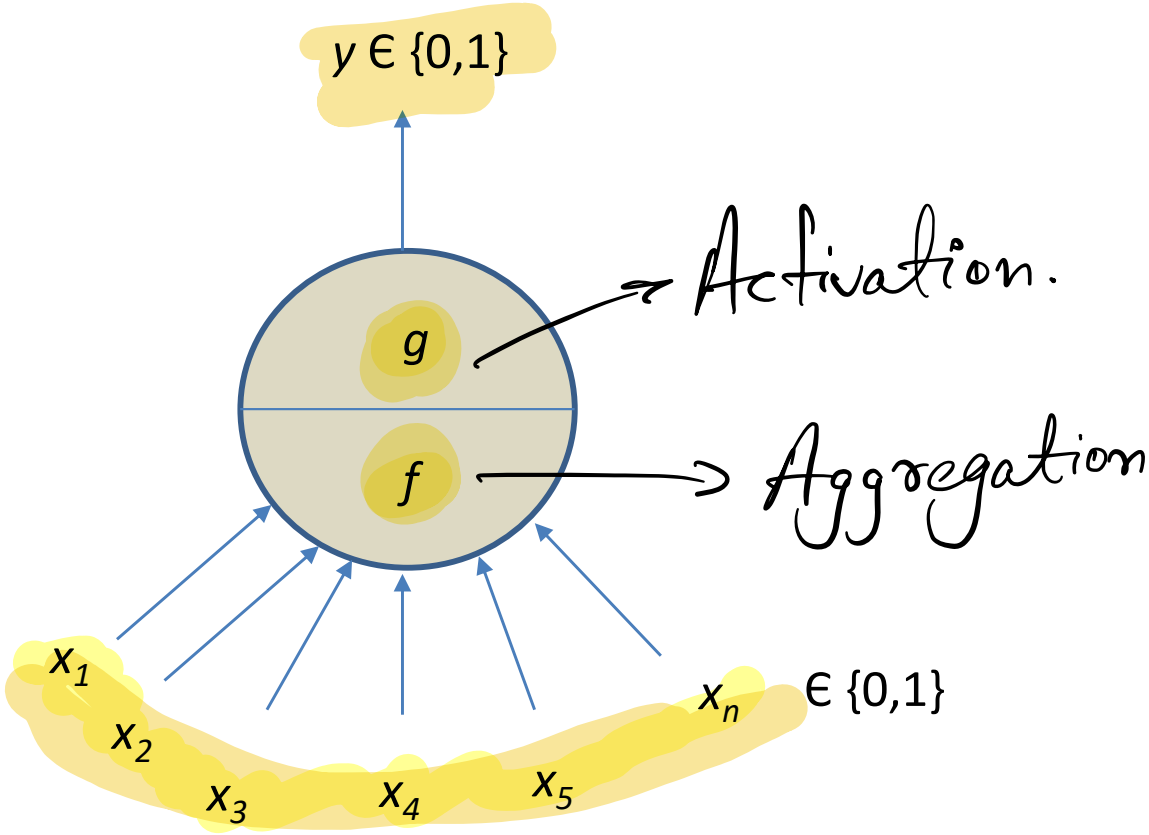
Deep Learning

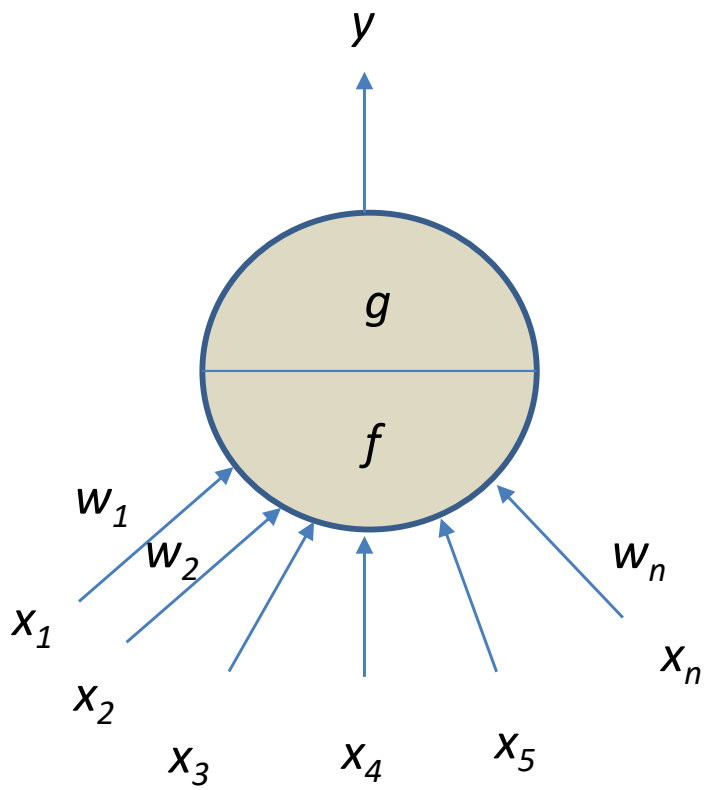
McCulloch Pitts (MP) Neuron

- 1943: McCulloch, neuroscientist, and Pitts, logician, proposed a simplified model of neuron



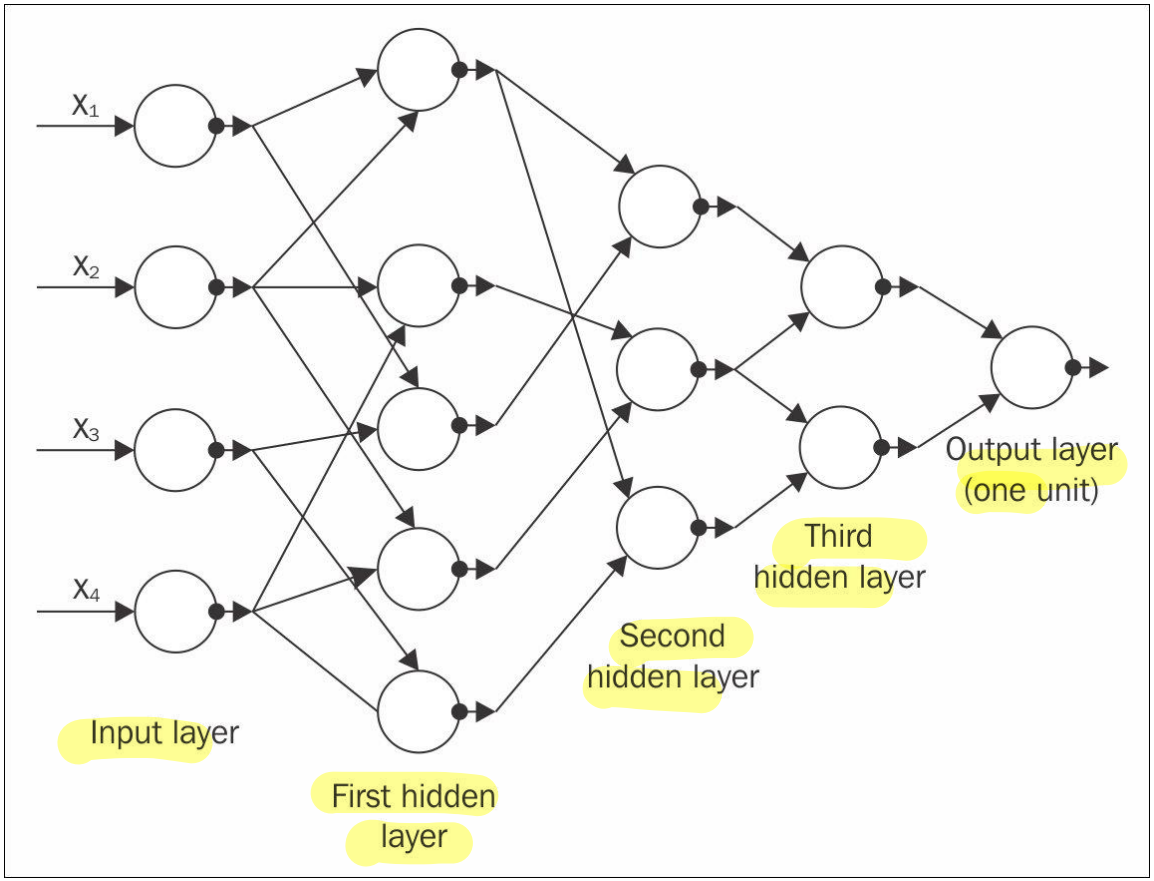
Perceptron

- 1957-58: Frank Rosenblatt, "….perceptron may eventually be able to learn, make decisions, and translate languages.."



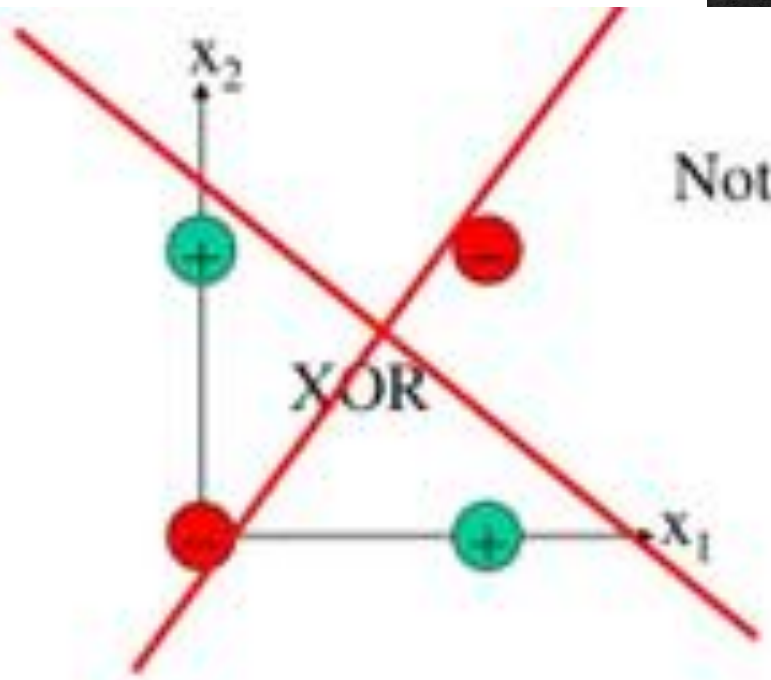
1st Gen Multi-Layer Perceptron

- 1965-68: Ivakhnenko et al.



Perceptron Limitations

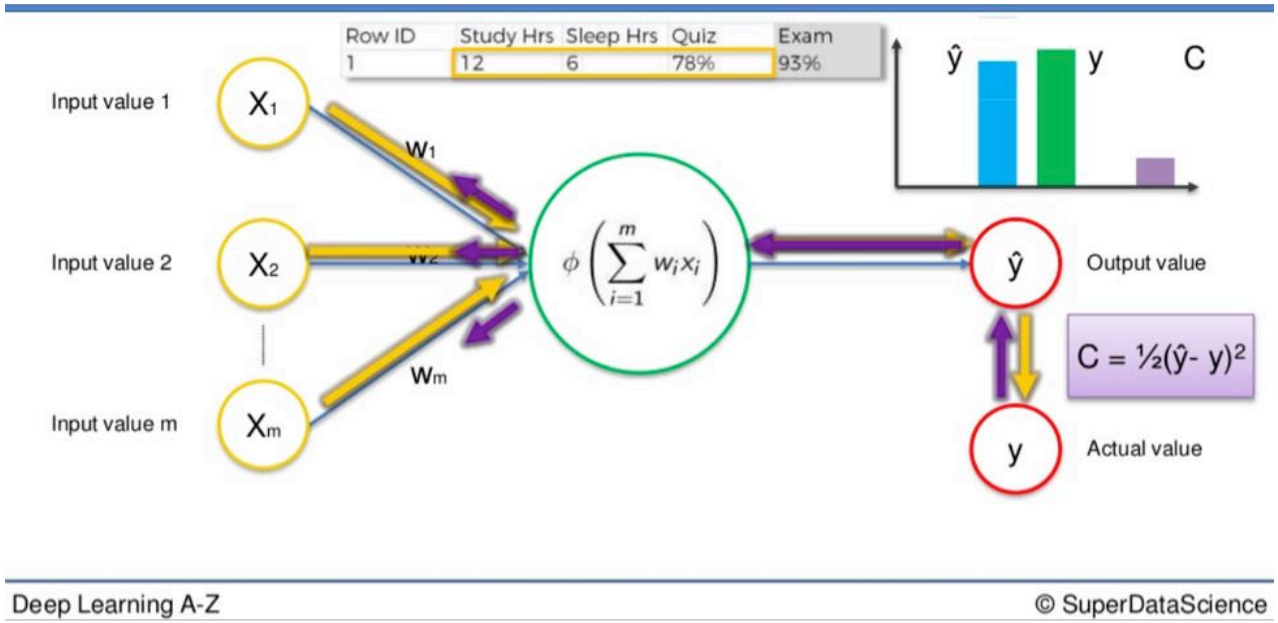
- 1969: “Perceptrons” by Minsky and Papert



Not linearly separable

Backpropagation

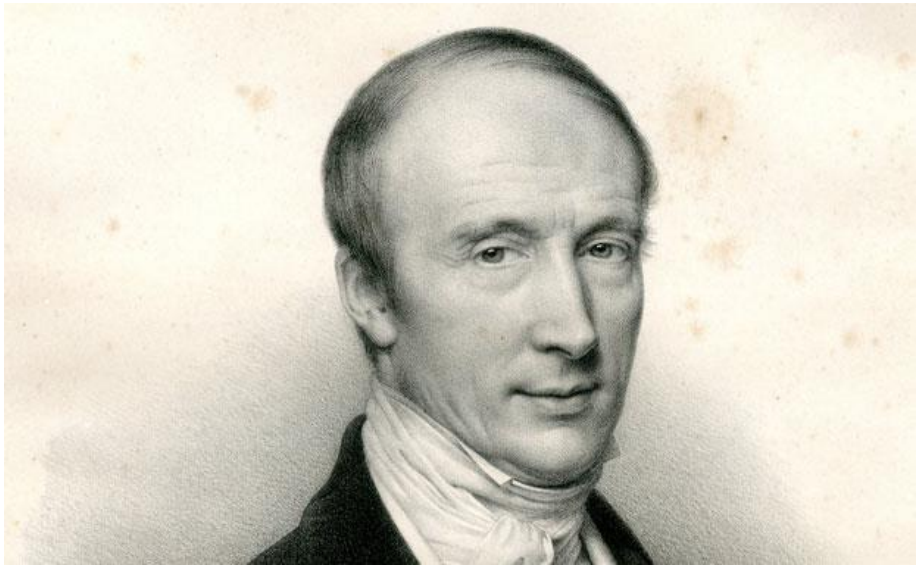
- 1960-70s: discovered/rediscovered
- 1982: Werbos used it in context of ANN
- 1986: Rumelhart et al. popularized it



https://www.superdatascience.com/blogs/artificial-neural-networks-backpropagation

