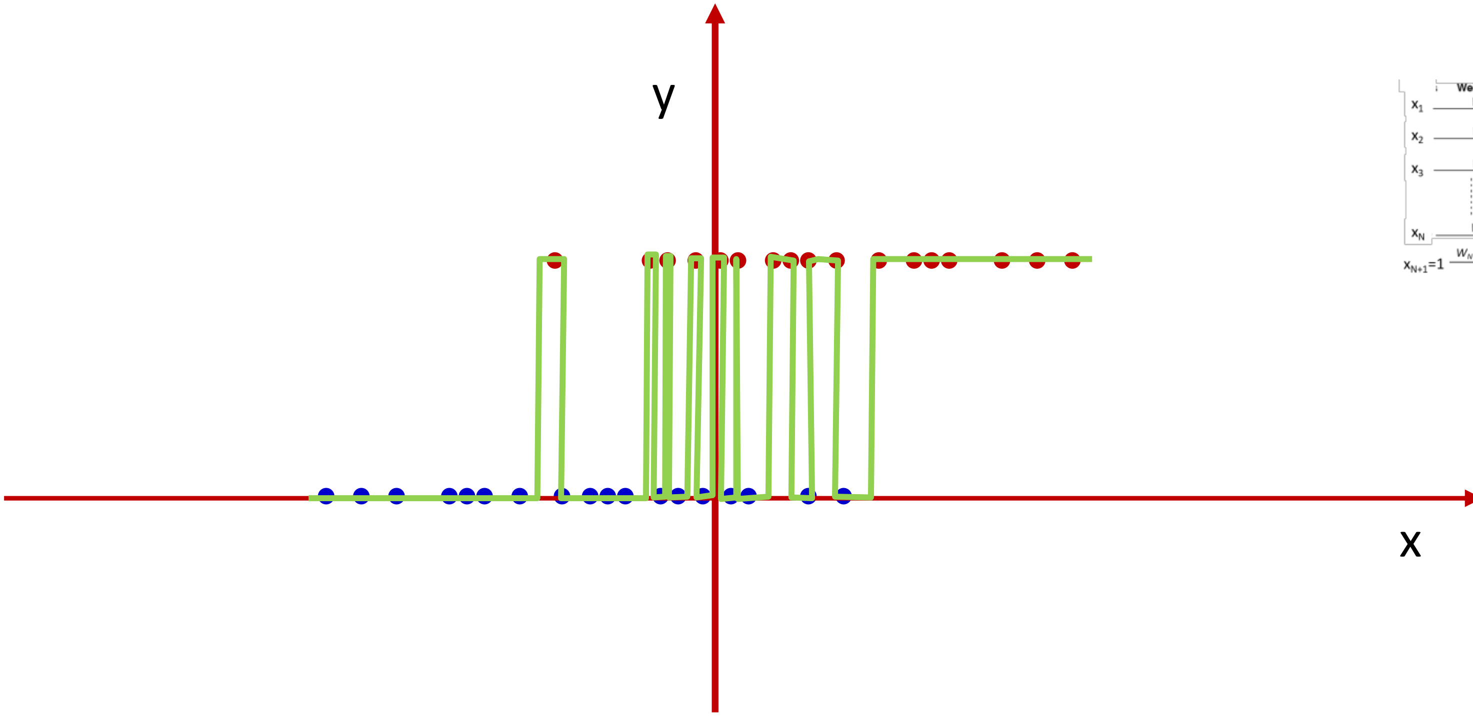
Non-linearly separable data: 1-D example

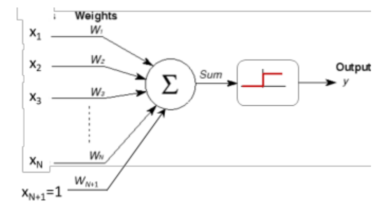




- One-dimensional example for visualization
  - All (red) dots at Y=1 represent instances of class Y=1
  - All (blue) dots at Y=0 are from class Y=0
  - The data are not linearly separable
    - In this 1-D example, a linear separator is a threshold
    - No threshold will cleanly separate red and blue dots

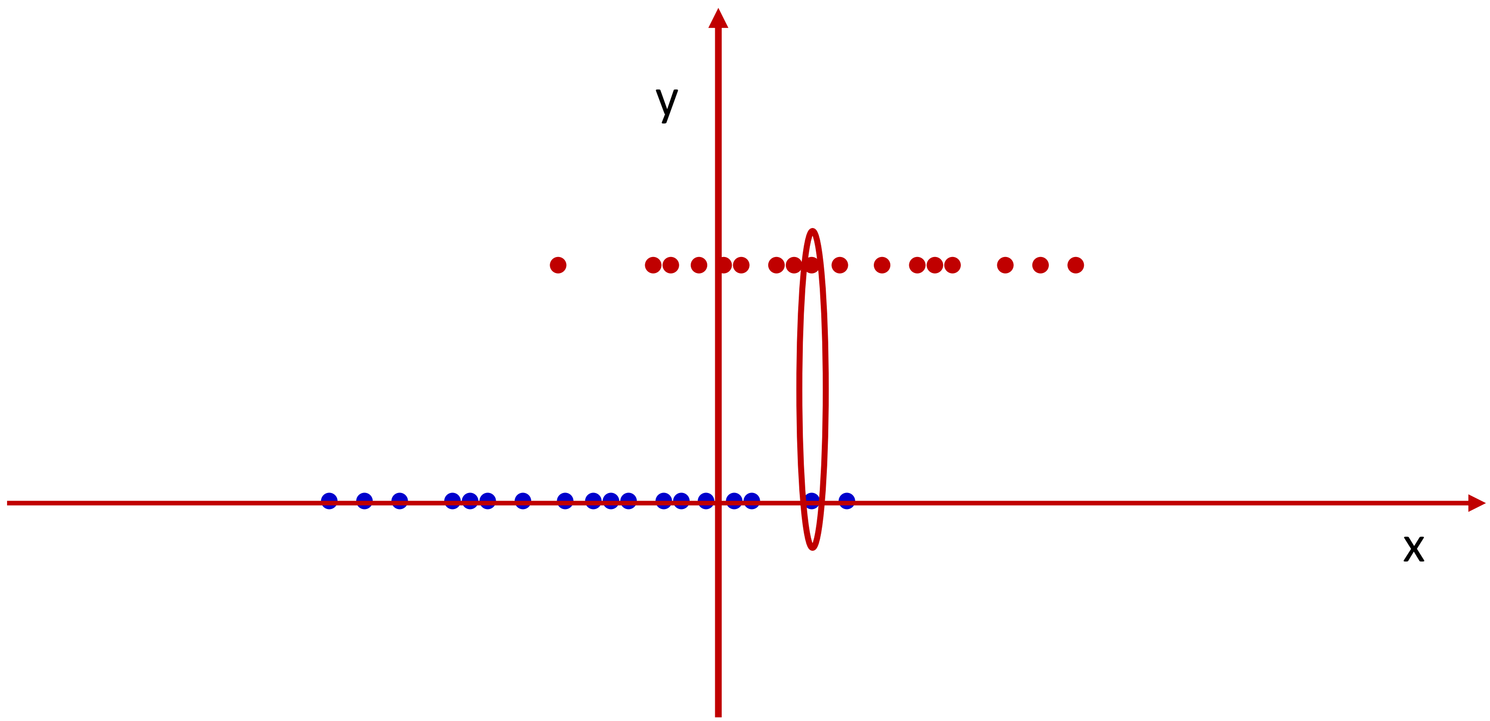
Undesired Function





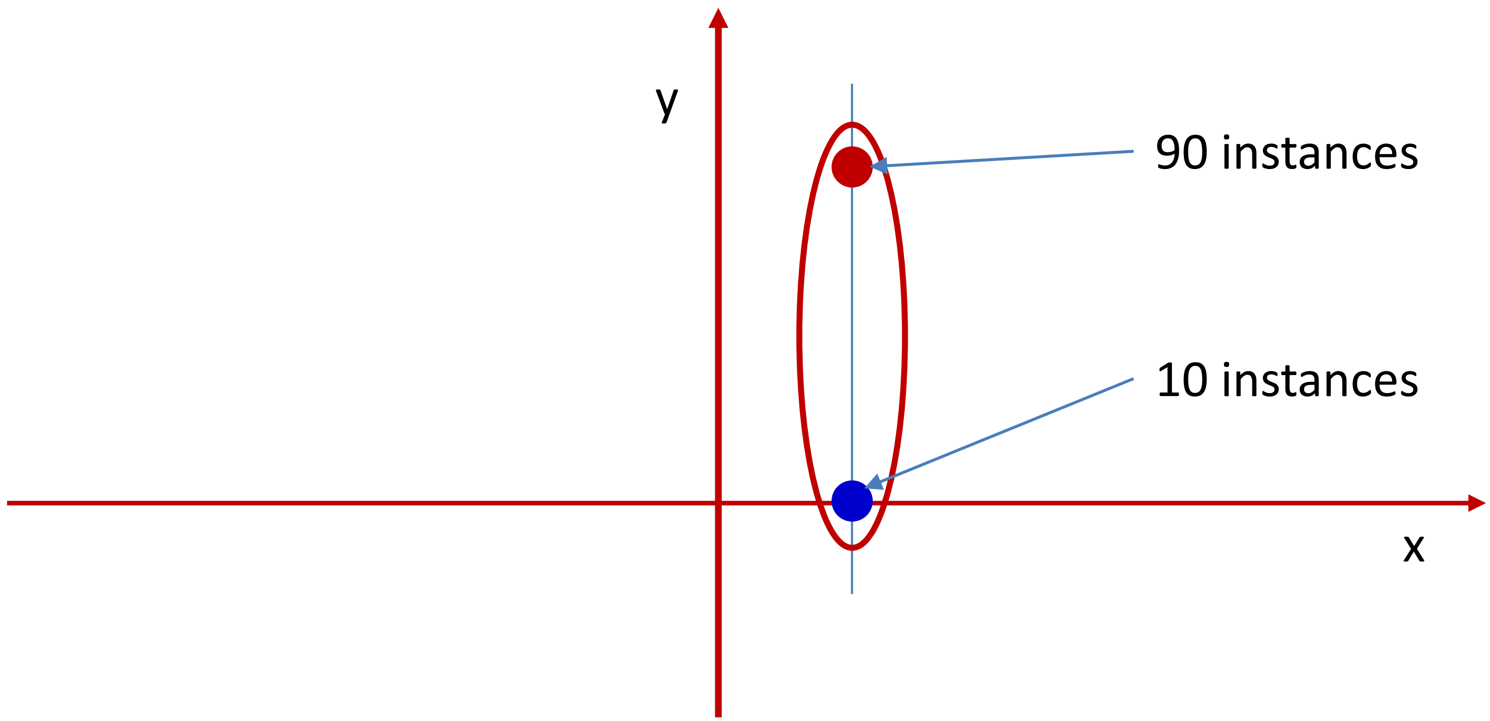
- One-dimensional example for visualization
  - All (red) dots at Y=1 represent instances of class Y=1
  - All (blue) dots at Y=0 are from class Y=0
  - The data are not linearly separable
    - In this 1-D example, a linear separator is a threshold
    - No threshold will cleanly separate red and blue dots

What if?



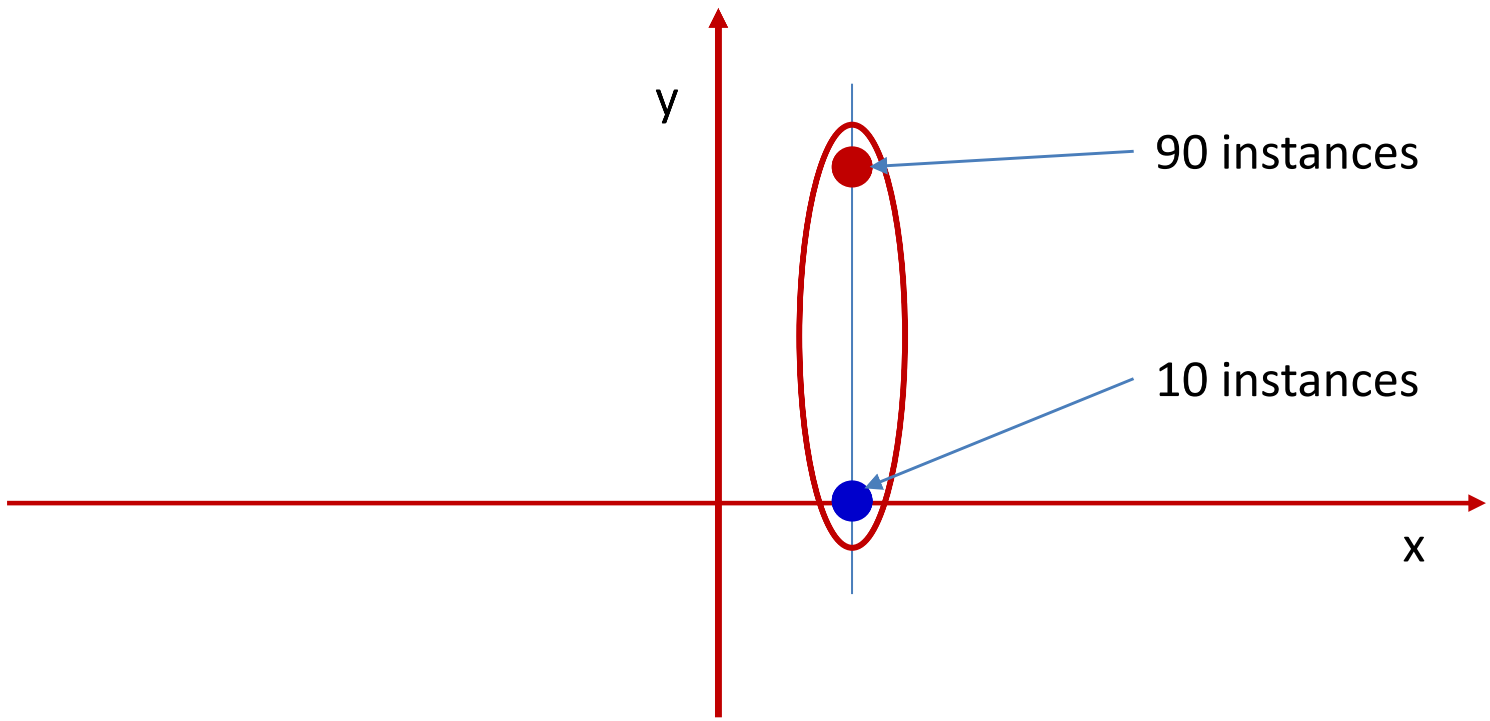
- One-dimensional example for visualization
  - All (red) dots at Y=1 represent instances of class Y=1
  - All (blue) dots at Y=0 are from class Y=0
  - The data are not linearly separable
    - In this 1-D example, a linear separator is a threshold
    - No threshold will cleanly separate red and blue dots

What if?



- What must the value of the function be at this X?
  - 1 because red dominates?
  - 0.9 : The average?

What if?

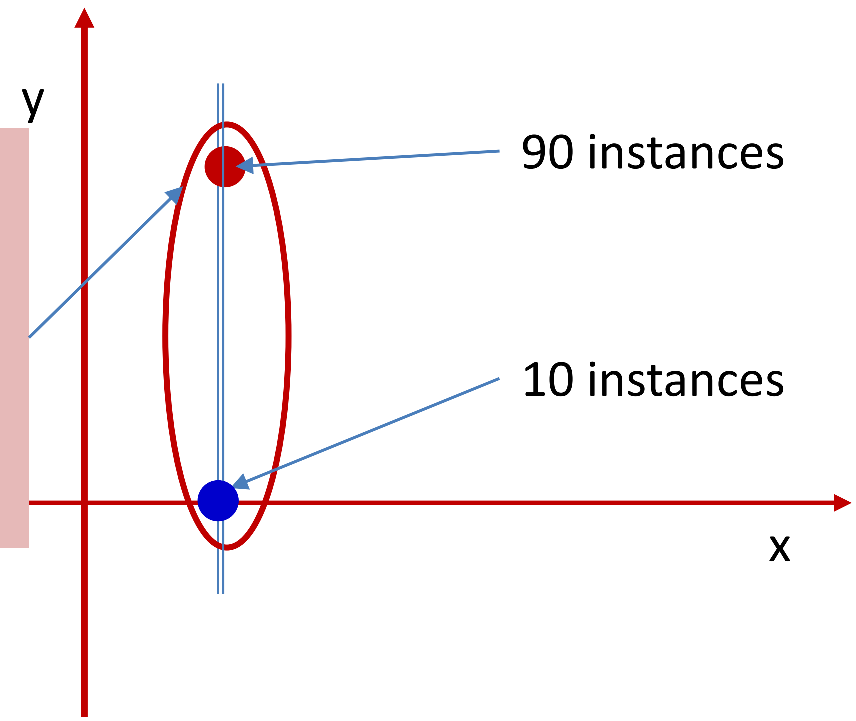


- What must the value of the function be at this X?
  - 1 because red dominates?
  - 0.9 : The average?

Estimate: $\approx P(1|X)$

Potentially much more useful than a simple 1/0 decision
Also, potentially more realistic

What if?



Should an infinitesimal nudge of the red dot change the function estimate entirely?

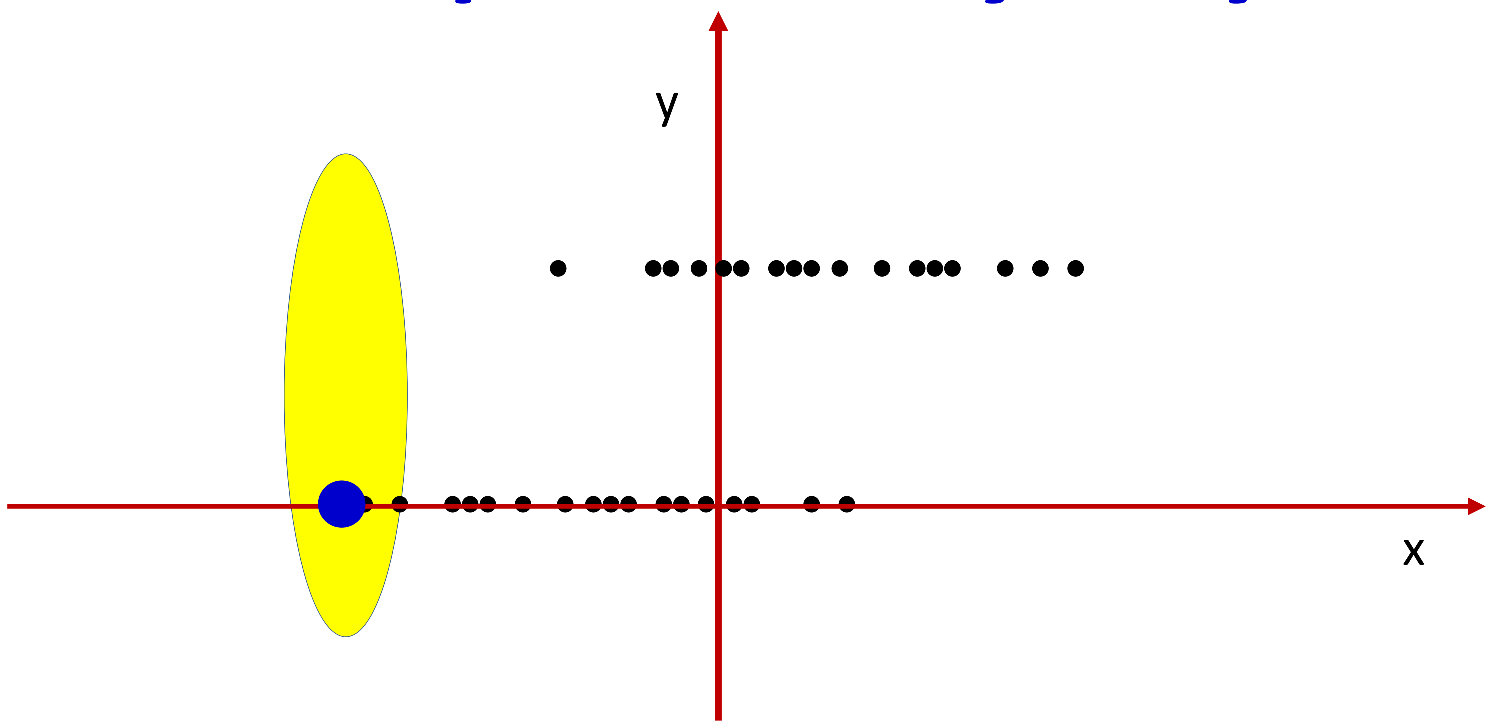
If not, how do we estimate $P(1|X)$? (since the positions of the red and blue X Values are different)

- What must the value of the function be at this X?
  - 1 because red dominates?
  - 0.9 : The average?

Estimate: $\approx P(1|X)$

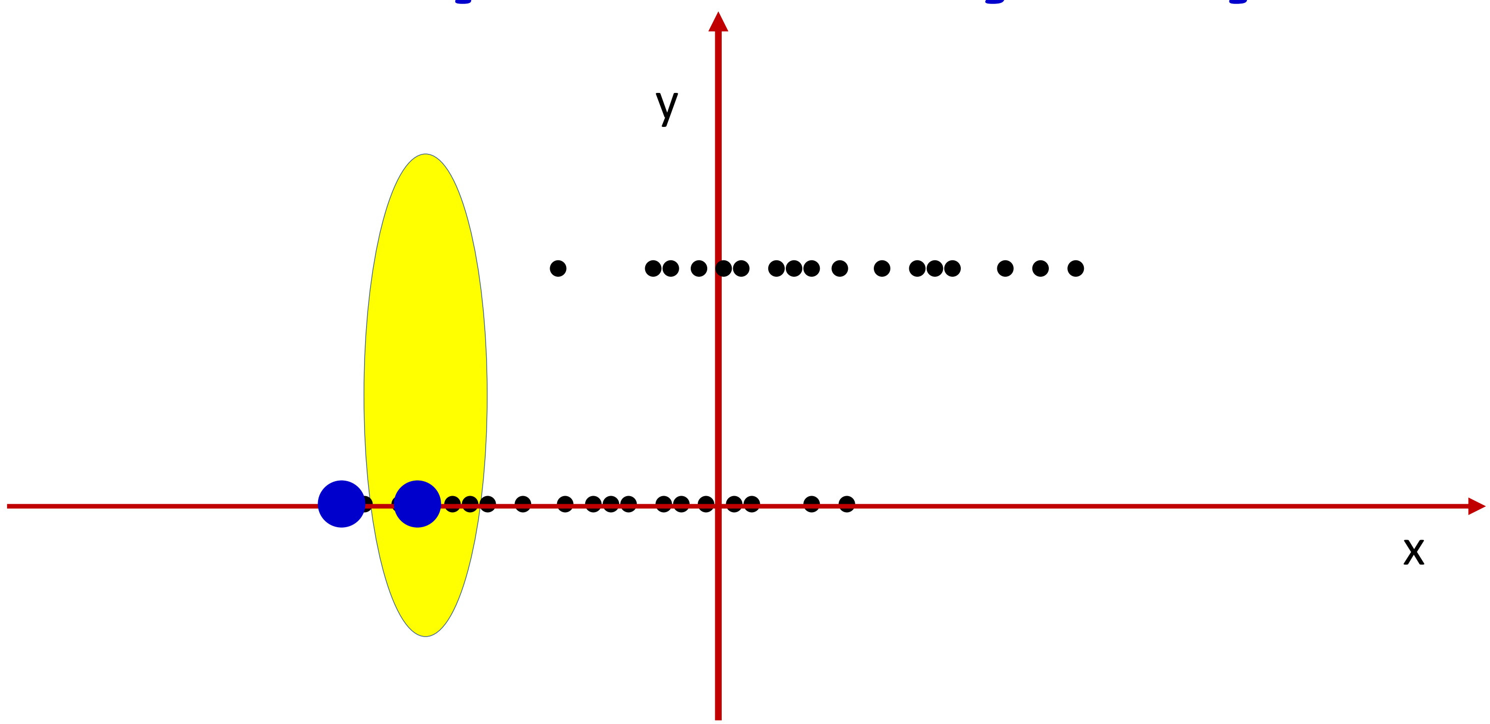
Potentially much more useful than a simple 1/0 decision
Also, potentially more realistic

The *probability* of y=1



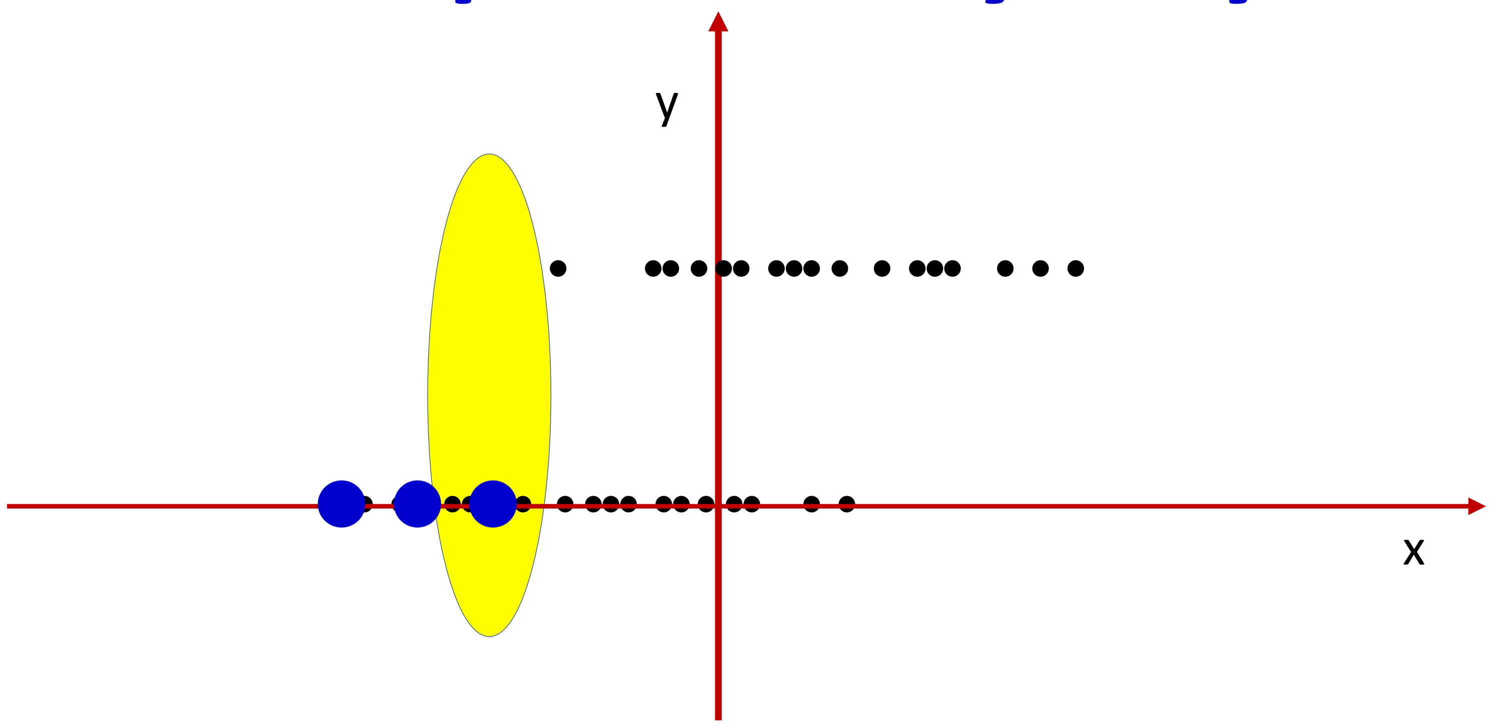
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of Y=1 at that point

The *probability* of y=1



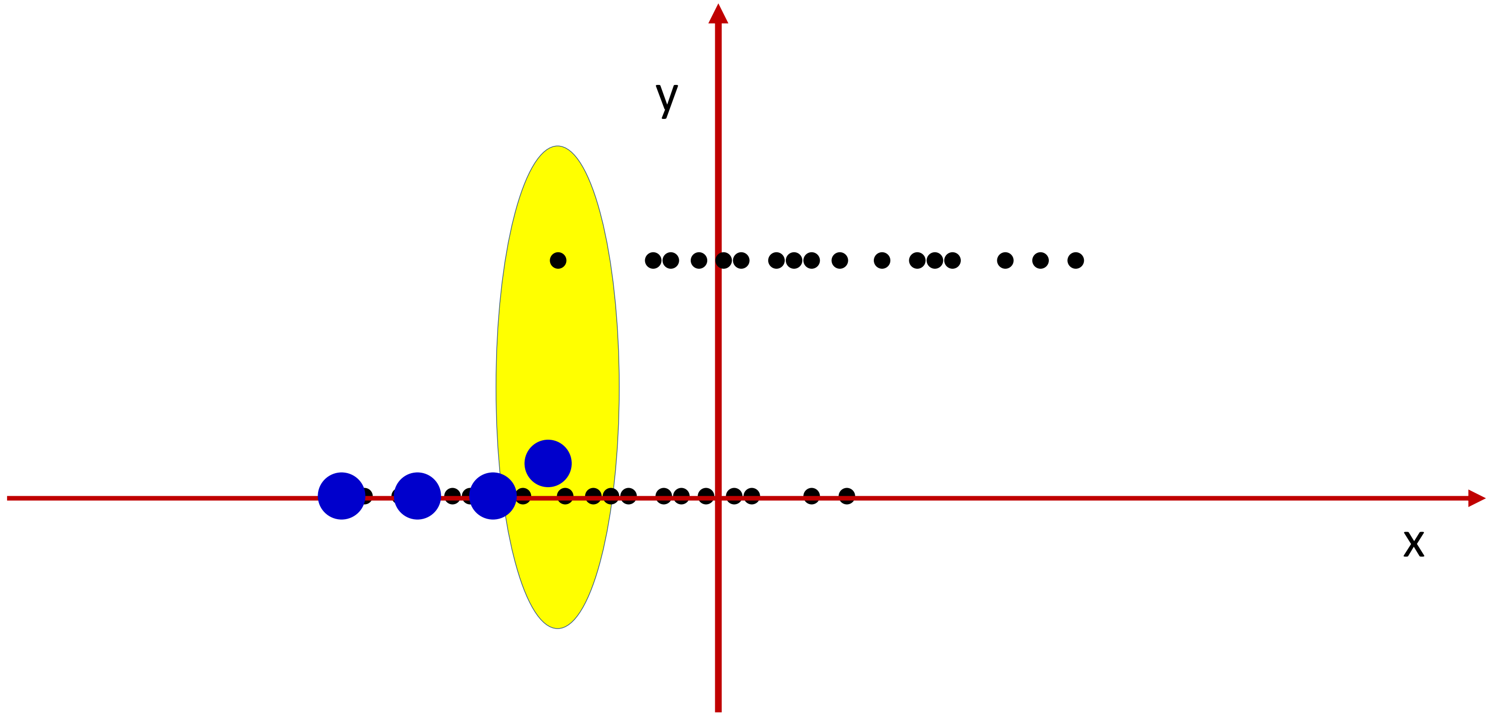
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



