CS1138

Machine Learning

Lecture : Introduction to Neural Networks

(Slide Credits: Prof Bhiksha Raj, and Andrew Ng)

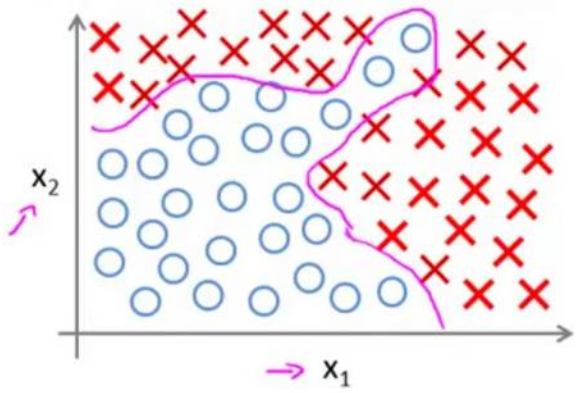
Arpan Gupta

Overview

- Motivation
- Perceptrons
- How perceptrons model boolean functions
- Universal function approximator
- Multi-layer perceptrons

Why do we need Neural Networks?

- Need to learn complex non-linear decision boundaries.



x_1 = size
x_2 = # bedrooms
x_3 = # floors
x_4 = age
...
x_{100}

n=100

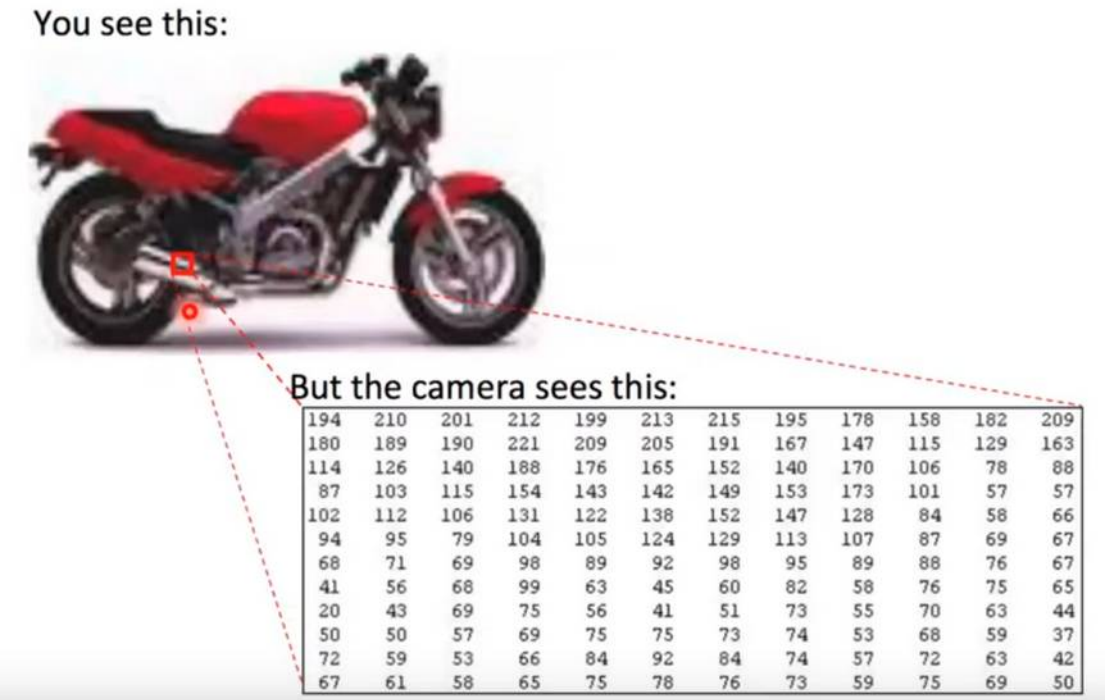
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

For logistic regression, we may need higher order terms. If we only include 2nd order terms, then for n=100, we need approx. 5000 features.

For many ML tasks, n will be quite large.

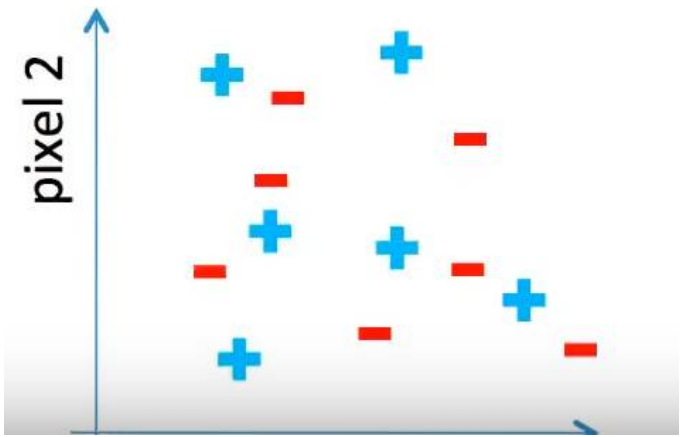
Task: Object Classification

- Given an unseen image, we want to develop a model that will classify the image, based on the object present in it.
- What the computer sees?

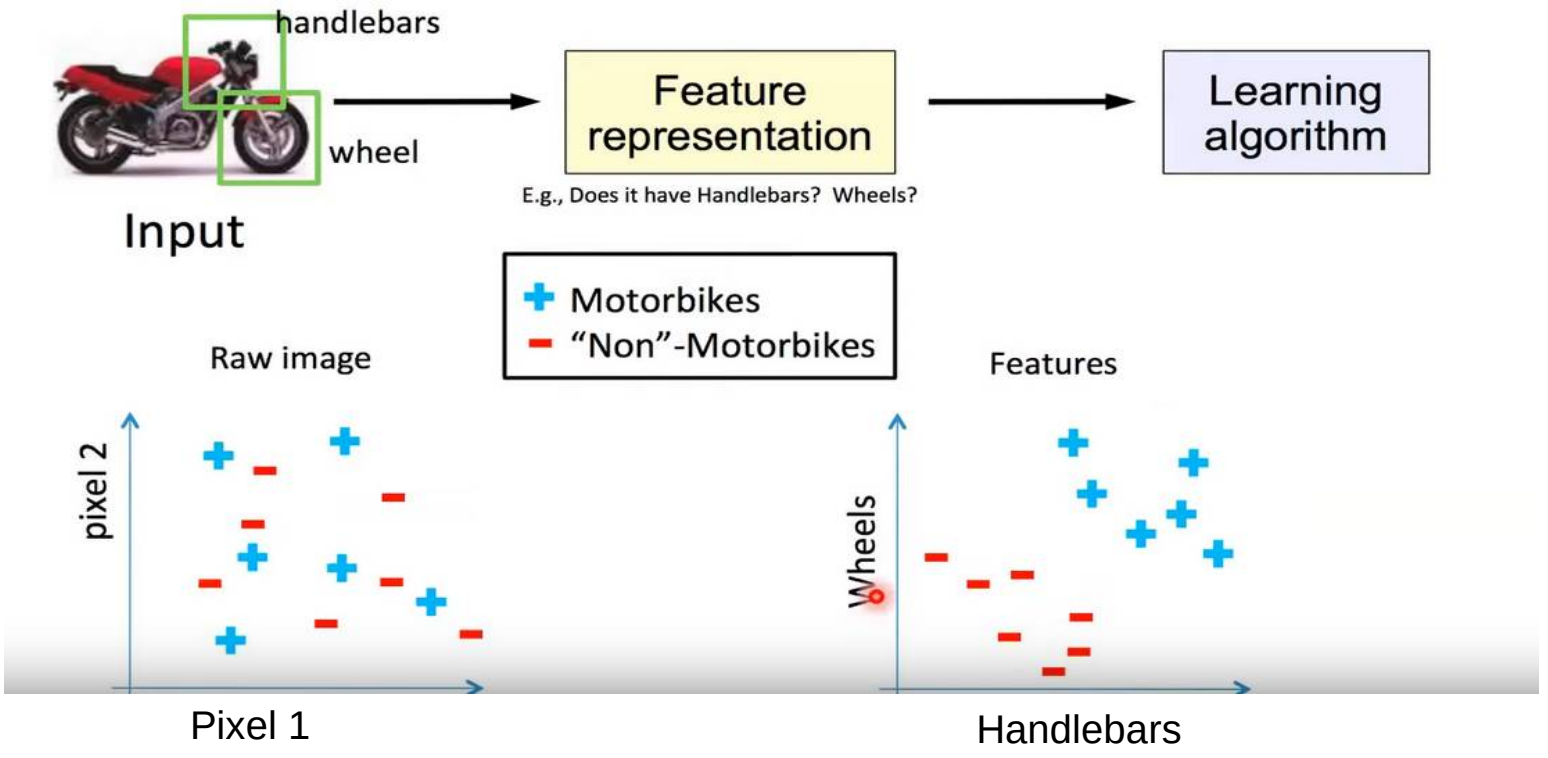


Pixel based representation

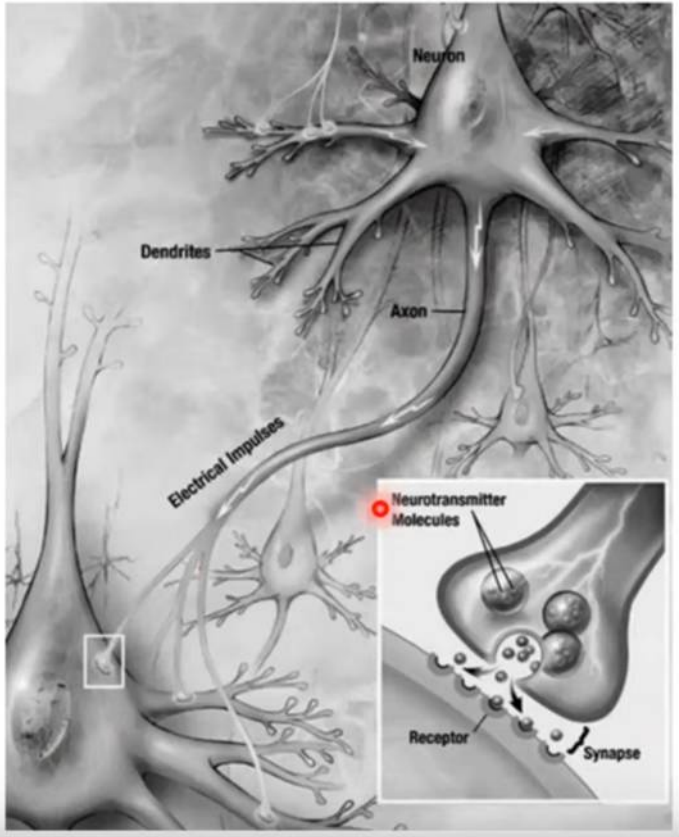
- The pixel positions can be chosen as features, but it will not have clear boundaries between different object classes.



