SECA4002 – DEEP LEARNING NEURAL NETWORKS

Dr. V. Vedanarayanan B.E., M.E., PhD
Course Co-ordinator
Assistant Professor, Department of ECE,
School of Electrical and Electronics
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

Topics To Be Covered in this Video Lecture are

- Course Introduction
- Course Objectives
- Course Outcomes
- Detailed Curriculum
- Recommended Text Books/ Reference Books
- Unit -1- Introduction to Machine Learning
- Neural Networks –Working Principle
- Training of Neural Networks
- Loss Functions / Approximation functions of Artificial Neural Networks

Course Introduction

Part -1:

- What is Artificial Neural Networks (ANN's)?
- Why we need ANN's?
- Applications of ANN's

Part - 2:

- What is Deep Learning?
- Usefulness' Of Deep Learning Strategies
- Applications of Deep Learning

Course Introduction Continued

Artificial Neural Networks:

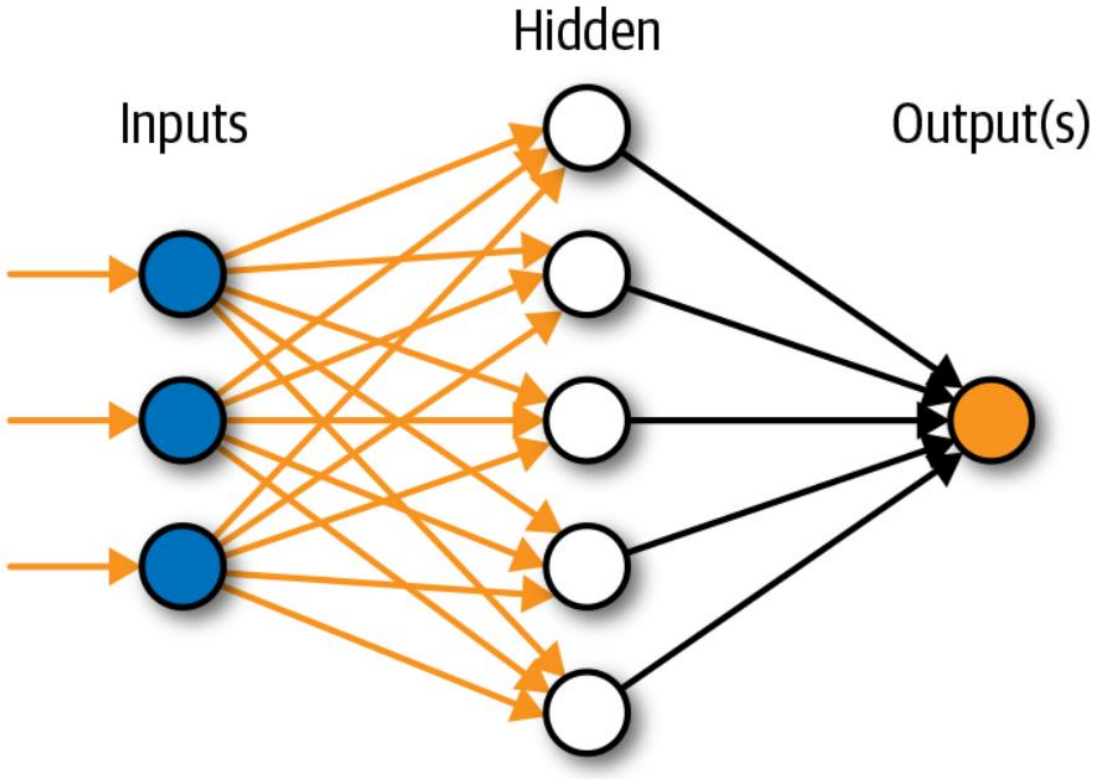


Image Source:https://www.innoarchitech.com/blog/artificial-intelligence-deep-learning-neural-networks-explained

Artificial Neural Networks:

- Interconnections of computing systems/processing units based on Biological neural system(BNN or BNS)

Course Introduction Continued.,

Need for Artificial Neural Networks(ANN):

- To develop a Fault Tolerant System
- Ability to learn
- Ability to model Non Linear & Complex problems
- To perform Regression analysis or Function Approximation
- Classification and Clustering Analysis
- Data Analytics, Data Processing, Data Compression
- Robotics Path calculations
- Control / Process applications

✓ Artificial Neural Networks are mainly used for Generalization

✓ Artificial Neural Networks is the main tool used in Artificial Intelligence/Natural Language Processing etc.

Course Introduction Continued.,

Applications of Artificial Neural Networks(ANN):

- Astronomy & Space Exploration
- Communication systems design and development
- Credit Rating / Targeted Marketing
- Defence R&D
- Financial forecasting & Portfolio Management
- Fraud Detection / Forensic Applications
- Intelligent Searching
- Medical Field
- Machine Diagnostics
- Process modelling and Control
- Robotics Path calculations

Course Introduction Continued

Part-2:

What is Deep Learning (DL)?

It is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example – [Courtesy: WWW. Mathworks.com]

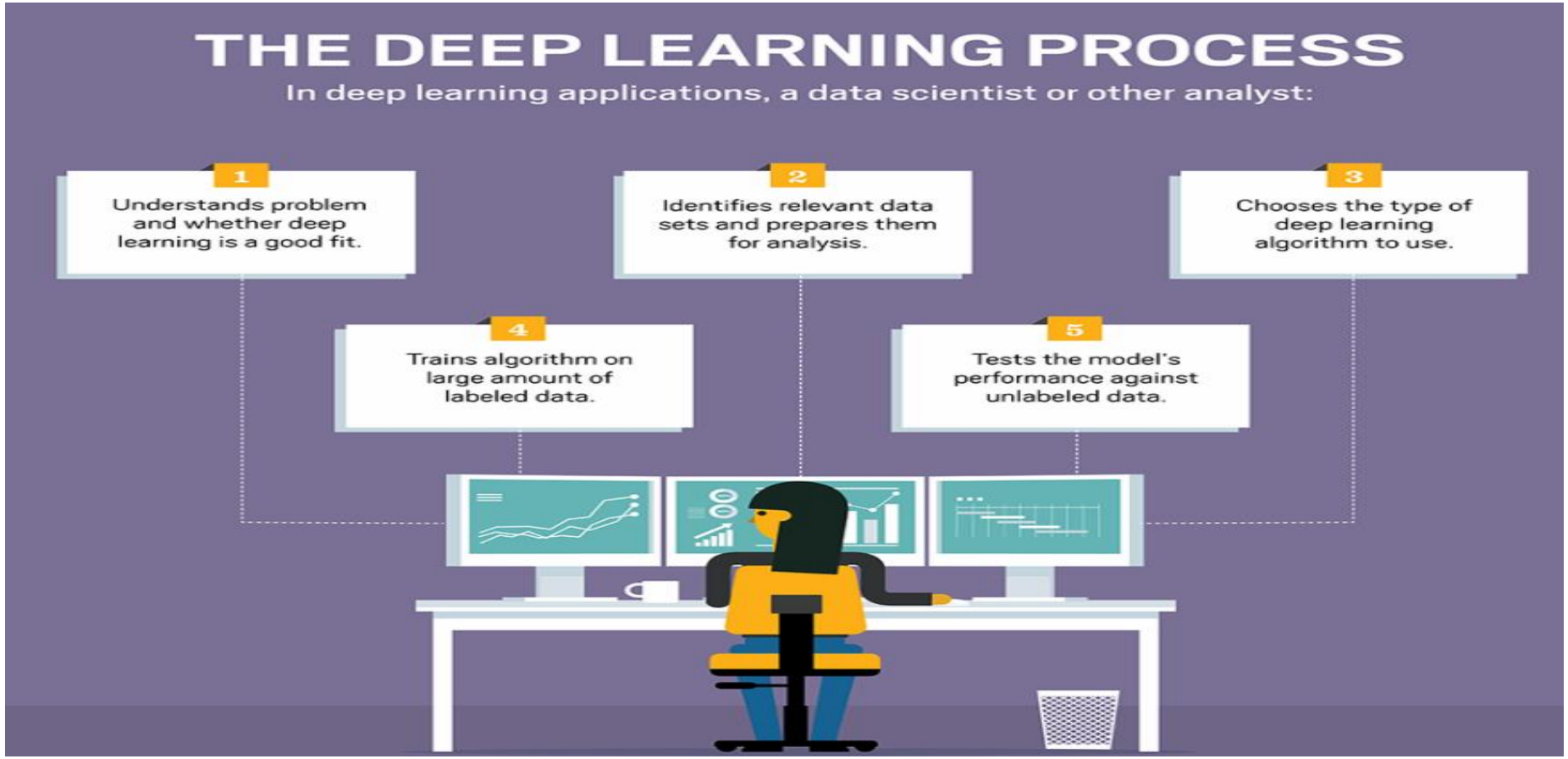


Image Source: https://searchenterpriseai.techtarget.com/definition/deep-learning-deep-neural-network

Course Introduction Continued.,

Need for Deep Learning (DL):

- Powered by Massive amounts of data
- Flexible system
- No need for a separate Feature Extraction module
- Tolerant to Imprecise Data
- Model Nonlinear Functions of higher complex nature
- No need for human supervision as like ML
- Can be Blended easily into Traditional Control applications

✓ Deep Learning is based on Artificial Neural Networks (ANN)

✓ Deep Learning uses Multiple layers of ANN(Hidden layers)

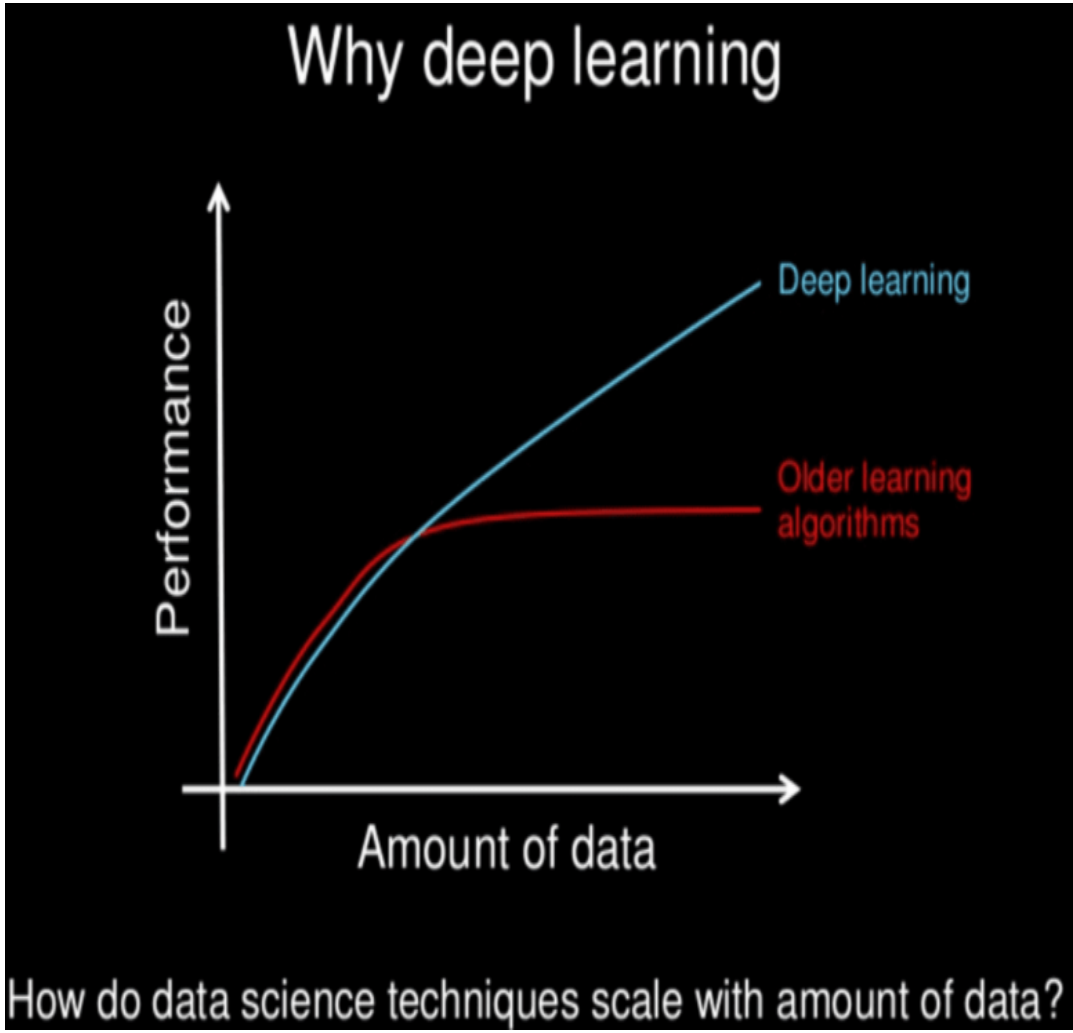


Image Source: https://machinelearningmastery.com/what-is-deep-learning/

Course Introduction Continued.,

Applications of Deep Learning (DL):

- Image/ Pattern Classification
- Image/ Pattern Association
- Self Driving Cars
- News Aggregation and Fraud News Detection
- Natural Language Processing
- Visual Recognition
- Fraud Detection
- Healthcare
- Automatic Handwriting Generation
- Automatic Game Playing
- Language Translations
- Demographic and Election Predictions

Course Outcomes

SECA4002 – DEEP LEARNING NEURAL NETWORKS

At The end of this Course, Student will be able to

CO1	Select suitable model parameters for different machine learning techniques
CO2	Evaluate the performance of existing deep learning models for various applications
CO3	Realign high dimensional data using reduction techniques
CO4	Analyze the performance of various optimization and generalization techniques in deep learning
CO5	Modify the existing architectures for domain specific applications
CO6	Develop a real time application using deep learning neural networks

Course Objectives

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Course Objectives:

- To present the mathematical, statistical and computational challenges of building neural networks
- To study the concepts of deep learning
- To introduce dimensionality reduction techniques
- To enable the students to know deep learning techniques to support real-time applications
- To examine the case studies of deep learning techniques

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 1: INTRODUCTION TO DEEP LEARNING

Introduction to machine learning- Linear models (SVMs and Perceptron's, logistic regression)- Intro to Neural Nets: What a shallow network computes- Training a network: loss functions, back propagation and stochastic gradient descent- Neural networks as universal function approximates

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 2: DEEP NETWORKS

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

UNIT 3: DEEP NETWORKS

Linear (PCA, LDA) and manifolds, metric learning - Auto encoders and dimensionality reduction in networks - Introduction to Convnet - Architectures – AlexNet, VGG, Inception, ResNet - Training a Convnet: weights initialization, batch normalization, hyperparameter optimization

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 4: OPTIMIZATION AND GENERALIZATION

Optimization in deep learning– Non-convex optimization for deep networks- Stochastic Optimization Generalization in neural networks- Spatial Transformer Networks- Recurrent networks, LSTM Recurrent Neural Network Language Models- Word-Level RNNs & Deep Reinforcement Learning - Computational & Artificial Neuroscience

UNIT 3: APPLICATIONS OF DEEP LEARNING

Imagenet- Detection-Audio WaveNet- Natural Language Processing Word2Vec - Joint Detection BioInformatics- Face Recognition- Scene Understanding- Gathering Image Captions

SECA4002 – DEEP LEARNING NEURAL NETWORKS

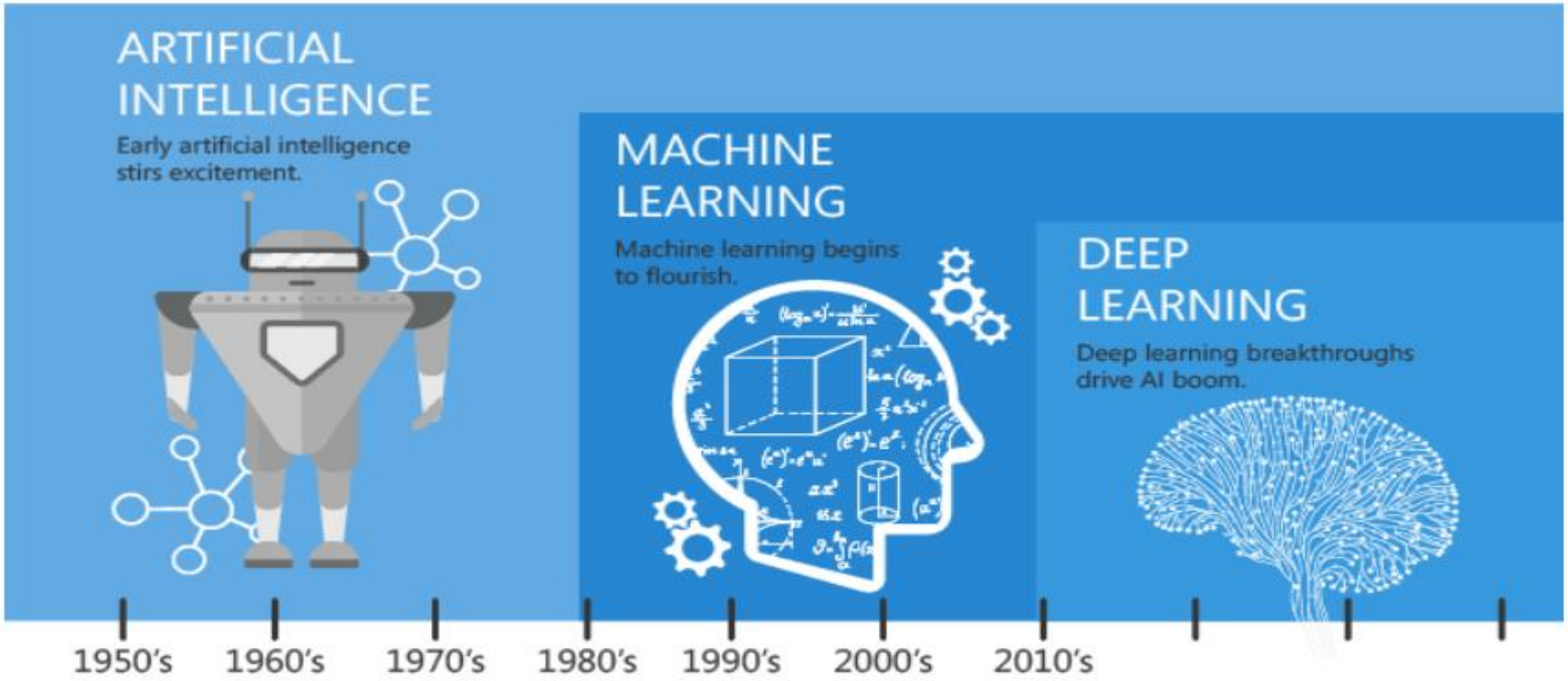
Recommended Text Books/ Reference Books

- Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
- Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
- Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

UNIT-1

Introduction To Machine Learning

Introduction - Machine Learning



Since an early flush of optimism in the 1950's, smaller subsets of artificial intelligence - first machine learning, then deep learning, a subset of machine learning - have created ever larger disruptions.

Image: Linked In | Machine Learning vs Deep learning

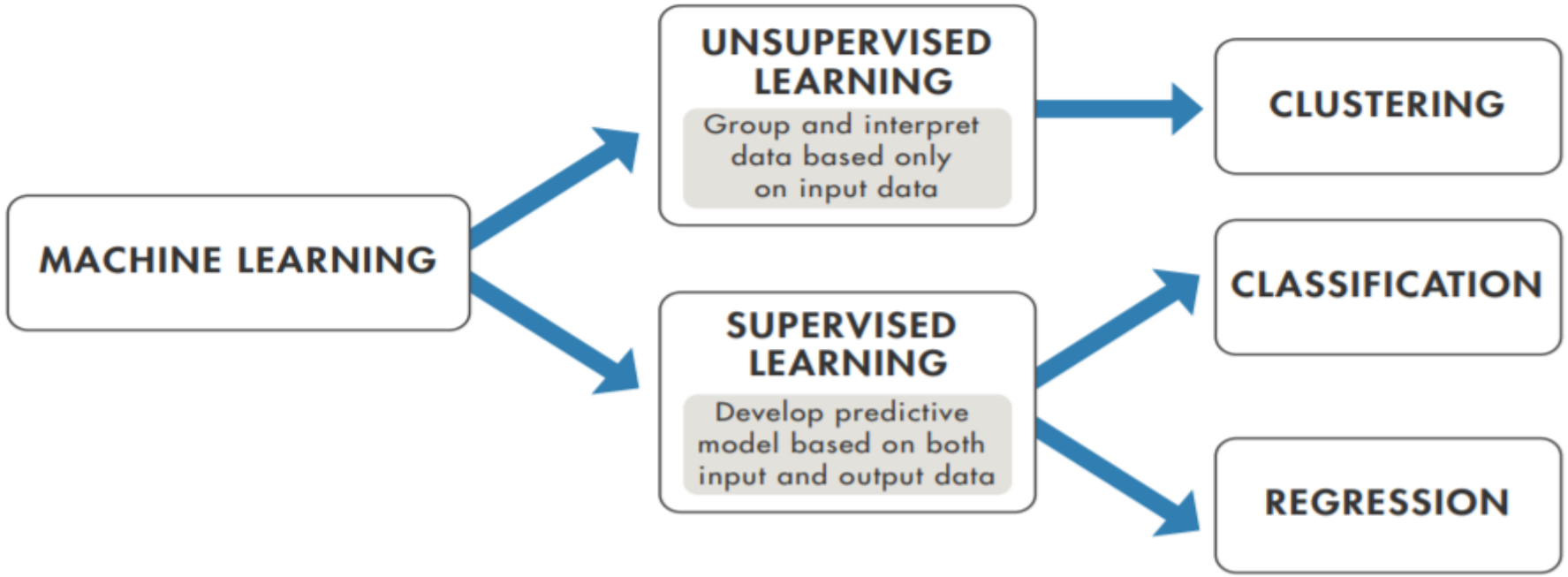
Introduction - Machine Learning

- The term Machine Learning was coined by Arthur Samuel in 1959
- Machine learning is a specific subset of AI that trains a machine how to learn

- Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. (Courtesy: www.ibm.com)
- Machine learning algorithms use computational methods to “learn” information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases.

Introduction - Machine Learning

How Machine Learning (ML) Works?

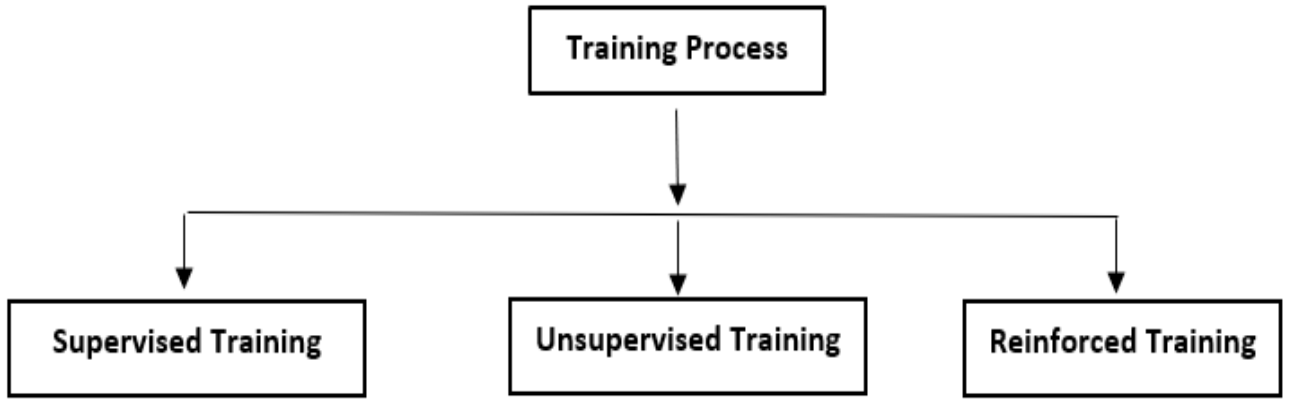


Courtesy: https://in.mathworks.com/content/dam/mathworks/ebook/gated/machine-learning-ebook-all-chapters.pdf

ARTIFICIAL NEURAL NETWORK(ANN)
Training Process

Training process of an Artificial Neural Networks can be broadly classified as

A. Supervised Training
B. Unsupervised Training
C. Reinforced Training



Supervised Learning

- Supervised Learning is the process of an learning(from the training dataset) can be thought of as a teacher who is supervising the entire learning process
- Learning algorithm iteratively makes predictions on the training data and is corrected by the “teacher”, and the learning stops when the algorithm achieves an acceptable level of performance(or the desired accuracy)

Supervised learning is the learning of the model where with input variable (say, X) and an output variable (say, Y) and an algorithm to map the input to the output.

