MLAI 504

NEURAL NETWORKS AND DEEP LEARNING

MIAI – 2026/2027

Dr. Zein Al Abidin IBRAHIM

zein.ibrahim@ul.edu.lb



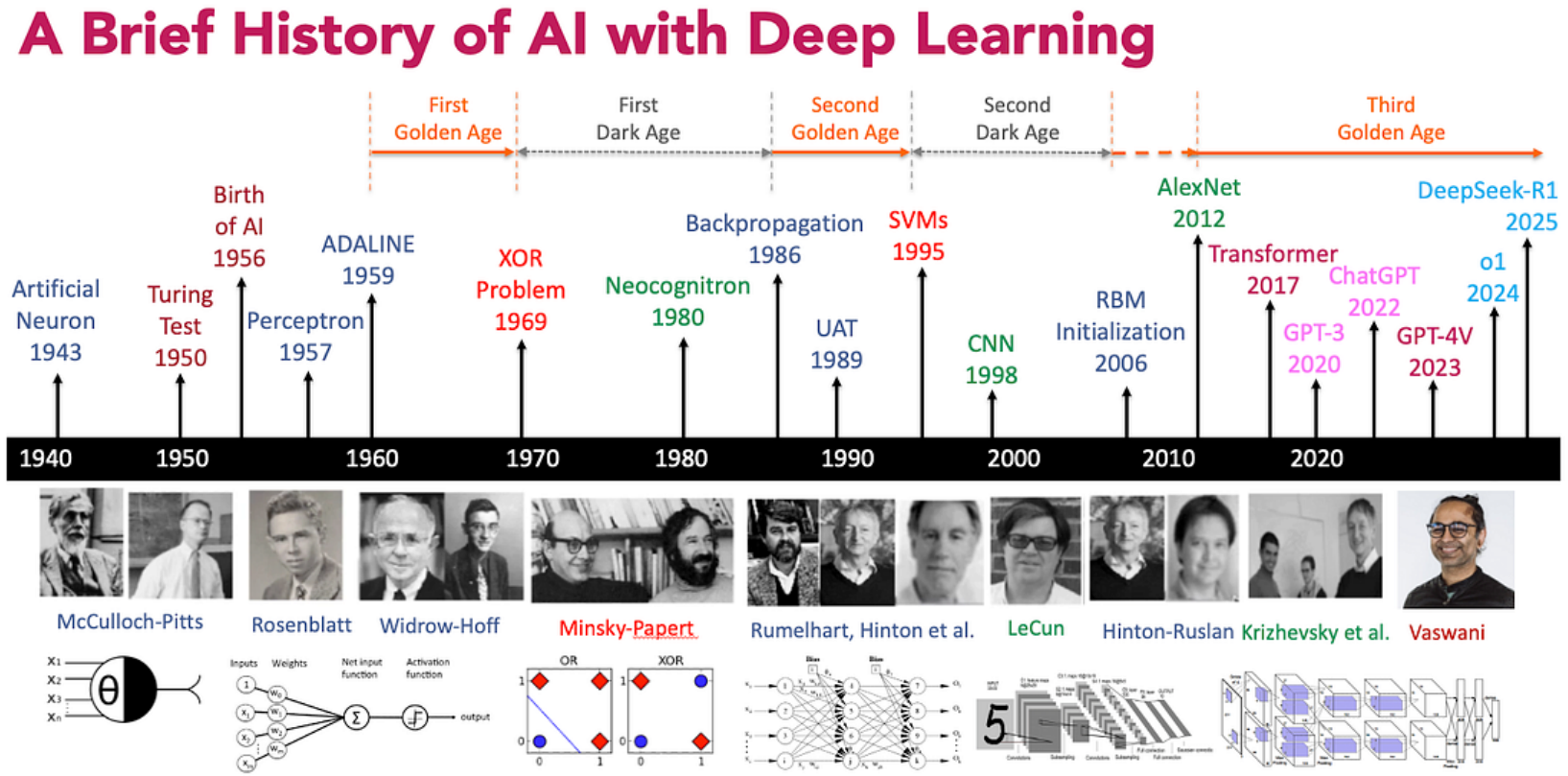
NEURAL NETWORKS: A QUICK OVERVIEW

DEEP LEARNING

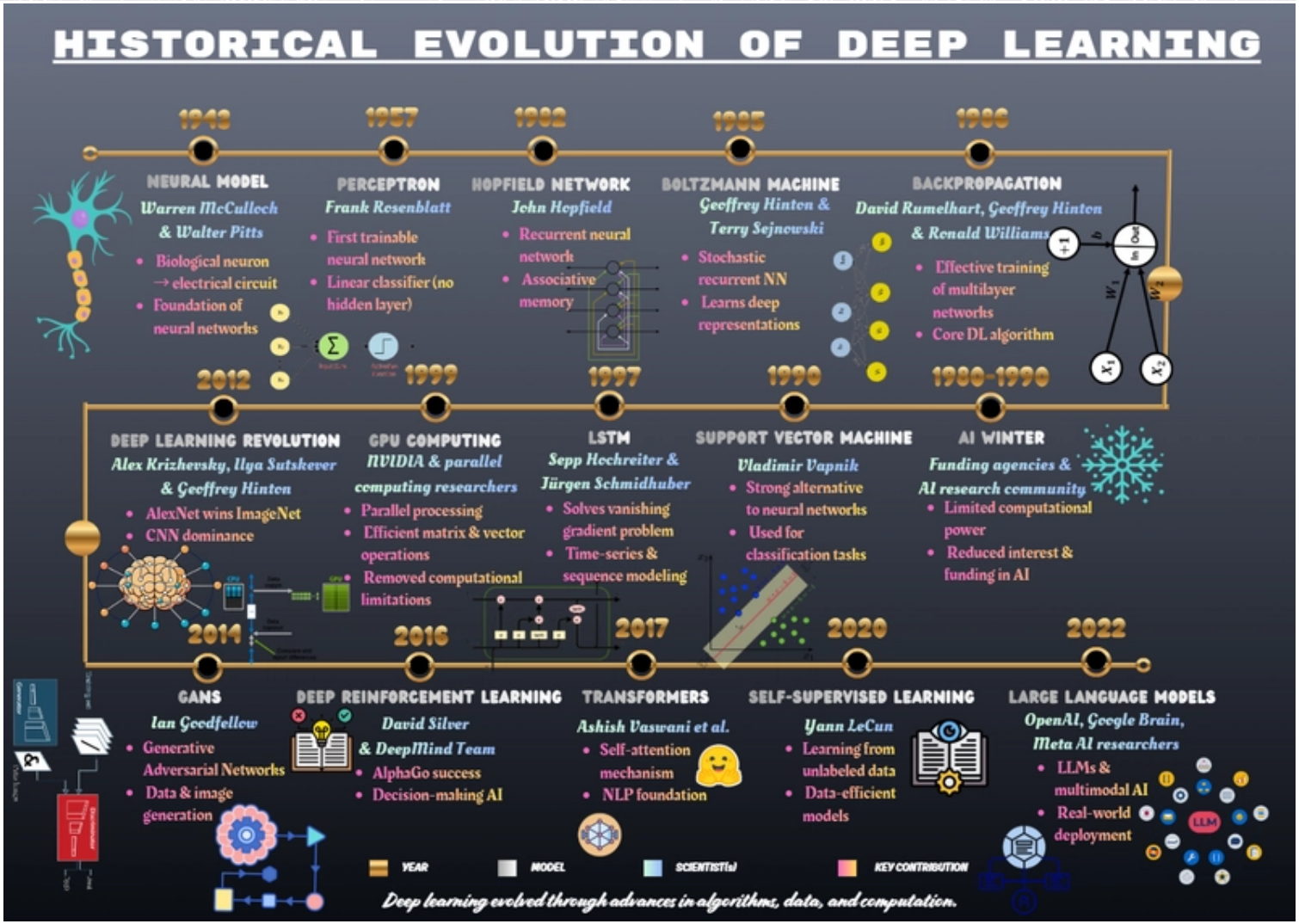
PERCEPTRON

NEURAL NETWORK

HISTORY


A Brief History of AI with Deep Learning
First Golden Age
First Dark Age
Second Golden Age
Second Dark Age
Third Golden Age
Artificial Neuron 1943
Turing Test 1950
Birth of AI 1956
Perceptron 1957
ADALINE 1959
XOR Problem 1969
Neocognitron 1980
Backpropagation 1986
UAT 1989
SVMs 1995
CNN 1998
RBM Initialization 2006
AlexNet 2012
Transformer 2017
GPT-3 2020
ChatGPT 2022
GPT-4V 2023
o1 2024
DeepSeek-R1 2025
1940
1950
1960
1970
1980
1990
2000
2010
2020
McCulloch-Pitts
Rosenblatt
Widrow-Hoff
Minsky-Papert
Rumelhart, Hinton et al.
LeCun
Hinton-Ruslan
Krizhevsky et al.
Vaswani
Inputs
Weights
Net input function
Activation function
output
OR
XOR

HISTORY


HISTORICAL EVOLUTION OF DEEP LEARNING
1943
NEURAL MODEL
Warren McCulloch & Walter Pitts
• Biological neuron → electrical circuit
• Foundation of neural networks
1957
PERCEPTRON
Frank Rosenblatt
• First trainable neural network
• Linear classifier (no hidden layer)
1982
HOPFIELD NETWORK
John Hopfield
• Recurrent neural network
• Associative memory
1985
BOLTZMANN MACHINE
Geoffrey Hinton & Terry Sejnowski
• Stochastic recurrent NN