Deciding features that are relevant



Neurons in the Brain



- Brain is composed of **neurons**
- A neuron receives input from other neurons (generally thousands) from its synapses
- Inputs are approximately **summed**
- When the input exceeds a threshold the neuron sends an electrical spike that travels from the body, down the axon, to the next neuron(s)

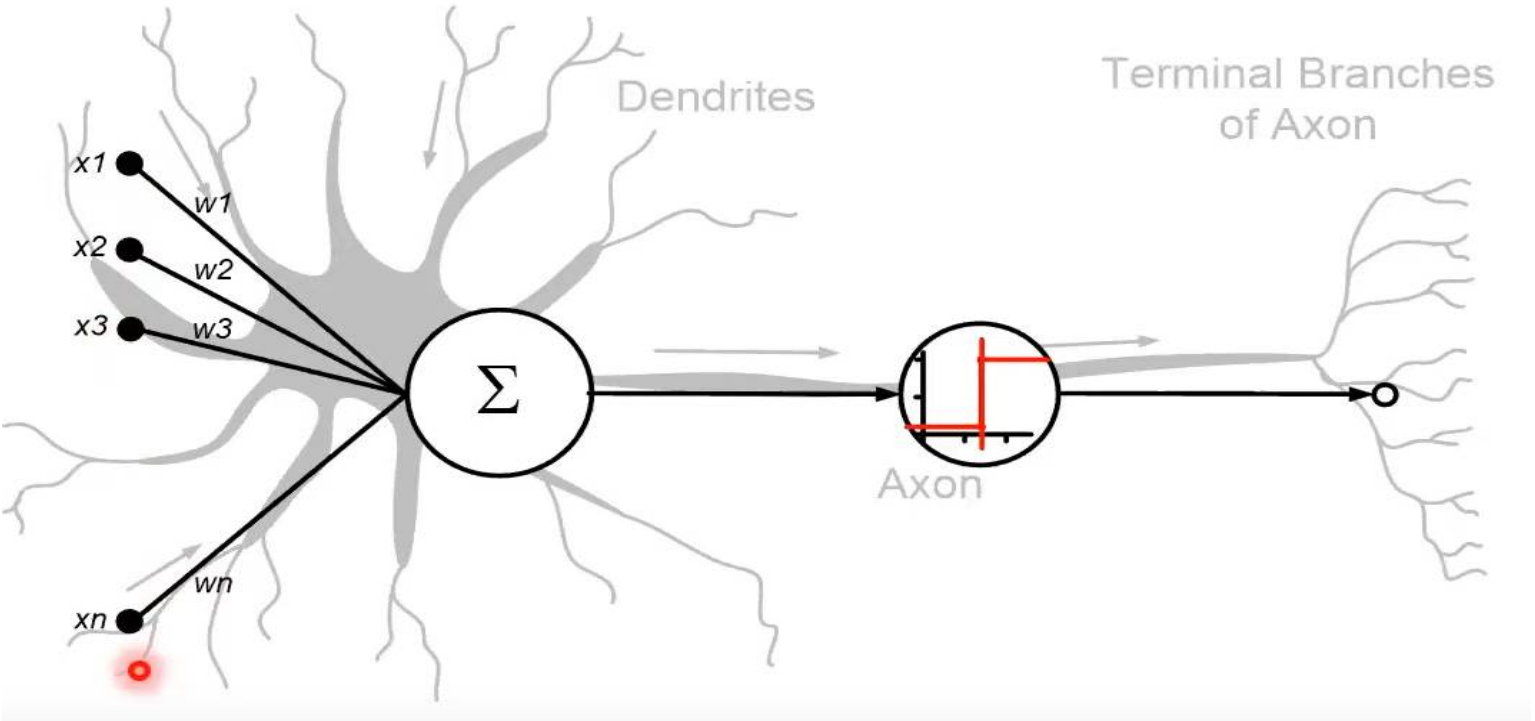
Introduction to Neural Networks

- Brain is a remarkable computer

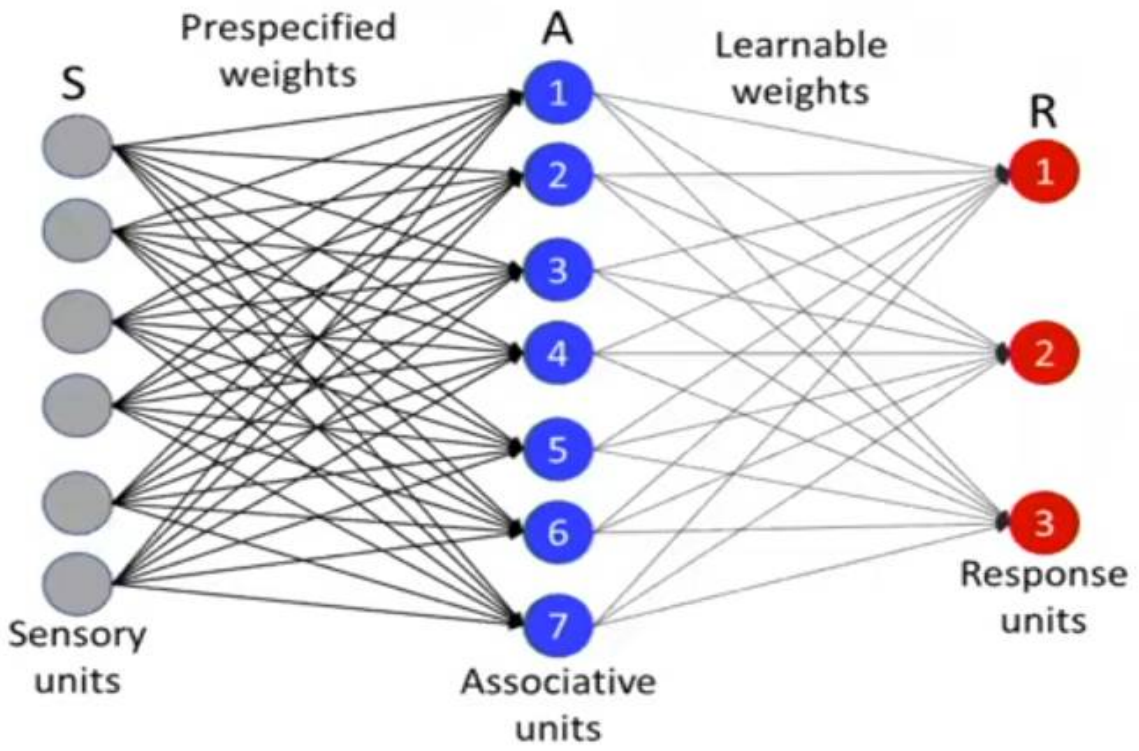
- 200 billion neurons, 32 trillion synapses
- Energy use: 25W
- Processing speed: 100 Hz
- Parallel, Distributed
- Fault Tolerant
- Learns: Yes
- Intelligent/Conscious: Usually

- 1 billion bytes RAM but trillions of bytes on disk
- Energy watt: 30-90W (CPU)
- Processing speed: 10^9 Hz
- Serial, Centralized
- Generally not Fault Tolerant
- Learns: Some
- Intelligent/Conscious: Generally No

Computational Implementation of Neural Activation Function

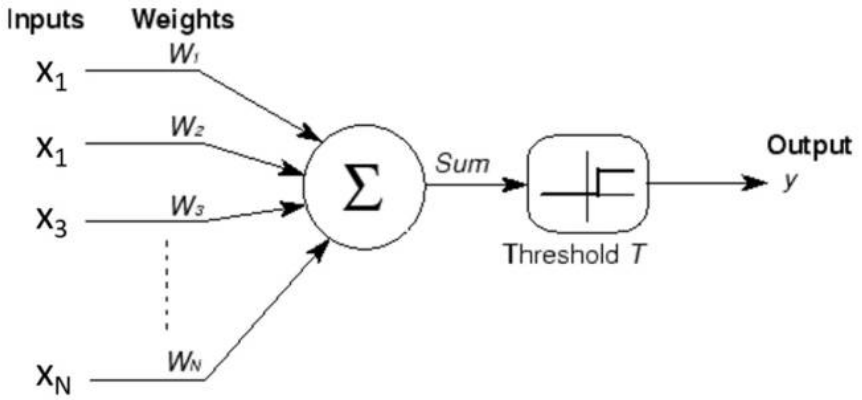


Frank Rosenblatt's Perceptron Model



Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Perceptron: Simplified Model (single response unit)



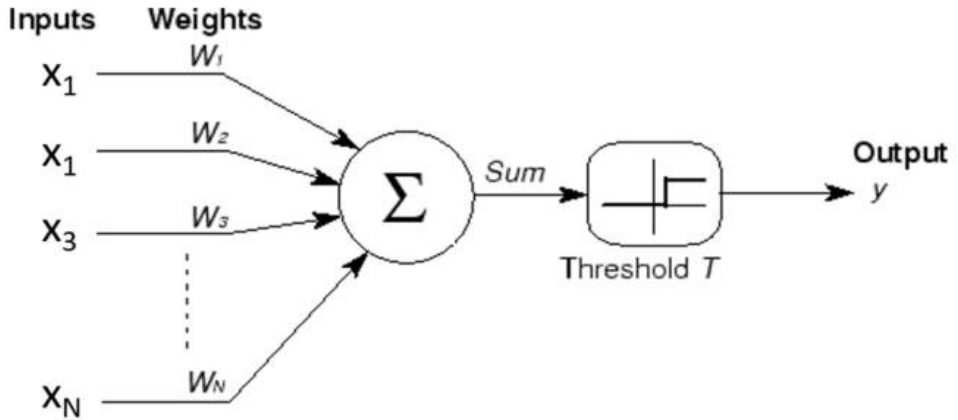
- Number of inputs combine linearly
  - Threshold logic: Fire if combined input exceeds threshold

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

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Universal Model

- 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



Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Also provided a learning algorithm

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

Sequential Learning:

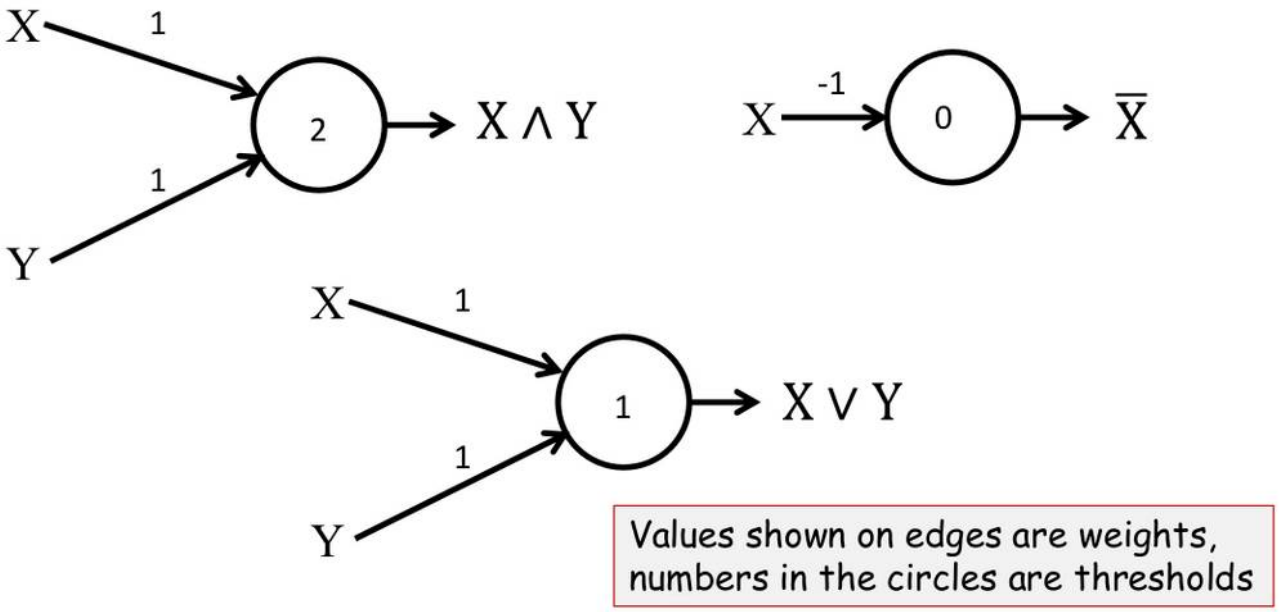
$d(x)$ is the desired output in response to input $\mathbf{x}$