Y = f(X)

Supervised Training Algorithms

- Any Algorithm which implements the supervised Training process is called **Supervised Training Algorithms**

Key Points:

- Each Input Vector is allotted with a corresponding Target vector
- The training Pair =[Input vector, Target Vector]
- Weights are adjusted to minimize the Difference between the Actual output & the Desired outputs
- Actual output = Output provided by the ANN
- Desired output = Output specified by the user (Target)

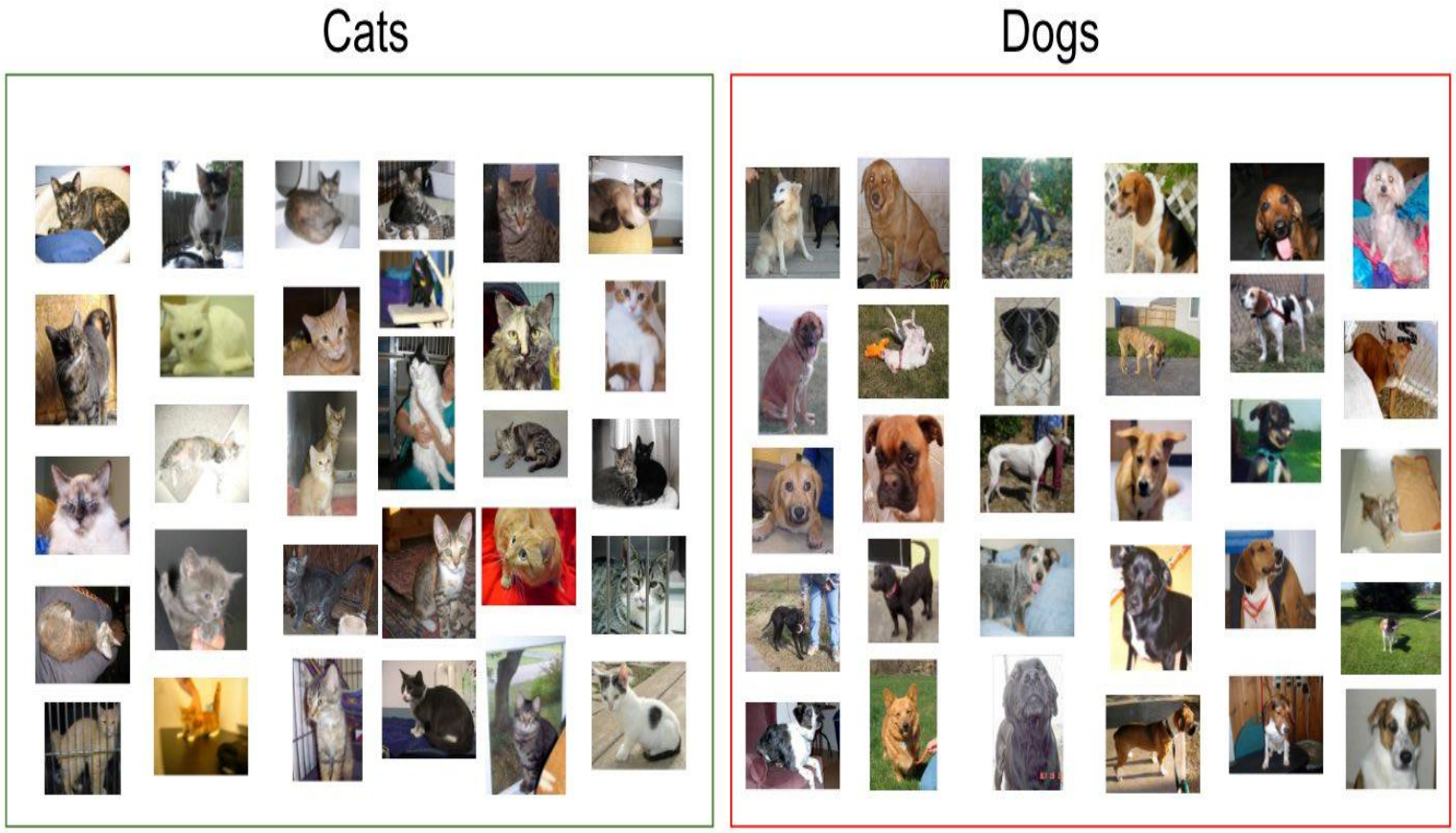
Supervised Training Algorithms- Continued

- **Supervised Training Procedure**

Explanation:

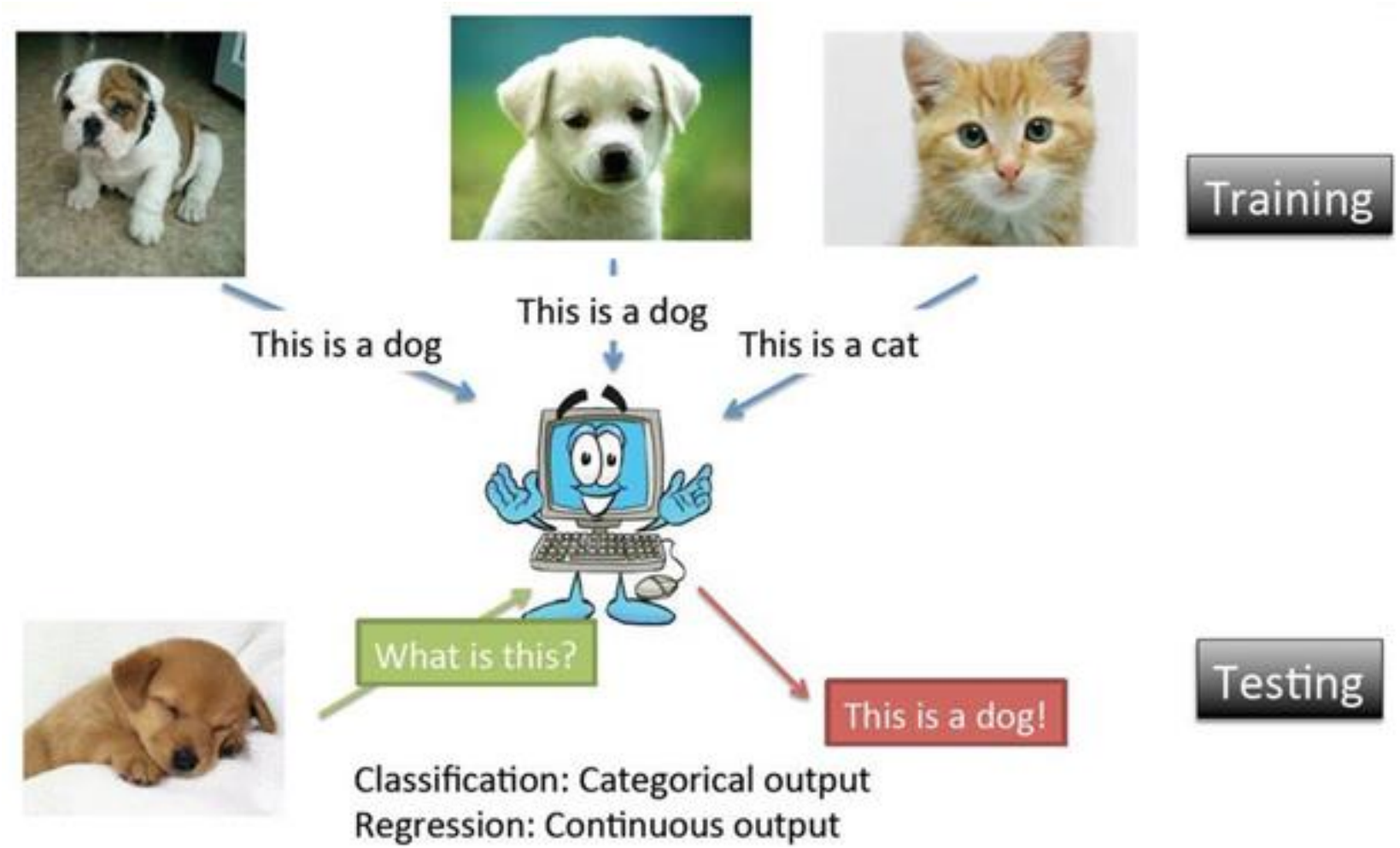
- Assume that a teacher is present
- Network is provided with the inputs & its associated outputs
- After processing based on Learning Law the network provides the output(Actual output)
- This out is compared with the Desired (Target) value
- If the desired value is Not Equal to the actual output ,then the teacher informs that an error has occurred error
- Now the teachers helps us in changing the weight (Based on learning Law)
- The process of Weight updating is continued until the desired output is Equal to the actual output (Stop Condition of the training Algorithm)

Supervised Learning



Sample of cats & dogs images from Kaggle Dataset

Supervised Learning



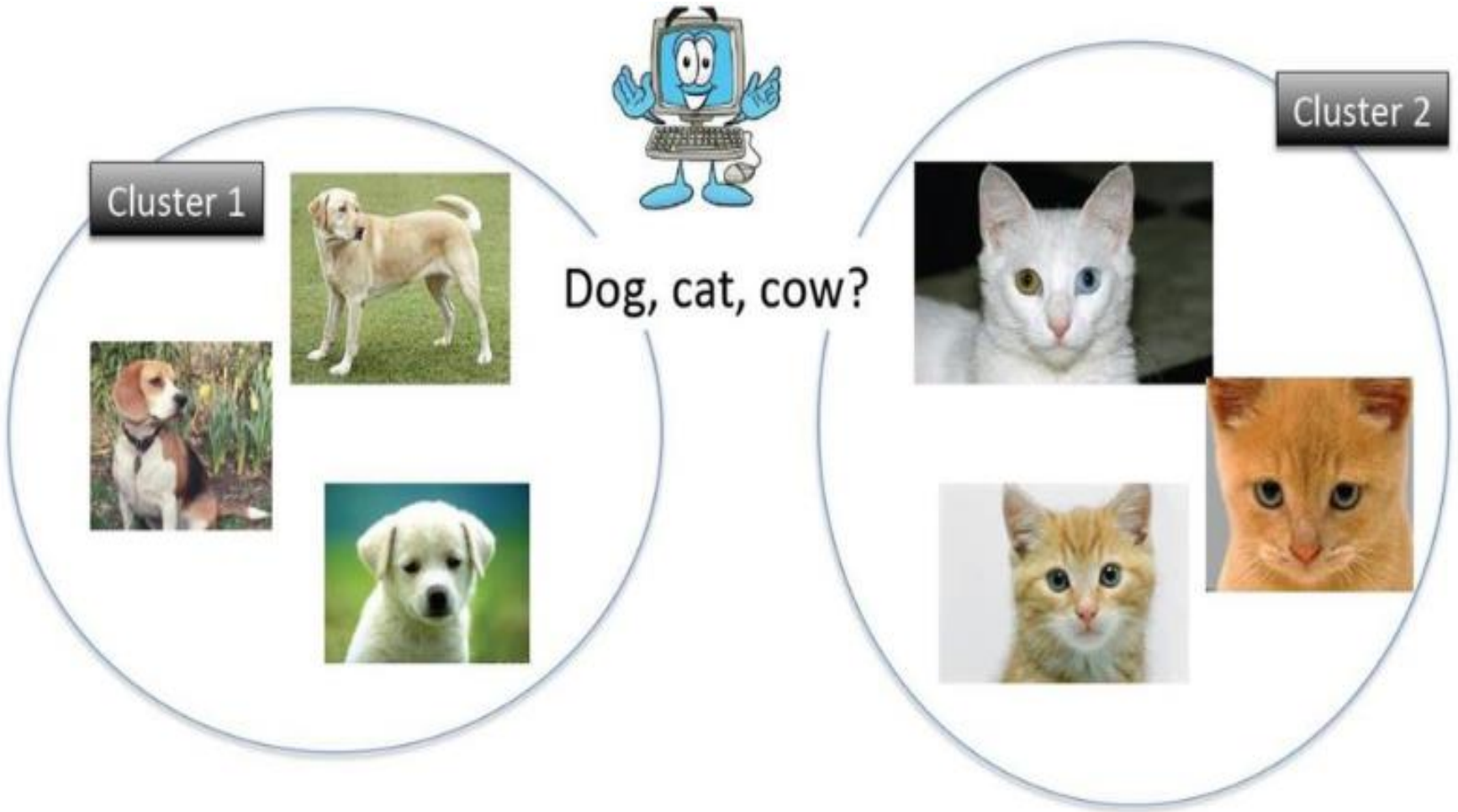
Unsupervised Training Algorithms

- Any Algorithm which implements the Unsupervised Training process is called **Unsupervised Training Algorithms**

Key Points:

- No Target is provided
- Let to discover the target pattern on its own
- No Teacher (Error detecting mechanism) is present
- Iterative process
- Consumes more time to converge(stop condition)
- More Accurate when compared with Supervised Training procedure
- Used to group unstructured data based on distinct features available within the data set

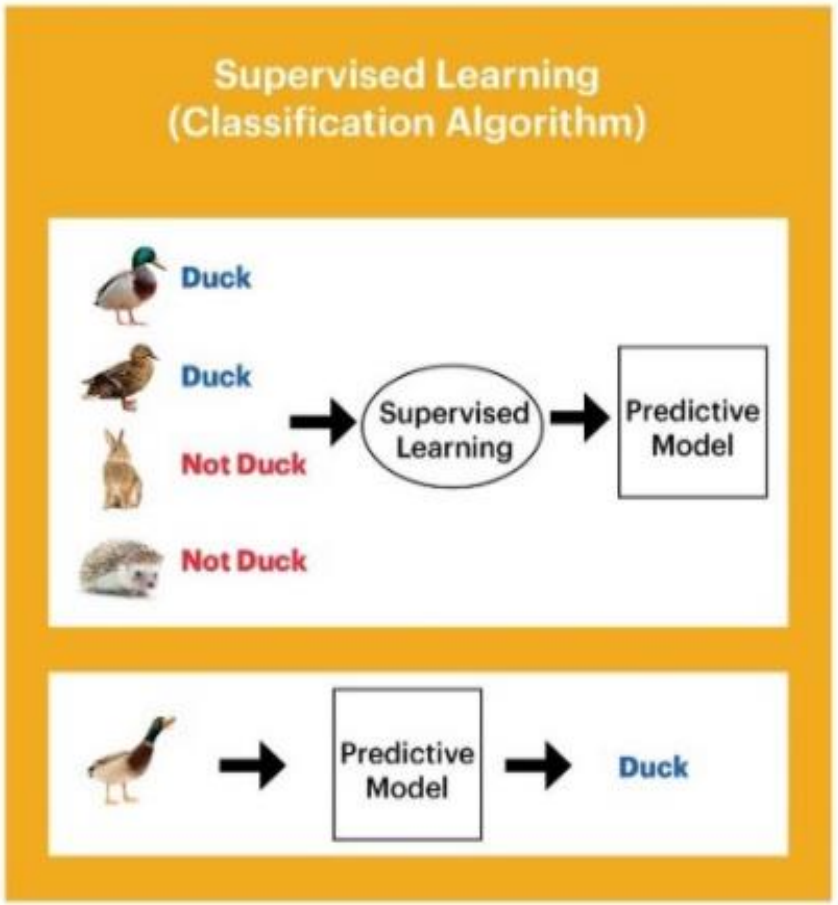
Unsupervised Learning

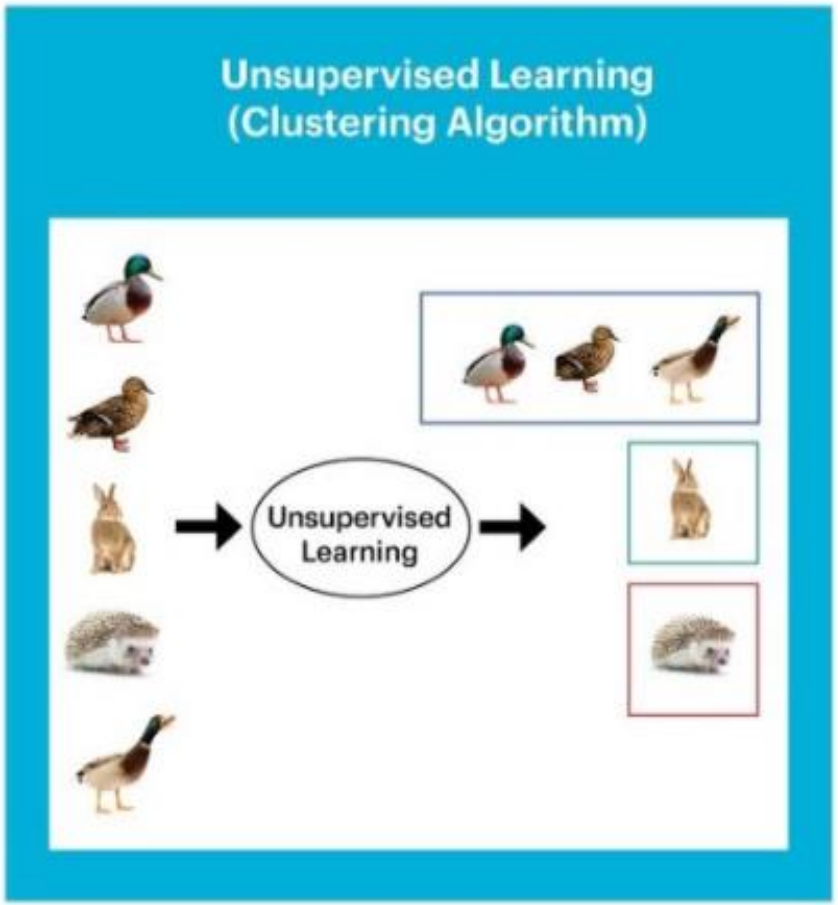


Comparison Between Supervised & Supervised

	SUPERVISED LEARNING	UNSUPERVISED LEARNING
Input Data	Uses Known and Labeled Data as input	Uses Unknown Data as input
Computational Complexity	Very Complex	Less Computational Complexity
Real Time	Uses off-line analysis	Uses Real Time Analysis of Data
Number of Classes	Number of Classes are known	Number of Classes are not known
Accuracy of Results	Accurate and Reliable Results	Moderate Accurate and Reliable Results

Comparison Between Supervised & Supervised





Reinforced Training Algorithms

- Any Algorithm which implements the Unique learning process in which the data's are not given, but generated by interactions with the environment is called **Reinforced Training Algorithms**

Points to Ponder:

- A teacher is present but does not specifies the Target
- The teacher just specifies the obtained output has correct or not
- Error detection is present but no error correction
- A reward is provided if the output is correct
- A penalty is provided for wrong answer
- Aim is to maximize the reward process by Trail & error process
- The network has to exploit what it already knows in order to obtain reward
- It also has to explore new unknown searches in order to make better action selections in the future

Reinforced Learning

- Reinforcement learning differs from the supervised learning
- In supervised learning the training data has the answer key with it so the model is trained with the correct answer itself
- In reinforcement learning, there is no answer but the reinforcement agent decides the action to be done based on the current state/Environment
- Agent interprets based on the given input
- If the action is good/best then a reward is provided
- Unsupervised learning is done
- Else a penalty is given
- Supervised learning is done i.e weights are changed based on the error

Reinforced Learning

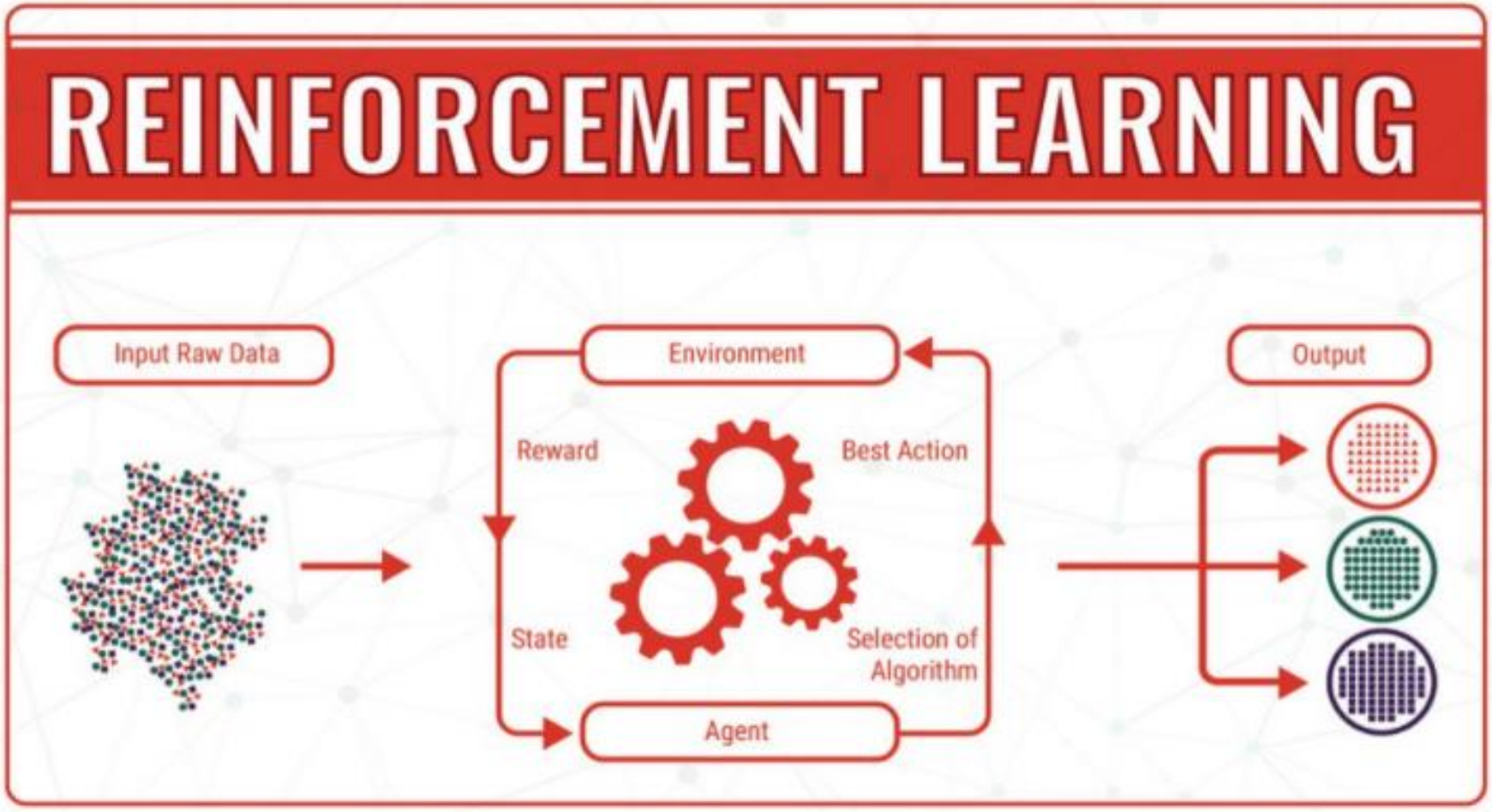
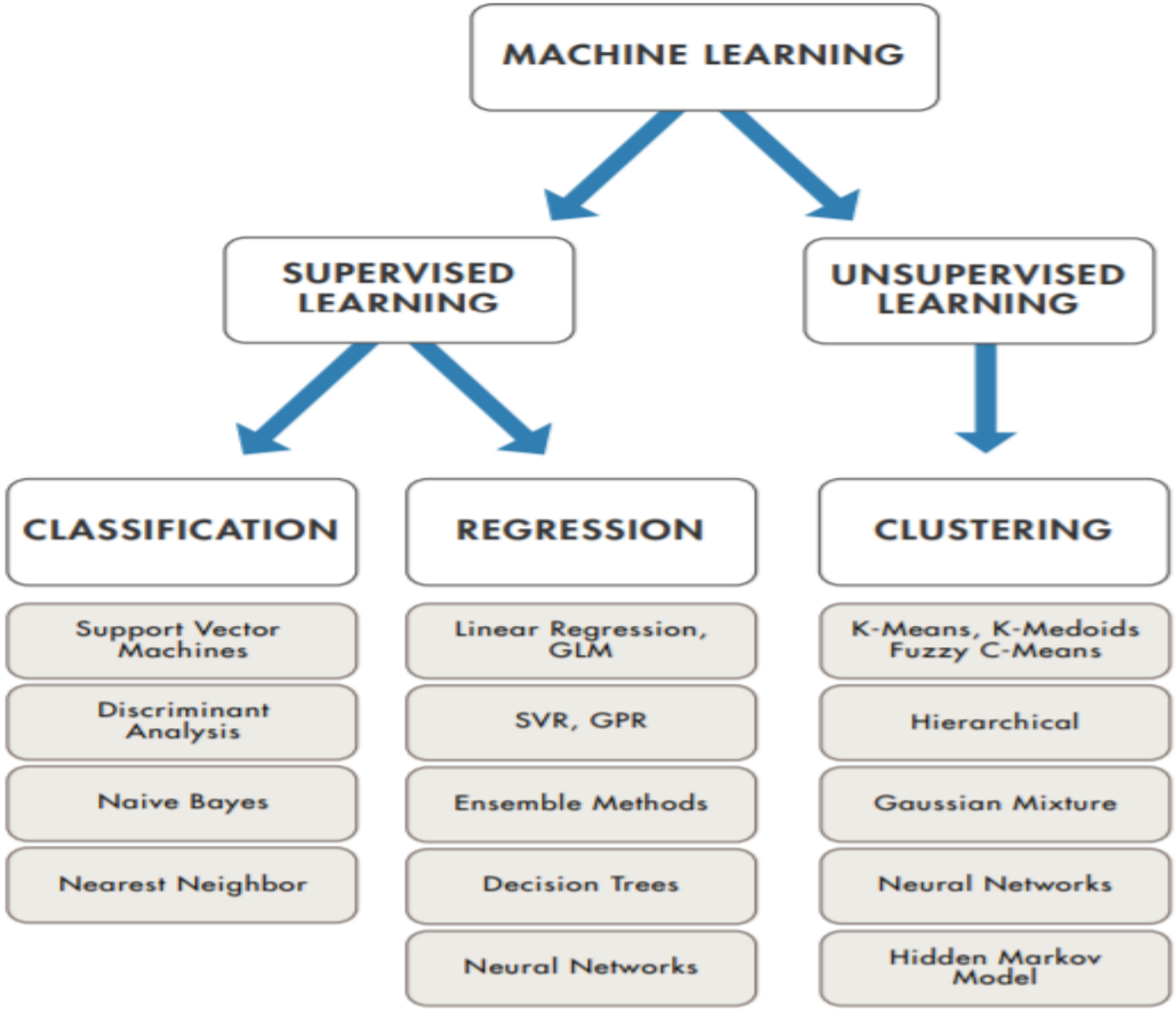


Image source: https://twitter.com/ipfconline1/status/1034281946380218369/photo/1

Introduction – Contd.,

How Do You Decide Which Algorithm to Use?