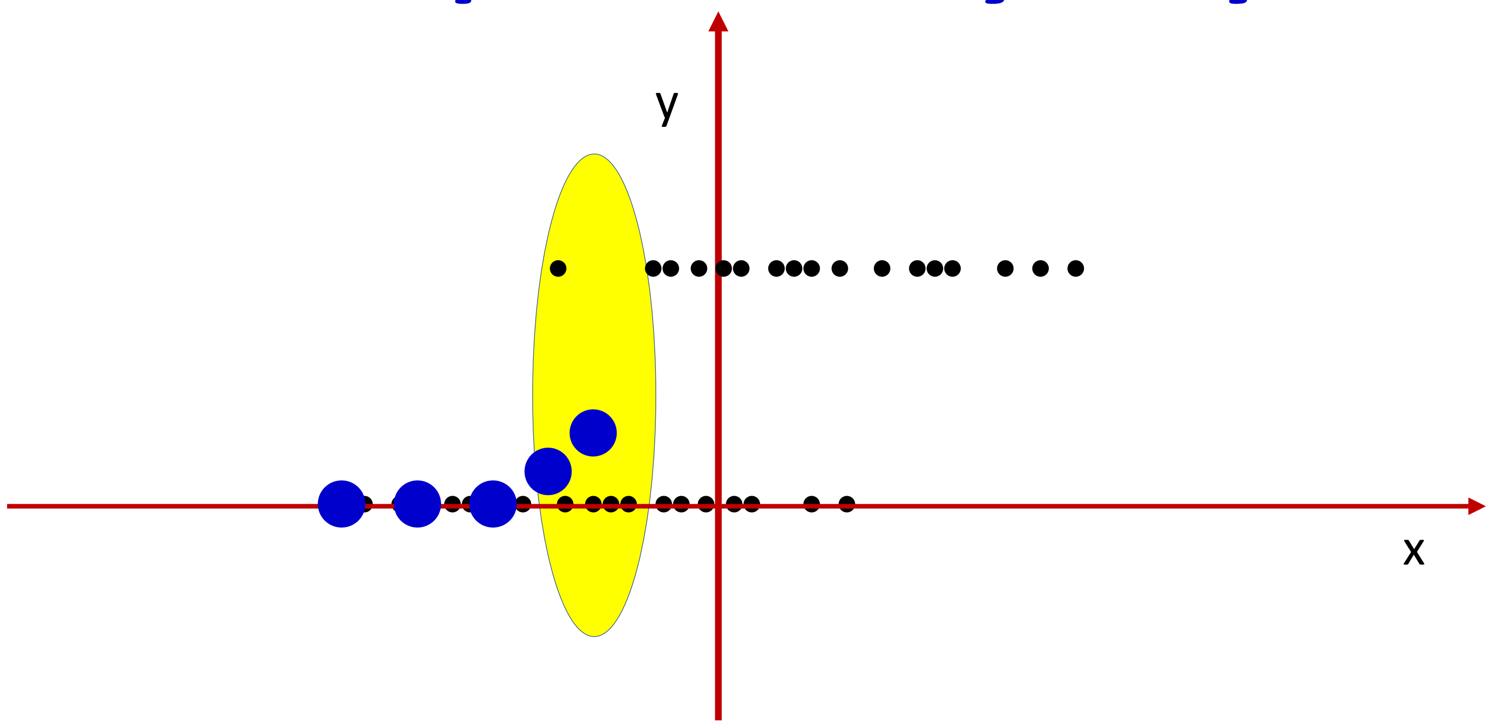
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



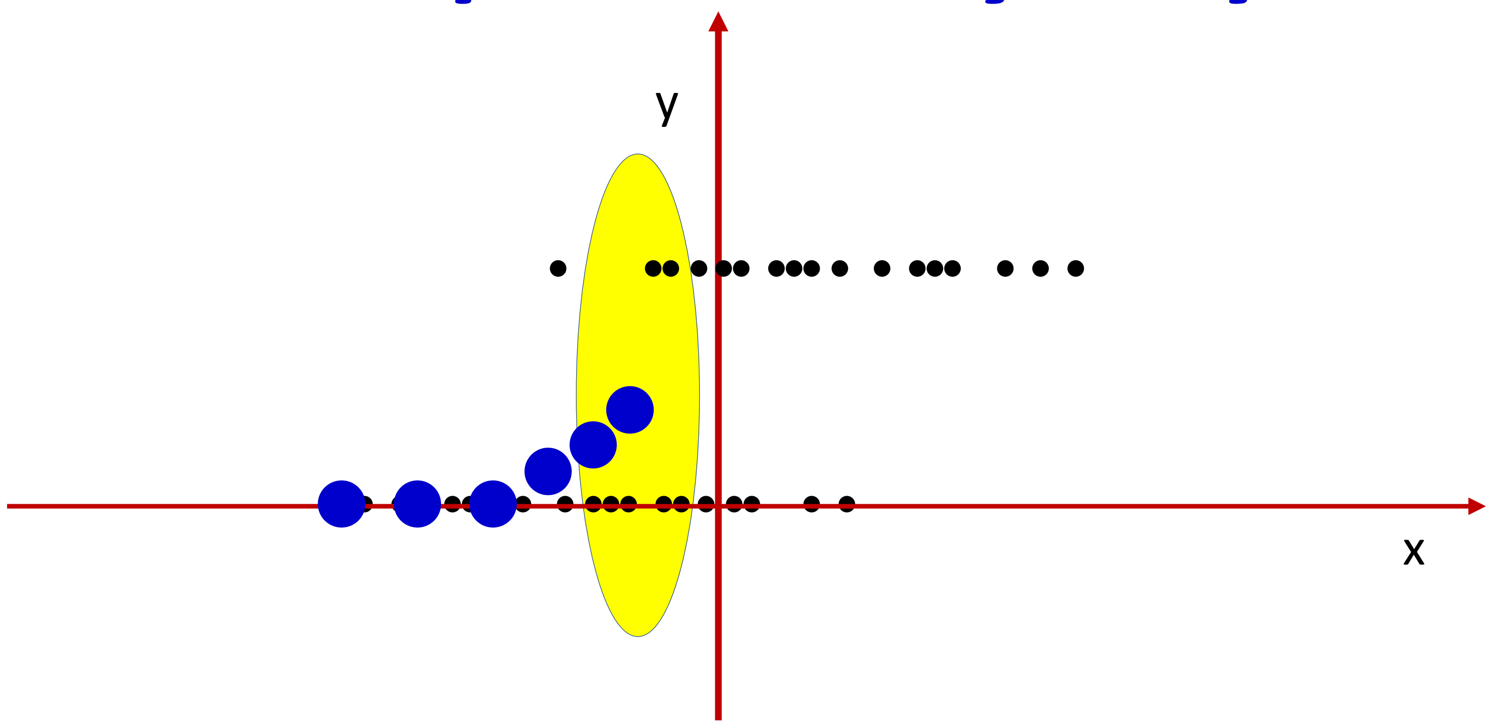
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



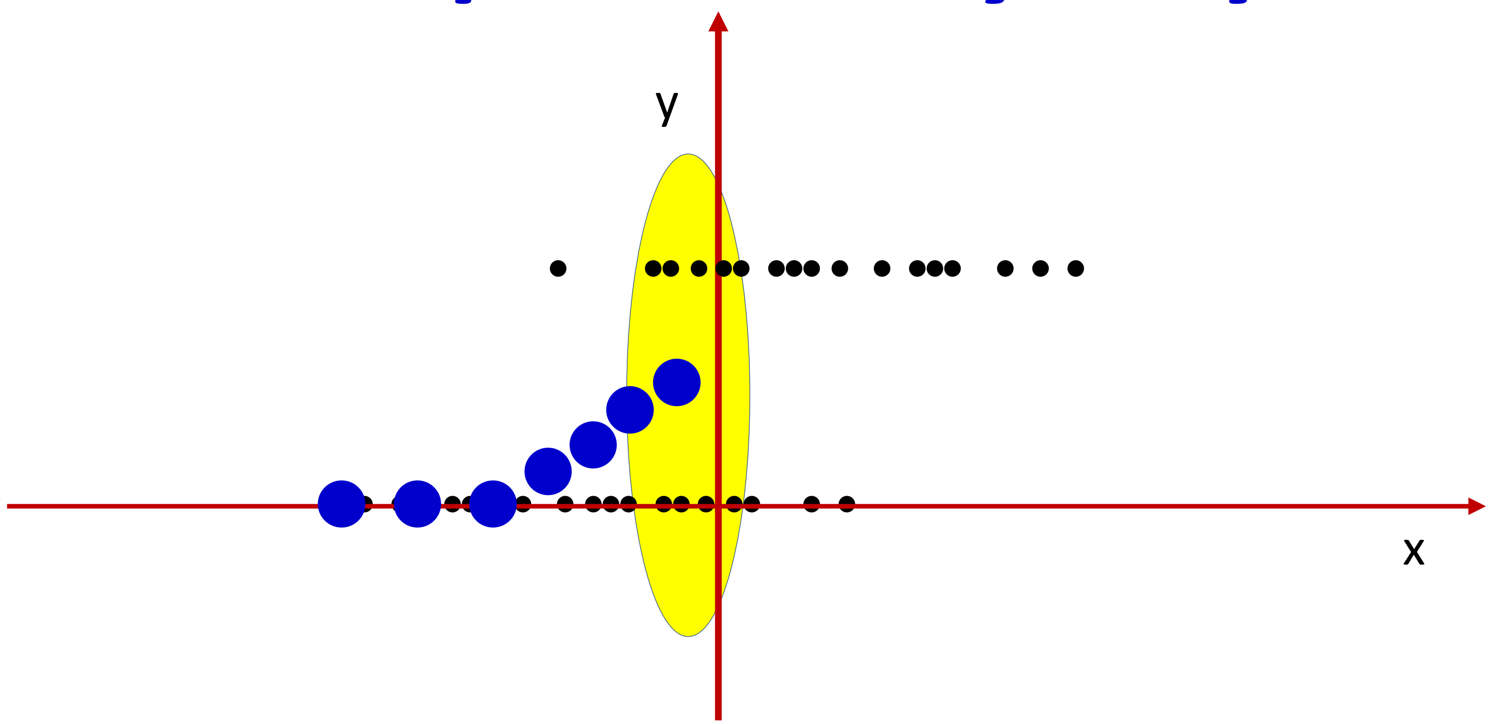
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



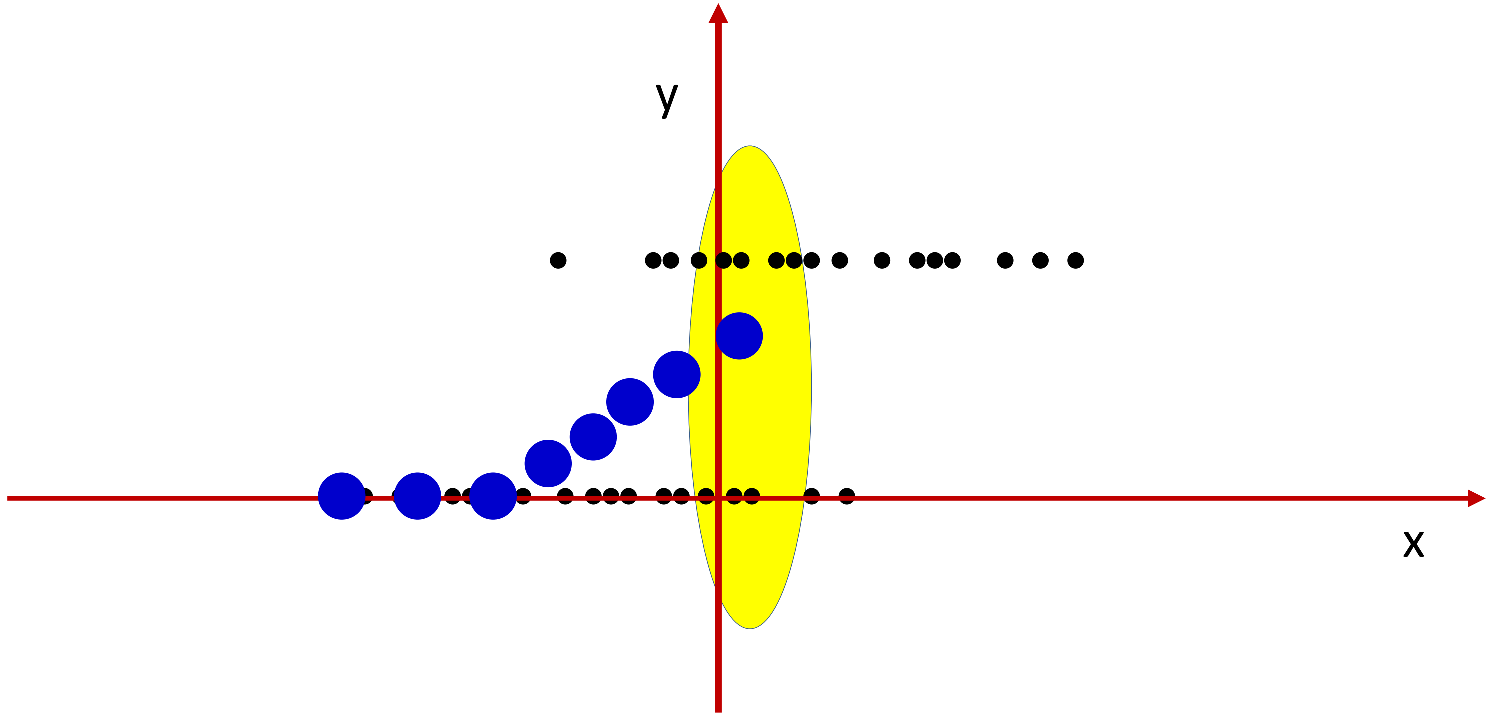
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



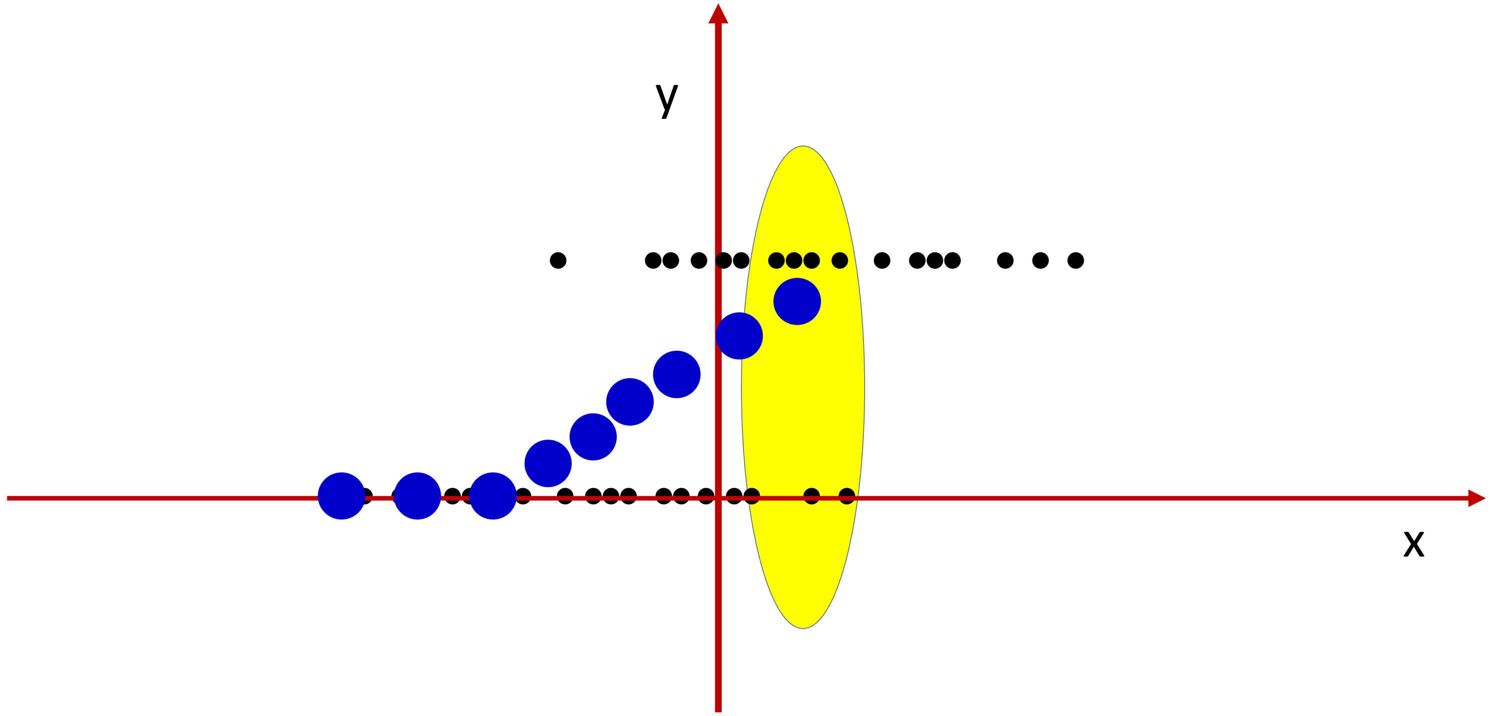
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



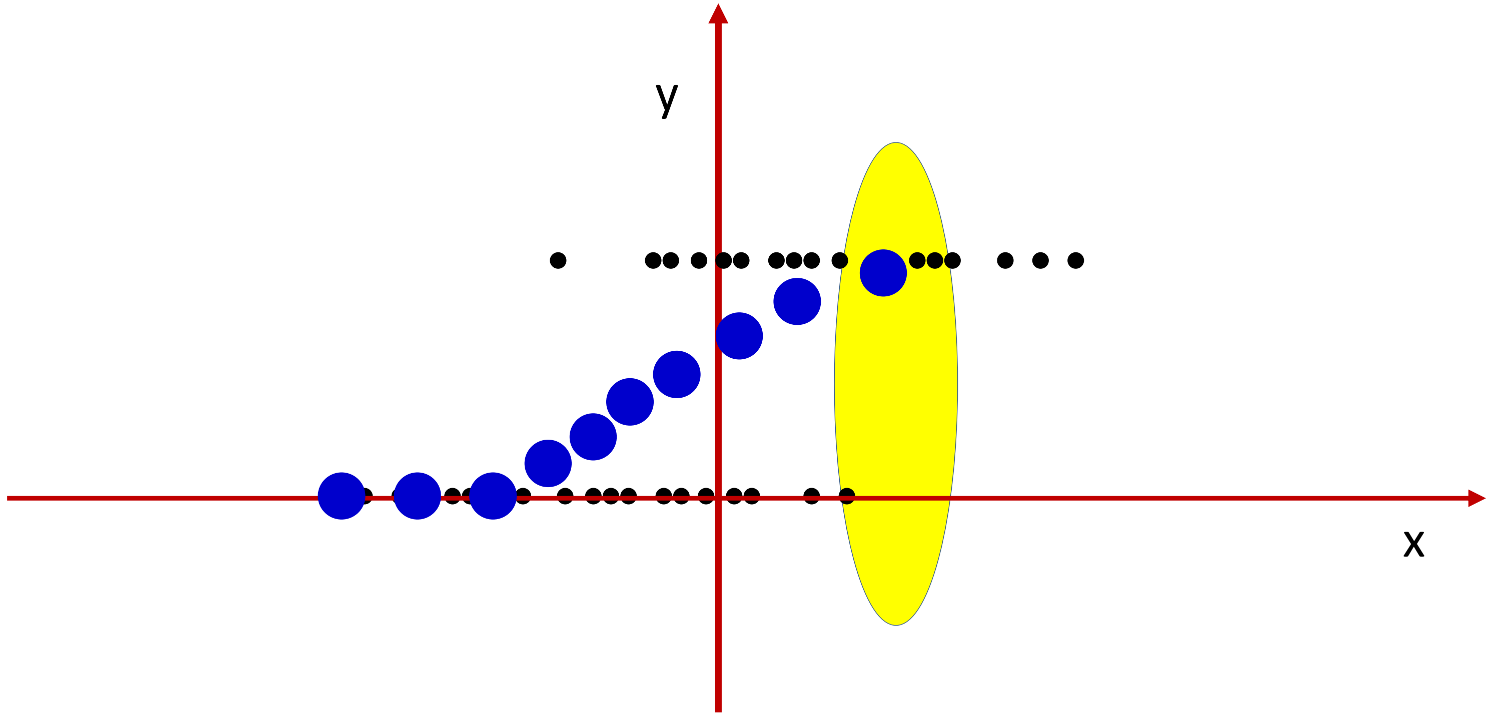
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



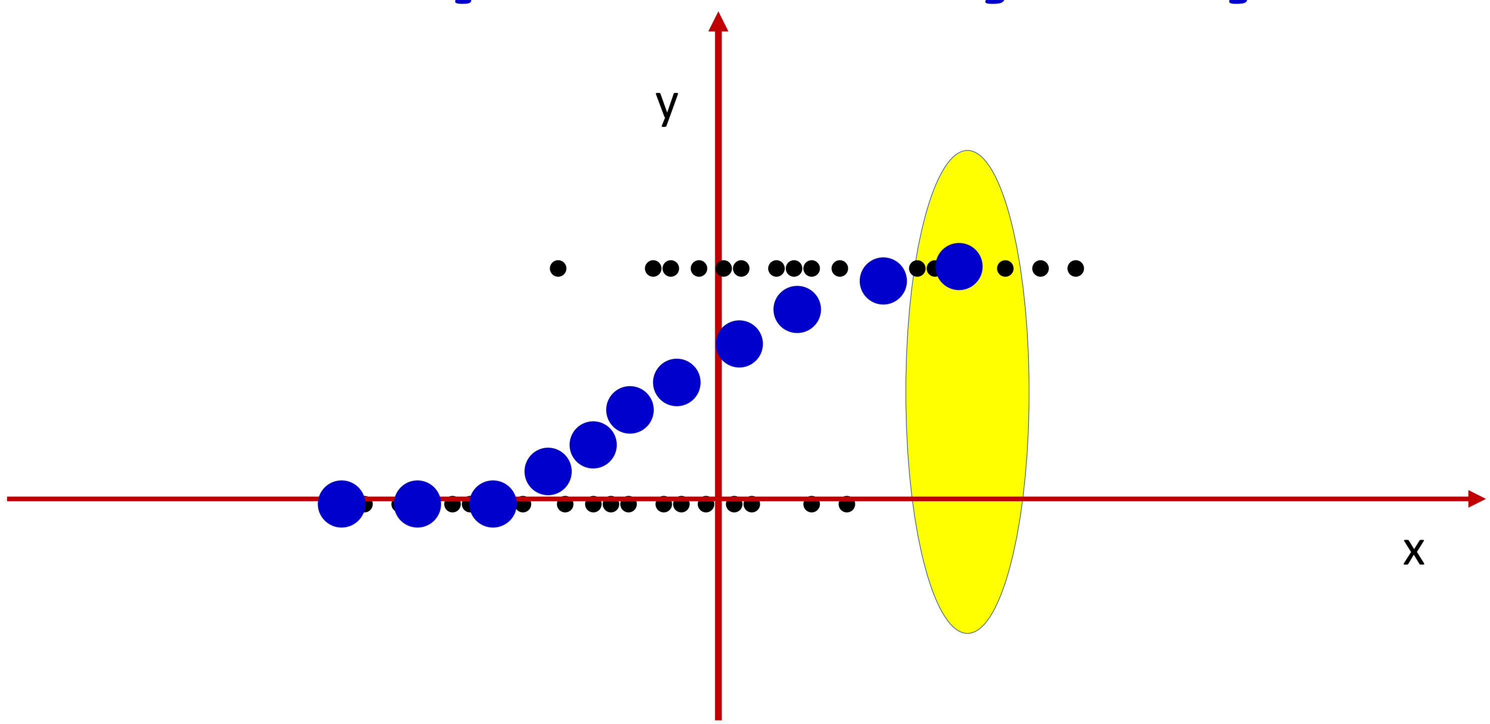
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



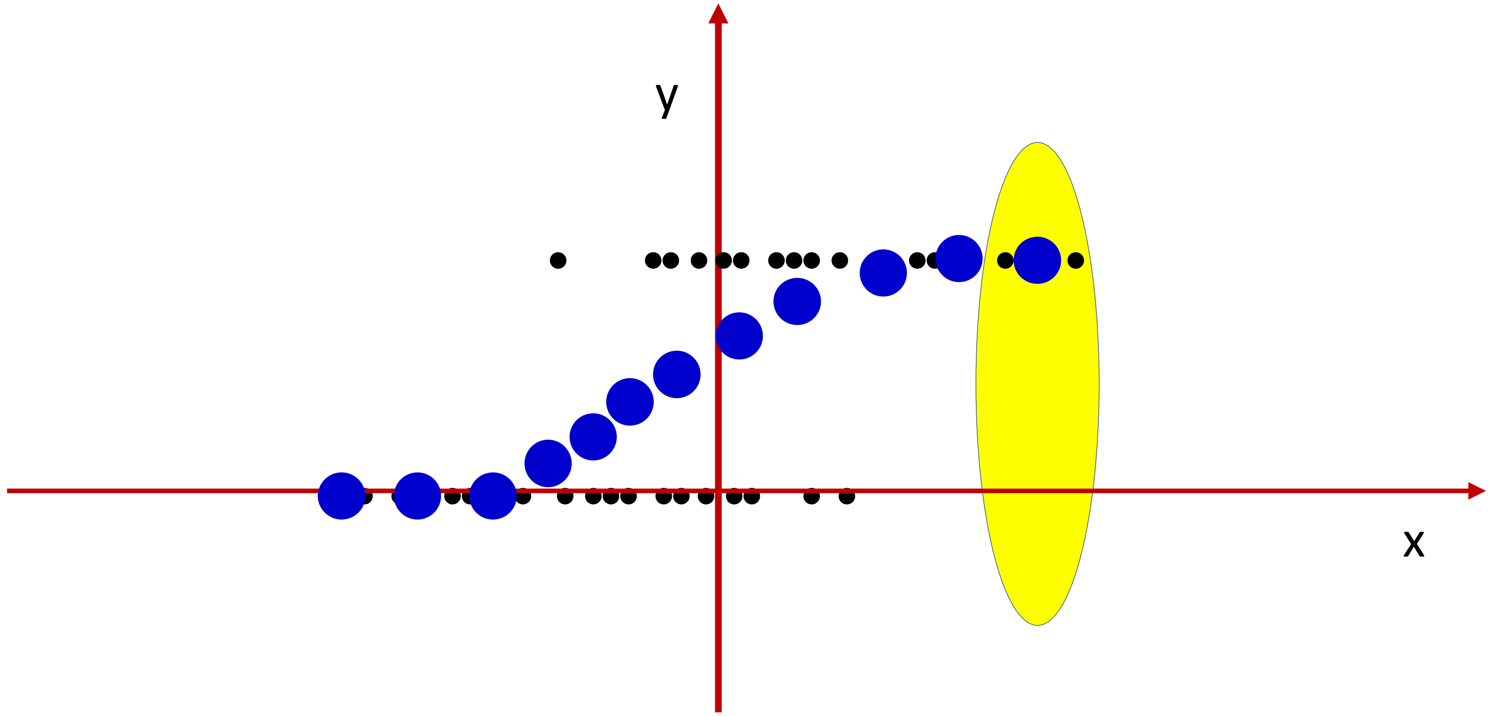
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



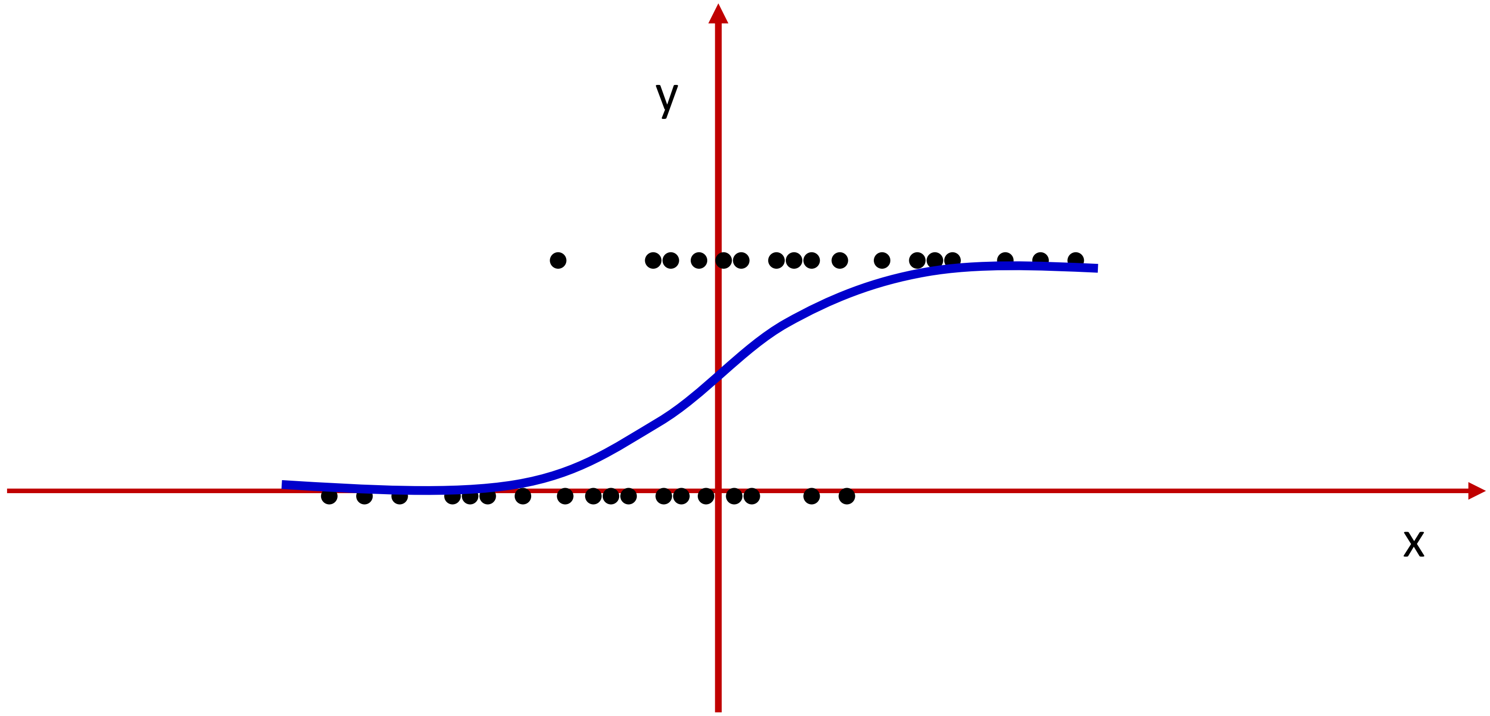
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