$y(x)$ is the actual output in response to $\mathbf{x}$

- Boolean tasks
- Update the weights whenever the perceptron output is wrong
  - Update the weight by the product of the input and the *error* between the desired and actual outputs
- Proved convergence for linearly separable classes

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Perceptron: Modeling boolean gates

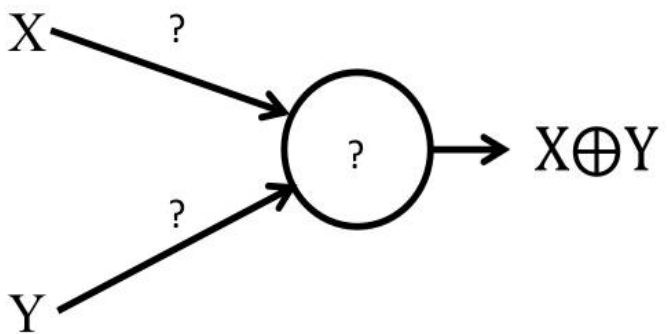


- Easily shown to mimic any Boolean gate
- But…

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Perceptron

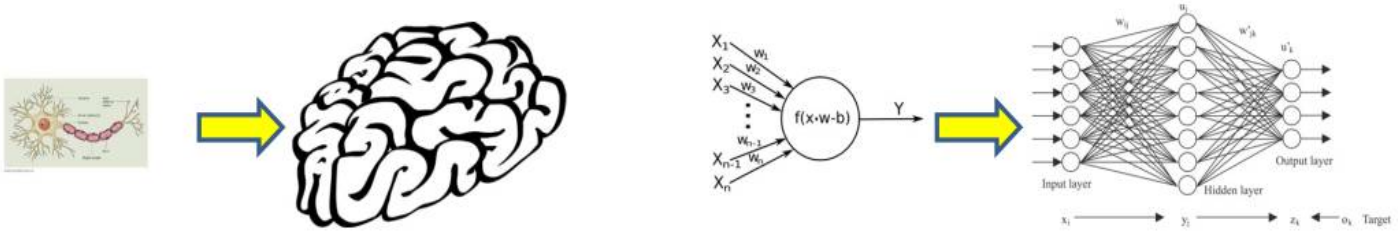
No solution for XOR!
Not universal!



- Minsky and Papert, 1968

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

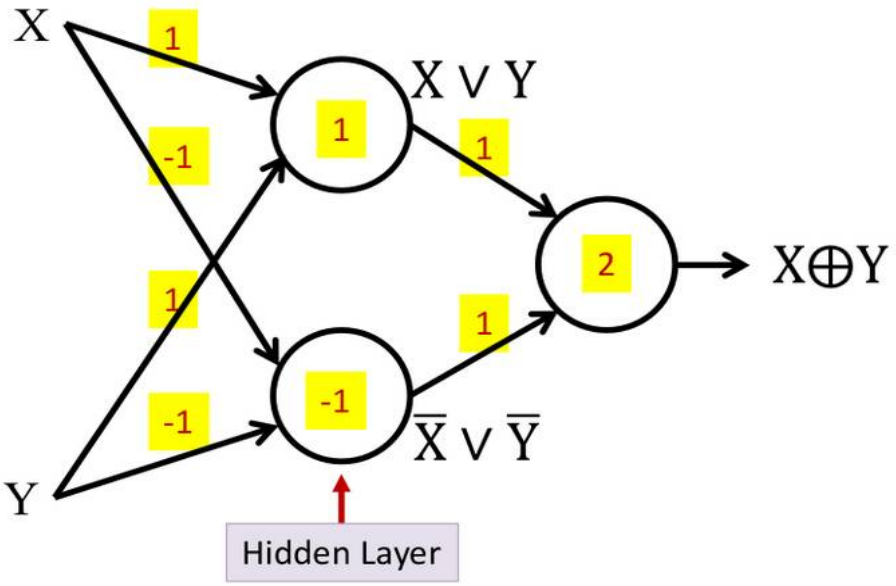
Single units are not enough



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

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Perceptron

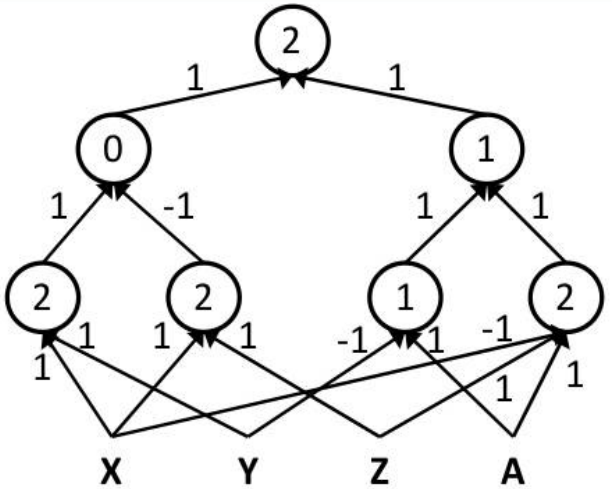


- XOR
  - The first layer is a "hidden" layer
  - Also originally suggested by Minsky and Papert 1968

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

A more generic model

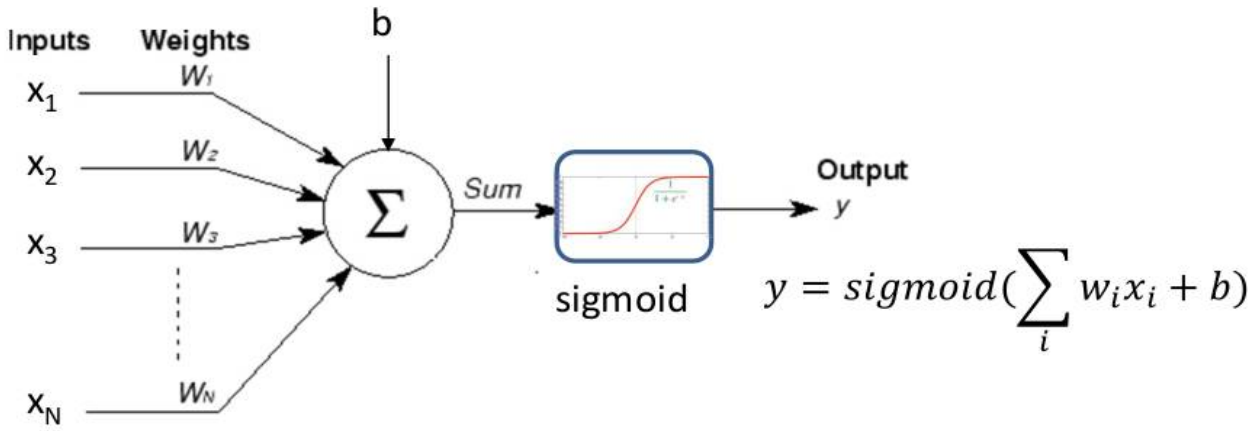
$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$



- A “multi-layer” perceptron
- Can compose arbitrarily complicated Boolean functions!
  - In cognitive terms: Can compute arbitrary Boolean functions over sensory input

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

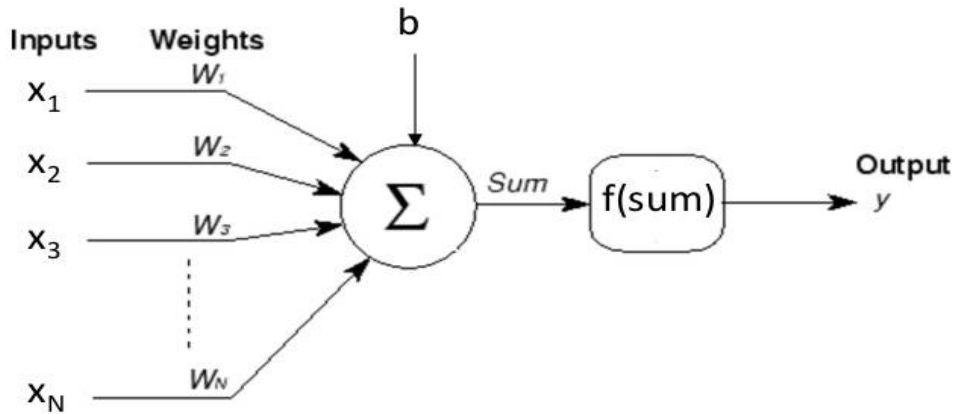
Perceptron with real inputs and a real output



- $x_1 \ldots x_N$ 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

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

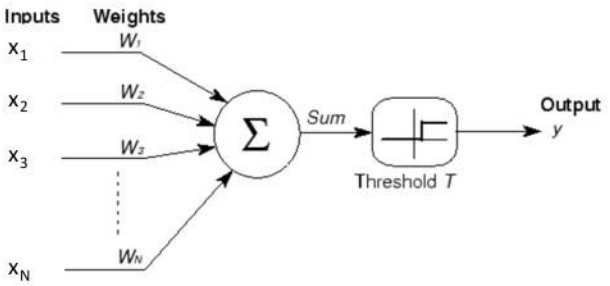
The real valued perceptron



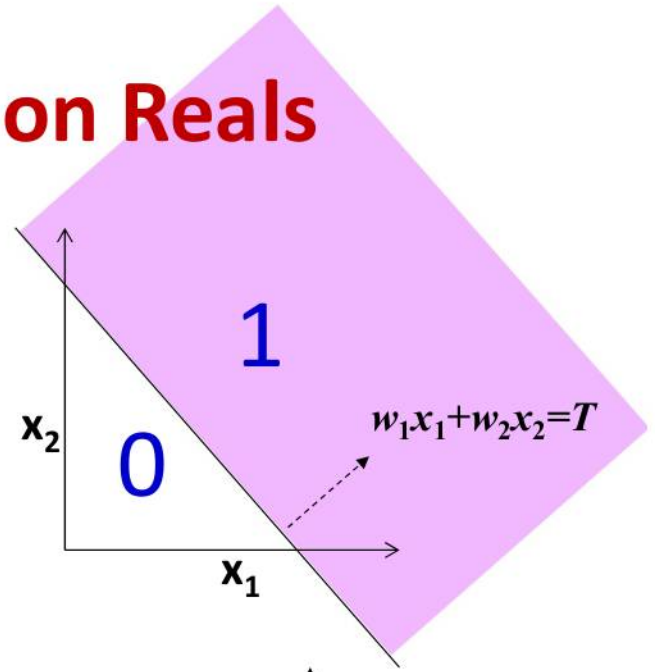
- Any real-valued "activation" function may operate on the affine function of the input
  - We will see several later
  - Output will be real valued
- The perceptron maps real-valued inputs to real-valued outputs
- *Is useful to continue assuming Boolean outputs though, for interpretation*