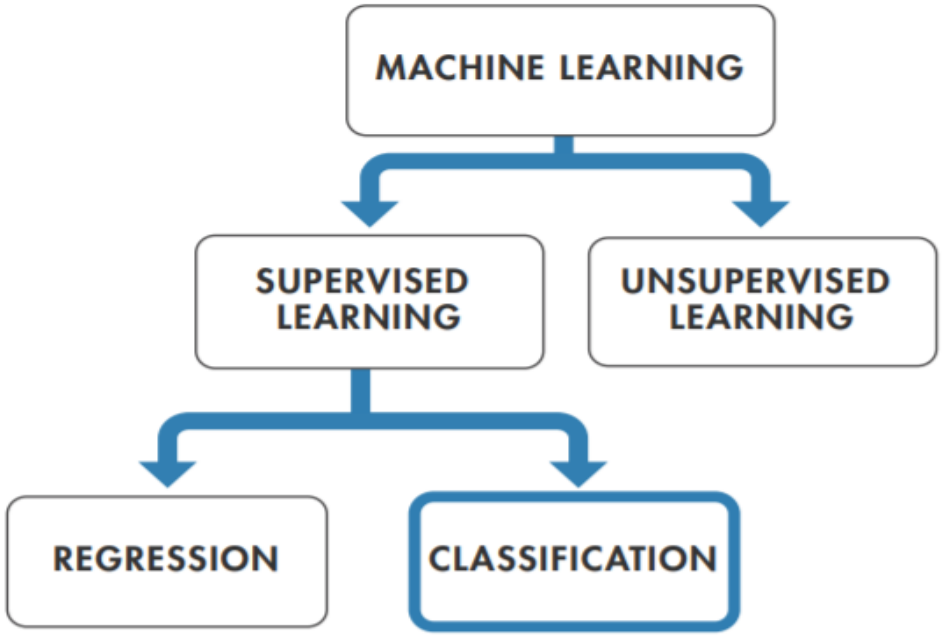
Courtesy: https://in.mathworks.com/content/dam/mathworks/ebook/gated/machine-learning-ebook-all-chapters.pdf

Questions to Consider Before You Start

Every machine learning workflow begins with three questions:

- What kind of data are you working with?
- What insights do you want to get from it?
- How and where will those insights be applied?

Your answers to these questions help you decide whether to use supervised or unsupervised learning.



Choose supervised learning if you need to train a model to make a prediction--for example, the future value of a continuous variable, such as temperature or a stock price, or a classification—for example, identify makes of cars from webcam video footage.

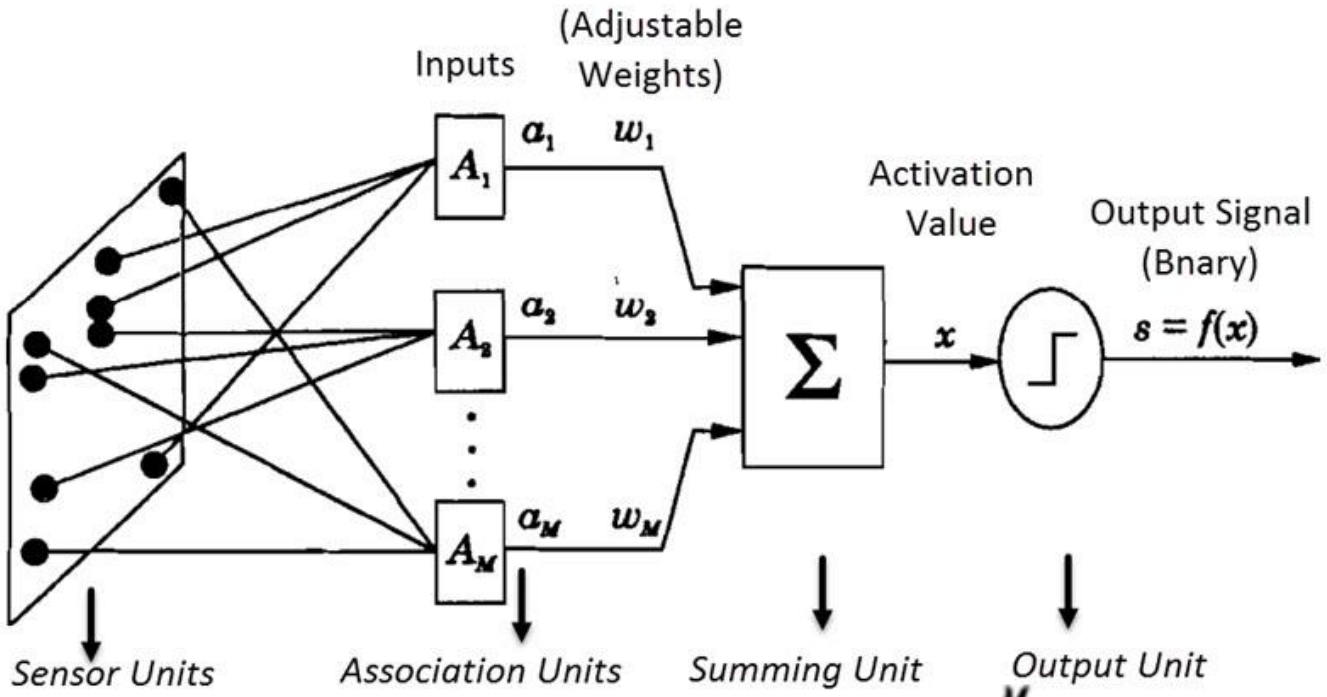
Choose unsupervised learning if you need to explore your data and want to train a model to find a good internal representation, such as splitting data up into clusters.

Courtesy: https://in.mathworks.com/content/dam/mathworks/ebook/gated/machine-learning-ebook-all-chapters.pdf

Models of ANN contd.

I. Perceptron Model

I. a. Single Layer Perceptron Model



Activation Function $x = \sum_{i=1}^{M} w_i a_i - \theta$

Output Signal $s = f(x)$

Error $\delta = b - s$

Weight change $\Delta w = \eta \delta a_i$

Single Layer Perceptron Model Algorithm

Algorithm:

1. Initialize the weights to some small random values near to Zero
2. Apply the Input : Output Training patterns (Vector Pairs)
3. Calculate the summing part value Net = Σ aiwi-θ
4. Apply Activation function and calculate the output

$$F(Net) = \begin{cases} 1 & \text{if Net} >= Q \\ 0 & \text{if Net} < Q \end{cases}$$

5. Calculate the Error $\delta = b_i - S_i$, if error is present then update the weight for that link using

$$\Delta w_i = \eta\,[b_i - \mathbf{sgn}(\mathbf{w}_i^T \mathbf{a})]\, a_j$$

$$= \eta\,(b_i - s_i)\, a_j\,, \qquad \text{for } j = 1, 2, \ldots, M$$

Single Layer Perceptron Model Algorithm

Algorithm (Continued)

6. Similarly change the Bias values
   New Bias = old Bias + change in Bias
7. Now repeat from step 3

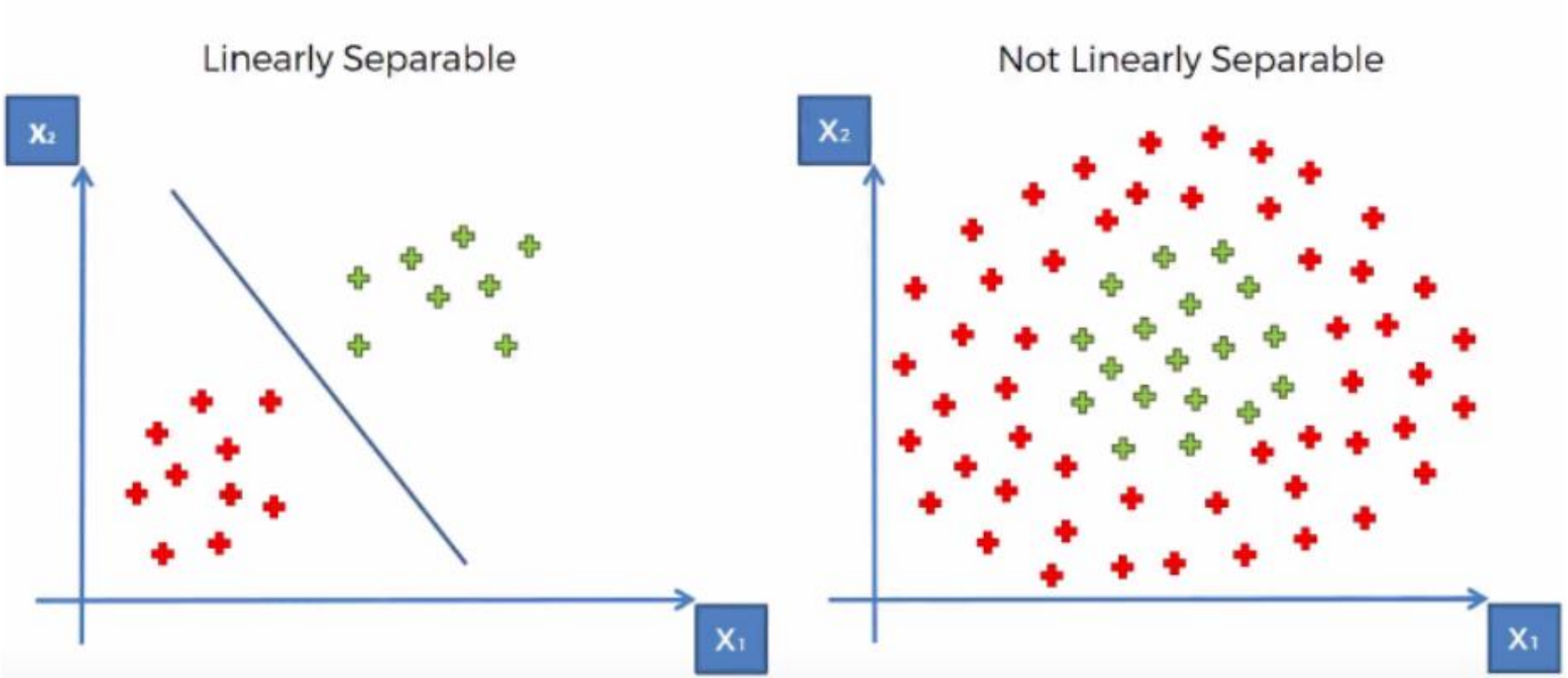
8. Check for error ,if error is present repeat the process ,Else Stop

Limitations:

- Uses only Binary Activation function
- Can be used only for Linear Networks
- Since uses Supervised Learning ,Optimal Solution is provided
- Training Time is More
- Cannot solve Linear In-separable Problem

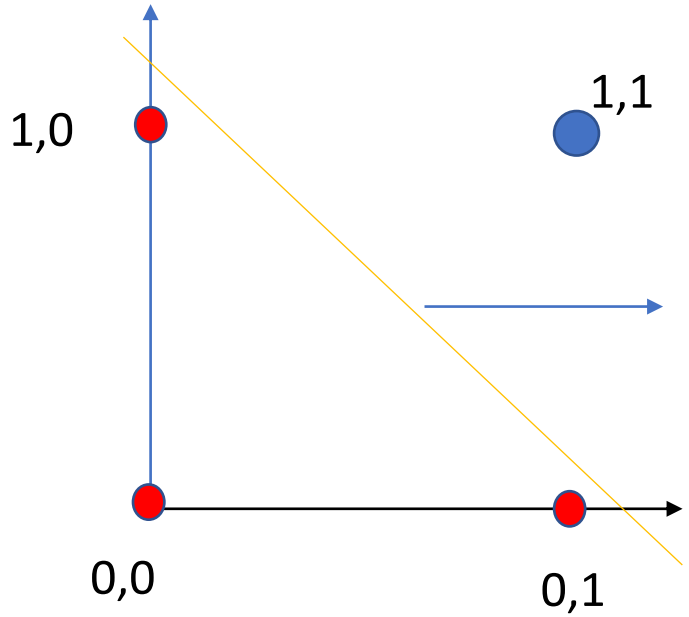
In order to solve these limitations, Multi Layer Perceptron is Implemented

Linear separable & Linear Inseparable issue



- Let us consider AND Gate as example for Linear separable and XOR Gate for Linear Inseparability

Linear separable & Linear Inseparable issue



Decision Boundary Line

AND Gate Table:

Input 1(a1)	Input 2 (a2)	Output(S)
0	0	0
0	1	0
1	0	0
1	1	1

$B_0+w_1x_1+w_2x_2 =0$

$w_2x_2 = - B_0 - w_1x_1$

$X_2 = \frac{-B_0}{w_2} - \frac{w_1 x_1}{w_2}$ → This equation is similar to Line equation

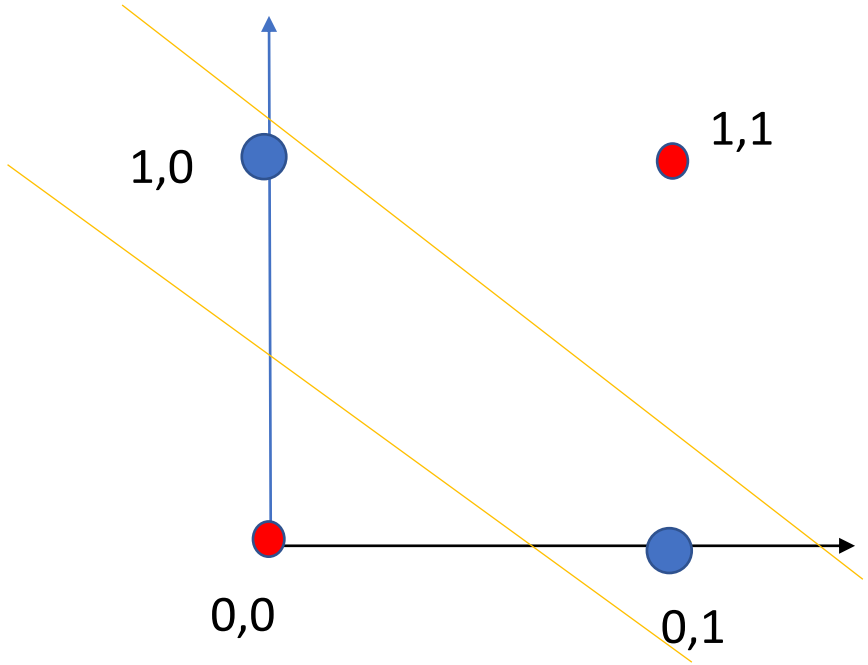
- Let us consider AND Gate as example for Linear separable analysis

● → Denotes a Binary Value of '1'

● → Denotes a Binary Value of '0'

Linear separable & Linear Inseparable issue

- Let us consider XOR Gate for Linear Inseparability



XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

Denotes a Binary Value of '1'

Denotes a Binary Value of '0'

- Here a single decision Line cannot separate the Zeros and Ones Linearly
- At least Two lines are required to separate Zeros and Ones

- With Single Layer networks we can have only one decision Line, so to solve non Linearity or Linear Inseparable problems Single Layer Networks cannot be used

To Overcome this Problem We use Convex Regions

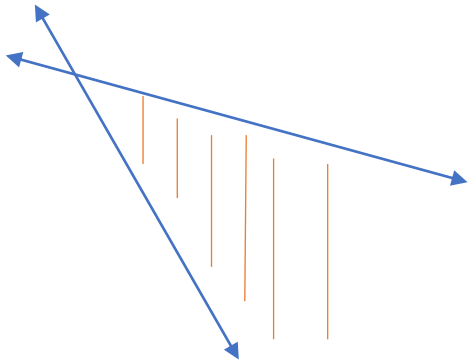
Linear separable & Linear Inseparable issue

- **Convex Region:**

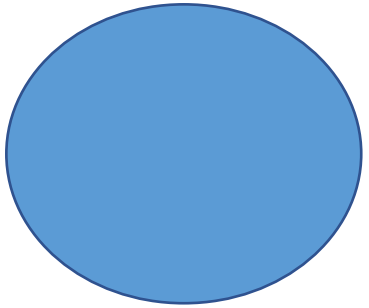
Select any Two points in a region and draw a straight line between these two points. If the points selected and the lines joining them both lie inside the region then that region is known as convex regions

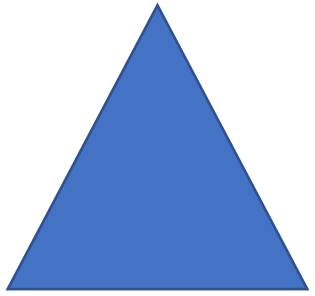
- Types:

(a) Open Convex region



(b) Closed Convex Region

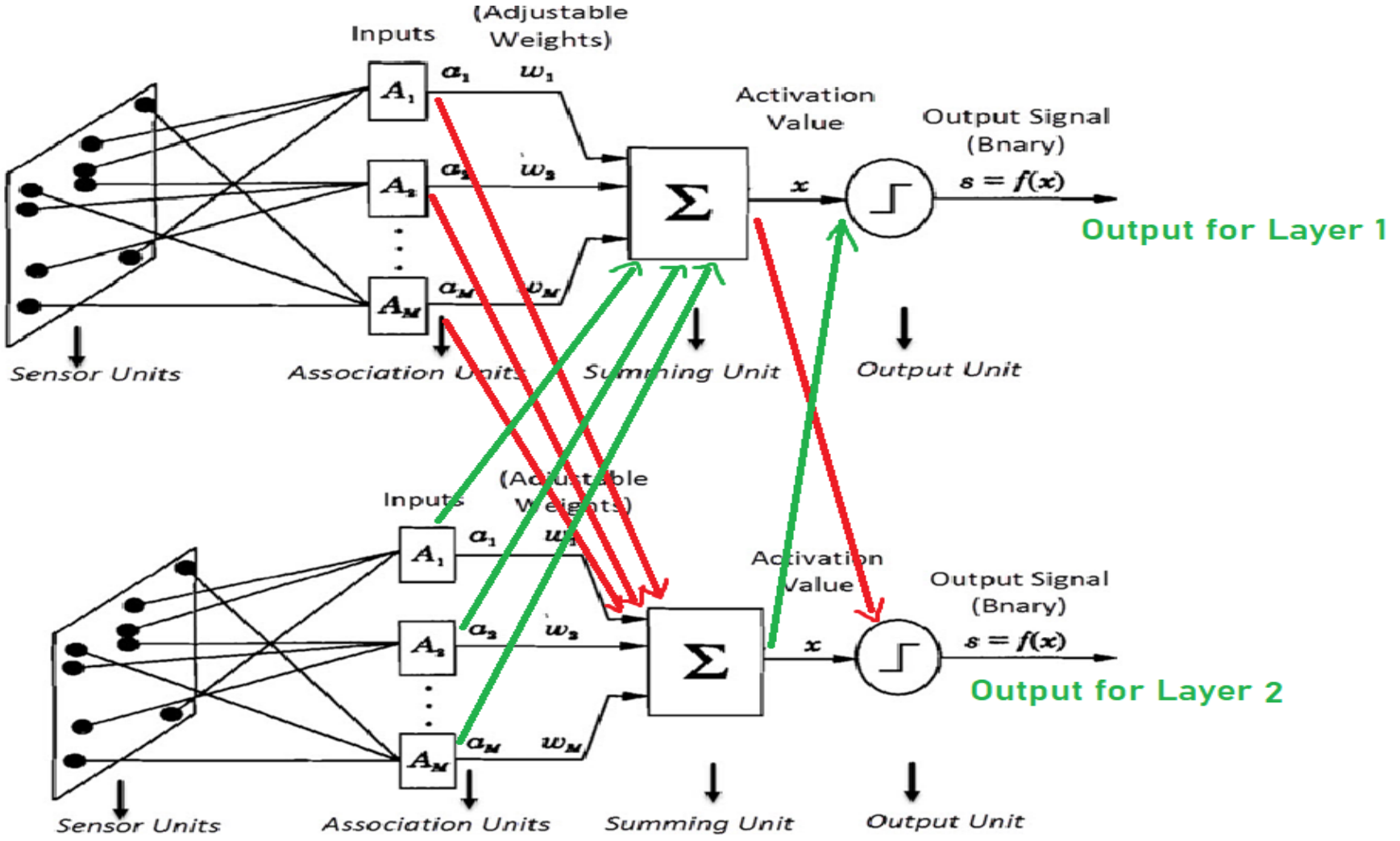




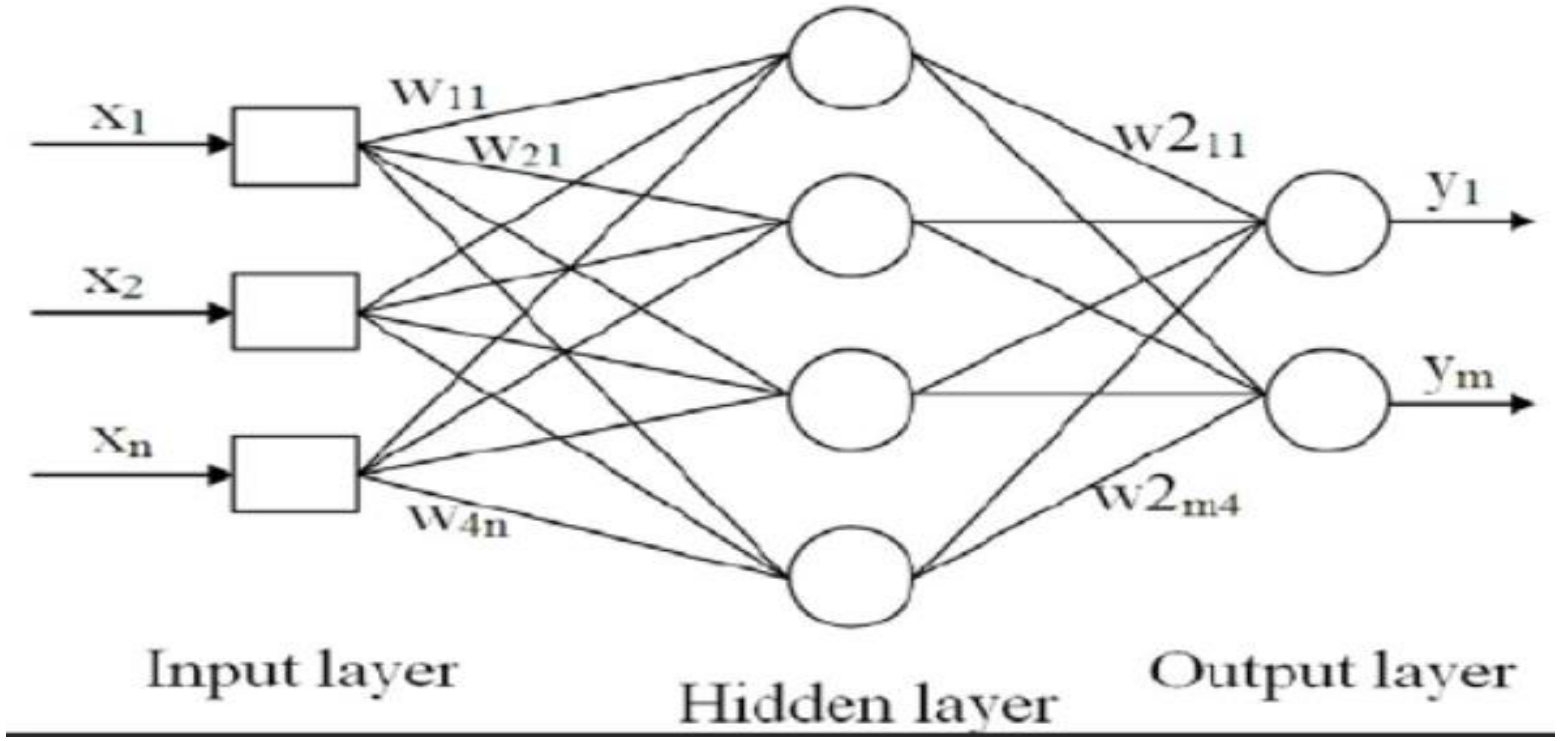
✓ **To over come Linear Inseparable problem we need at least minimum two or more decision lines, Hence we move towards Multi Layer Networks**