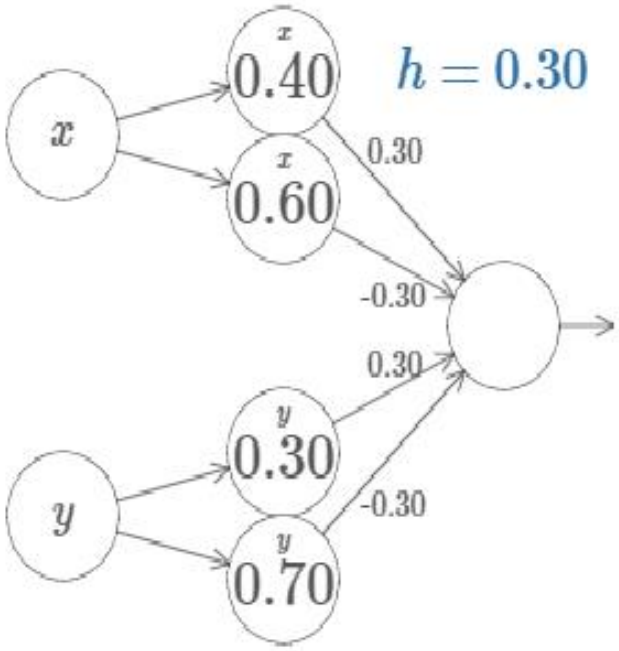
Gradient Descent

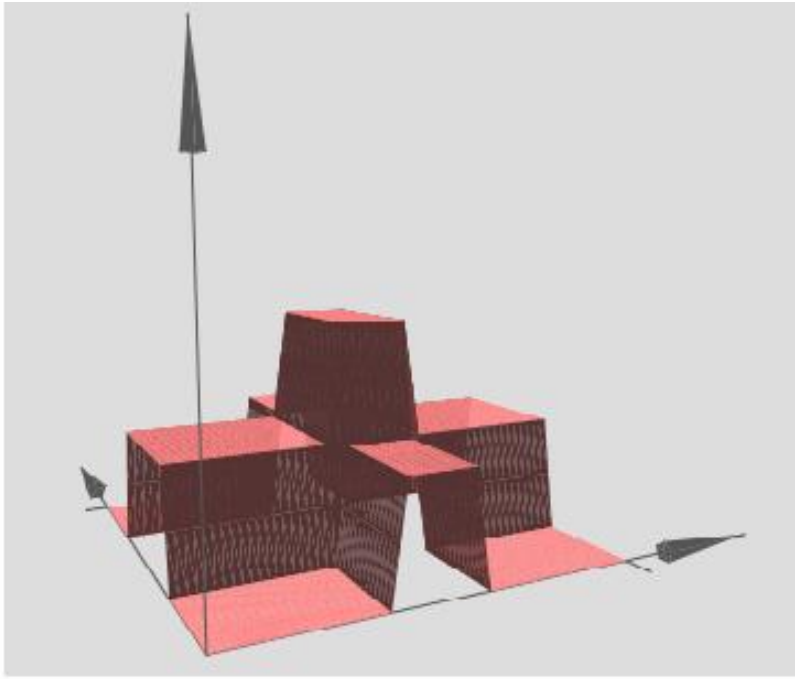
- Discovered by Cauchy in 1847, motivated by need to compute orbit of heavenly bodies
  - Used for backpropagation in neural networks



Universal Approximation Theorem

- 1989-90: A multi-layered network of neurons with a single hidden layer can be used to approximate any continuous function to any desired precision

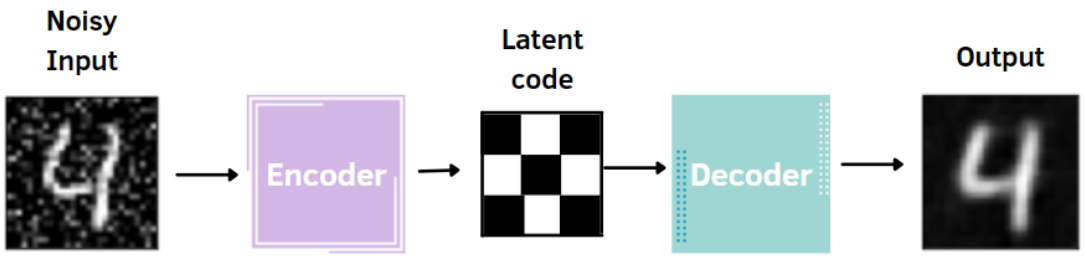




Unsupervised Pre-training

- 2006: Hinton and Salakhutdinov propose method of initializing weights that allows autoencoders to learn low-dimensional representation of data

- 2006-2009: Further explorations



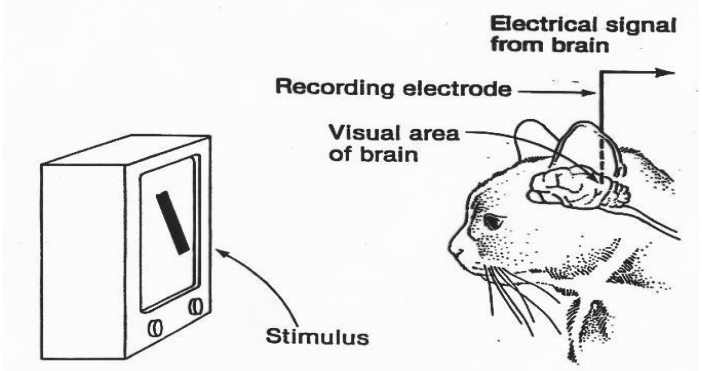
Success Stories

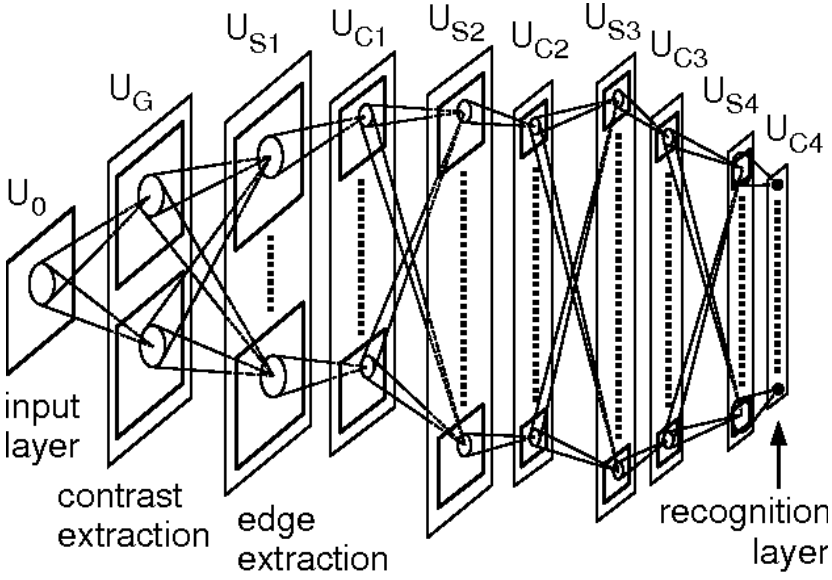
- 2009: Handwriting recognition (Graves et al.)
- 2010: Speech recognition (Dahl et al.)
- 2010: Handwritten digits recognition (MNIST), Ciresan et al.
- 2011: Visual Pattern Recognition (Ciresan et al., IJCNN Traffic Sign Recognition Competition)
- 2012-2016: Visual Recognition Challenges (ImageNet)

Cats to CNN

- 1959: Hubel and Wiesel experiment
  - A neuron fires only in response to a visual stimuli in a specific region in visual space

- 1980: Neocognitron
  - Used for handwritten character and pattern recognition (Fukushima et al.)





CNN

- 1989: handwritten digit recognition using back propagation over CNN (LeCun et al.)

- LeNet-5 introduced MNIST

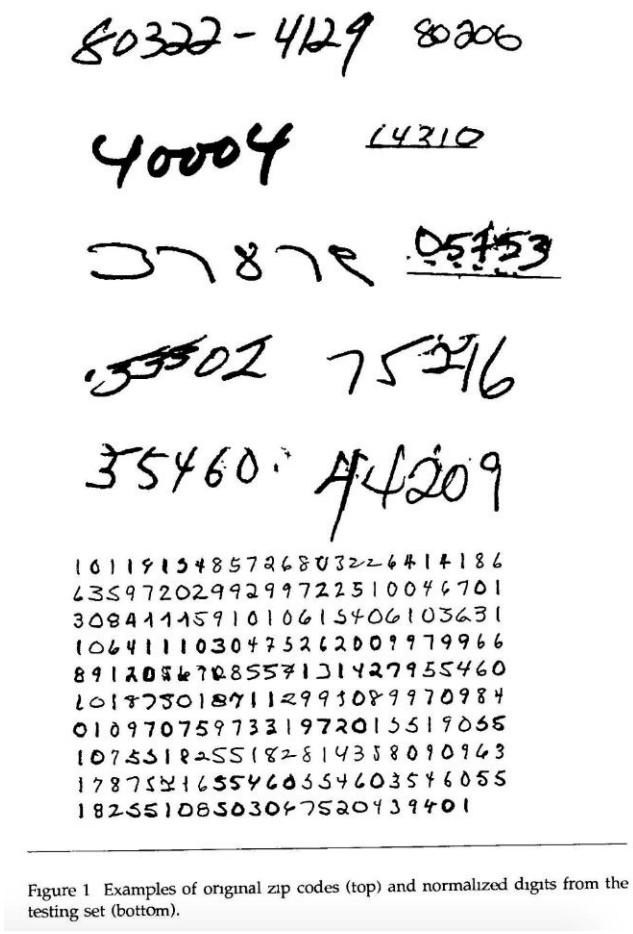


Figure 1 Examples of original zip codes (top) and normalized digits from the testing set (bottom).

Sequences

- Time series, speech, music, text, video
  - Each unit in sequence interacts with others
  - Models needed to capture interaction
- 1982: Hopfield – Content addressable memory systems for storing and retrieving patterns
- 1986: Jordan network – Output of each time step fed to next time step allowing interactions between time steps
- 1990: Elman network – Hidden state of each time step fed to next time step allowing interactions

Sequences

- 1991-1994: Drawbacks of training RNNs (Bengio et al.)
- 1997: LSTMs – can solve long time lag tasks
- 2014: Attention mechanism – sequence to sequence learning
- RL for Attention: Schmidhuber and Huber – RNNs with reinforcement learning

Better Optimization

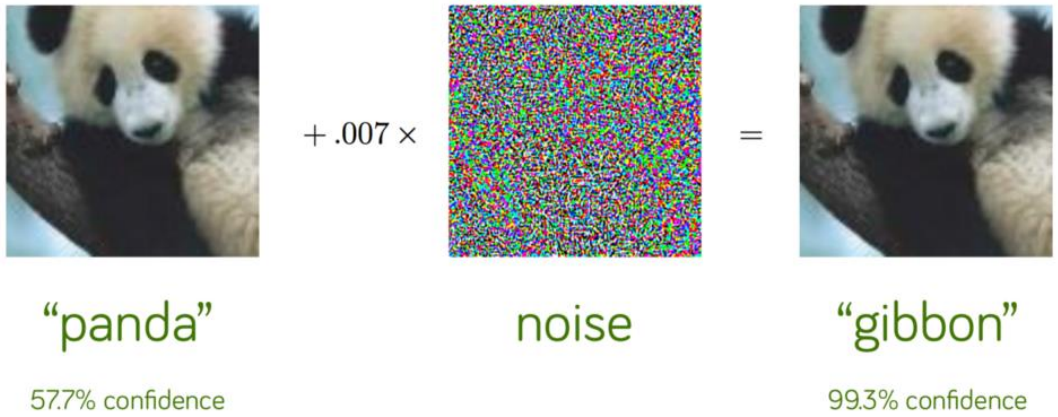
- 2011: Adagrad
- 2012: RMSProp
- 2015: Adam, Batch Normalization
- 2018: Beyond Adam
- Faster convergence, better accuracies

Beating Humans at Playing Games

- 2015: Atari Games
- Alpha Go, GO
- Poker: Deep Stack
- Defense of the Ancients

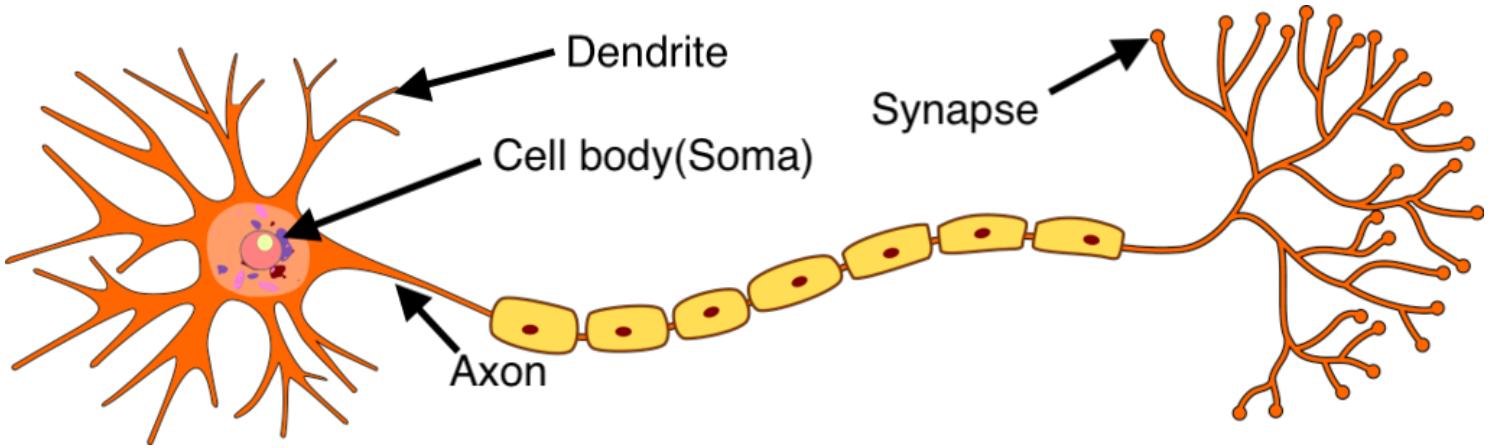
Paradox of Deep Learning

- DL work well despite:
  - High capacity (susceptible to overfitting)
  - Numerical instability (vanishing/exploding gradients)
  - Sharp minima (leads to overfitting)
  - Non-robust

