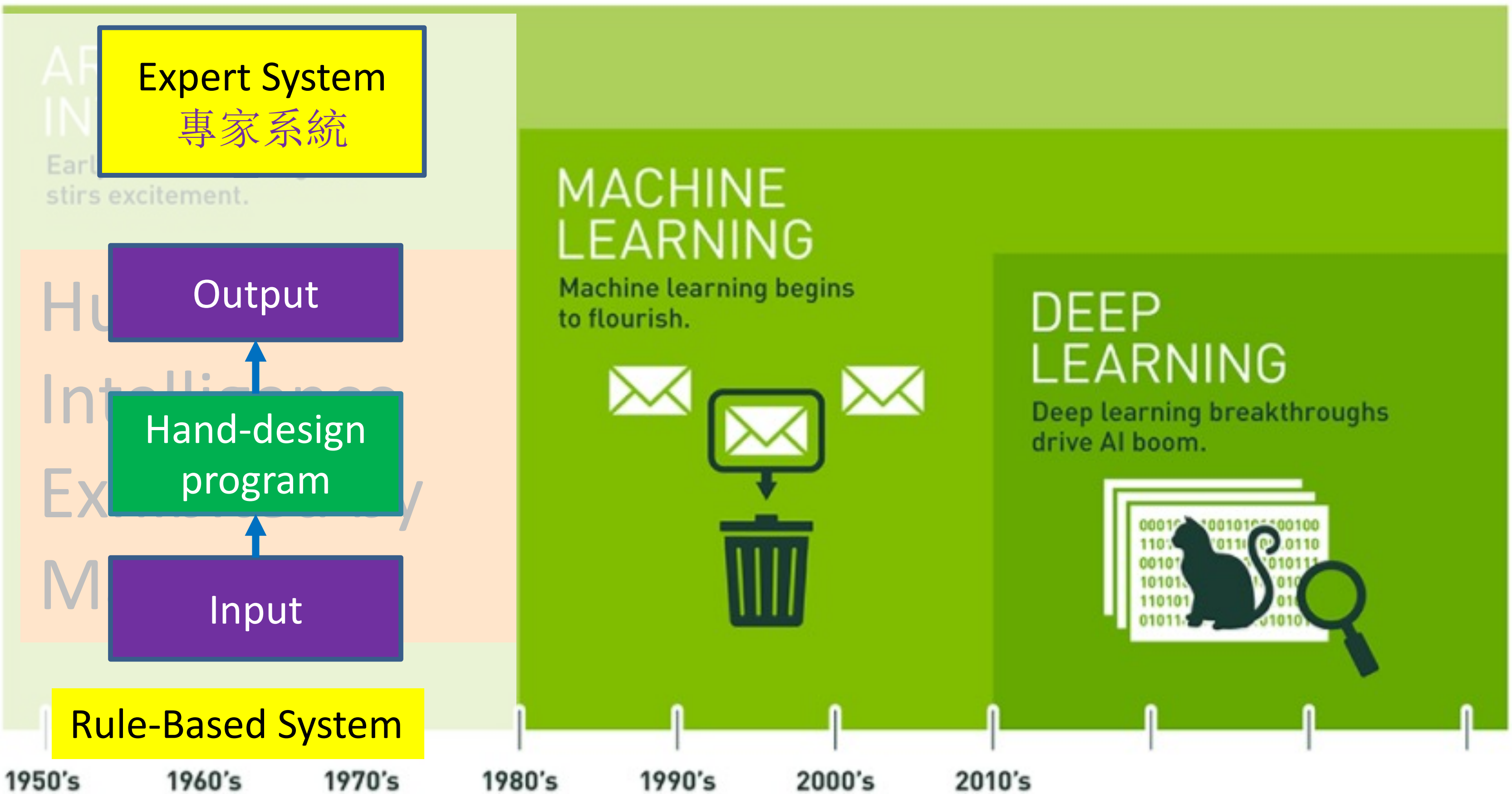
DASC7606 – 2A
Deep Learning

Artificial Neural Networks

Dr Bethany Chan
Professor Francis Chin
2024

Revision

Artificial Intelligence → **Machine Learning.** → **Deep Learning**



Require an expert (professional)
to define the input (features) and rules

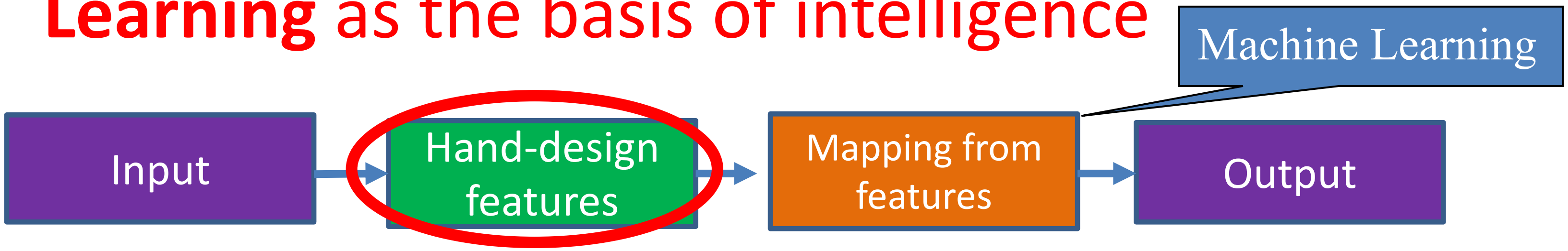
Source: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Revision

Limited success with rule-based expert system:

e.g., chess, go game, chatbot, image search, spam, .

- **Learning** as the basis of intelligence



- **Start from a child's brain** (simple algorithm) that learns instead of an adult's brain (complicated algorithm)
  - i.e., **iterative improvement (gradient descent)**

Simple algorithm:

guess, [check error, guess again intelligently]*

Revision

Linear models ("line" specified by coefficients $\boldsymbol{\theta}$ to approximate a hidden function *f*) to solve classification and regression problems

- $h(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_N x_N$
- $h(\boldsymbol{x}_i)= \boldsymbol{x}_i\boldsymbol{\theta} = \theta_0 + \theta_1 x_{i,1} \ldots + \theta_N x_{i,N}$ (true answer y_i)
- Parameters $\boldsymbol{\theta}$ are to be learned (modifying $\boldsymbol{\theta}$)
- Modifying $\boldsymbol{\theta}$ using gradient descent:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$$

Depends on

- cost function $J(\boldsymbol{\theta})$
- Slope (magnitude and opposite direction (descent))
- Learning rate

Revision

Linear Models where **gradient descent** is used on the **loss/cost function** to adjust the line (learn θ):

Linear Regression $J(\theta) = \frac{1}{2M}\sum_{i=1}^{M}(h(\boldsymbol{x}_i) - y_i)^2$ (MSE)

M is the number of data points

$\boldsymbol{x}_i = i$th data point, an (N+1)-dim vector
(e.g., no of bedrooms, size, age,...)

$h(\boldsymbol{x}_i) = \boldsymbol{x}_i\boldsymbol{\theta} = \theta_0 + \theta_1 x_{i,1} \ldots + \theta_N x_{i,N}$ = prediction for y_i

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{M}\sum_{i=1}^{M}(h(\boldsymbol{x}_i) - y_i)\, x_{i,j} \quad \text{for } j = 0,\ldots,N$$

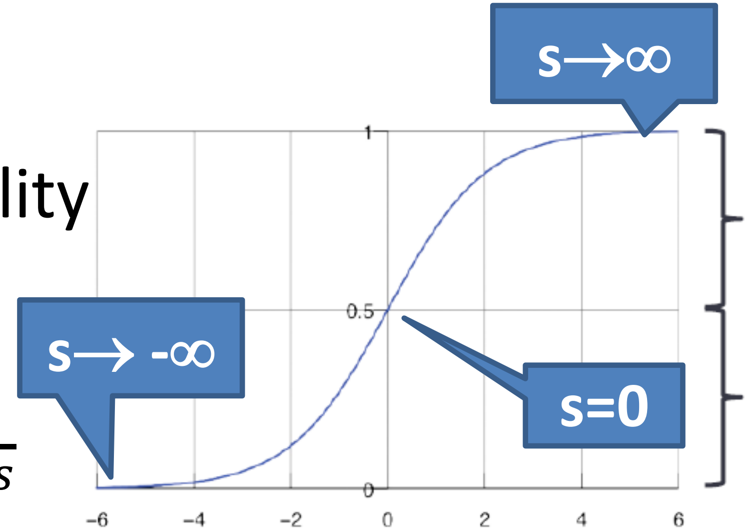
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{M}\mathrm{X}^{\mathrm{T}}(\mathrm{X}\boldsymbol{\theta} - \boldsymbol{y})$ in vectorized form,

where X = Mx(N+1) matrix; $\boldsymbol{\theta}$= (N+1)x1 vector matrix

Revision

Classification: Predict the probability of an event occurring

Logistic Regression uses Sigmoid function $\sigma(s) = \frac{1}{1+e^{-s}}$



where $h(\boldsymbol{x}_i) = \sigma(s_i) \in [0,1]$

and $s_i = \boldsymbol{x}_i\boldsymbol{\theta} = \theta_0 x_{i,0} + \ldots\ \theta_j x_{i,j} + \ldots\ \theta_N x_{i,N}$ (linear equation)

$J(\boldsymbol{\theta}) = -\frac{1}{M}\sum_{i=1}^{M}[y_i \log h(\boldsymbol{x}_i) + (1-y_i)\log(1-h(\boldsymbol{x}_i))]$

(cross entropy cost function)

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_j} = -\frac{1}{M}\sum_{i=1}^{M}\big(y_i - h(\boldsymbol{x}_i)\big)x_{i,j}$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{M}\sum_{i=1}^{M}(h(\boldsymbol{x}_i) - y_i)\, x_{i,j} \quad \text{for } j = 0,\ldots,N$$

Same equation as Linear Regression

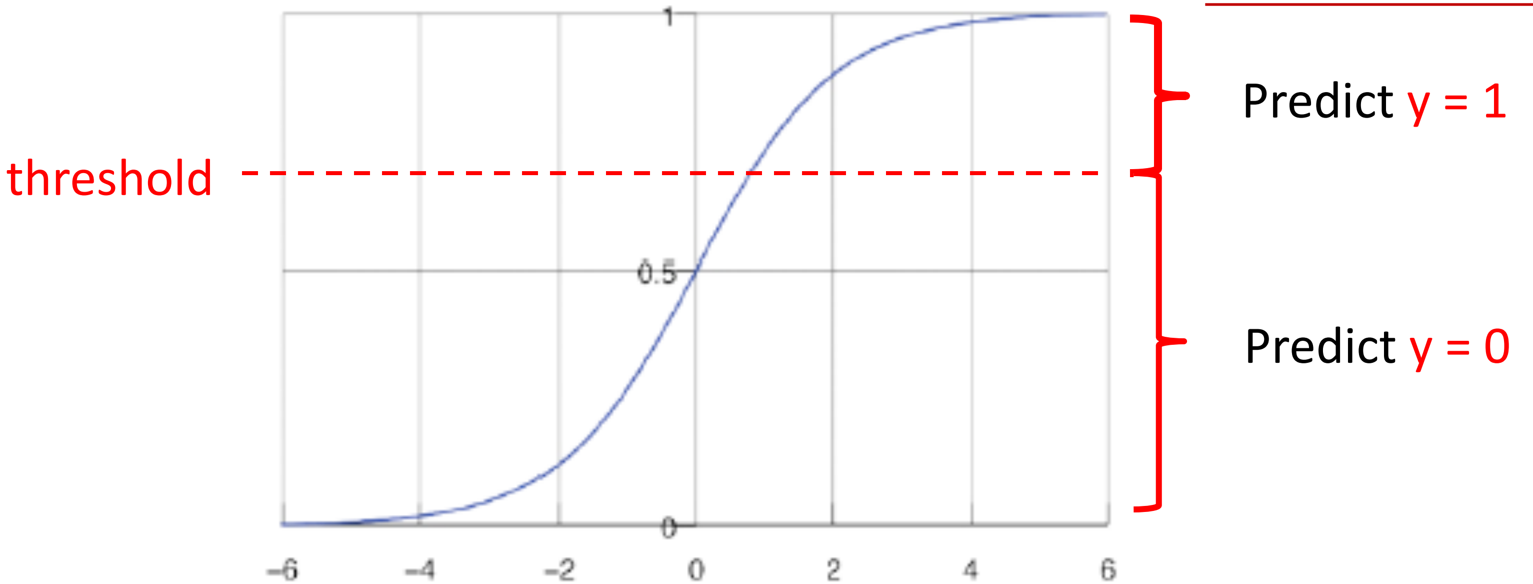
Outline

- The story behind Deep Learning
- AI and Rule-based systems
- Supervised learning, classification & regression problems, linear models
- Linear regression and gradient descent
- Logistic regression and classification
- **Performance measures of prediction**
- Multi-classification

Logistic Function and classification

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

For classification



Outcomes of Binary Classification

predict 1 if $f(x) > \text{threshold}$
predict 0 otherwise

	Actual 0	Actual 1
Predicted 0	true negative	false negative

- **True positives:**
  data points predicted as positive that are actually positive
- **False positives:**
  data points predicted as positive that are actually negative
- **True negatives:**
  data points predicted as negative that are actually negative
- **False negatives:**
  data points predicted as negative that are actually positive

Performance Measures of Predictions

predict 1 if $f(x) > \text{threshold}$
predict 0 otherwise

	Actual 0	Actual 1
Predicted 0	**true negative**	**false negative**
Predicted 1	**false positive**	**true positive**

precision means how "accurate" is the answer. i.e., measure the positive patterns that are correctly predicted from the total predicted patterns.

$$\text{precision} = \frac{\text{true positives}}{\text{predicted positives}} = \frac{\text{true positives}}{\text{false positives} + \text{true positives}}$$

Performance Measures of Predictions

predict 1 if $f(x) >$ threshold
predict 0 otherwise

	Actual 0	Actual 1
Predicted 0	true negative	false negative
Predicted 1	false positive	true positive

recall ("**sensitivity**" or "**true positive rate**") means how "*good*" is the answer, i.e., ability to find correct ones.

$$\text{recall} = \frac{\text{true positives}}{\text{actual positives}} = \frac{\text{true positives}}{\text{false negatives} + \text{true positives}}$$

or probability of positive result, given it is truly positive.
or fraction of positive patterns that are correctly classified.

For negative class, it is known as **specificity**
ability to correctly reject actual positive without a condition.

$$\text{specificity} = \frac{\text{true negatives}}{\text{actual negatives}} = \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}}$$

Accuracy of Predictions

predict 1 if $f(x) >$ threshold
predict 0 otherwise

	Actual 0	Actual 1
Predicted 0	**true negative**	**false negative**
Predicted 1	**false positive**	**true positive**

$$\textbf{precision} = \frac{\text{true positives}}{\text{predicted positives}} = \frac{\text{true positives}}{\text{false positives} + \text{true positives}}$$

Higher threshold => more difficult to be positive => less false positive

↑ threshold	**↑ precision**
↓ threshold	**↓ precision**

Accuracy of Predictions

predict 1 if $f(x) >$ threshold
predict 0 otherwise

	Actual 0	Actual 1
Predicted 0	**true negative**	**false negative**
Predicted 1	**false positive**	**true positive**

$$\textbf{recall} = \frac{\text{true positives}}{\text{actual positives}} = \frac{\text{true positives}}{\text{false negatives} + \text{true positives}}$$

Higher threshold => more difficult to be positive => more false negative

↑ threshold	**↑ precision**	**↓ recall**
↓ threshold	**↓ precision**	**↑ recall**

Combining Precision and Recall

- Ideally both precision and recall are 1
- With different thresholds, we can have higher precision and recall values
- Depending on applications
  - Disease screening - higher recall
  - Prosecution of criminals – higher precision
  - Identify terrorists - both

$$\mathbf{F1} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

- Harmonic mean instead of simple average to penalize extreme values

Receiver Operating Characteristic (ROC)

$$\textbf{True Positive Rate (TPR)} = \frac{\text{true positives}}{\text{false negatives} + \text{true positives}}$$

recall

Actual positive

$$\textbf{False Positive Rate (FPR)} = \frac{\text{false positives}}{\text{false positives} + \text{true negatives}}$$

False alarm

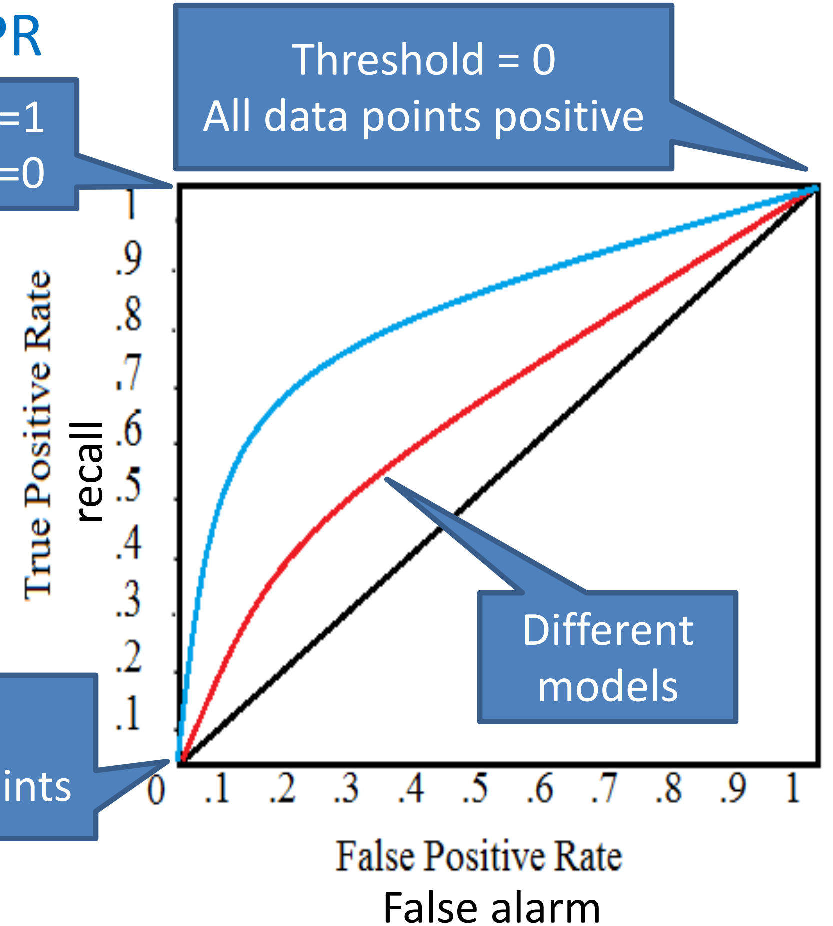
Actual negative

ROC curve plots TPR (recall) against FPR (probability of false alarm).