I. Perceptron Model

II. b. Multi Layer Perceptron Model



I. b. Multi Layer Perceptron Model



Activation Function $x = \sum_{i=1}^{M} w_i a_i - \theta$

Output Signal $s = f(x)$

Error $\delta = b - s$

Weight change $\Delta w = \eta \delta a_i$

II. b. Multi Layer Perceptron Model

Algorithm:

1. Initialize the weights (W_i) & Bias (B_0) to small random values near Zero
2. Set learning rate η or α in the range of "0" to "1"
3. Check for stop condition. If stop condition is false do steps 3 to 7
4. For each Training pairs do step 4 to 7
5. Set activations of Output units: $x_i = s_i$ for i=1 to N
6. Calculate the output Response

   $y_{in} = b_0 + \Sigma x_i w_i$
7. Activation function is

For Multi Layer networks, based on the number of layers steps 6 & 7 are repeated

$$Y = F(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \Theta \\ 0 & \text{if } y_{in} <= \Theta \\ -1 & \text{if } y_{in} <- \Theta \end{cases}$$

- Bipolar Activation is used

I. b. Multi Layer Perceptron Model

Algorithm: (Continued)

8. If the Targets is (not equal to) = to the actual output (Y), then update weights and bias based on Perceptron Learning Law

$W_{i\,(new)} = W_{i\,(old)} + \text{Change in weight vector}$

Change in weight vector $= \eta t_i x_i$

Where η = Learning Rate

t_i = Target output of i^{th} unit

x_i = i^{th} Input vector

$b_{0(new)} = b_{0\,(old)} + \text{Change in Bias}$

Change in Bias $= \eta t_i$

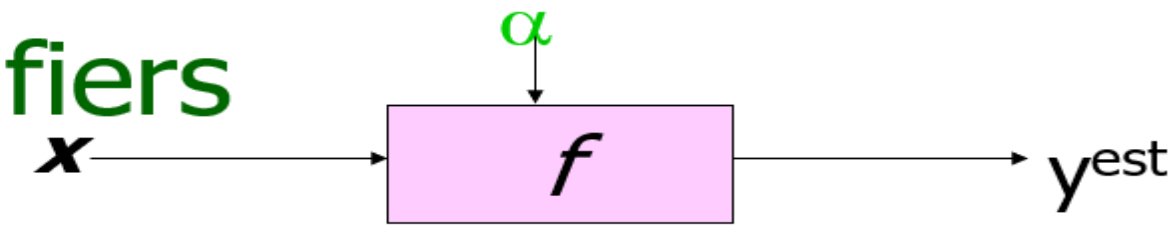
Else $W_{i\,(new)} = W_{i\,(old)}$

$b_{0(new)} = b_{0\,(old)}$

9. Test for Stop condition

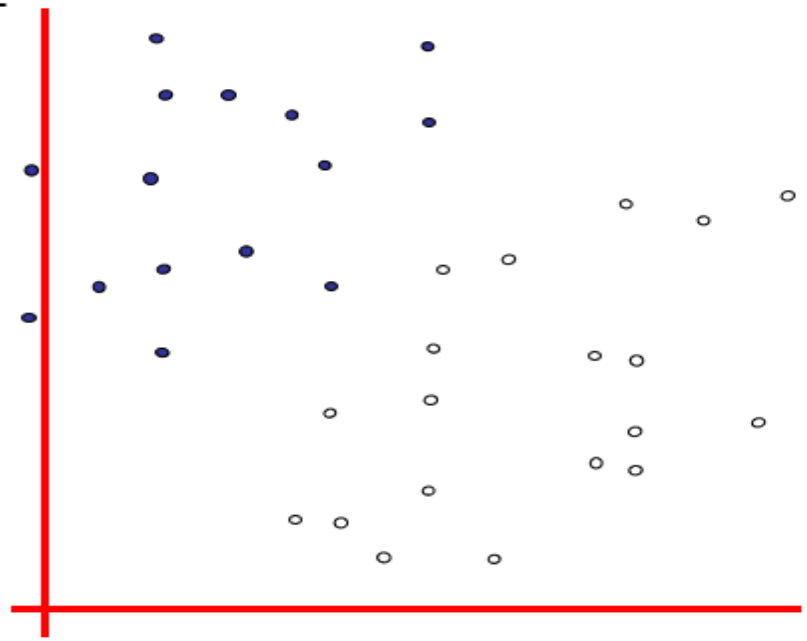
Support Vector Machines (SVM)

Linear Classifiers



$f(x,w,b) = sign(w.x - b)$

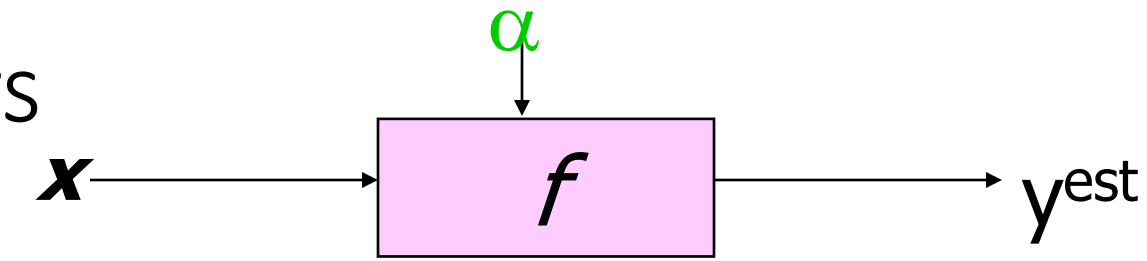
- denotes +1
- denotes -1



How would you classify this data?

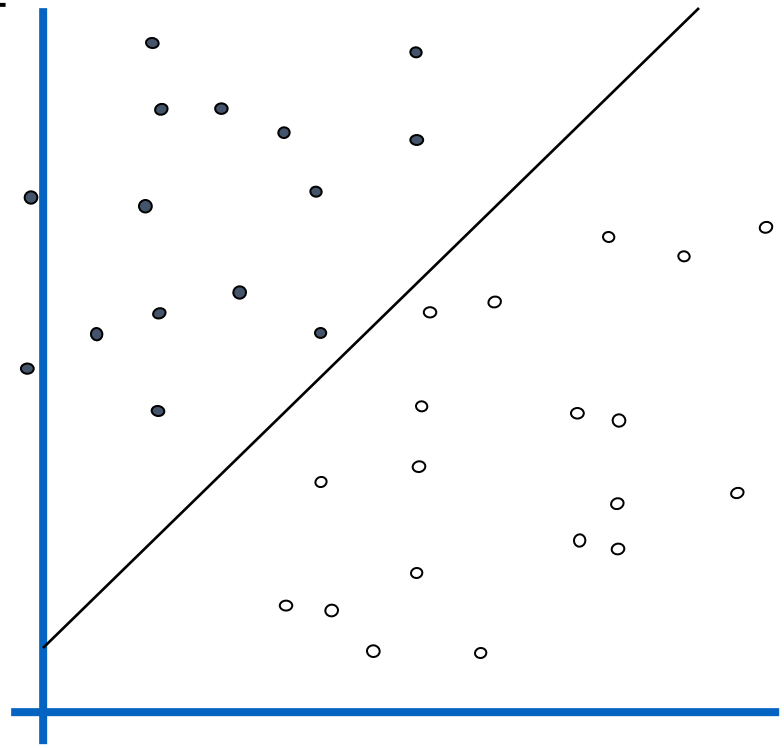
Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Linear Classifiers



$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

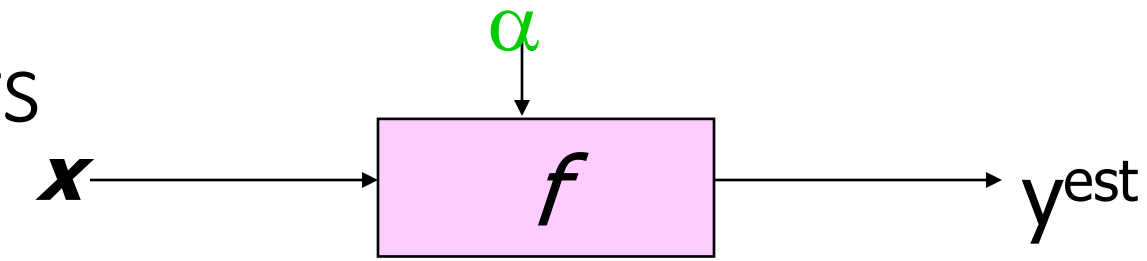
- denotes +1
- denotes -1



How would you classify this data?

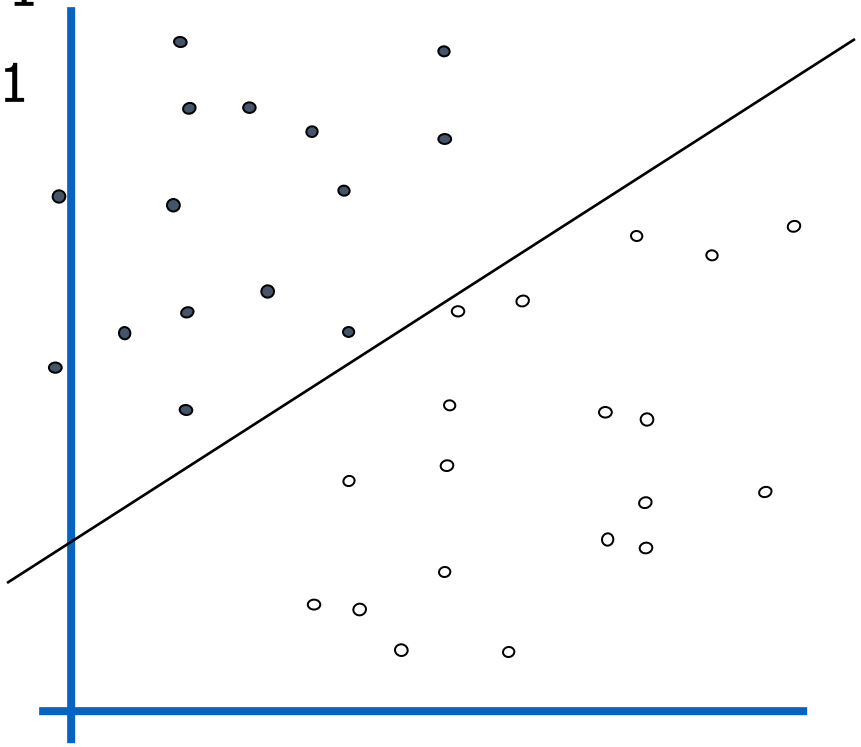
Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Linear Classifiers



$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

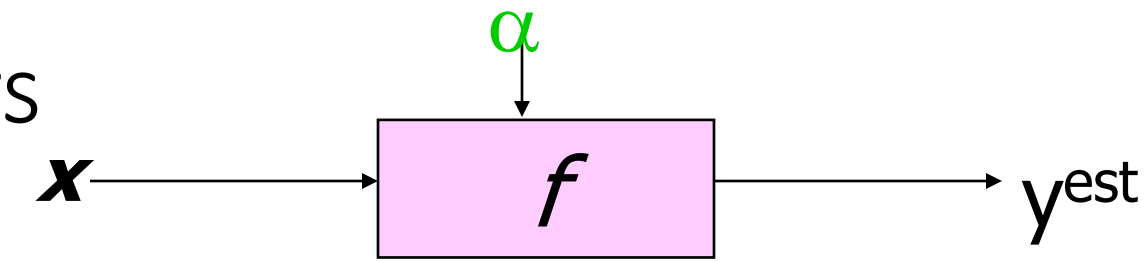
- denotes +1
- denotes -1



How would you classify this data?

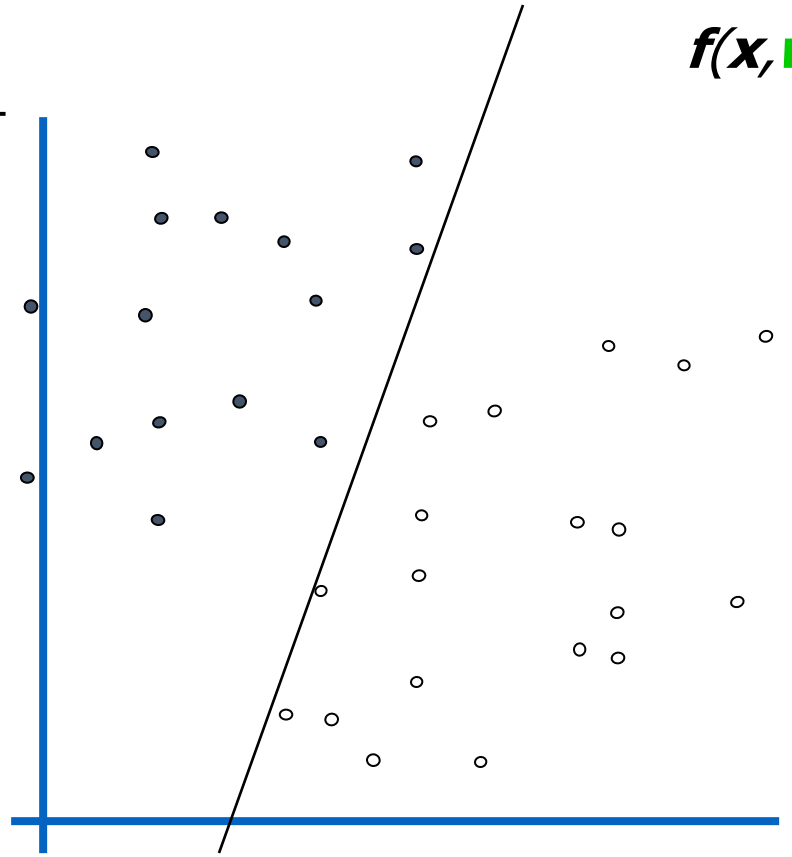
Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Linear Classifiers



- denotes +1
- denotes -1

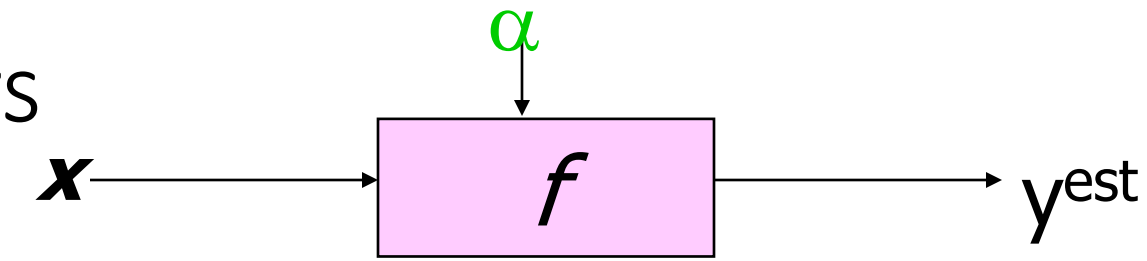
$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$



How would you classify this data?

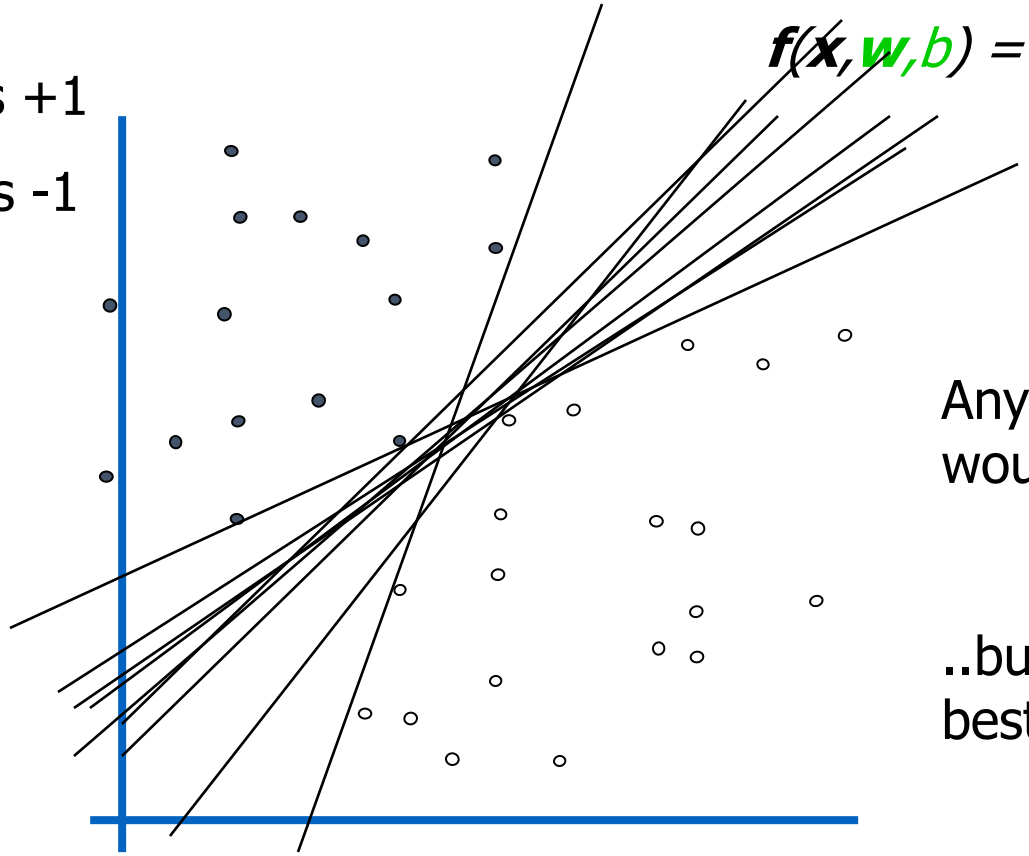
Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Linear Classifiers



- denotes +1
- denotes -1

$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

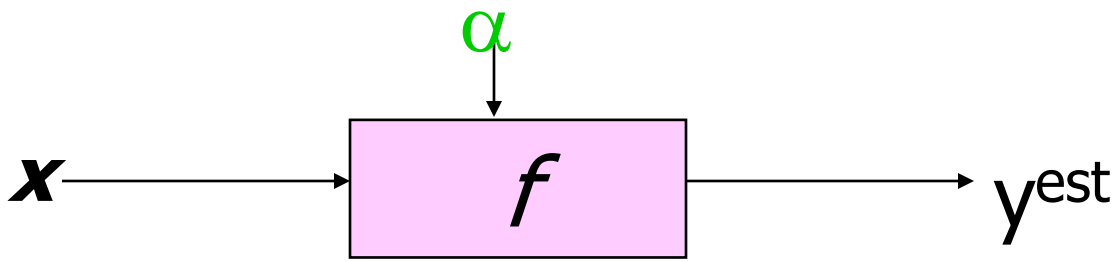


Any of these would be fine..

..but which is best?