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

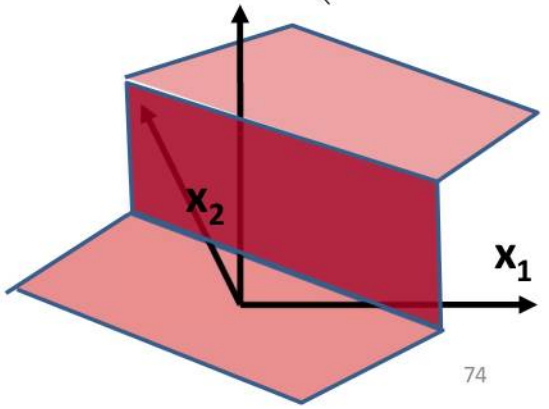
A Perceptron on Reals



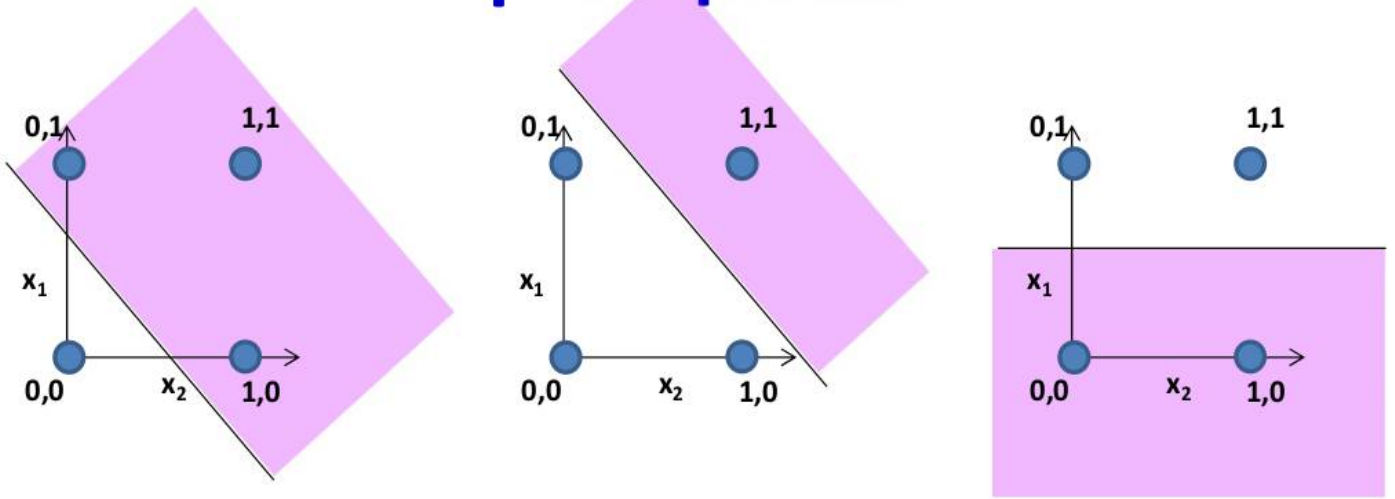
$$y = \begin{cases} 1 \; if \; \sum_i w_i x_i \geq T \\ 0 \; else \end{cases}$$



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

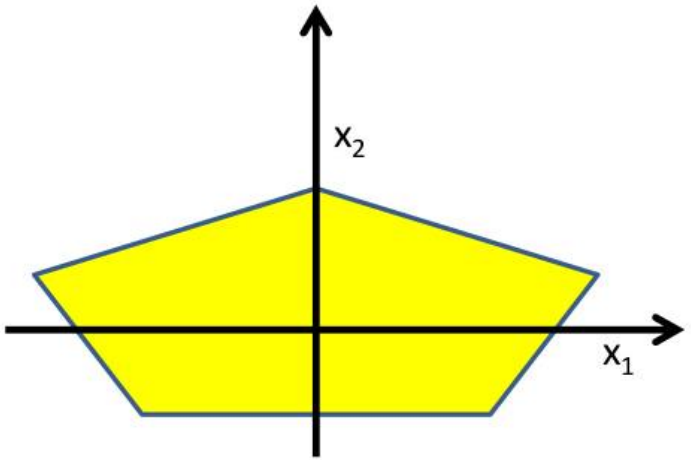


Boolean functions with a real perceptron



- 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

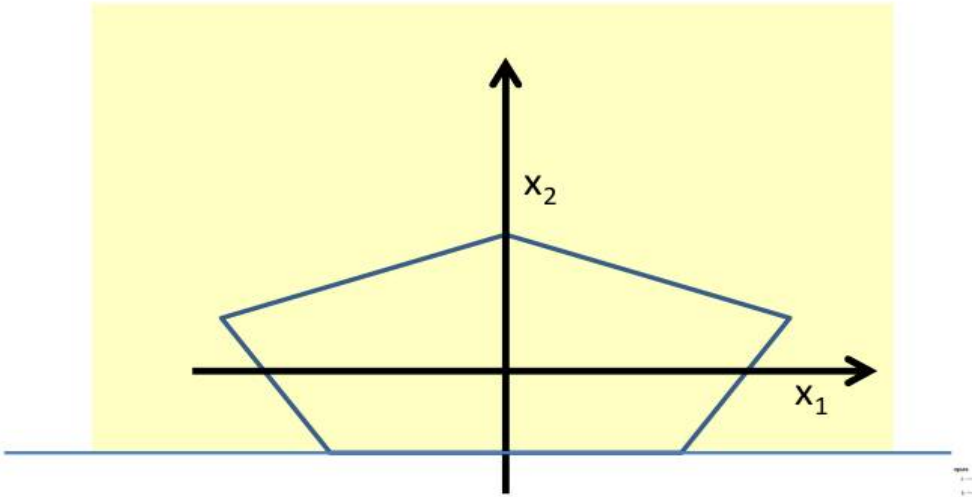


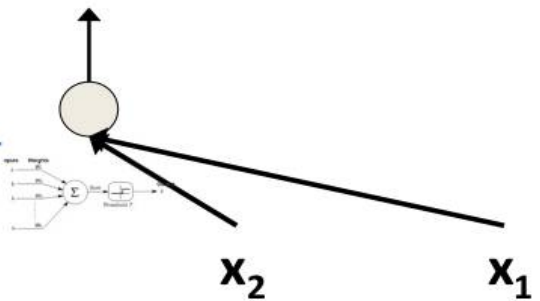
Can now be composed into “networks” to compute arbitrary classification “boundaries”