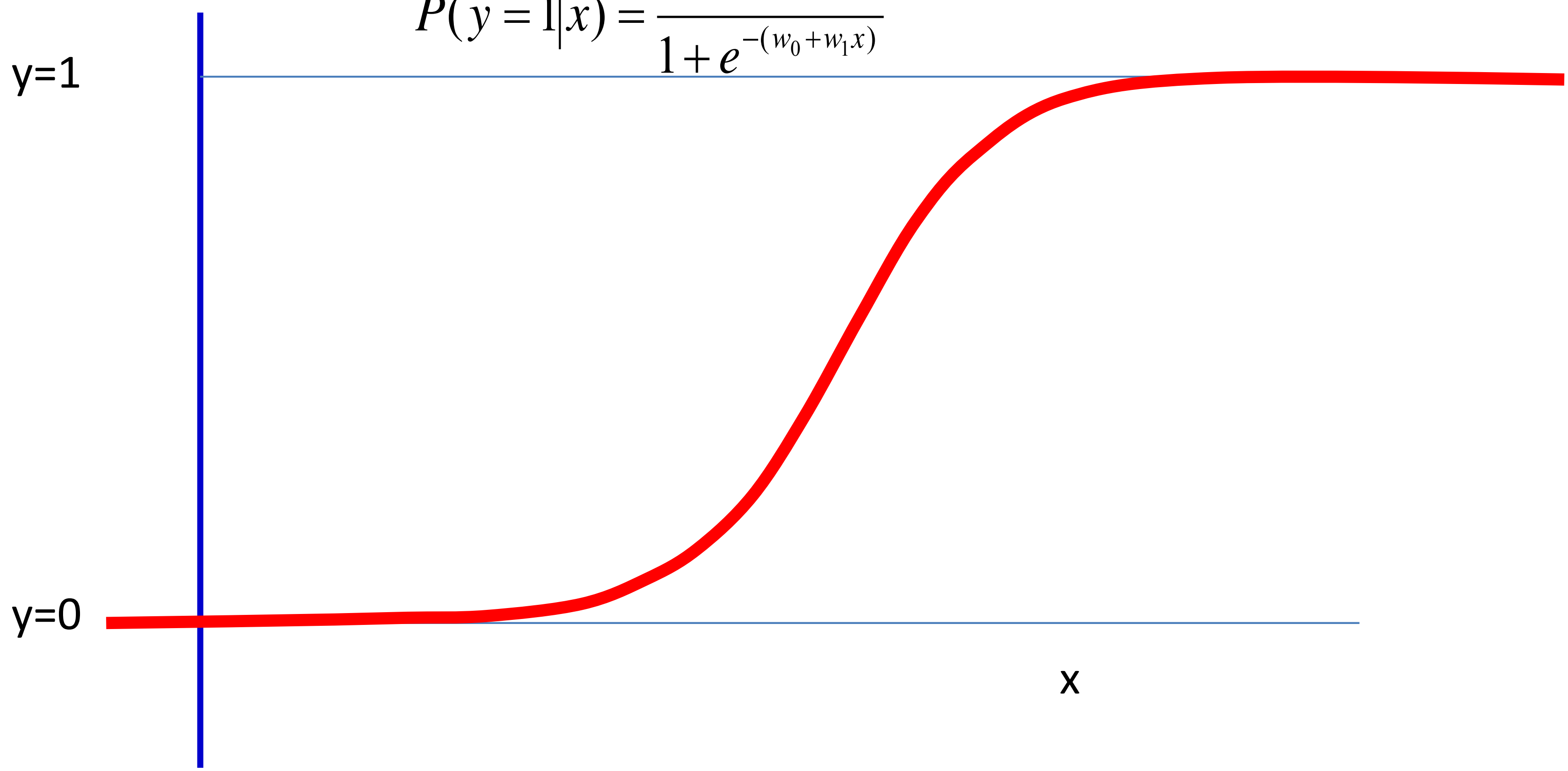
- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

The *probability* of y=1



- Consider this differently: at each point look at a small window around that point
- Plot the average value within the window
  - This is an approximation of the *probability* of 1 at that point

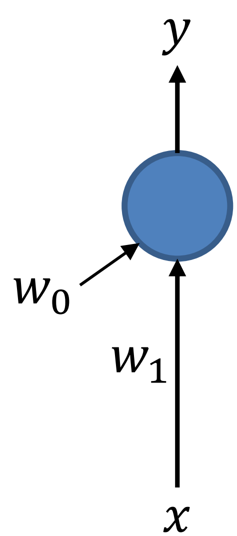
The logistic regression model



$$P(y=1|x)=\frac{1}{1+e^{-(w_0+w_1 x)}}$$

- Class 1 becomes increasingly probable going left to right
  - Very typical in many problems

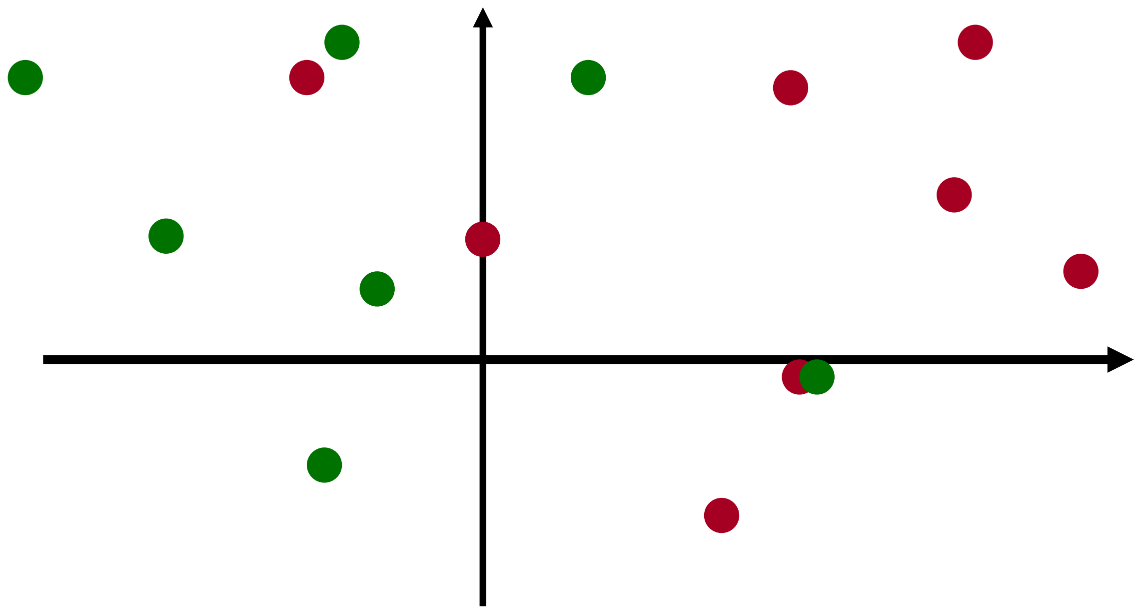
The logistic perceptron

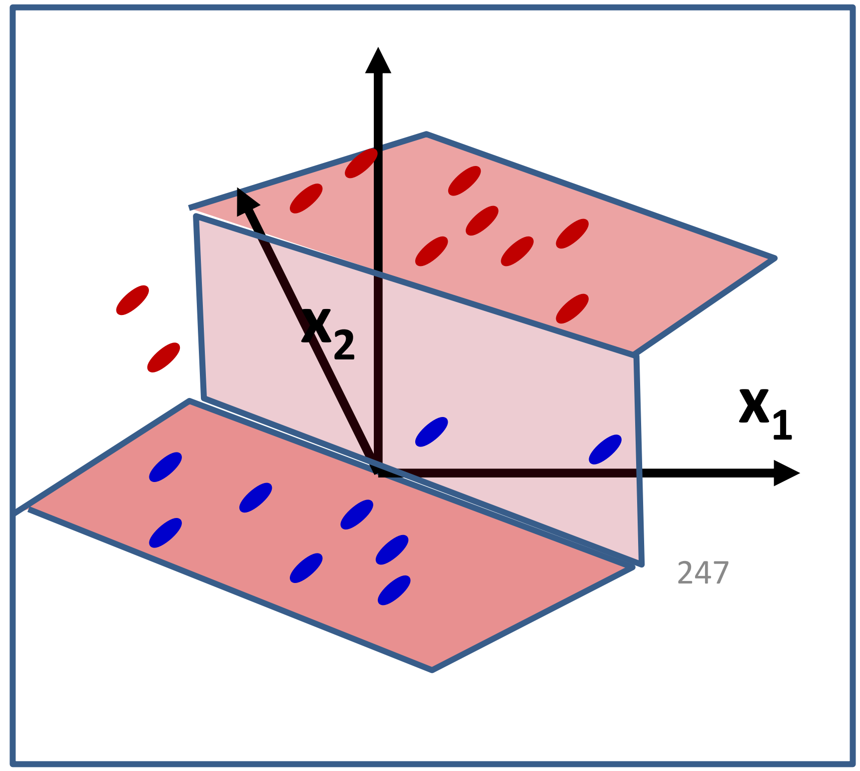


$$P(y|x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

- A sigmoid perceptron with a single input models the *a posteriori* probability of the class given the input

Non-linearly separable data

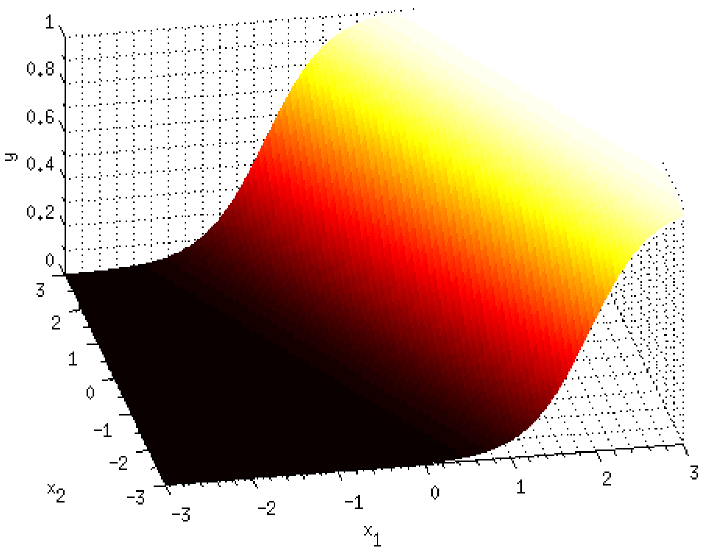


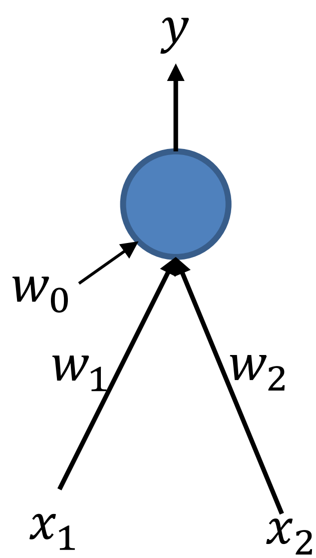


- Two-dimensional example
  - Blue dots (on the floor) on the "red" side
  - Red dots (suspended at Y=1) on the "blue" side
  - No line will cleanly separate the two colors

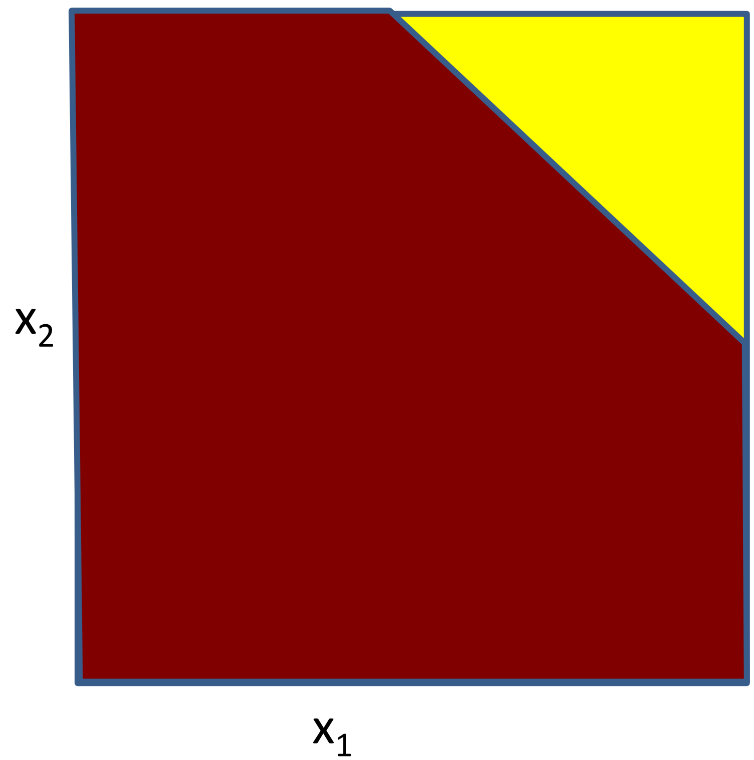
Logistic regression

$$P(Y=1|X) = \frac{1}{1+\exp\left(-\left(\sum_i w_i x_i + w_0\right)\right)}$$





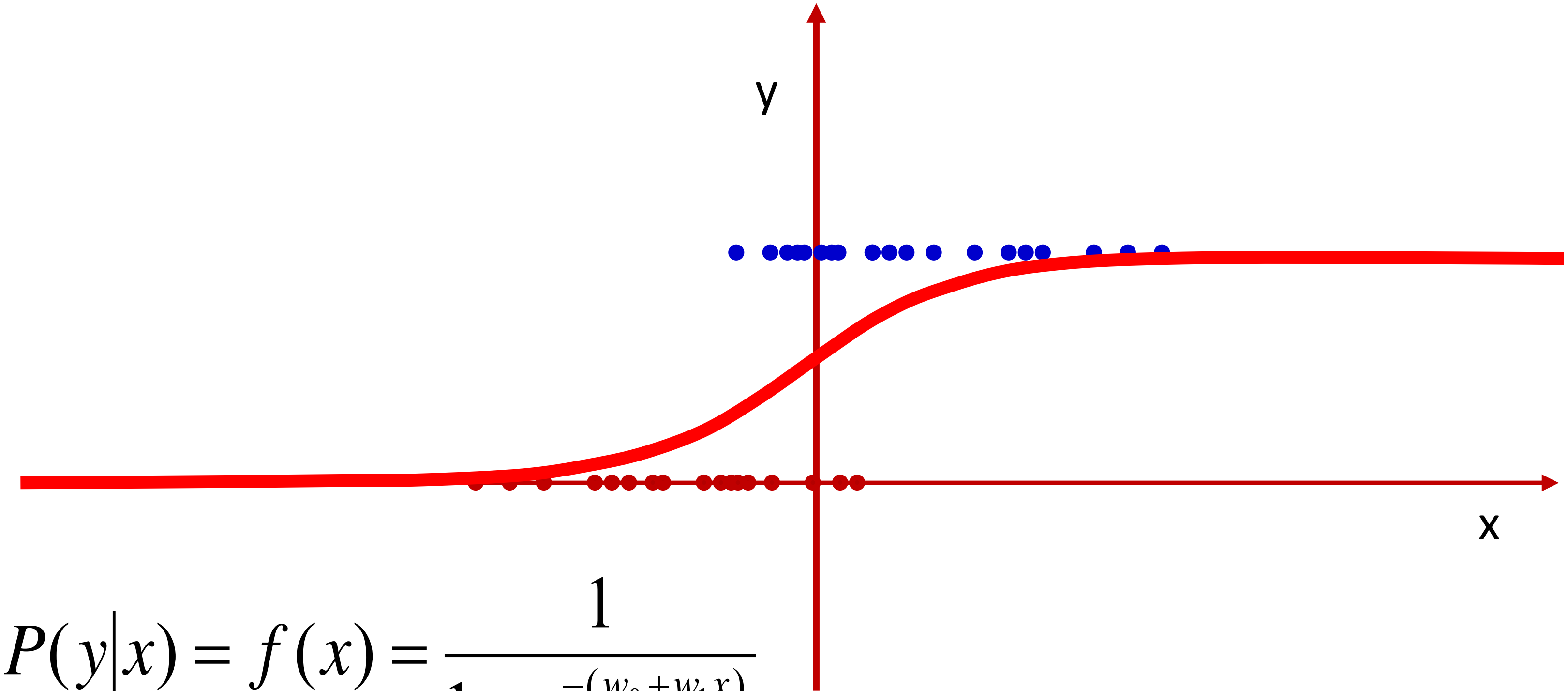
Decision: y > 0.5?



When X is a 2-D variable

- This the perceptron with a sigmoid activation
  - It actually computes the *probability* that the input belongs to class 1
  - Decision boundaries may be obtained by comparing the probability to a threshold
    - These boundaries will be lines (hyperplanes in higher dimensions)
    - The sigmoid perceptron is a *linear classifier*

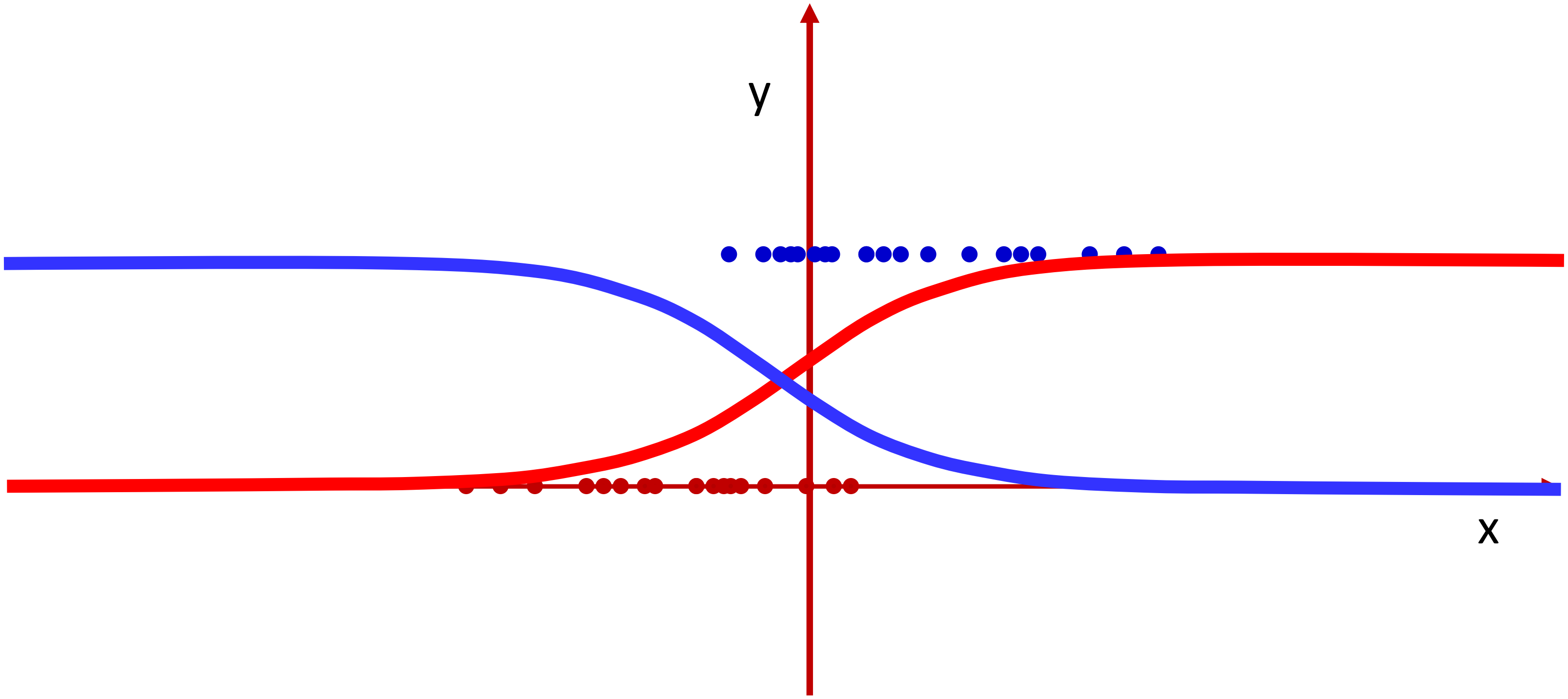
Estimating the model



$$P(y|x) = f(x) = \frac{1}{1 + e^{-(w_0 + w_1 x)}}$$

- Given the training data (many (x, y) pairs represented by the dots), estimate w_0 and w_1 for the curve

Estimating the model



- Easier to represent using a y = +1/-1 notation

$$P(y=1|x)=\frac{1}{1+e^{-(w_0+w_1x)}}$$

$$P(y=-1|x)=\frac{1}{1+e^{(w_0+w_1x)}}$$

$$\boxed{P(y|x)=\frac{1}{1+e^{-y(w_0+w_1x)}}}$$

Estimating the model

- Given: Training data

$$(X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)$$

- Xs are vectors, ys are binary (0/1) class values
- Total probability of data

$$P\big((X_1, y_1), (X_2, y_2), \ldots, (X_N, y_N)\big) = \prod_i P(X_i, y_i)$$

$$= \prod_i P(y_i | X_i\,) P(X_i) = \prod_i \frac{1}{1 + e^{-y_i(w_0 + w^T X_i)}} P(X_i)$$

Estimating the model

- Likelihood

$$P(Training\ data) = \prod_i \frac{1}{1 + e^{-y_i(w_0 + w^T X_i)}} P(X_i)$$

- Log likelihood

$$\log P(Training\ data) =$$

$$\sum_i \log P(X_i) - \sum_i \log\left(1 + e^{-y_i(w_0 + w^T X_i)}\right)$$

Maximum Likelihood Estimate

$$\hat{w}_0, \hat{w}_1 = \underset{w_0, w_1}{\operatorname{argmax}} \log P(Training\ data)$$

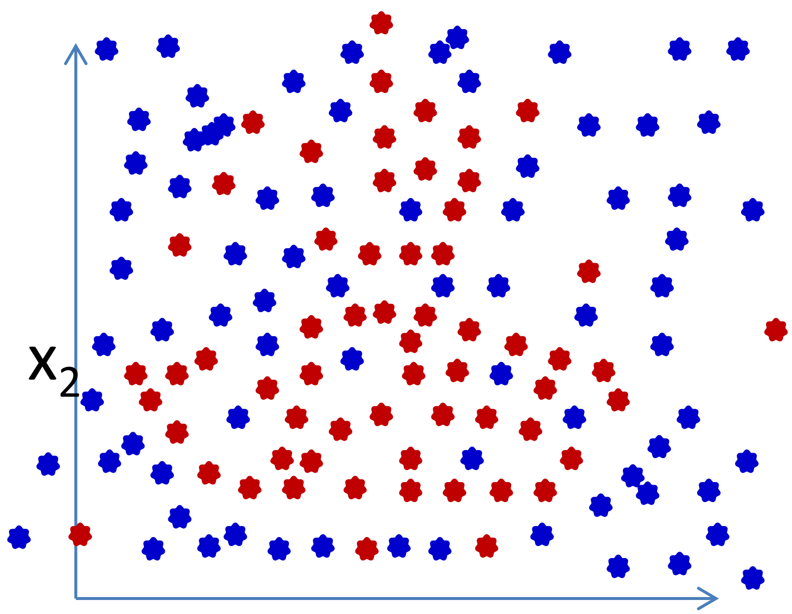
- Equals (note argmin rather than argmax)

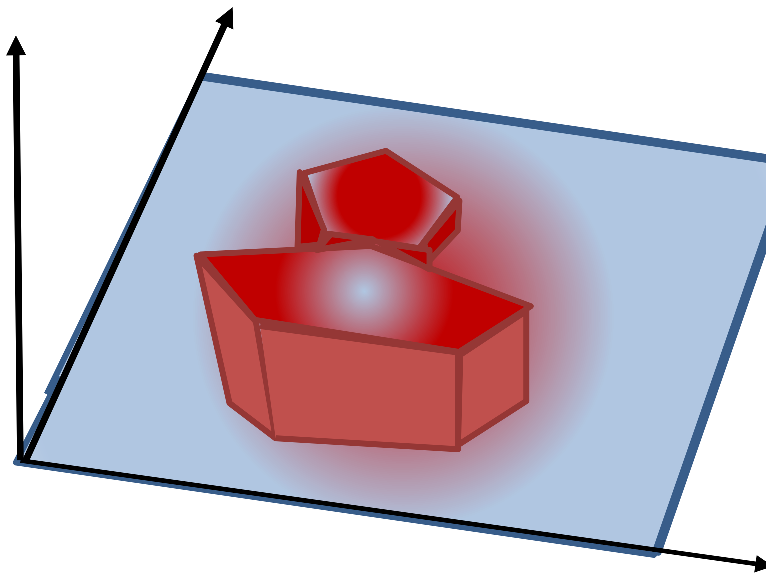
$$\hat{w}_0, \hat{w}_1 = \underset{w_0, w}{\operatorname{argmin}} \sum_i \log\left(1 + e^{-y_i(w_0 + w^T X_i)}\right)$$

- Identical to minimizing the KL divergence between the desired output y and actual output $\frac{1}{1+e^{-(w_0+w^T X_i)}}$

- Cannot be solved directly, needs gradient descent

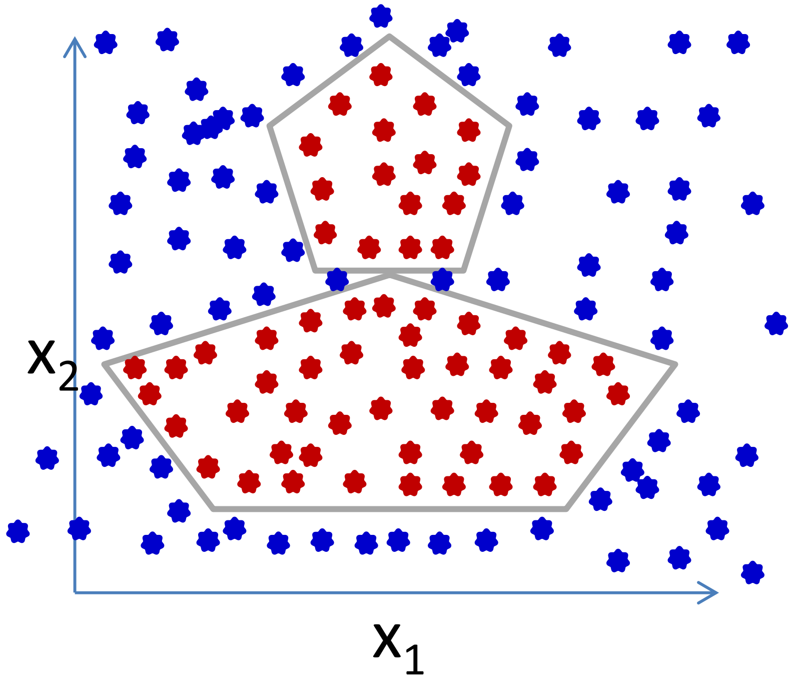
So what about this one?

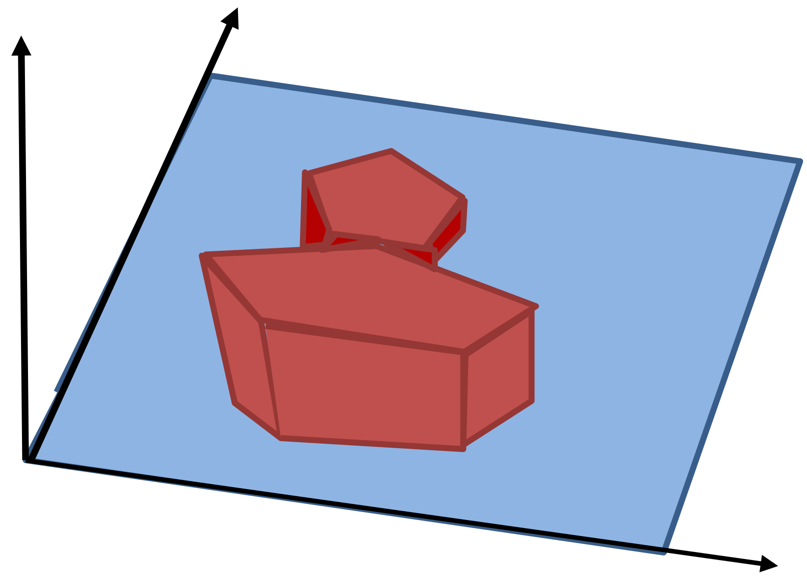




- Non-linear classifiers..

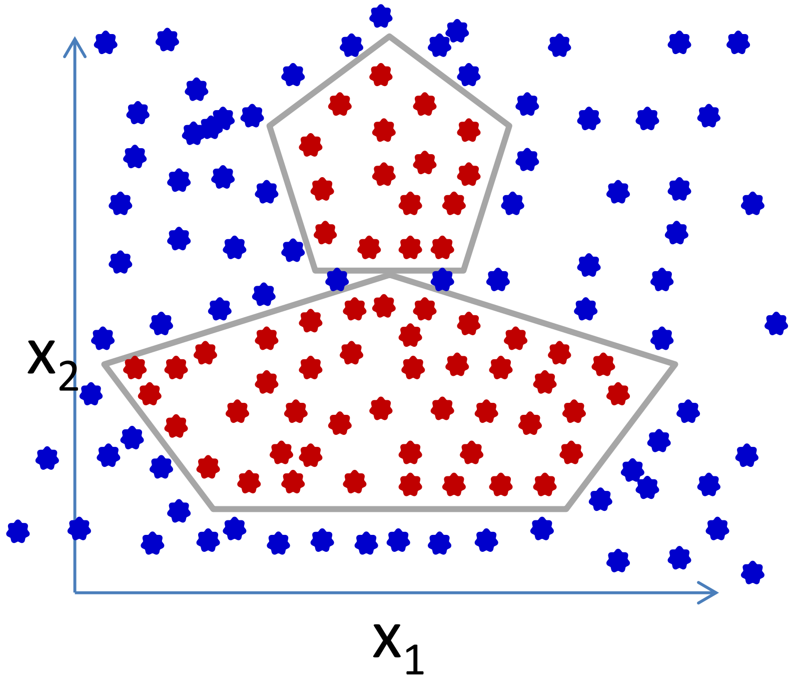
First consider the separable case..

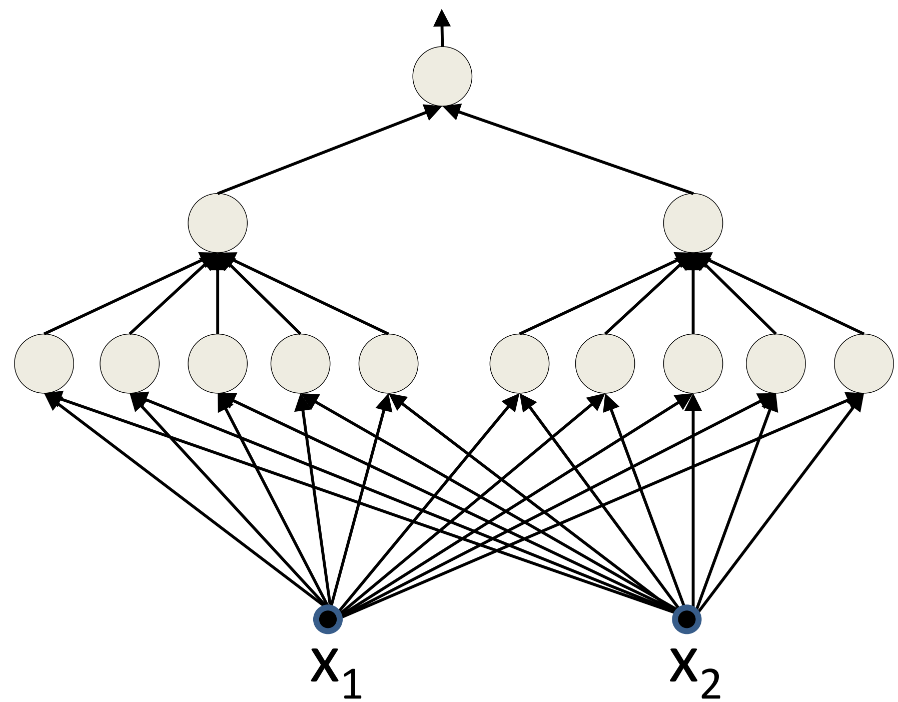




- When the net must learn to classify..