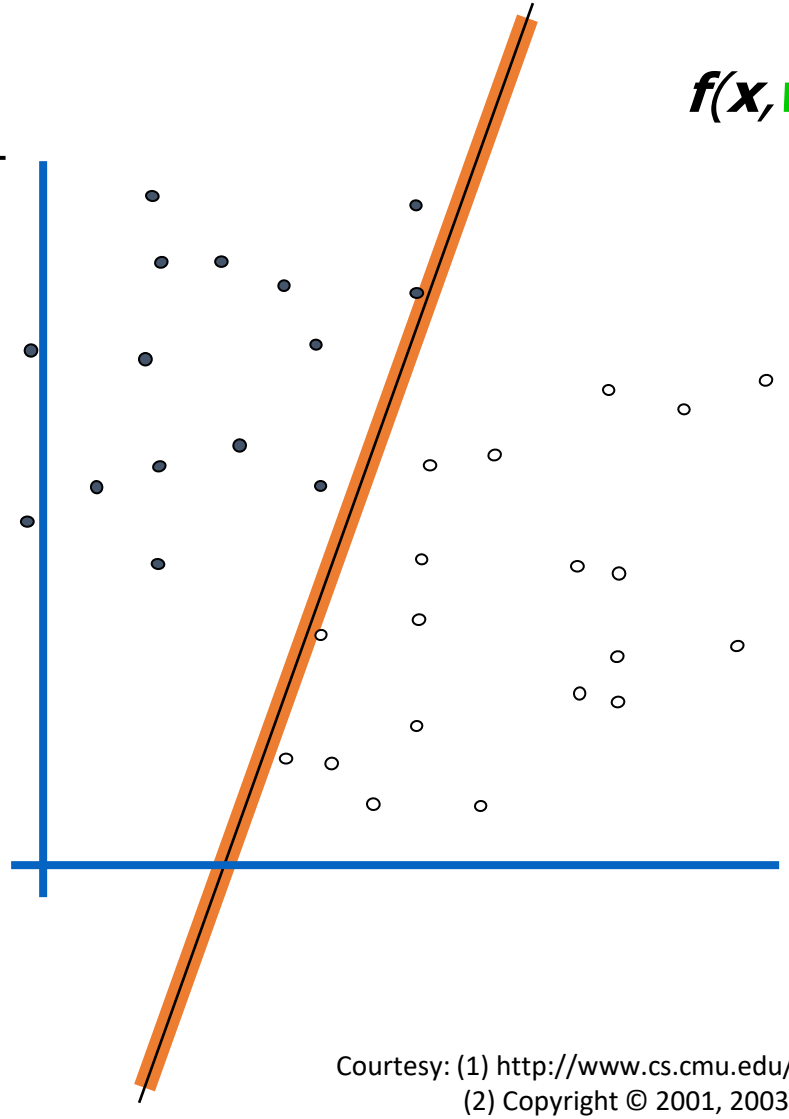
Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Classifier Margin



$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}.\ \mathbf{x} - b)$

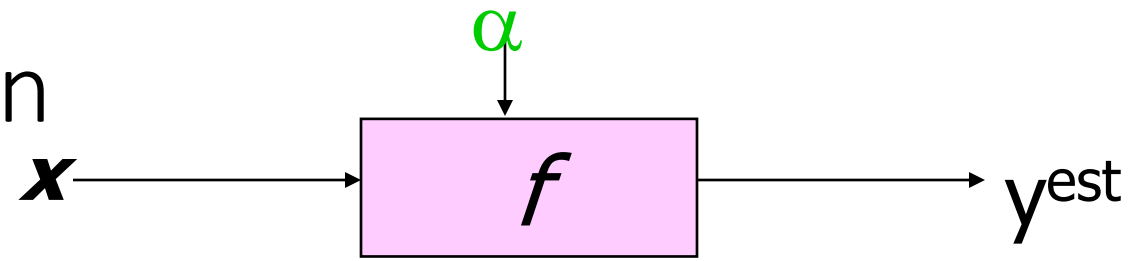
- denotes +1
- denotes -1



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

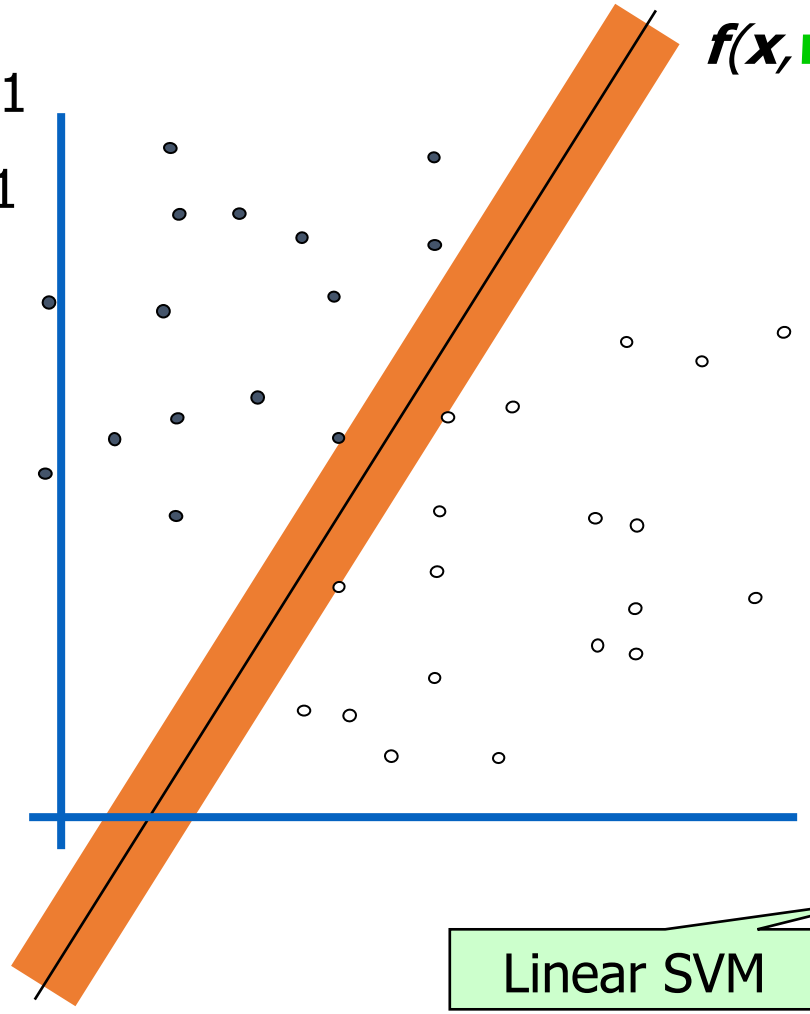
Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1



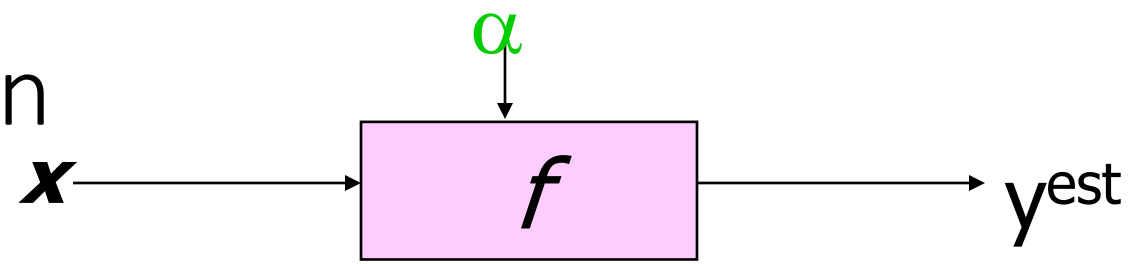
The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Maximum Margin



$f(\mathbf{x},\mathbf{w},b) = sign(\mathbf{w}.\mathbf{x} - b)$

- denotes +1
- denotes -1

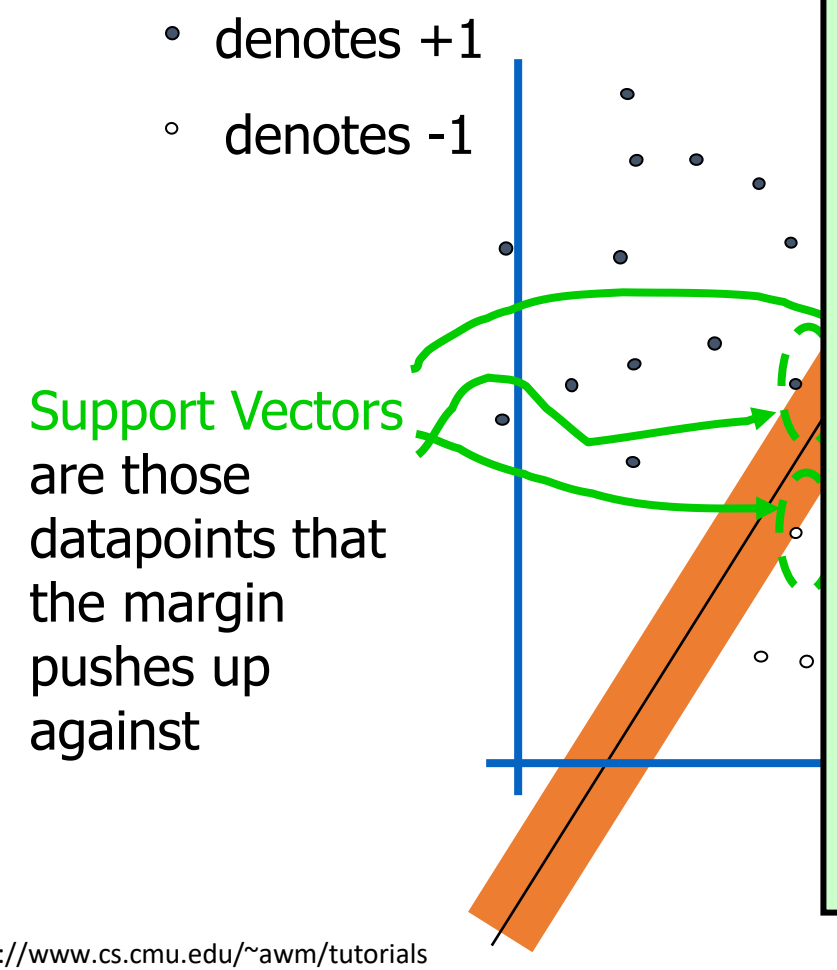
Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin.

This is the simplest kind of SVM (Called an LSVM)

Linear SVM

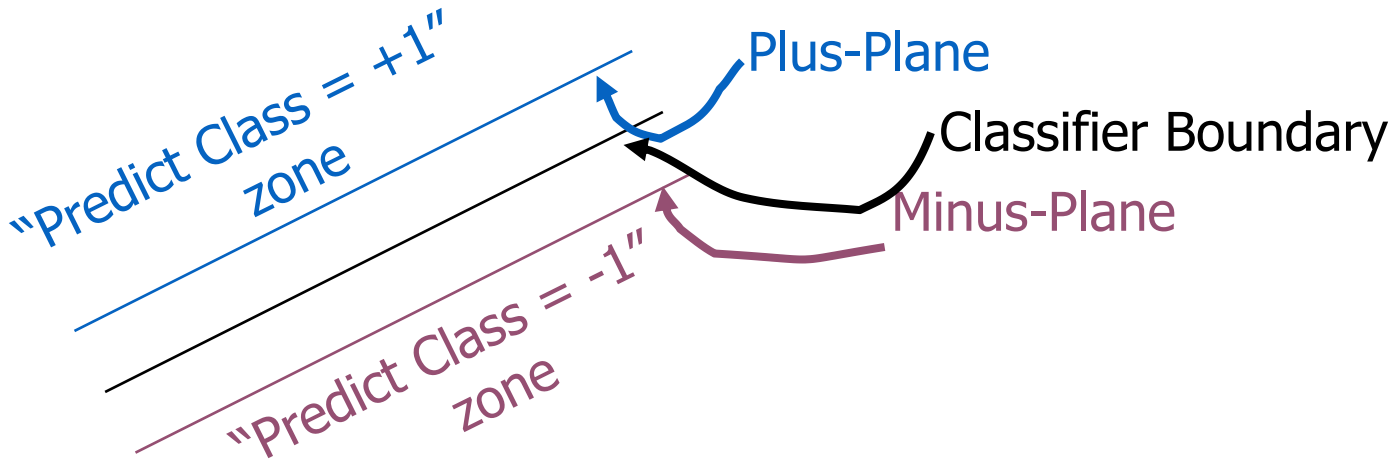
Why Maximum Margin?



1. Intuitively this feels safest.
2. If we've made a small error in the location of the boundary (it's been jolted in its perpendicular direction) this gives us least chance of causing a misclassification.
3. LOOCV is easy since the model is immune to removal of any non-support-vector datapoints.
4. There's some theory (using VC dimension) that is related to (but not the same as) the proposition that this is a good thing.
5. Empirically it works very very well.

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

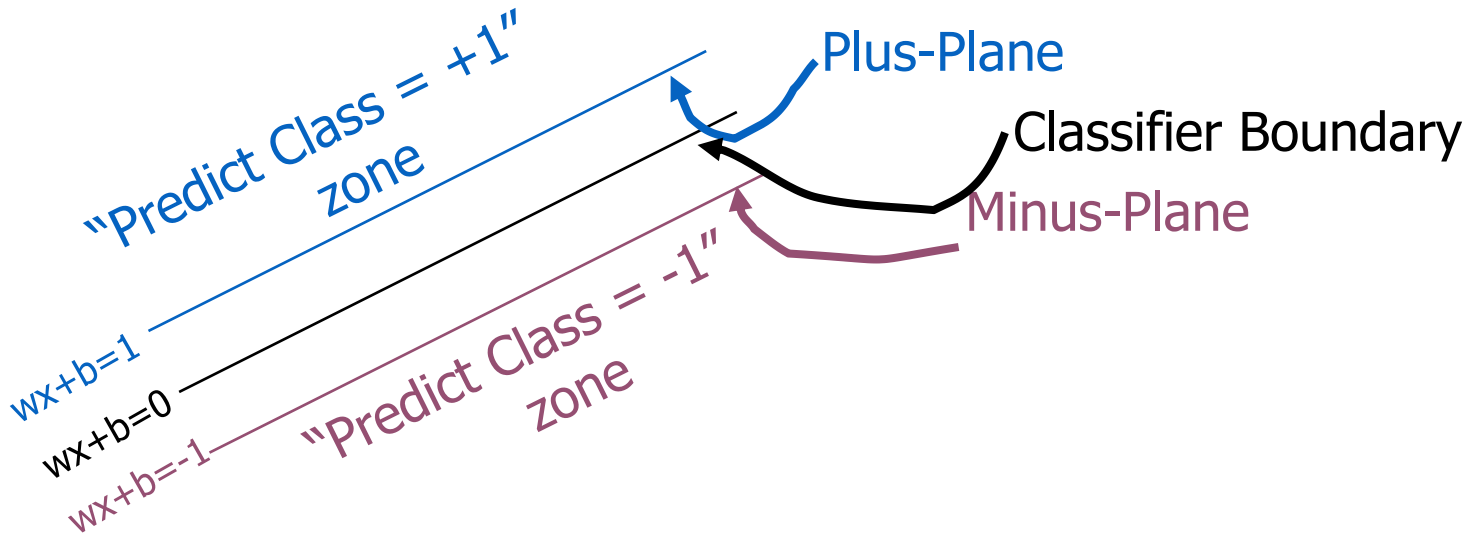
Specifying a line and margin



- How do we represent this mathematically?
- …in *m* input dimensions?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Specifying a line and margin

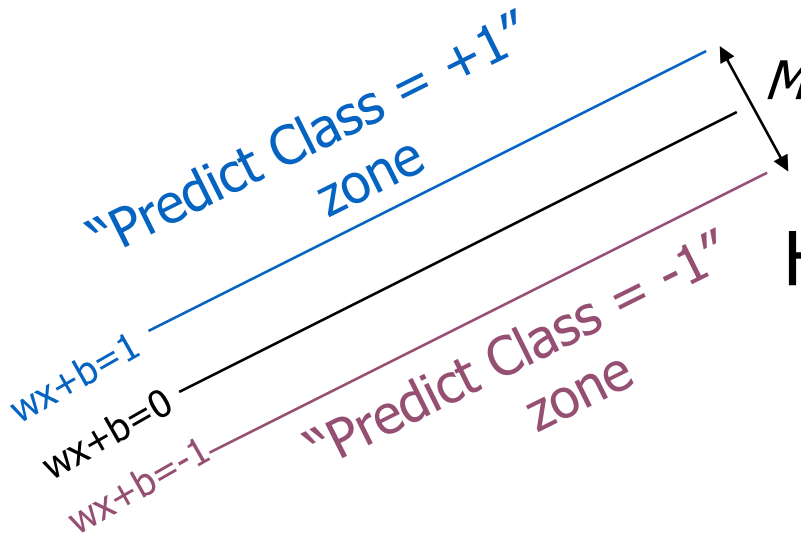


- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \}$

Classify as..	+1	if	$\boldsymbol{w} \cdot \boldsymbol{x} + b >= 1$
	-1	if	$\boldsymbol{w} \cdot \boldsymbol{x} + b <= -1$
	Universe explodes	if	$-1 < \boldsymbol{w} \cdot \boldsymbol{x} + b < 1$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



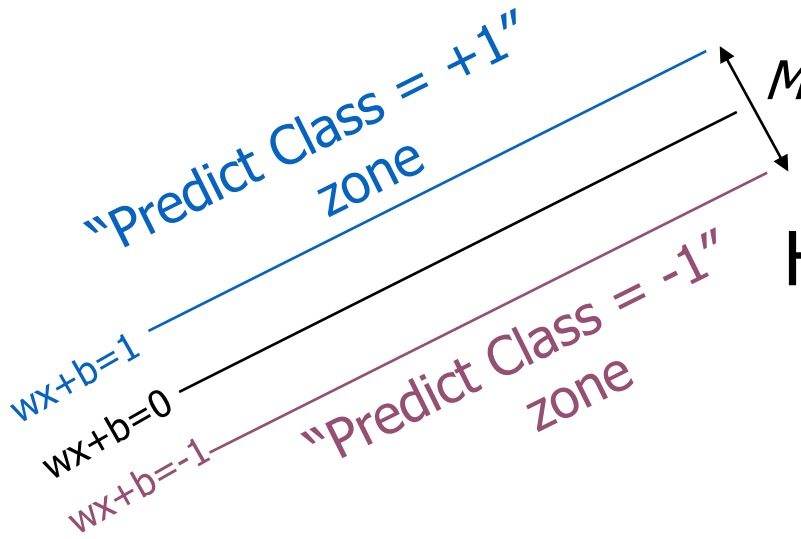
How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \}$

Claim: The vector **w** is perpendicular to the Plus Plane. Why?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \}$

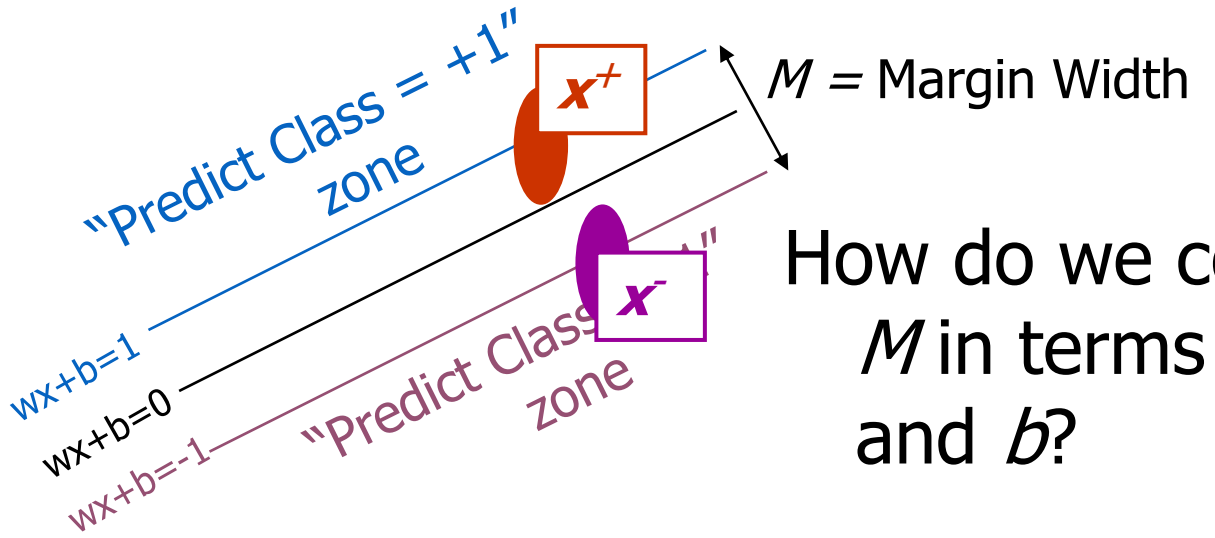
Claim: The vector **w** is perpendicular to the Plus Plane. Why?

Let **u** and **v** be two vectors on the Plus Plane. What is $\boldsymbol{w} \cdot (\boldsymbol{u} - \boldsymbol{v})$?

And so of course the vector **w** is also perpendicular to the Minus Plane

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



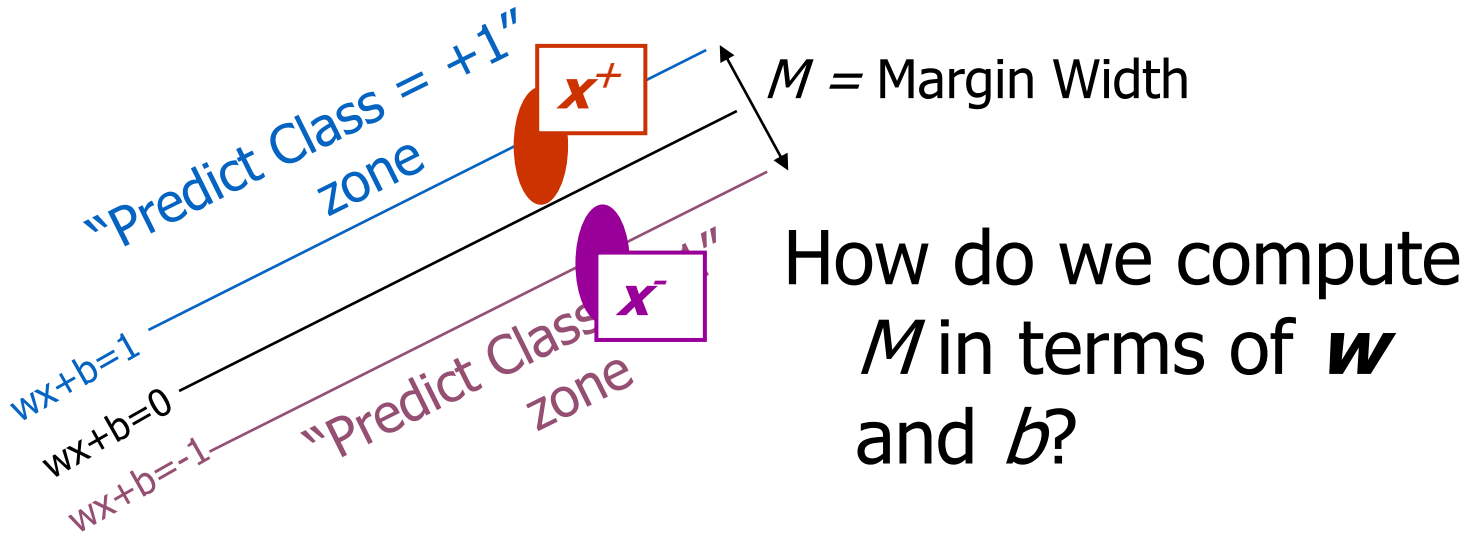
How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\boldsymbol{x}^-$ be any point on the minus plane
- Let $\boldsymbol{x}^+$ be the closest plus-plane-point to $\boldsymbol{x}^-$.

Any location in R^m: not necessarily a datapoint

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width

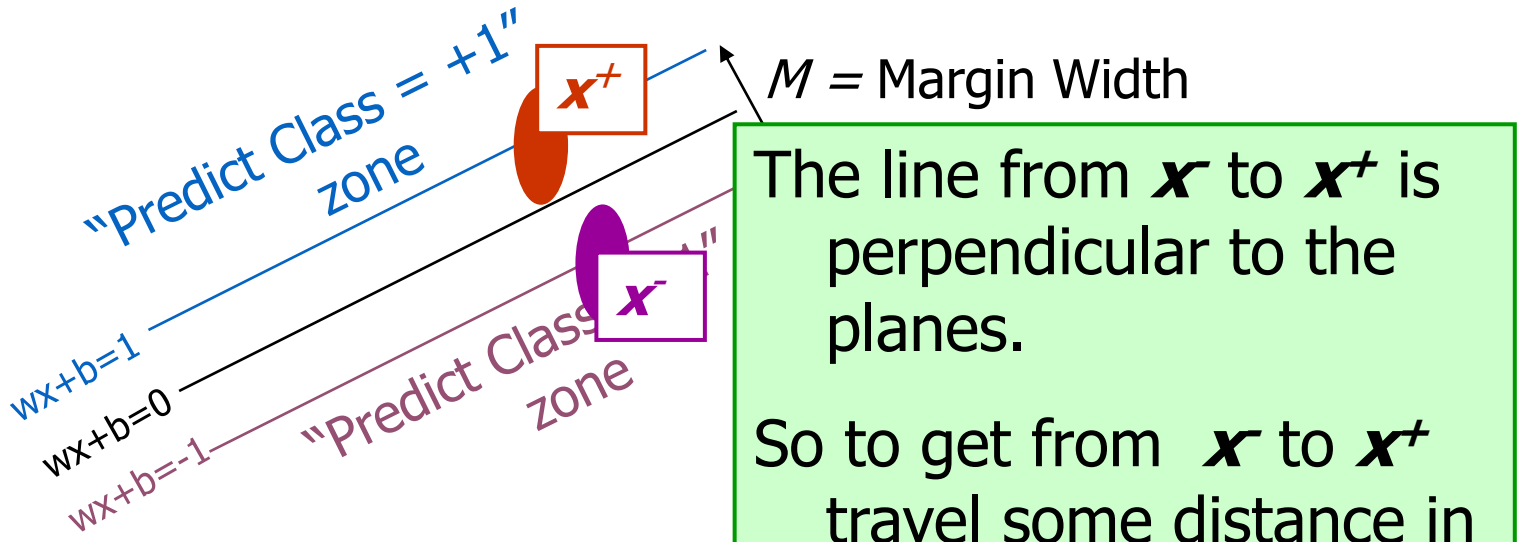


How do we compute *M* in terms of ***w*** and *b*?