• Learns deep representations
1986
BACKPROPAGATION
David Rumelhart, Geoffrey Hinton & Ronald Williams
• Effective training of multilayer networks
• Core DL algorithm
2012
DEEP LEARNING REVOLUTION
Alex Krizhevsky, Ilya Sutskever & Geoffrey Hinton
• AlexNet wins ImageNet
• CNN dominance
1999
GPU COMPUTING
NVIDIA & parallel computing researchers
• Parallel processing
• Efficient matrix & vector operations
• Removed computational limitations
1997
LSTM
Sepp Hochreiter & Jürgen Schmidhuber
• Solves vanishing gradient problem
• Time-series & sequence modeling
1990
SUPPORT VECTOR MACHINE
Vladimir Vapnik
• Strong alternative to neural networks
• Used for classification tasks
1980-1990
AI WINTER
Funding agencies & AI research community
• Limited computational power
• Reduced interest & funding in AI
2014
GANS
Ian Goodfellow
• Generative Adversarial Networks
• Data & image generation
2016
DEEP REINFORCEMENT LEARNING
David Silver & DeepMind Team
• AlphaGo success
• Decision-making AI
2017
TRANSFORMERS
Ashish Vaswani et al.
• Self-attention mechanism
• NLP foundation
2020
SELF-SUPERVISED LEARNING
Yann LeCun
• Learning from unlabeled data
• Data-efficient models
2022
LARGE LANGUAGE MODELS
OpenAI, Google Brain, Meta AI researchers
• LLMs & multimodal AI
• Real-world deployment
YEAR
MODEL
SCIENTIST(s)
KEY CONTRIBUTION
Deep learning evolved through advances in algorithms, data, and computation.