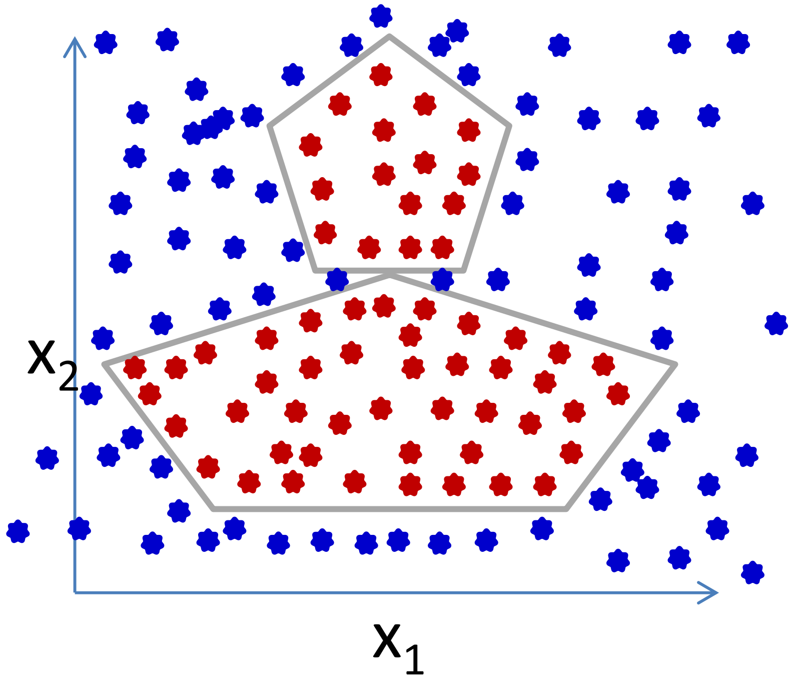
First consider the separable case..

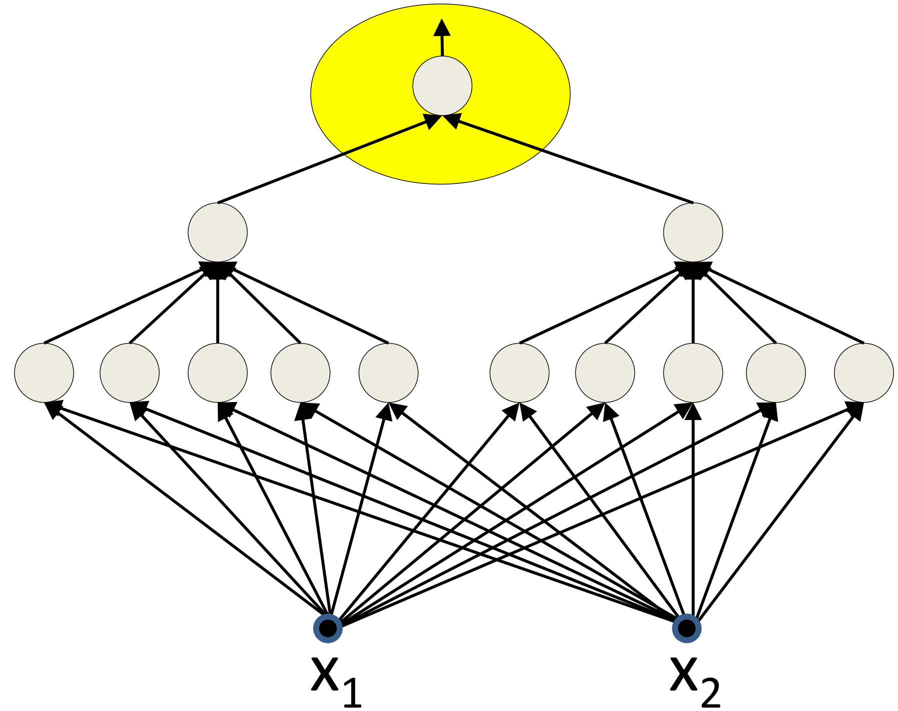




- For a “sufficient” net

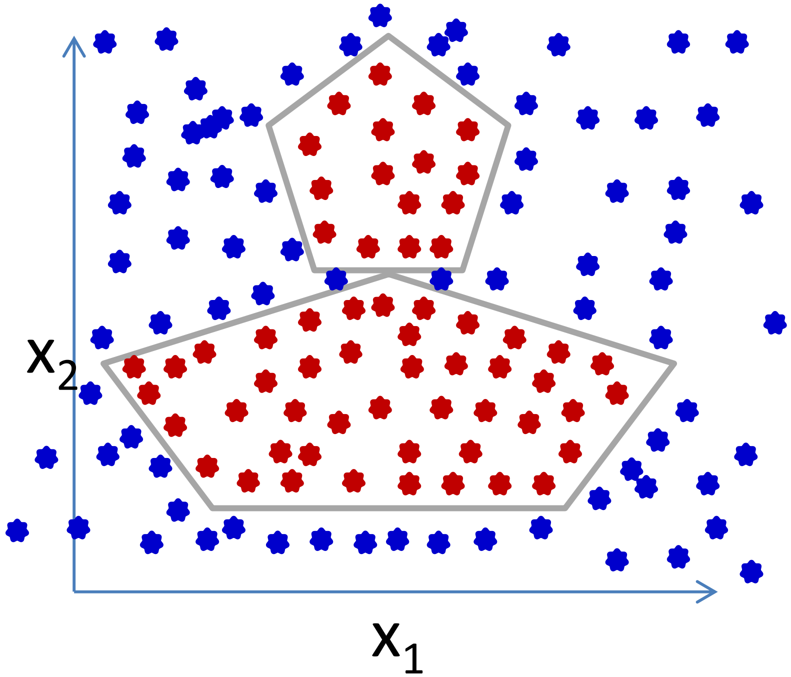
First consider the separable case..

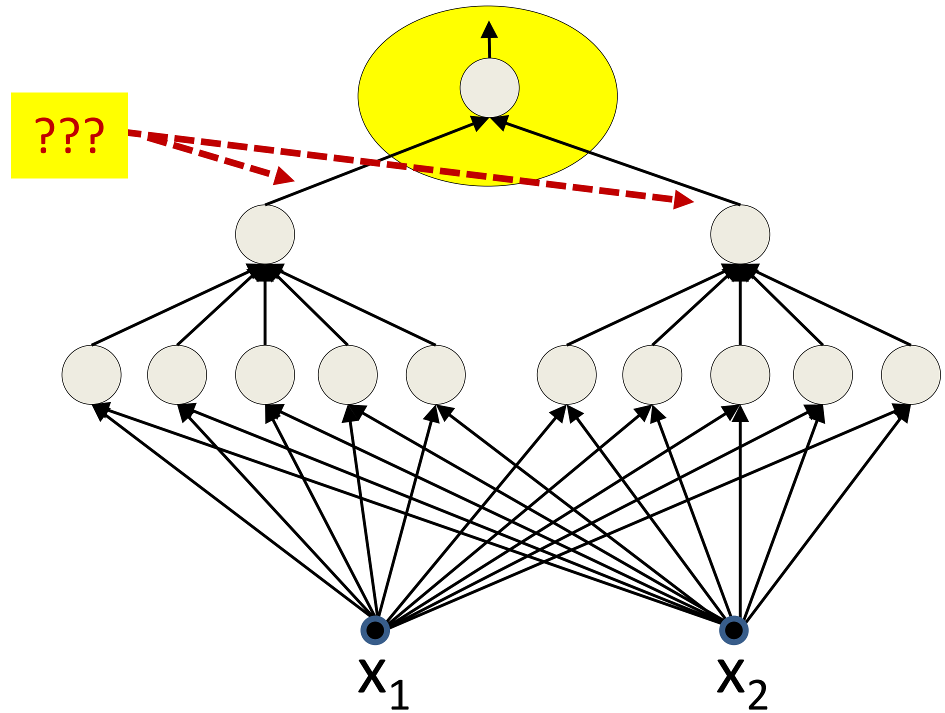




- For a "sufficient" net
- This final perceptron is a linear classifier

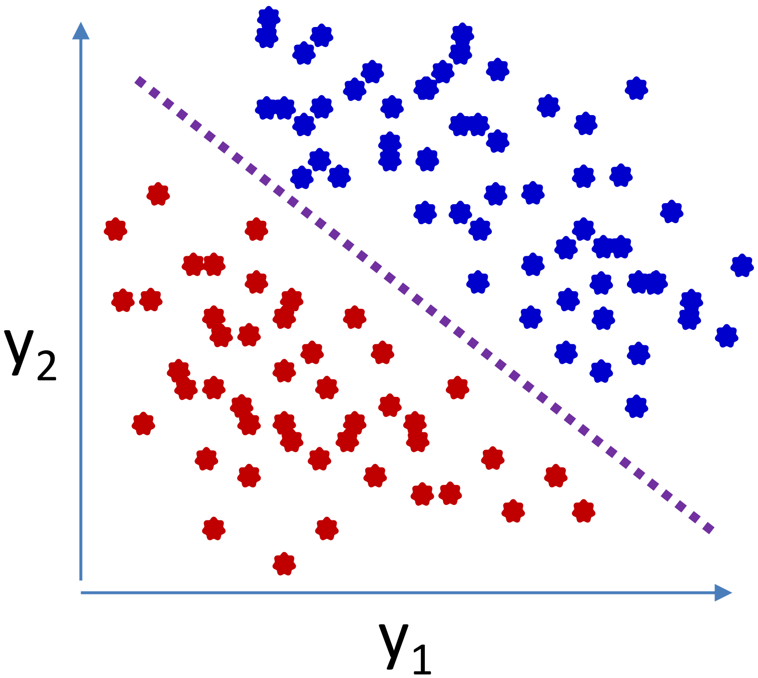
First consider the separable case..

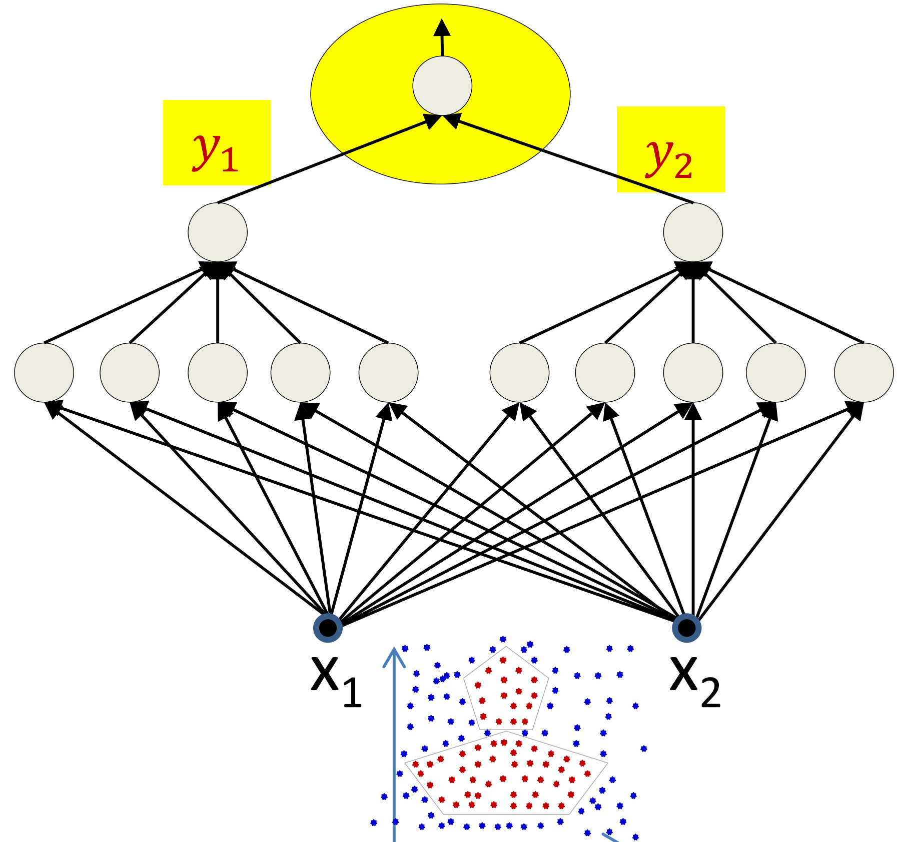




- For a "sufficient" net
- This final perceptron is a linear classifier over the output of the penultimate layer

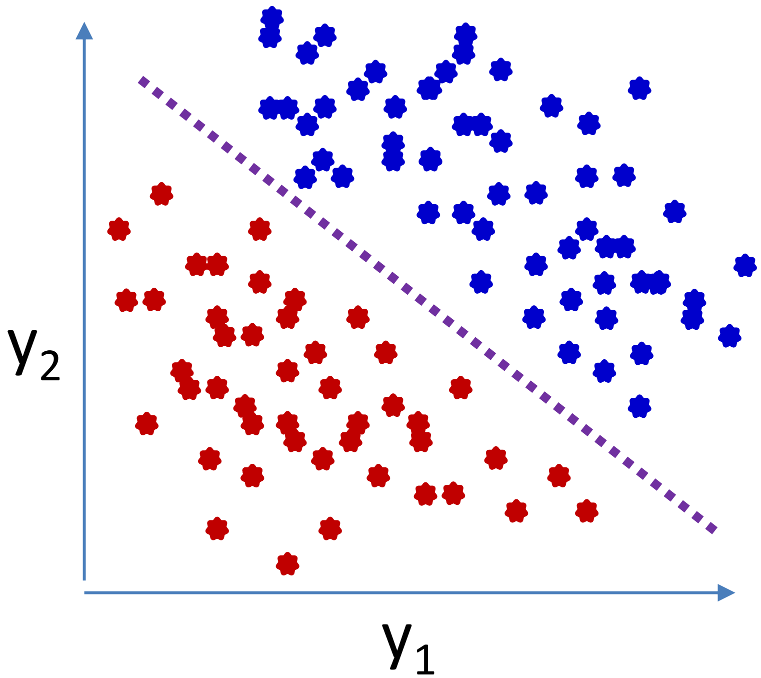
First consider the separable case..

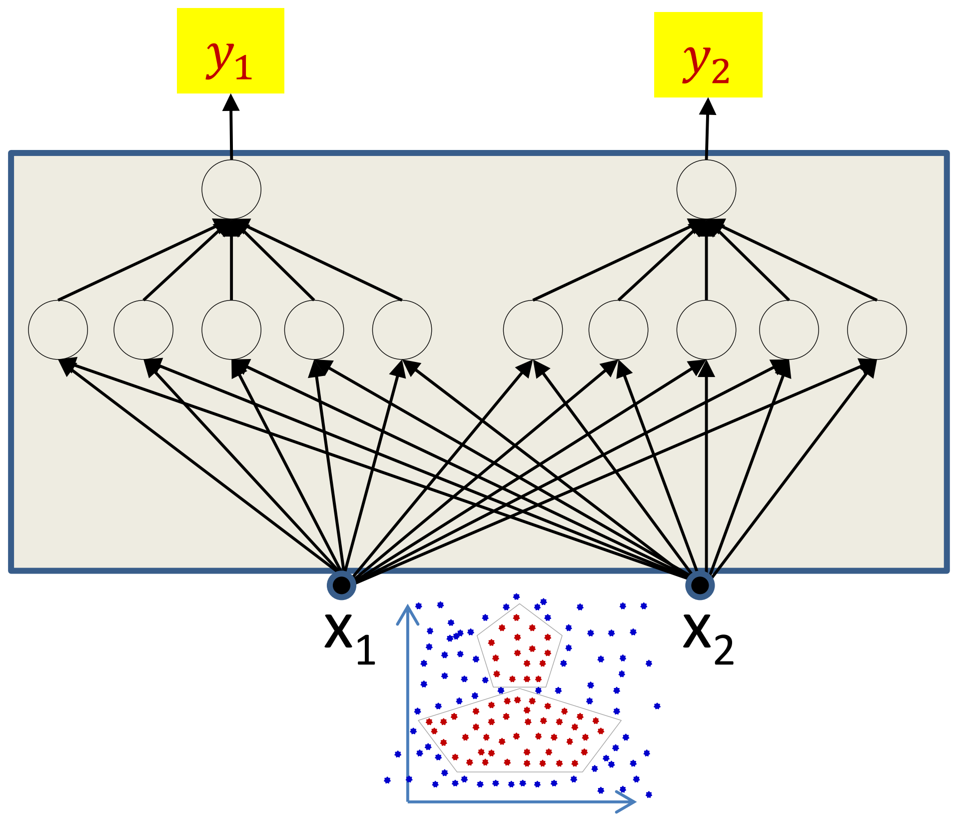




- For perfect classification the output of the penultimate layer must be linearly separable

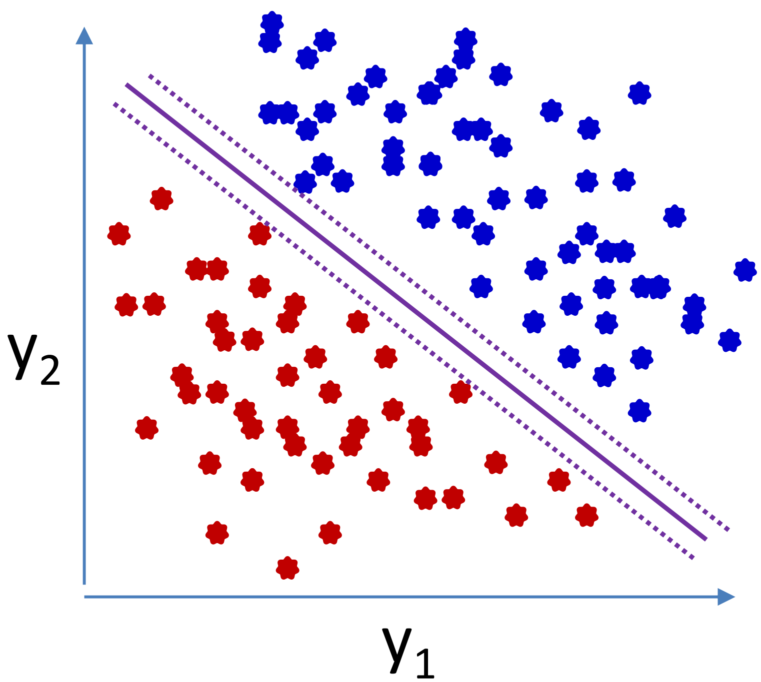
First consider the separable case..

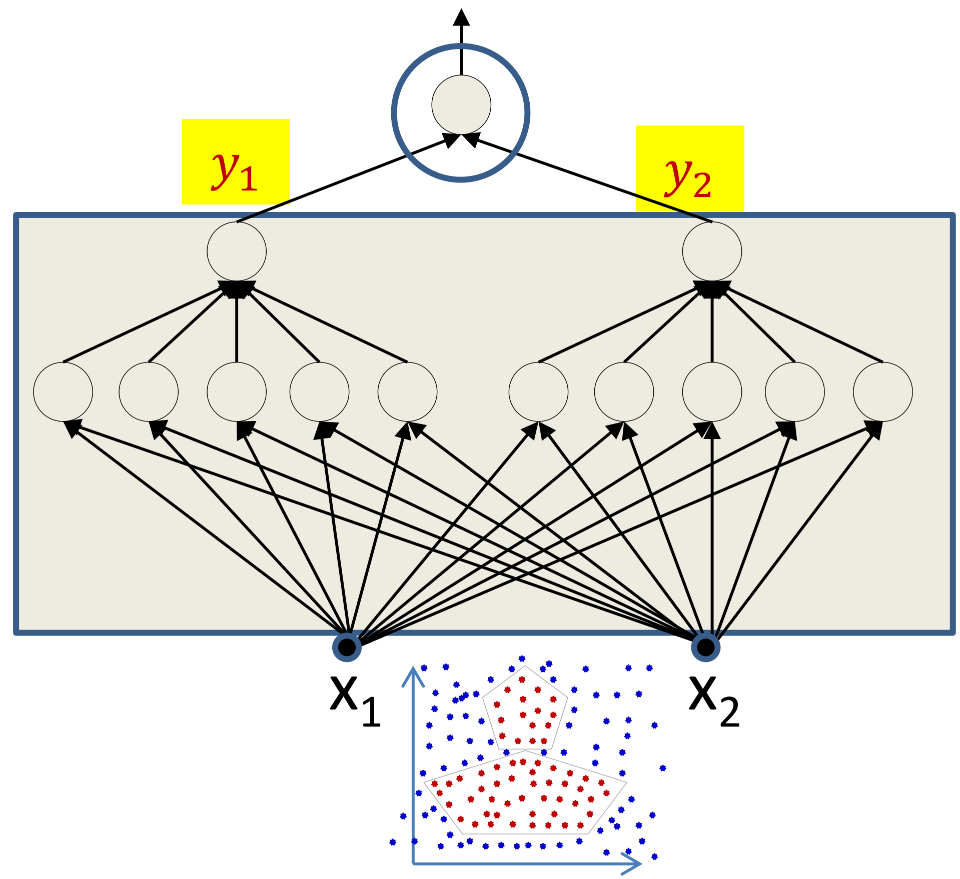




- The rest of the network may be viewed as a transformation that transforms data from non-linear classes to linearly separable features

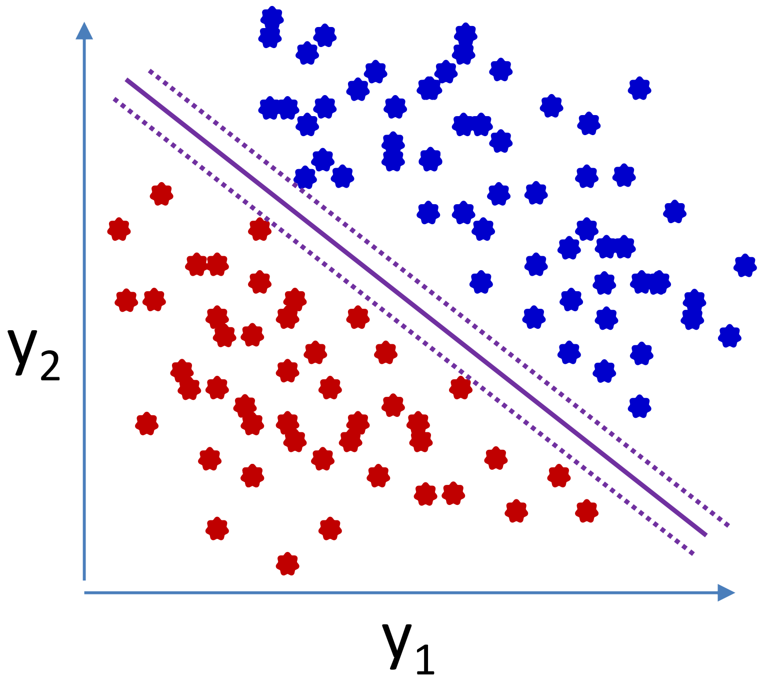
First consider the separable case..

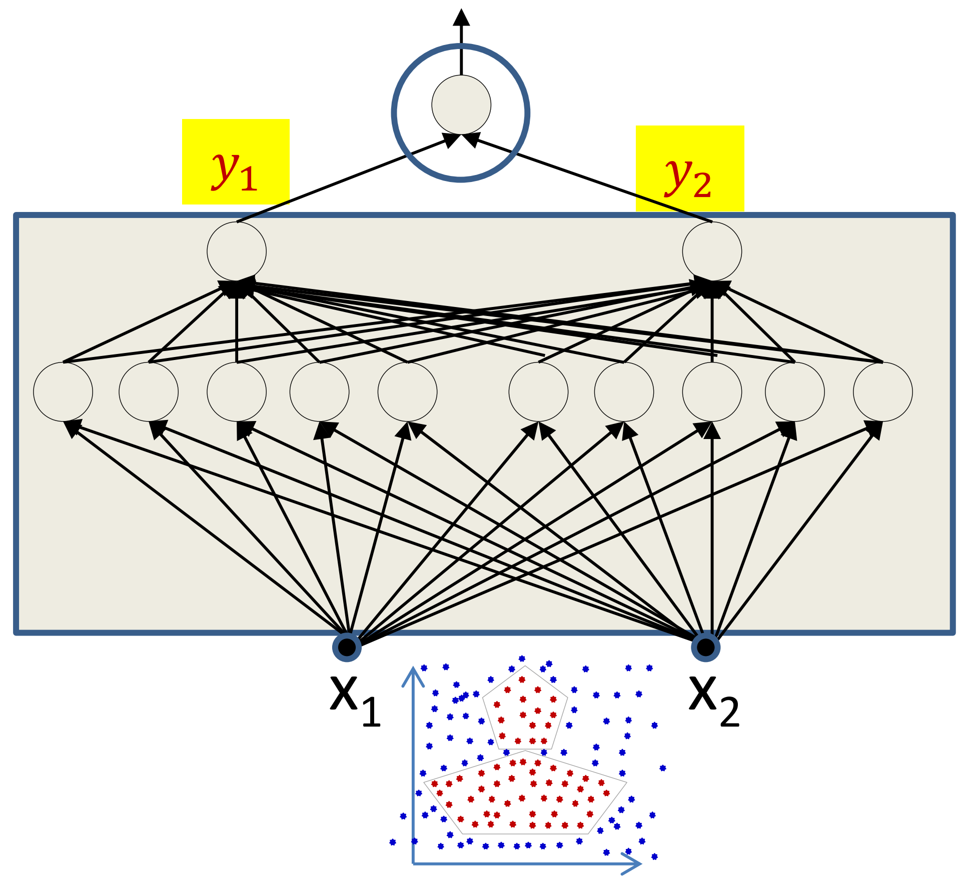




- The rest of the network may be viewed as a transformation that transforms data from non-linear classes to linearly separable features
  - We can now attach *any* linear classifier above it for perfect classification
  - Need not be a perceptron
  - In fact, for ***binary*** classifiers an SVM on top of the features may be more generalizable!

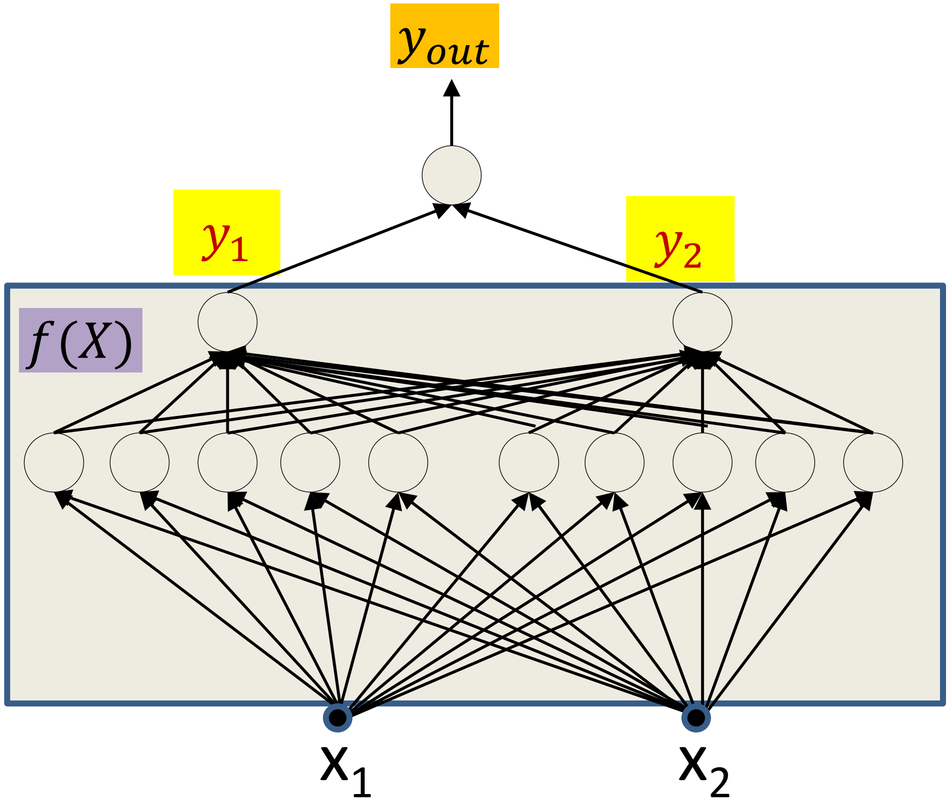
First consider the separable case..

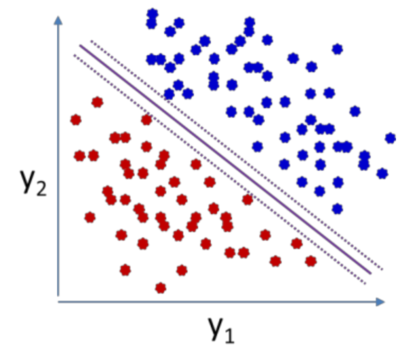




- This is true of *any* sufficient structure
  - Not just the optimal one
- For *insufficient* structures, the network may *attempt* to transform the inputs to linearly separable features
  - Will fail to separate
  - Still, for binary problems, using an SVM with slack may be more effective than a final perceptron!

Mathematically..



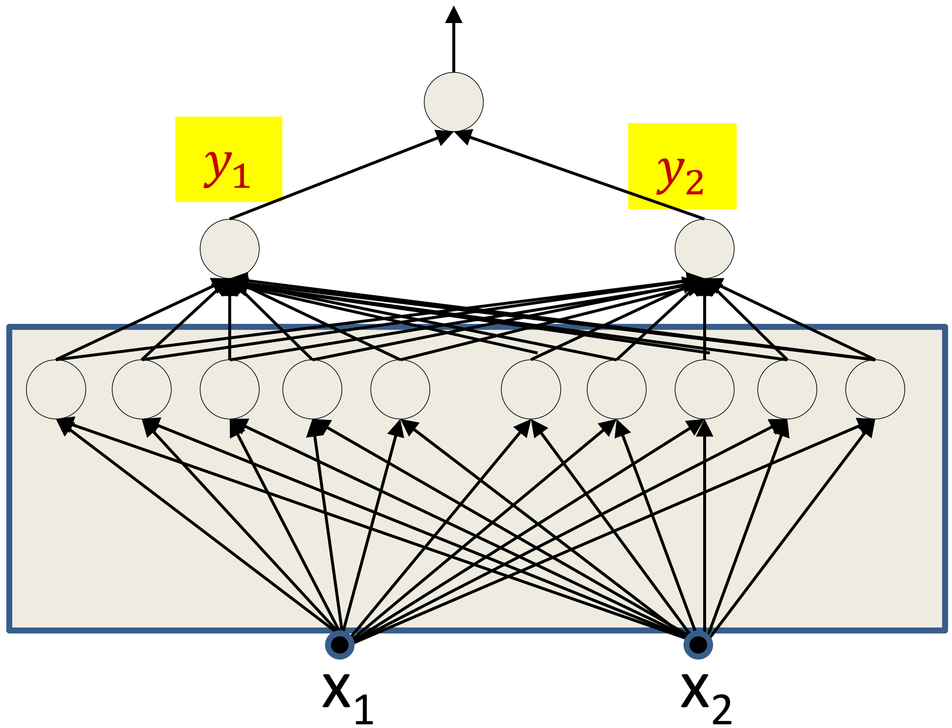


- $y_{out} = \frac{1}{1+\exp(b+W^T Y)} = \frac{1}{1+\exp(b+W^T f(X))}$
- The data are (almost) linearly separable in the space of Y
- The network until the second-to-last layer is a non-linear function $f(X)$ that converts the input space of X into the feature space Y where the classes are maximally linearly separable

Story so far

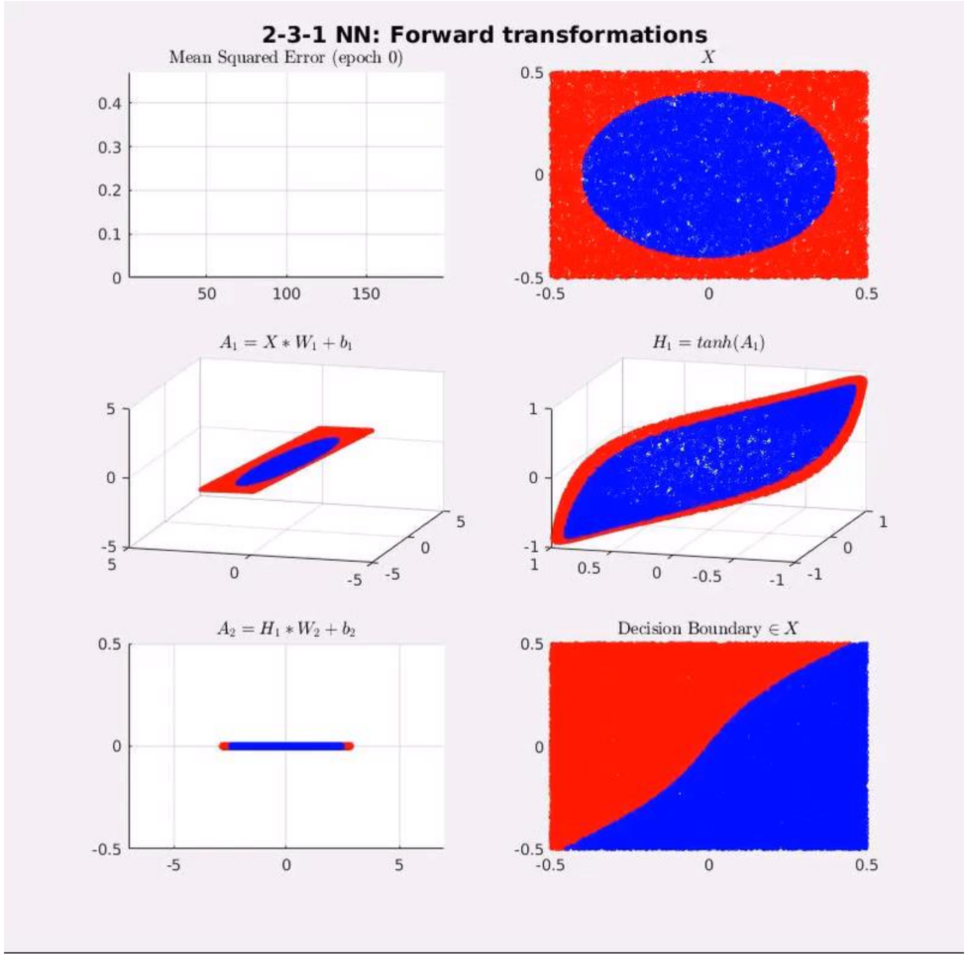
- A classification MLP actually comprises two components
  - A "feature extraction network" that converts the inputs into linearly separable features
    - Or *nearly* linearly separable features
  - A final linear classifier that operates on the linearly separable features

How about the lower layers?