Threshold=1 $\Rightarrow$ miss all TP cases

Threshold=0 $\Rightarrow$ many false alarms

- Decrease threshold
  = move right and upward
- Area Under Curve (AUC), ranged between 0 and 1, higher the better
- Best performance is TPR=1 and FPR=0, AUC=1

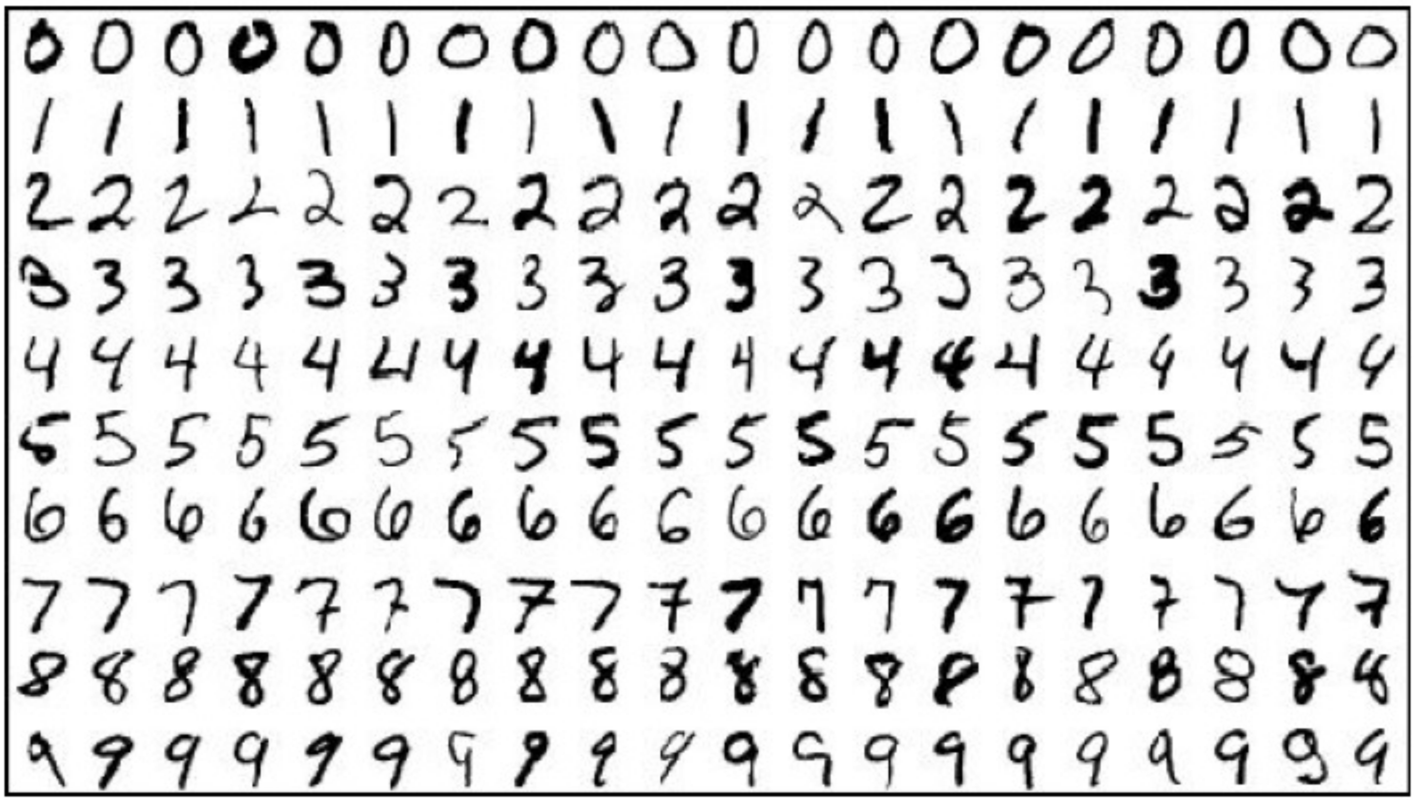


Outline

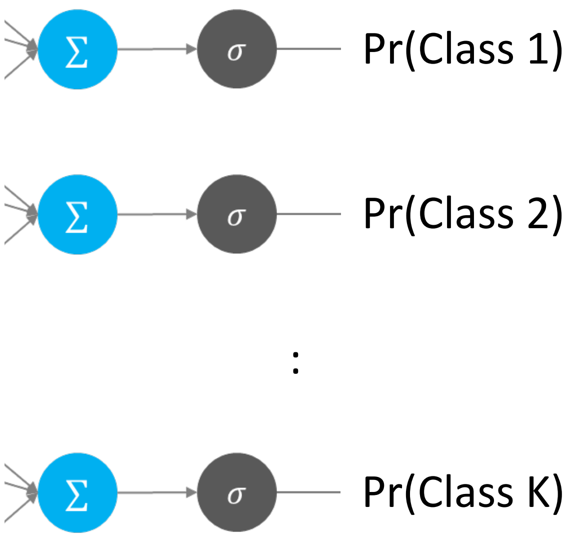
- The story behind Deep Learning
- AI and Rule-based systems
- Supervised learning, classification & regression problems, linear models
- Linear regression and gradient descent
- Logistic regression and classification
- Performance measures of prediction
- **Multi-classification**

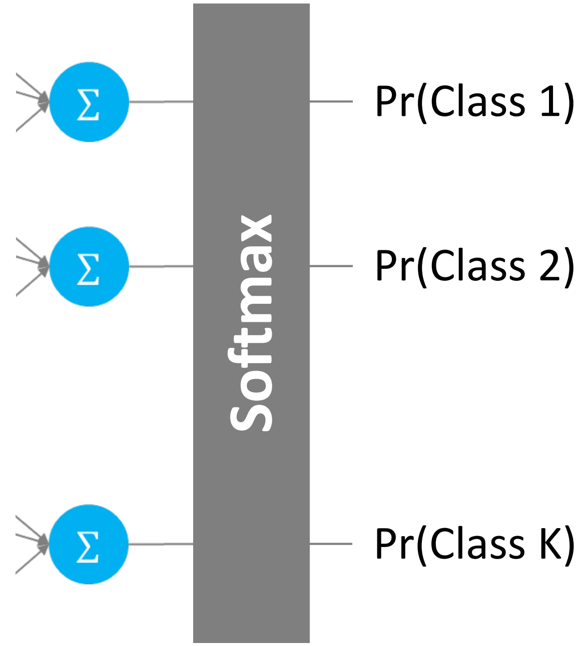
Multi-classification example

MNIST (Mixed National Institute of Standards and Technology) database



Multi-Classification





Pr(Class x) = Probability of Class x being the correct class

Class with **highest** probability is the predicted

Softmax function: not only normalizes a set of scores to numbers in [0,1] but also makes sure that the numbers all add up to 1

Sigmoid vs. Softmax

For **sigmoid** (2-class):

Class 1 probability: $\sigma(s) = \frac{1}{1+e^{-s}}$

Class 2 probability: $1 - \sigma(s) = \frac{e^{-s}}{1+e^{-s}} = \frac{1}{1+e^{s}}$

For **softmax** (K-class):

Given scores $s_1, s_2, \ldots, s_K$

Class k probability: $\frac{e^{s_k}}{\sum e^{s_i}}$

For **softmax** (2-class):

Class 1 probability: $\frac{e^{s_1}}{e^{s_1}+e^{s_2}} = \frac{1}{1+e^{s_2-s_1}}$

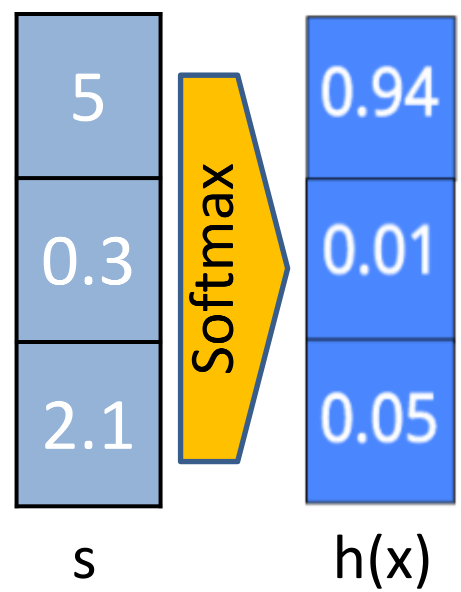
Class 2 probability: $\frac{e^{s_2}}{e^{s_1}+e^{s_2}} = \frac{1}{1+e^{s_1-s_2}}$

- Question: Why do we have to pass each value through an exponential before normalizing them? Why can’t we just normalize the values themselves?
- Answer: This is because the goal of softmax is to make sure **one value is very high** (close to 1) and all **other values are very low** (close to 0).
- Softmax uses exponential to make sure this happens.

For **softmax** (K-class):

Given scores $s_1, s_2, \ldots, s_K$

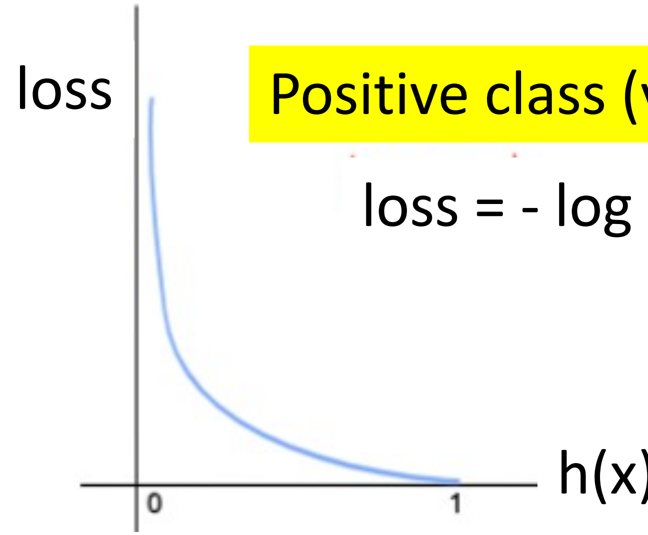
Class k probability: $\frac{e^{s_k}}{\sum e^{s_i}}$



Loss function for multi-classification

Categorical Cross Entropy

$$\mathrm{Loss}(h(x), y) = -\sum_{C} y \log\big(h(x)\big)$$



Positive class (y = 1)

loss = - log (h(x))

Ex:

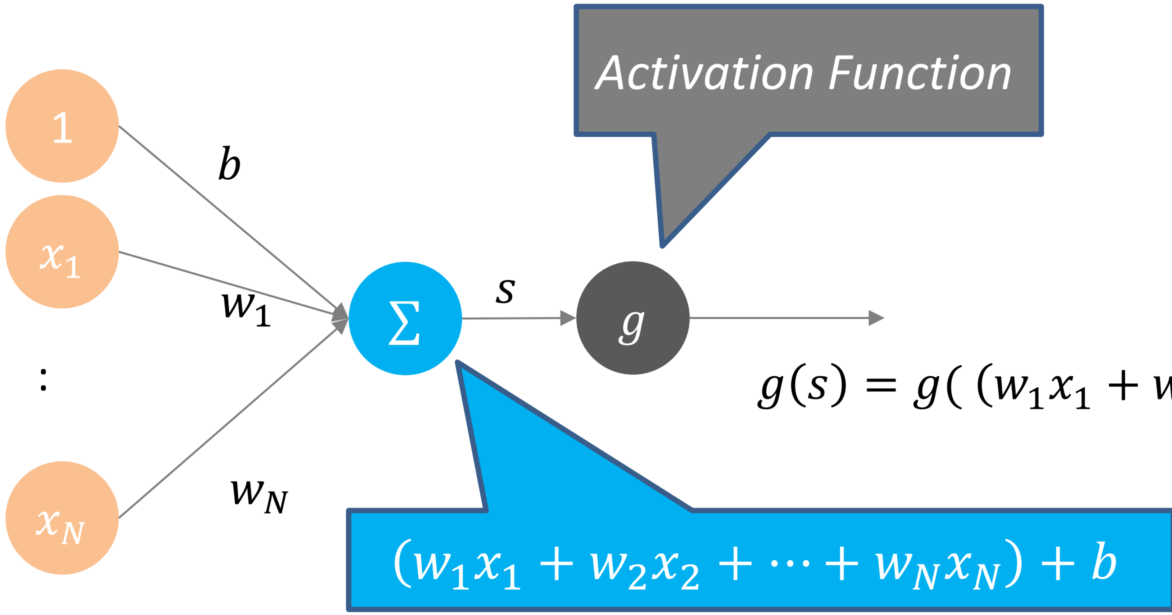
Predicted h(x)	Label y
[0.94, 0.01, 0.05]	[0, 1, 0]

Loss = $-[0 * \log(0.94) + 1 * \log(0.01) + 0 * \log(0.05)] = -\log(0.01)$

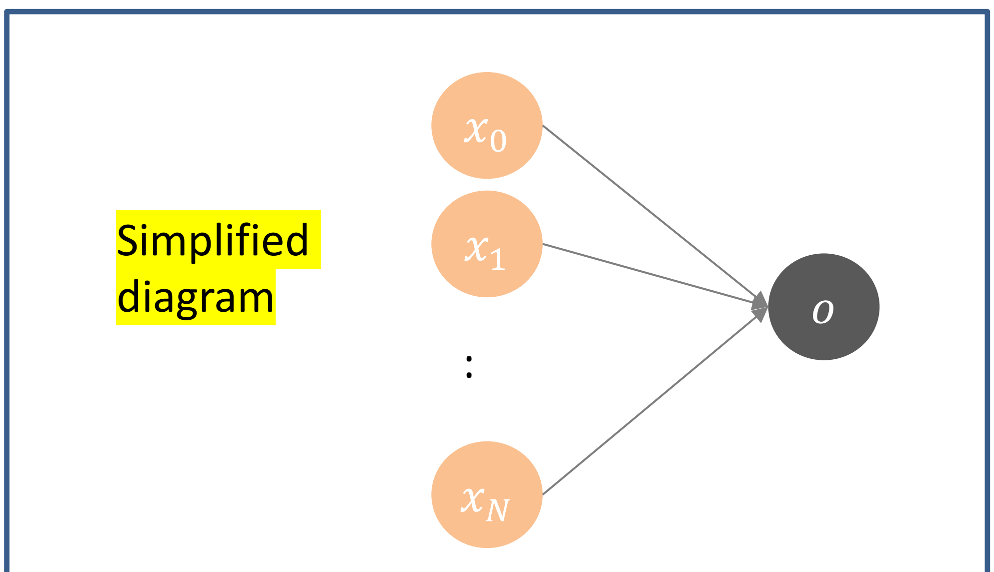
Outline

- **The Perceptron and Deep Neural Networks**
- Power of NN: Computing logic functions and arbitrary functions

The Perceptron



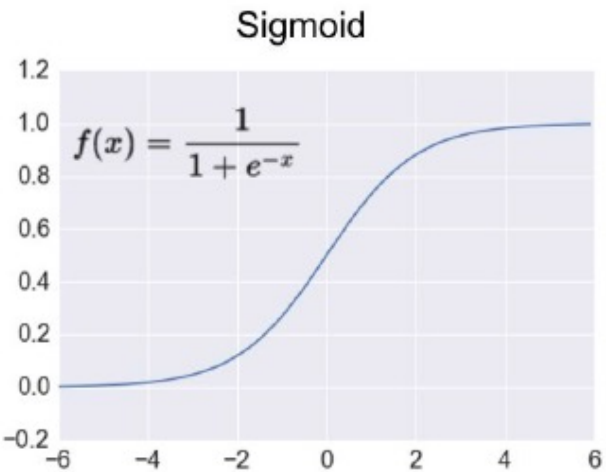
$$g(s) = g(\,(w_1x_1 + w_2x_2 + \cdots + w_Nx_N) + b)$$

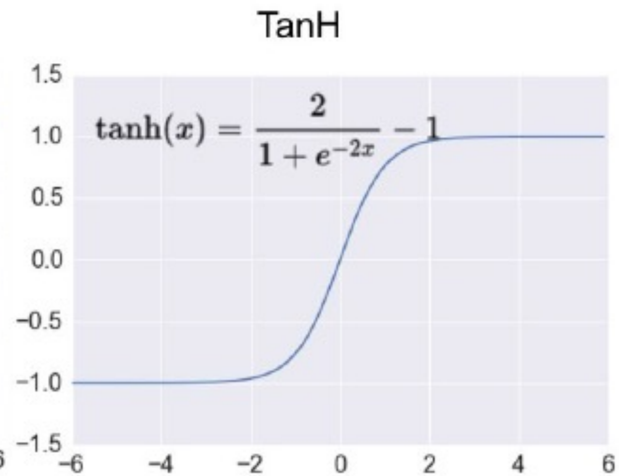


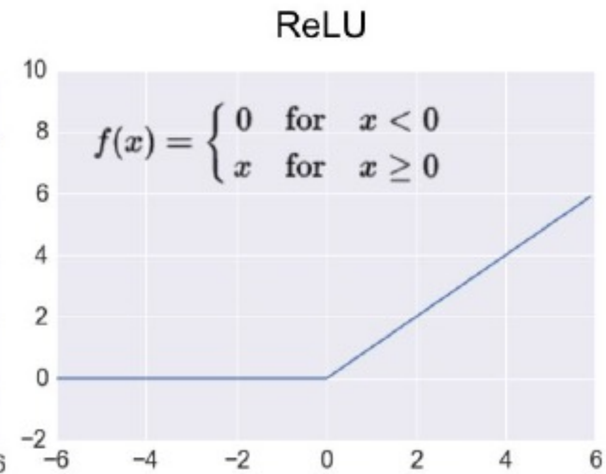
Notation:

- Weight w and parameter θ are used interchangeably.
- w_0 and b are the same with x_0=1

Common Activation Functions







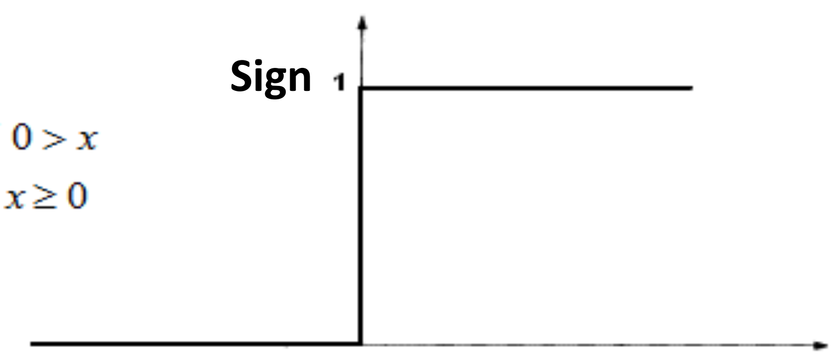
Output values [0,1]
For logistic regression

Output values [-1,1]
Hyperbolic tangent
Zero centered

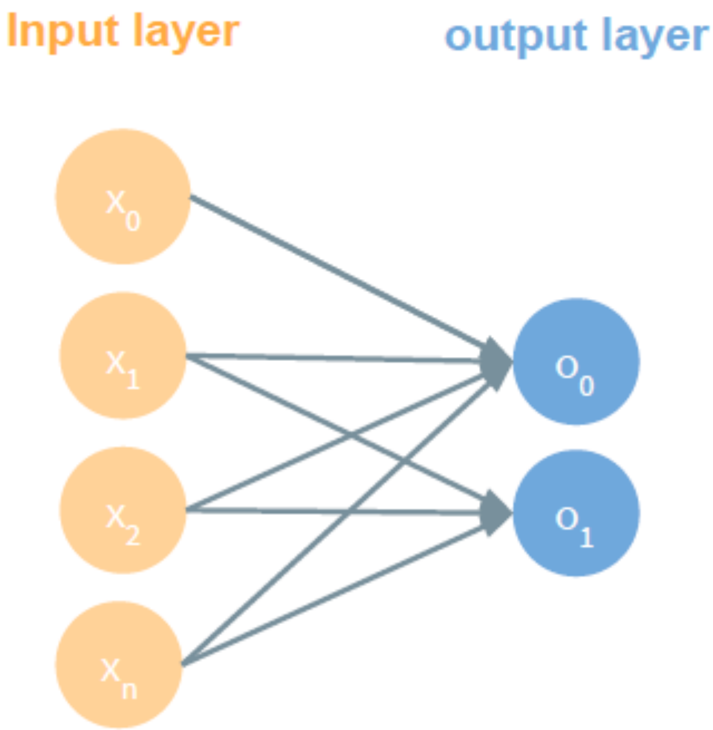
Good for backpropagation

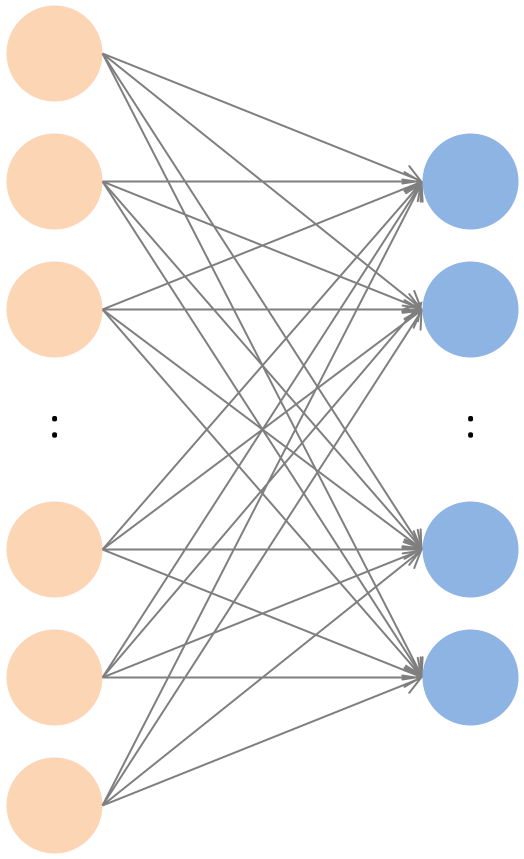
Unit step (threshold)

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$

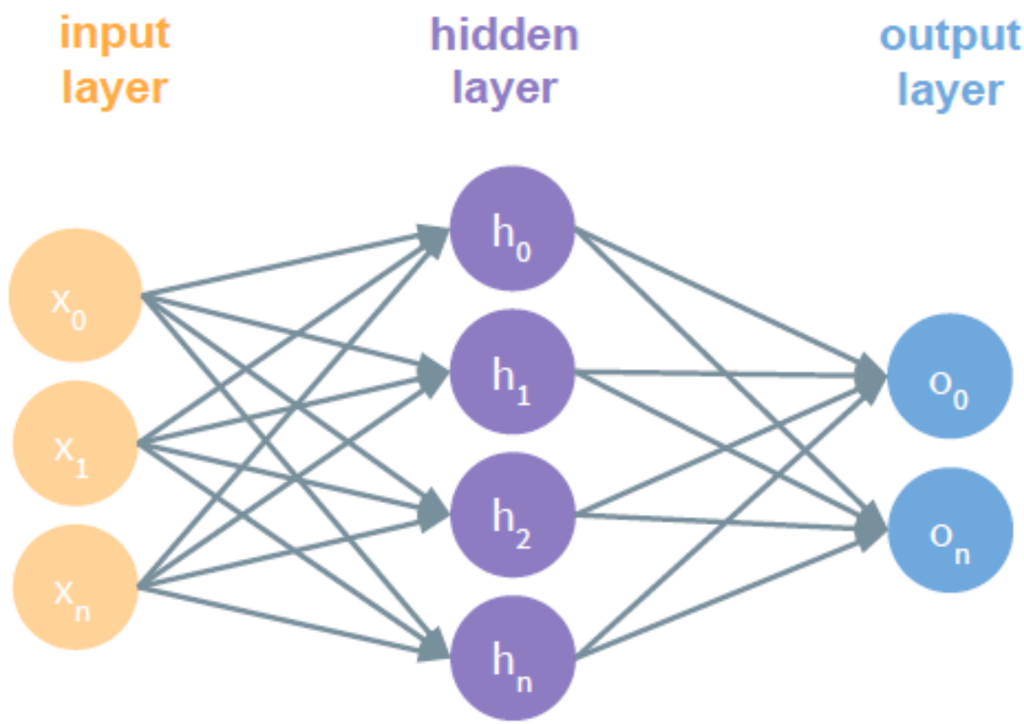


Multi-Output Perceptron

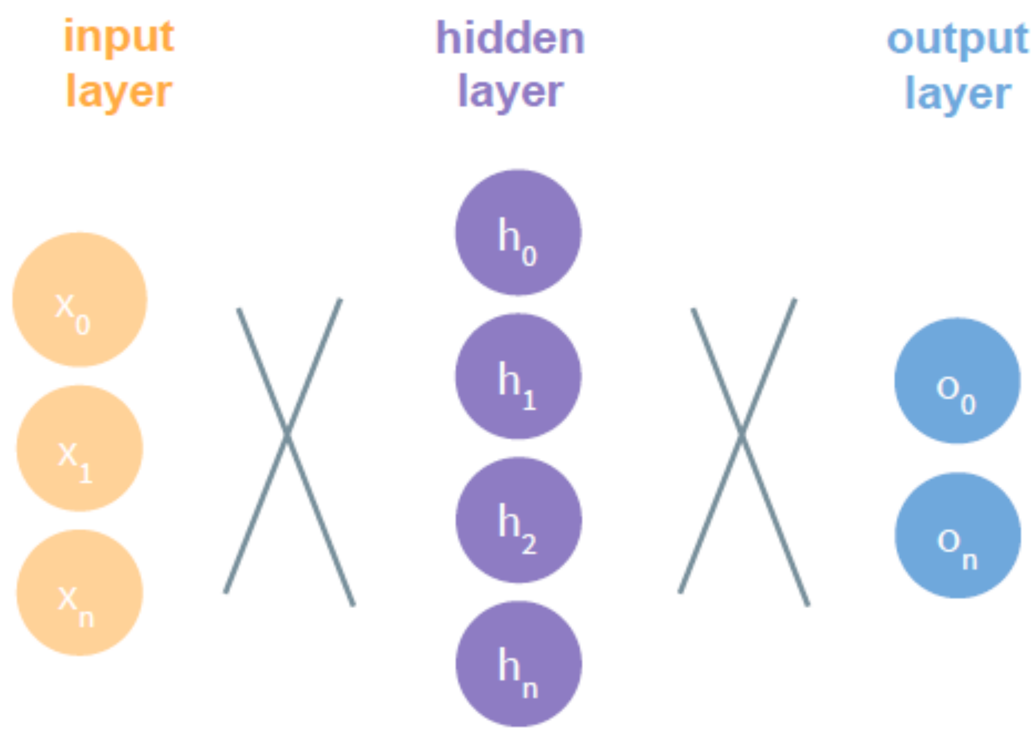




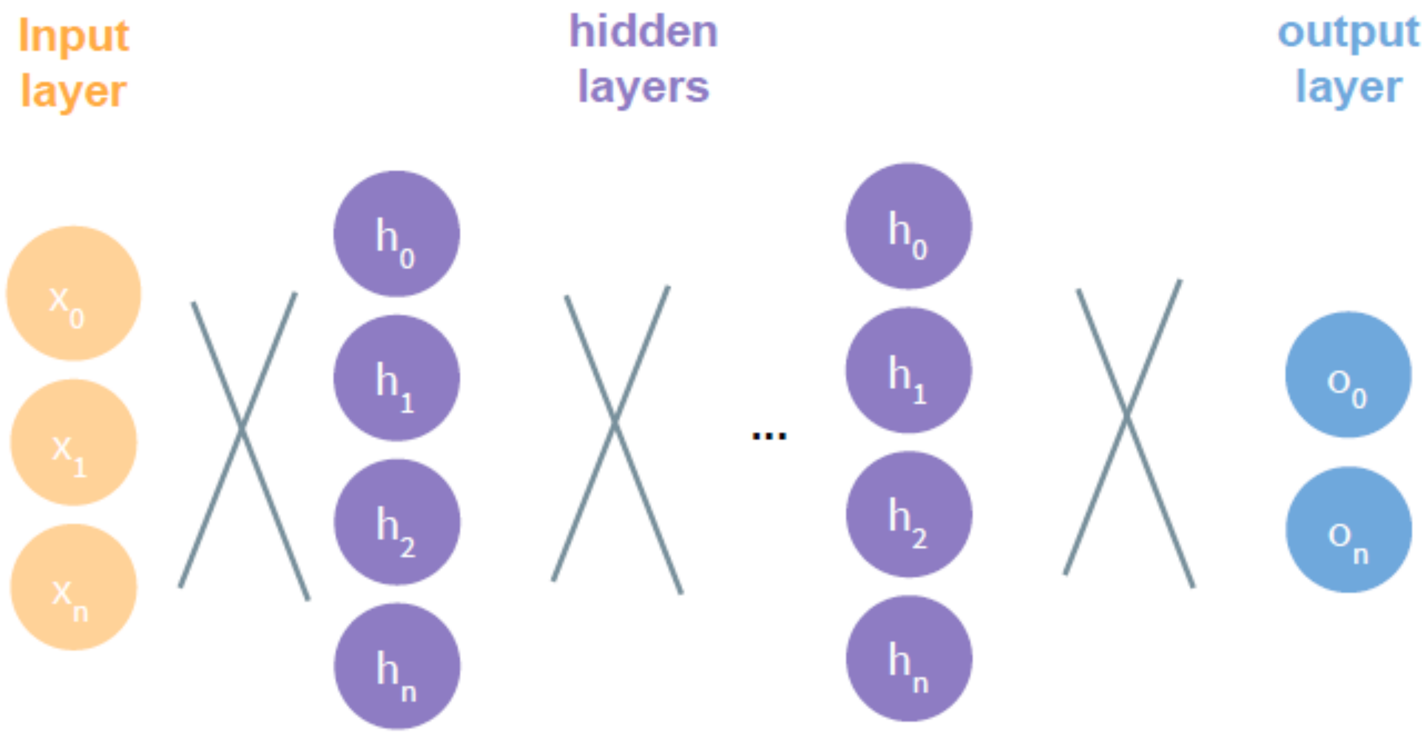
Multi-Layered Perceptron

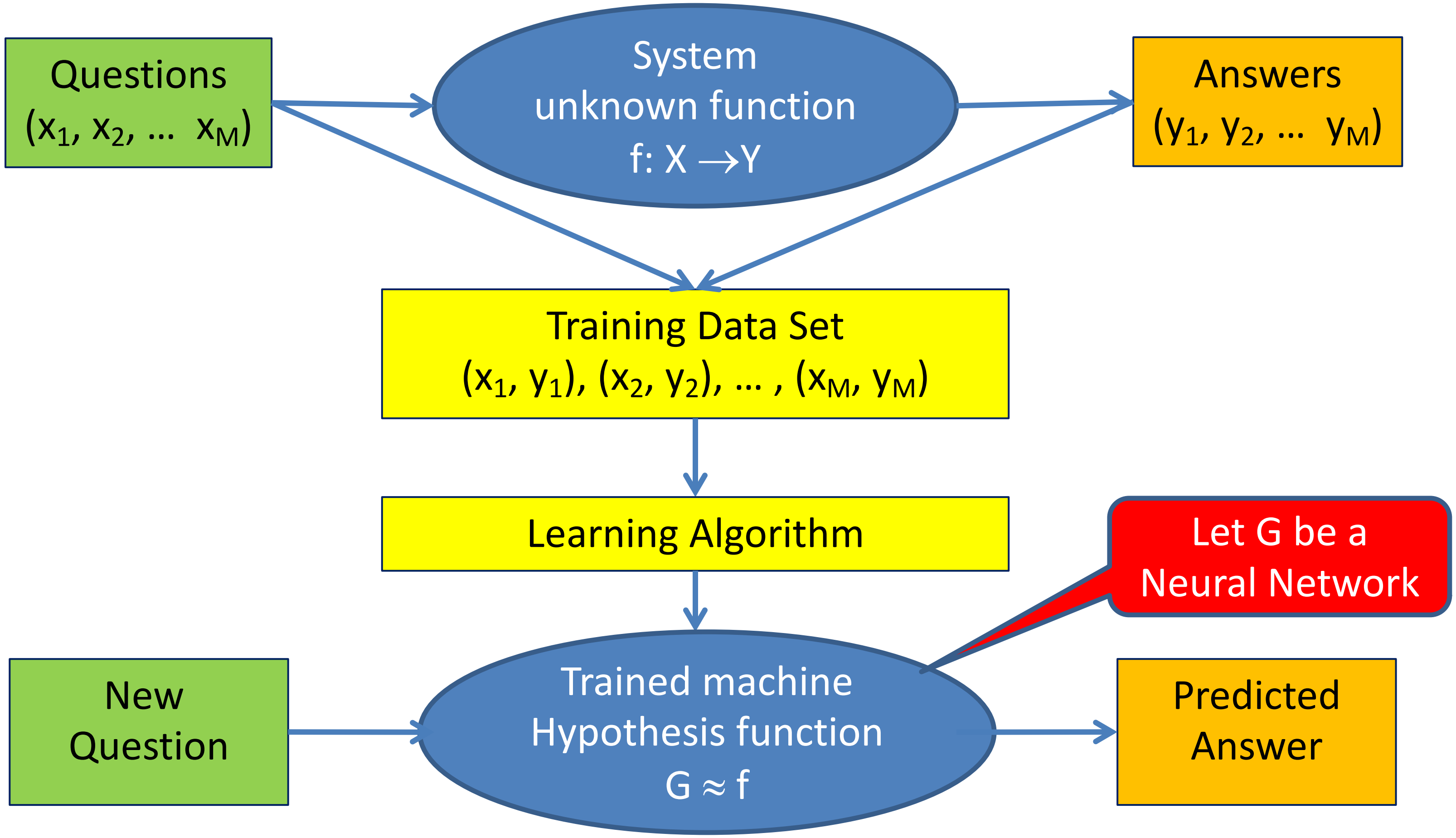


Multi-Layered Perceptron (simplified diagram)



Deep Neural Network





- Deep Learning uses NN (instead of linear models) for G. Why?
- NN can approximate any function (to be shown)

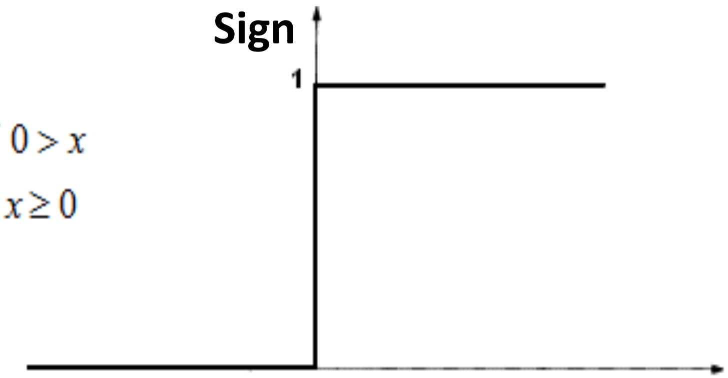
Outline

- The Perceptron and Deep Neural Networks
- **Power of NN: Computing logic functions and arbitrary functions**