THE PERCEPTRON CONVERSION THEOREM

- For the perceptron to function properly, the two classes must be linearly separable.
  - If the classes are allowed to be close as shown in Figure below (b) then it becomes beyond the capability of perceptron.

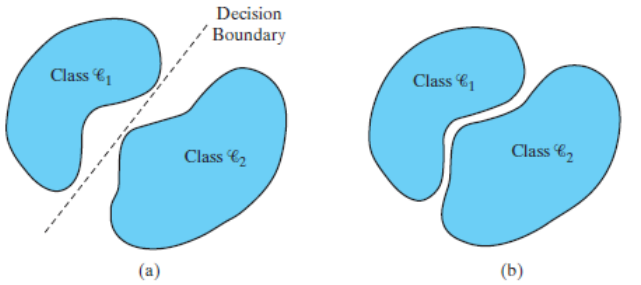
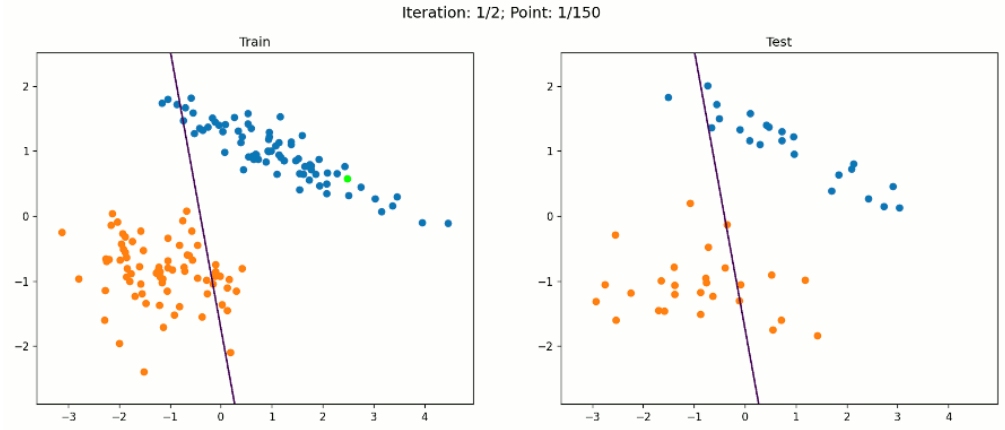


FIGURE 1.4 (a) A pair of linearly separable patterns. (b) A pair of non-linearly separable patterns.



MCCULLOCH-PITTS VS ROSENBLATT PERCEPTRON

Feature	McCulloch-Pitts Model	Rosenblatt Perceptron
Year	1943	1958
Purpose	Theoretical model of biological neurons	Practical implementation of a single artificial neuron
Inputs	Binary (0 or 1)	Real-valued
Weights	All identical	Can be different and positive or negative
Threshold	Single, fixed	Single, adjustable during learning
Activation function	Step function	Step function
Learning rule	None	Perceptron learning rule
Applications	Theoretical foundation for artificial neural networks	Simple pattern recognition, linear binary classification
Strengths	Laid the groundwork for artificial neurons, provided a simple model to understand	Introduced the concept of learning in neural networks, enabled practical applications
Weaknesses	Limited to binary inputs and outputs, couldn't solve non-linearly separable problems, no learning mechanism	Limited to linearly separable problems, slow convergence in some cases

THE PERCEPTRON CONVERSION THEOREM

- Suppose that two classes to be classified are linearly separable then:
  - **$w^T x > 0$ for each input vector x belonging to class 1**
  - **$w^T x \leq 0$ for each input vector x belonging to class 2**
  - **For the case $w^T x = 0$ we arbitrary said that it belongs to class 2**
- The algorithm for adapting weight vector of the elementary perceptron may be formulated as follows:
  1. If the nth member of the training set, $x(n)$, is **correctly** classified by the weight vector $w(n)$ computed at the nth iteration of the algorithm, no changes are done to the previous weights: **(no change)**
     1. **$w(n+1) = w(n)$ if $w^T(n)x(n) > 0$ and $x(n)$ belongs to class 1**
     2. **$w(n+1) = w(n)$ if $w^T(n)x(n) \leq 0$ and $x(n)$ belongs to class 2**
  2. Otherwise, the weight vectors are updated in accordance to the below:
     1. **$w(n+1) = w(n) - \eta(n)x(n)$ if $w^T(n)x(n) > 0$ and $x(n)$ belongs to class 2**
     2. **$w(n+1) = w(n) + \eta(n)x(n)$ if $w^T(n)x(n) \leq 0$ and $x(n)$ belongs to class 1**

     Where $\eta(n)$ is the learning rate parameter that controls the adjustment applied to the weight vector at iteration n

LEARNING RATE

- If $\eta(n) = \eta > 0$ where η is a constant independent of the iteration number n then we have **a fixed-increment adaption rule** for the perceptron.
- Types of learning rates used by other algorithms:
  - Constant learning rate → Easy to implement & can be effective for simple problems
  - Decreasing learning rate → Helps model to converge smoothly and avoid oscillations
  - Adaptive learning rate → Adjusts itself dynamically based on parameters and loss function
  - Others → Cyclic learning rate, Warmup learning rate, ….

TABLE 1.1 Summary of the Perceptron Convergence Algorithm

Variables and Parameters:

$\mathbf{x}(n)$ = $(m+1)$-by-1 input vector
$= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T$
$\mathbf{w}(n)$ = $(m+1)$-by-1 weight vector
$= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T$
b = bias
$y(n)$ = actual response (quantized)
$d(n)$ = desired response
η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \text{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

where sgn(·) is the signum function.