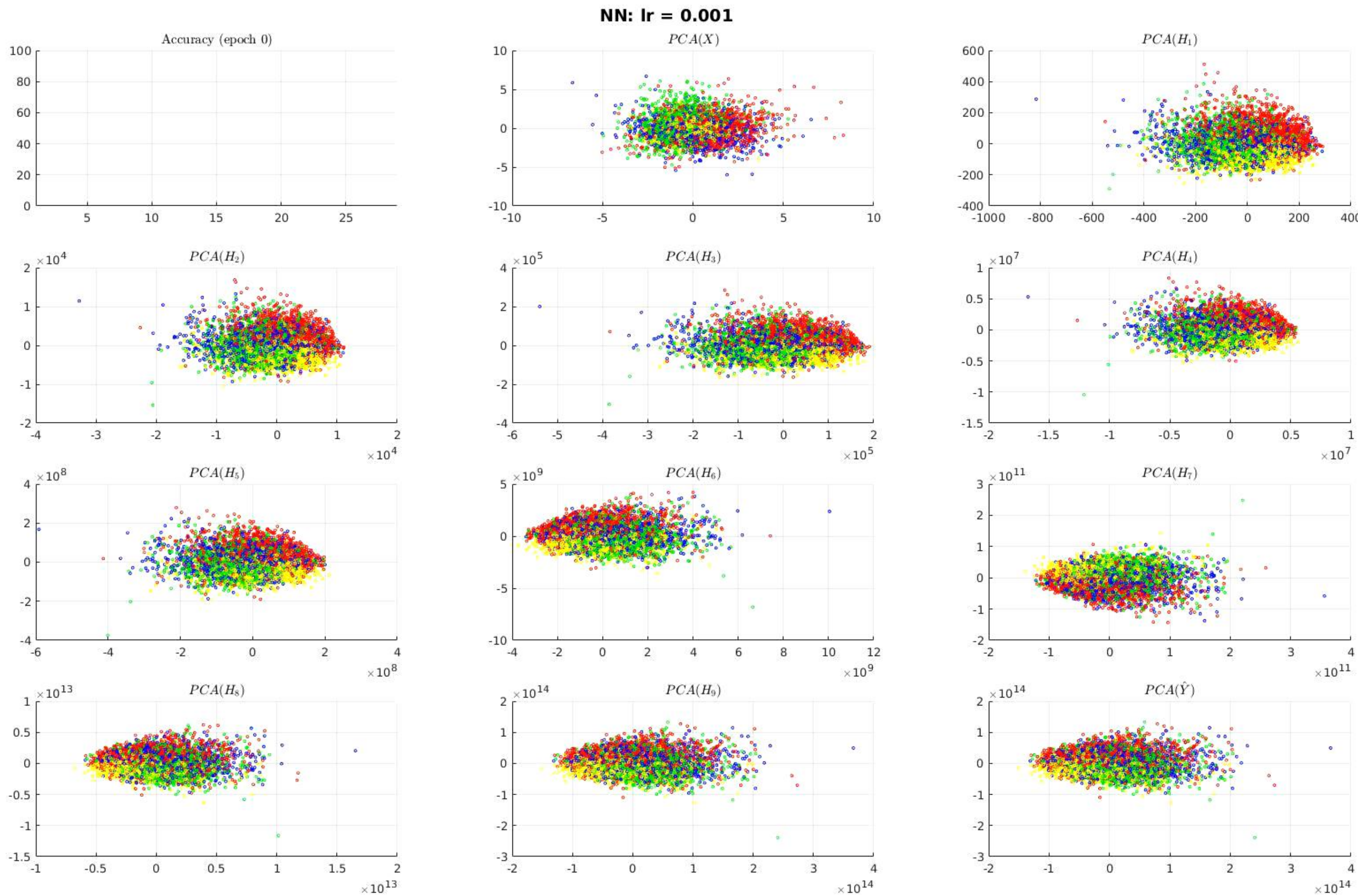
- How do the lower layers respond?
  - They too compute features
  - But how do they look
- Manifold hypothesis: For separable classes, the classes are linearly separable on a non-linear manifold
- Layers sequentially "straighten" the data manifold
  - Until the final layer, which fully linearizes it

The behavior of the layers



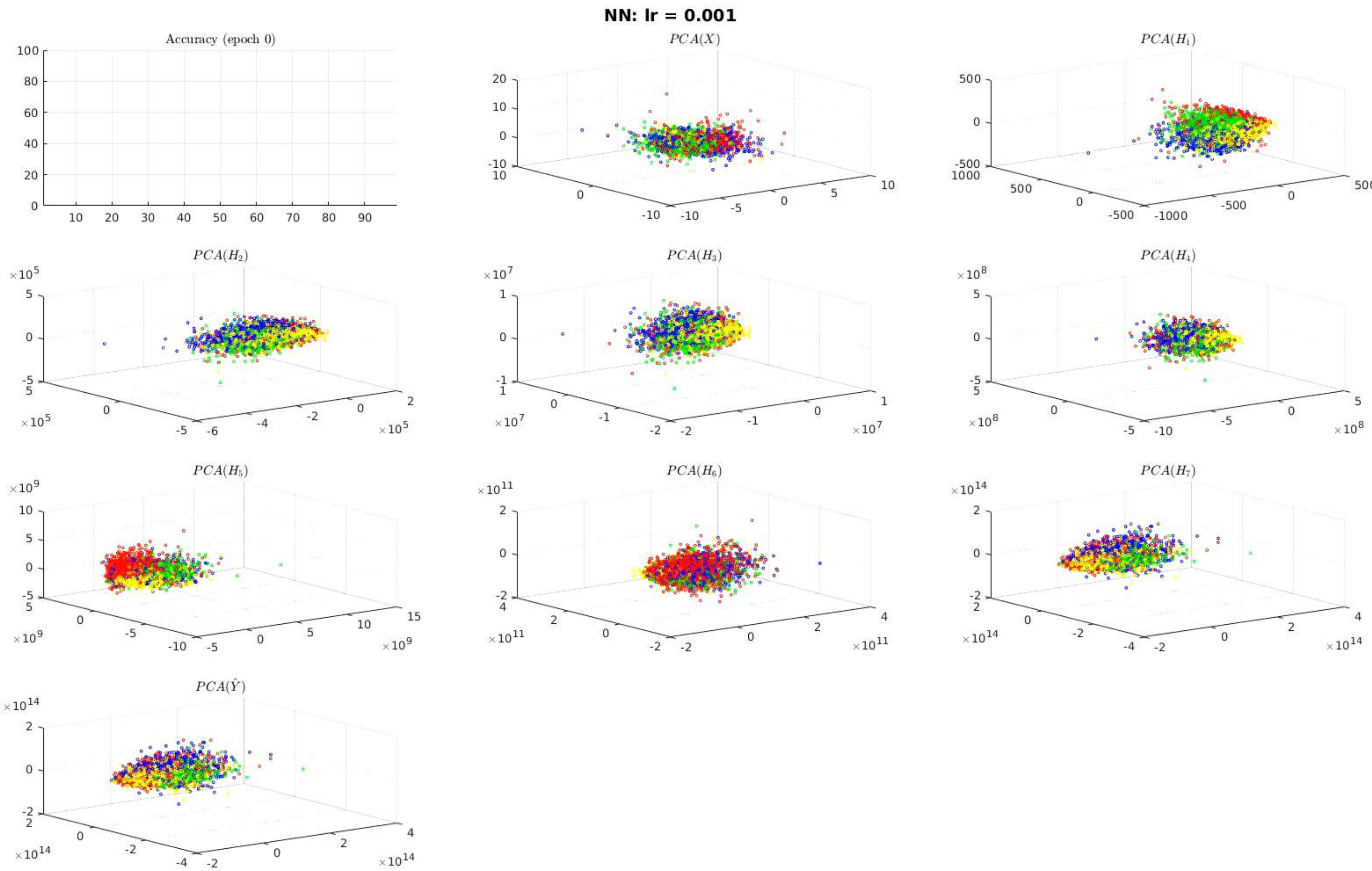
- Synthetic example: Feature space

The behavior of the layers



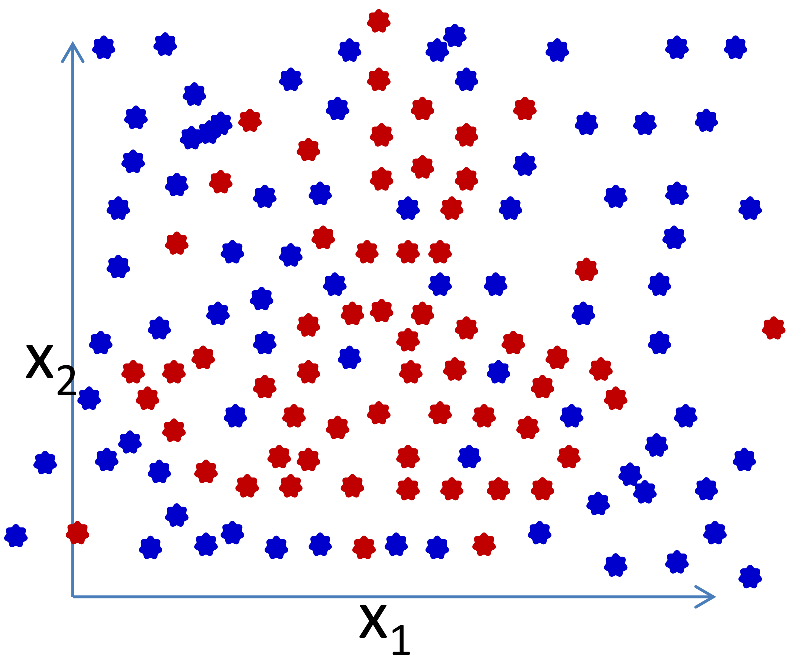
- CIFAR

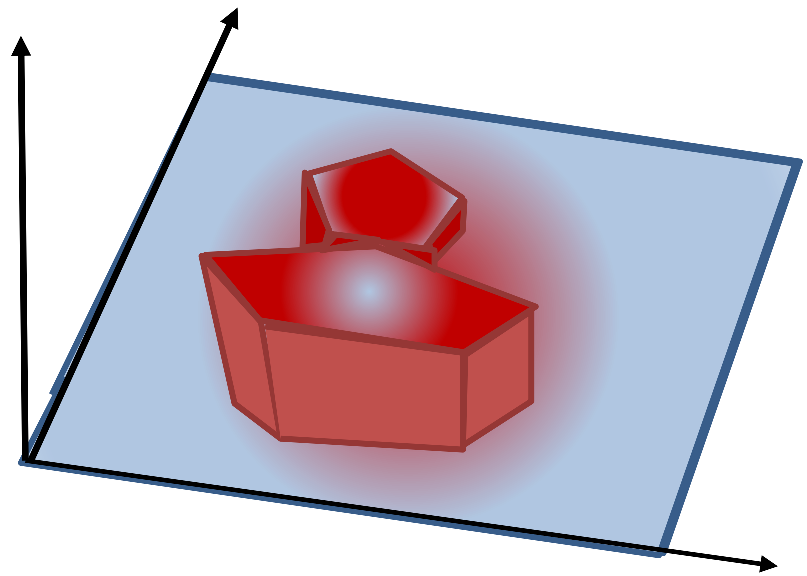
The behavior of the layers



- CIFAR

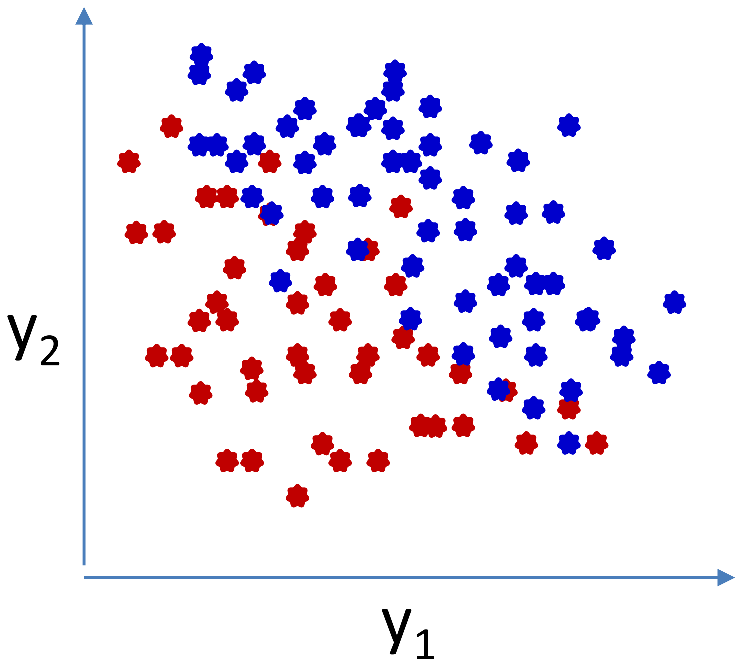
When the data are not separable and boundaries are not linear..

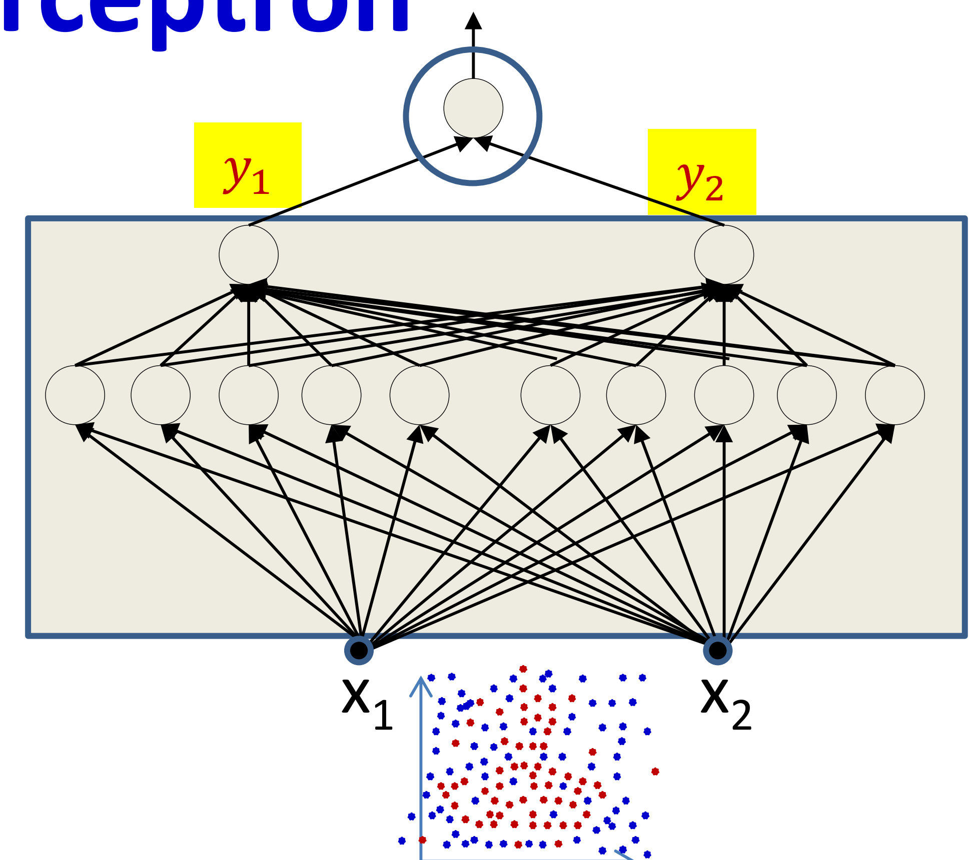




- More typical setting for classification problems

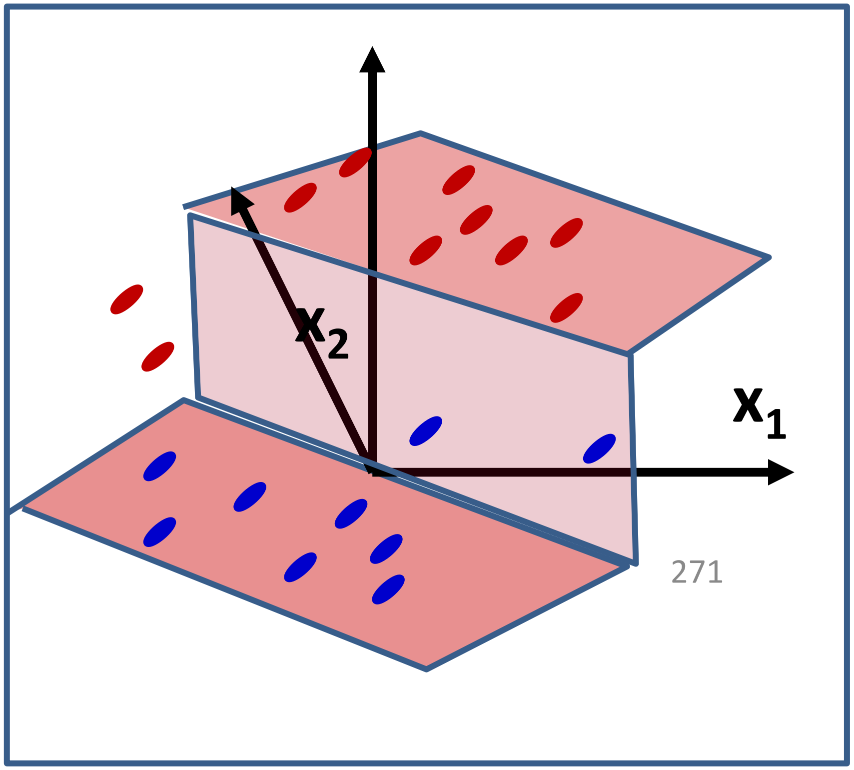
Inseparable classes with an output logistic perceptron

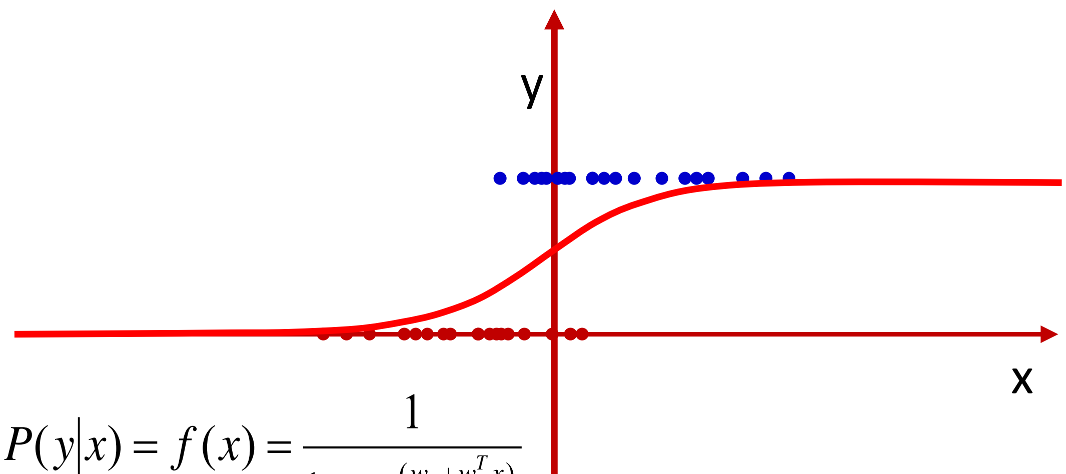




- The "feature extraction" layer transforms the data such that the posterior probability may now be modelled by a logistic

Inseparable classes with an output logistic perceptron

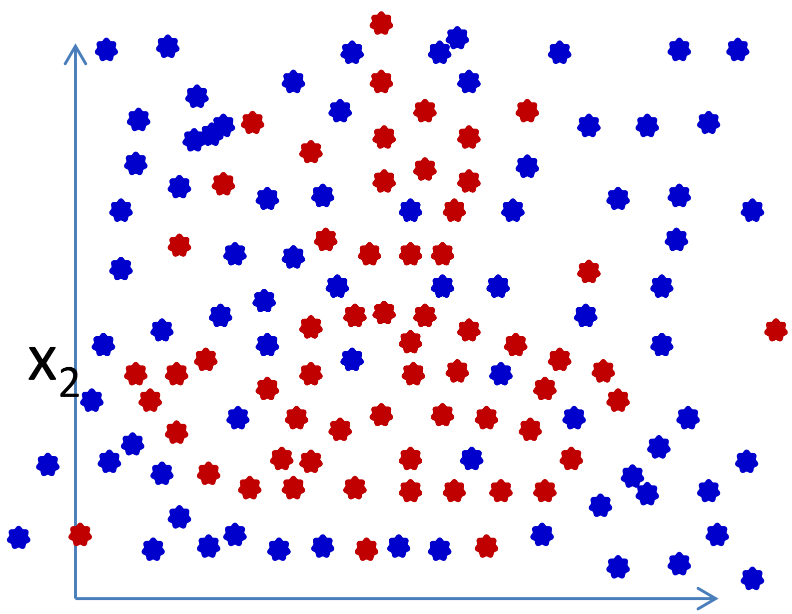


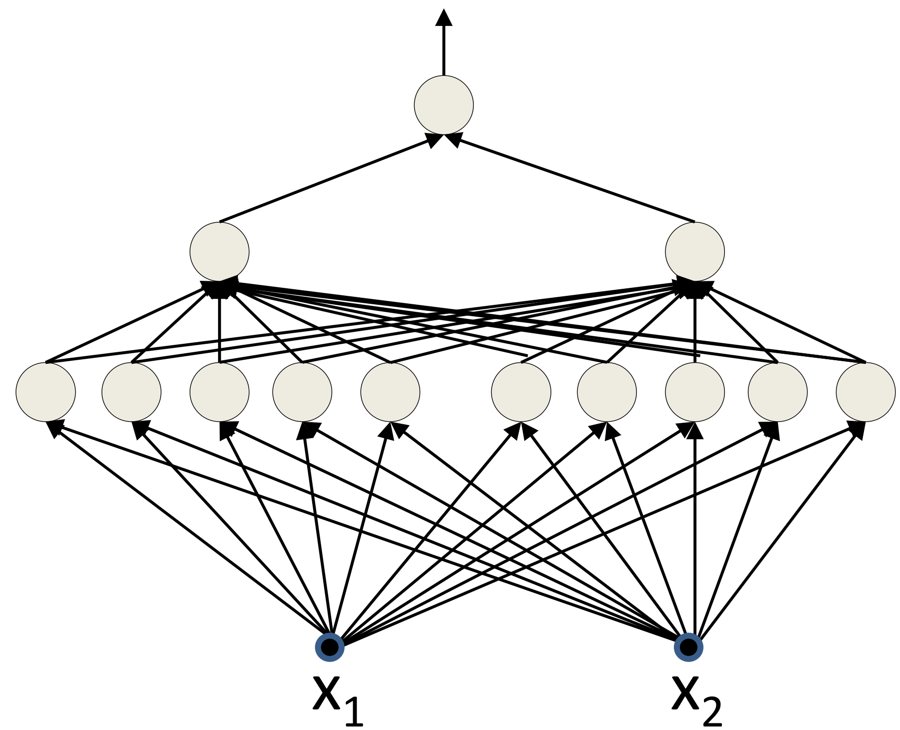


$$P(y|x) = f(x) = \frac{1}{1 + e^{-(w_0 + w^T x)}}$$

- The “feature extraction” layer transforms the data such that the posterior probability may now be modelled by a logistic
  - The output logistic computes the posterior probability of the class given the input

When the data are not separable and boundaries are not linear..

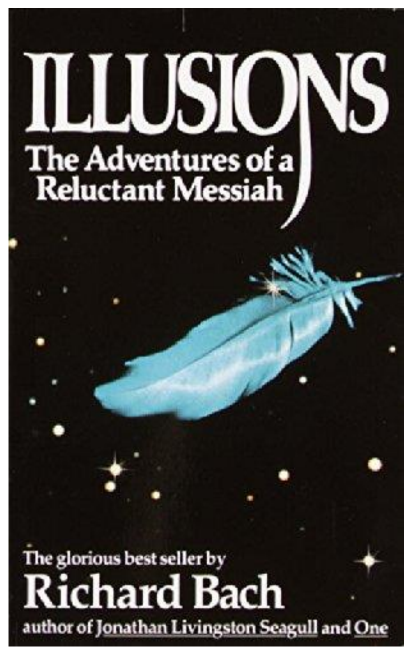




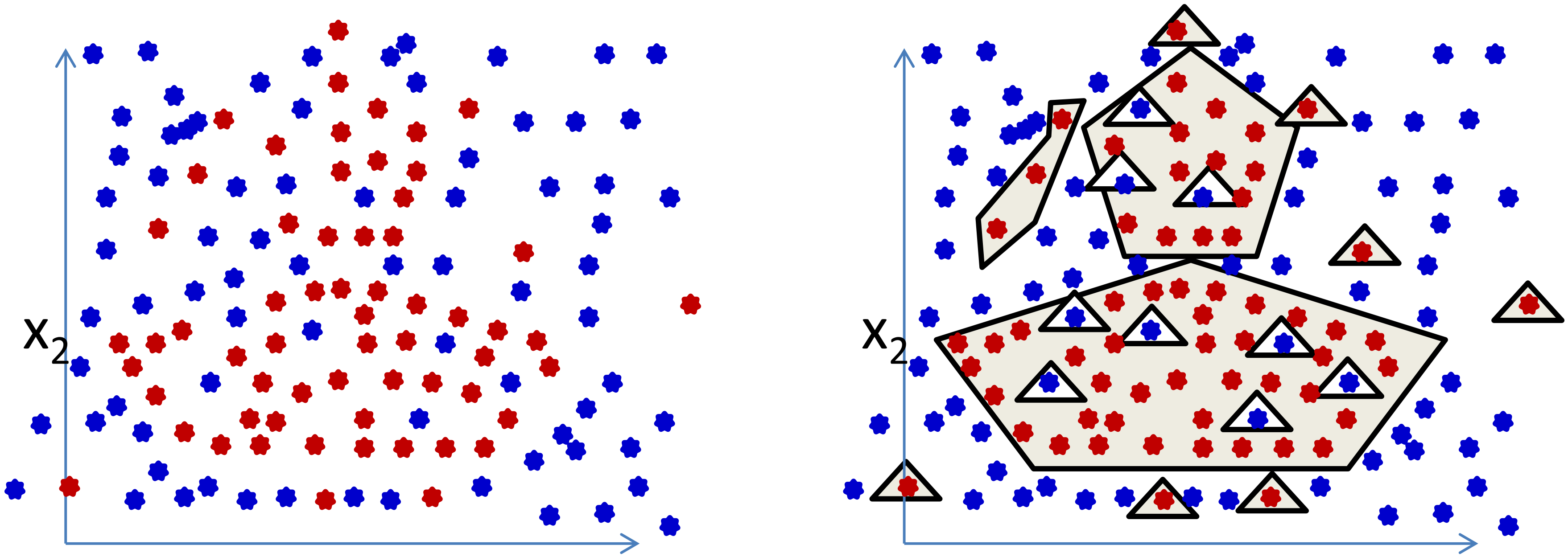
- The output of the network is $P(y|x)$
  - For multi-class networks, it will be the *vector* of a posteriori class probabilities

Everything in this book may be wrong!

- Richard Bach (Illusions)



There's no such thing as inseparable classes

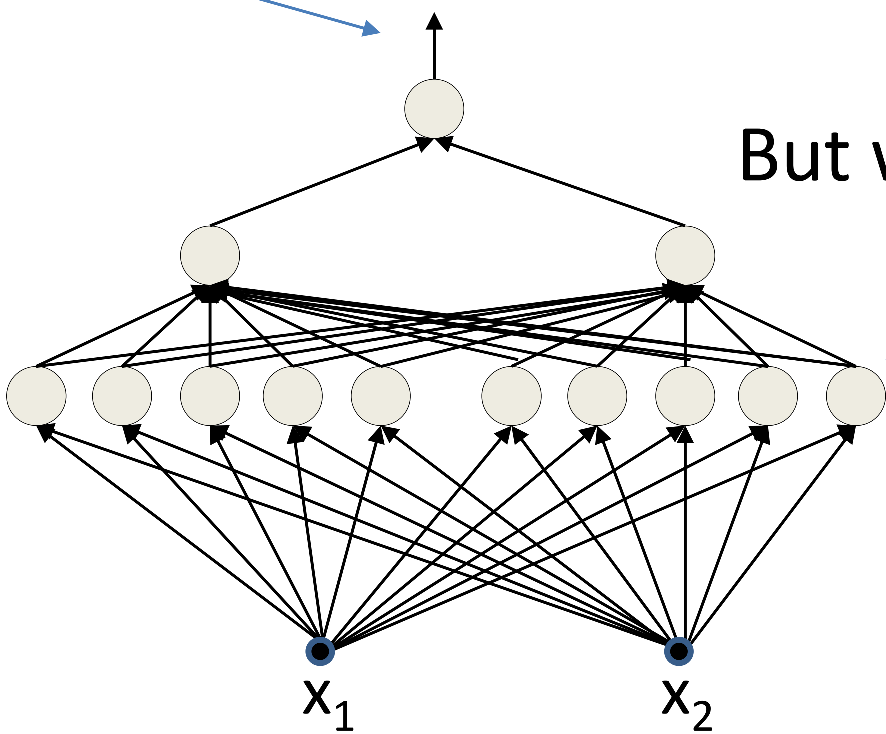


- A sufficiently detailed architecture can separate nearly *any* arrangement of points
  - "Correctness" of the suggested intuitions subject to various parameters, such as regularization, detail of network, training paradigm, convergence etc..