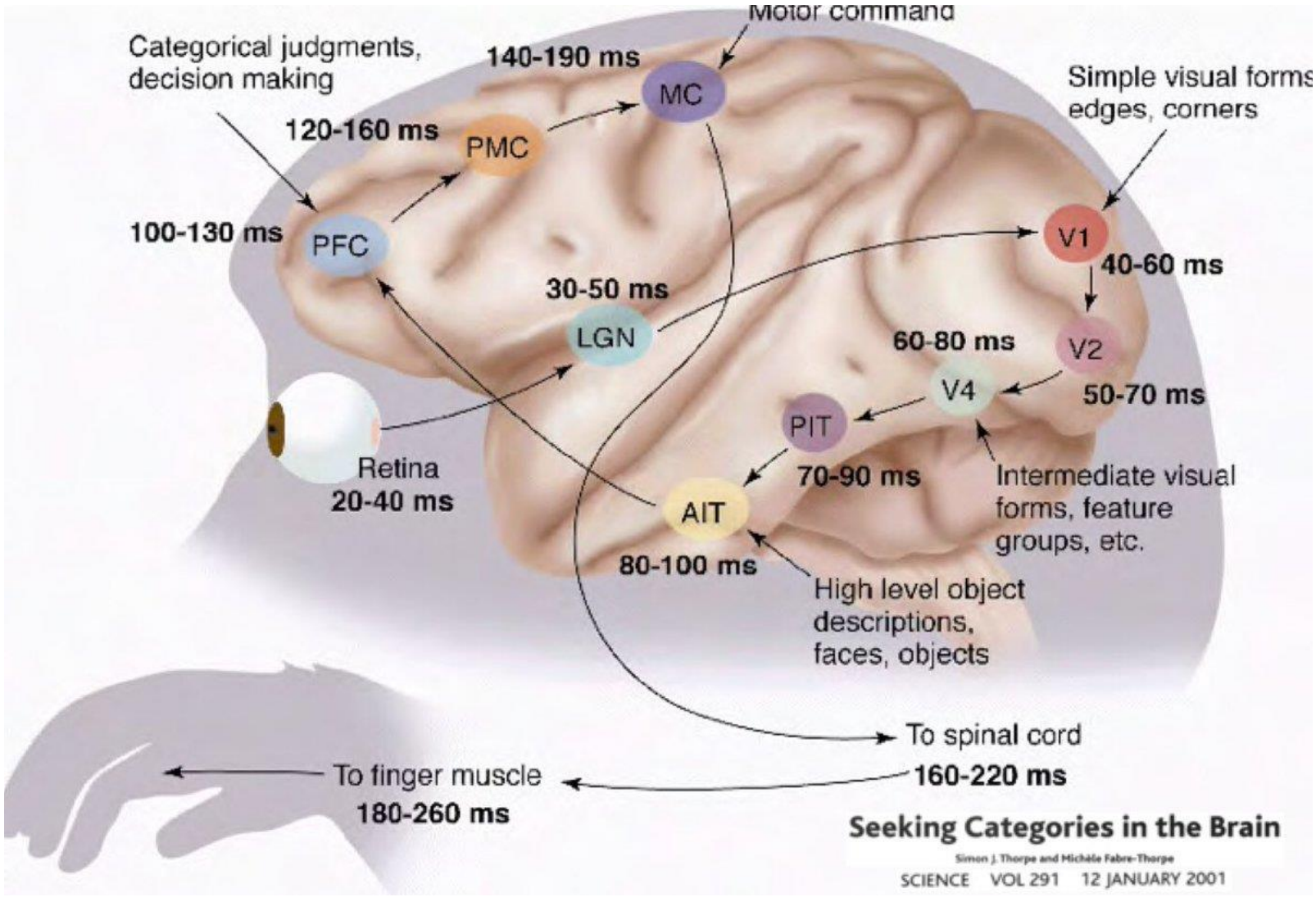


Perceptrons

Biological Neuron

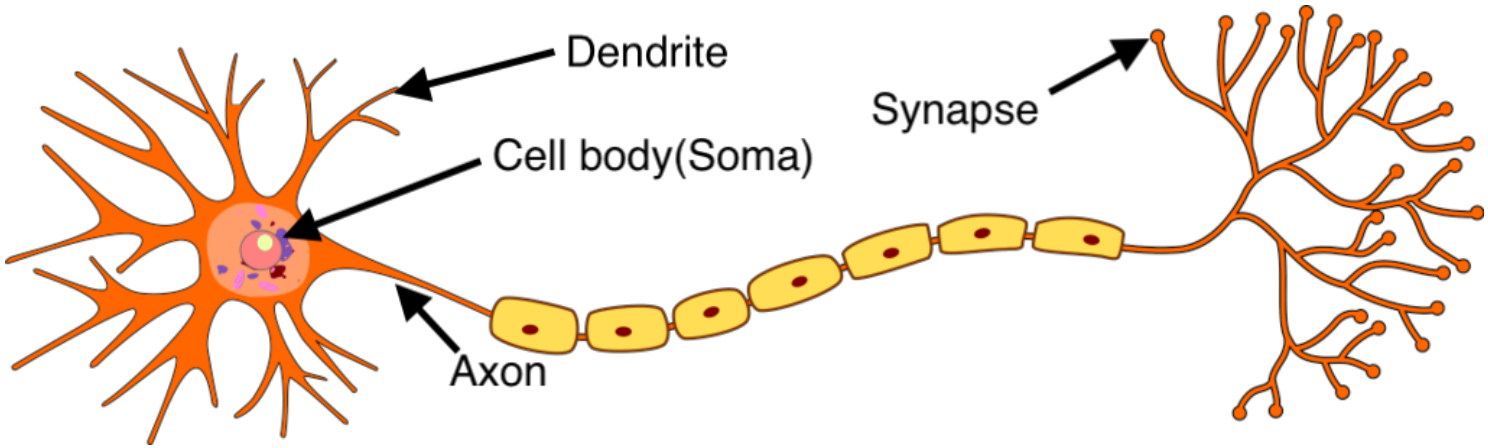


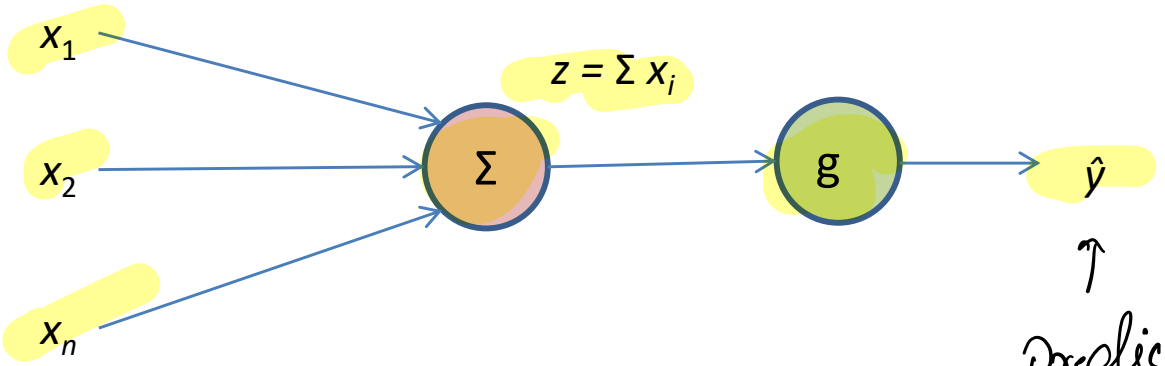
- Massively parallel interconnected network of neurons
- Sense organs relay information to lowest layer of neurons
  - Fired neurons relay information to other connected neurons
  - Division of work – respond to certain stimulus
- Average human brain has around 10^{11} neurons



Picture by Simon Thorpe

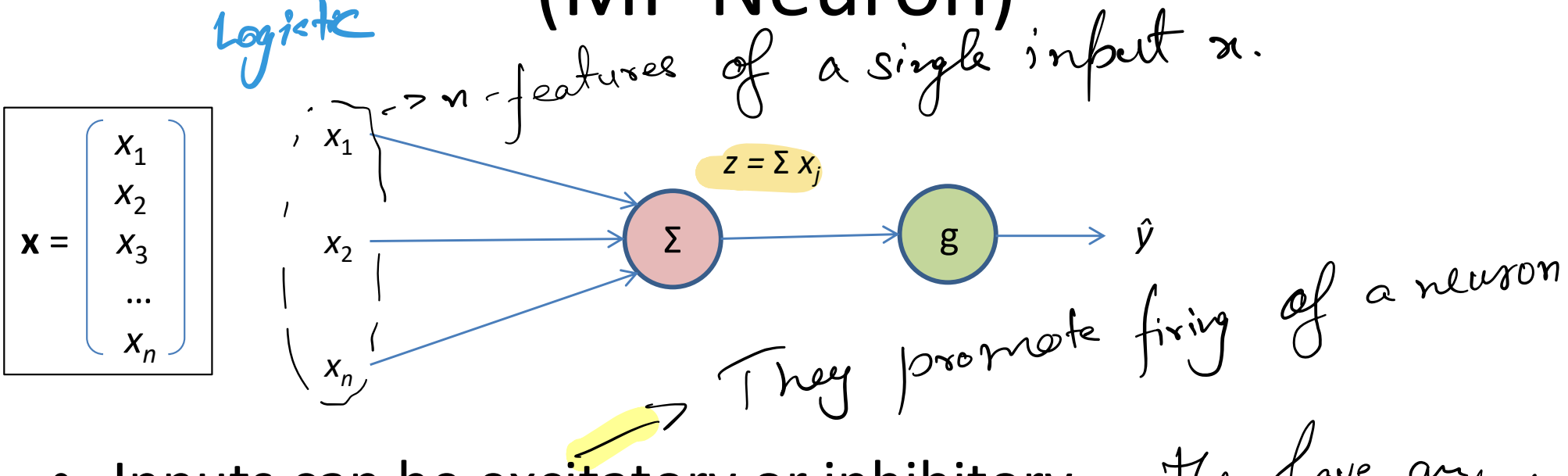
Basic Building Block





predicted output.

Basic Building Block (MP Neuron)



- Inputs can be excitatory or inhibitory → they have an opposite effect.
- Assuming, **x** is an n-dimensional vector and $x_i \in \{0,1\}$

$$\hat{y} = g(z) = g(\Sigma x_i) \qquad \forall\, i = 1 \dots n$$

$\hat{y} = 1$ if $z >= \theta$

$= 0$ if $z < \theta$

How to handle?

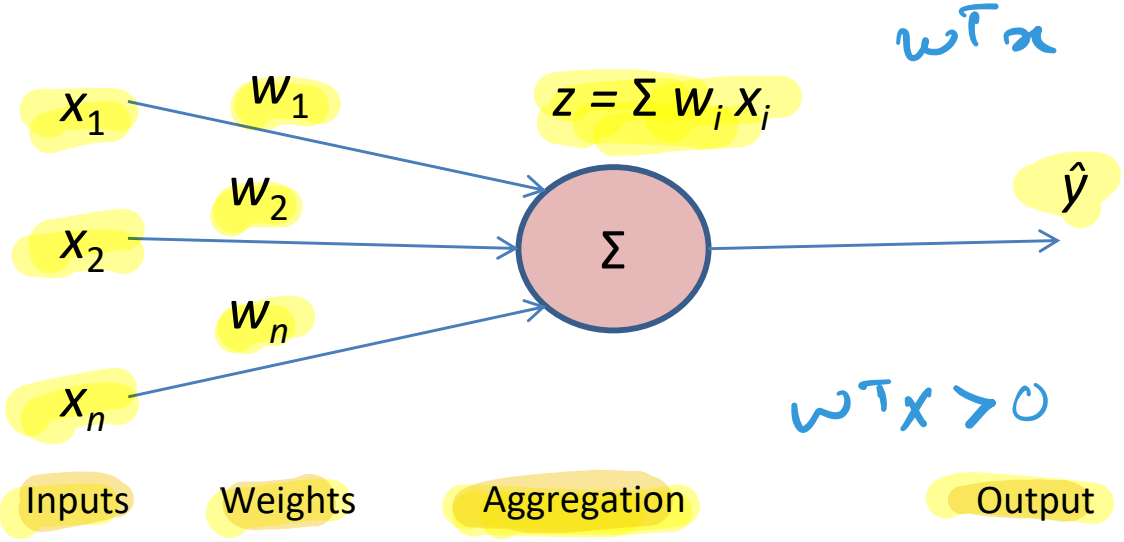
- What about non-Boolean, real inputs?
- Always need to hard-code threshold?
- Are all inputs equal?? Assign more weights to some inputs?
- Non-linearly separable functions??

Classical Perceptron

- Frank Rosenblatt proposed classic perceptron (1958)
  - Numerical weights for inputs – mechanism for learning weights
  - Inputs not limited to Boolean values
- Refined by Minsky and Papert – referred to as perceptron model

Classical Perceptron

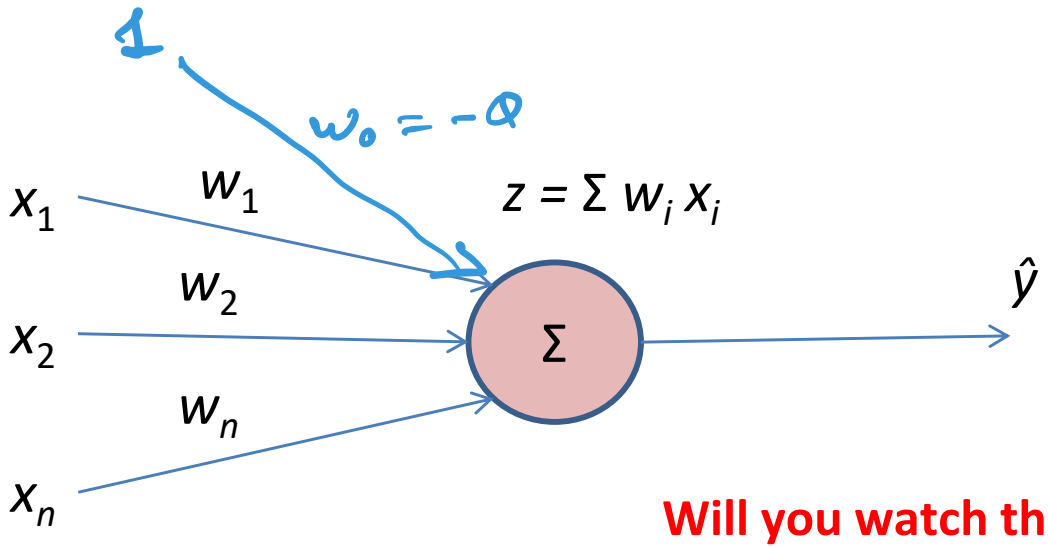
$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_n \end{pmatrix} \qquad \mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ \dots \\ w_n \end{pmatrix}$$



$$\hat{y} = (\Sigma\, w_i\, x_i) = \mathbf{w}^\top \mathbf{x}$$

$$\forall\, i = 1 \dots n$$

Perceptron – Basic Building Block



Will you watch this movie???

$\hat{y} = (\Sigma\, w_i\, x_i)$ $\quad \forall\, i = 1 \ldots n$

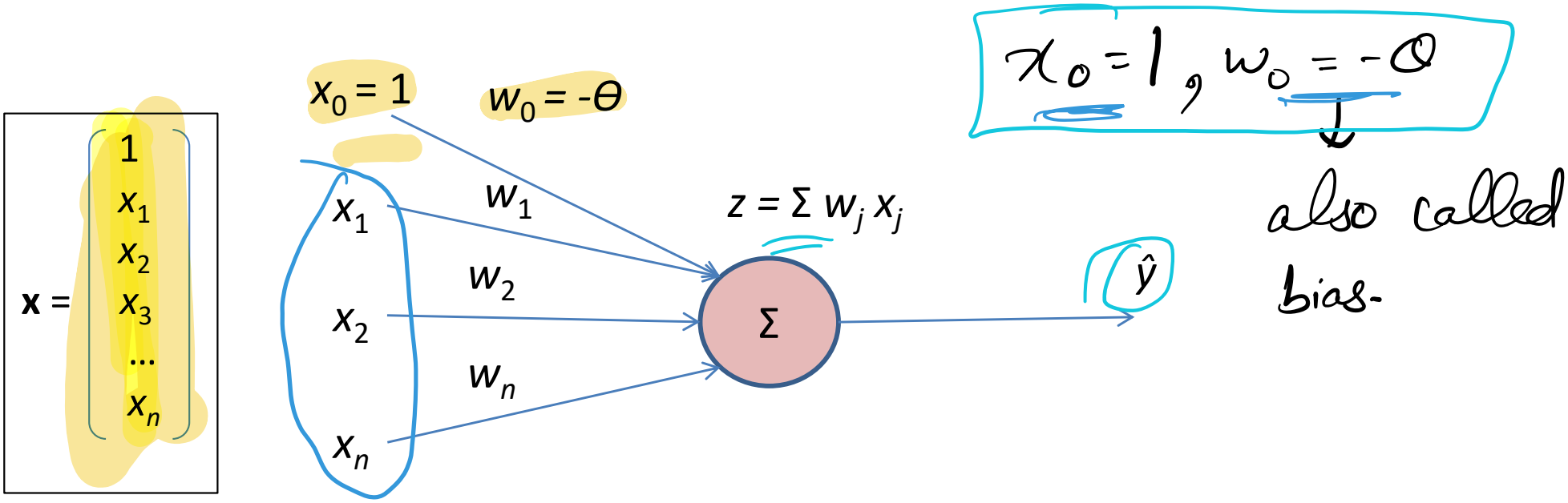
$\hat{y} = 1 \quad if\ \Sigma\, w_i\, x_i >= \theta$

$= 0 \quad if\ \Sigma\, w_i\, x_i < \theta$

$\hat{y} = 1 \quad if\ \Sigma\, w_i\, x_i - \theta >= 0$

$= 0 \quad if\ \Sigma\, w_i\, x_i - \theta < 0$

Perceptron – Basic Building Block



$x_0 = 1$, $w_0 = -\Theta$

also called bias.