4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

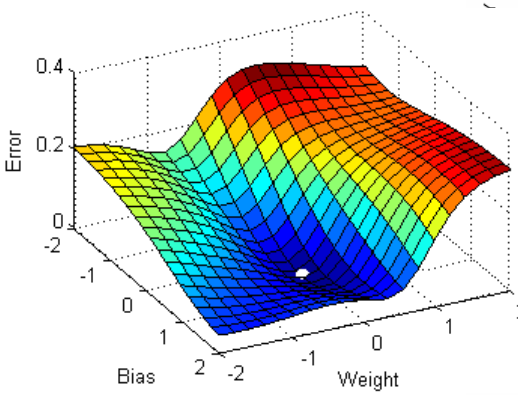
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

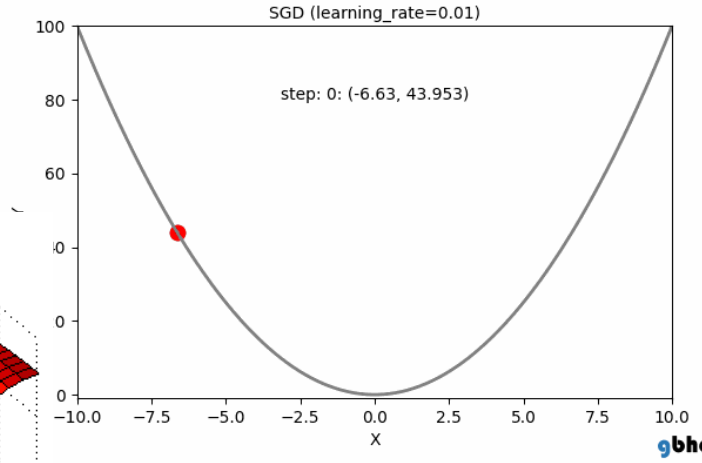
where

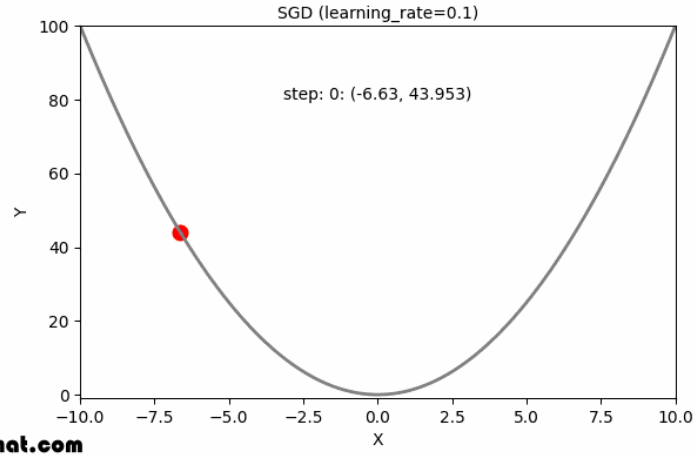
$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

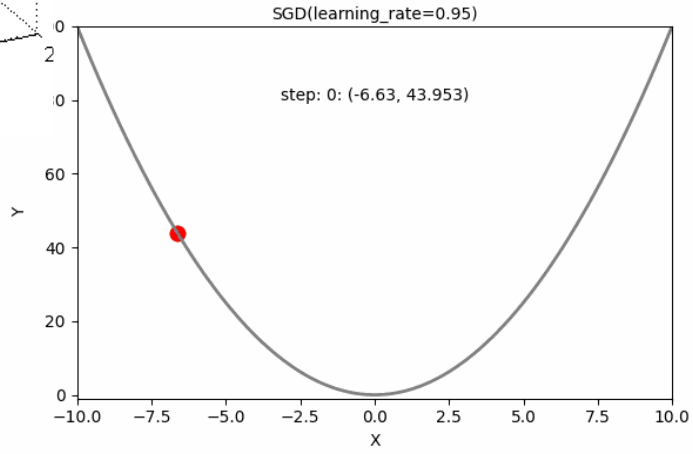
GRADIENT DESCENT LEARNING RATES

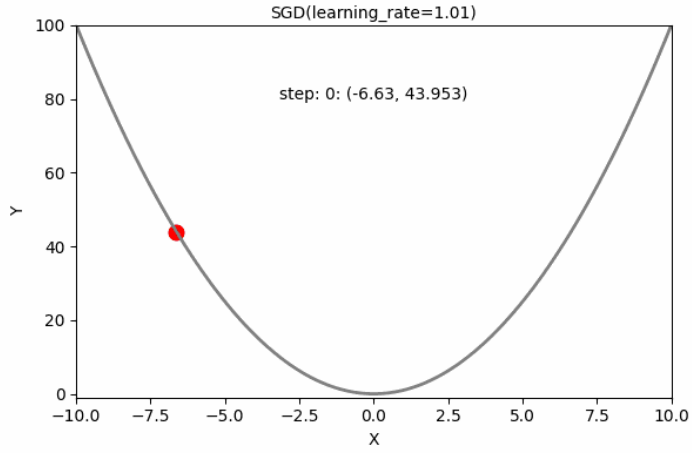






gbhat.com





THE BATCH PERCEPTRON ALGORITHM

- The conversion algorithm of the perceptron was presented without reference to a **cost function**.
  - Moreover, the derivation focused on a single-sample correction.
- In the **batch algorithm** we will present two things:
  - Introduce the generalized form of a perceptron cost function
  - Use the cost function to formulate the batch version.
- The cost function is one that permits the application for a **gradient search**. The perceptron cost function is defined as:
  - $J(w) = \sum_{x(n) \in \mathcal{H}} (-w^T x(n) d(n))$
  - Where $\mathcal{H}$ is the set of samples x misclassified by a perceptron using w as its weight vector
  - If all the samples are classified correctly, then the set $\mathcal{H}$ is empty and $J(w)$ is zero