Changing gears..

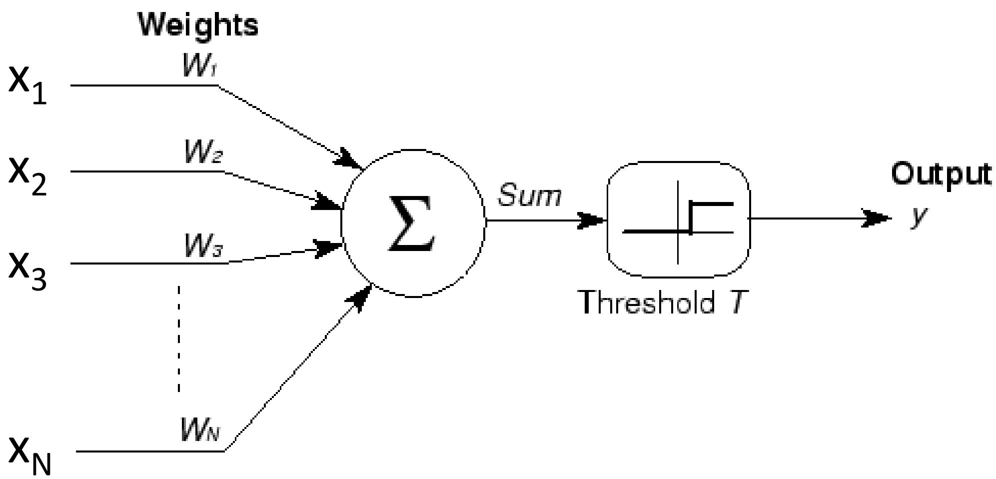
Intermediate layers

We’ve seen what the network learns here



But what about here?

Recall: The basic perceptron

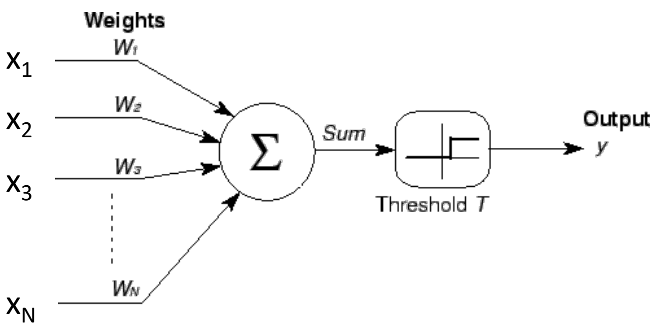


$$y = \begin{cases} 1 \; if \; \sum_i w_i \mathrm{x}_i \geq T \\ 0 \; else \end{cases}$$

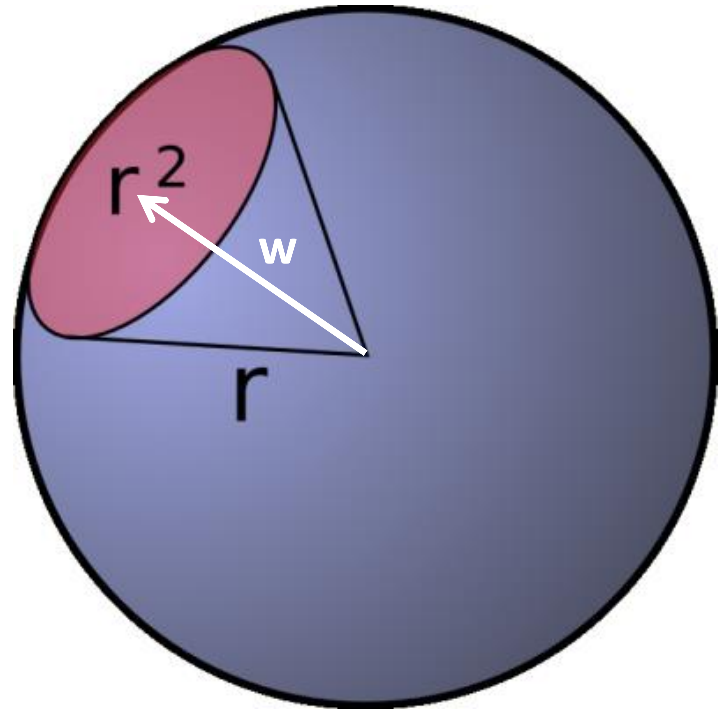
$$y = \begin{cases} 1 \; if \; \mathbf{x}^T \mathbf{w} \geq T \\ 0 \; else \end{cases}$$

- What do the *weights* tell us?
  - The neuron fires if the inner product between the weights and the inputs exceeds a threshold

Recall: The weight as a “template”

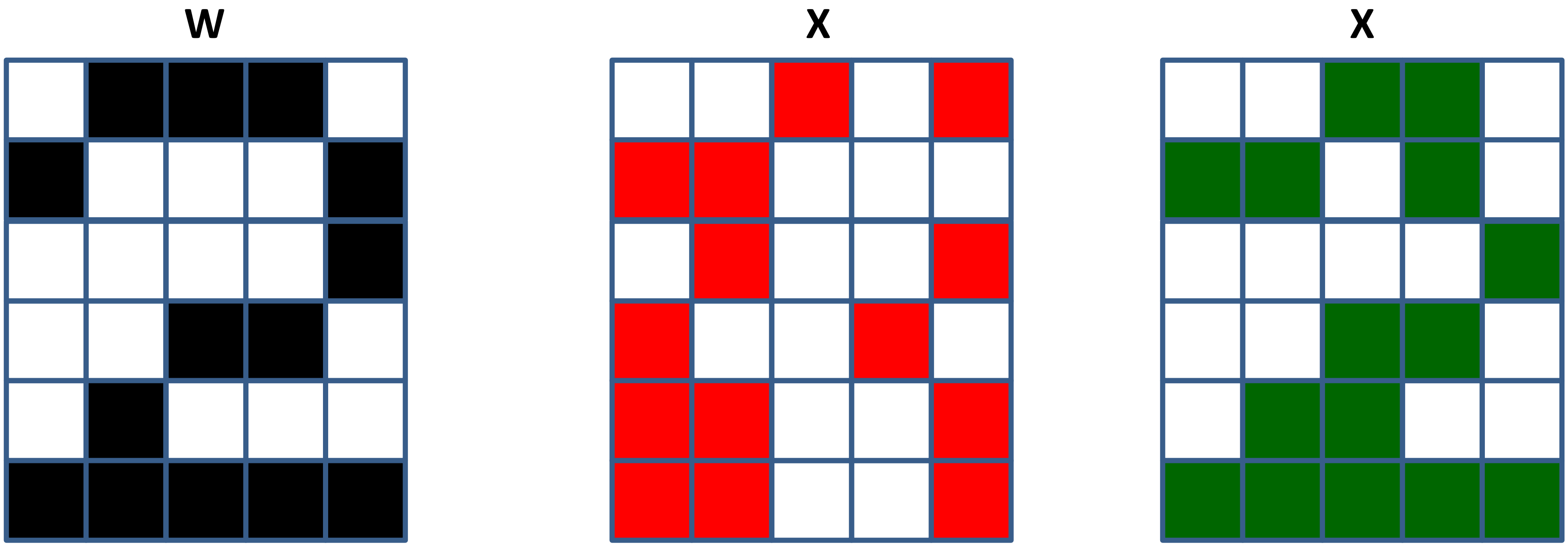


$$X^T W > T$$
$$\cos\theta > \frac{T}{|X|}$$
$$\theta < \cos^{-1}\left(\frac{T}{|X|}\right)$$



- The perceptron fires if the input is within a specified angle of the weight
  - Represents a convex region on the surface of the sphere!
  - The network is a Boolean function over these regions.
    - The overall decision region can be arbitrarily nonconvex
- Neuron fires if the input vector is close enough to the weight vector.
  - If the input pattern matches the weight pattern closely enough

Recall: The weight as a template



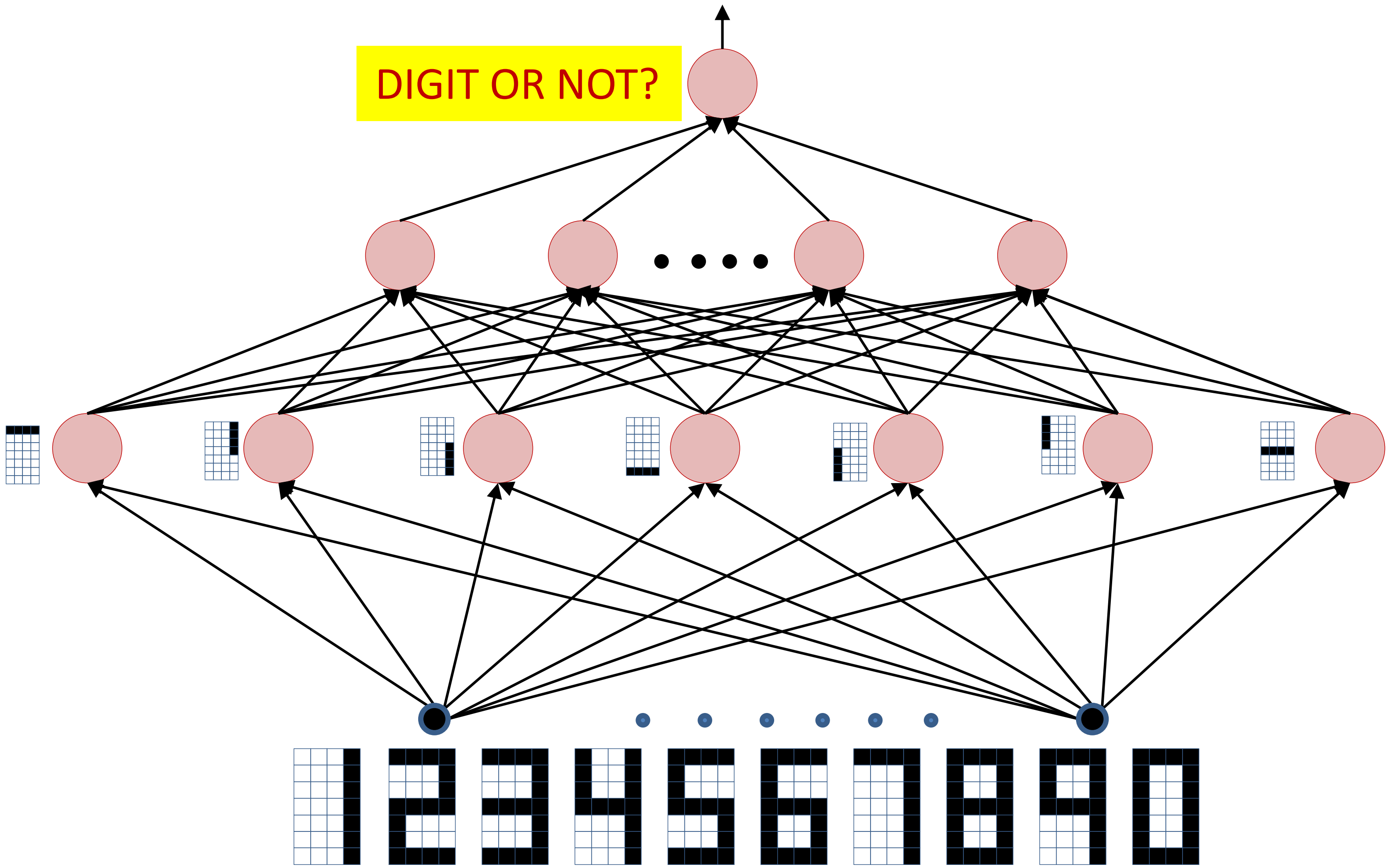
Correlation = 0.57

Correlation = 0.82

$$y = \begin{cases} 1 \ if \ \sum_i w_i \mathrm{x}_i \geq T \\ 0 \ else \end{cases}$$

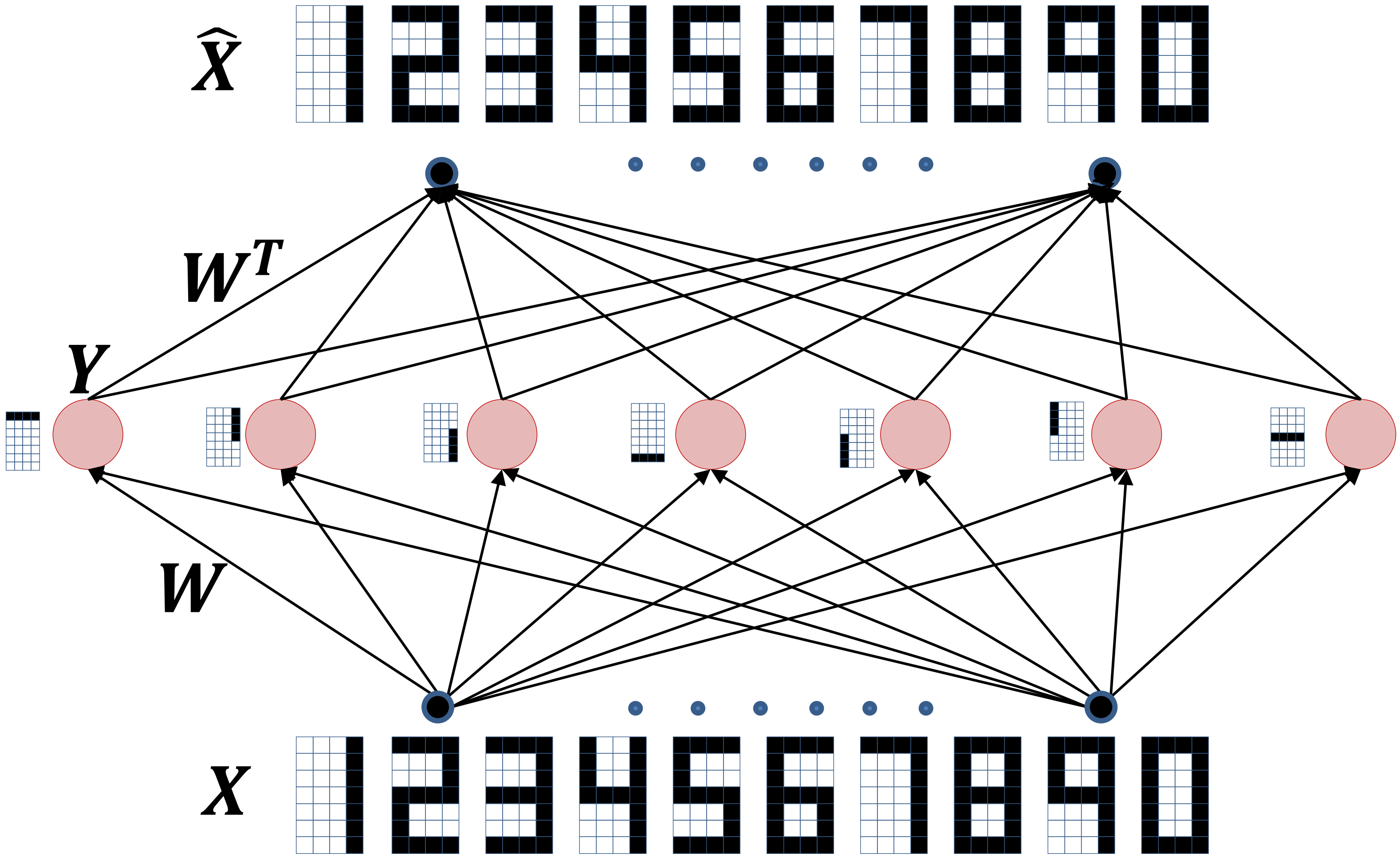
- If the *correlation* between the weight pattern and the inputs exceeds a threshold, fire
- The perceptron is a *correlation filter!*

Recall: MLP features



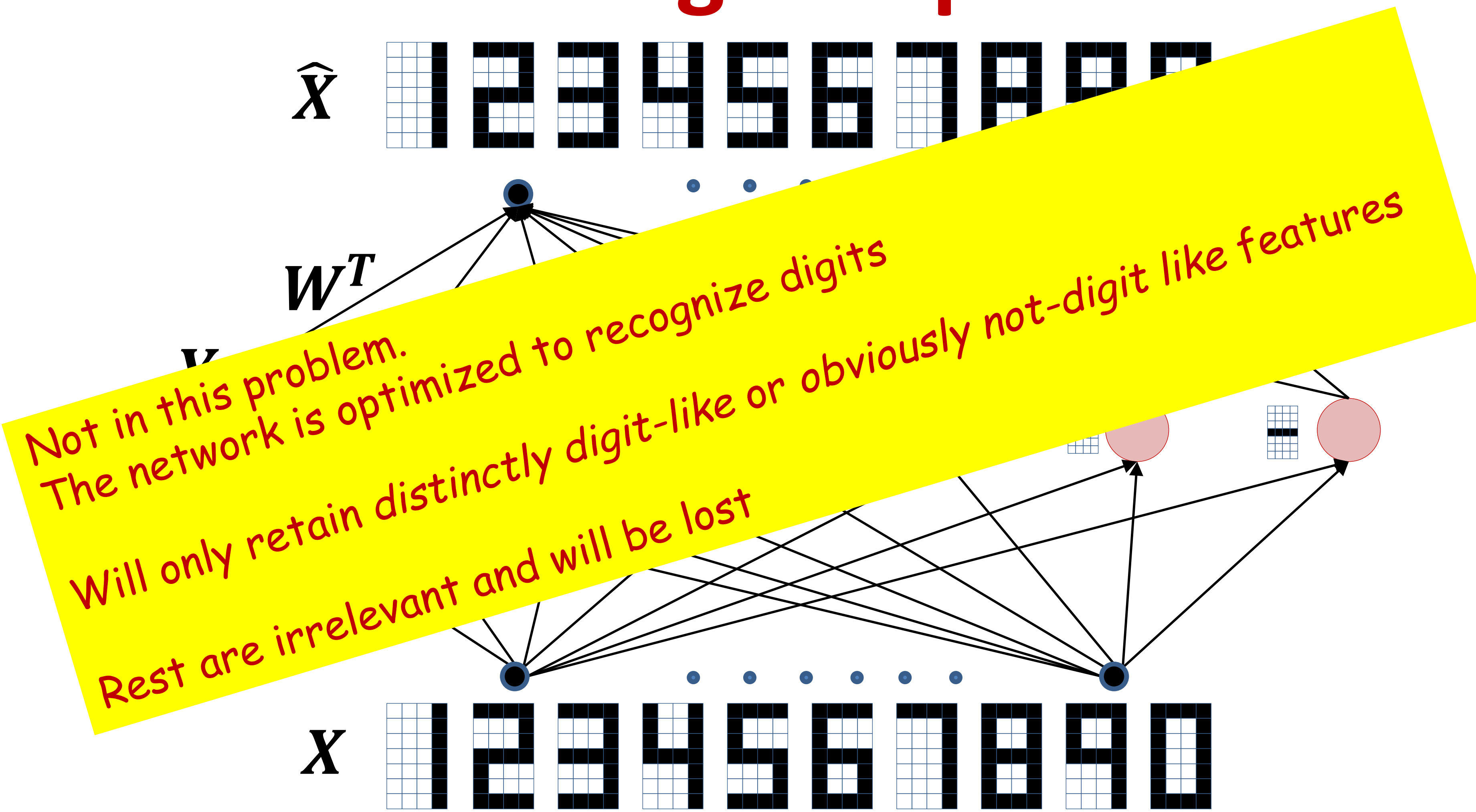
- The lowest layers of a network detect significant features in the signal
- **The signal could be (partially) reconstructed using these features**
  - **Will retain all the significant components of the signal**

Making it explicit



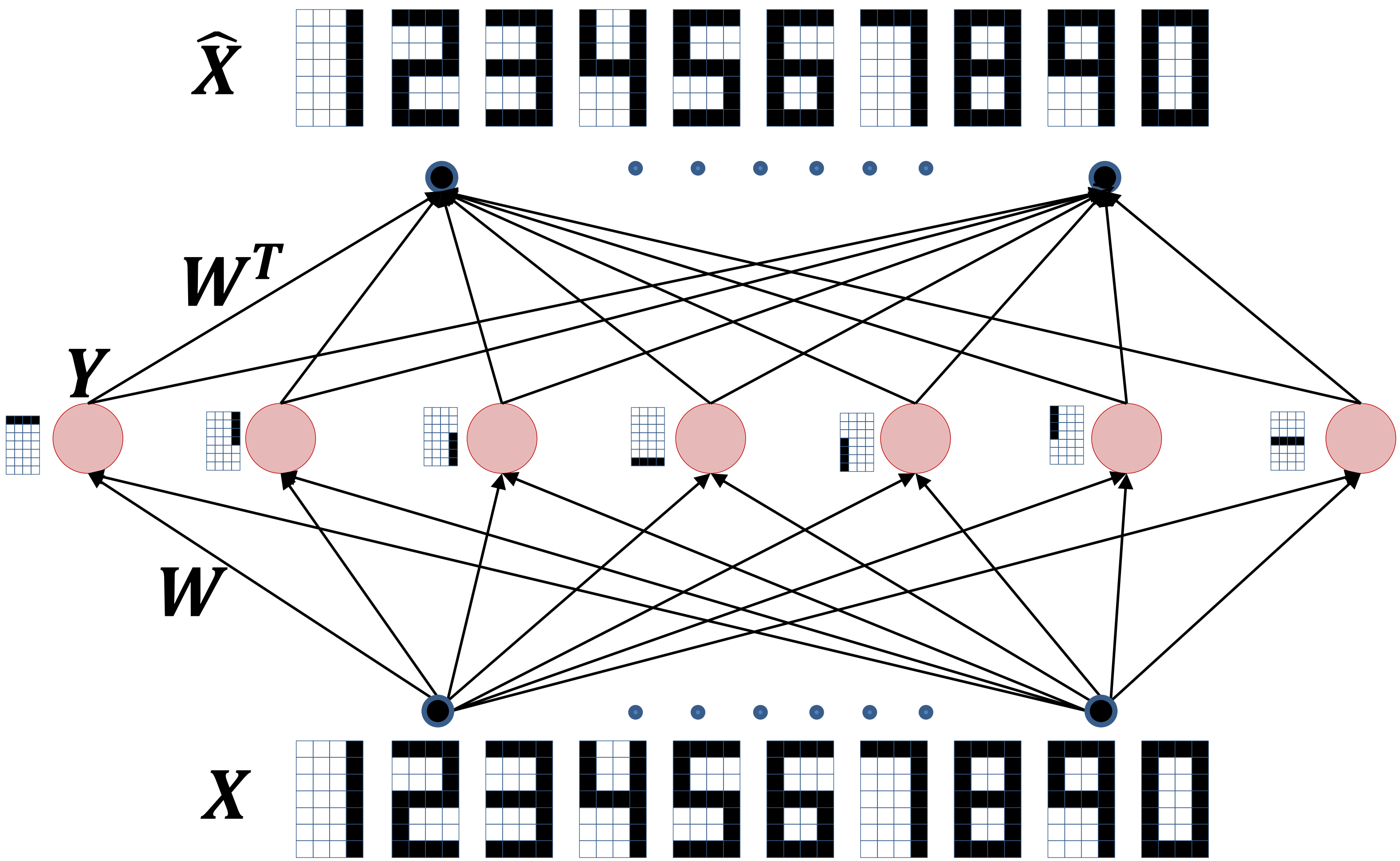
- **The signal could be (partially) reconstructed using these features**
  - **Will retain all the significant components of the signal**
- Simply *recompose* the detected features
  - Will this work?

Making it explicit



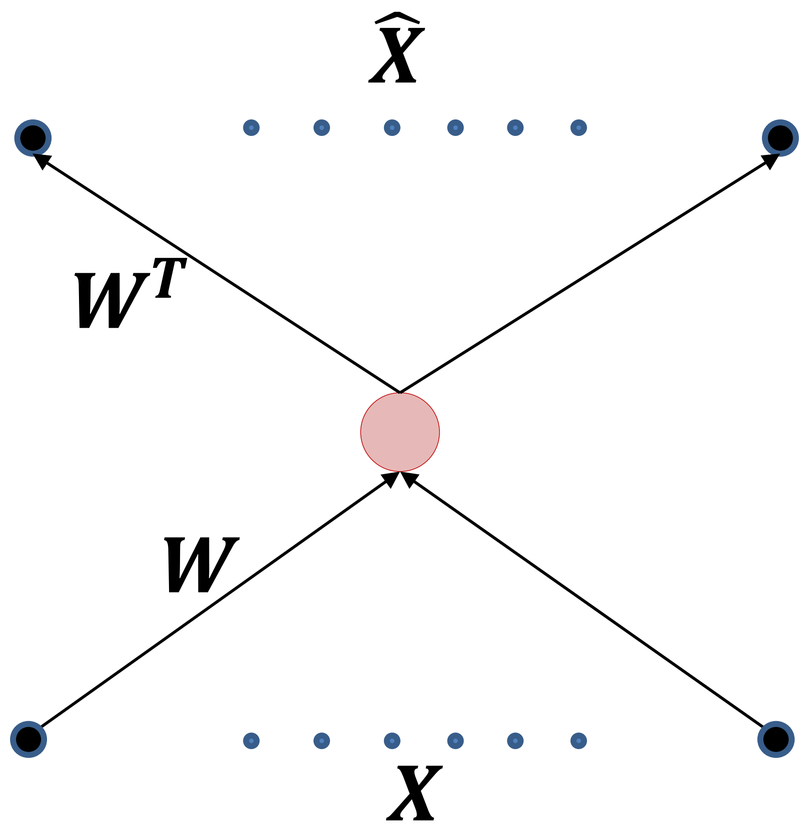
- **The signal could be (partially) reconstructed using these features**
  - **Will retain all the significant components of the signal**
- Simply *recompose* the detected features
  - Will this work?

Making it explicit: an autoencoder



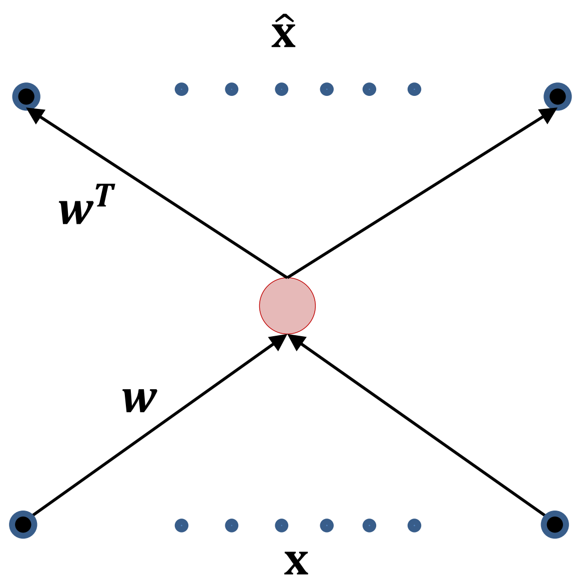
- A neural network can be trained to predict the input itself
- This is an *autoencoder*
- An *encoder* learns to detect all the most significant patterns in the signals
- A *decoder* recomposes the signal from the patterns

The Simplest Autencoder



- A single hidden unit
- Hidden unit has *linear* activation
- What will this learn?

The Simplest Autencoder



Training: Learning W by minimizing L2 divergence

$$\hat{\mathrm{x}} = w^T w \mathrm{x}$$

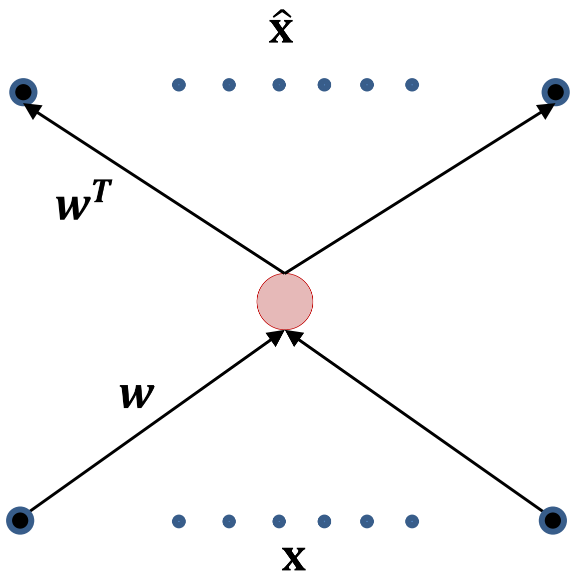
$$div(\hat{\mathrm{x}}, \mathrm{x}) = \|\mathrm{x} - \hat{\mathrm{x}}\|^2 = \|\mathrm{x} - \mathrm{w}^T w \mathrm{x}\|^2$$

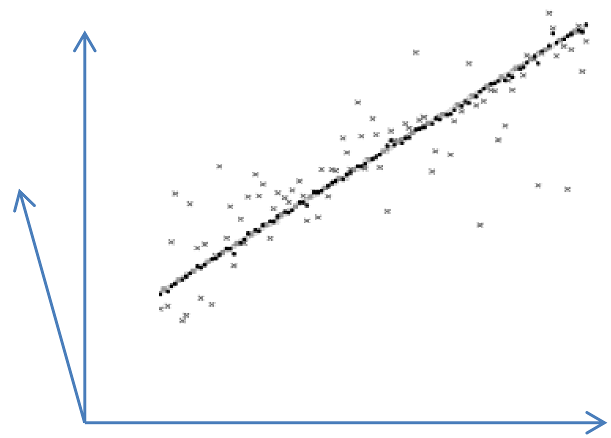
$$\widehat{W} = \underset{W}{\operatorname{argmin}} E[div(\hat{\mathrm{x}}, \mathrm{x})]$$

$$\widehat{W} = \underset{W}{\operatorname{argmin}} E[\|\mathrm{x} - \mathrm{w}^T w \mathrm{x}\|^2]$$

- This is just PCA!