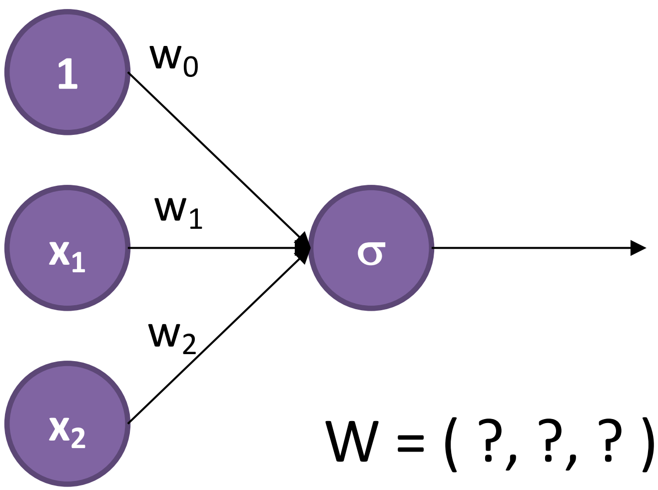
Computing OR

x_1	x_2	$OR(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	1

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



Activation function



$W = (\,?, ?, ?\,)$

$w_0 = ?$ when $x_1 = x_2 = 0$

$w_0 < 0$

$w_1 = ?$ when $x_1 = 1$, $x_2 = 0$

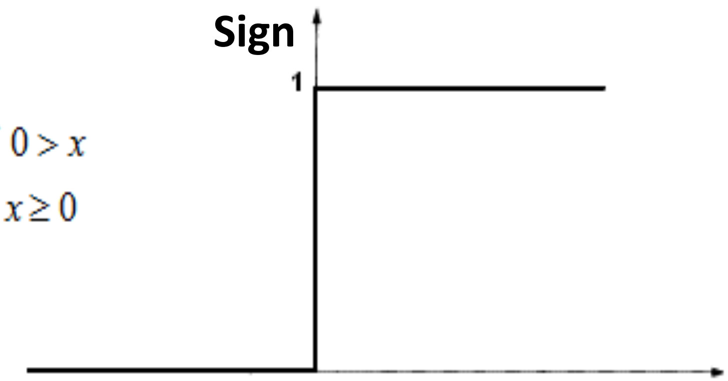
$w_1 + w_0 > 0$

$w_2 = ?$ w_2 same as w_1

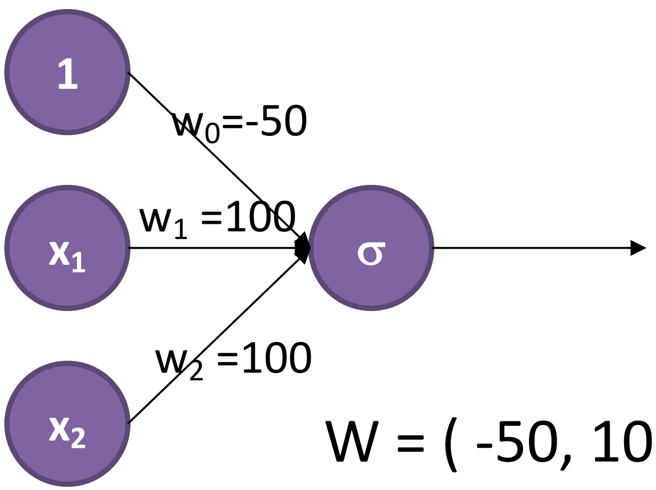
Computing OR

x_1	x_2	$OR(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	1

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



Activation function



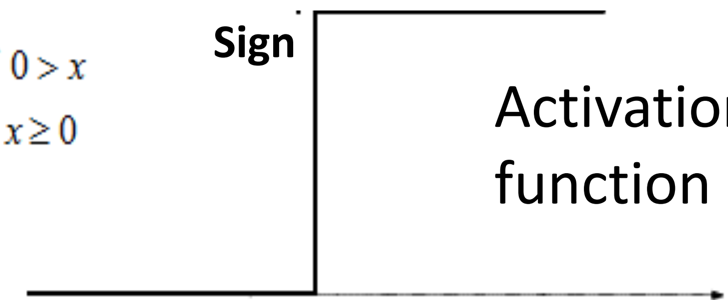
W = (-50, 100, 100)

x_1	x_2	$OR(x_1, x_2)$
0	0	
0	1	
1	0	
1	1	

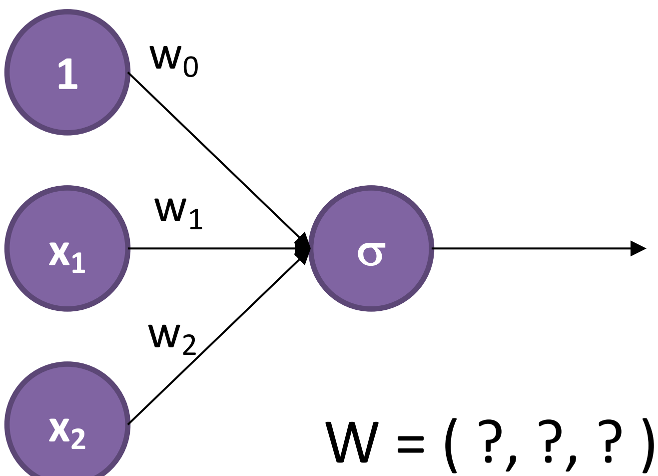
Computing AND

x_1	x_2	AND(x_1 ,x_2)
0	0	0
0	1	0
1	0	0
1	1	1

$$f(x)=\begin{cases}0 \text{ if } 0>x\\ 1 \text{ if } x\geq 0\end{cases}$$



Activation function



W = (?, ?, ?)

w_0 = ?

when x_1= x_2= 0

$w_0 < 0$, say w_0 =-150

$w_1 = w_2$ = ?

when x_1=1, x_2= 0

$w_1 + w_0 < 0$

when x_1=1, x_2= 1

$w_1 + w_2 + w_0 > 0$

$w_i > 0$; $|w_i|<|w_0|$;

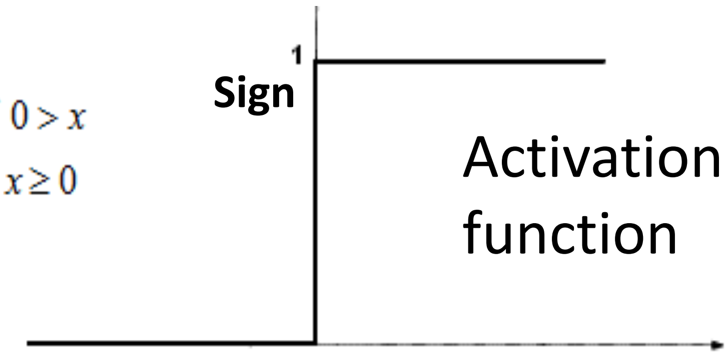
$2|w_i|>|w_0|$;

Say w_i = 100

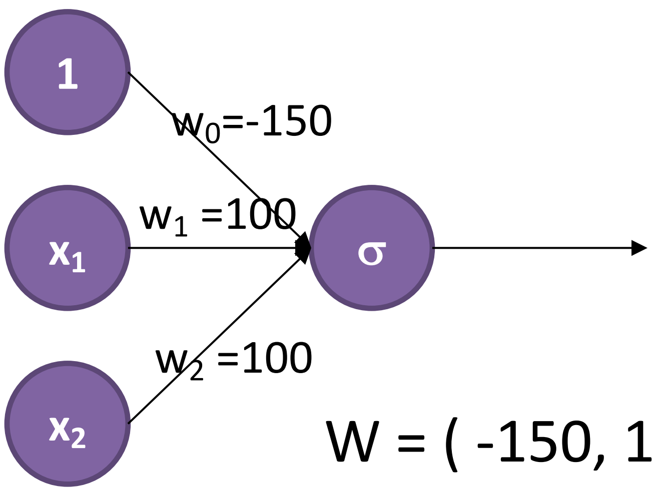
Computing AND

x_1	x_2	AND(x_1 ,x_2)
0	0	0
0	1	0
1	0	0
1	1	1

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



Activation function



W = (-150, 100, 100)

x_1	x_2	AND(x_1 ,x_2)
0	0	
0	1	
1	0	
1	1	

What is XOR?

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

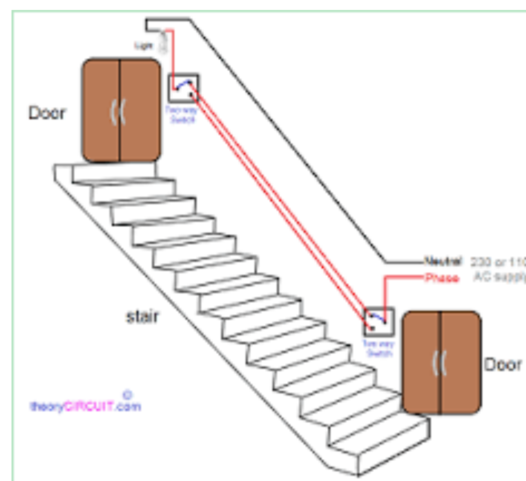
$x_1 \oplus x_2$

x_1 x_2

y

y – one car passes

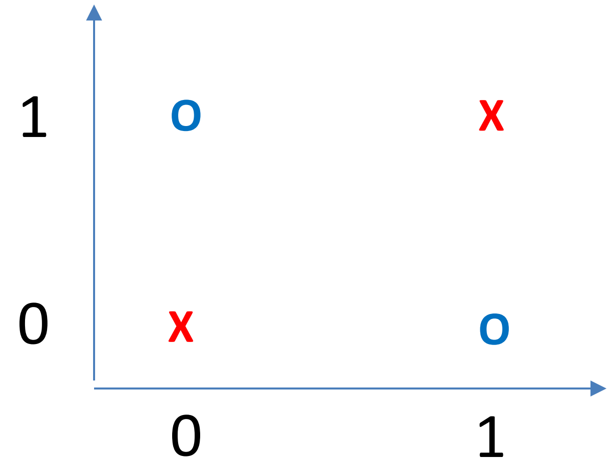
- $y = x_1 \oplus x_2 = 1$ iff either x_1 or x_2 is 1, but not both
- Real world examples:
  - 2 cars with a narrow road
  - 2-way switch
  - Parity bit

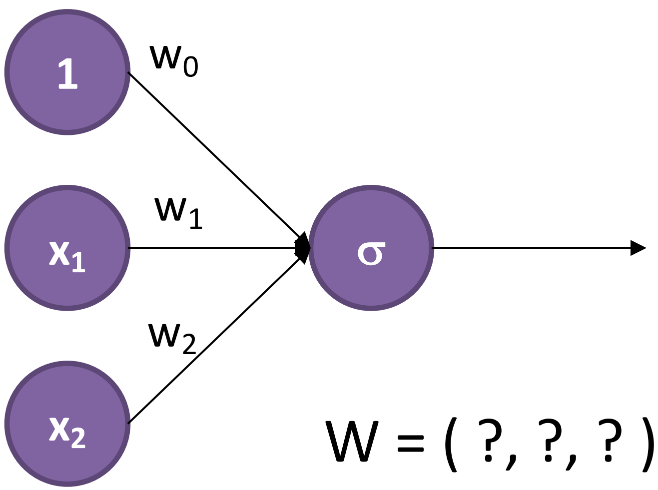


What about XOR?

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

$x_1 \oplus x_2$





W = (?, ?, ?)

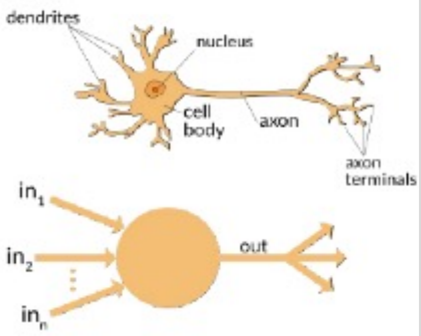
w_0 = ?

$w_0 < 0$, say $w_0 = -50$
when $x_1 = x_2 = 0$

$w_1 = w_2$ = ?

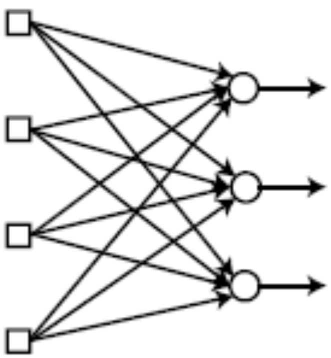
$w_i + w_0 > 0$
say $w_i = 100$
$w_1 + w_2 + w_0 < 0$
when $x_1 = 1$, $x_2 = 1$
????

Historical Perspective



Perceptron

1954



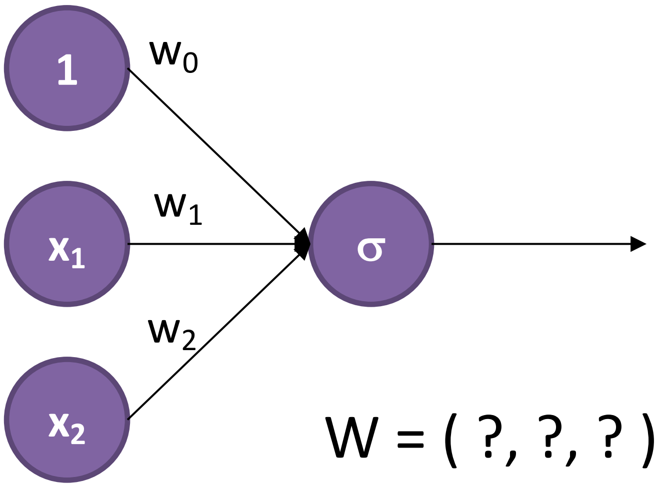
Single-layer Neural Network

Quickly found it couldn't compute simple functions such as XOR (MIT book by Minsky)

What about XOR?

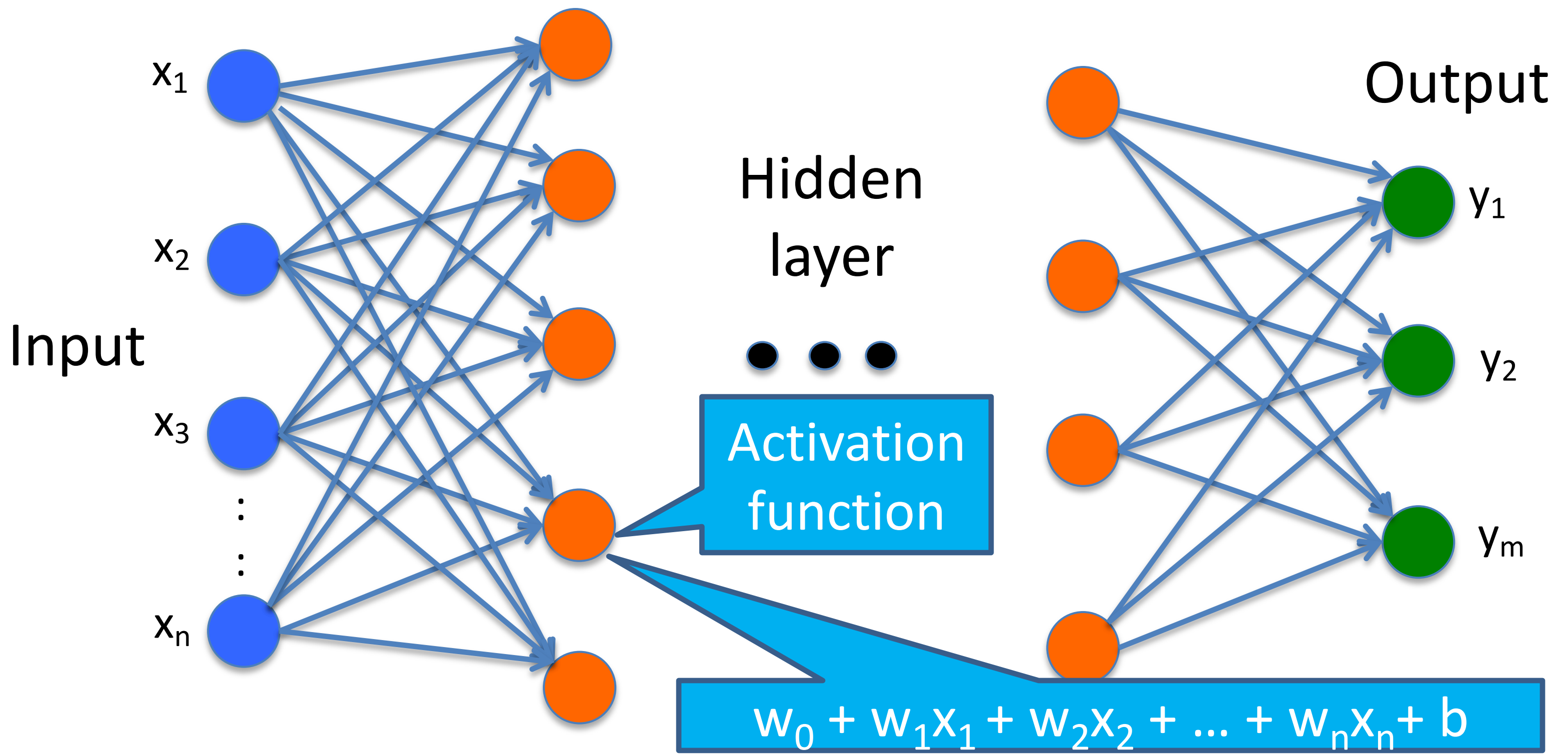
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Minsky and Papert (1969) showed that a single artificial neuron is incapable of implementing some simple functions such as the XOR logical function, parity, connectivity



W = (?, ?, ?)

Can Multilayer Perceptron Help?



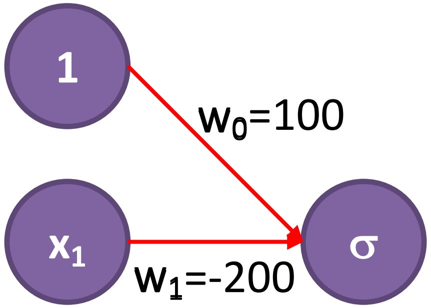
- Since each perceptron is a linear function, no matter how many perceptrons or layers, the final output remains linear.
- Hence multilayer neural network is no more powerful than a single perceptron in solving XOR (which is nonlinear).
- Pitfall : activation function is nonlinear.

Is this argument valid?

Power of layering perceptrons

NOT

x_1	
0	1
1	0



W = (?, ?) $W_{NOT} = (100, -200)$

Logical complete with NOT, AND and OR

i.e., any logical function can be computed.