$\hat{y} = 1$ *if* $\Sigma\, w_i x_i - \Theta >= 0$

$= 0$ *if* $\Sigma\, w_i x_i - \Theta < 0$

$\forall\, i = 1 \ldots n$

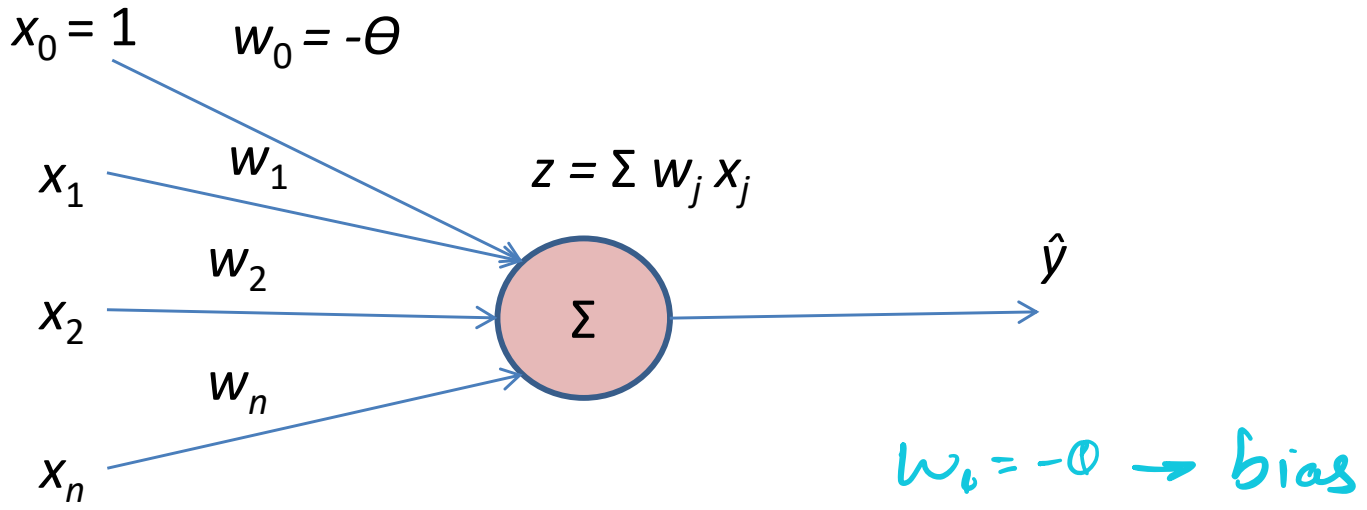
$\hat{y} = 1$ *if* $\Sigma\, w_i x_i >= 0$

$= 0$ *if* $\Sigma\, w_i x_i < 0$

$\forall\, i = 0 \ldots n$, $w_0 = -\Theta$, $x_0 = 1$

w_0 / b: bias, represents the prior or prejudice

Perceptron – Basic Building Block



$\hat{y} = 1 \quad if\ \Sigma\, w_i x_i >= 0$

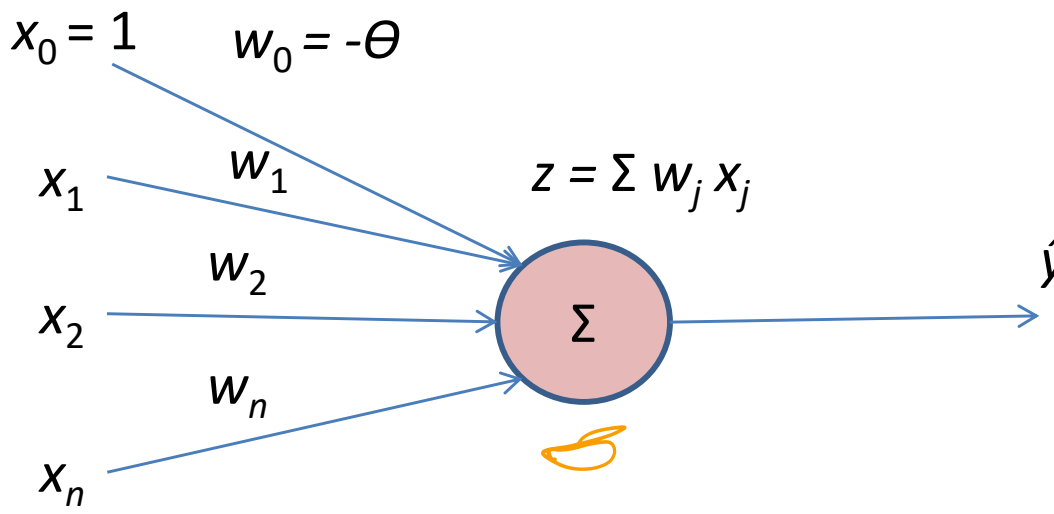
$= 0 \quad if\ \Sigma\, w_i x_i < 0$

$\forall\ i = 0 \ldots n,\ w_0 = -\Theta,\ x_0 = 1$

w_0 : bias

- A movie buff may have a very low threshold and may watch any movie [$\Theta = 0$]
- A selective movie watcher may watch only a thriller, starring Matt Damon and directed by Nolan [$\Theta = 3$]

Perceptron – Basic Building Block



$\hat{y} = 1 \quad if\ \Sigma\, w_i x_i >= 0$

$= 0 \quad if\ \Sigma\, w_i x_i < 0$

$\forall\ i = 0 \ldots n,\ w_0 = -\Theta,\ x_0 = 1$

Linearly separable decision boundary: $\mathbf{w^T x = 0}$

Perceptron separates input space into two, but:

- Weights (and threshold) can be learned
- Inputs are real-valued

OR function with Perceptron

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

$w_0 < 0$

$w_1 + w_0 \geq 0$

$w_1 + w_2 + w_0 \geq 0$

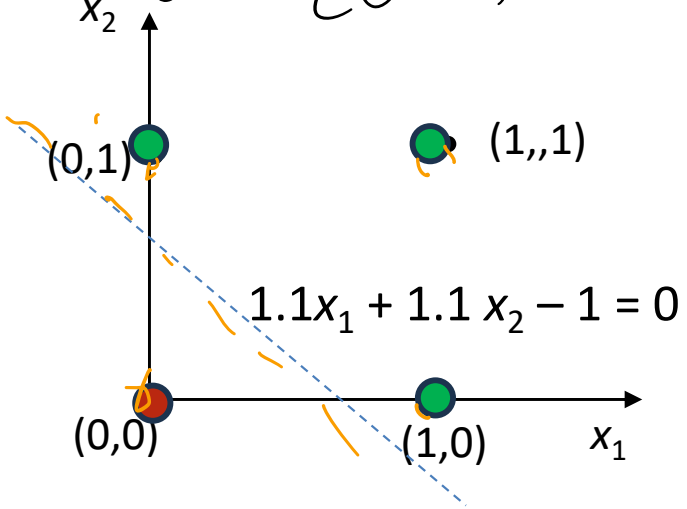
we came to this by seeing the actual classification

$w_0 + w_1.0 + w_2.0 < 0 \Rightarrow w_0 < 0$

$w_0 + w_1.0 + w_2.1 \geq 0 \Rightarrow w_2 > -w_0$

$w_0 + w_1.1 + w_2.0 \geq 0 \Rightarrow w_1 > -w_0$

$w_0 + w_1.1 + w_2.1 \geq 0 \Rightarrow w_1 + w_2 > -w_0$



One possible solution to set of inequalities:

$w_0 = -1$, $w_1 = 1.1$, $w_2 = 1.1$

$1.1x_1 + 1.1\,x_2 - 1 = 0$

OR function

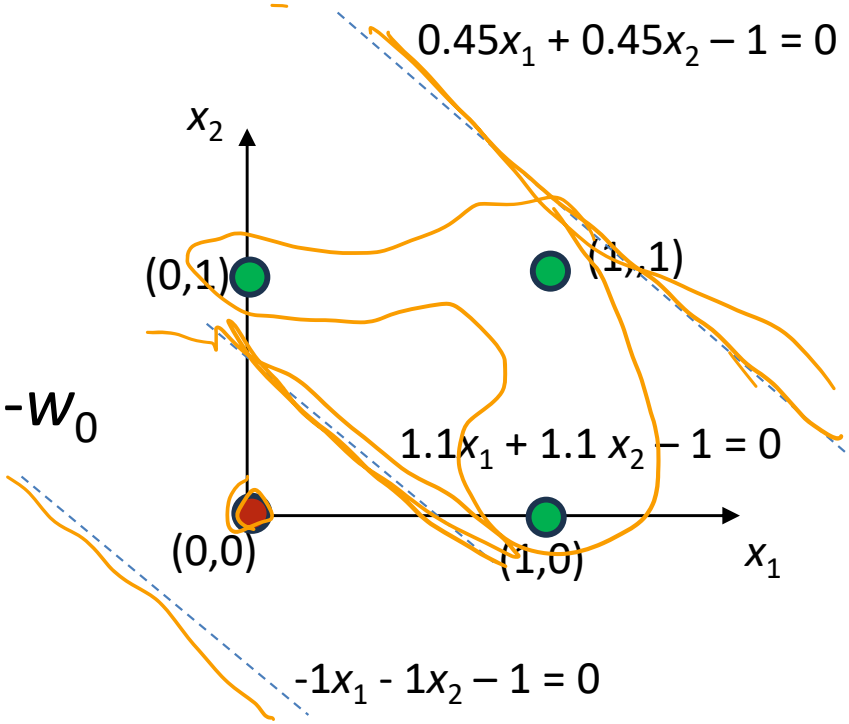
$w_0 + w_1.0 + w_2.0 < 0 \Rightarrow w_0 < 0$

$w_0 + w_1.0 + w_2.1 \geq 0 \Rightarrow w_2 > -w_0$

$w_0 + w_1.1 + w_2.0 \geq 0 \Rightarrow w_1 > -w_0$

$w_0 + w_1.1 + w_2.1 \geq 0 \Rightarrow w_1 + w_2 > -w_0$

Fix threshold: $w_0 = -1$



Try different values of w_1 , w_2

Ex.,$w_1 = -1$, $w_2 = -1$ $\Rightarrow$ one error

Ex.,$w_1 = 0.45$, $w_2 = 0.45$ $\Rightarrow$ 3 errors

Interested in those values of w_0 , w_1 , w_2 , which gives zero error

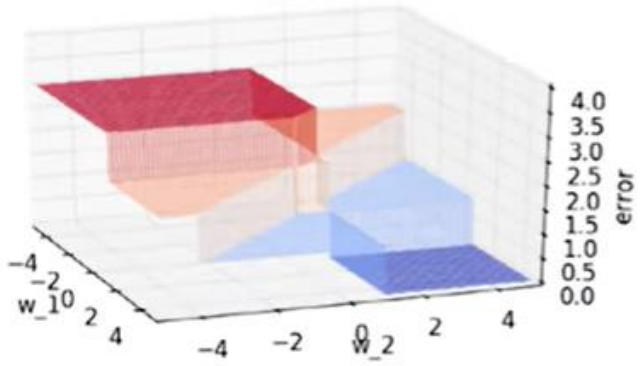
Error Surfaces

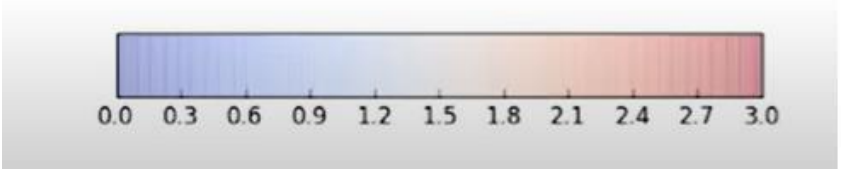
Fix threshold: $w_0 = -1$, plot error for different values of w_1, w_2

- For given w_0, w_1, w_2 compute - $w_0 + w_1 x_1 + w_2 x_2$ ($\mathbf{w}^T\mathbf{x}$) for all combinations of (x_1, x_2) and note down errors

For the OR function, error occurs

- if (x_1, x_2) = (0,0) but $\mathbf{w}^T\mathbf{x} \geq 0$
- or if (x_1, x_2) ≠ (0,0) but $\mathbf{w}^T\mathbf{x} < 0$





Perceptron Learning Algorithm

- How to handle arbitrary functions: $y = f(x)$ instead of Boolean functions

where,

$\mathbf{x} \in R^n$, $y \in R$

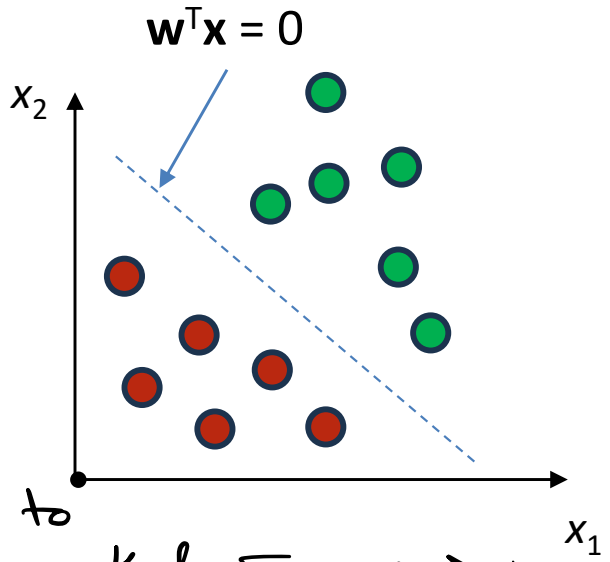
$[x_1, x_2, x_3 \ldots . x_n]^T \Rightarrow y$

Given the *n* parameters, how much oil will be produced??

- *n* parameters – some real, some Boolean
- Assuming data is linearly separable, want perceptron to learn how to make decision
- Need a network which can approximately represent these functions

Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize $\mathbf{w} = [w_0, w_1, w_2, \ldots w_n]^T$ randomly;
while *!convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P$ *and* $\sum w_i * x_i < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$; → Adding value to weights so that $\sum w_i x_i \geq 0$
    end
    if $\mathbf{x} \in N$ *and* $\sum w_i * x_i \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end



Convergence- not making any more errors on training data or predictions are not changing

Perceptron Learning Algorithm

- Consider vectors **w** and **x**

$$\mathbf{w} = [w_0, w_1, w_2, \ldots w_n]^T$$

$$\mathbf{x} = [1, x_1, x_2, \ldots x_n]^T$$

Perceptron rule:

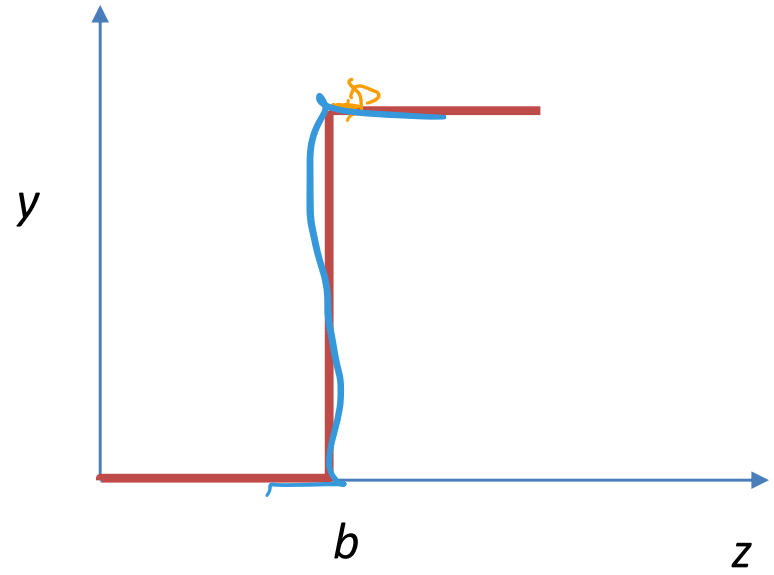
$$\hat{y} = 1 \quad \text{if } \mathbf{w}^T\mathbf{x} \geq 0$$

$$= 0 \quad \text{if } \mathbf{w}^T\mathbf{x} < 0$$

- Find line $\mathbf{w}^T\mathbf{x} = 0$ which divides input space into two
- Every point **x** on this line satisfies $\mathbf{w}^T\mathbf{x} = 0$