THE BATCH PERCEPTRON ALGORITHM

- The cost function $J(w)$ can be differentiated on w which yields the gradient vector:
  - $\nabla J(w) = \sum_{x(n) \in \mathcal{H}} (-x(n)d(n))$
  - Where the gradient operator is :
    - $\nabla = \left[\frac{\partial}{\partial w_1}, \frac{\partial}{\partial w_2}, \ldots, \frac{\partial}{\partial w_m}\right]^T$
  - In the method of the **steepest descent**, the adjustment of the weight vector w at each timestep of the algorithm is applied in a direction opposite to the gradient vector $\nabla J(w)$ accordingly:
    - $w(n+1) = w(n) - \eta(n) \nabla J(w)$
    - $= w(n) + \eta(n) \sum_{x(n) \in \mathcal{H}} (x(n)d(n))$
    - This include the single sample correction version as a special case of the convergence algorithm.
    - The adjustment of the weights at timestep $n+1$ is defined by the sum of all samples misclassified by the weight vector $w(n)$ scaled by the learning rate $\eta(n)$.
    - The algorithm is called batch since at each time step a batch of misclassified samples are used to compute the adjustment.

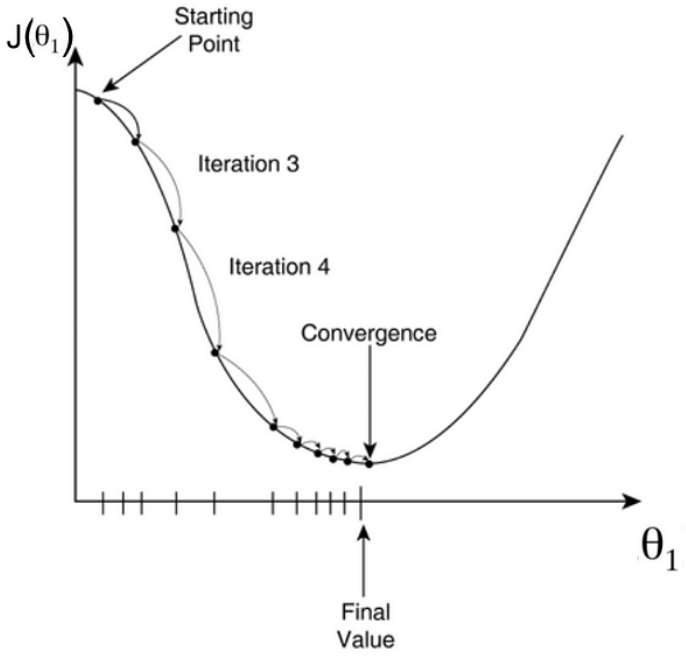
GRADIENT DESCENT

- Fundamental algorithm in machine learning and optimization.
- Used to train various models.
- How it works:
  - Imagine you're lost in a mountainous landscape and trying to find the valley (minimum point).
  - Gradient descent would be like taking small steps downhill, always following the steepest slope you can feel.
- Here's the gist:
  - **Imagine a function as a landscape**: The function's output (like error) varies depending on its inputs. High points represent areas with high error (think of them as hills).
  - **Start at a random point**: This is your initial guess for the model's parameters (weights).
  - **Calculate the slope (gradient)**: This tells you how much and in which direction the error changes when you move a little bit.
  - **Take a small step downhill**: Adjust the parameters slightly in the direction of the steepest descent.
  - **Repeat**: Keep calculating the slope and taking small steps downhill, until you reach a point where the slope is almost zero (a minimum).

GRADIENT DESCENT – A PRE-LOOK

- Different types of gradient descent:
  - **Batch gradient descent**: Considers the entire dataset for each step, can be slow for large datasets.
  - **Stochastic gradient descent**: Considers only one data point at a time, faster but can be noisy.
  - **Mini-batch gradient descent**: Considers a small batch of data points at a time, balances speed and stability.
- Gradient descent is powerful, but it's not perfect:
  - Can get stuck in local minima: Imagine being stuck in a small valley instead of the main one. You'd need a good starting point or momentum to escape.
  - Requires differentiable **functions**: Not all functions are smooth enough for gradient descent to work.
  - Can be sensitive to learning rate: A too-high rate might overshoot the minimum, too-low might get stuck.

GRADIENT DESCENT



Cost Function – "One Half Mean Squared Error":

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

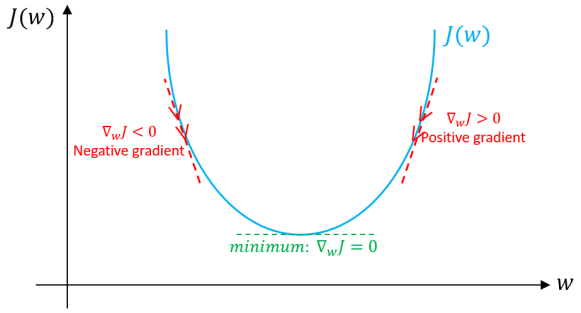
Objective:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

Derivatives:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$



PERCEPTRON VS GRADIENT DESCENT

Similarity:

- Both are iterative optimization algorithms used to minimize an error function.
- Both update the parameters (weights) of a model based on the gradient of the error function.
- Both work well for linear problems or can be used with non-linear problems if combined with other techniques like feature engineering or neural networks.