$(\bar{x}_1 \text{AND } \bar{x}_2)$ **OR** $(x_1 \text{AND } \bar{x}_2)$

How many logical functions?

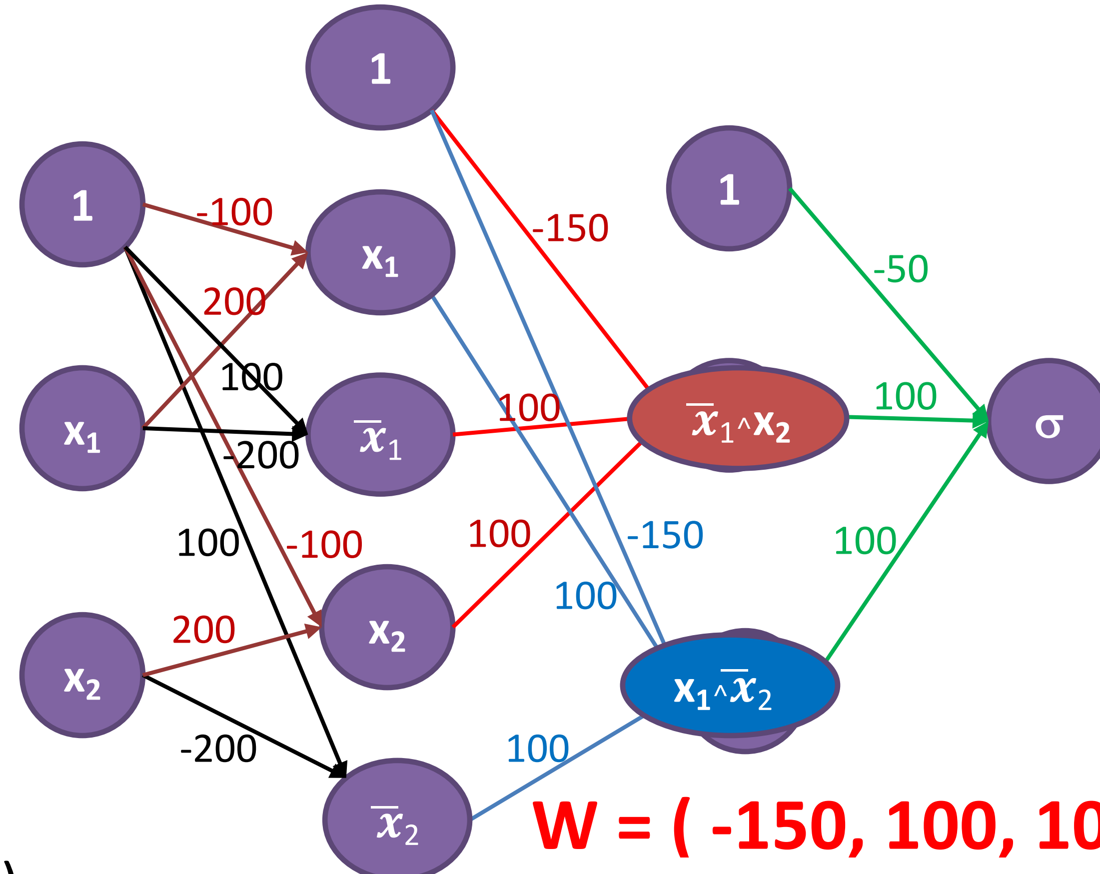
Answer: 2^4=16

x_1	x_2	
0	0	1
0	1	0
1	0	1
1	1	0

Power of layering perceptrons

$(\bar{x}_1 \text{AND } x_2)$ OR $(x_1 \text{AND } \bar{x}_2)$

x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0



W_{NOT} = (100, -200)

W_{IND} = (-100, 200)

W_{AND} = (-150, 100, 100)

W_{OR} = (-50, 100, 100)

W = (-150, 100, 100)

W = (-150, 100, 100)

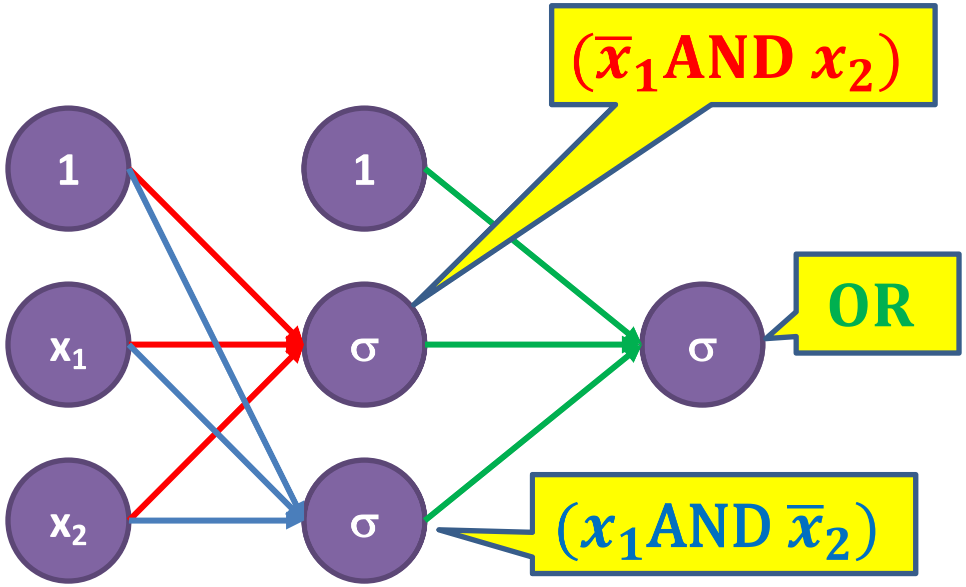
W = (-50, 100, 100)

Power of layering perceptrons

"NOR" gate can be constructed with two layers

$x_1 \oplus x_2 = (\bar{x}_1 \text{ AND } x_2) \text{ OR } (x_1 \text{AND } \bar{x}_2)$

x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0

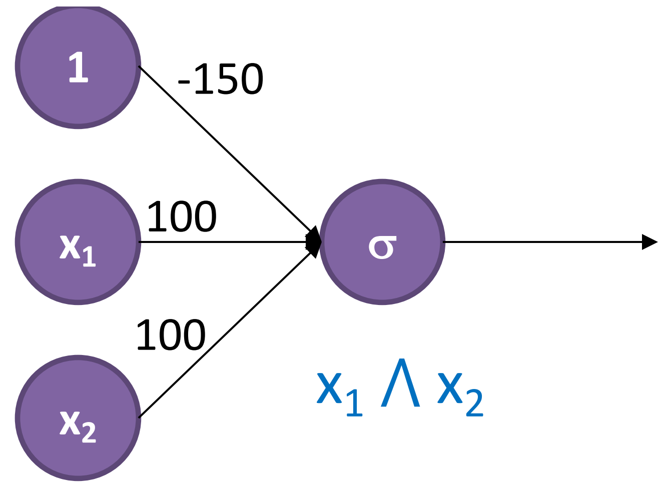


Remember
W_{AND} = (-150, 100, 100)
W_{OR} = (-50, 100, 100)

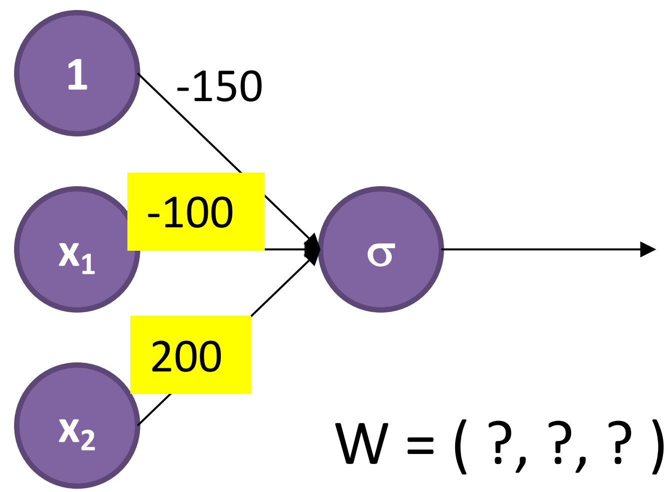
W = (?, ?, ?)
W = (?, ?, ?)
W = (?, ?, ?)

Computing $(\overline{x}_1 \text{ AND } x_2)$

x_1	x_2	AND(x_1, x_2)
0	0	0
0	1	0
1	0	0
1	1	1



x_1	x_2	$(\overline{x}_1 \text{ AND } x_2)$
0	0	0
0	1	1
1	0	0
1	1	0



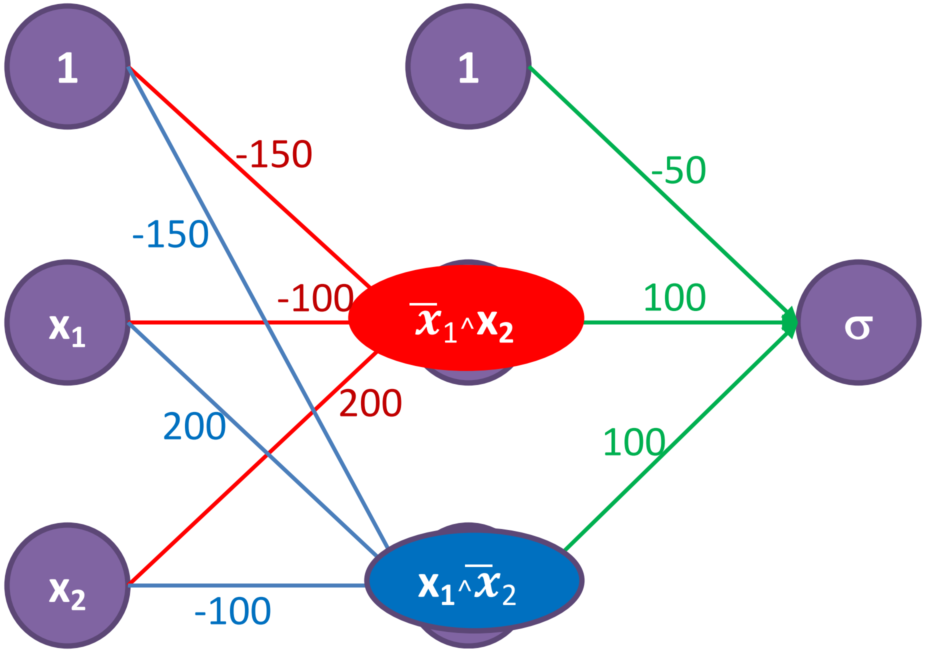
say w_0 = -150 $w_1 = w_2$ = ?

w_1 = -100; w_2 = 200

Power of layering perceptrons

$(\bar{x}_1 \textbf{AND } x_2)$ OR $(x_1 \textbf{AND } \bar{x}_2)$

x_1	x_2	
0	0	0
0	1	1
1	0	1
1	1	0



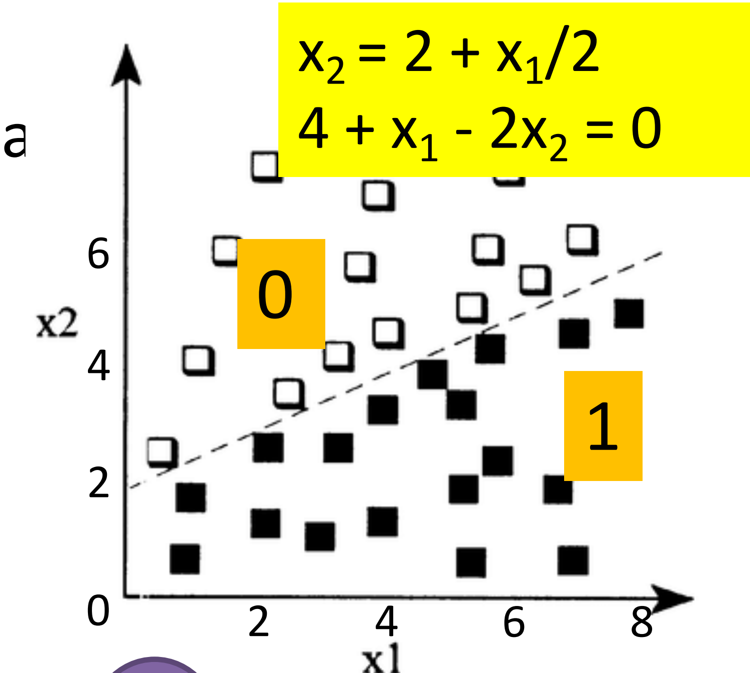
W_{AND} = (-150, 100, 100)
W_{OR} = (-50, 100, 100)
$(\bar{x}_1 \textbf{AND } x_2)$= (-150, -100, 200)

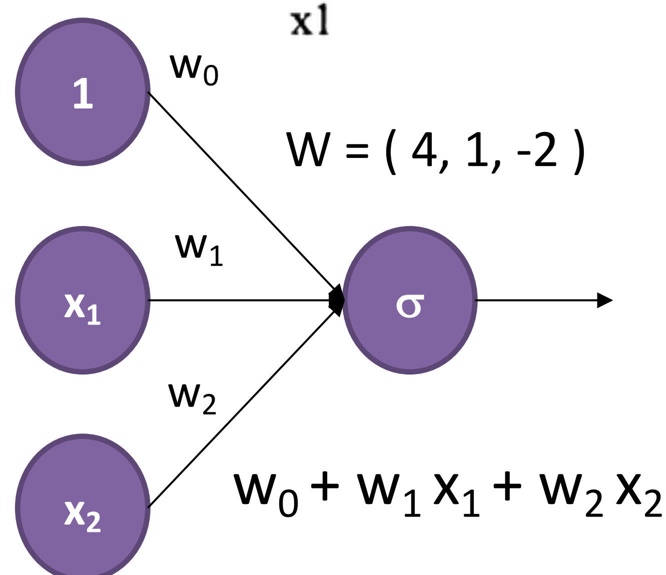
W = (-150, -100, 200)
W = (-150, 200, -100)
W = (-50, 100, 100)

Linearly-Separable Classification

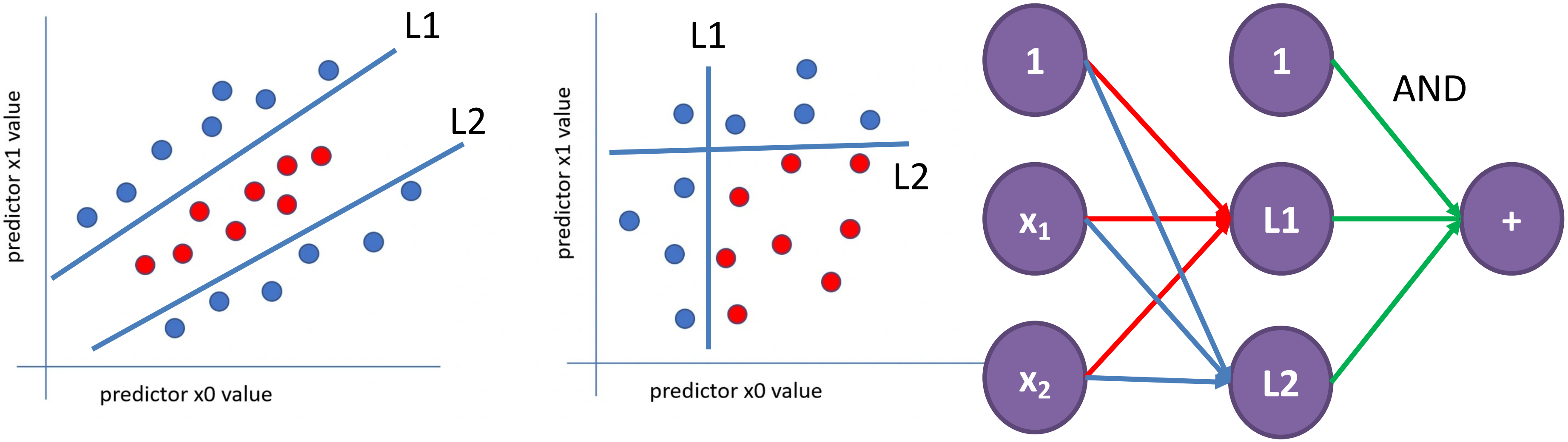
- **Linearly-separable** means there is a hyperplane, which can split two classes of input points into two half-spaces.
- In 2D plane, linearly-separable means there is a line separating points of one class from points of the other class.
- Classification problem can be solved by NN
- In general, the weights of NN corresponding to the hyperplane