Perceptron Logic

- A perceptron will fire if weighted sum of inputs is greater than threshold ($b = -w_0$)



AND function

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

$$X = \begin{bmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}$$

$x_0 w_0 + x_1 w_1 + x_2 w_2$

$w_0 < 0$

$w_0 + w_2 < 0$

$w_0 + w_1 < 0$

$w_0 + w_1 + w_2 \geq 0$

Linearly separable

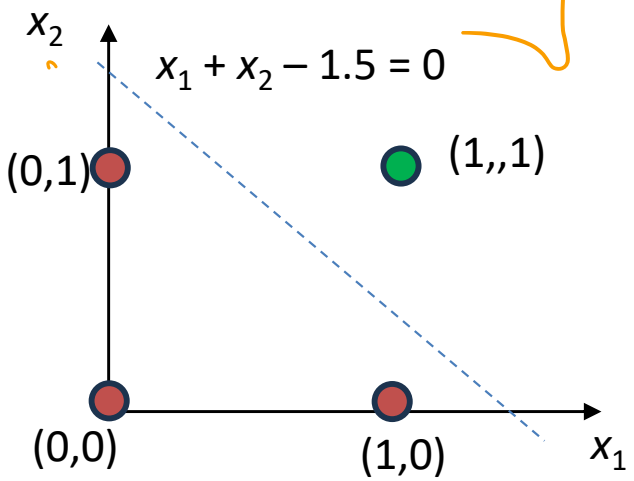
$$\mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \quad \mathbf{w} = \begin{pmatrix} -1.5 \\ 1 \\ 1 \end{pmatrix}$$

i/p inputs

$\mathbf{W}^T = [-1.5 \quad 1 \quad 1]$

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$w^T x$

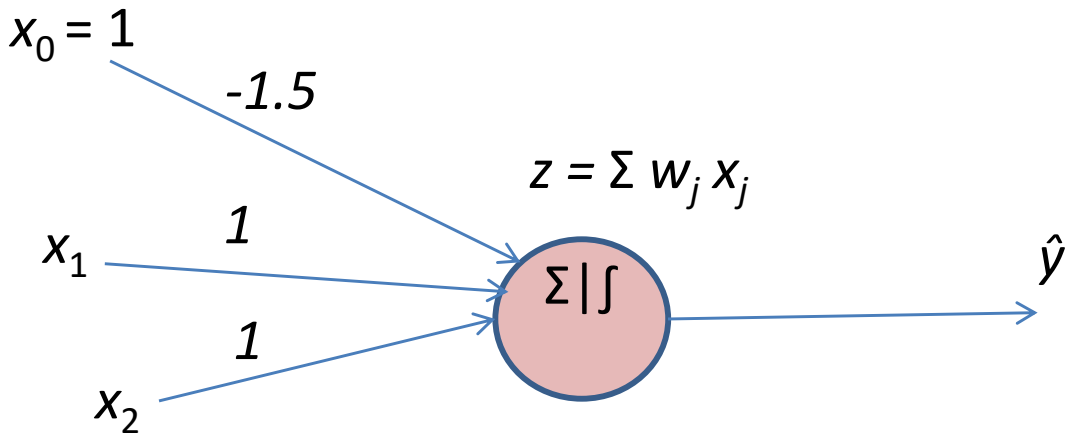


AND function

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}_{3\times 4} \qquad \mathbf{W}^T = \begin{bmatrix} -1.5 & 1 & 1 \end{bmatrix}_{1\times 3}$$

$$\mathbf{W}^T\mathbf{X} = \begin{bmatrix} -1.5 & -0.5 & -0.5 & 0.5 \end{bmatrix}_{[1\times 4]} \xrightarrow{\text{Nonlinear threshold}} \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

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



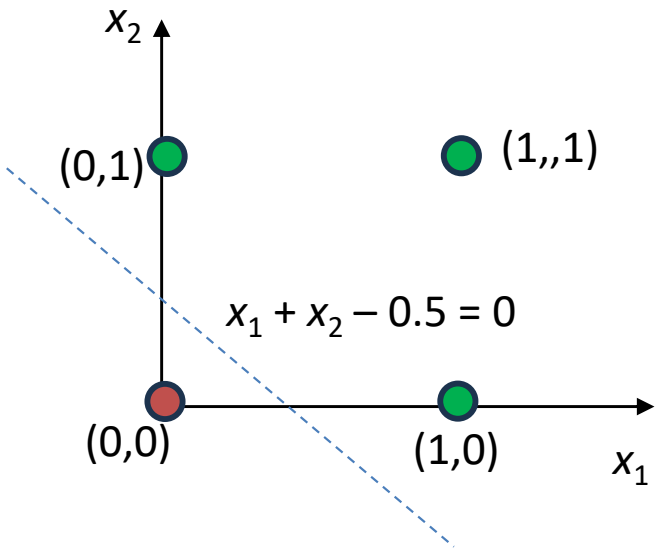
OR function

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

Linearly separable

$$\mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} -0.5 \\ 1 \\ 1 \end{pmatrix}$$

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad \mathbf{W}^T = [-1.5 \quad 1 \quad 1]$$

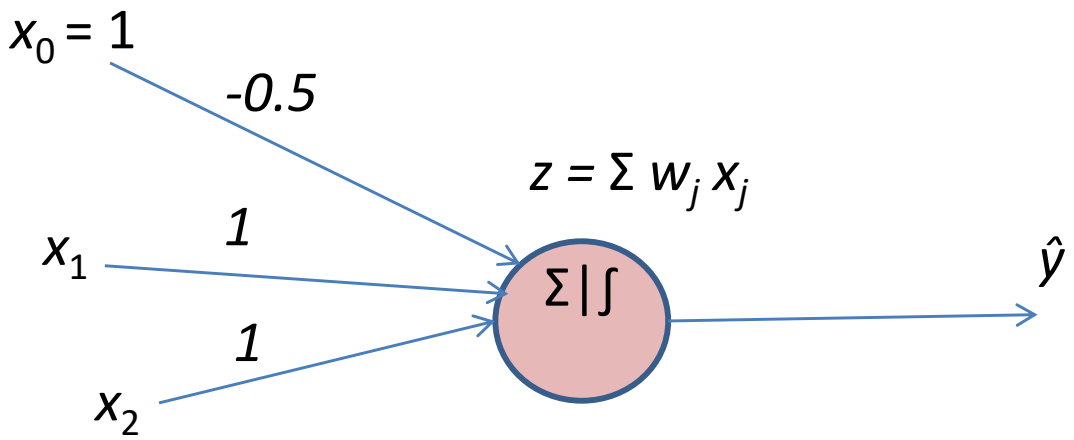


OR function

$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad \mathbf{W}^T = [-1.5 \quad 1 \quad 1]$$

$(x_1 + x_2) \cdot C$

$$\mathbf{W}^T\mathbf{X} = [-0.5 \quad 0.5 \quad 0.5 \quad 0.5] \xrightarrow{\text{Nonlinear threshold}} [0 \quad 1 \quad 1 \quad 1]$$

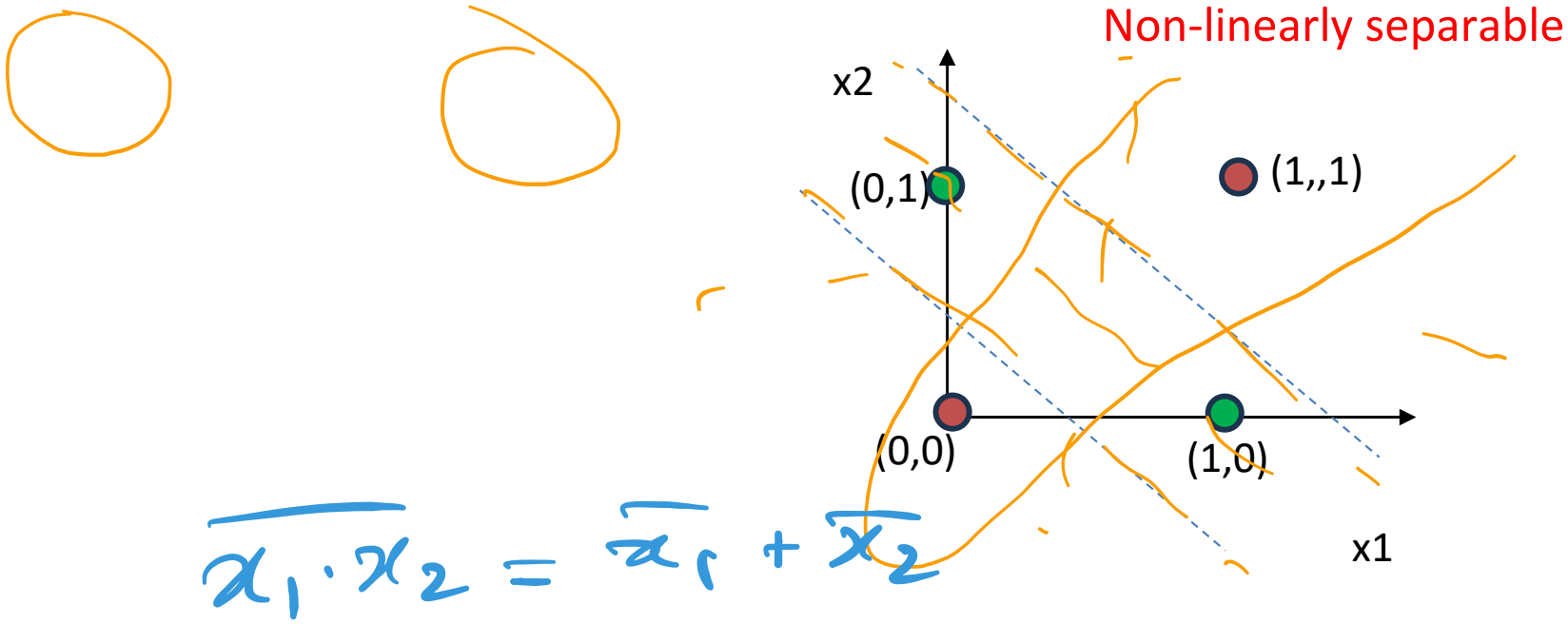


XOR function

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

$x_1 \oplus x_2 =$

$x_1 +$



$\overline{x_1 \cdot x_2} = \overline{x_1} + \overline{x_2}$

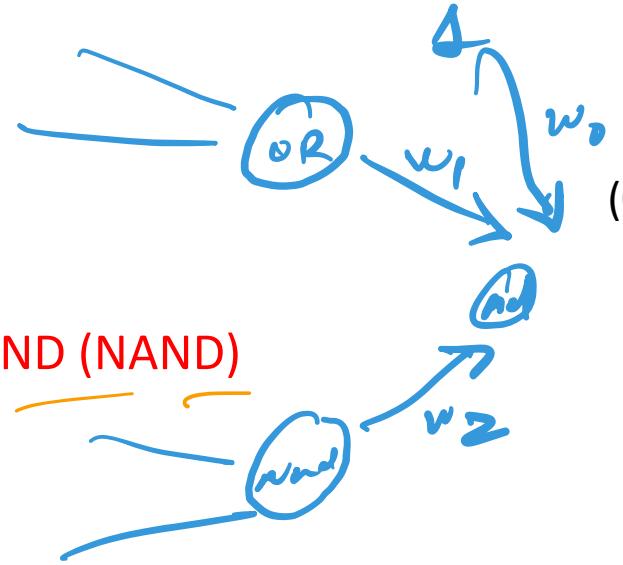
XOR function

x_1 XOR x_2 = $(x_1+x_2).(\overline{x}_1 + \overline{x}_2)$

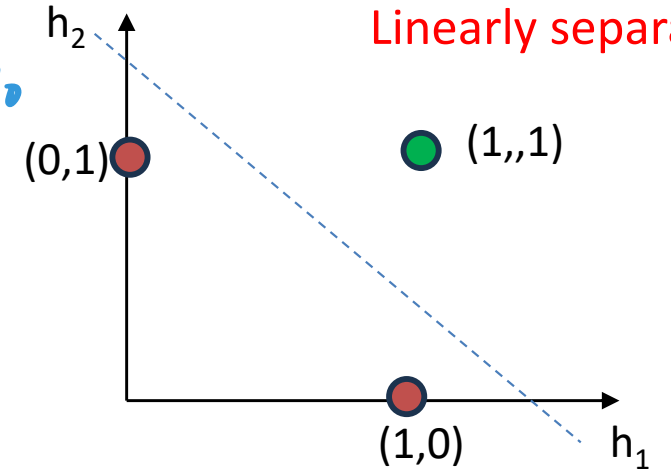
OR operation

NAND operation

x_1	x_2	$h_1 = x_1+x_2$	$h_2 = \overline{x}_1+\overline{x}_2$	y
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

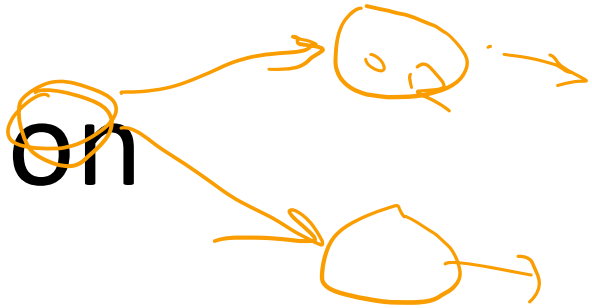


XOR = (OR) AND (NAND)



Linearly separable

XOR function



$$\mathbf{X} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \qquad \mathbf{W}_1 = \begin{pmatrix} -0.5 & 1.5 \\ 1 & -1 \\ 1 & -1 \end{pmatrix}$$

Weights for OR

Weights for NAND operation (complement of AND weights)

$$\mathbf{W}_1^T \mathbf{X} = \begin{pmatrix} -0.5 & 0.5 & 0.5 & 1.5 \\ 1.5 & 0.5 & 0.5 & -0.5 \end{pmatrix}$$

Nonlinear threshold

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \leftarrow \begin{matrix} h_1 \\ h_2 \end{matrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} -1.5 \\ 1 \\ 1 \end{pmatrix}$$

Weights for AND operation

$w_2^T X' = [0\ 1\ 1\ 0]$

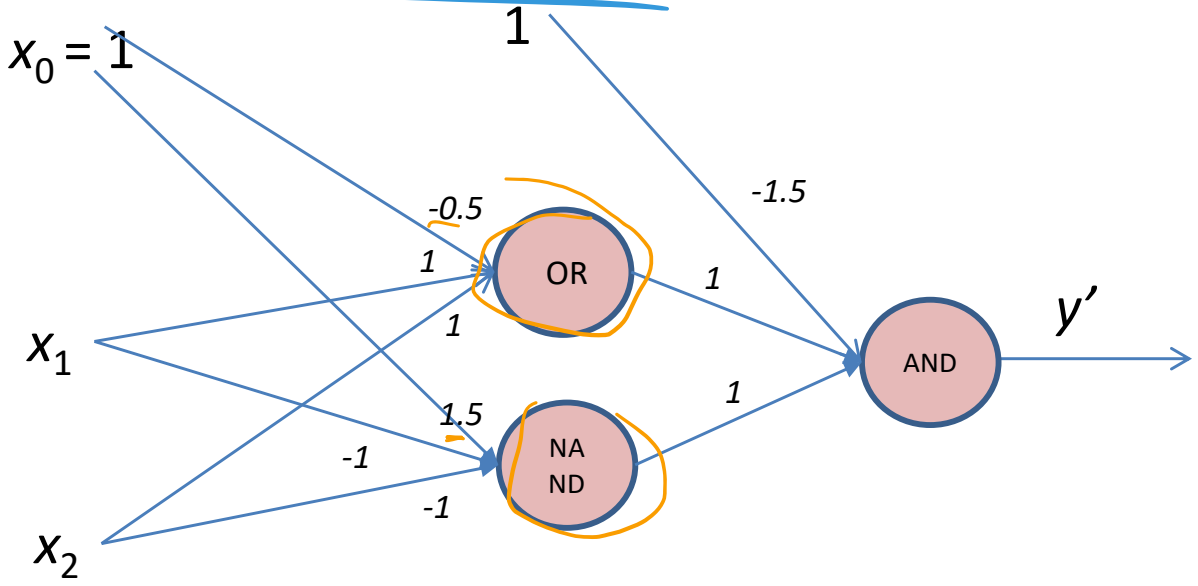
$$\mathbf{h}^T\mathbf{W}_2 = \begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} -1.5 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.5 \\ 0.5 \\ 0.5 \\ -0.5 \end{pmatrix}$$