Feature	Gradient Descent	Perceptron
Objective function	Any differentiable function	Misclassification error (for binary classification)
Learning rule	Adjusts weights proportionally to the gradient	Updates weights only if prediction is wrong
Convergence guarantee	No guaranteed convergence (can get stuck in local minima)	Guaranteed convergence for linearly separable problems
Number of updates per iteration	Updates all weights at once	Updates only weights of misclassified inputs
Applications	Wider range of tasks, including regression, multi-class classification, etc.	Simple binary classification problems

PROBLEM SOLVING

Perceptron

DATA ANALYSIS

- Given the below dataset

Training Example	x_1	x_2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

1. Construct by hand a Perceptron which correctly classifies the above data. Use your knowledge of plane geometry to choose appropriate values for the weights w_0, w_1 and w_2.
2. Demonstrate the Perceptron Learning Algorithm on the above data, using a learning rate of 1.0 and initial weight values of

$$w_0 = -1.5,\ w_1 = \ \ 0,\ w_2 = \ \ 2$$

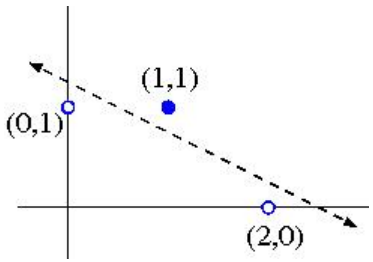
In your answer, you should clearly indicate the new weight values at the end of each training step.

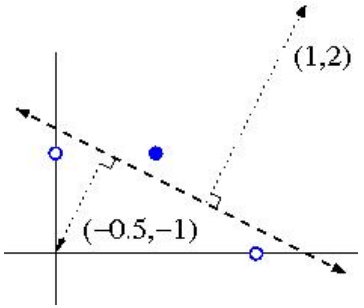
DATA ANALYSIS

- Solution (part a)
- First → plot the data in the 2D space (figure bottom left)
- This line has slope -1/2 and x_2-intersect 5/4, so its equation is:
- $x_2 = 5/4 - x_1/2$, i.e. $2x_1 + 4x_2 - 5 = 0$.
- Taking account of which side is positive, this corresponds to these weights:
- $w_0 = -5$, $w_1 = 2$, $w_2 = 4$

Training Example	x_1	x_2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

- Alternatively, we can derive weights $w_1=1$ and $w_2=2$ by drawing a vector normal to the separating line, in the direction pointing towards the positive data points:
- The bias weight w_0 can then be found by computing the dot product of the normal vector with a perpendicular vector from the separating line to the origin. In this case $w_0 = 1(-0.5) + 2(-1) = -2.5$
- (Note: these weights differ from the previous ones by a normalizing constant, which is fine for a Perceptron)





DATA ANALYSIS

- Solution (part b)

Training Example	x_1	x_2	Class
a.	0	1	-1
b.	2	0	-1
c.	1	1	+1

Iteration	w_0	w_1	w_2	Training Example	x_1	x_2	Class	$s=w_0+w_1x_1+w_2x_2$	Action
1	-1.5	0	2	a.	0	1	-	+0.5	Subtract
2	-2.5	0	1	b.	2	0	-	-2.5	None
3	-2.5	0	1	c.	1	1	+	-1.5	Add
4	-1.5	1	2	a.	0	1	-	+0.5	Subtract
5	-2.5	1	1	b.	2	0	-	-0.5	None
6	-2.5	1	1	c.	1	1	+	-0.5	Add
7	-1.5	2	2	a.	0	1	-	+0.5	Subtract
8	-2.5	2	1	b.	2	0	-	+1.5	Subtract
9	-3.5	0	1	c.	1	1	+	-2.5	Add
10	-2.5	1	2	a.	0	1	-	-0.5	None
11	-2.5	1	2	b.	2	0	-	-0.5	None
12	-2.5	1	2	c.	1	1	+	+0.5	None

PROBLEM SOLVING SESSION

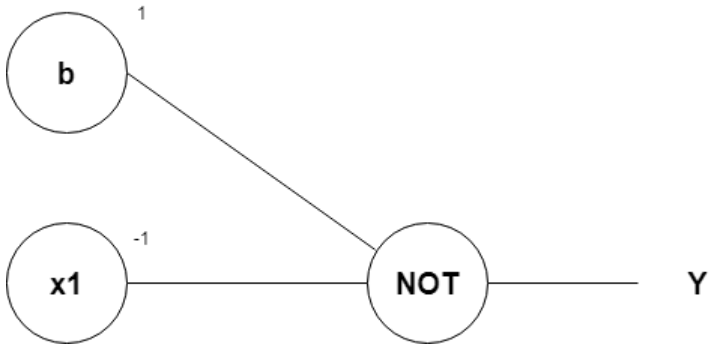
- Consider a McCulloch-Pitts Neural Network. Design one that implements a 1 input NOT Gate

Input 1	Output
0	1
1	0

PROBLEM SOLVING SESSION

- Consider a McCulloch-Pitts Neural Network. Design one that implements a 1 input NOT Gate

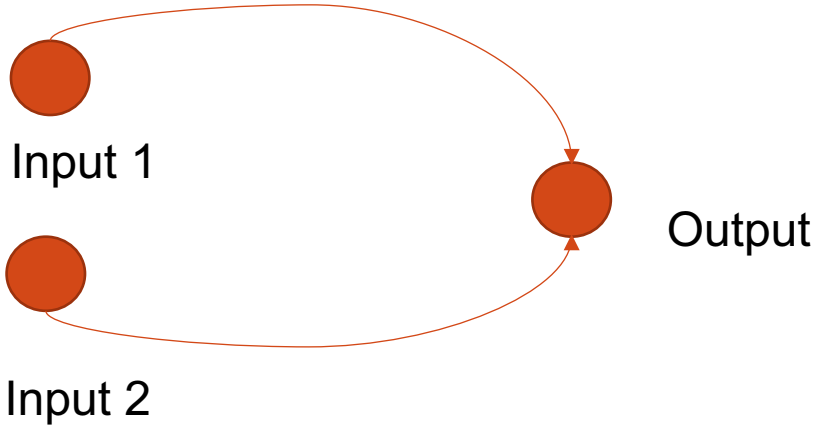
Input 1	Output
0	1
1	0



PROBLEM SOLVING SESSION

- Consider a McCulloch-Pitts Neural Network. Design one that implements a 2 input AND Gate