$$w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

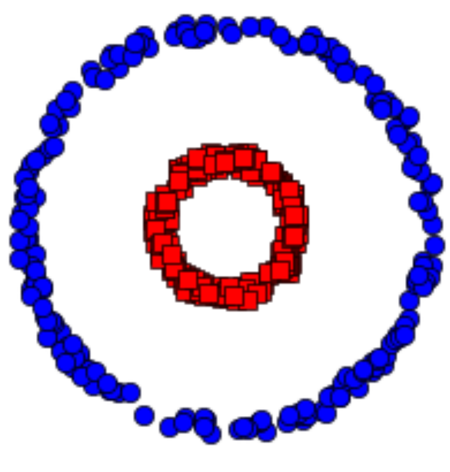


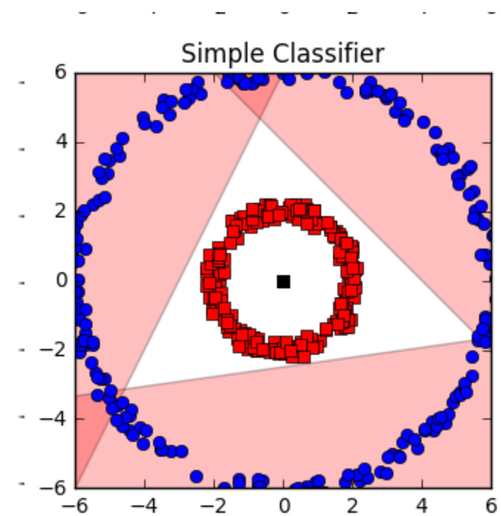


Non-Linearly Separable Data

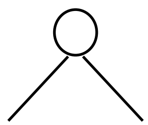
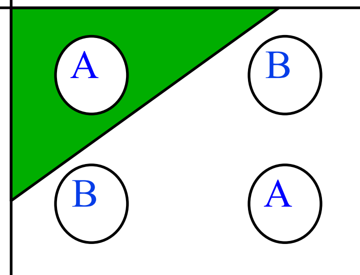
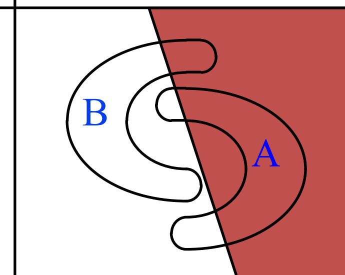
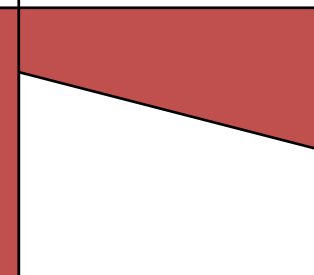
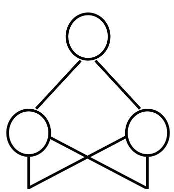
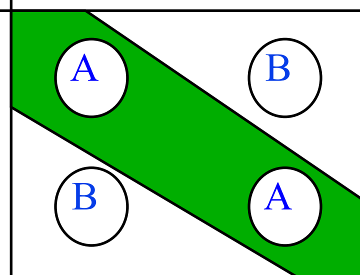
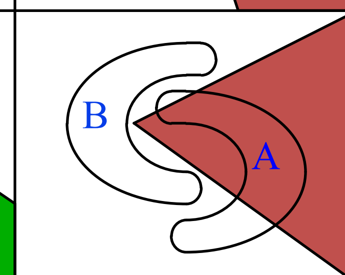
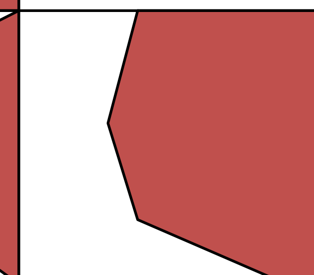
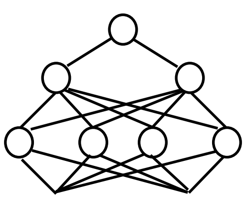
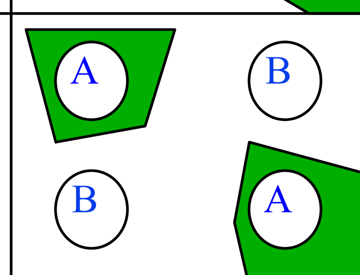
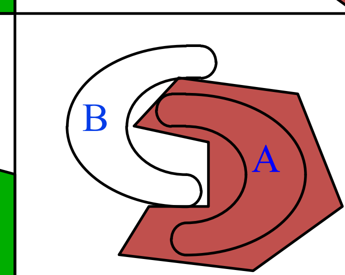
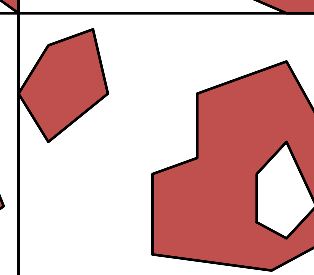


Non-linearly Separable classification problems can be solved by NN with more layers using "AND", "OR", "NOT" functionality



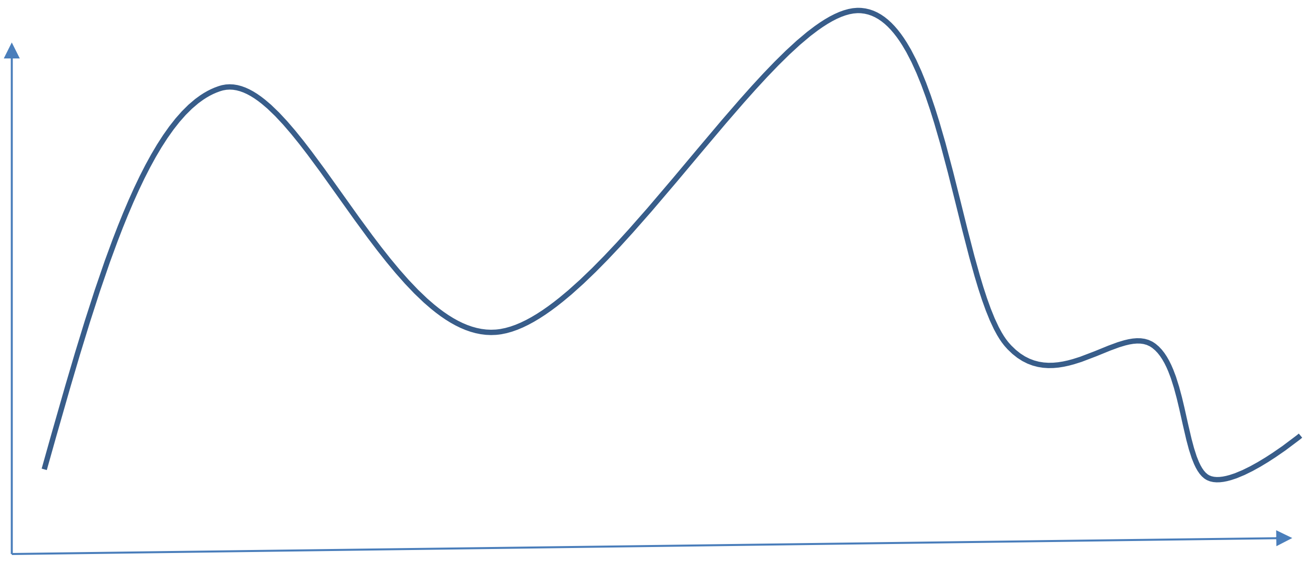


Different Non-Linearly Separable Problems

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Abitrary (Complexity Limited by No. of Nodes)			

Neural Networks – An Introduction Dr. Andrew Hunter

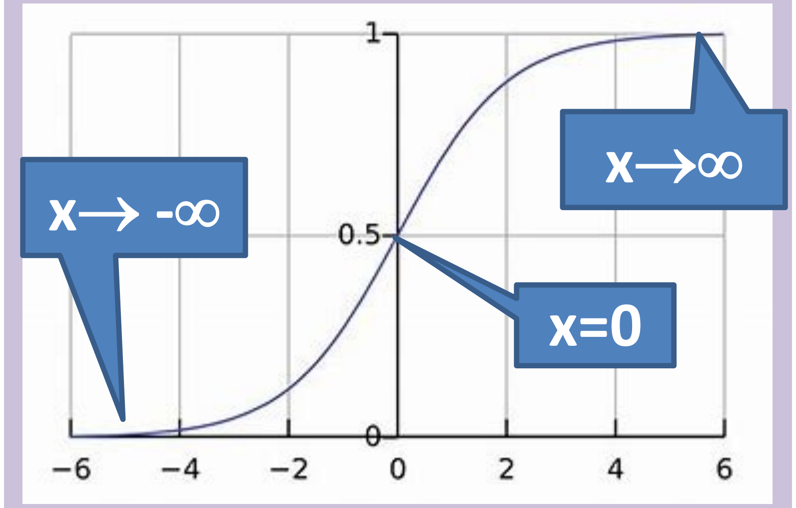
What about arbitrary functions?



Neural Network can compute any function

Consider **sigmoid** as activation function

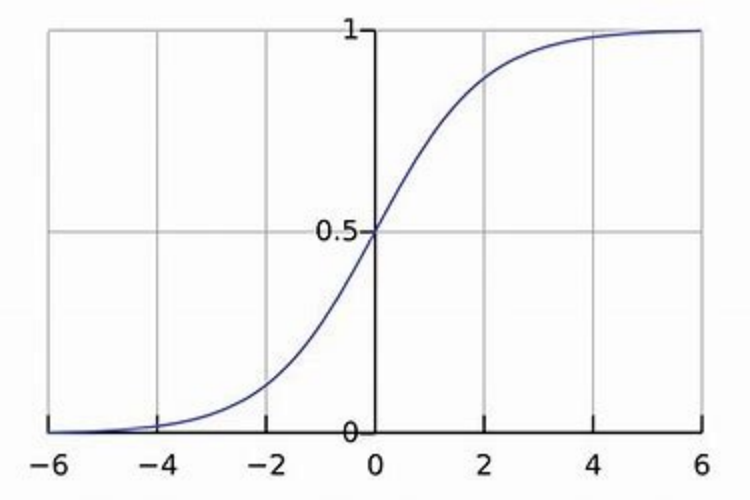
$$g(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