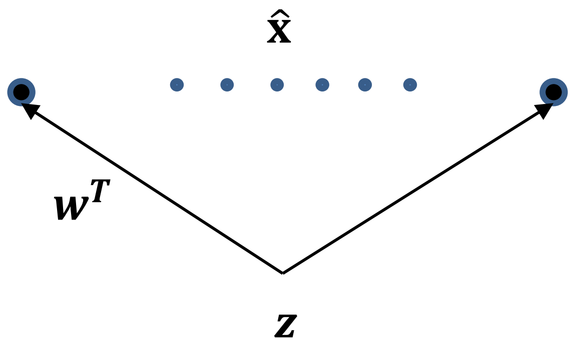
The Simplest Autencoder

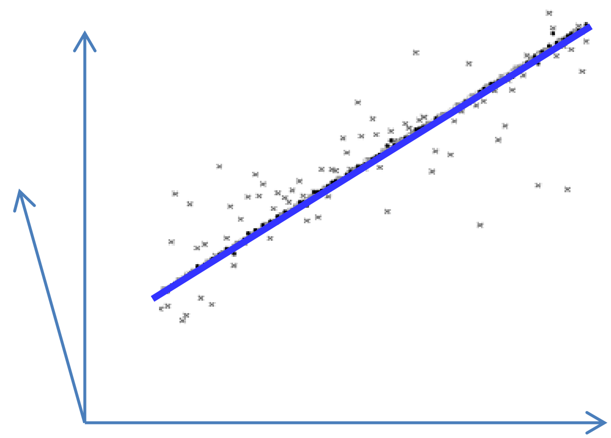




- The autoencoder finds the direction of maximum energy
  - Variance if the input is a zero-mean RV
- All input vectors are mapped onto a point on the principal axis

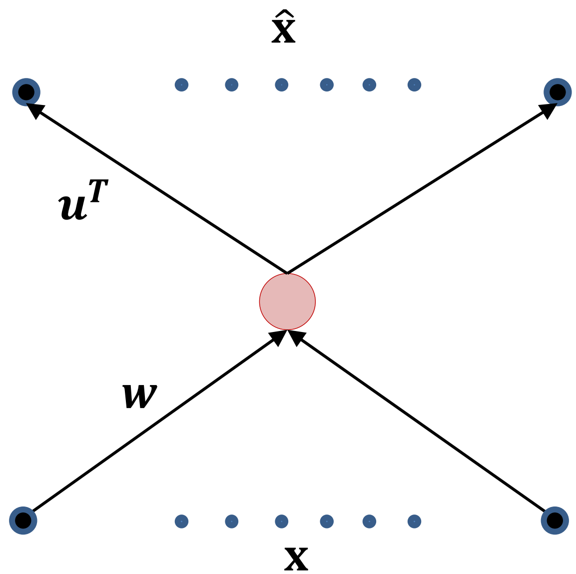
The Simplest Autencoder

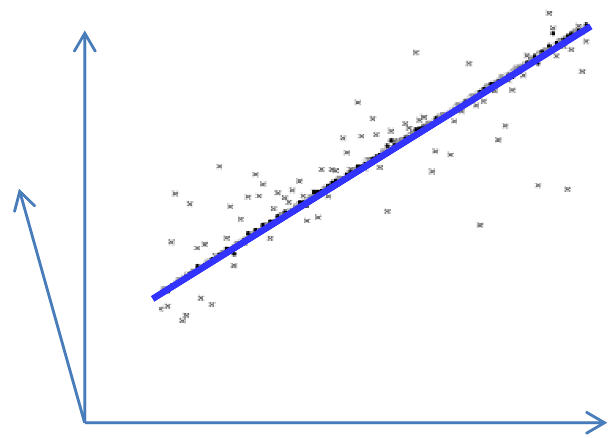




- Simply varying the hidden representation will result in an output that lies along the major axis

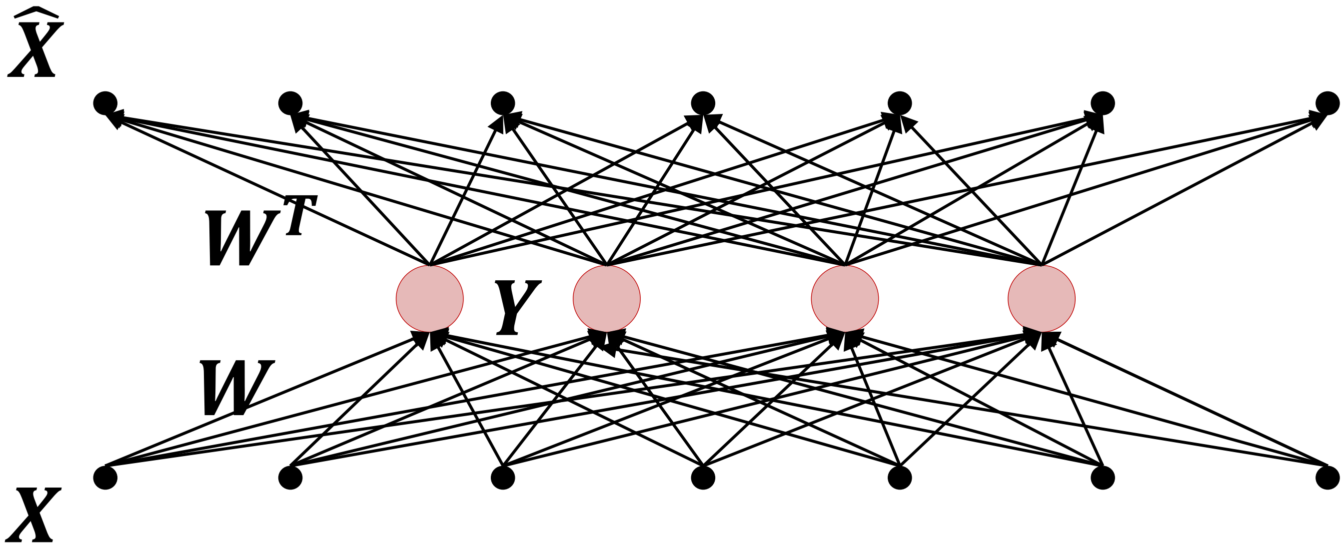
The Simplest Autencoder

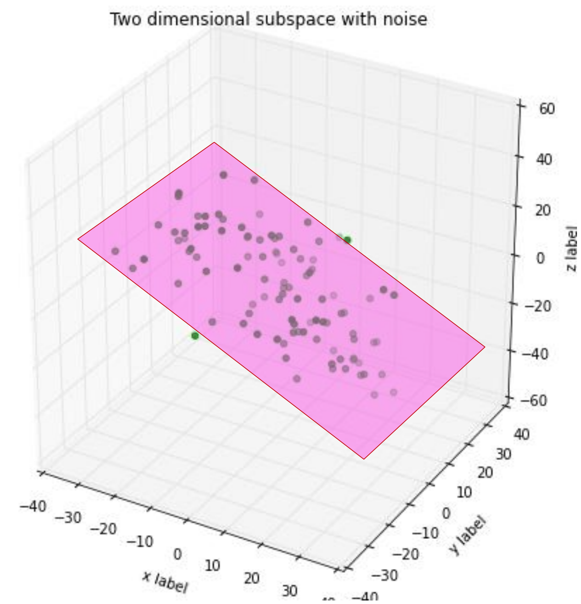




- Simply varying the hidden representation will result in an output that lies along the major axis
- This will happen even if the learned output weight is separate from the input weight
  – The minimum-error direction *is* the principal eigen vector

For more detailed AEs without a non-linearity

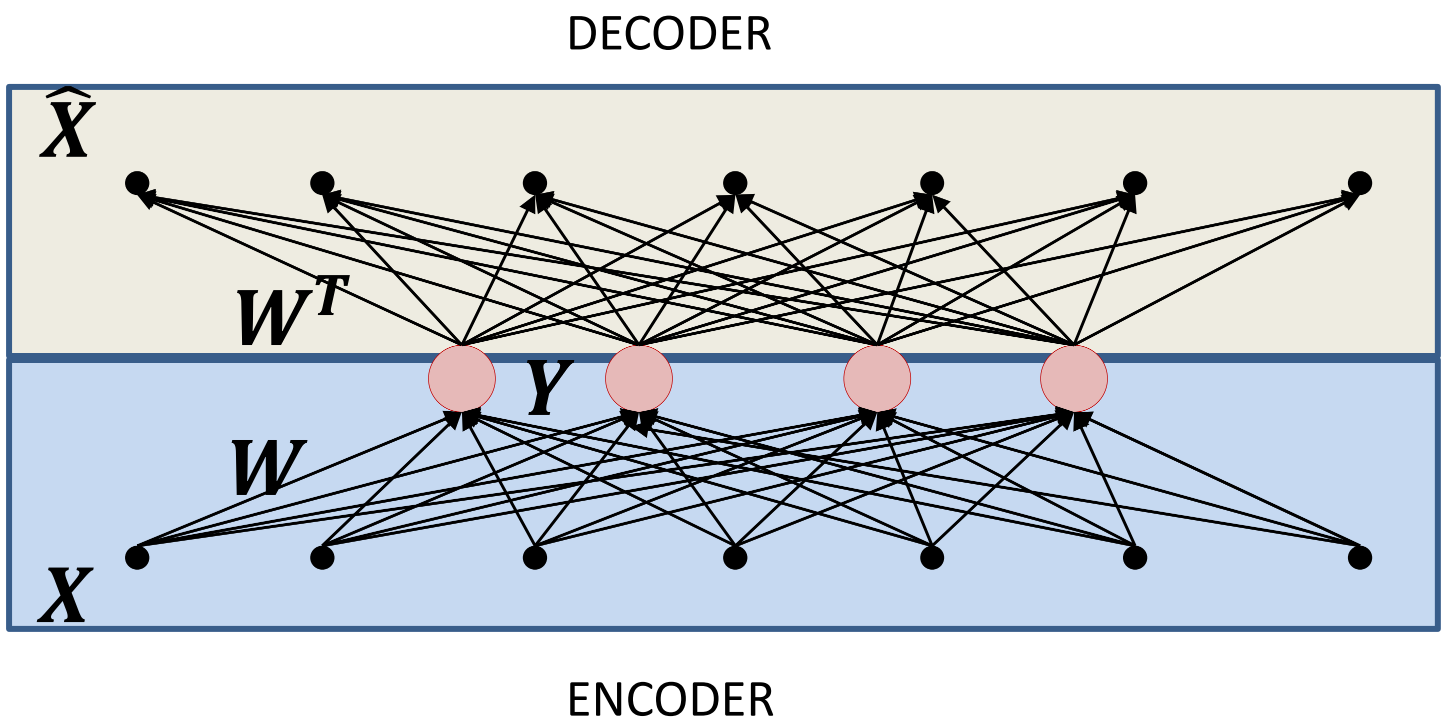




$\boxed{\mathbf{Y} = \mathbf{W}\mathbf{X}}$ $\boxed{\hat{\mathbf{X}} = \mathbf{W}^T\mathbf{Y}}$ $E = \|\mathbf{X} - \mathbf{W}^T\mathbf{W}\mathbf{X}\|^2$ Find W to minimize Avg[E]

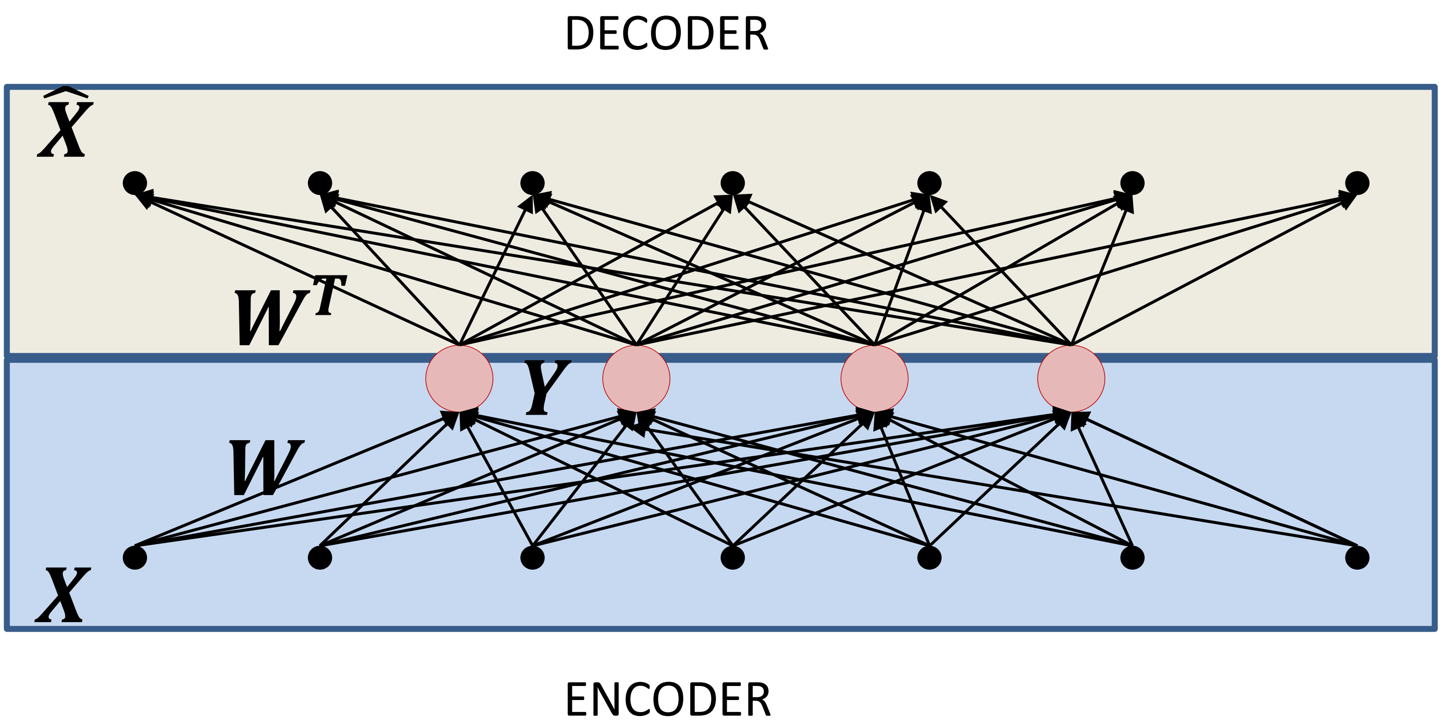
- This is still just PCA
  - The output of the hidden layer will be in the principal subspace
    - Even if the recomposition weights are different from the "analysis" weights

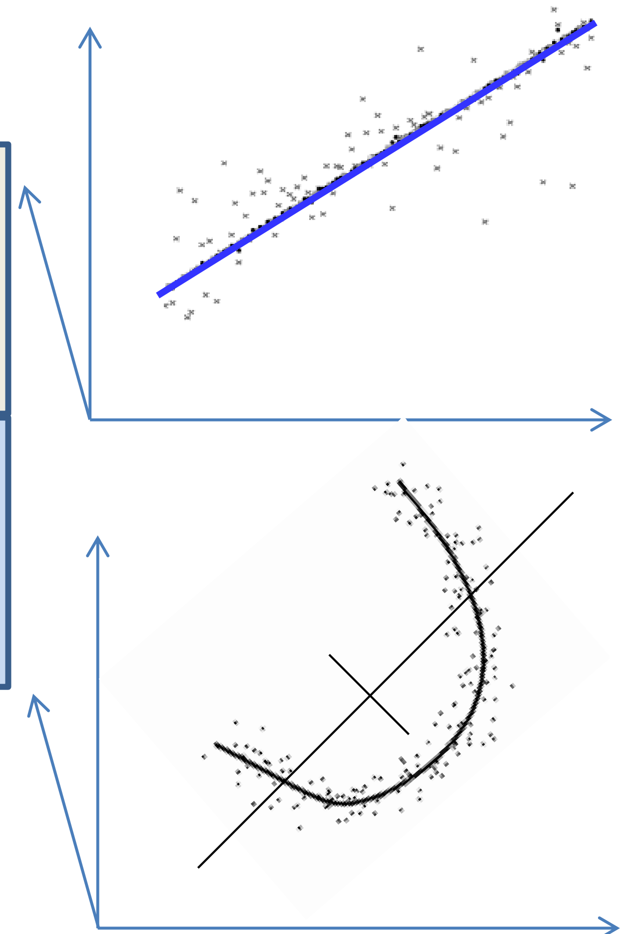
Terminology



- Terminology:
  - **Encoder:** The “Analysis” net which computes the hidden representation
  - **Decoder:** The “Synthesis” which recomposes the data from the hidden representation

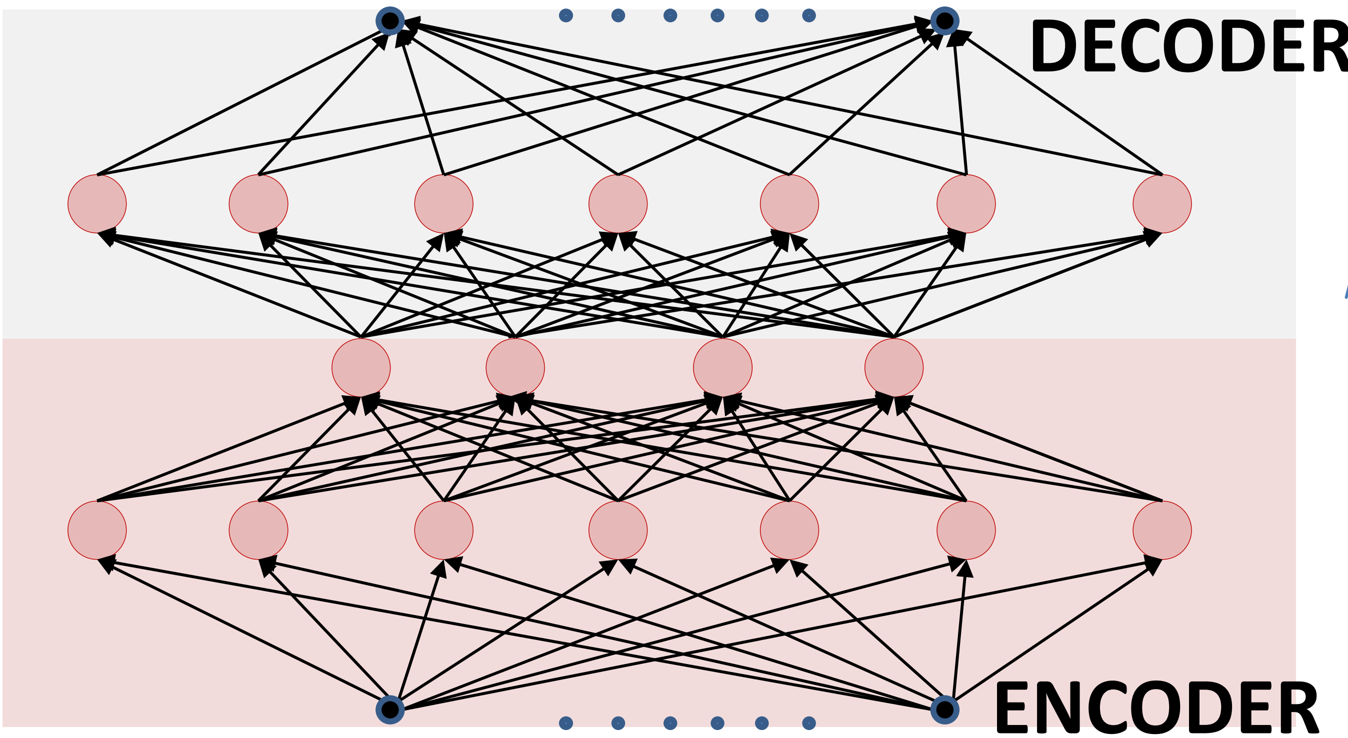
Introducing *nonlinearity*

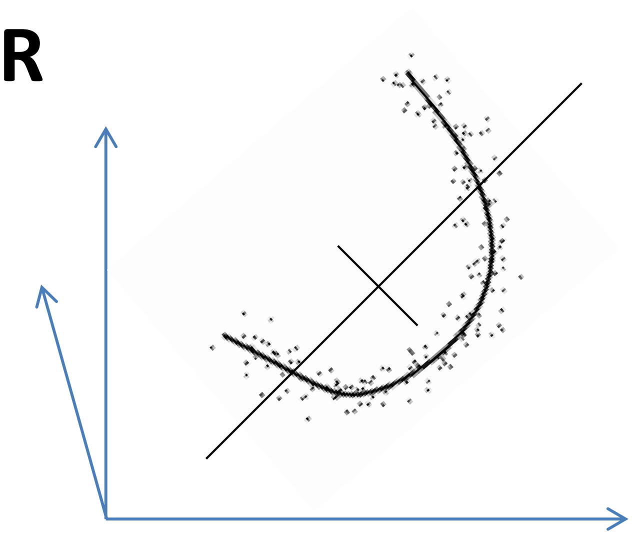




- When the hidden layer has a *linear* activation the decoder represents the best *linear* manifold to fit the data
  - Varying the hidden value will move along this linear manifold
- **When the hidden layer has non-linear activation, the net performs *nonlinear PCA***
  - The decoder represents the best non-linear manifold to fit the data
  - Varying the hidden value will move along this non-linear manifold

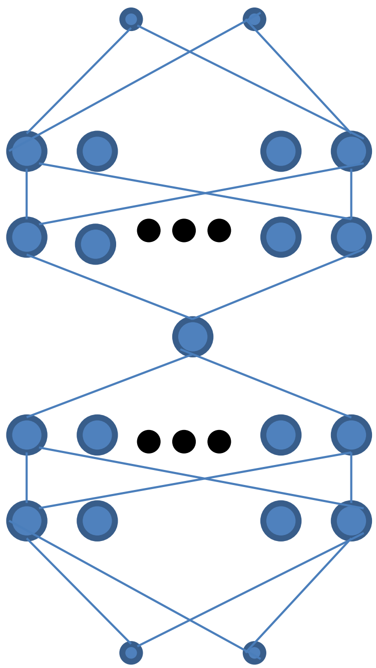
The AE

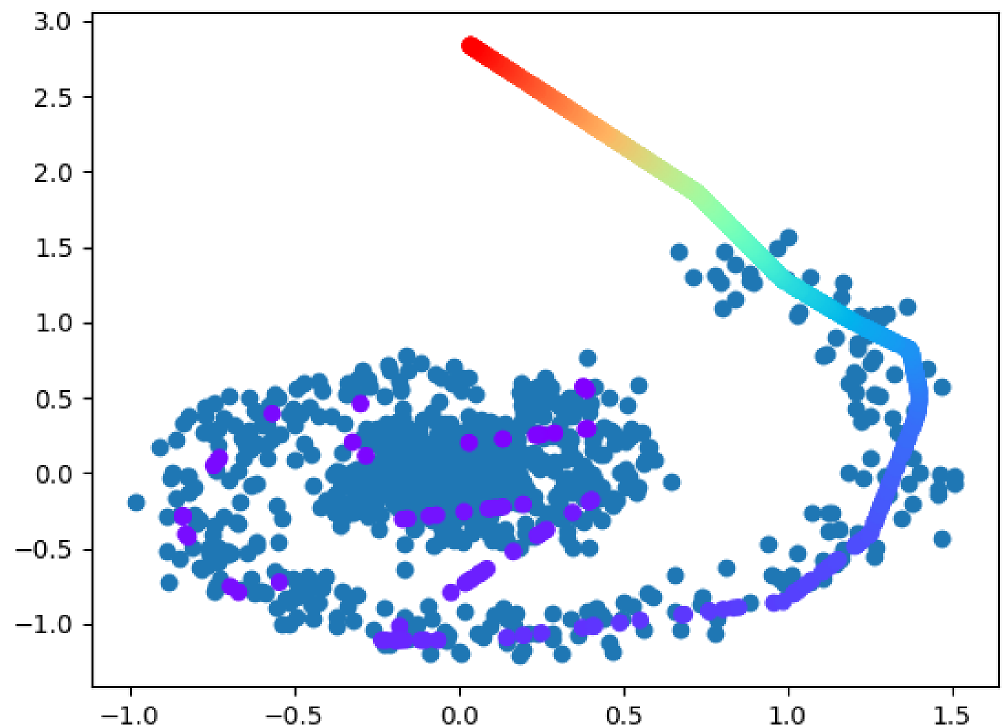




- With non-linearity
  - "Non linear" PCA
  - Deeper networks can capture more complicated manifolds
    - "Deep" autoencoders

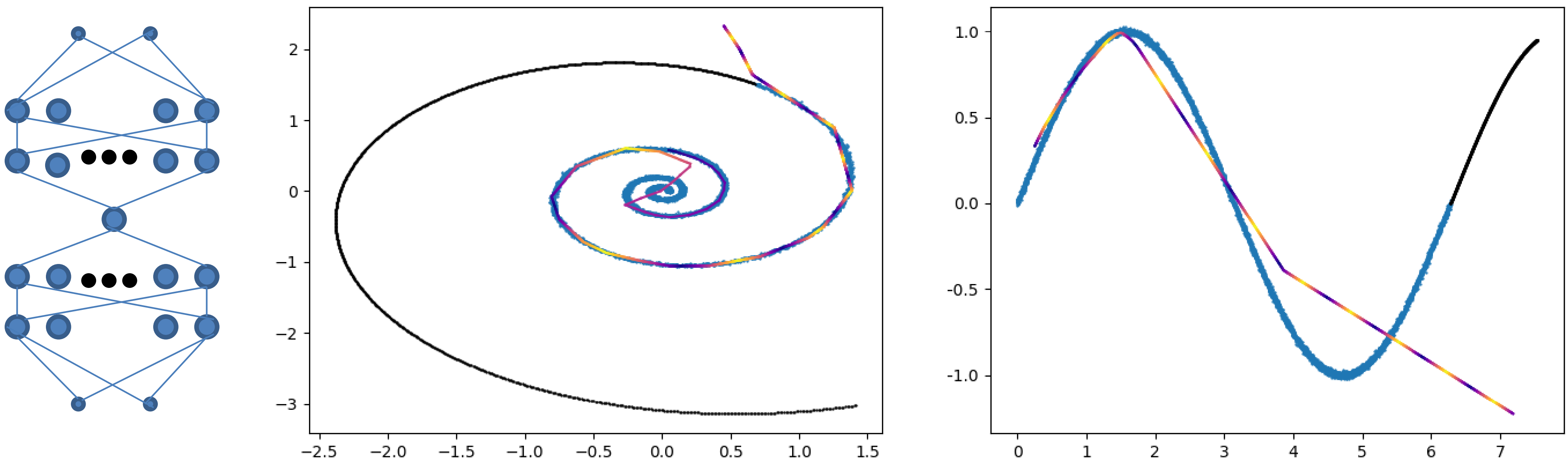
Some examples





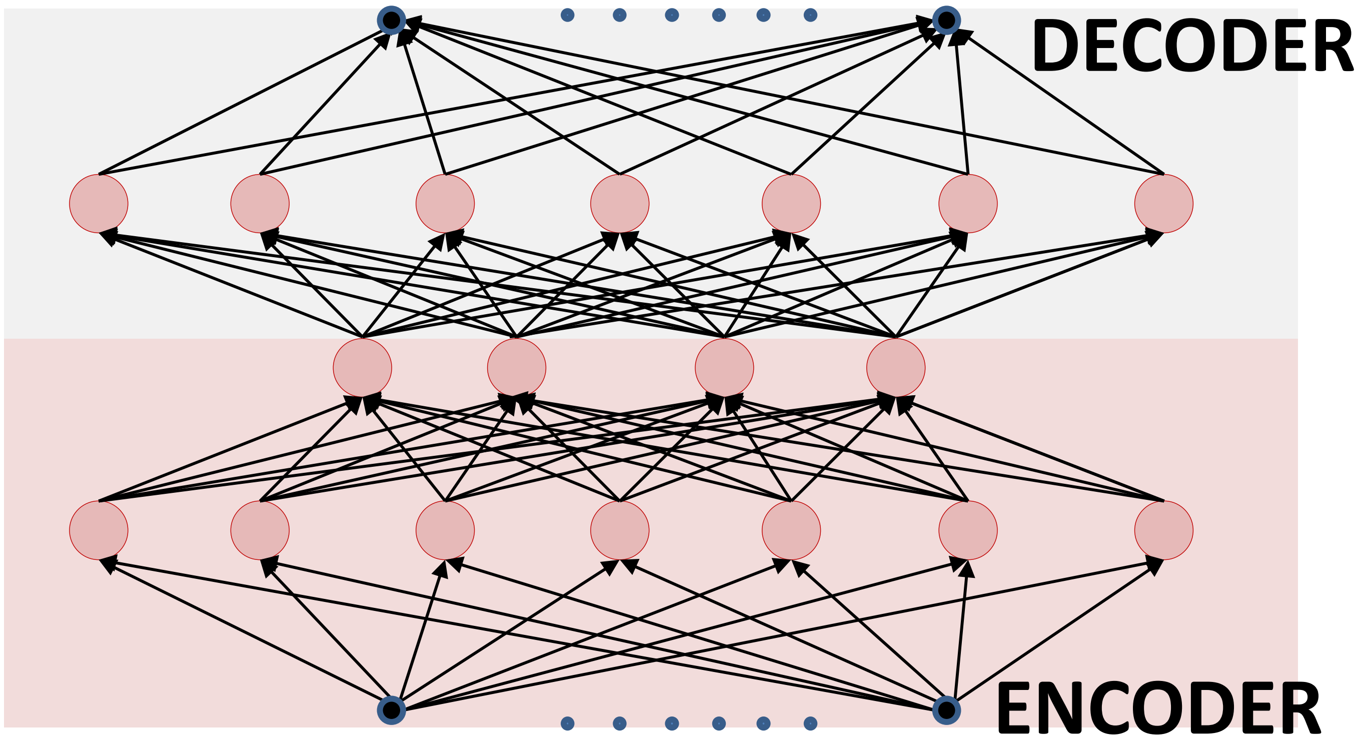
- 2-D input
- Encoder and decoder have 2 hidden layers of 100 neurons, but hidden representation is unidimensional
- Extending the hidden "z" value beyond the values seen in training extends the helix linearly

Some examples



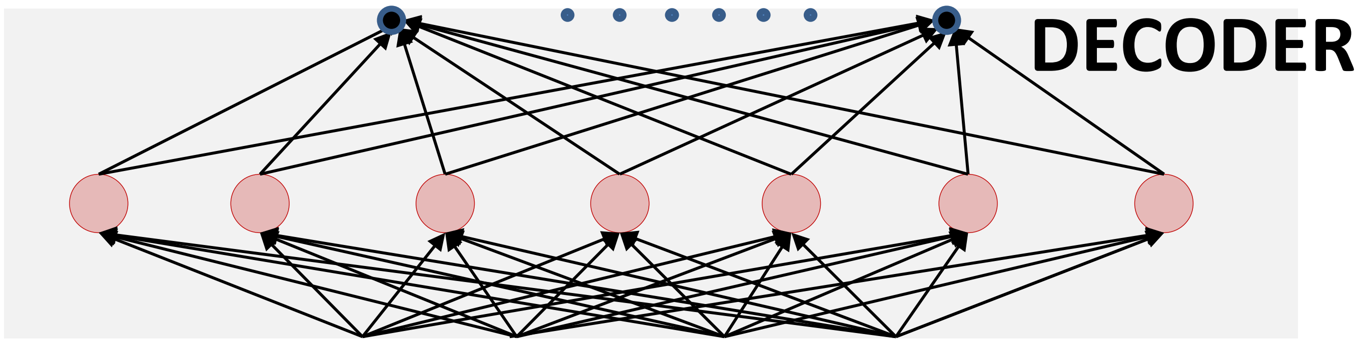
- The model is specific to the training data..
  – Varying the hidden layer value only generates data along the learned manifold
    - May be poorly learned
  – *Any input* will result in an output along the learned manifold