- Plus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = +1 \}$
- Minus-plane = $\{ \boldsymbol{x} : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let $\boldsymbol{x}^-$ be any point on the minus plane
- Let $\boldsymbol{x}^+$ be the closest plus-plane-point to $\boldsymbol{x}^-$.
- Claim: $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$ for some value of λ. Why?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



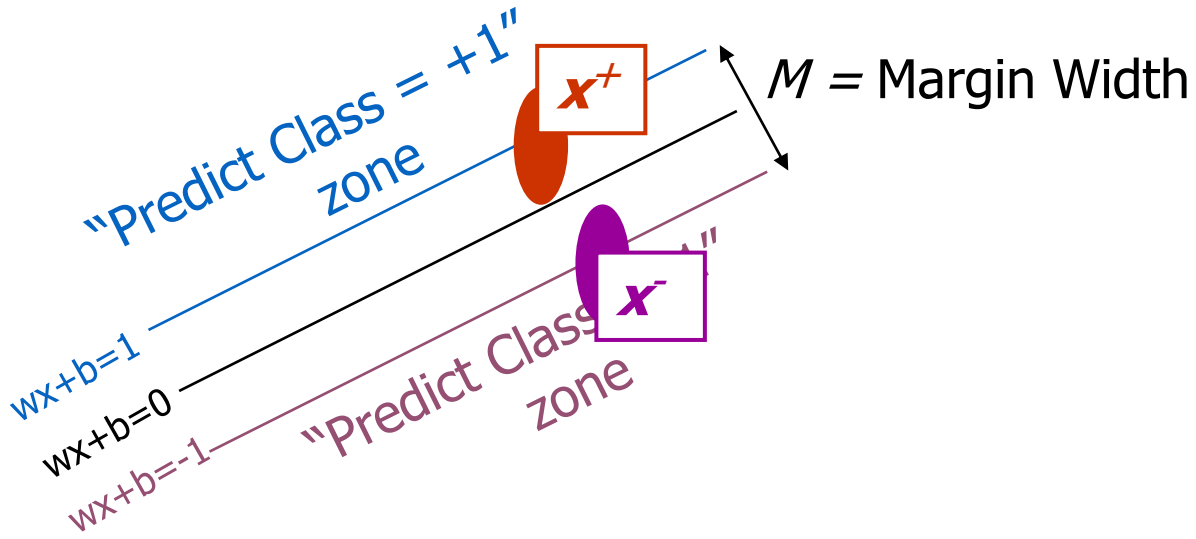
The line from x^- to x^+ is perpendicular to the planes.

So to get from x^- to x^+ travel some distance in direction w.

- Plus-plane = $\{ x : w \cdot x + b = +1 \}$
- Minus-plane = $\{ x : w \cdot x + b = -1 \}$
- The vector **w** is perpendicular to the Plus Plane
- Let x^- be any point on the minus plane
- Let x^+ be the closest plus-plane-point to x^-.
- Claim: $x^+ = x^- + \lambda w$ for some value of λ. Why?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



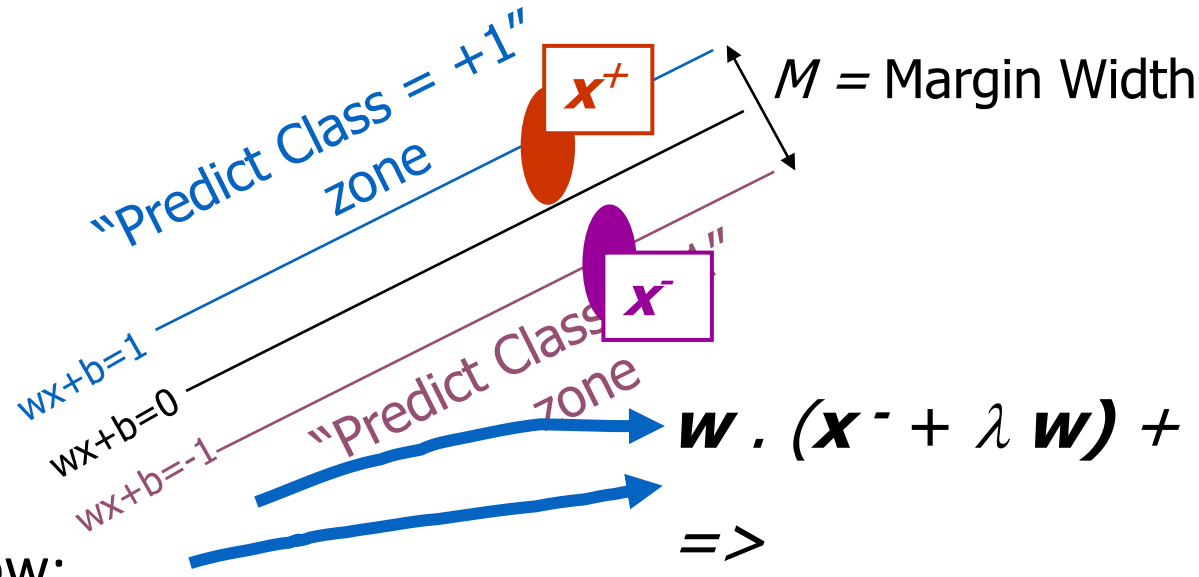
What we know:

- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = M$

It's now easy to get M in terms of $\boldsymbol{w}$ and b

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



What we know:

- $\boldsymbol{w} \cdot \boldsymbol{x}^+ + b = +1$
- $\boldsymbol{w} \cdot \boldsymbol{x}^- + b = -1$
- $\boldsymbol{x}^+ = \boldsymbol{x}^- + \lambda \boldsymbol{w}$
- $|\boldsymbol{x}^+ - \boldsymbol{x}^-| = M$

It's now easy to get M in terms of $\boldsymbol{w}$ and b

$\boldsymbol{w} \cdot (\boldsymbol{x}^- + \lambda \boldsymbol{w}) + b = 1$

$\Rightarrow$

$\boldsymbol{w} \cdot \boldsymbol{x}^- + b + \lambda \boldsymbol{w} \cdot \boldsymbol{w} = 1$

$\Rightarrow$

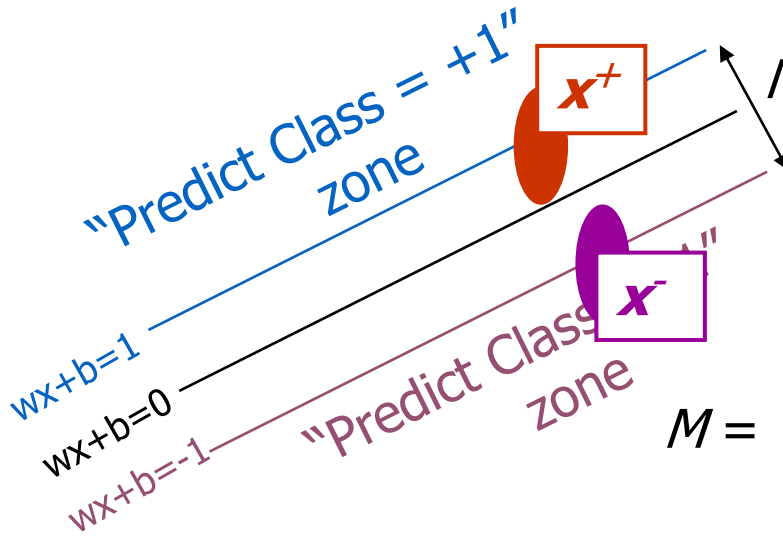
$-1 + \lambda \boldsymbol{w} \cdot \boldsymbol{w} = 1$

$\Rightarrow$

$$\lambda = \frac{2}{\mathbf{w} \cdot \mathbf{w}}$$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Computing the margin width



M = Margin Width = $\frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

What we know:

- $\mathbf{w} \cdot \mathbf{x}^+ + b = +1$
- $\mathbf{w} \cdot \mathbf{x}^- + b = -1$
- $\mathbf{x}^+ = \mathbf{x}^- + \lambda \mathbf{w}$
- $|\mathbf{x}^+ - \mathbf{x}^-| = M$
- $$\lambda = \frac{2}{\mathbf{w}.\mathbf{w}}$$

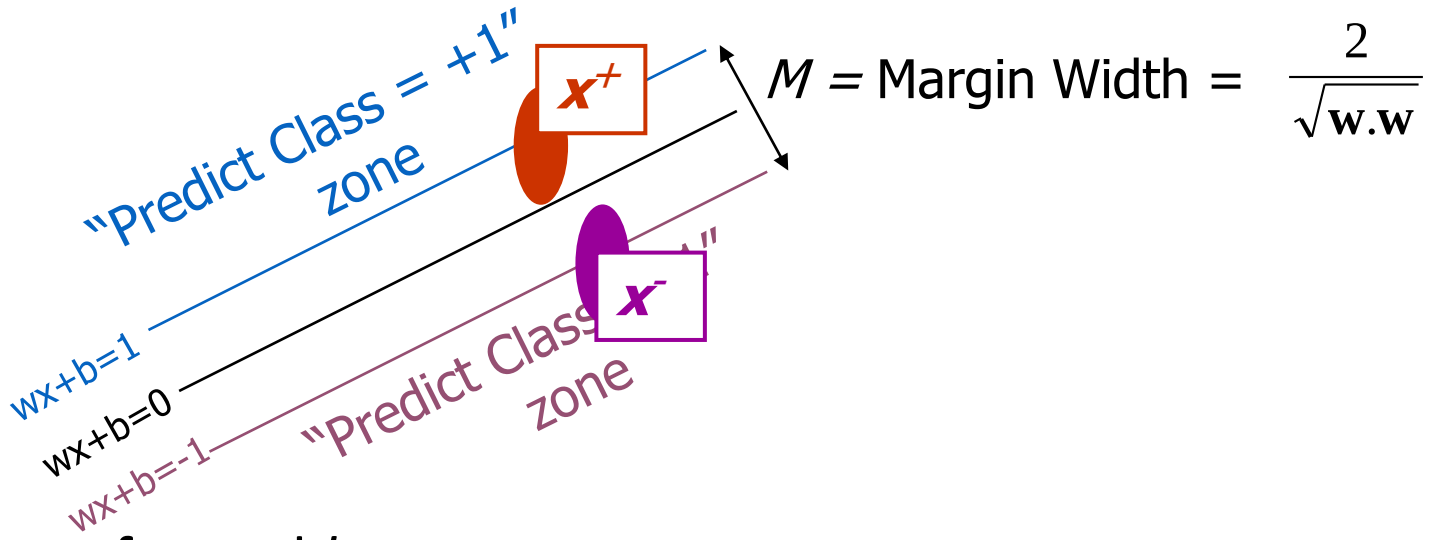
$$M = |\mathbf{x}^+ - \mathbf{x}^-| = |\lambda \mathbf{w}| =$$

$$= \lambda |\mathbf{w}| = \lambda \sqrt{\mathbf{w}.\mathbf{w}}$$

$$= \frac{2\sqrt{\mathbf{w}.\mathbf{w}}}{\mathbf{w}.\mathbf{w}} = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Learning the Maximum Margin Classifier



Given a guess of ***w*** and *b* we can

- Compute whether all data points in the correct half-planes
- Compute the width of the margin

So now we just need to write a program to search the space of **w**'s and *b*'s to find the widest margin that matches all the datapoints. *How?*

Gradient descent? Simulated Annealing? Matrix Inversion? EM? Newton's Method?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Learning via Quadratic Programming

- QP is a well-studied class of optimization algorithms to maximize a quadratic function of some real-valued variables subject to linear constraints.

Courtesy: (1) http://www.cs.carnegie-mu... Moore
(2) Copyright © 2001, 2003, Andrew W. Moore

Quadratic Programming

Find $\arg\max_{\mathbf{u}} \; c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$ ← Quadratic criterion

Subject to

$$a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m \le b_1$$
$$a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m \le b_2$$
$$\vdots$$
$$a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m \le b_n$$

} *n* additional linear inequality constraints

And subject to

$$a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m = b_{(n+1)}$$
$$a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m = b_{(n+2)}$$
$$\vdots$$
$$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m = b_{(n+e)}$$

} *e* additional linear equality constraints

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Quadratic Programming

Find $\arg\max_{\mathbf{u}} \quad c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$

Quadratic criterion

Subject to

...additional linear inequality constraints

And subj...

There exist algorithms for finding such constrained quadratic optima much more efficiently and reliably than gradient ascent.

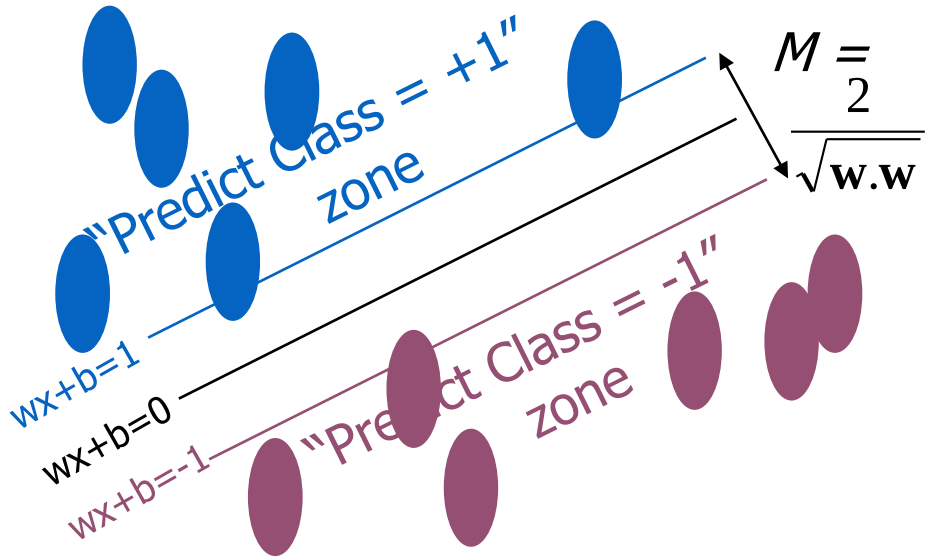
(But they are very fiddly…you probably don't want to write one yourself)

$a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + .. + a_{(n+e)m}u_m = b_{(n+e)}$

e additional linear equality constraints

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Learning the Maximum Margin Classifier



Given guess of $\boldsymbol{w}$, b we can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

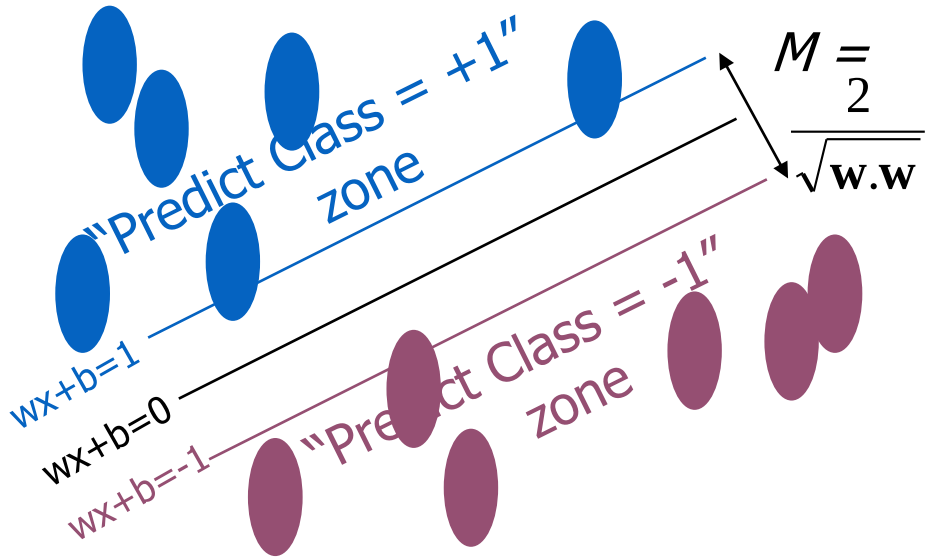
What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Learning the Maximum Margin Classifier



Given guess of $\boldsymbol{w}$, b we can

- Compute whether all data points are in the correct half-planes
- Compute the margin width

Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\boldsymbol{w.w}$

How many constraints will we have? R

What should they be?

$\boldsymbol{w} \cdot \boldsymbol{x}_k + b >= 1$ if $y_k = 1$

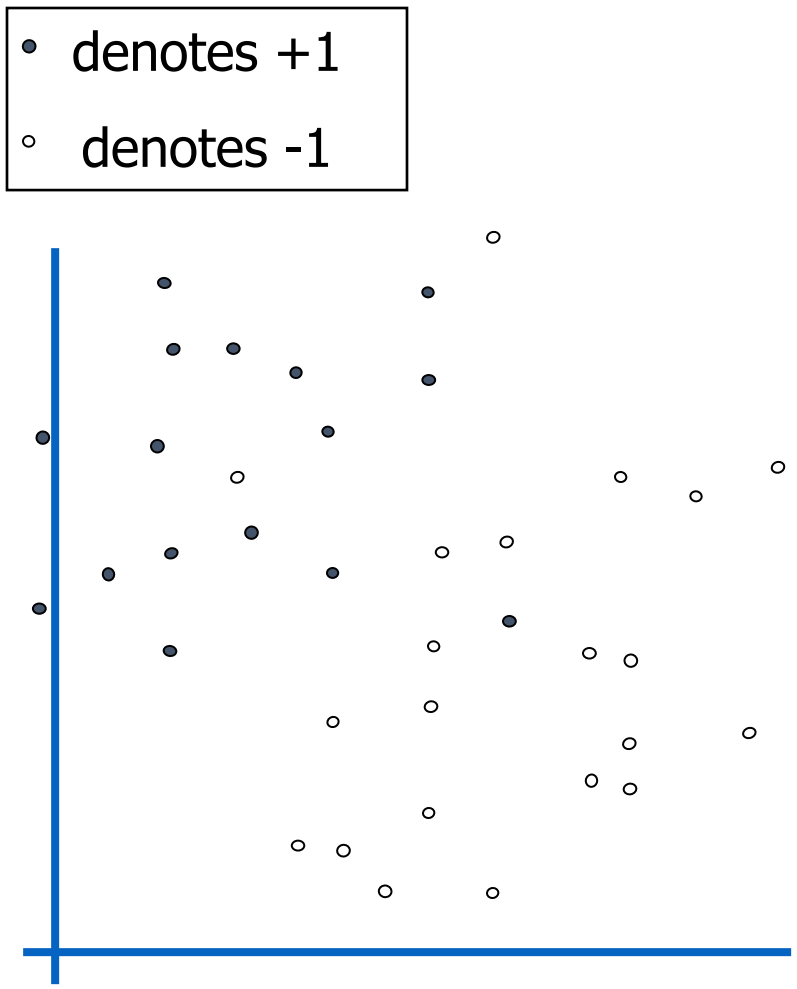
$\boldsymbol{w} \cdot \boldsymbol{x}_k + b <= -1$ if $y_k = -1$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Uh-oh!

This is going to be a problem!

What should we do?



Uh-oh!

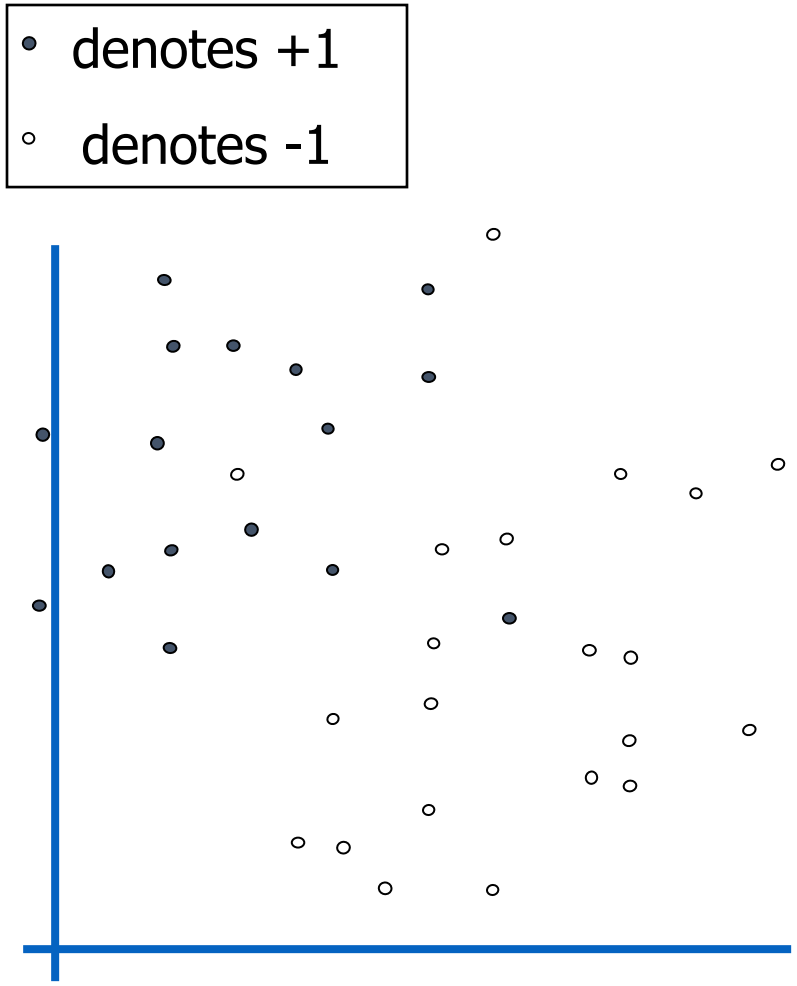
This is going to be a problem!

What should we do?

Idea 1:

Find minimum ***w.w***, while minimizing number of training set errors.

Problemette: Two things to minimize makes for an ill-defined optimization



Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Uh-oh!

This is going to be a problem!

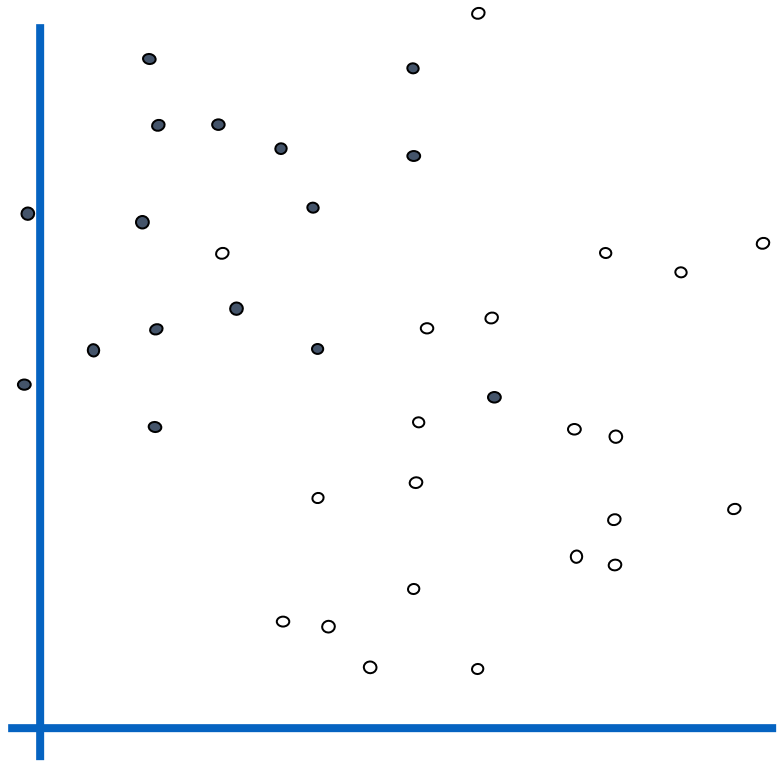
What should we do?