Nonlinear threshold

$$\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

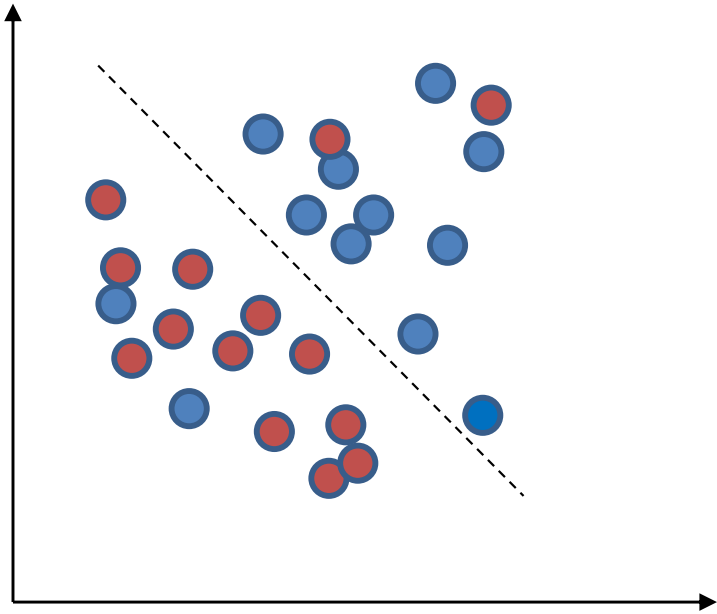
XOR function

- Two layer network
- 1s layer:
  - 2 neurons: one for OR, one for NAND
- 2nd layer:
  - 1 neuron: AND operation

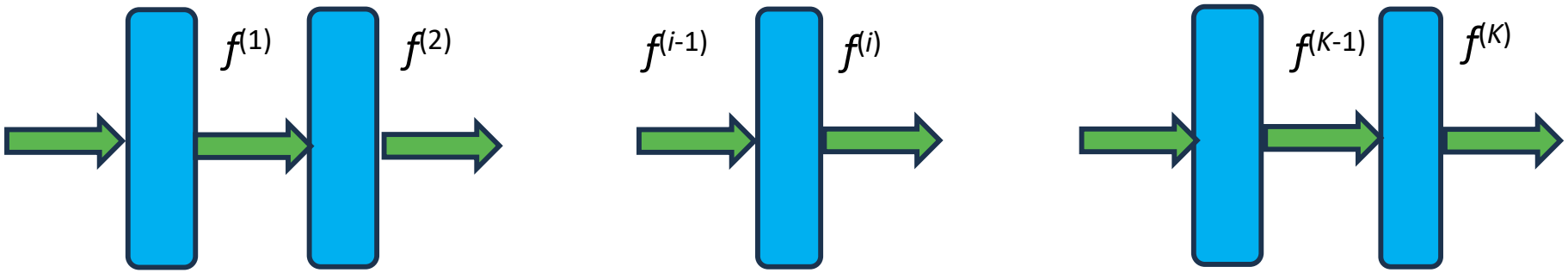


Single perceptron

- Single perceptron cannot deal with data which is not linearly separable
  - Have to be flexible with convergence statement while determining weights…
  - Till almost (say 90%) points are satisfying condition
  - Leads to few errors
  - May not be acceptable in critical real-world applications



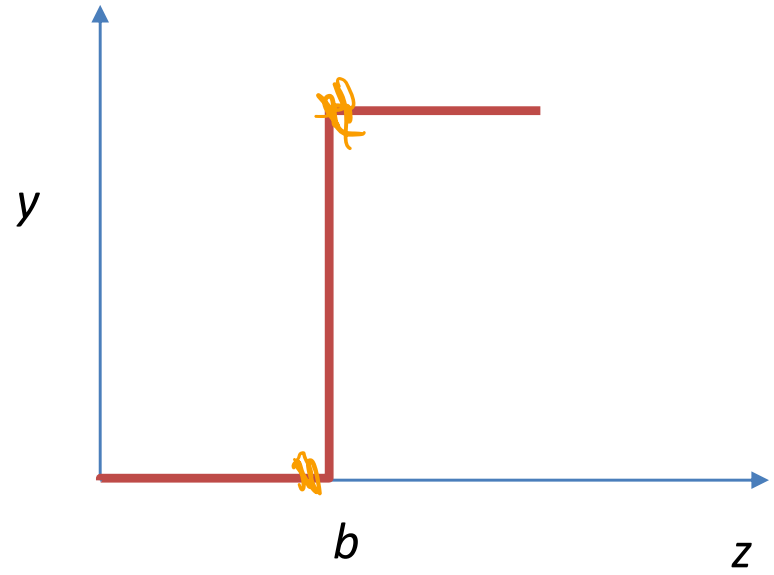
Neural Network Function



$$f^{(K)}(f^{(K-1)}(\ldots f^{(i)}\ldots(f^{(2)}(f^{(1)}(X)))))$$

Perceptron Logic

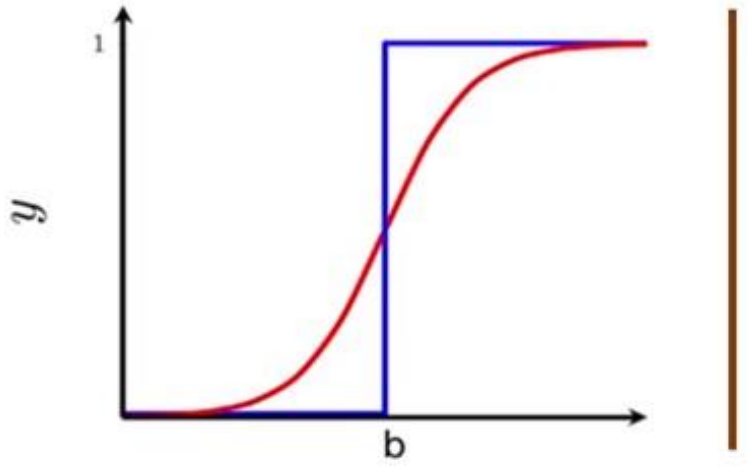
- A perceptron will fire if weighted sum of inputs is greater than threshold ($b = -w_0$)
  - Thresholding logic is harsh



If x=0.51, watch movie, if x=0.49, do not watch??

For real world problems, we need a smoother decision function

Sigmoid neuron



$z = \sum w_i x_i$

$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

$\sigma'(z) = \sigma(1-\sigma)$

If $z = (w^T x + b) \longrightarrow \infty$, then $y = 1$

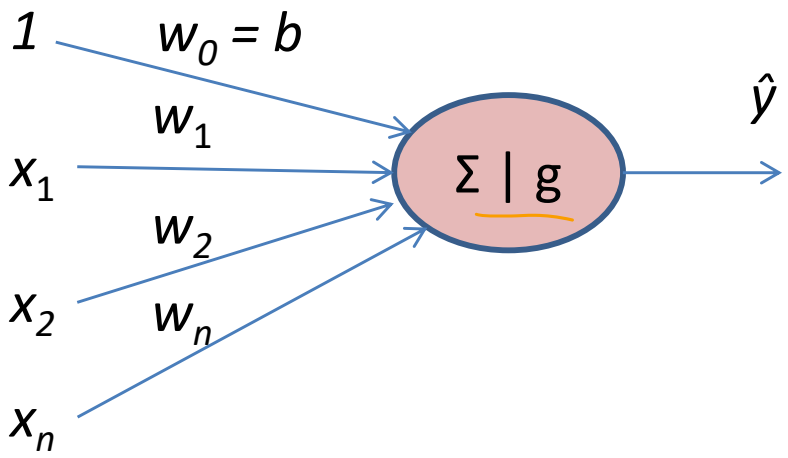
If $z = (w^T x + b) \longrightarrow -\infty$, then $y = 0$

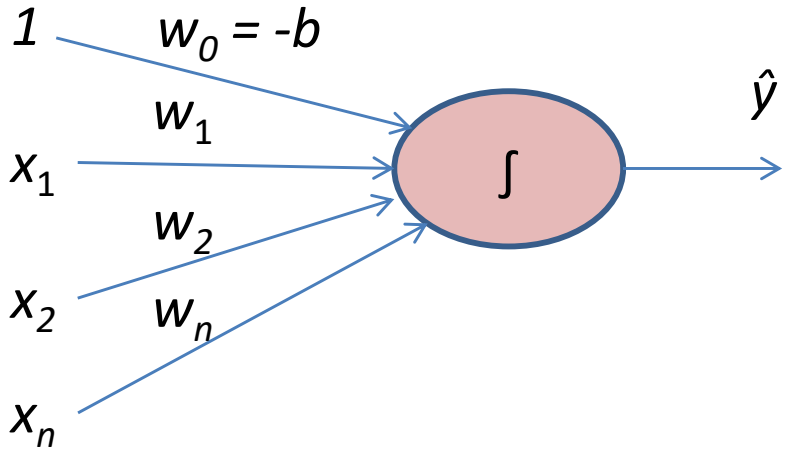
If $z = (w^T x + b) = 0$, then $y = 0.5$

Range of sigmoid function: 0 to 1

Can be interpreted as probability

Perceptron vs Sigmoid





$\hat{y} = 1 \quad if \; \Sigma \, w_i x_i + b \; >= 0$

$= 0 \quad if \; \Sigma \, w_i x_i + b \; < 0$

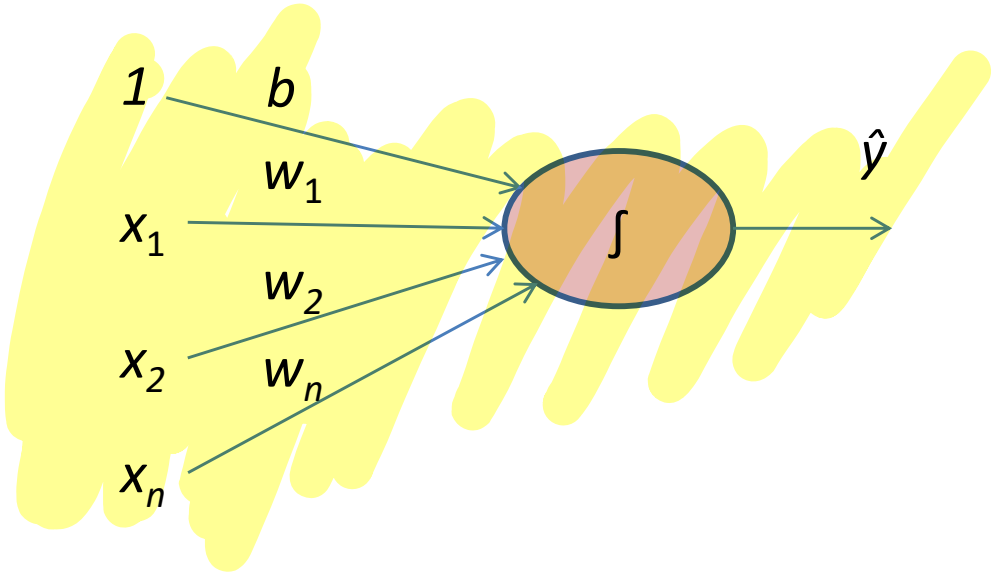
$\forall \; i = 1 \ldots n$

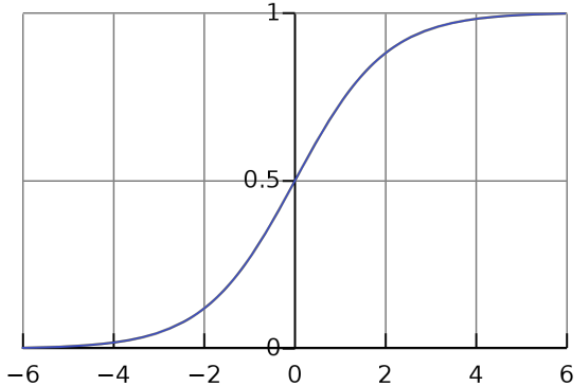
$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

- Not smooth
- Not continuous
- Not differentiable

- Smooth
- Continuous
- Differentiable

Perceptron – Forward propagation

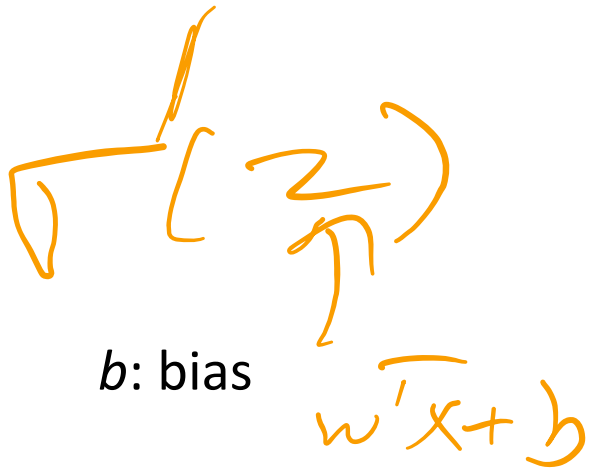




$$\hat{y} = g(z) = g(\Sigma\, w_i x_i) \qquad \forall\, i = 0 \ldots n$$
$$= g(b + \Sigma\, w_i x_i) \qquad \forall\, i = 1 \ldots n$$
$$= g(b + \boldsymbol{X}^T \boldsymbol{W})$$

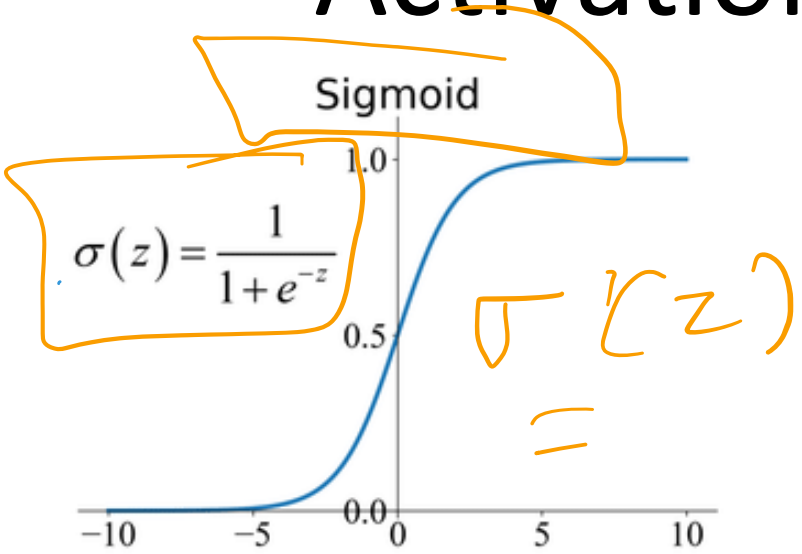
where, $\mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix}$ $\mathbf{W} = \begin{pmatrix} w_1 \\ w_2 \\ \ldots \\ w_n \end{pmatrix}$

b: bias



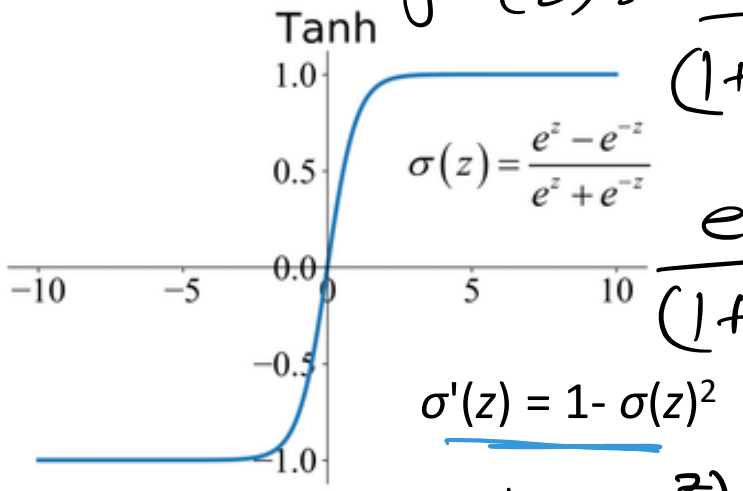
https://en.wikipedia.org/wiki/Sigmoid_function