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

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Booleans over the reals

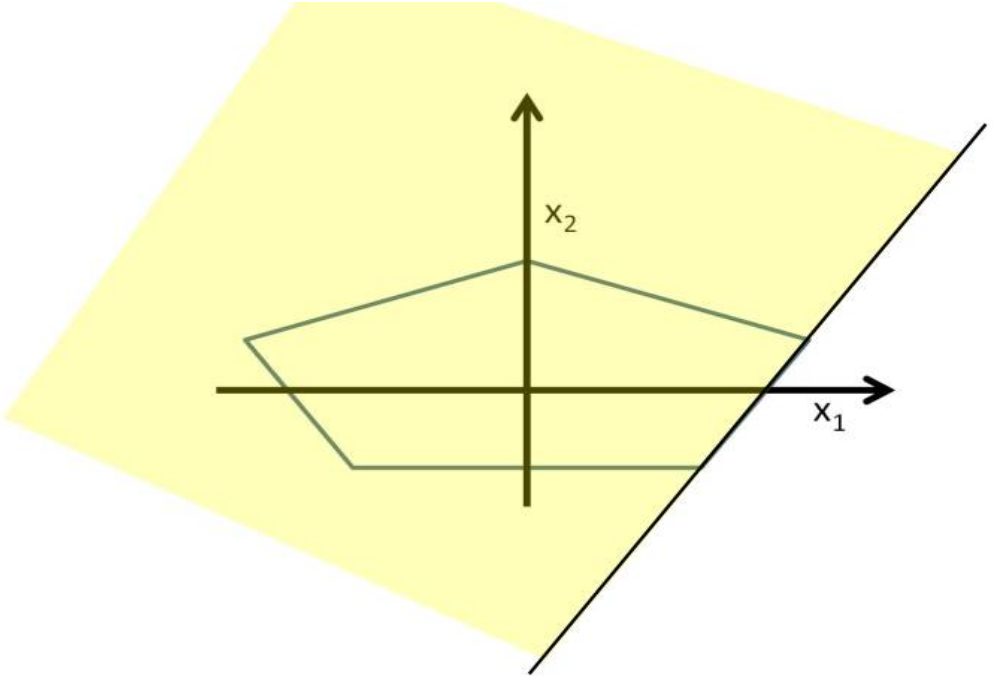


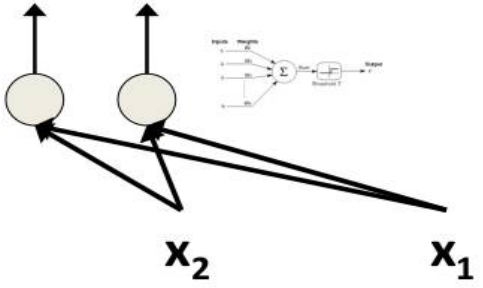


- The network must fire if the input is in the coloured area

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Booleans over the reals

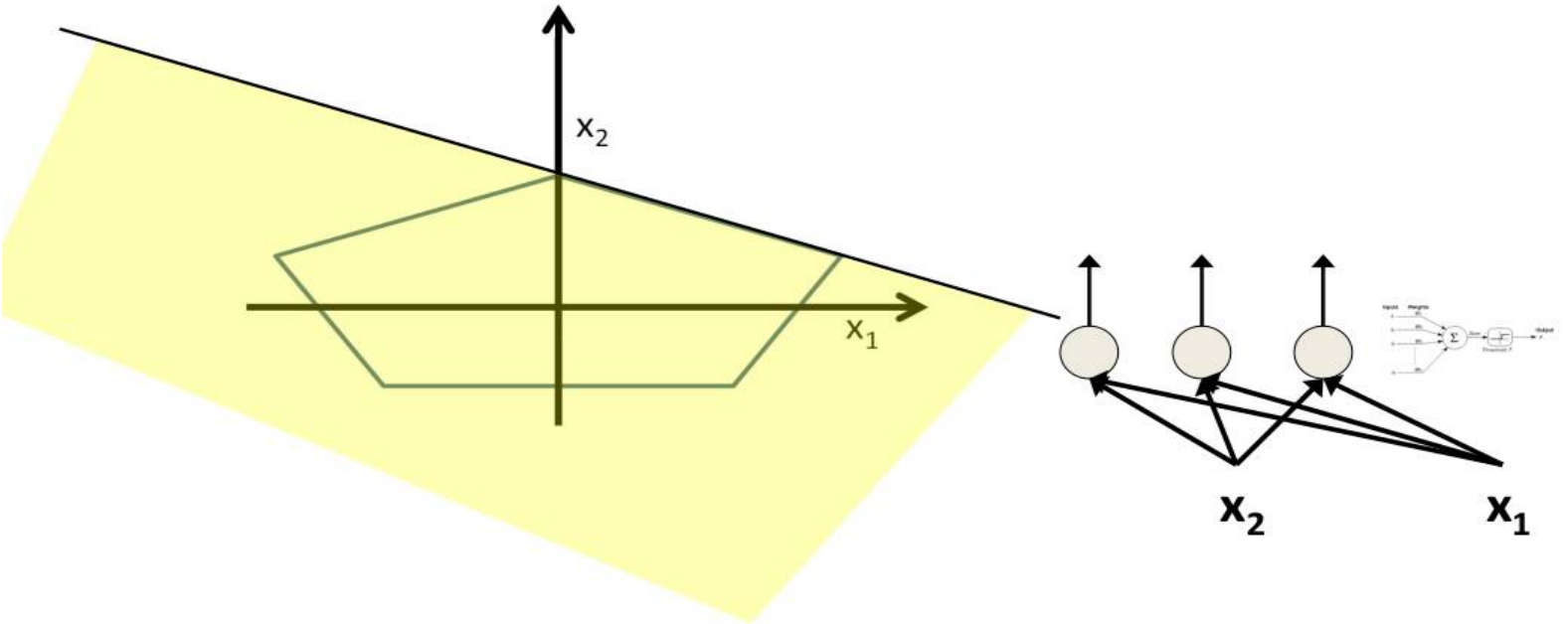




- The network must fire if the input is in the coloured area

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

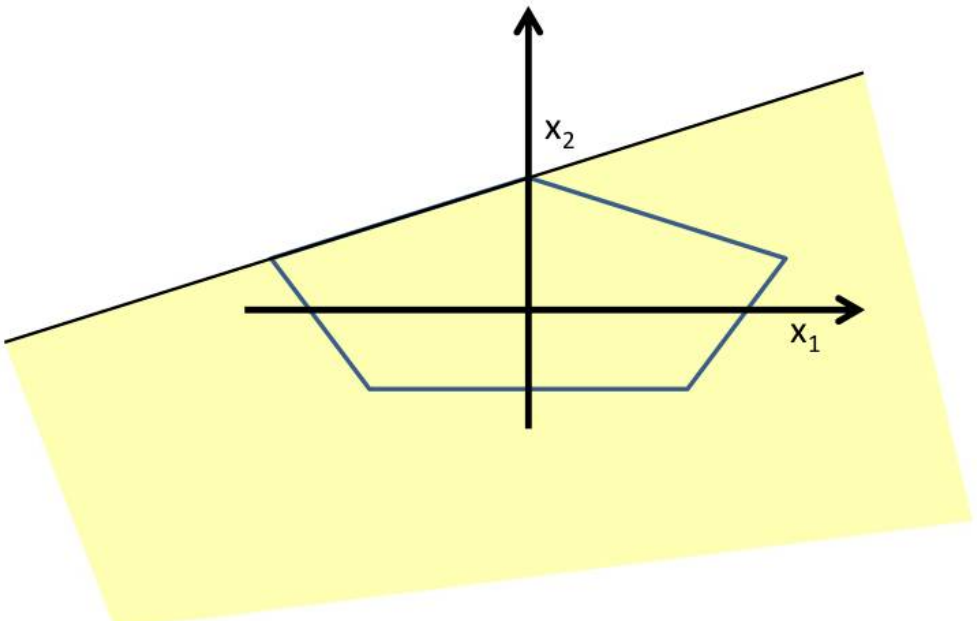
Booleans over the reals

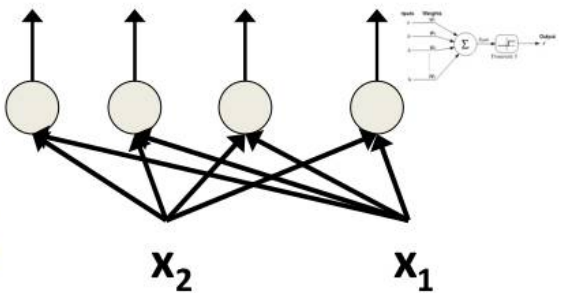


- The network must fire if the input is in the coloured area

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Booleans over the reals

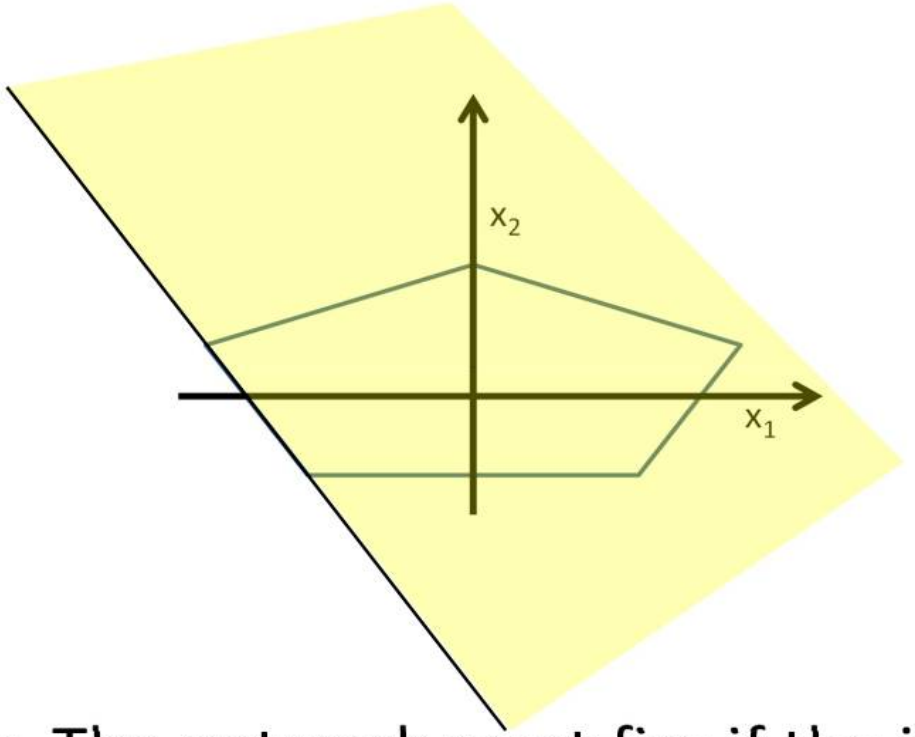


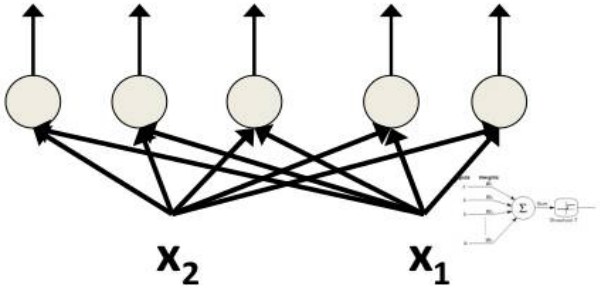


- The network must fire if the input is in the coloured area

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Booleans over the reals

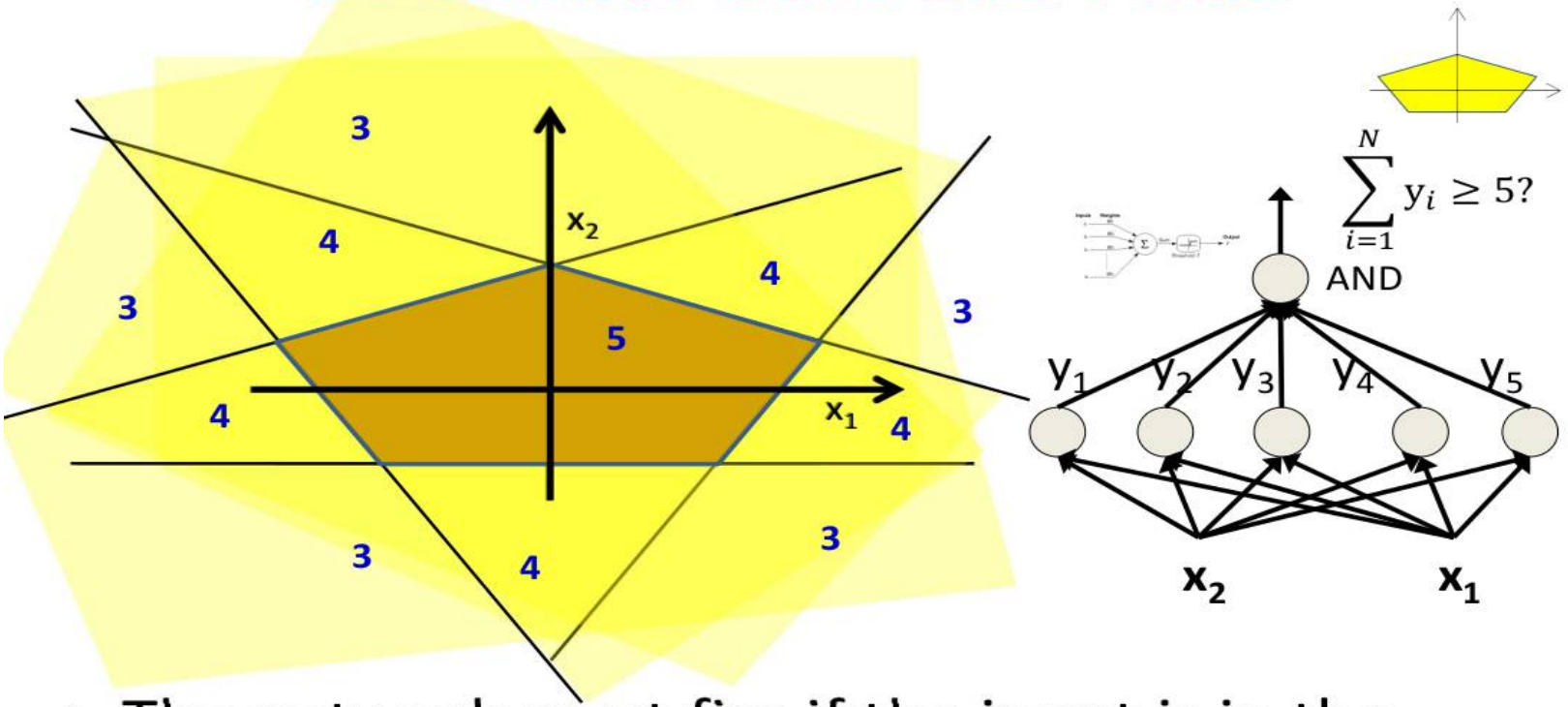




- The network must fire if the input is in the coloured area

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

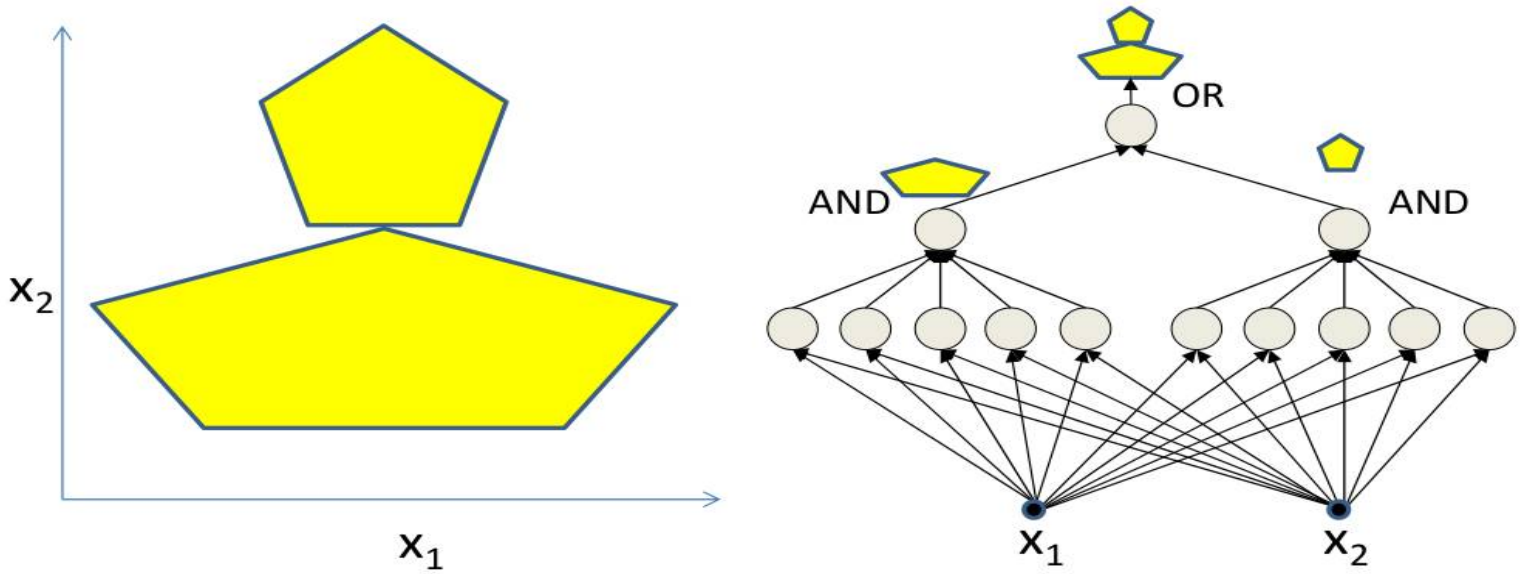
Booleans over the reals



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

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

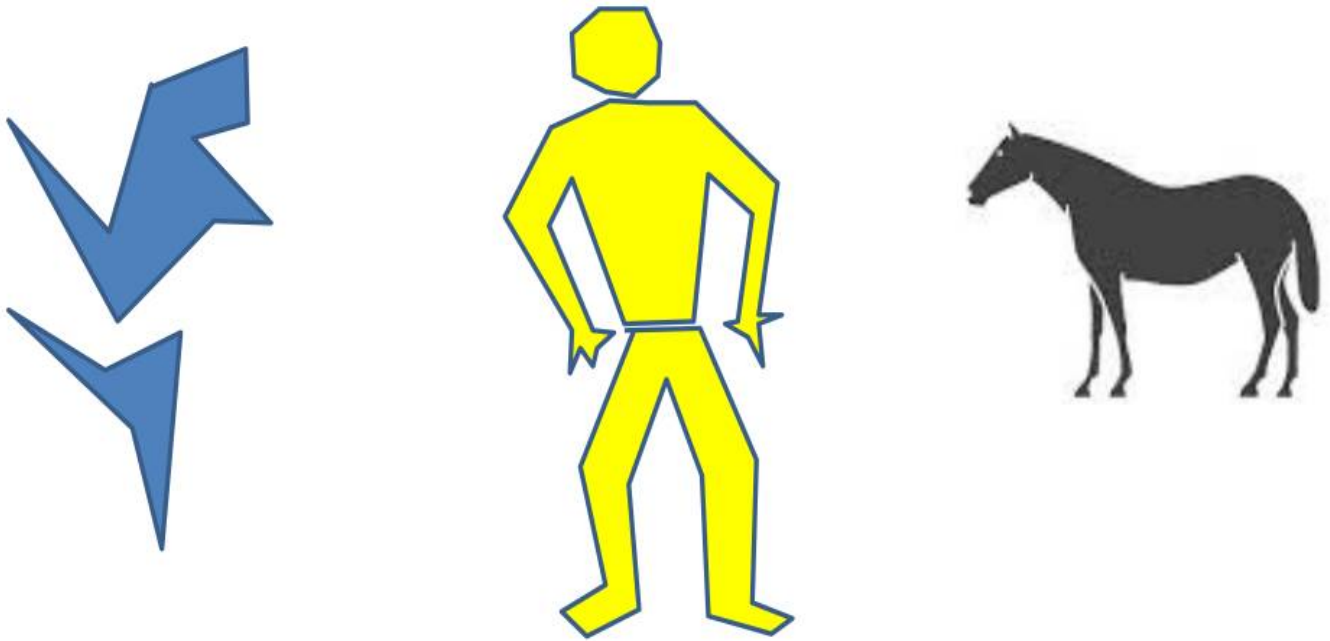
More complex decision boundaries



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

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Complex decision boundaries



- Can compose very complex decision boundaries

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Quiz

- How many threshold activation perceptrons will we need in an MLP to model a hexagonal decision region (a decision region bounded by a six-sided polygon) over a 2D input space?