Idea 1.1:

Minimize

w.w *+ C (#train errors)*

Tradeoff parameter



There's a serious practical problem that's about to make us reject this approach. Can you guess what it is?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Uh-oh!

This is going to be a problem!

What should we do?

Idea 1.1:

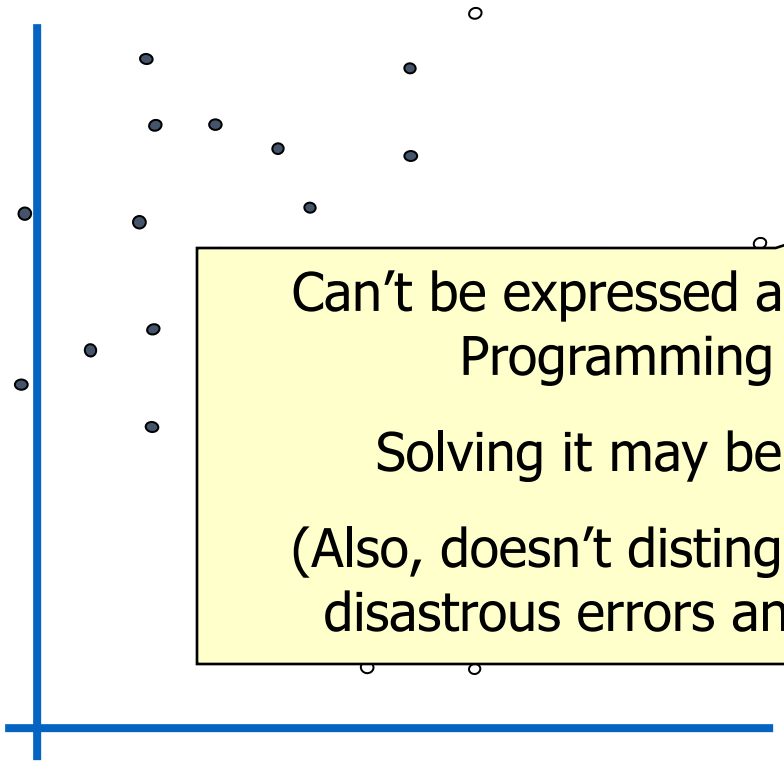
Minimize

w.w *+ C (#train errors)*

Tradeoff parameter

• denotes +1

◦ denotes -1



Can't be expressed as a Quadratic Programming problem.

Solving it may be too slow.

(Also, doesn't distinguish between disastrous errors and near misses)

So… any other ideas?

you guess wh

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Uh-oh!

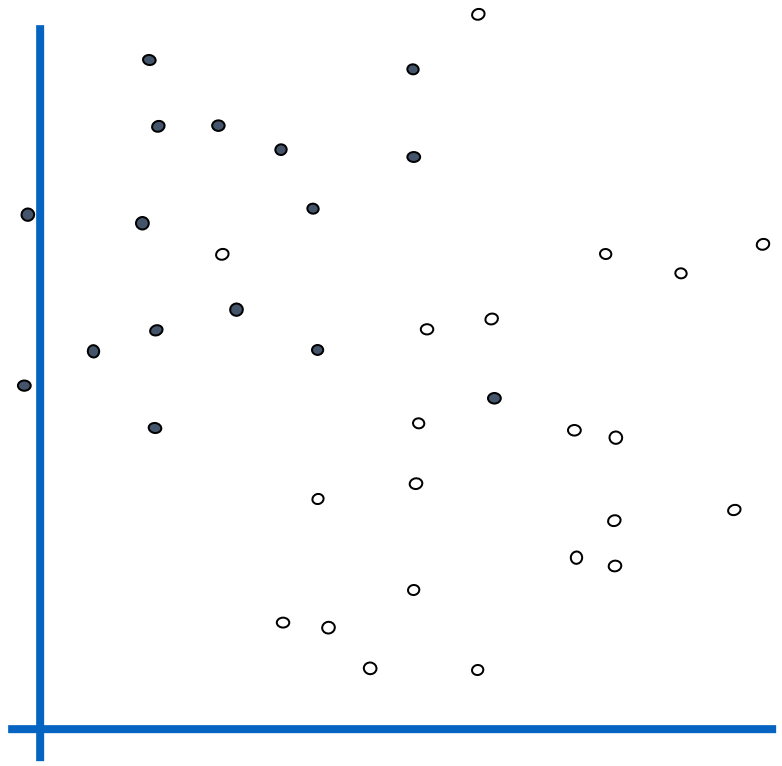
This is going to be a problem!

What should we do?

Idea 2.0:

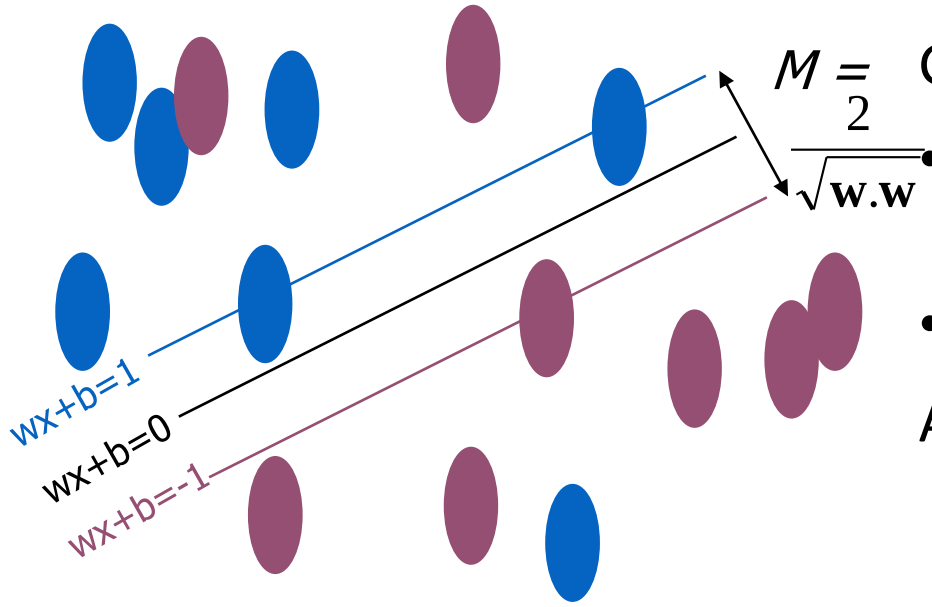
Minimize

w.w *+ C (distance of error points to their correct place)*



Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Learning Maximum Margin with Noise



Given guess of $\boldsymbol{w}$, b we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume R datapoints, each $(\boldsymbol{x}_k, y_k)$ where $y_k = +/- 1$

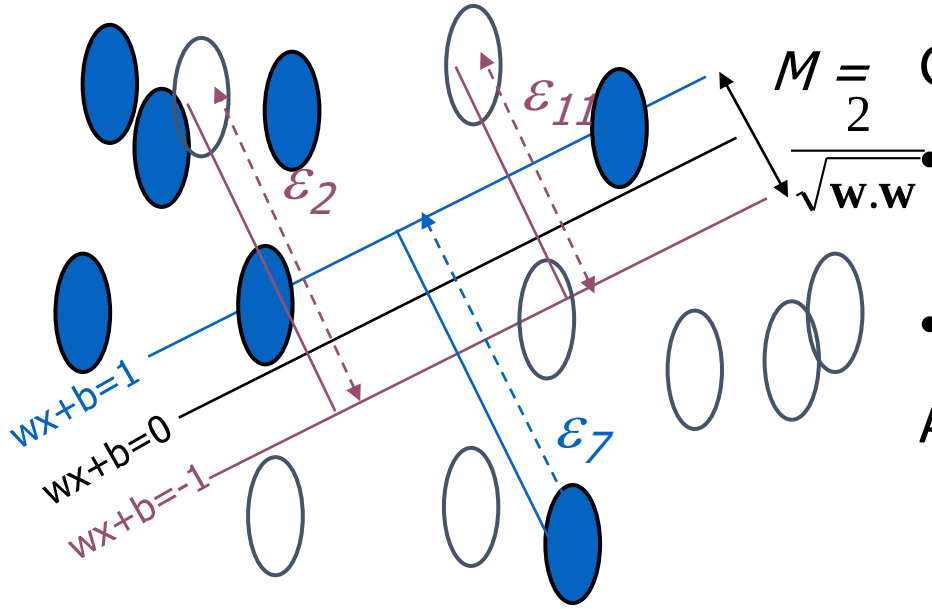
What should our quadratic optimization criterion be?

How many constraints will we have?

What should they be?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Learning Maximum Margin with Noise



Given guess of ***w*** , *b* we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

How many constraints will we have? *R*

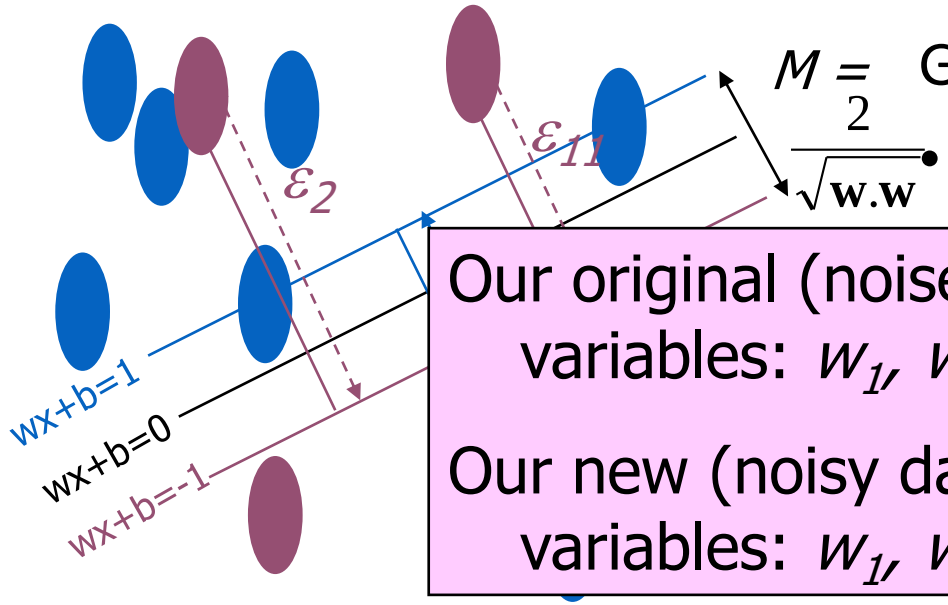
What should they be?

$\mathbf{w} . \mathbf{x}_k + b >= 1-\varepsilon_k$ *if* $y_k = 1$

$\mathbf{w} . \mathbf{x}_k + b <= -1+\varepsilon_k$ *if* $y_k = -1$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Learning Maximum Margin with Noise



$M = \frac{2}{\sqrt{\mathbf{w}.\mathbf{w}}}$

Given gu… can

Compute sum of d…tances of points to their correct zones

m = # input dimensions

Our original (noiseless data) QP had *m+1* variables: $w_1, w_2, \ldots w_m$, and *b*.

Our new (noisy data) QP has *m+1+R* variables: $w_1, w_2, \ldots w_m, b, \varepsilon_k, \varepsilon_1, \ldots \varepsilon_R$

R= # records

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

How many constrain… have? *R*

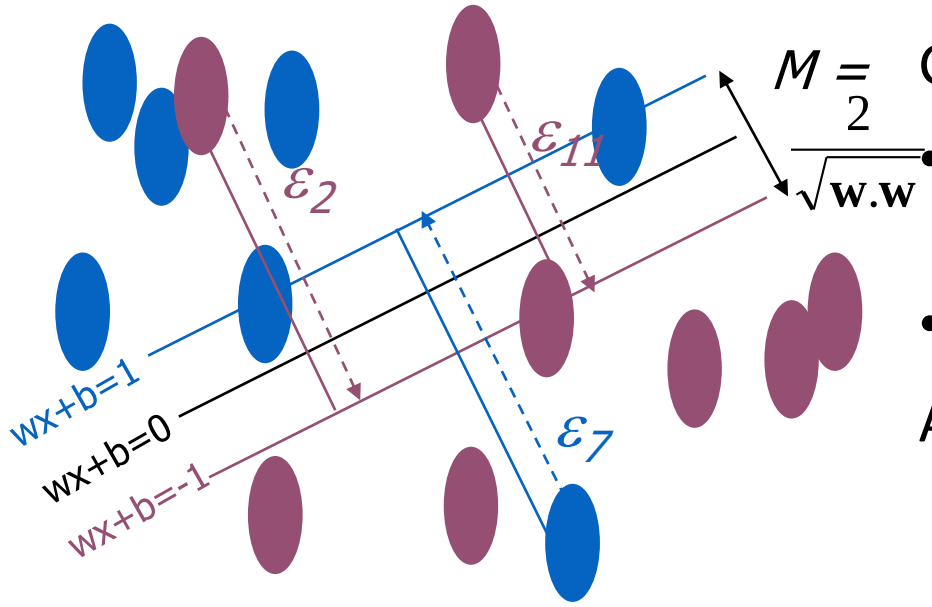
What should they be?

$\mathbf{w} . \mathbf{x}_k + b >= 1-\varepsilon_k$ *if* $y_k = 1$

$\mathbf{w} . \mathbf{x}_k + b <= -1+\varepsilon_k$ *if* $y_k = -1$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Learning Maximum Margin with Noise



Given guess of ***w*** , *b* we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where y_k = +/- 1

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

How many constraints will we have? *R*

What should they be?

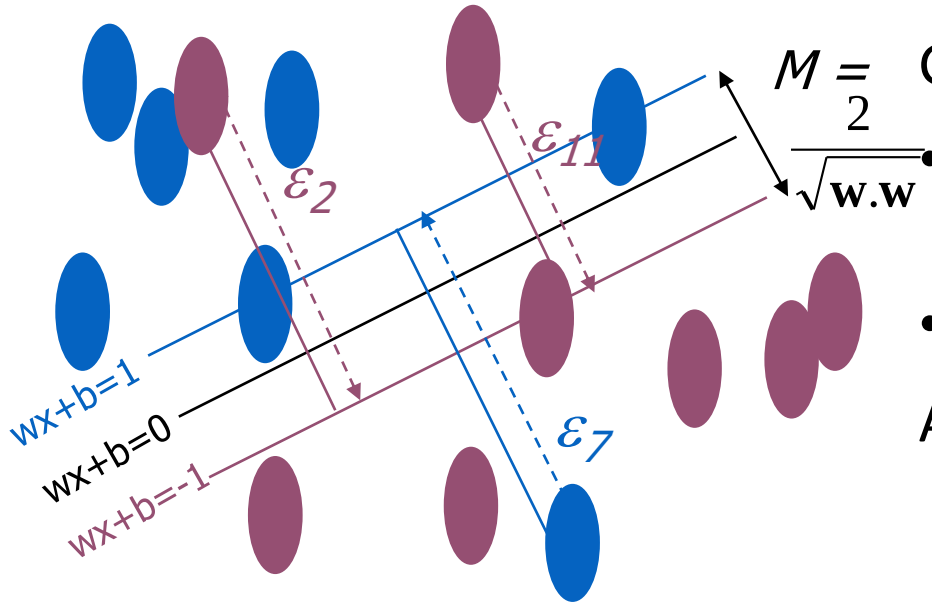
$\mathbf{w} . \mathbf{x}_k + b >= 1-\varepsilon_k$ if $y_k = 1$

$\mathbf{w} . \mathbf{x}_k + b <= -1+\varepsilon_k$ if $y_k = -1$

There's a bug in this QP. Can you spot it?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Learning Maximum Margin with Noise



Given guess of ***w*** , *b* we can

- Compute sum of distances of points to their correct zones
- Compute the margin width

Assume *R* datapoints, each $(\mathbf{x}_k, y_k)$ where $y_k = +/- 1$

What should our quadratic optimization criterion be?

Minimize $\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$

How many constraints will we have? *2R*

What should they be?

$\mathbf{w} . \mathbf{x}_k + b >= 1-\varepsilon_k$ *if* $y_k = 1$

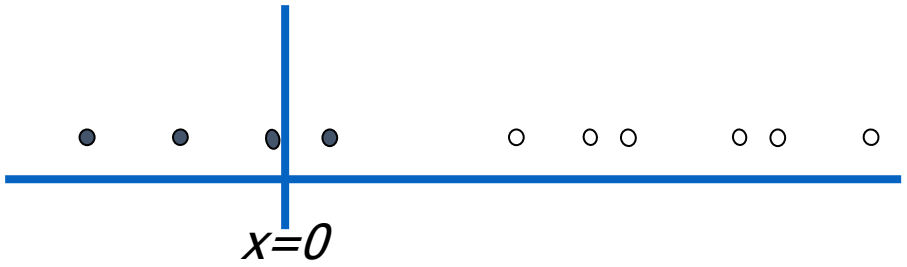
$\mathbf{w} . \mathbf{x}_k + b <= -1+\varepsilon_k$ *if* $y_k = -1$

$\varepsilon_k >= 0$ for all k

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

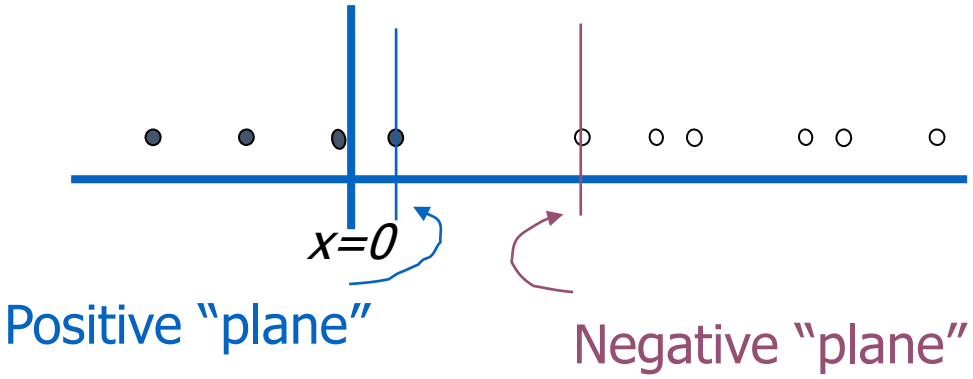
Suppose we're in 1-dimension

What would SVMs do with this data?



Suppose we're in 1-dimension

Not a big surprise



Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

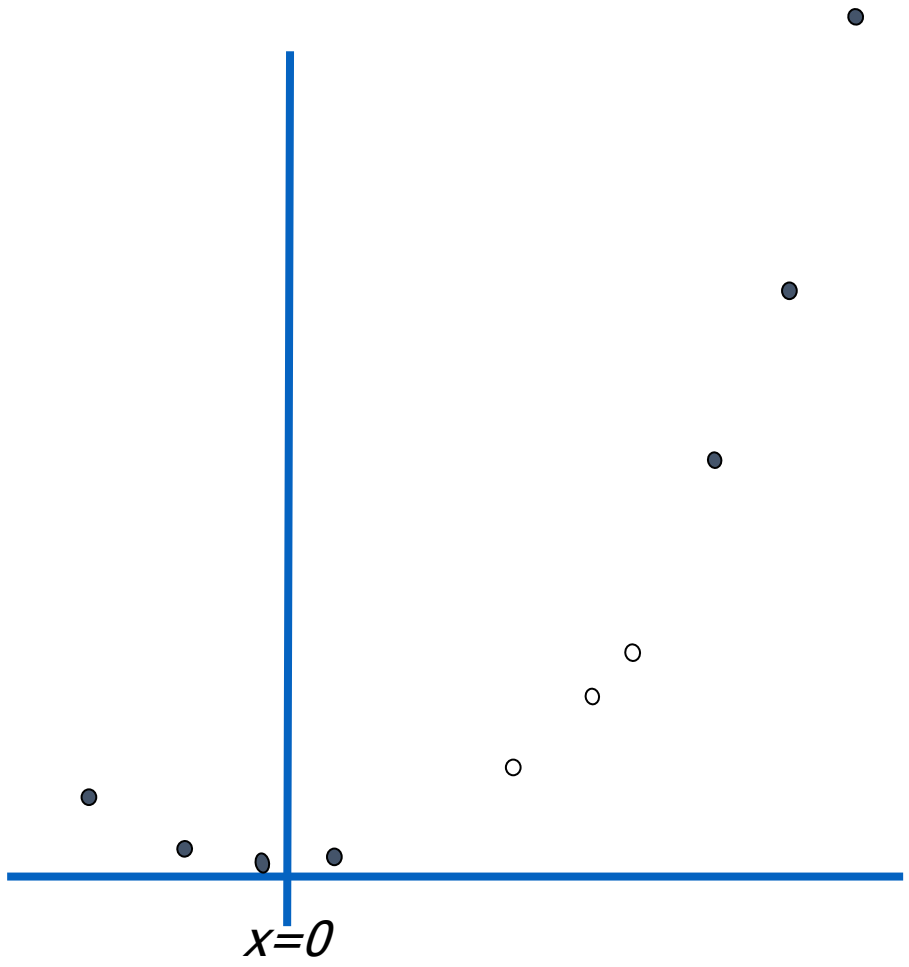
Harder 1-dimensional dataset

That's wiped the smirk off SVM's face.

What can be done about this?

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials
(2) Copyright © 2001, 2003, Andrew W. Moore

Harder 1-dimensional dataset



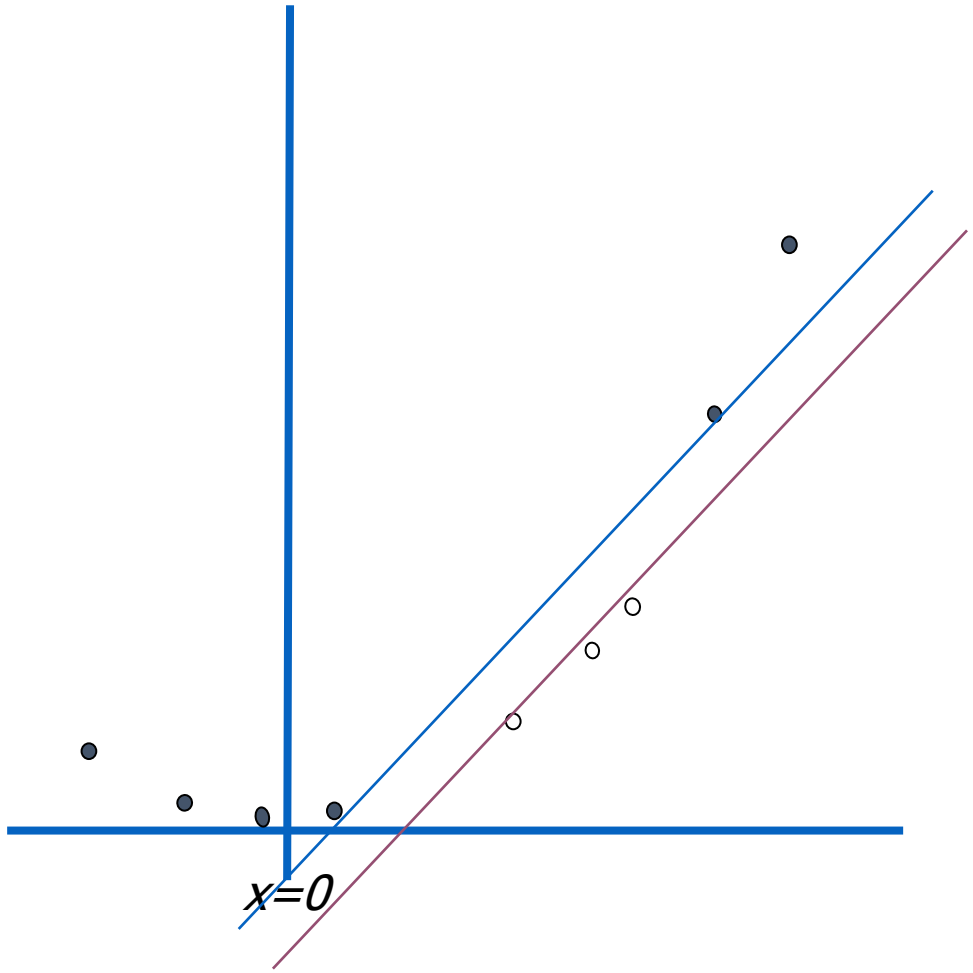
Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Harder 1-dimensional dataset



Remember how permitting non-linear basis functions made linear regression so much nicer?