Activation functions



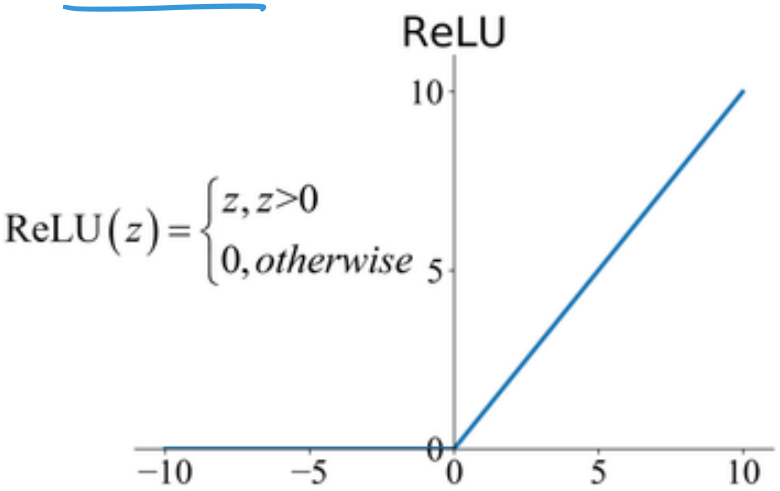
(a)

$\sigma'(z) = \sigma(z)(1-\sigma(z))$



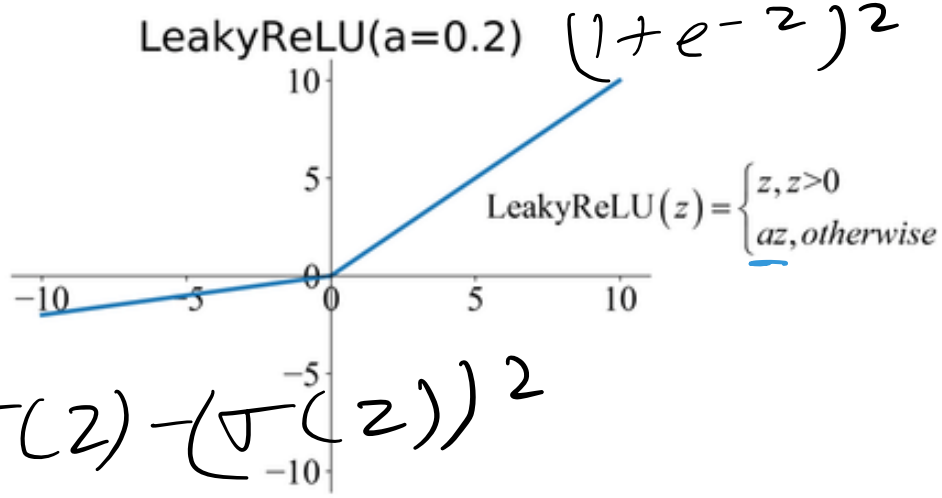
(b)

$\sigma'(z) = 1-\sigma(z)^2$



(c)

$\sigma'(z) = 1 \quad z>=0$
$= 0 \quad z<0$



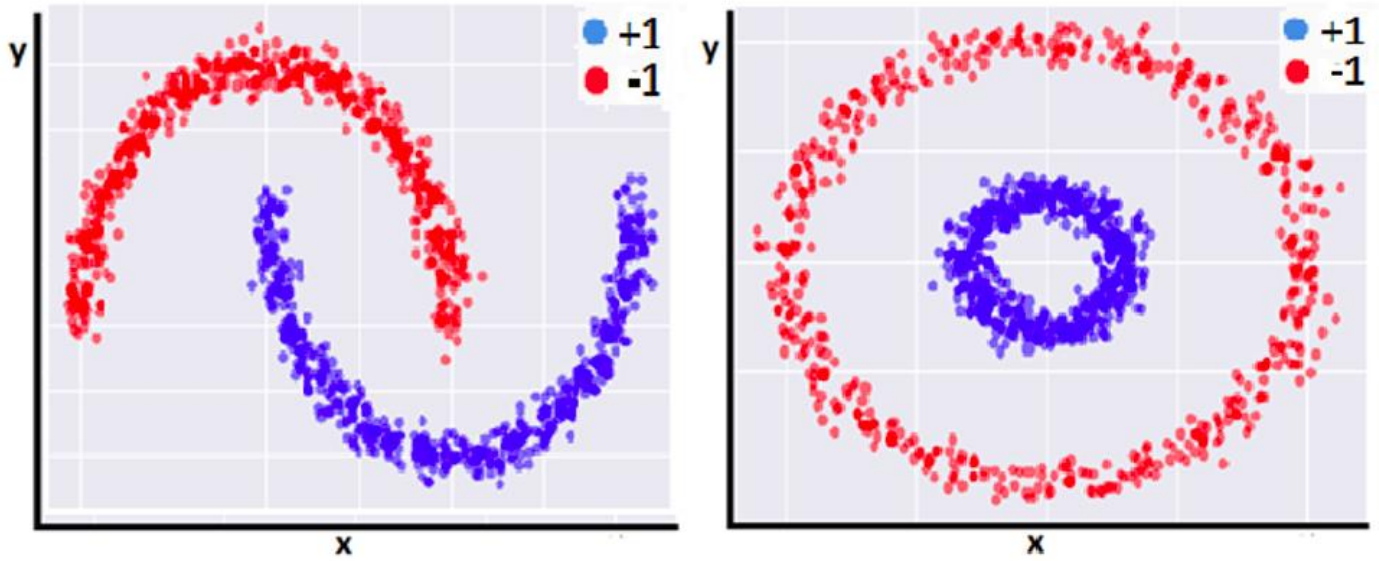
(d)

$\sigma'(z) = 1 \quad z>=0$
$= a \quad z<0$

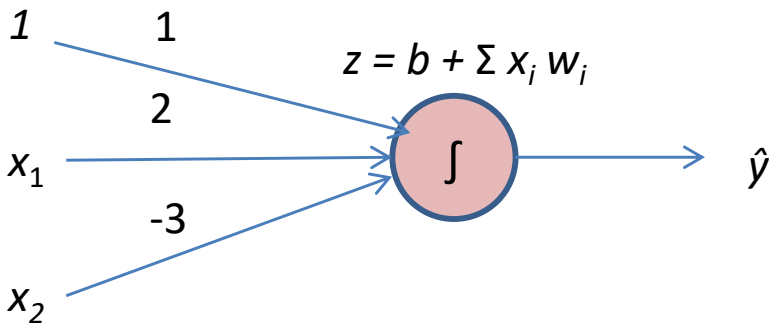
$$\tanh = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Importance - Activation functions

- Introduce non-linearity in the network
  - Allows to deal with non-linear data
  - Allows to approximate complex functions



Perceptron – Example



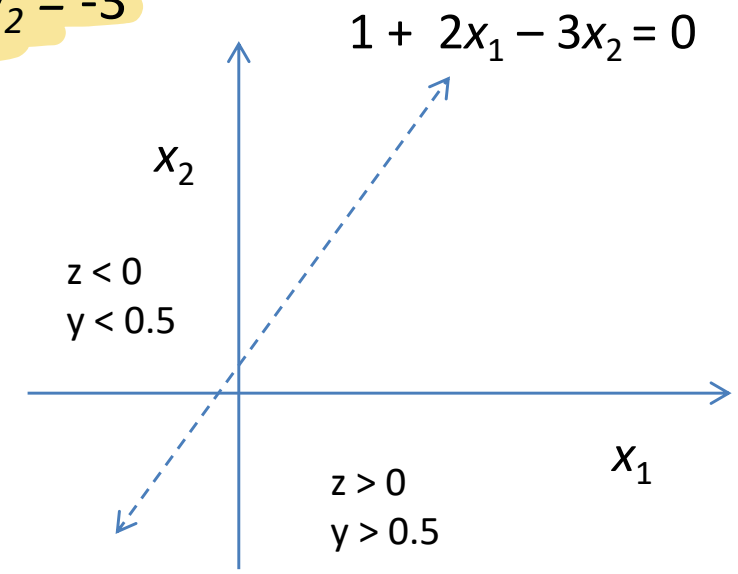
$$\boldsymbol{X} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad \boldsymbol{W} = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$$

$b = 1,\ w_1 = 2,\ w_2 = -3$

$$\hat{y} = g(b + \boldsymbol{X}^T\boldsymbol{W})$$

$$\hat{y} = g\left(1 + \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}^T \begin{pmatrix} 2 \\ -3 \end{pmatrix}\right)$$

$$\hat{y} = g(1 + 2x_1 - 3x_2)$$



Typical Machine Learning Problem

x_1	x_2	x_3	…	x_n	y

- Actor, Producer, Genre…..Watch a movie?
- Salinity, Density….Will find oil?
- Salary, Occupation, Tax filed…..Commit fraud? $y = f(x)$
- **Model**: Approximation of relation between x and y

  $\hat{y} = f'(x, w)$

  - w parameter needs to be learned from data
- **Learning algorithm**: Algorithm for learning parameters of model
- **Objective/Loss/Error function**: To guide the learning algorithm – learning algorithm should aim to minimize loss function

Typical Machine Learning Problem

- Actor, Producer, Genre…..Watch a movie?
- Input for training:

  $\{x_i, y_i\}$ $\quad i = 1..m$ $\quad m$ pairs of (x, y)
- Model: $y = f'(x, w, b)$
- Learning algorithm: Ex. Gradient Descent
- Objective/Loss/Error function:
  - One possibility: $L(w,b) = \sum (\hat{y}_i - y_i)^2$ for $i = 1..m$

Real → Regress.

- Learning algorithm should find w to minimize the objective function
  - Hope to find sigmoid function such that given x_i, y_i lie on that function

$-\left(\sum p_i \log(q_i)\right)$

Example

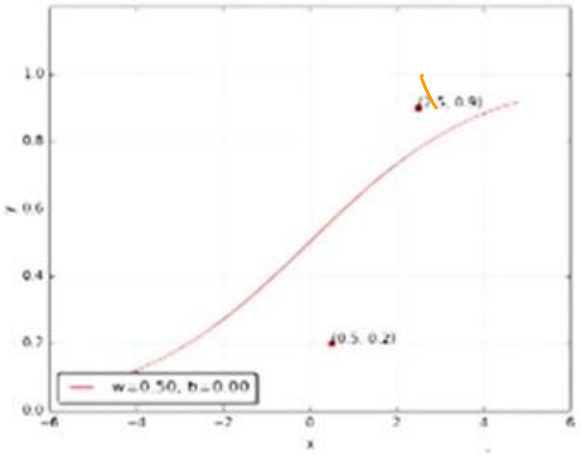
- Ex. data points: (0.5,0.2), (2.5, 0.9)
- At end of training, expect to find w^*, b^* such that

  f(0.5) -> 0.2, f(2.5) -> 0.9
- Loss = $(1/2)^*\sum(y - \hat{y})^2$ given: $w = 0.5$, $b = 0$

  $= (1/2)^*[(0.2 - f(0.5))^2 + (0.9 - f(2.5)^2]$

  $= 0.073$

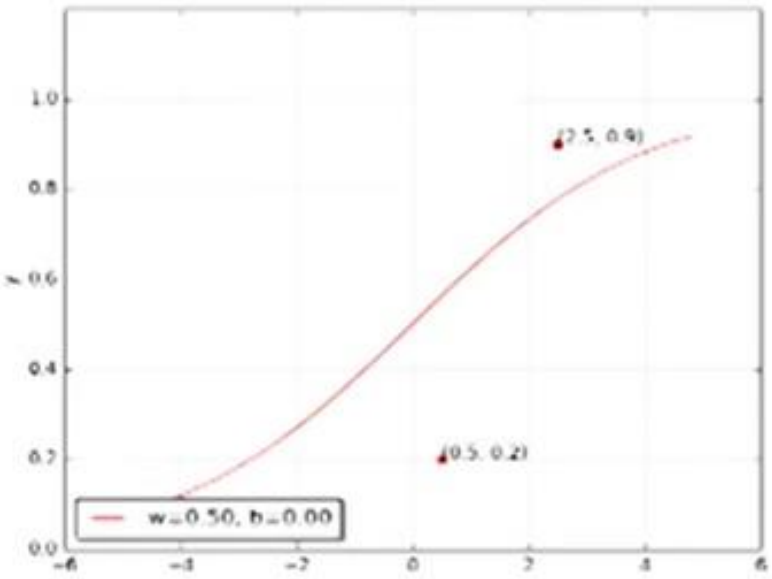
$$y = \frac{1}{1+e^{-(w^T x+b)}}$$



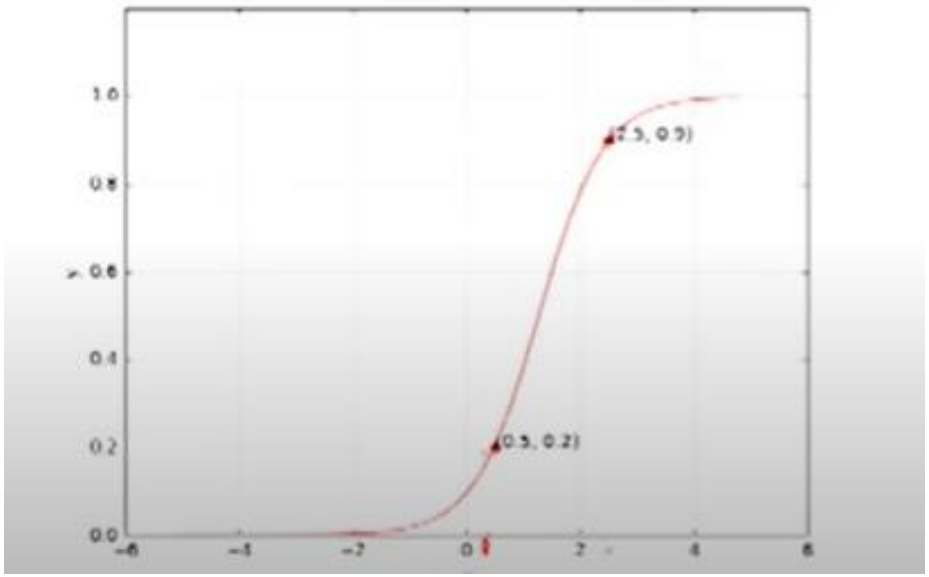
$w = 0.5$, $b = 0$

Example

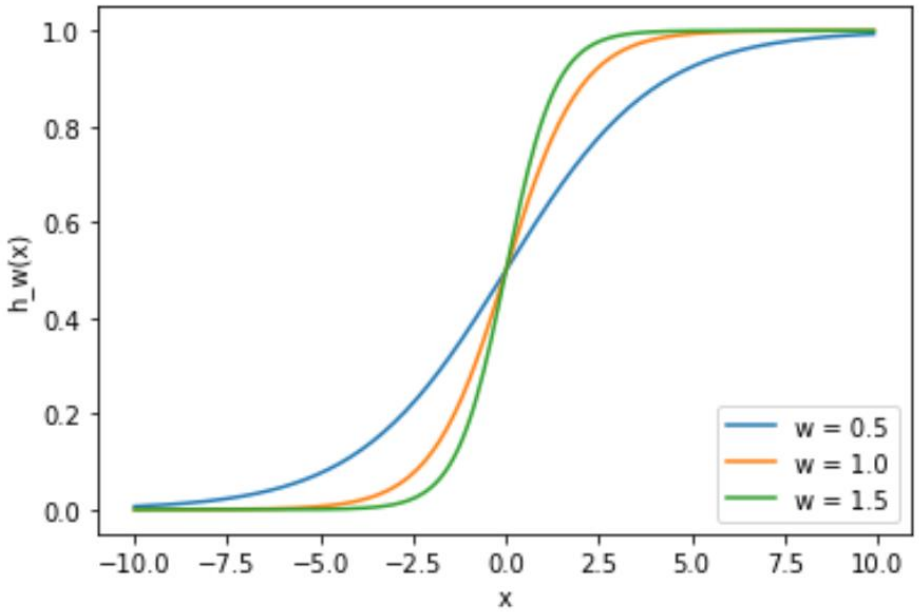
- Ex. data points: (0.5,0.2), (2.5, 0.9)
- Hope to find a sigmoid function that the data points lie on the function