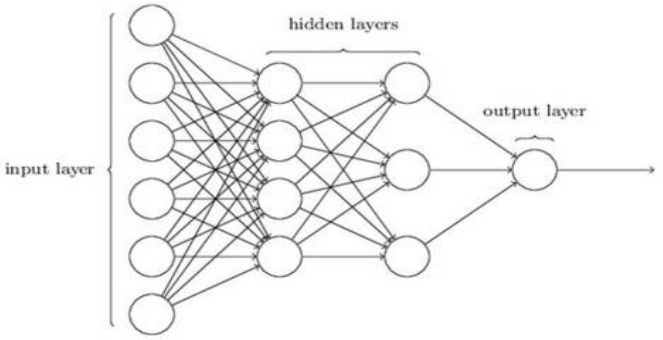
- Answer : 7

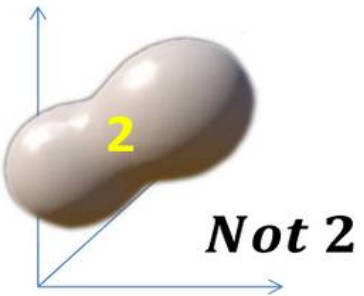
Perceptron Model

- Use is grounded in theory
  - Universal approximation theorem (Goodfellow 6.4.1)
- Can represent a NAND circuit, from which any binary function can be built by compositions of NANDs
- With enough parameters, it can approximate any function.

Complex decision boundaries

784 dimensions
(MNIST)



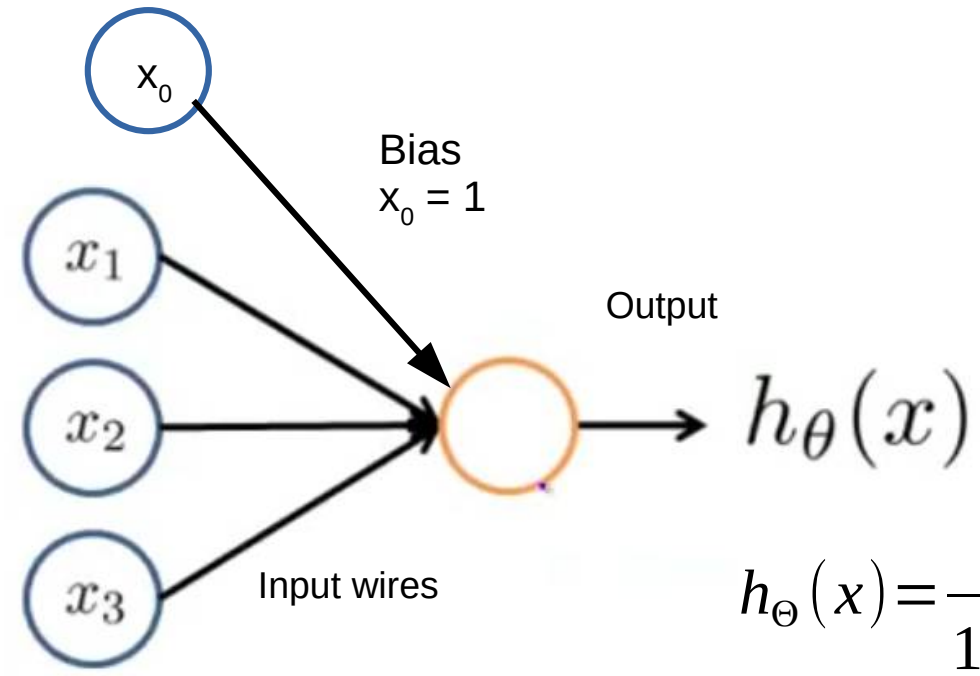


784 dimensions

- Classification problems: finding decision boundaries in high-dimensional space
  - Can be performed by an MLP
- MLPs can *classify* real-valued inputs

Source: Prof. Bhiksha Raj. https://deeplearning.cs.cmu.edu/F21/document/slides/lec1.intro.pdf

Neuron Model: Logistic Unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Weights or parameters.
May also be denoted by w.

$$h_\Theta(x) = \frac{1}{1+e^{-\Theta^T x}}$$

Sigmoid (logistic) activation function

Binary Classifying an image

- Each pixel of an image would be an input.
- So for a 28 x 28 image, we vectorize.

  Dimension of x = 784 x 1
- Θ is vector of weights, and b is the bias.

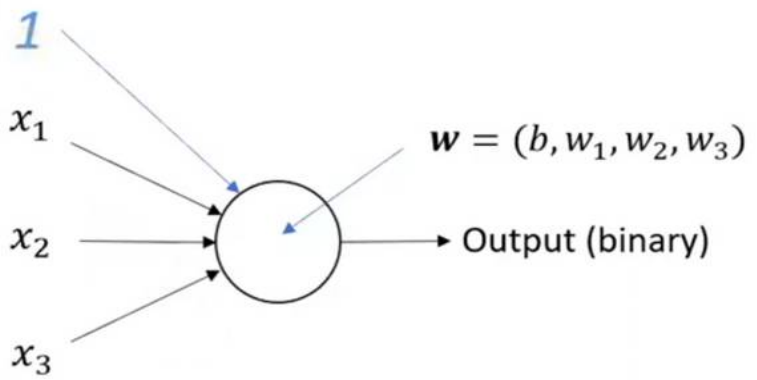
  Θ is 1 x 784 , b is a scalar (may also be denoted as Θ_0)

  Result = $h_\Theta(x) = \Theta x + b$ = Dims: (1 x 784) x (784 x 1) + b
- Total learnable weights / parameters = 784 + 1 = 785

Bias Convenience

Let's turn this operation into a multiplication only:

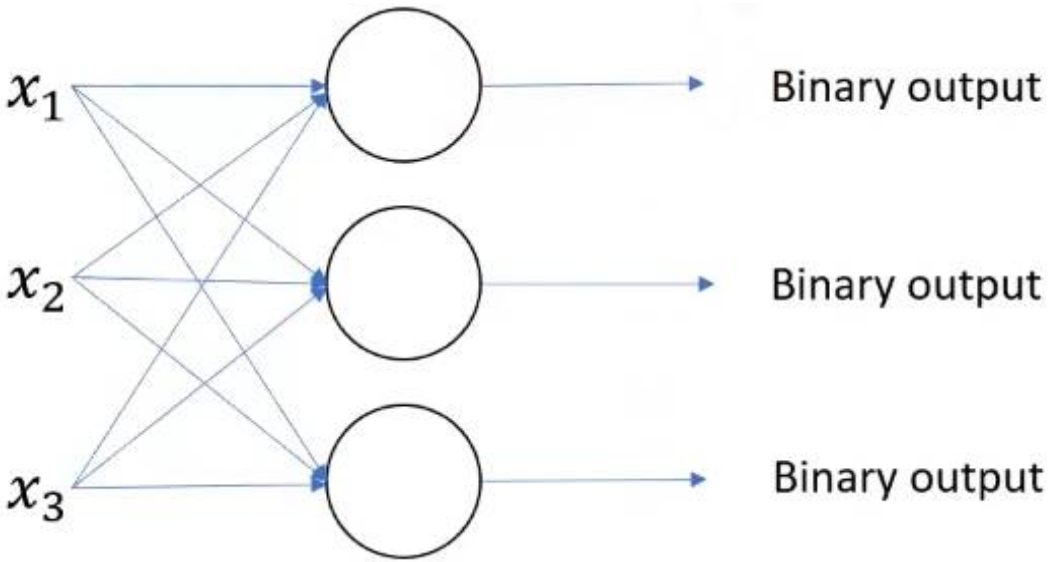
- Create a 'fake' feature with value 1 to represent the bias
- Add an extra weight that can vary



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x \leq 0 \\ 1 & \text{if } w \cdot x > 0 \end{cases} \qquad w \cdot x \equiv \sum_j w_j x_j$$