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

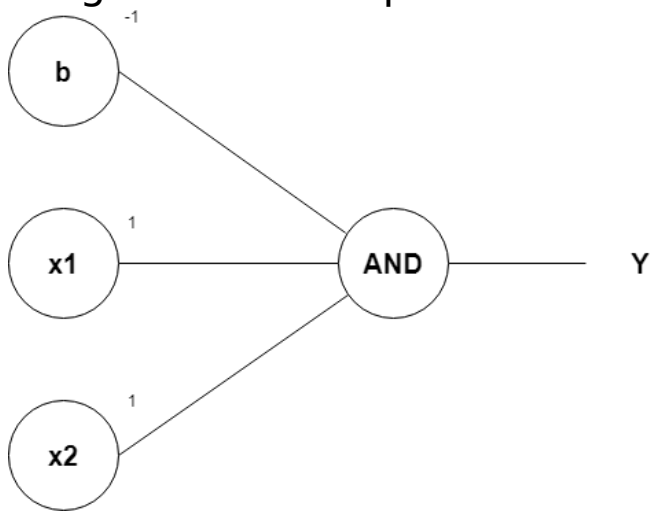


- We can deduce weights based on inference:
  - The only place the neuron fires is when the inputs are (1, 1)
  - Given the formula is
  - $v^k = \sum_{j=1}^{m} w^{kj} x^j$ and $y(k) = \begin{cases} 1, & v^k \geq 0 \\ \&0, & v^k < 0 \end{cases}$

PROBLEM SOLVING SESSION

- Consider a McCulloch-Pitts Neural Network. Design one that implements a 2 input AND Gate

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

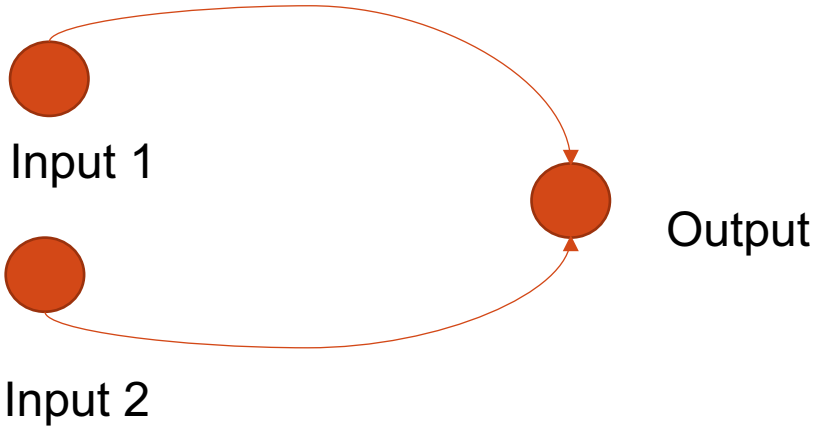


- We can deduce weights based on inference:
  - The only place the neuron fires is when the inputs are (1, 1)
  - Given the formula is
  - $v^k = \sum_{j=1}^{m} w^{kj} x^j$ and $y(k) = \begin{cases} 1, & v^k \geq 0 \\ \&0, & v^k < 0 \end{cases}$
  - We can solve the equations by placing the values of x1 and x2 and the output to get 4 equations with 3 unknowns. We end up with w1=1 and w2=1 and Threshold=2

PROBLEM SOLVING SESSION

- Consider a McCulloh Pitts Neural Network. Design one that implements a 2 input AND Not Gate

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	1
1	1	0



- We can deduce weights based on inference:
  - The only place the neuron fires is when the inputs are (1, 0)
  - Given the formula is
  - $v^k = \sum_{j=1}^{m} w^{kj} x^j$ and $y(k) = \begin{cases} 1, & v^k \geq 0 \\ \&0, & v^k < 0 \end{cases}$

PROBLEM SOLVING SESSION

- Consider a McCulloch-Pitts Neural Network. Design one that implements a 2 input AND Not Gate

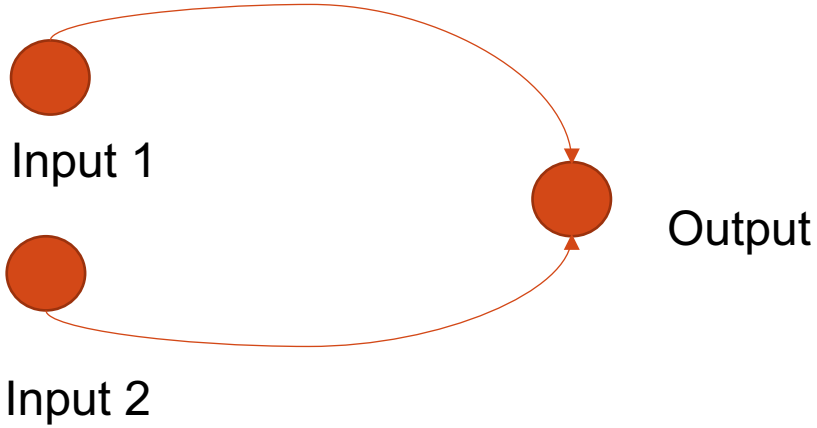
Input 1	Input 2	Output
0	0	0
0	1	0
1	0	1
1	1	0