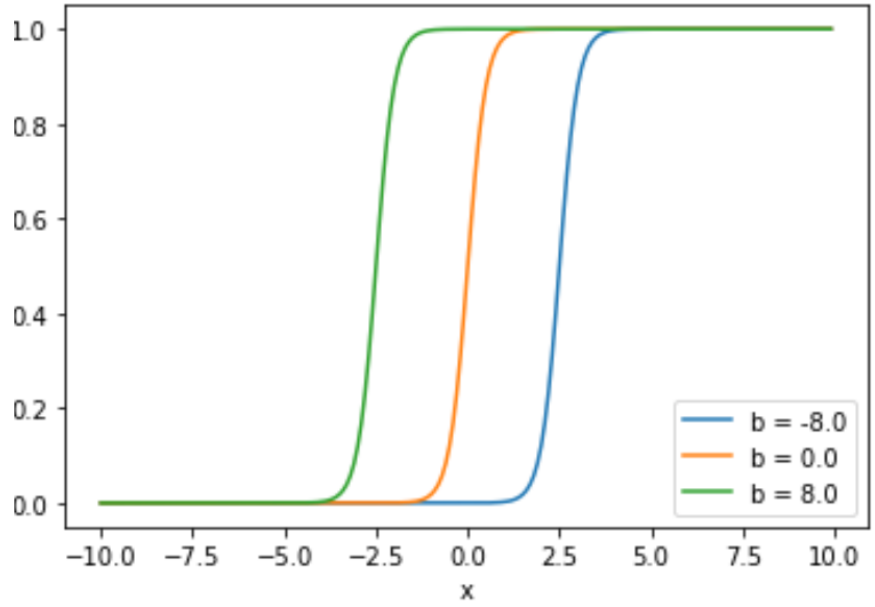


$w = 0.5, b = 0$



Sigmoid function





Changes w.r.t *w*

↳ weights change
slope

Changes w.r.t *b*

↓
bias change

position.

Guesses!!

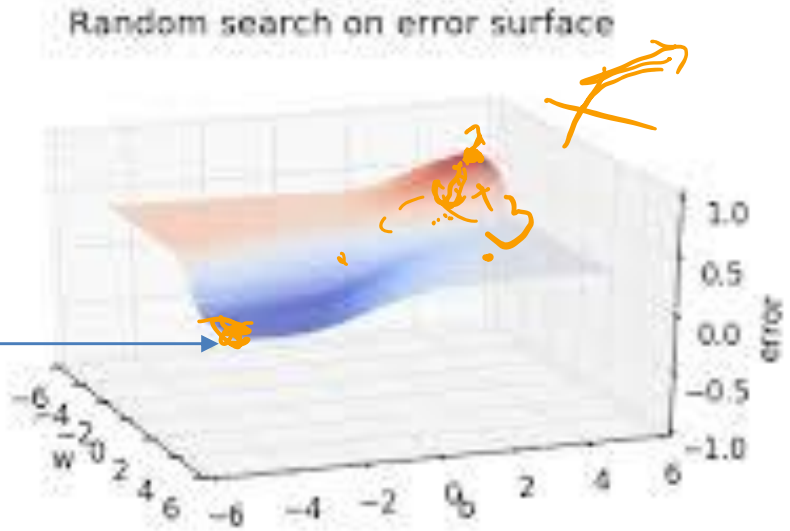
w	b	Loss
0.5	0	0.073
-0.1	0	0.1481
0.94	-0.94	0.0214
1.42	-1.73	0.0028
1.65	-2.08	0.0003
1.78	-2.27	0.0000

Error surface

- 2 data points, 2 parameters (*w, b*)

Lowest error

Other *w, b* values can be checked



- How to handle more data points and parameters??

Gradient Descent

- Find a way to traverse error surface to reach minimum value quickly
- Parameters: $\theta = [w, b]$
- Change in parameters: $\Delta\theta = [\Delta w, \Delta b]$
- New parameters: $\theta_{new} = \theta + \Delta\theta = [w_{new}, b_{new}]$
  - Change with a small stride η

$$w_{new} = w + \eta . \Delta w$$

$$b_{new} = b + \eta . \Delta b$$

- How to choose Δw and Δb?

Gradient Descent

- According to Taylor series:

$$L(\theta + \eta\Delta\theta) = L(\theta) + \eta*(\Delta\theta)^T\nabla_\theta L(\theta) + (\eta^2/2!)*(\Delta\theta)^T\nabla_\theta^2 L(\theta)(\Delta\theta) + \ldots$$

$$= L(\theta) + \eta(\Delta\theta)^T\nabla_\theta L(\theta) \quad \text{(if } \eta \text{ is small)}$$

- New loss should be less than old loss
- $(\eta\Delta\theta)$ would be favourable only if:

$$L(\theta + \eta\Delta\theta) < L(\theta)$$

$$L(\theta + \eta\Delta\theta) - L(\theta) < 0$$

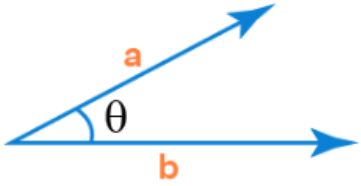
- Implies: $(\Delta\theta)^T \nabla_\theta L(\theta) < 0$ (η is positive constant)

$$\nabla_\theta L(\theta) = \frac{\partial L(\theta)}{\partial \theta}$$

Gradient Descent

- Desired: $(\Delta\theta)^T \nabla_\theta L(\theta) < 0$
  - Want $\Delta\theta$ to be as negative as possible
  - Dot product of two vectors: product of their magnitudes multiplied by the cosine of the angle between them
- Let β be the angle between $\Delta\theta$ and $\nabla_\theta L(\theta)$
- Then:

$$-1 \leq \cos(\beta) = \frac{(\Delta\theta)^T \nabla_\theta L(\theta)}{||\Delta\theta|| * ||\nabla L_\theta(\theta)||} \leq 1$$



$$\cos\theta = \frac{a.b}{|a|\,|b|}$$

Most negative: $\cos(\beta) = -1$ when $\beta = 180^\circ$

-> $\Delta\theta$ should be such that it is at 180° to the gradient $\nabla L(\theta)$

Gradient Descent

- $(\Delta\theta)^T \nabla L(\theta) < 0$
- Move in direction opposite to gradient (180° w.r.t. the gradient)

$$w_{t+1} = w_t - \eta \nabla w_t$$

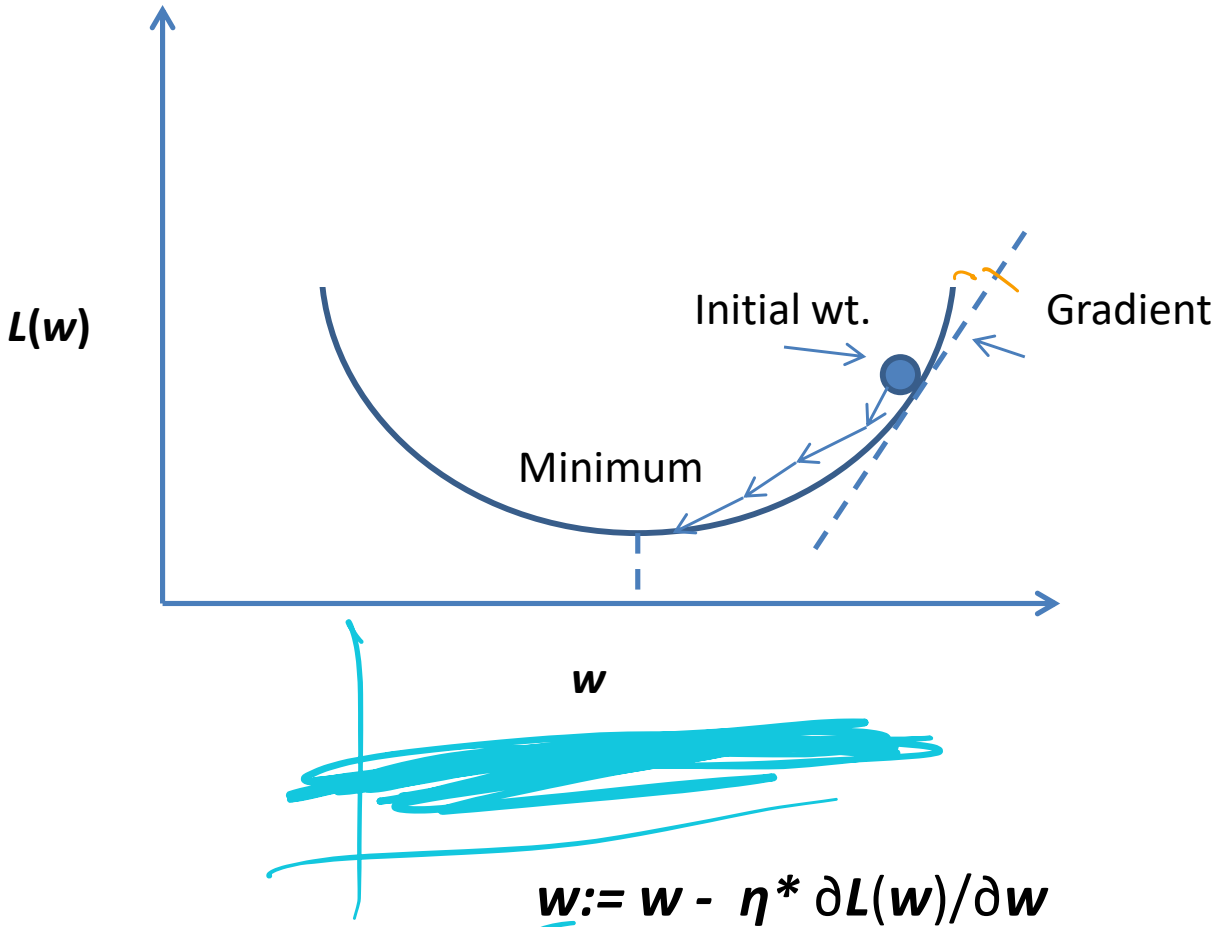
$$b_{t+1} = b_t - \eta \nabla b_t$$

where, $\nabla w_t = \frac{\partial L(w,b)}{\partial w}$ at $w = w_t$ and $b = b_t$

$$\nabla b_t = \frac{\partial L(w,b)}{\partial b}$$ at $w = w_t$ and $b = b_t$

- Repeat till convergence

Gradient Descent



Randomly pick (w_0, w_1)

Compute *L*

Compute gradient $\partial L(w)/\partial w$ - gives ascent at that point

Take small step in opposite direction of gradient

Repeat until convergence

$$w := w - \eta * \partial L(w)/\partial w$$

$$SSE = \sum (\hat{y} - y)^2$$

$$SST = \sum (y - \bar{y})^2$$

$$R^2 = 1 - \frac{SSE}{SST}$$

Gradient Descent

Assuming, $L(w, b) = (1/2)*\sum(\hat{y} - y)^2$

$\hat{y} = f(x) = 1/(1+ e^{-(wx+b)})$

Assuming one data point only:

$$\frac{\partial L(w,b)}{\partial w} = (1/2)*[2*(f(x) - y) * \frac{\partial (f(x)-y)}{\partial w}]$$

$$= (f(x) - y) * \frac{\partial}{\partial w}[\frac{1}{1+e^{-(wx+b)}}]$$

$$\frac{\partial L(w,b)}{\partial w} = (f(x) - y) * f(x)*(1-f(x))*x = (\hat{y} - y) * \hat{y}*(1- \hat{y})*x$$

$$\frac{\partial L(w,b)}{\partial b} = (f(x) - y) * \frac{\partial}{\partial b}[\frac{1}{1+e^{-(wx+b)}}]$$

$$= (f(x) - y) * f(x)*(1-f(x)) = (\hat{y} - y) * \hat{y}*(1- \hat{y})$$

Gradient Descent

$\nabla w = \sum[(\hat{y}_i - y_i) * \hat{y}_i*(1 - \hat{y}_i)*x_i]$ for all points

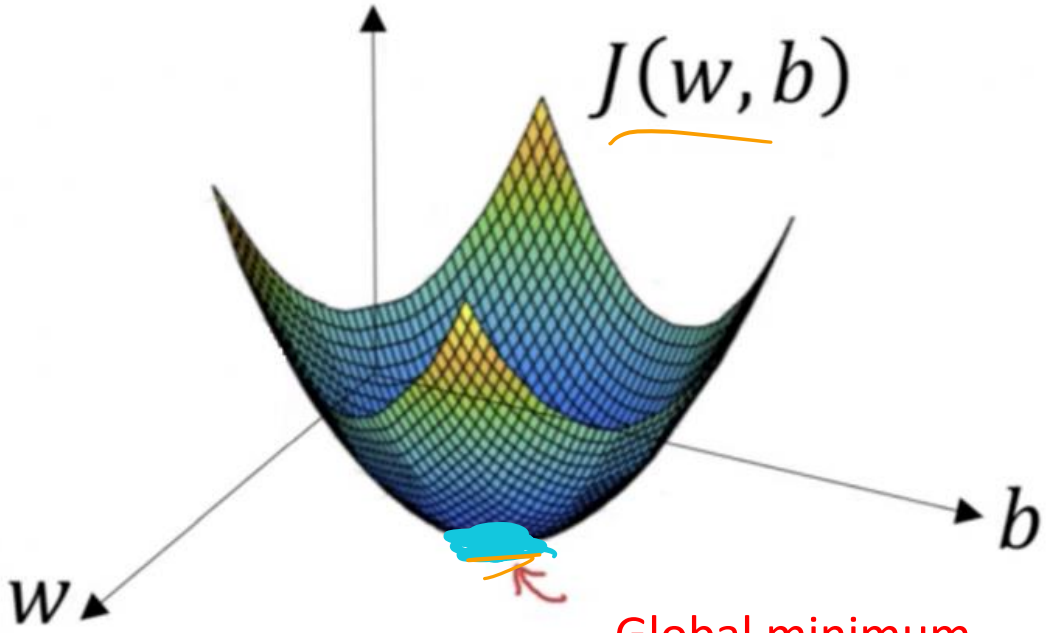
$\nabla b = \sum[(\hat{y}_i - y_i) * \hat{y}_i*(1 - \hat{y}_i)]$ for all points

- Algorithm:
  1. Initialize weights randomly
  2. Loop until convergence:
     1. Compute gradient $\frac{\partial L(w,b)}{\partial w}, \frac{\partial L(w,b)}{\partial b}$
     2. Update weights $\boldsymbol{w := w - \eta * \frac{\partial L(w,b)}{\partial w}}$

        $\boldsymbol{b := b - \eta * \frac{\partial L(w,b)}{\partial b}}$

  Return weights

Gradient Descent



Global minimum
Assuming $J(W)$ to be a convex function

https://builtin.com/data-science/gradient-descent