Neural Networks: Multi-class

Add more perceptrons



Multi-class Classifying an image

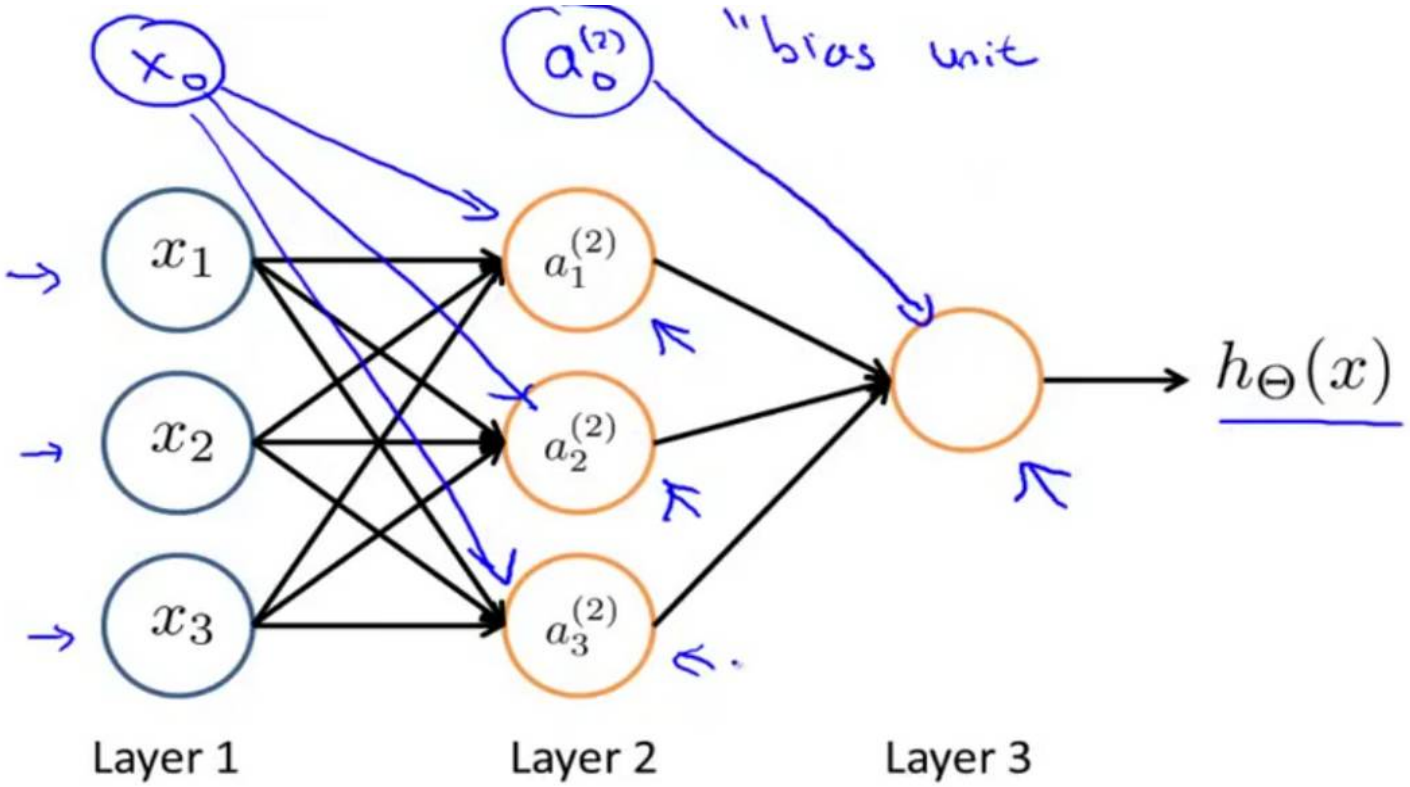
- Each pixel of an image would be an input.
- So for a 28 x 28 image, we vectorize.

  Dimension of x = 784 x 1
- Θ is matrix of weights, and b is the vector of biases.

  Θ is 10 x 784 , b is a vector (10 x 1)

  Result = $h_{\Theta}(x) = \Theta x + b$ = Dims: (10 x 784) x (784 x 1) + (10 x 1)
- Total learnable weights / parameters = 10 x 784 + 10 = 7850

Neural Network

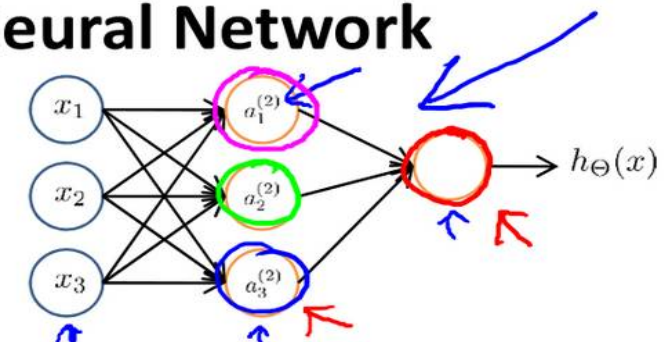


Input Layer

Hidden Layer

Output Layer

Neural Network



→ $a_i^{(j)}$ = "activation" of unit i in layer j

→ $\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$h_\Theta(x)$

→ $a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$

→ $a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$

→ $a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$

$\Theta^{(2)}$

→ $h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$

→ If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

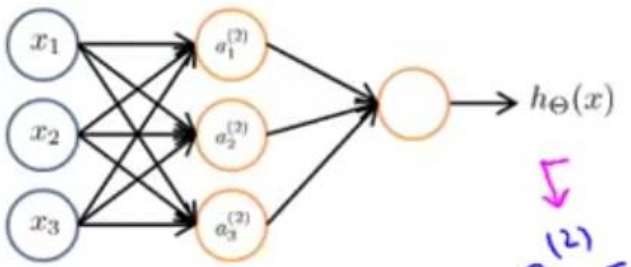
$s_{j+1} \times (s_j + 1)$

Quiz

- If a Neural Network has 1 hidden layer of size 100, input features given to the network for classification are of size 50, and they need to be classified into 1000 classes, then how many learnable parameters does the network contain.

- (50 x 100 + 100) + (100 x 1000 + 1000) = 106100

Forward Propagation: Vectorized Implementation



$a_1^{(2)} = g(z_1^{(2)})$

$z_1^{(2)}$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$z_2^{(2)}$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$z_3^{(2)}$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$a_3^{(2)} = g(z_3^{(2)})$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$z^{(2)} = \Theta^{(1)} x$$

$$a^{(2)} = g(z^{(2)})$$

$a^{(2)} \in \mathbb{R}^3$, $z^{(2)} \in \mathbb{R}^3$

$$a^{(2)} = g(z^{(2)})$$

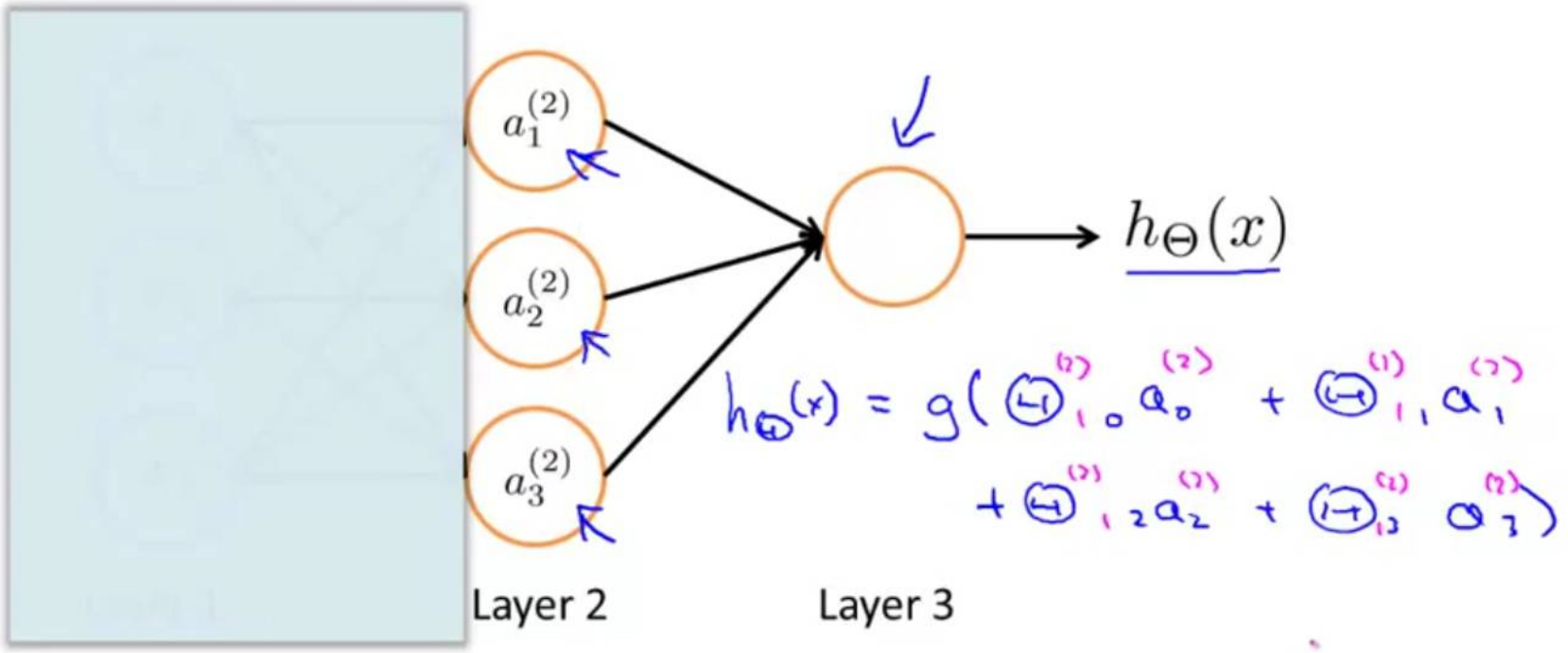
$a^{(2)} \in \mathbb{R}^3$, $z^{(2)} \in \mathbb{R}^3$

Add $a_0^{(2)} = 1$.