- We can deduce weights based on inference:
  - The only place the neuron fires is when the inputs are (1, 1)
  - Given the formula is
  - $v^k = \sum_{j=1}^{m} w^{kj} x^j$ and $y(k) = \begin{cases} 1, & v^k \geq 0 \\ \&0, & v^k < 0 \end{cases}$
  - We can solve the equations by placing the values of x1 and x2 and the output to get 4 equations with 3 unknowns. We end up with w1=2 and w2=-1 and Threshold=2

PROBLEM SOLVING SESSION

- Consider a McCulloch-Pitts Neural Network. Design one that implements a 2 input OR

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

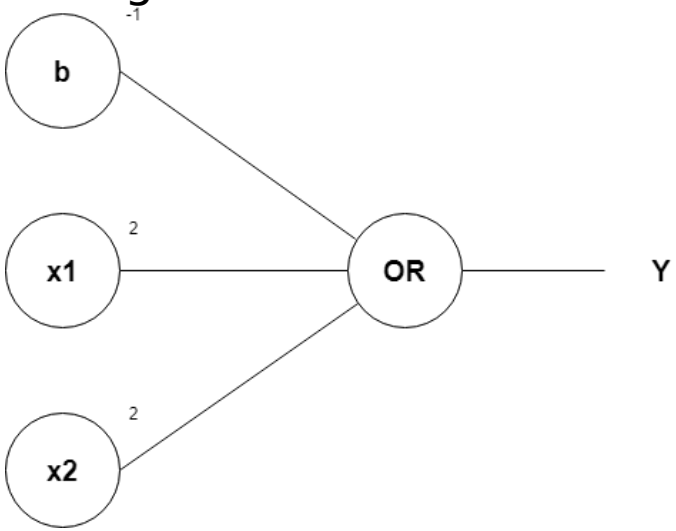


- We can deduce weights based on inference:
  - The only place the neuron does not fire is when the inputs are (0, 0)
  - Given the formula is
  - $v^k = \sum_{j=1}^{m} w^{kj} x^j$ and $y(k) = \begin{cases} 1, & v^k \geq 0 \\ \&0, & v^k < 0 \end{cases}$

PROBLEM SOLVING SESSION

- Consider a McCulloch-Pitts Neural Network. Design one that implements a 2 input OR

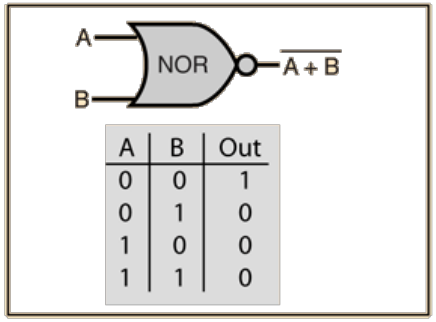
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1



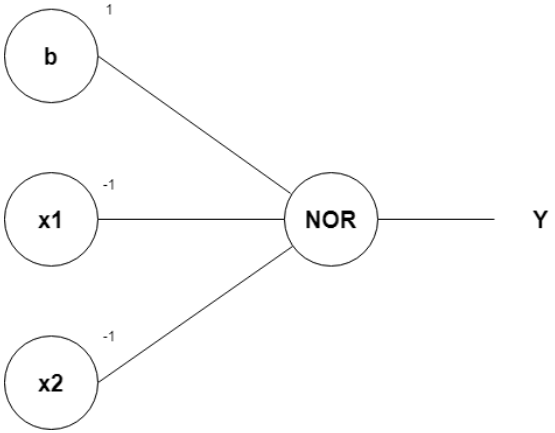
- We can deduce weights based on inference:
  - The only place the neuron does not fire is when the inputs are (0, 0)
  - Given the formula is
  - $v^k = \sum_{j=1}^{m} w^{kj} x^j$ and $y(k) = \begin{cases} 1, & v^k \geq 0 \\ \&0, & v^k < 0 \end{cases}$

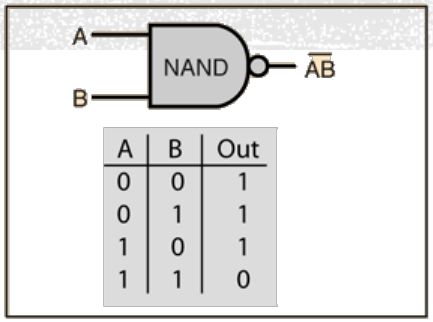
PROBLEM SOLVING SESSION

- Think to other gates like

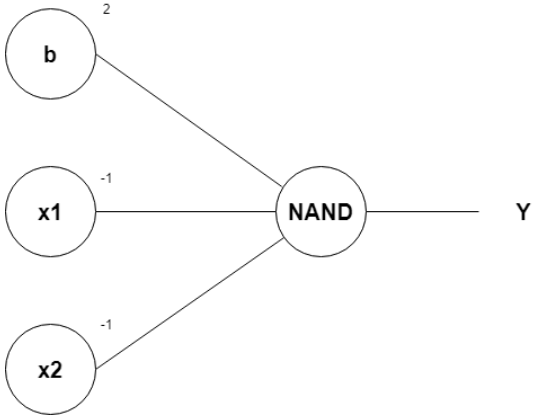


A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0



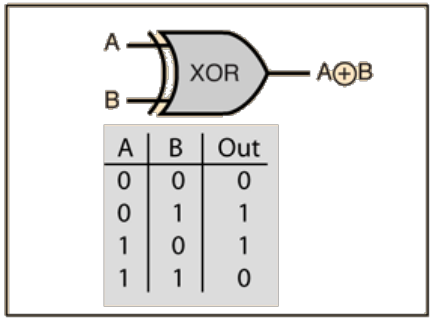


A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0



PROBLEM SOLVING SESSION

- What about the following



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0