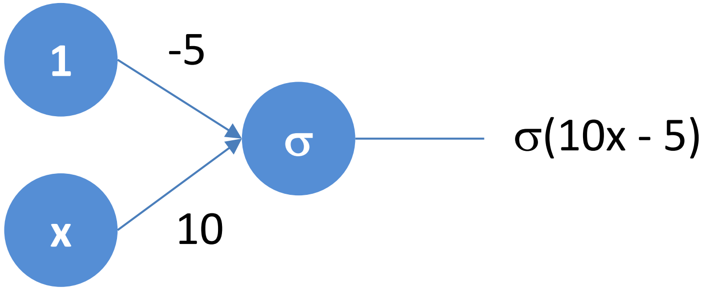


Neural Network can compute any function

Consider **sigmoid** as activation function

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



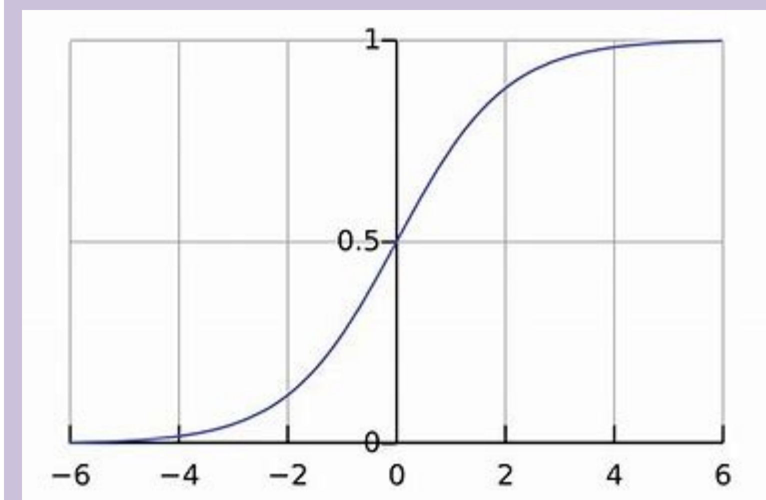


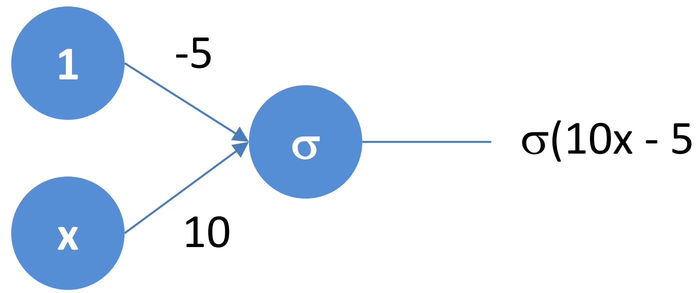
Assuming $0 \le x \le 1$

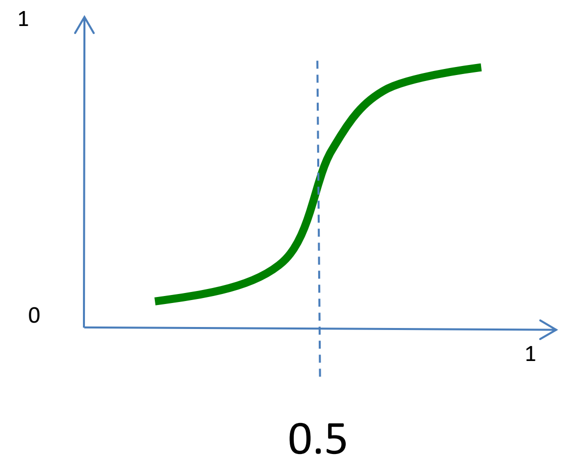
Neural Network can compute any function

Consider **sigmoid** as activation function

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



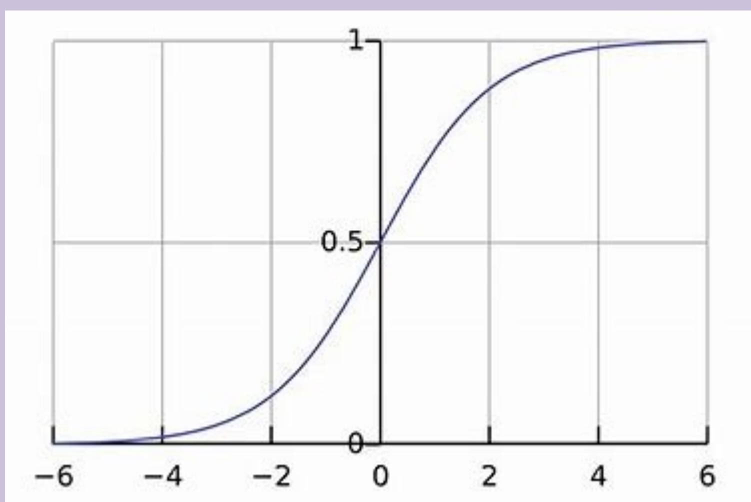


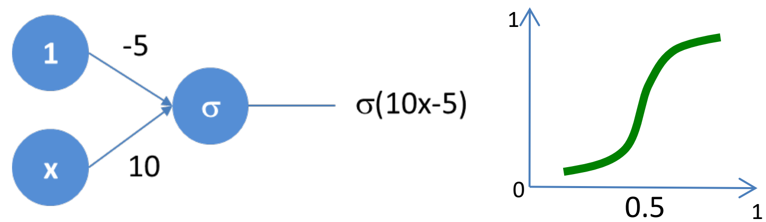


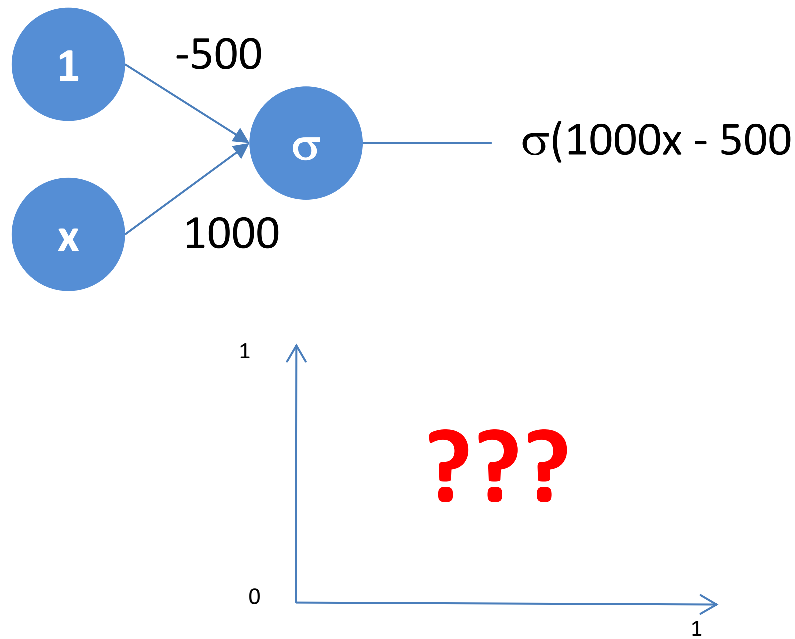
Neural Network can compute any function

Consider **sigmoid** as activation function

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



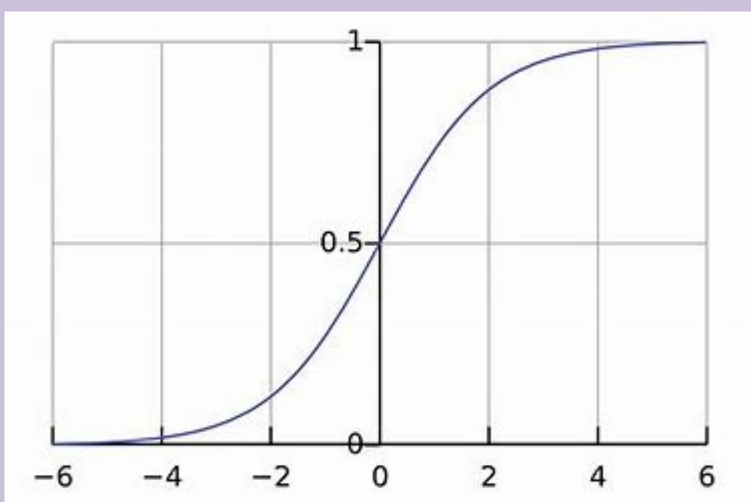


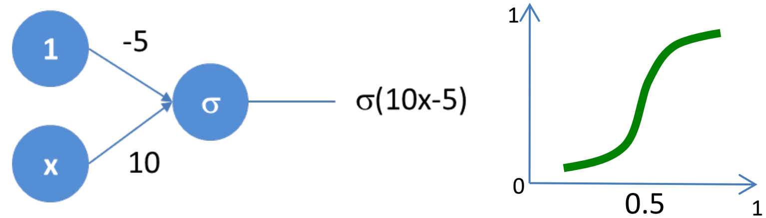


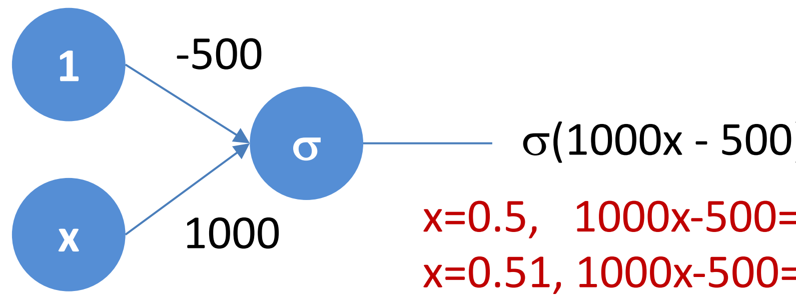
Neural Network can compute any function

Consider **sigmoid** as activation function

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

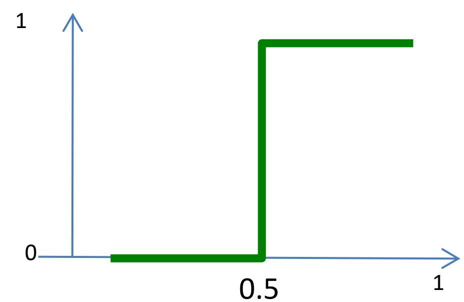






x=0.5, 1000x-500=0

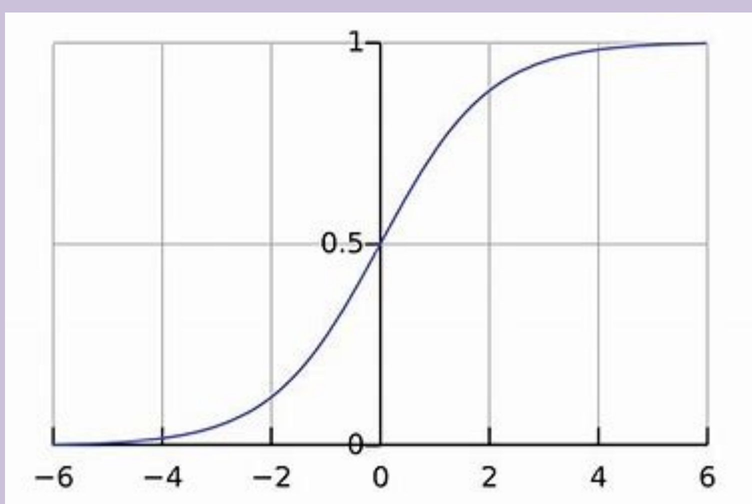
x=0.51, 1000x-500=10

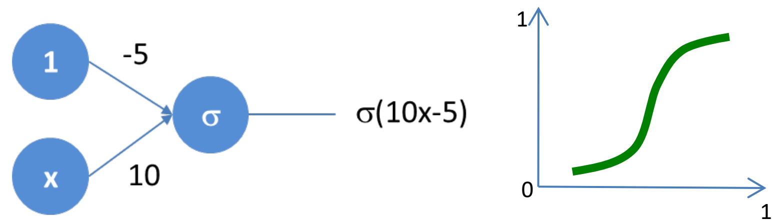


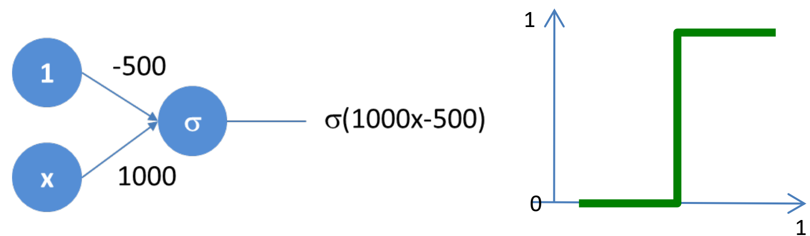
Neural Network can compute any function

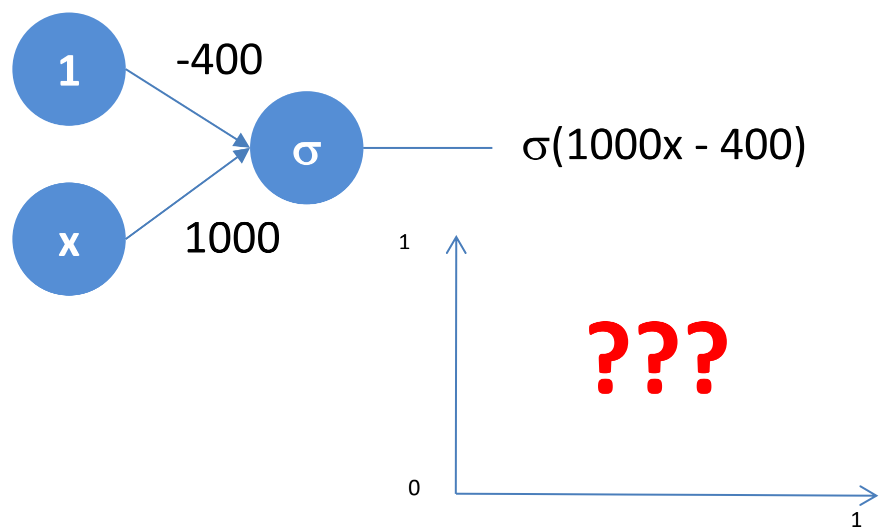
Consider **sigmoid** as activation function

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





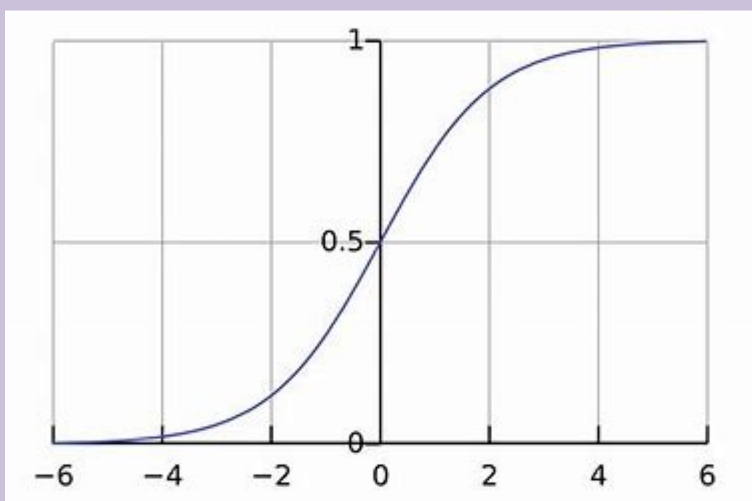


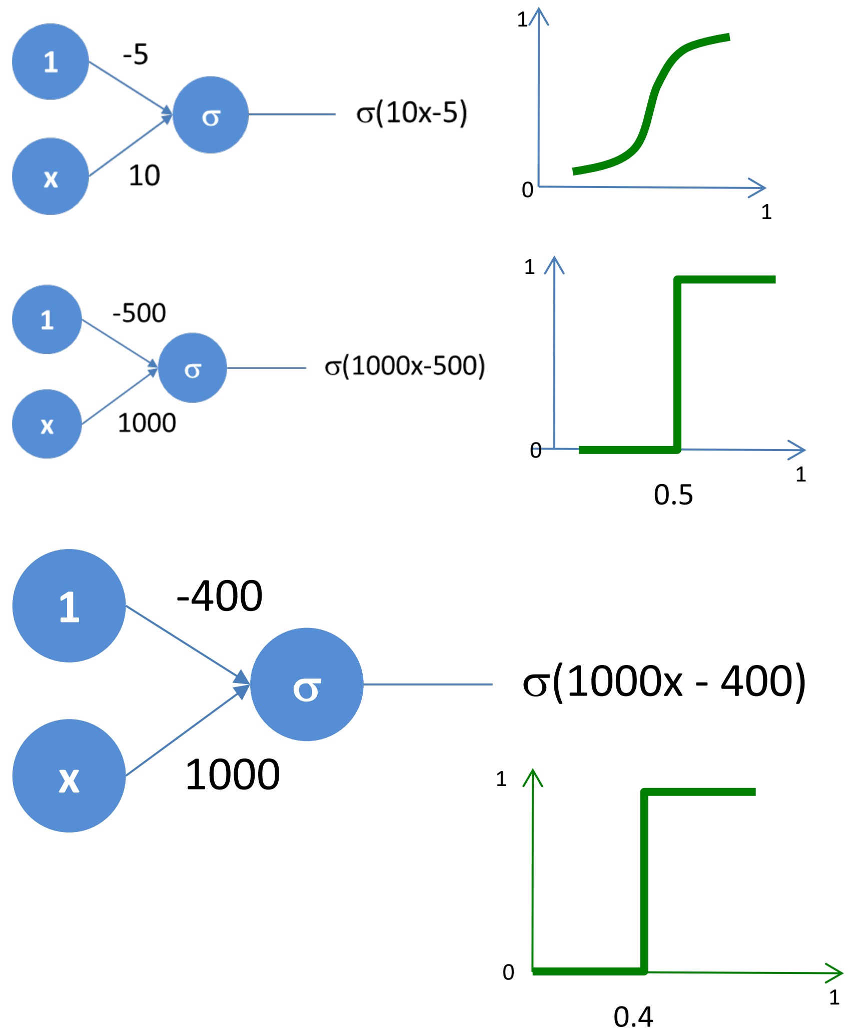


Neural Network can compute any function

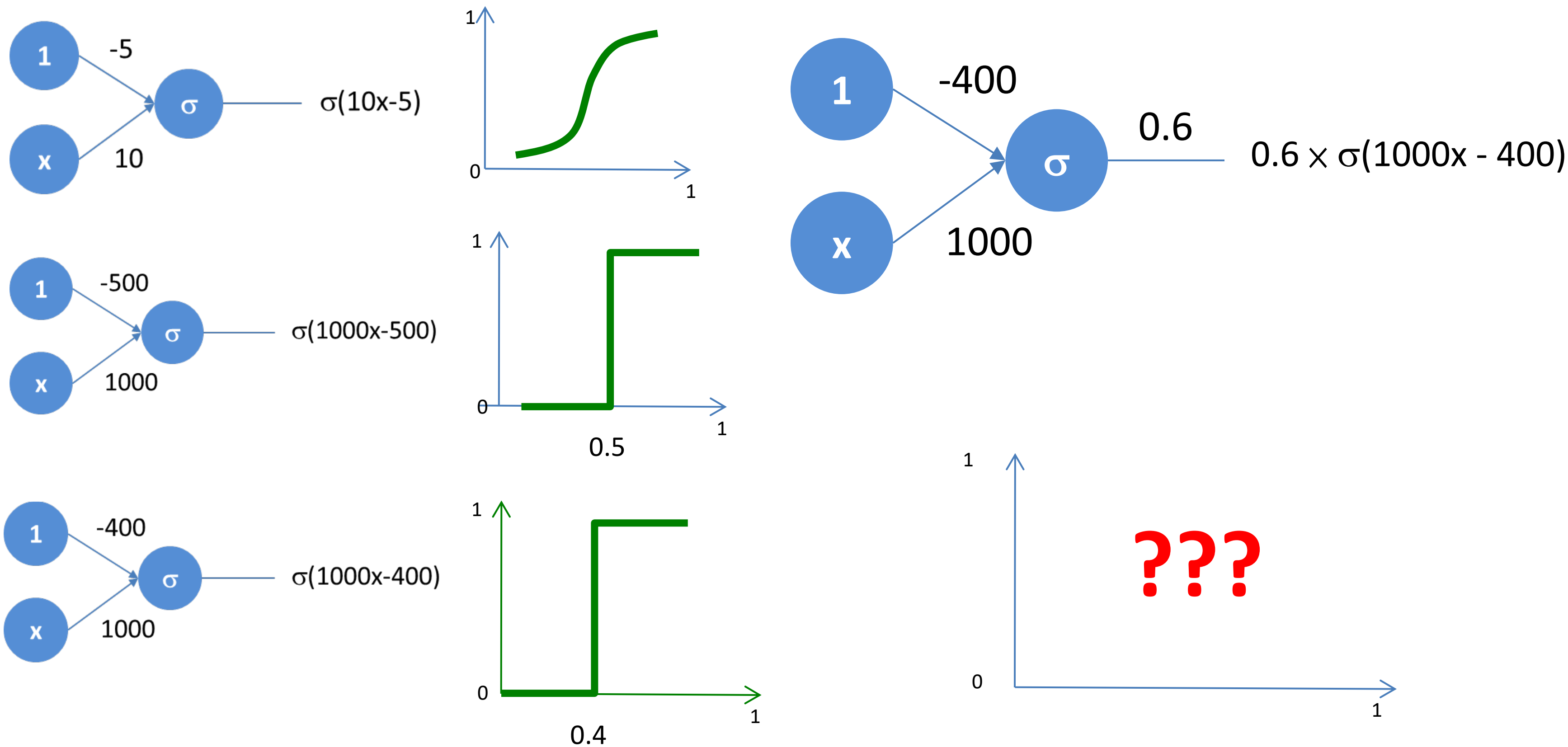
Consider **sigmoid** as activation function