$$z^{(3)} = \Theta^{(2)} a^{(2)}$$

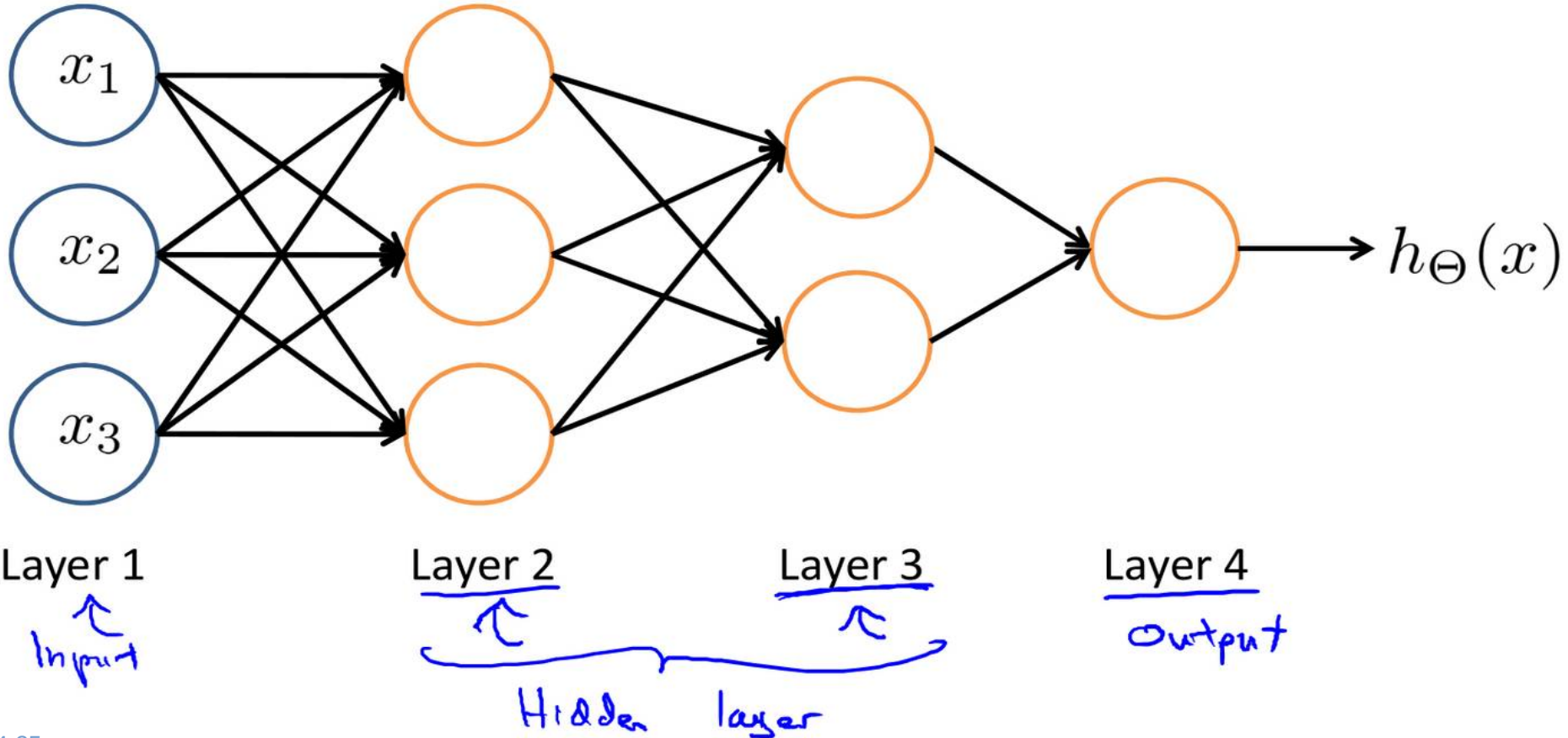
$$h_\Theta(x) = a^{(3)} = g(z^{(3)})$$

Neural Network learning its own features



It is just like logistic regression, but instead of using x_1, x_2, x_3 it is using the new features $a_1^{(2)}$, $a_2^{(2)}$, $a_3^{(2)}$.

Other network architectures



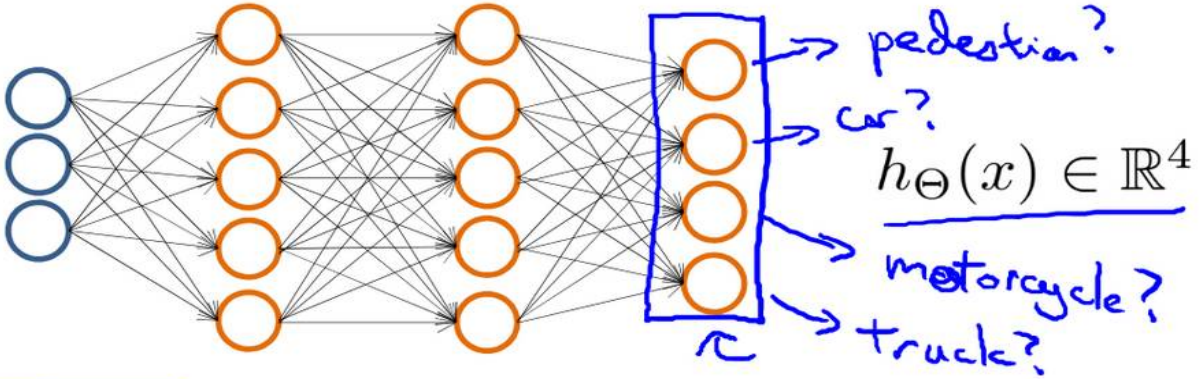
Multiple output units: One-vs-all.

Pedestrian

Car

Motorcycle

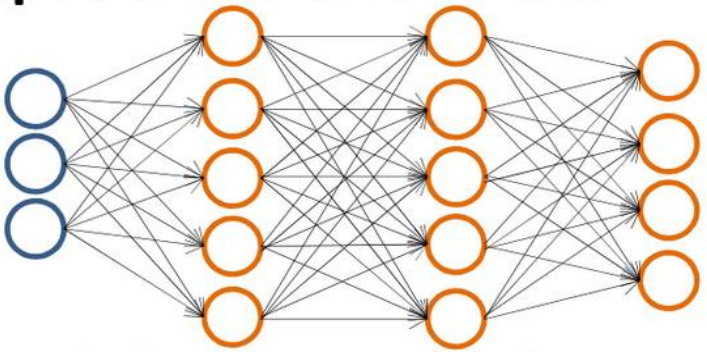
Truck



Want $h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Multiple output units: One-vs-all.



$$h_\Theta(x) \in \mathbb{R}^4$$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$\Rightarrow$ $y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

$(x^{(i)}, y^{(i)})$

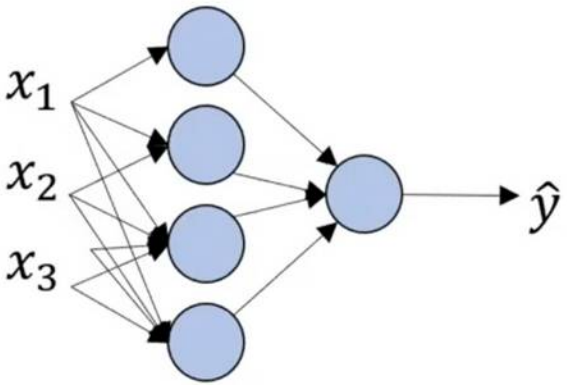
Previously $y \in \{1, 2, 3, 4\}$

$h_\Theta(x^{(i)}) \approx y^{(i)}$

$\in \mathbb{R}^4$

Andrew Ng

Why adding non-linearity (eg. sigmoid) is important?
Problem 1 with all linear functions



$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\,\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\,\mathbf{z}^{[1]} + \mathbf{b}^{[2]}$$

$$\begin{aligned}\mathbf{z}^{[2]} &= \mathbf{W}^{[2]}\,\mathbf{z}^{[1]} + \mathbf{b}^{[2]} \\ &= \mathbf{W}^{[2]}\,[\mathbf{W}^{[1]}\,\mathbf{x} + \mathbf{b}^{[1]}] + \mathbf{b}^{[2]} \\ &= \mathbf{W}^{[2]}\,\mathbf{W}^{[1]}\,\mathbf{x} + \mathbf{W}^{[2]}\,\mathbf{b}^{[1]} + \mathbf{b}^{[2]} \\ &= \mathbf{W}\,\mathbf{x} + \mathbf{b}\end{aligned}$$

$$\hat{y} = \mathbf{z}^{[2]} = \mathbf{W}\,\mathbf{x} + \mathbf{b}$$

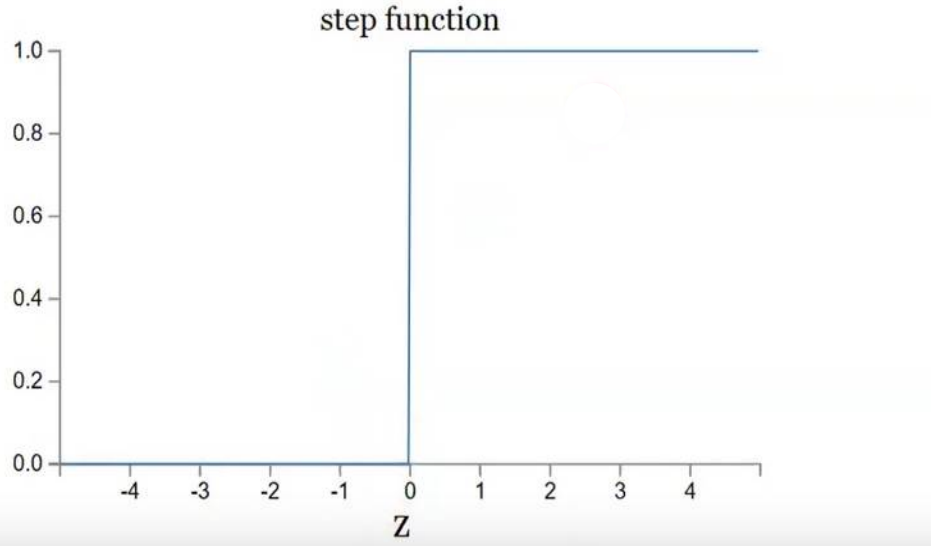
The output is always a linear function of the input!

Problem 2 with all linear functions

Linear classifiers:

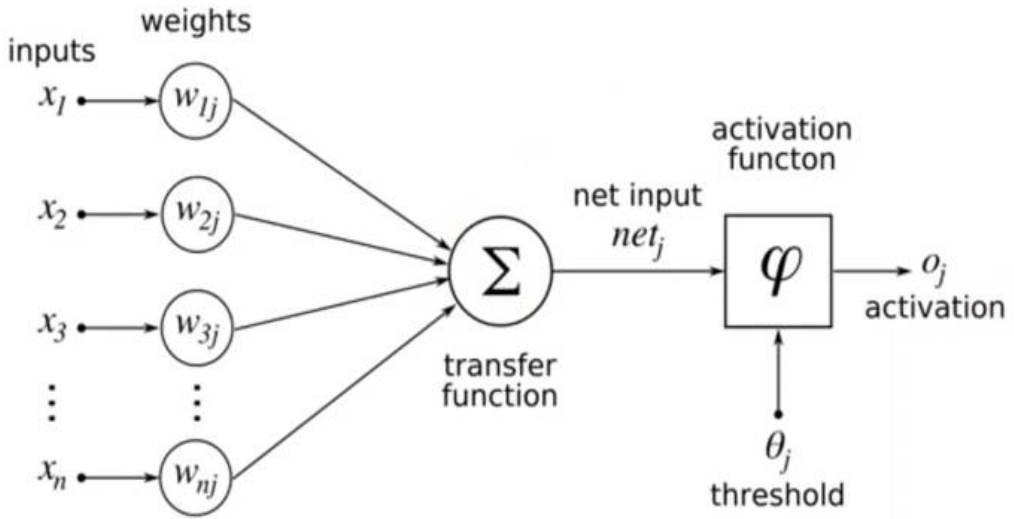
small change in input can cause large change in binary output

Activation function for a perceptron: Heaviside function



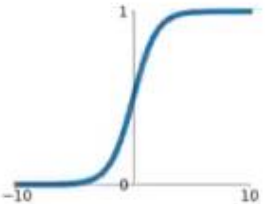
Activation Functions

Introducing Non-linearity



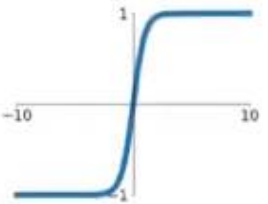
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



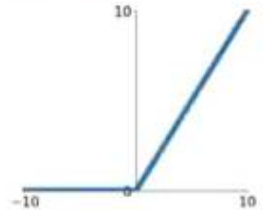
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



Vanishing Gradient Problem

- A disadvantage of using sigmoid activation is the vanishing gradient problem.
- The gradients (i.e., g’(z)) for large absolute values of z is close to zero. Therefore, when more layers are added with sigmoid activations, the earlier layer parameters may not get updated due to the gradients becoming zero till the time they reach these layers during backprop.
- **Vanishing gradients** refer to a situation where the gradients of the loss function with respect to the model’s parameters become extremely small as they are backpropagated through the layers of a deep neural network. When gradients vanish, it means that the network’s weights and biases hardly get updated during training, leading to slow convergence or even complete stagnation in learning. This phenomenon is particularly problematic in deep networks with many layers.

MLP Performance on MNIST

- MNIST
  - Dataset of handwritten digits
  - 60K training samples
  - 10K testing samples
- A 6-layer MLP [1]
- Number of neurons
  - 2500, 2000, 1500, 1000, 500, and 10
- Performance - 99.65%

[1] Cireşan, Dan Claudiu, et al. "Deep, big, simple neural nets for handwritten digit recognition." *Neural computation* 22.12 (2010): 3207-3220.

End of Lecture