The AE



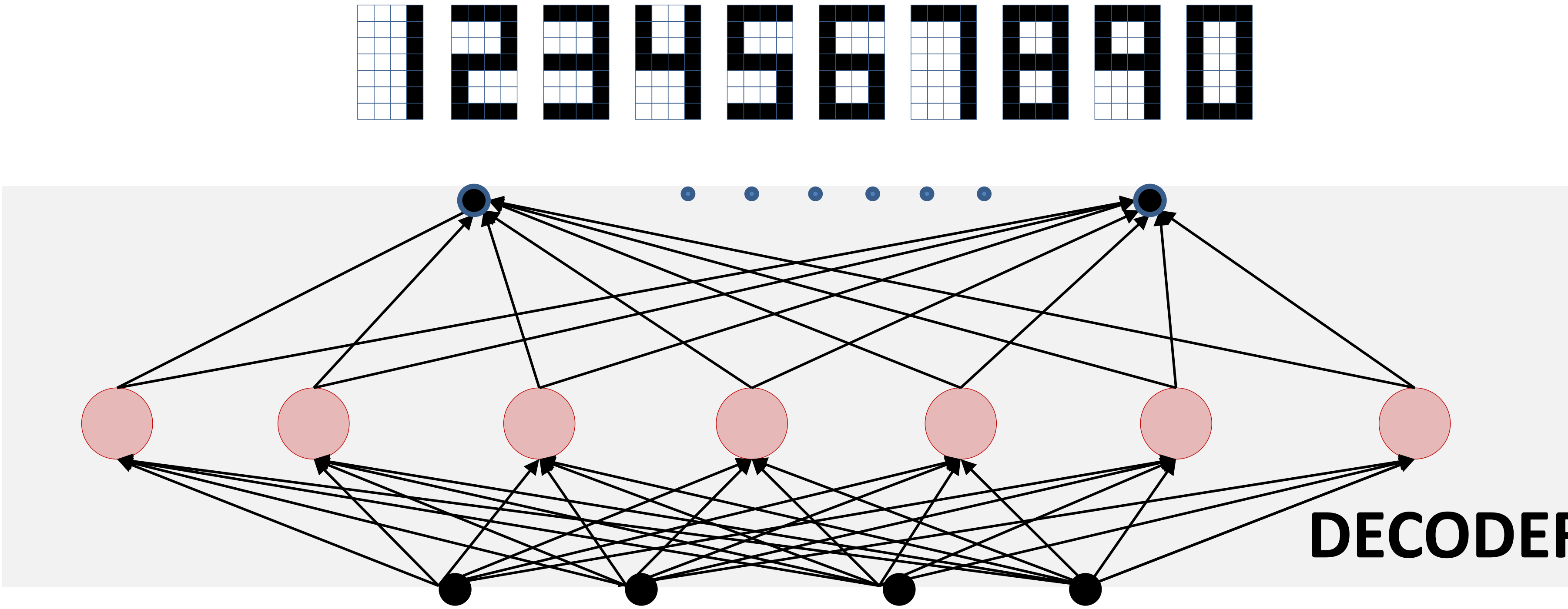
- When the hidden representation is of lower dimensionality than the input, often called a “**bottleneck**” network
  – Nonlinear PCA
  – Learns the manifold for the data
    - If properly trained

The AE



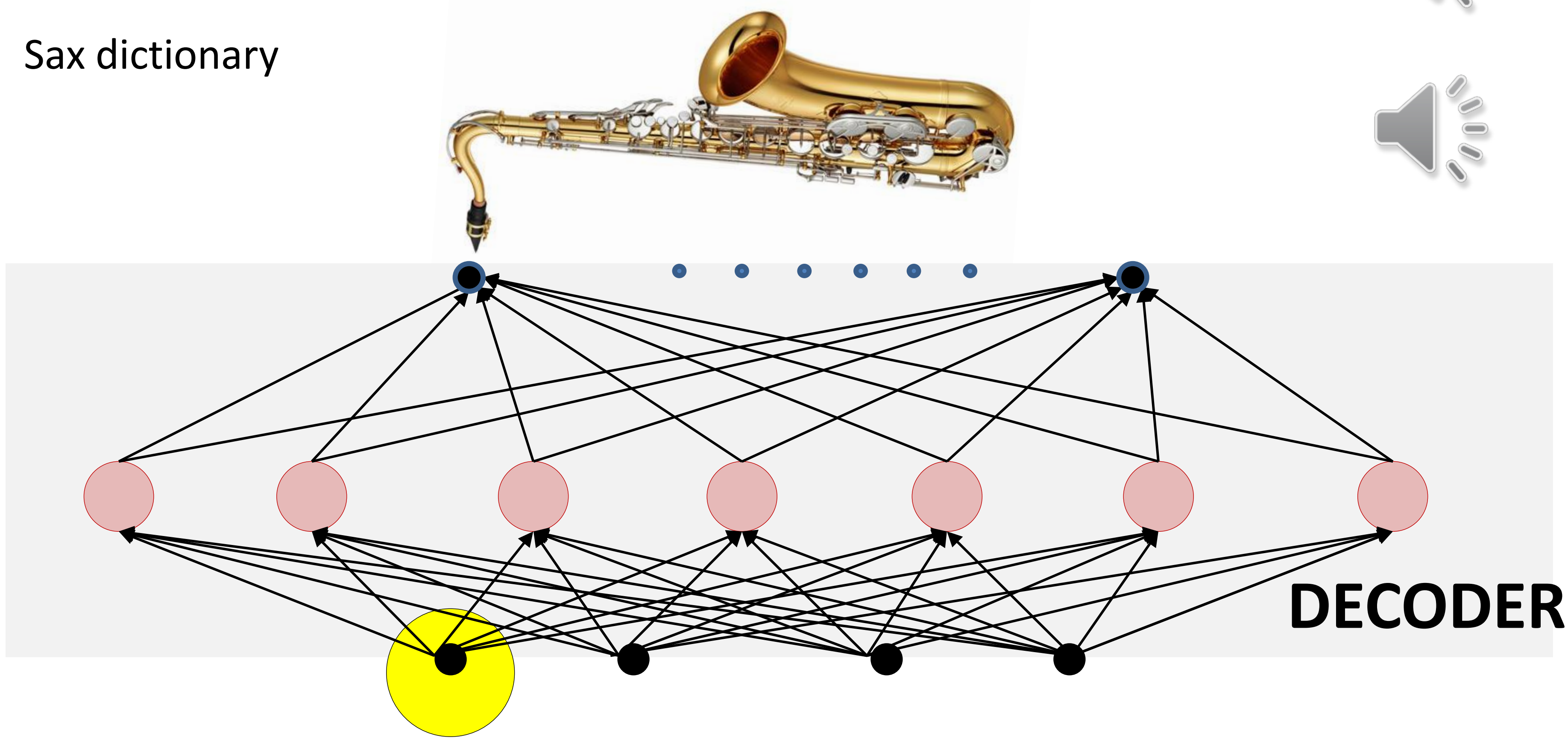
- The decoder can only generate data on the manifold that the training data lie on
- This also makes it an excellent "generator" of the distribution of the training data
  - Any values applied to the (hidden) input to the decoder will produce data similar to the training data

The Decoder:



- The decoder represents a source-specific generative *dictionary*
- Exciting it will produce typical data from the source!

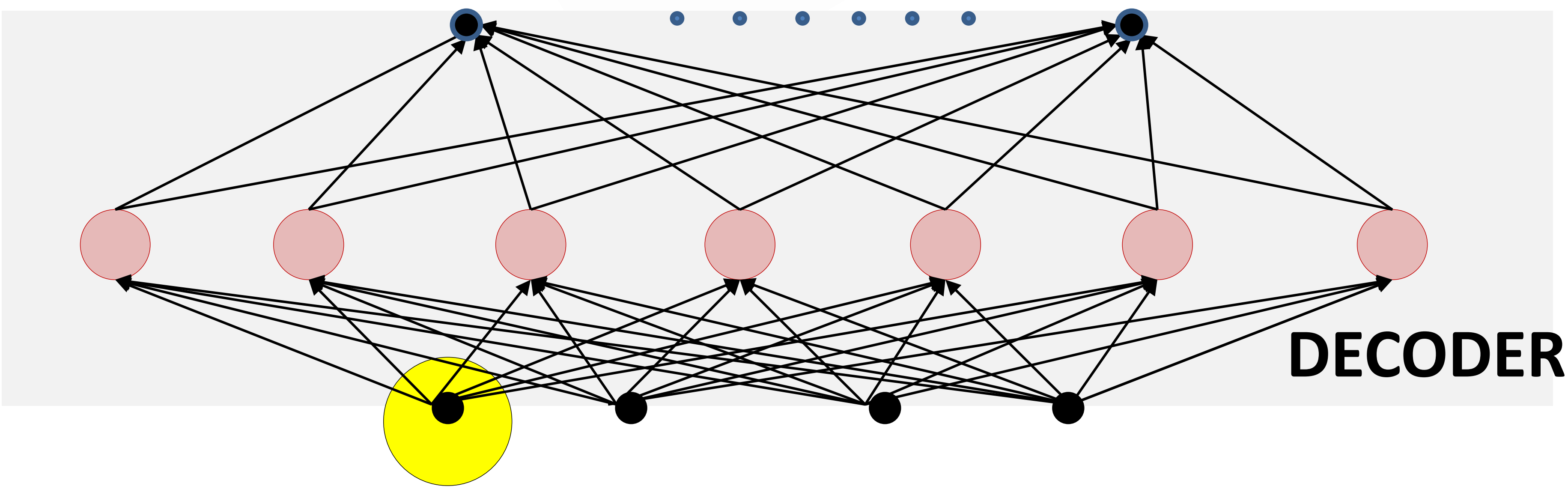
The Decoder:



- The decoder represents a source-specific generative *dictionary*
- Exciting it will produce typical data from the source!

The Decoder:

Clarinet dictionary



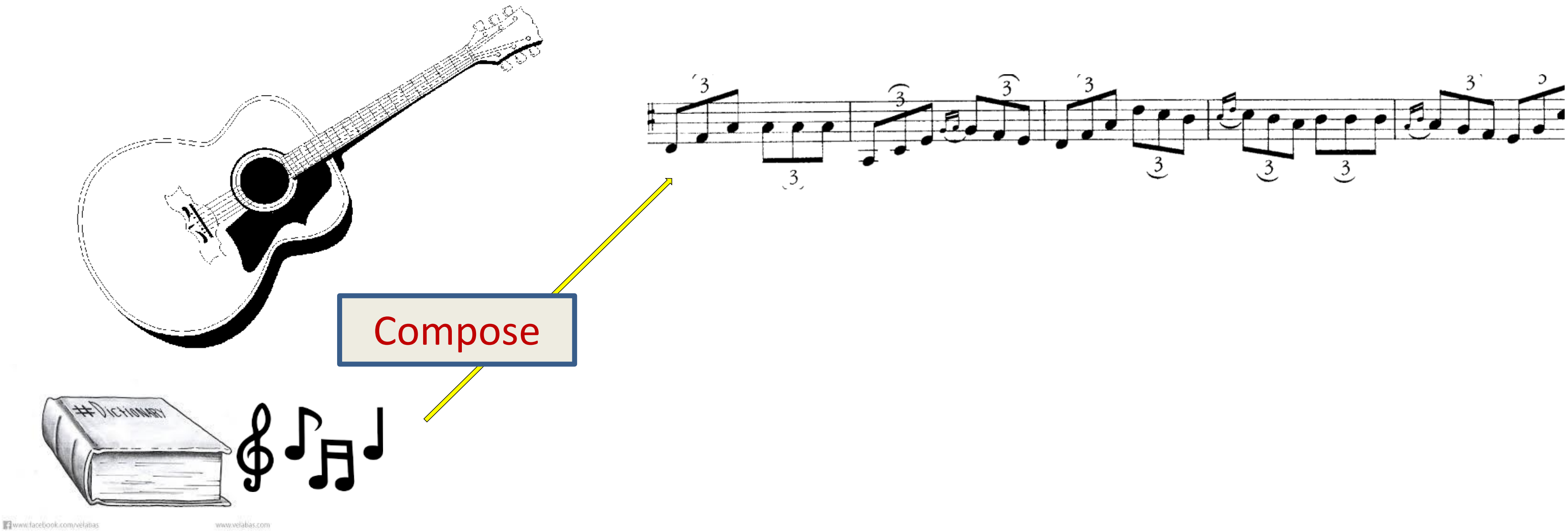
- The decoder represents a source-specific generative *dictionary*
- Exciting it will produce typical data from the source!

A cute application..

- Signal separation…

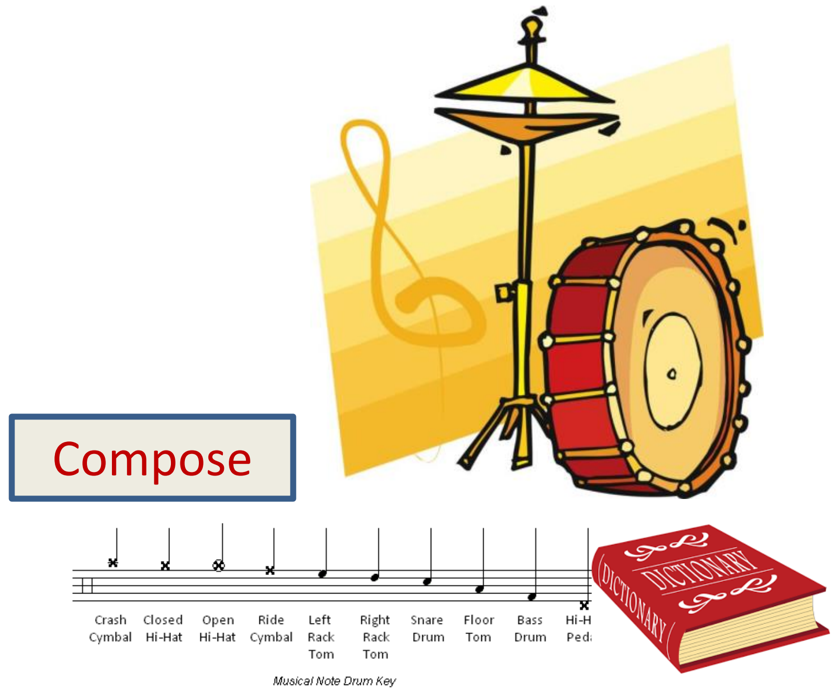
- Given a mixed sound from multiple sources, separate out the sources

Dictionary-based techniques



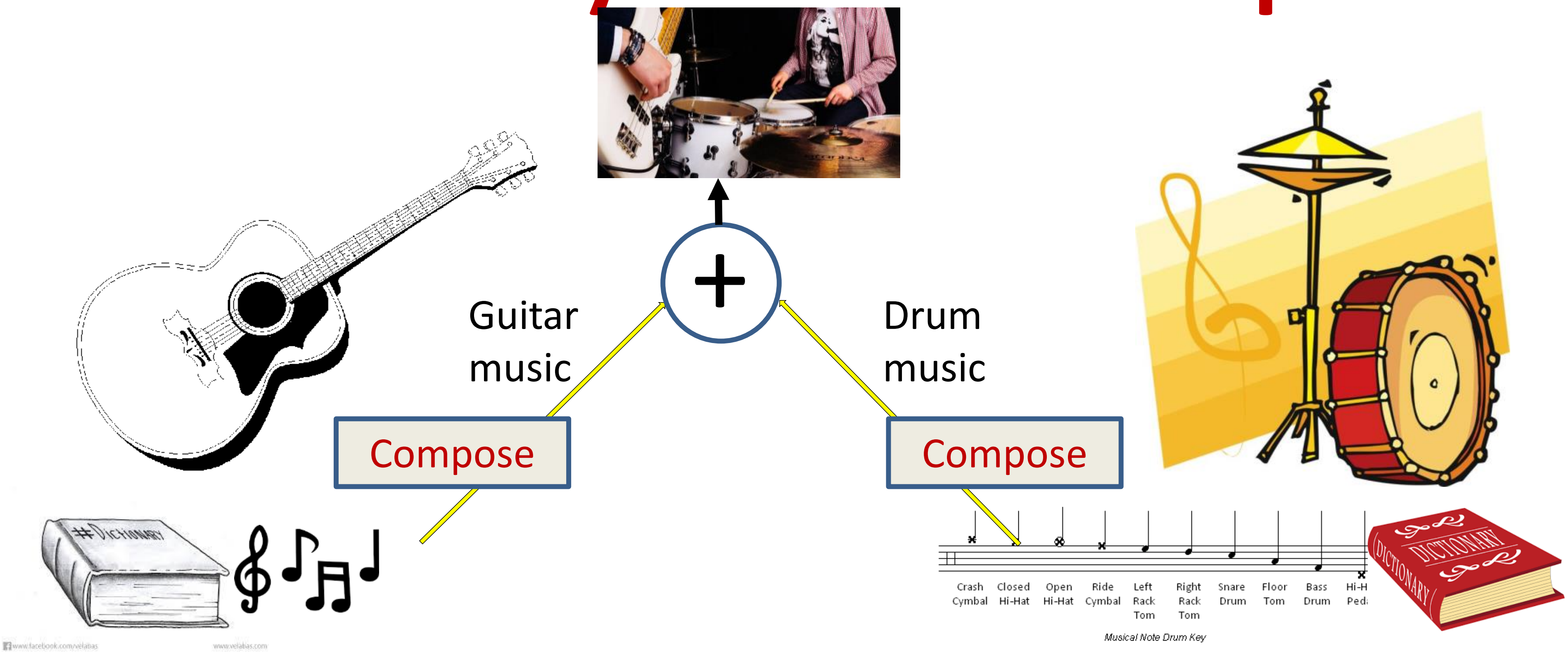
- Basic idea: Learn a dictionary of “building blocks” for each sound source
- All signals by the source are composed from entries from the dictionary for the source

Dictionary-based techniques



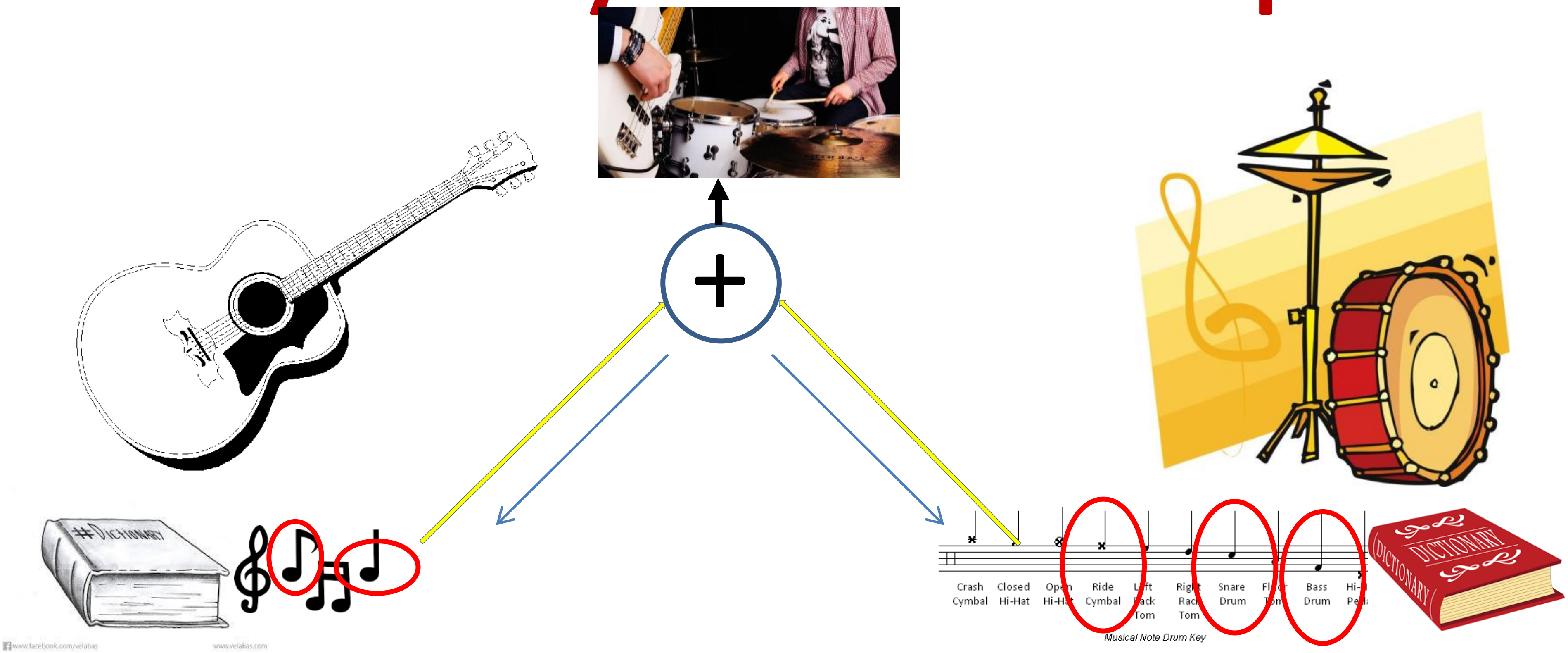
- Learn a similar dictionary for all sources expected in the signal

Dictionary-based techniques



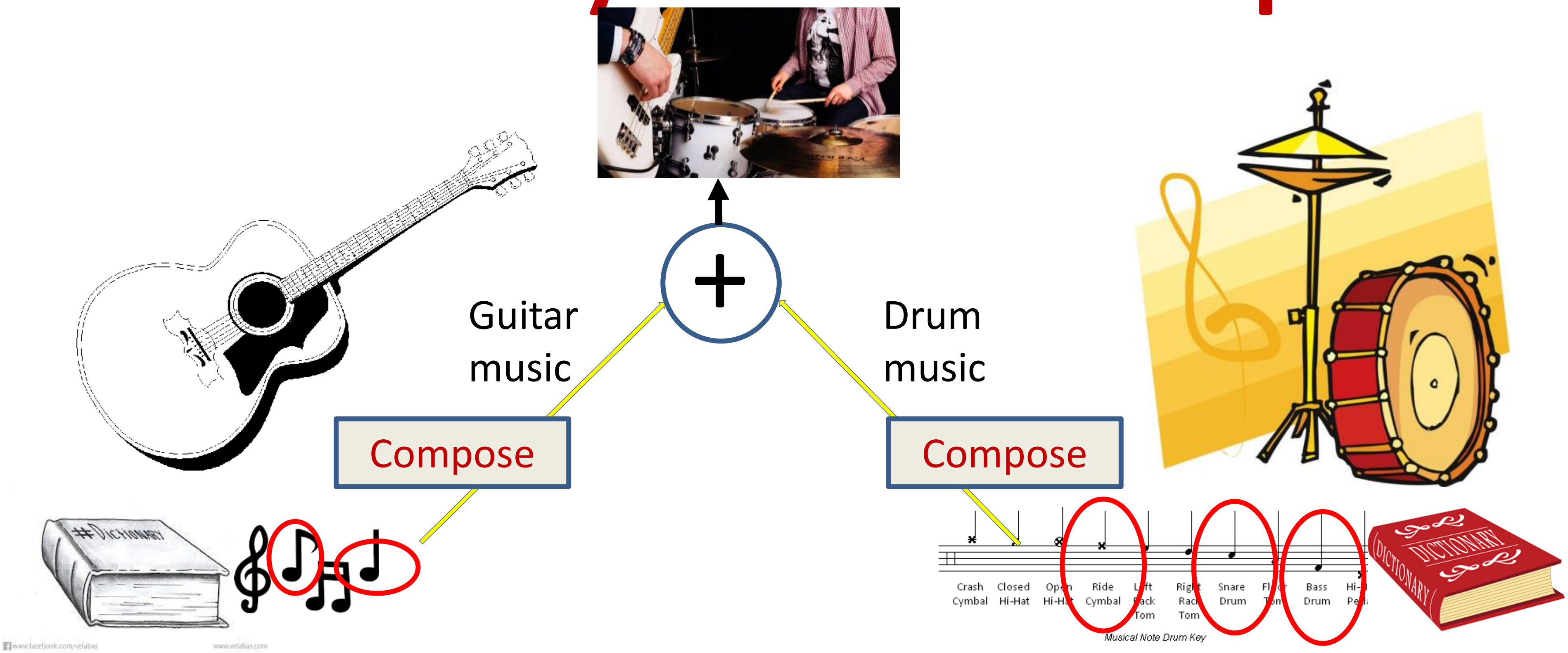
- A mixed signal is the linear combination of signals from the individual sources
  - Which are in turn composed of entries from its dictionary

Dictionary-based techniques



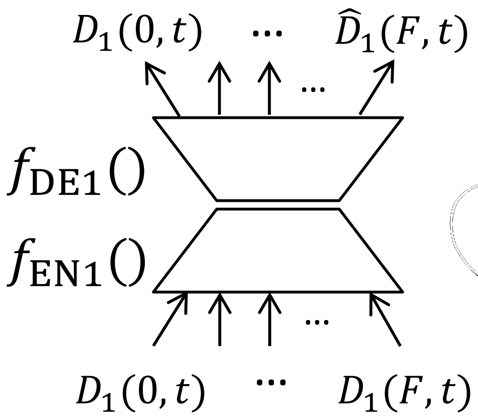
- Separation: Identify the combination of entries from both dictionaries that compose the mixed signal

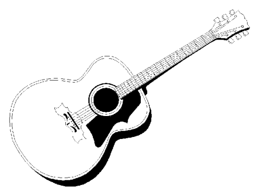
Dictionary-based techniques

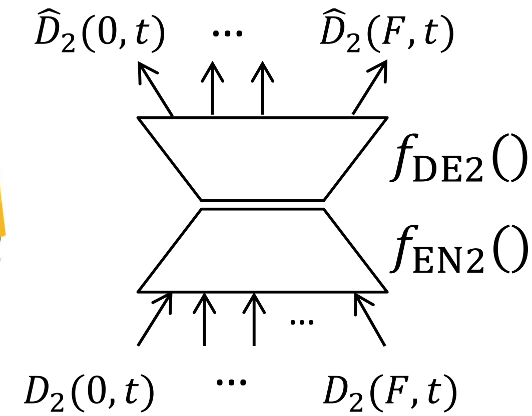


- Separation: Identify the combination of entries from both dictionaries that compose the mixed signal
  - The composition from the identified dictionary entries gives you the separated signals

Learning Dictionaries



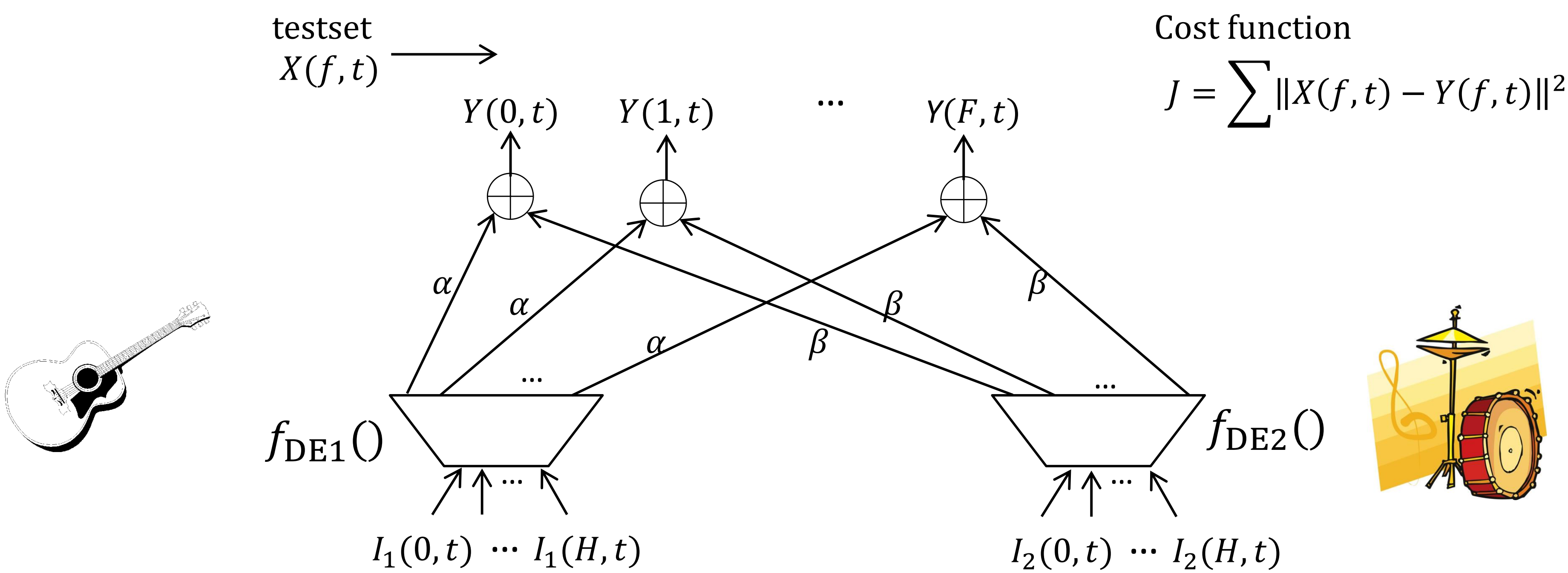




- Autoencoder dictionaries for each source
  - Operating on (magnitude) spectrograms

- For a well-trained network, the "decoder" dictionary is highly specialized to creating sounds for that source

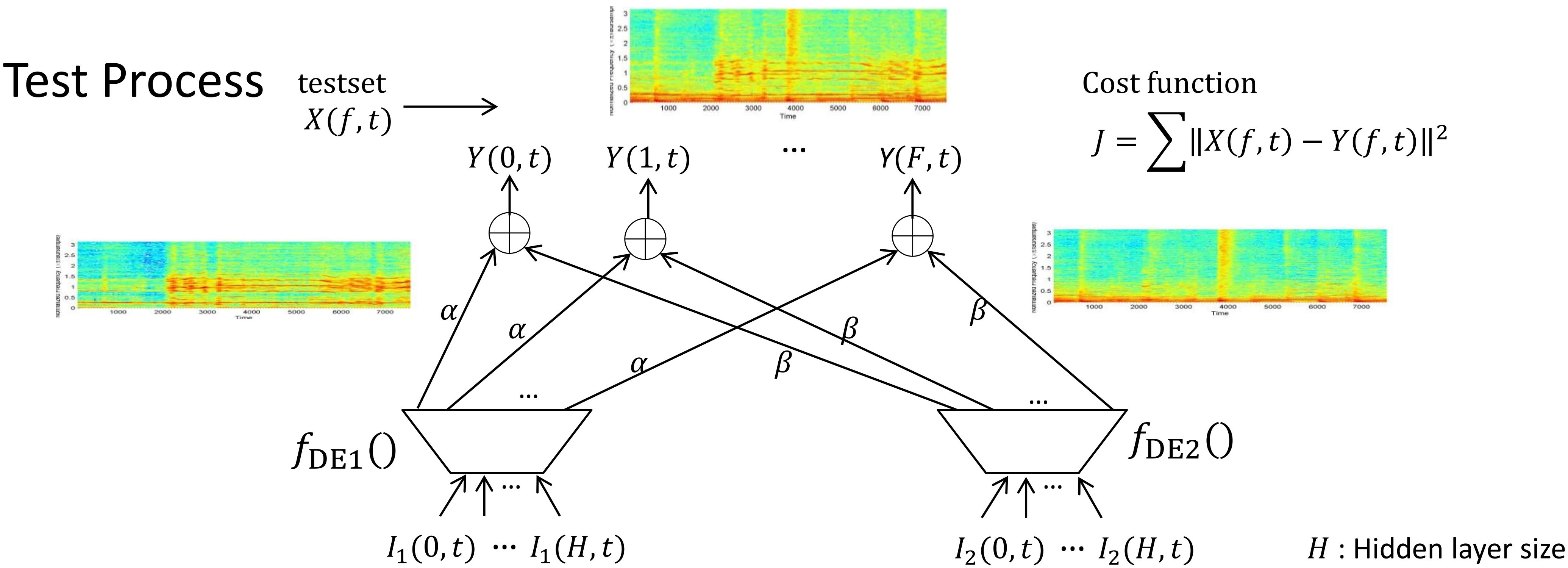
Model for mixed signal



Estimate $I_1()$ and $I_2()$ to minimize cost function $J(\,)$

- The sum of the outputs of both neural dictionaries
  - For some unknown input

Separation



Estimate $I_1()$ and $I_2()$ to minimize cost function $J(\ )$

- Given mixed signal and source dictionaries, find excitation that best recreates mixed signal
  - Simple backpropagation
- Intermediate results are separated signals

Example Results

Mixture | Separated | Separated

Original | Original

5-layer dictionary, 600 units wide

- Separating music

Story for the day

- Classification networks learn to predict the *a posteriori* probabilities of classes
  - The network until the final layer is a feature extractor that converts the input data to be (almost) linearly separable
  - The final layer is a classifier/predictor that operates on linearly separable data
- Neural networks can be used to perform linear or non-linear PCA
  - "Autoencoders"
  - Can also be used to compose constructive dictionaries for data
    - Which, in turn can be used to model data distributions