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

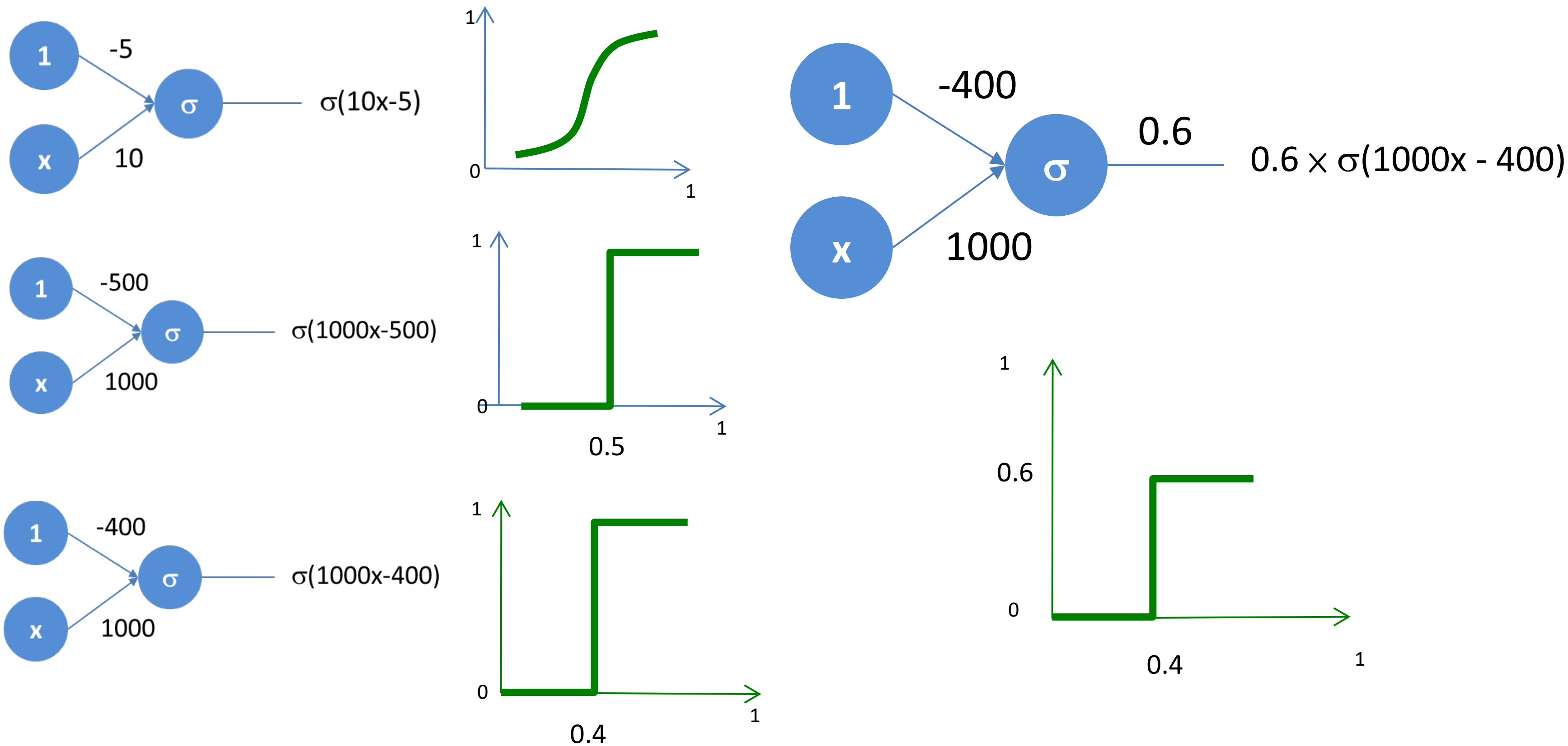


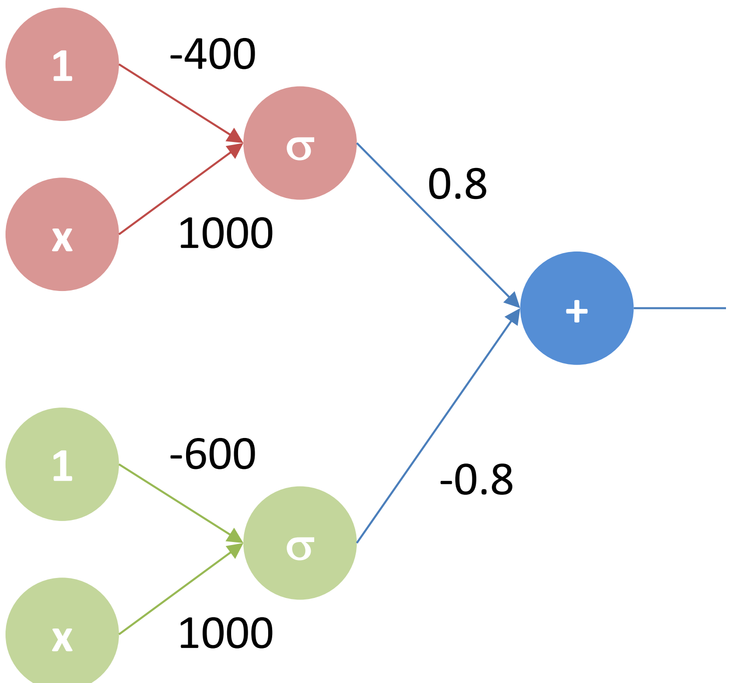


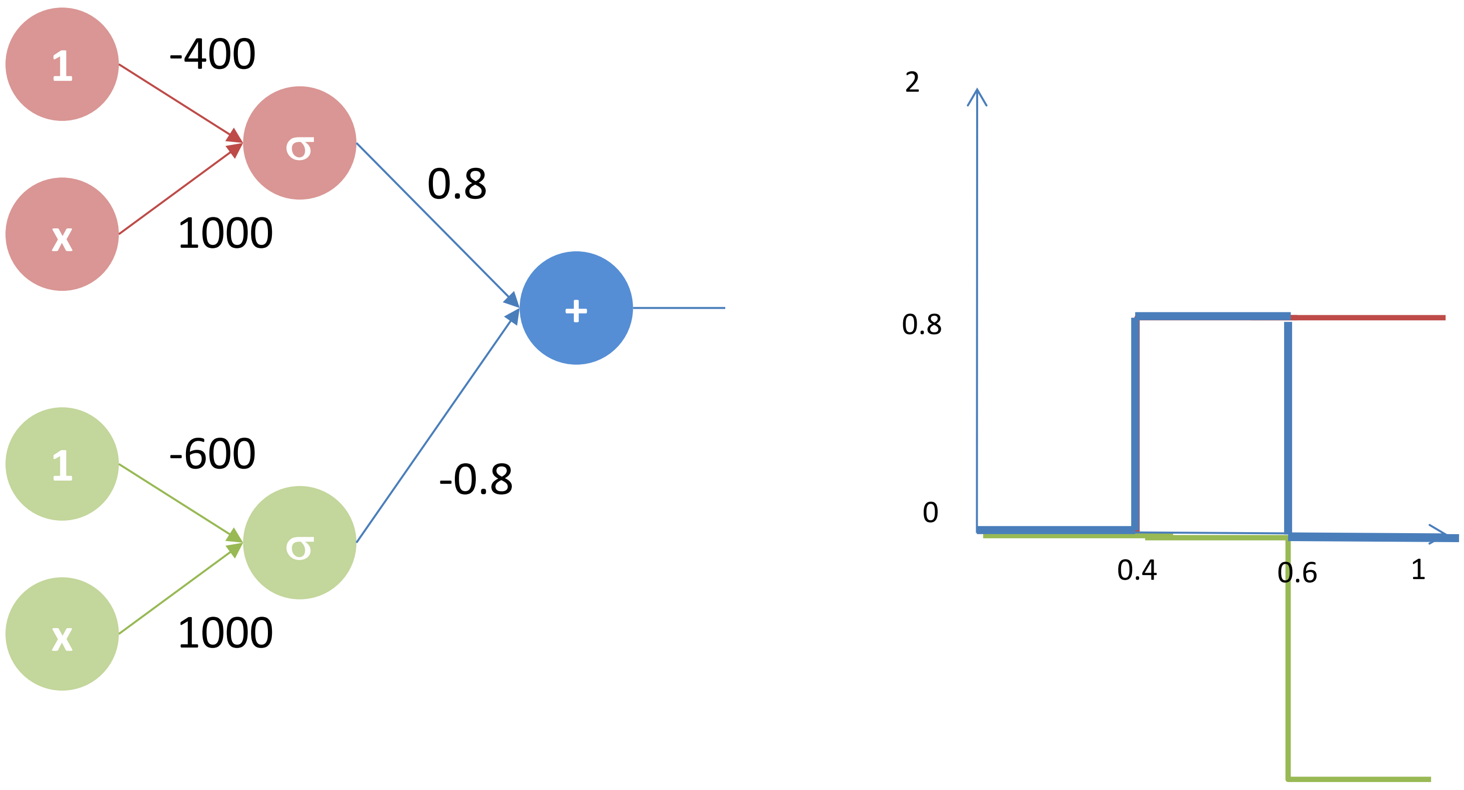
Neural Network can compute any function

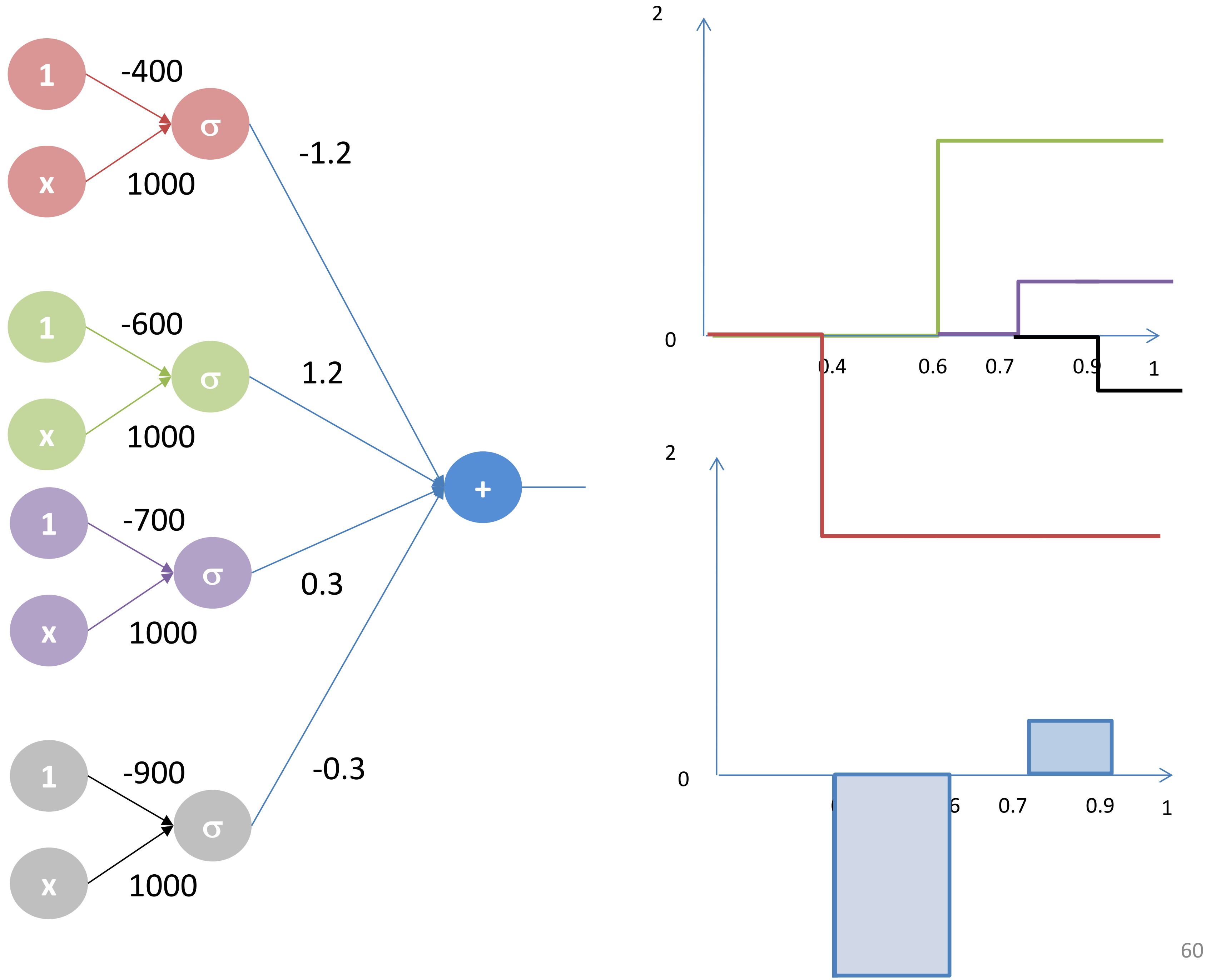


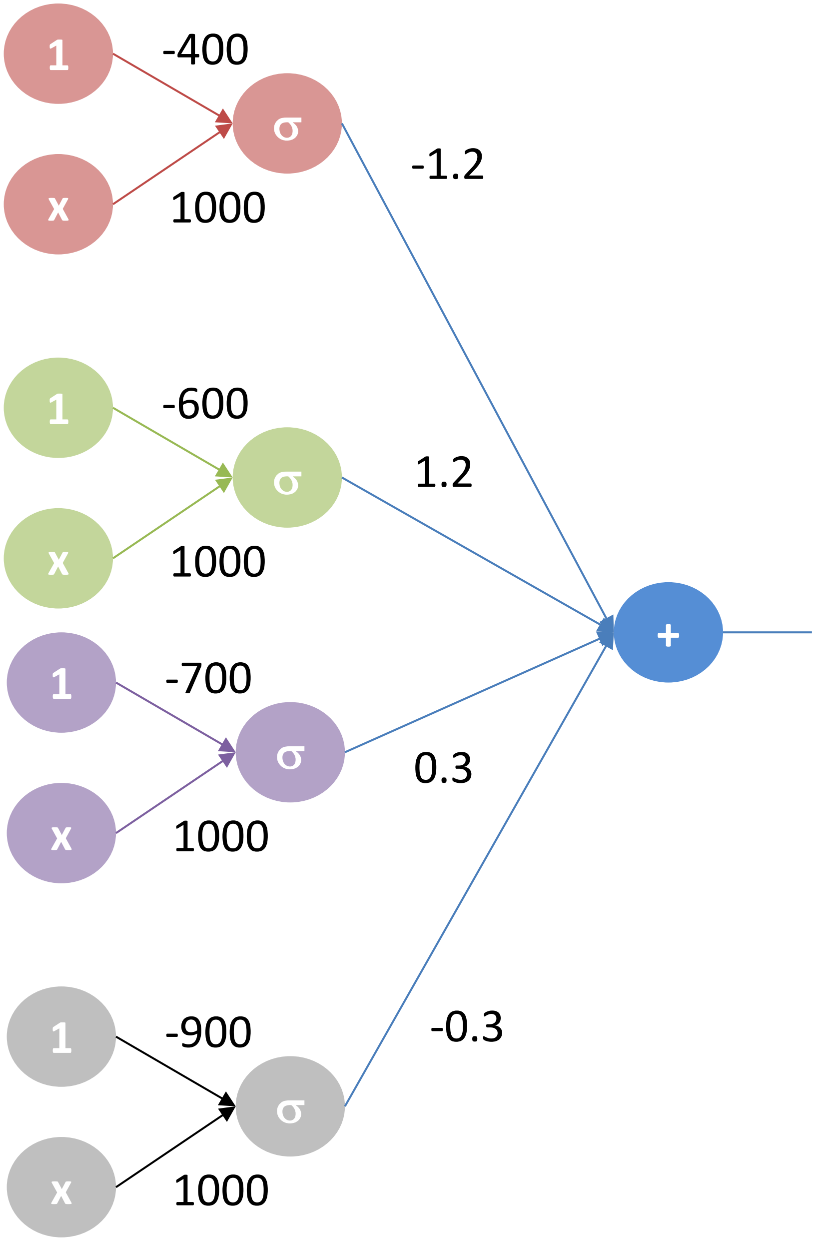
Neural Network can compute any function

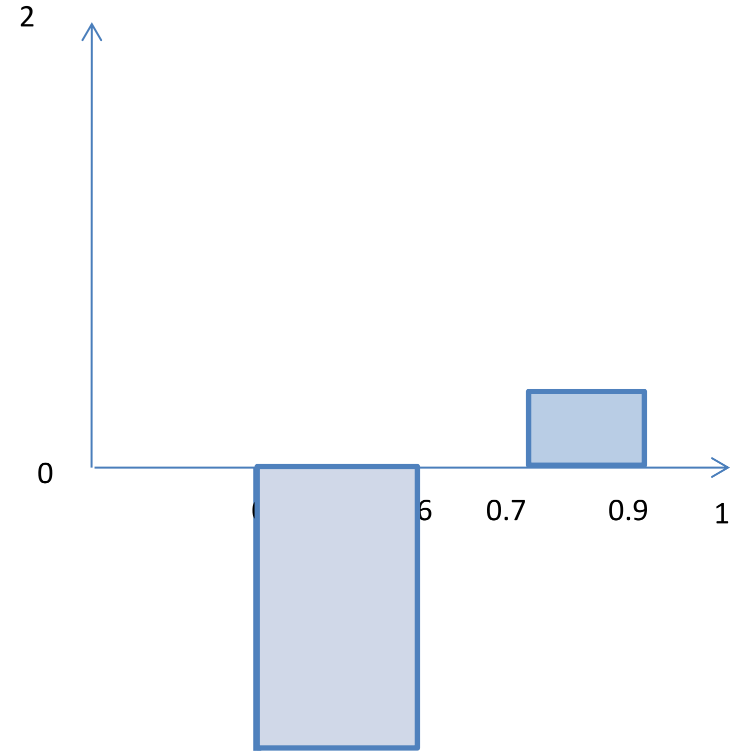


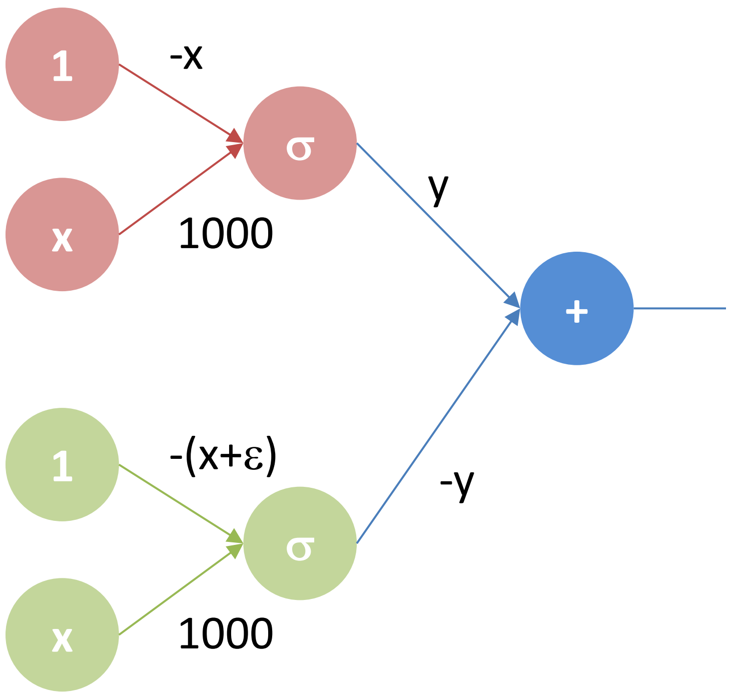

1
-400
σ
x
1000
0.8
+
1
-600
σ
-0.8
x
1000

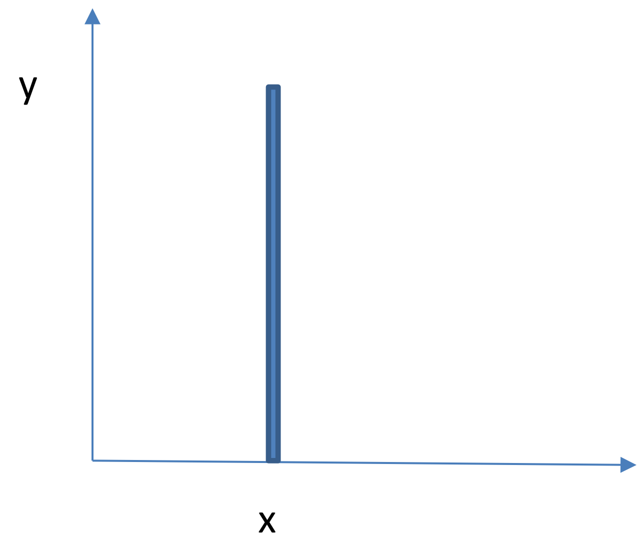

1
-400
σ
x
1000
0.8
+
1
-600
σ
x
1000
-0.8
2
0.8
0
0.4
0.6
1


1
x
-400
1000
σ
-1.2
1
x
-600
1000
σ
1.2
1
x
-700
1000
σ
0.3
1
x
-900
1000
σ
-0.3
+
2
0
0.4
0.6
0.7
0.9
1
2
0
0.7
0.9
1

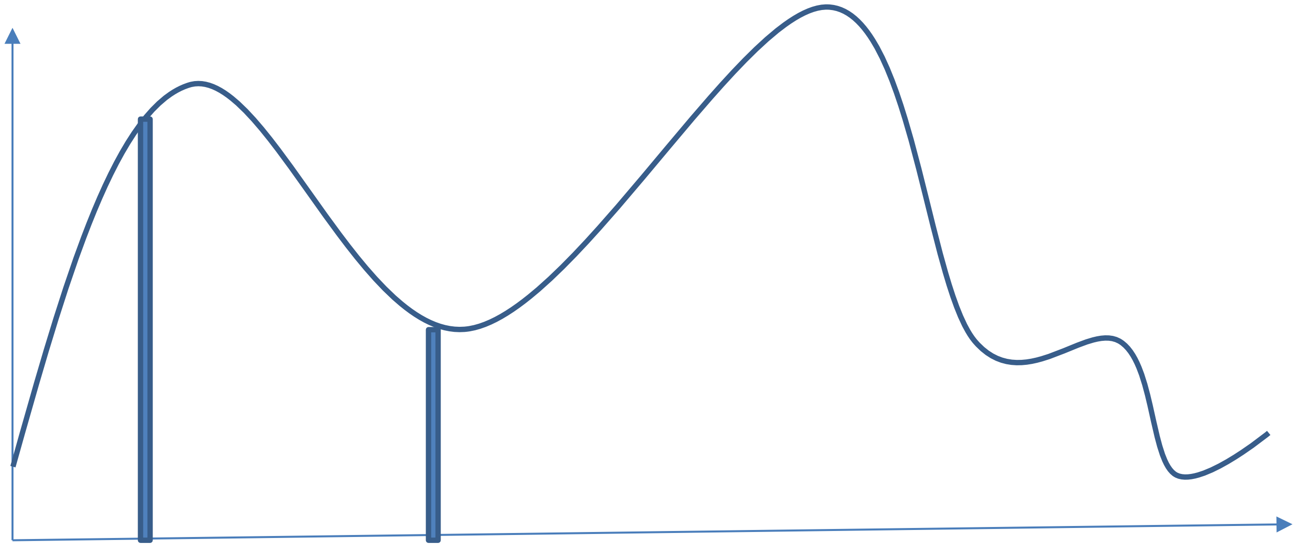

1
x
σ
-400
1000
-1.2
1
x
σ
-600
1000
1.2
1
x
σ
-700
1000
0.3
1
x
σ
-900
1000
-0.3
+


2
0
0.7
0.9
1


1
-x
σ
x
1000
y
+
1
-(x+ε)
σ
-y
x
1000


y
x

Any function



NN with sufficient complexity can approximate any measurable function to any desired degree of accuracy. Thus, it is the basis to model numerous advanced applications