Let's permit them here too

$$\mathbf{z}_k = (x_k, x_k^2)$$

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Common SVM basis functions

$\boldsymbol{z}_k$ = (polynomial terms of $\boldsymbol{x}_k$ of degree 1 to q)

$\boldsymbol{z}_k$ = (radial basis functions of $\boldsymbol{x}_k$)

$$\mathbf{z}_k[j] = \varphi_j(\mathbf{x}_k) = \text{KernelFn}\left(\frac{|\mathbf{x}_k - \mathbf{c}_j|}{\text{KW}}\right)$$

$\boldsymbol{z}_k$ = (sigmoid functions of $\boldsymbol{x}_k$)

This is sensible.

Is that the end of the story?

No…there's one more trick!

Courtesy: (1) http://www.cs.cmu.edu/~awm/tutorials

Support Vector Machines (SVM)

Three main ideas:

1. Define what an optimal hyperplane is (taking into account that it needs to be computed efficiently): maximize margin
2. Generalize to non-linearly separable problems: have a penalty term for misclassifications
3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data are mapped implicitly to this space

Courtesy: http://www.cs.tau.ac.il/~rshamir/abdbm/pres/17/SVM.pdf

Logistic Regressions

What is logistic regression?

- Logit -, a specialized form of regression
- used when the dependent variable is BInary and categorical while the independent variable(s) could be any type.
- Dependent variable is binary (yes/no, Male/Female) outcome.
- Independent variables are continuous interval
- Examples:
  - Relation of weight and BP to 10 year risk of death

Logistic Regressions

What is logistic regression? - Continued

- Form of regression that allows the prediction of discrete variables by a mix of continuous and discrete/nominal predictors.
- Addresses the same questions that discriminant function analysis and multiple regression do but with no distributional assumptions on the predictors (the predictors do not have to be normally distributed, linearly related or have equal variance in each group)
- Does not assume a linear relationship between DV and IV

Logistic Regressions

Why we use ?

- No assumptions about the distributions of the predictor variables.
- Predictors do not have to be normally distributed
- Does not have to be linearly related.
- When equal variances , covariance doesn't exist across the groups.

Logistic Regressions

Types of Logistic Regression?

- ***BINARY LOGISTIC REGRESSION***

  It is used when the dependent variable is binary in nature.

MULTINOMIAL LOGISTIC REGRESSION

It is used when the dependent or outcomes variable has more than two categories.

Logistic Regressions

Binary logistic regression expression

$$Y = b_o + b_1X_1 + b_2X_2 + \ldots\ldots\ldots\ldots b_kX_k + E$$

BINARY Y

Y = Dependent Variables

β_0 = Constant

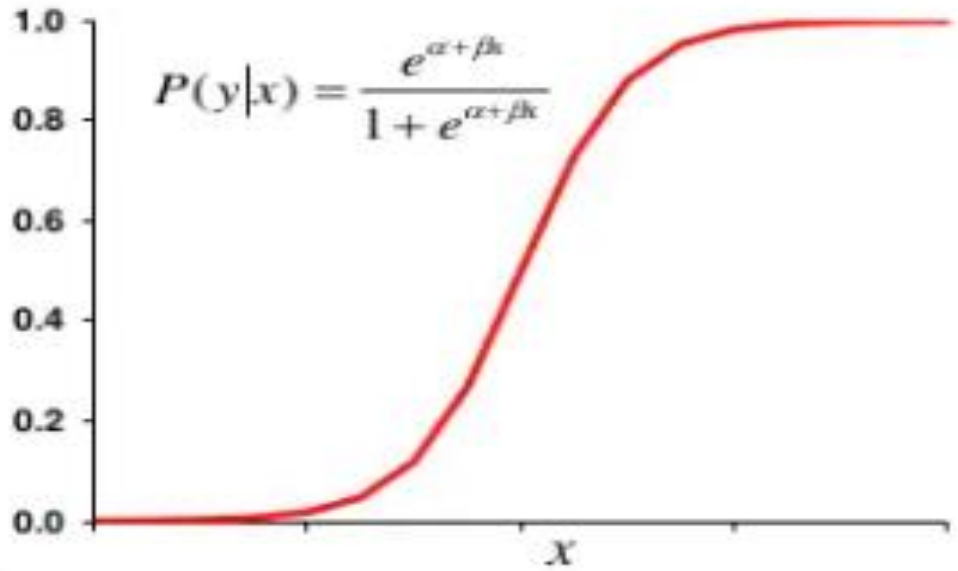
β_1 = Coefficient of variable X_1

X_1 = Independent Variables

E = Error Term

Logistic Regressions

- We want a model that predicts probabilities between 0 and 1, that is, S-shaped.
- There are lots of S-shaped curves. We use the logistic model:
- Probability = $1/[1 + \exp(\beta_0 + \beta_1 X)]$ or $\log_e[P/(1-P)] = \beta_0 + \beta_1 X$
- The function on left, $\log_e[P/(1-P)]$, is called the logistic function.



Logistic Regressions

Logistic regression models the logit of the outcome

=Natural logarithm of the odds of the outcome

=ln(Probability of the outcome (p)/Probability of not having the outcome (1-p))

$$\ln\left(\frac{P}{1-P}\right) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_i x_i$$

β = log odds ratio associated with predictors

e^{β} = odds ratio

Logistic Regressions

Summary:

- Logistic regression is a probabilistic model that classifies the instances in terms of probabilities.
- Since the classification is probabilistic, a natural approach for optimizing the parameters is to ensure that the predicted probability of the observed class for each training instance is as large as possible.
- This is achieved by using the notion of maximum-likelihood estimation in order to learn the parameters of the model.
- The likelihood of the training data is defined as the product of the probabilities of the observed labels of each training instance.
- Larger values of this objective function are better.
- By using the negative logarithm of this value, one obtains an a loss function in minimization form.
- Therefore, the output node uses the negative log-likelihood as a loss function.
- This loss function replaces the squared error used in the Widrow-Hoff method.
- The output layer can be formulated with the sigmoid activation function, which is very common in neural network design.

Loss Functions

- Loss functions measure how far an estimated value is from its true value.
- A loss function maps decisions to their associated costs.
- Loss functions are not fixed, they change depending on the task in hand and the goal to be met.

A way to measure how far a particular iteration of the model is from the actual values.

Loss Function for Regressions:

(1) Mean Absolute Error (MAE)

Mean Absolute Error (also called L1 loss) is one of the most simple yet robust loss functions used for regression models.

Defined as the average sum of the absolute differences between the actual and the predicted values.

For a data point x_i and its predicted value y_i, n being the total number of data points in the dataset, the mean absolute error is defined as:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

Loss Functions - Regressions

(2) Mean Squared Error (MSE):

Mean Squared Error (also called L2 loss)

Defined as the average of the squared differences between the actual and the predicted values. For a data point Y_i and its predicted value $\hat{Y}_i$, where n is the total number of data points in the dataset, the mean squared error is given as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

(3) Mean Bias Error (MBE):

Mean Bias Error is used to calculate the average bias in the model. Bias, in short, is overestimating or underestimating a parameter.

Mean Bias Error takes the actual difference between the target and the predicted value, and not the absolute difference

Loss Functions - Regressions

(3) Mean Bias Error (MBE):

One has to be cautious as the positive and the negative errors could cancel each other out, which is why it is one of the lesser-used loss functions

$$MBE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}$$

(4) Mean Squared Logarithmic Error (MSLE):

We may not want to penalize the model too much for predicting unscaled quantities directly. Relaxing the penalty on huge differences can be done with the help of Mean Squared Logarithmic Error.

Mean Squared Logarithmic Error is the same as Mean Squared Error, except the natural logarithm of the predicted values is used rather than the actual values

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(\log(Y_i) - \log(\hat{Y}_i))^2$$

Loss Functions - Classifications

(1) Binary Cross Entropy Loss:

This is the most common loss function used for classification problems that have two classes

Entropy is the measure of randomness in the information being processed, and cross entropy is a measure of the difference of the randomness between two random variables.

$$J = -\sum_{i=1}^{N} y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

- Where y_i is the true label and $h_\theta(x_i)$ is the predicted value post hypothesis.
- Since binary classification means the classes take either 0 or 1, if $y_i = 0$, that term ceases to exist and if $y_i = 1$, the $(1-y_i)$ term becomes 0.

Loss Functions - Classifications

(2) Categorical Cross Entropy Loss:

Categorical Cross Entropy loss is essentially Binary Cross Entropy Loss expanded to multiple classes. One requirement when categorical cross entropy loss function is used is that the labels should be one-hot encoded.

(3) Hinge Loss:

Commonly used loss function for classification is the hinge loss. Hinge loss is primarily developed for support vector machines for calculating the maximum margin from the hyperplane to the classes.

Loss functions penalize wrong predictions and does not do so for the right predictions. So, the score of the target label should be greater than the sum of all the incorrect labels by a margin of (at the least) one.

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

Loss Functions - Classifications

(4) Kullback Leibler Divergence Loss (KL Loss):

Kullback Leibler Divergence Loss is a measure of how a distribution varies from a reference distribution (or a baseline distribution).

A Kullback Leibler Divergence Loss of zero means that both the probability distributions are identical.

The number of information lost in the predicted distribution is used as a measure.

$$D_{KL}(P||Q) = \begin{cases} -\sum_x P(x).\log\frac{Q(x)}{P(x)} = \sum_x P(x).\log\frac{P(x)}{Q(x)}, & \text{for discrete distributions} \\ -\int P(x).\log\frac{Q(x)}{P(x)}.dx = \int P(x).\log\frac{P(x)}{Q(x)}.dx, & \text{for continuous distributions} \end{cases}$$

Back Propagation- Algorithm

Algorithm 1 Backpropagation

procedure BACKPROPAGATION
  for each observation pair $(\mathbf{x}, \mathbf{t})$, **do**
    Feed-forward: Feed the input $\mathbf{x}$ into the network, store all outputs o_i.
    Error at the output layer: Calculate the error δ_j at the output layer $\delta_j o_i = \frac{\partial Q}{\partial w_{ij}}$.
    Backpropagate to hidden layers: Calculate the hidden layer's errors as $\delta_h = \sum_q w_{hq}\delta_q$.
    Weight updates: Update the weights of all connections as $w_{ij} := w_{ij} - \eta o_i \delta_j$.

1. *Feed-forward:* The input $\mathbf{x}$ is fed into the input layer, and forward propagate through the network. Store the output of each neuron i as o_i.
2. *Error at the output layer:* The error δ_j of the j-th neuron in the output layer is computed as

$$o_i \delta_j = \frac{\partial Q}{\partial w_{ij}}, \tag{14}$$

   where w_{ij} is the weight from the i-th neuron to the j-th neuron, and o_i is the output of the i-th neuron, or conversely, the input into the j-th.
3. *Backpropagate to the hidden layers:* Since the expected output of neurons in the hidden layers is not known, it is instead estimated by errors in the layer closer to the output (in the case of the final hidden layer, the output layer). If w_{hq} is the connection from the h-th to the q-th neuron in the hidden layer, then the error is estimated as

$$\delta_h = \sum_q w_{hq}\delta_q \tag{15}$$

4. *Weight updates:* Update the weights of each connection as

$$w_{ij} := w_{ij} - \eta o_i \delta_j \tag{16}$$

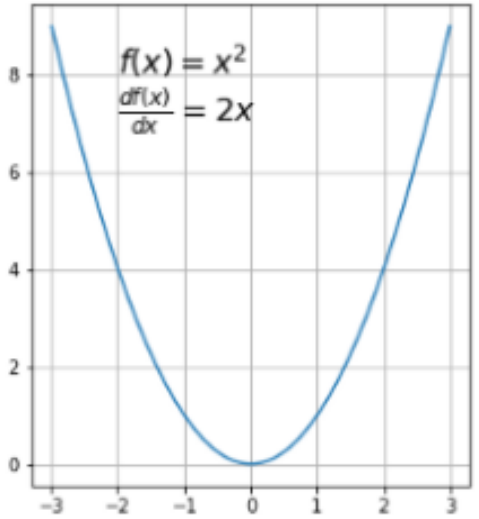
Gradient Descent- Algorithm

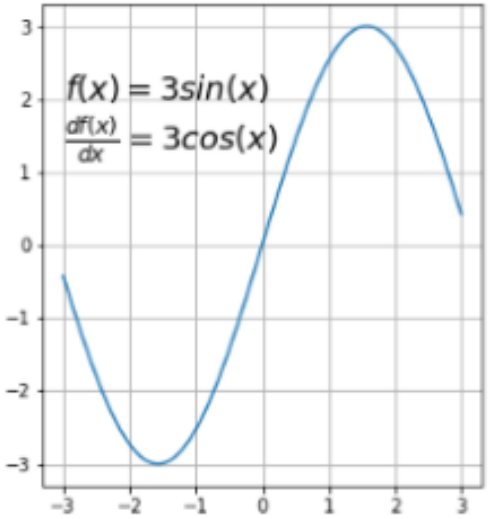
Gradient descent (GD) : is an iterative first-order optimization algorithm used to find a local minimum/maximum of a given function. This method is commonly used in *machine learning* (ML) and *deep learning*(DL) to minimize a cost/loss function .

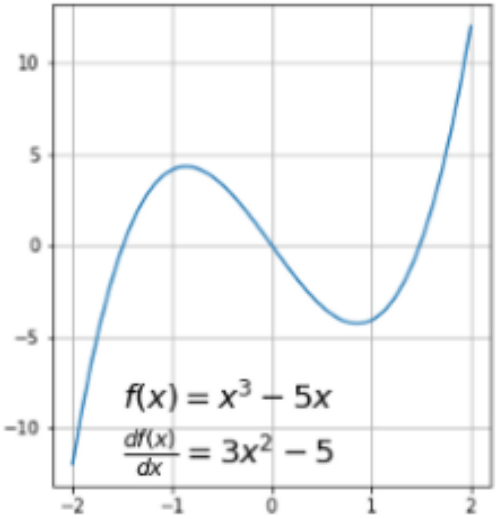
Function requirements of GD:

Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:

(1) Differentiable -If a function is differentiable it has a derivative for each point in its domain

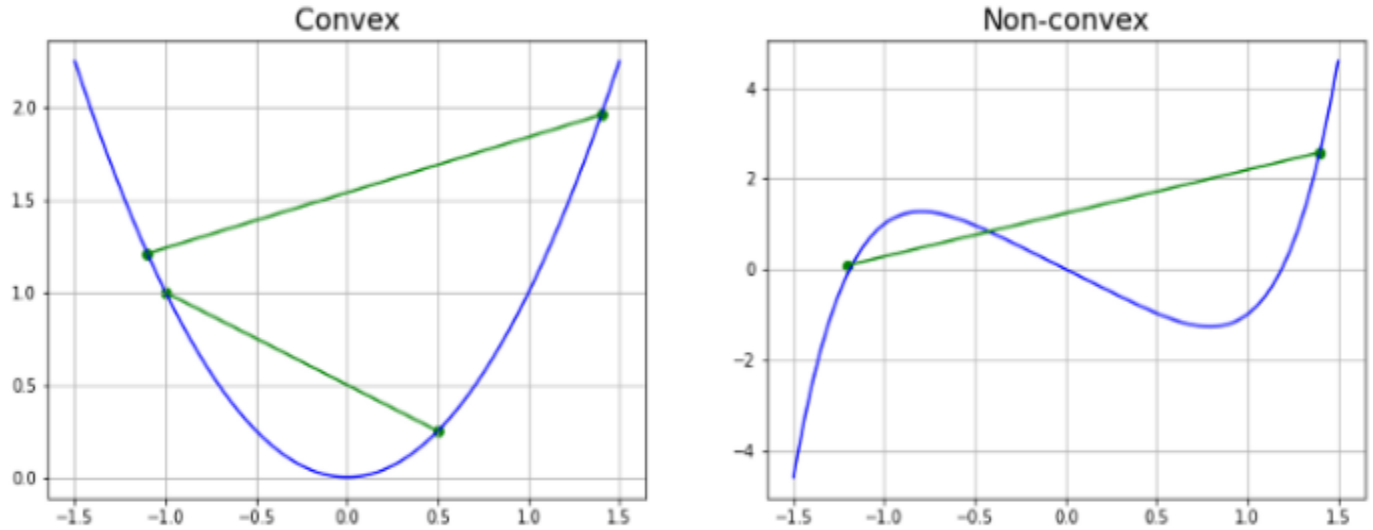






Gradient Descent- Algorithm

- (2) Convex -For a univariate function, this means that the line segment connecting two function's points lays on or above its curve (it does not cross it)



- Another way to check mathematically if a univariate function is convex is to calculate the second derivative and check if its value is always bigger than 0.

$$\frac{d^2 f(x)}{dx^2} > 0$$

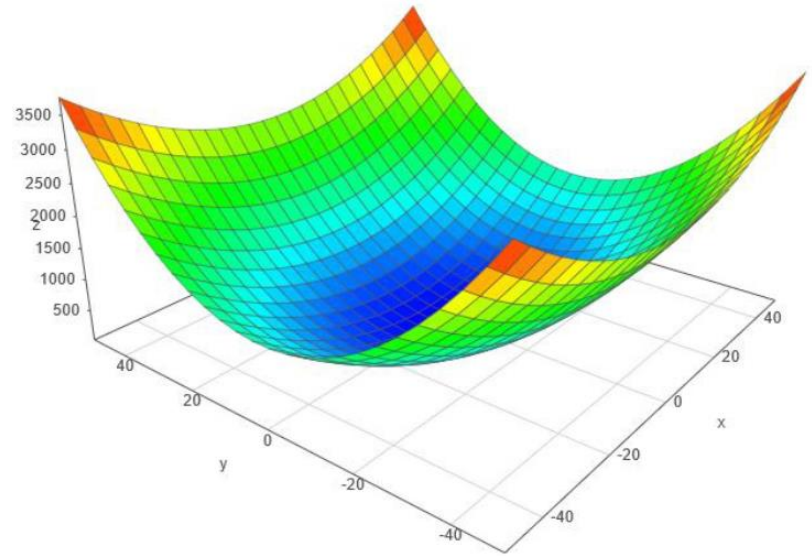
Gradient Descent- Algorithm

Gradient: **I**t is a slope of a curve at a given point in a specified direction.

- In the case of **a univariate function**, it is simply the **first derivative at a selected point**.
- In the case of **a multivariate function**, it is a **vector of derivatives** in each main direction (along variable axes).
- we are interested only in a slope along one axis and we don't care about others these derivatives are called **partial derivatives**.

A gradient for an n-dimensional function f(x) at a given point p is defined as follows:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix}$$



Gradient Descent- Algorithm

Gradient Descent Algorithm iteratively calculates the next point using gradient at the current position, then scales it (by a learning rate) and subtracts obtained value from the current position (makes a step). It subtracts the value because we want to minimize the function (to maximize it would be adding). This process can be written as:

$$p_{n+1} = p_n - \eta \nabla f(p_n)$$

Gradient Descent method's steps are:

1. Choose a starting point (initialization)
2. Calculate gradient at this point
3. Make a scaled step in the opposite direction to the gradient
   (objective: minimize)
4. Repeat points 2 and 3 until one of the criteria is met:
   - maximum number of iterations reached
   - step size is smaller than the tolerance.

Gradient Descent Algorithm - Issues

Challenges in Gradient Descent:

- For a good generalization we should have a large training set, which comes with a **huge computational cost**.
- i.e., as the training set grows to billions of examples, the time taken to take a **single gradient step** becomes long.

Note:

•The smaller learning rate the longer GD converges, or may reach maximum iteration before reaching the optimum point

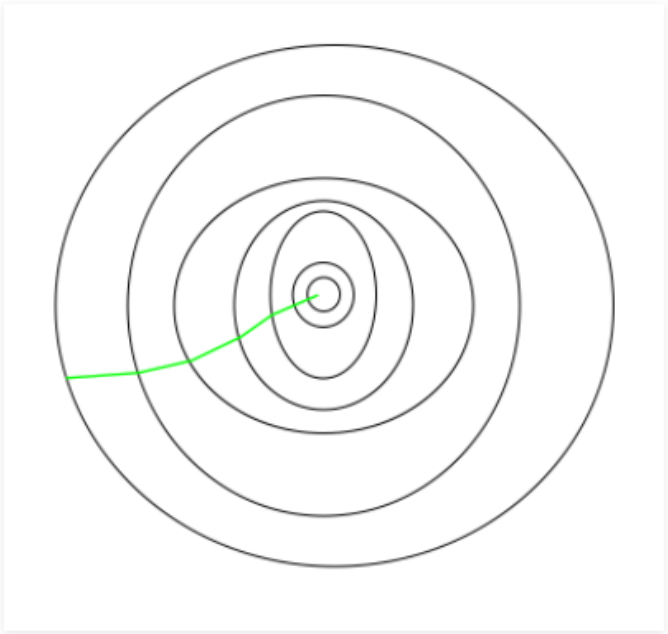
•If learning rate is too big the algorithm may not converge to the optimal point (jump around) or even to diverge completely.

Stochastic Gradient Descent- Algorithm

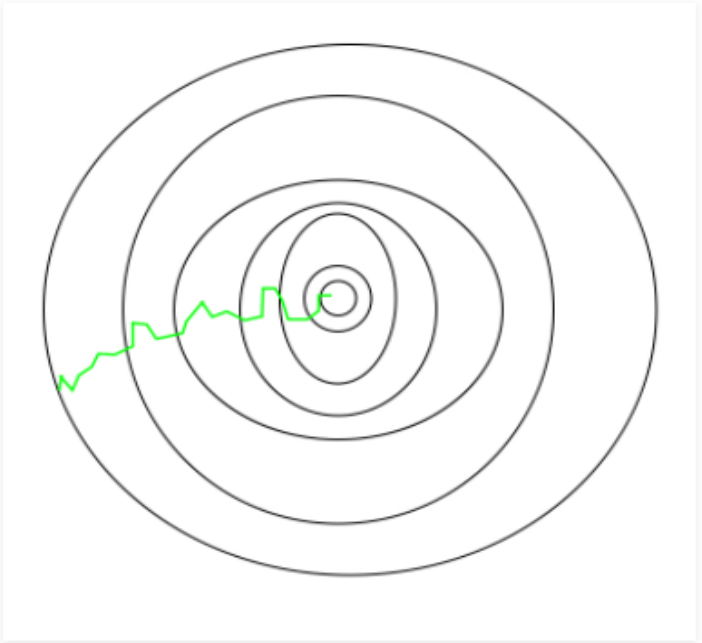
- Stochastic Gradient Descent is the extension of Gradient Descent.
- Any Machine Learning/ Deep Learning function works on the same objective function f(x) to reduce the error and generalize when a new data comes in.
- This is a type of gradient descent which processes 1 training example per iteration.
- Parameters can get updated even for one data level of processing also.
- Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large.
- It replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data)
- In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration.
- In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration. The sample is randomly shuffled and selected for performing the iteration.
- So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.

Stochastic Gradient Descent- Algorithm

Path taken by Batch Gradient Descent –



Path taken by Stochastic Gradient Descent –



The path travelled by SGD will be more noisy than GD, But as long as we reach the minima and with significantly shorter training time.

Stochastic Gradient Descent- Algorithm

The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

The disadvantages of Stochastic Gradient Descent include:

- SGD requires a number of hyperparameters such as the regularization parameter and the number of iterations.
- SGD is sensitive to feature scaling.

Note:

SGD updates the current weight using the current gradient $\partial L/\partial w$ multiplied by some factor called the learning rate, α.

$$w_{t+1} = w_t - \alpha \frac{\partial L}{\partial w_t}$$

Shallow Networks

Shallow neural networks give us basic idea about deep neural network which consist of only 1 or 2 hidden layers. Understanding a shallow neural network gives us an understanding into what exactly is going on inside a deep neural network A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input ***X***. These can also be called the ***forward-propagation*** equations.

$$Z^{[1]} = W^{[1]T} X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

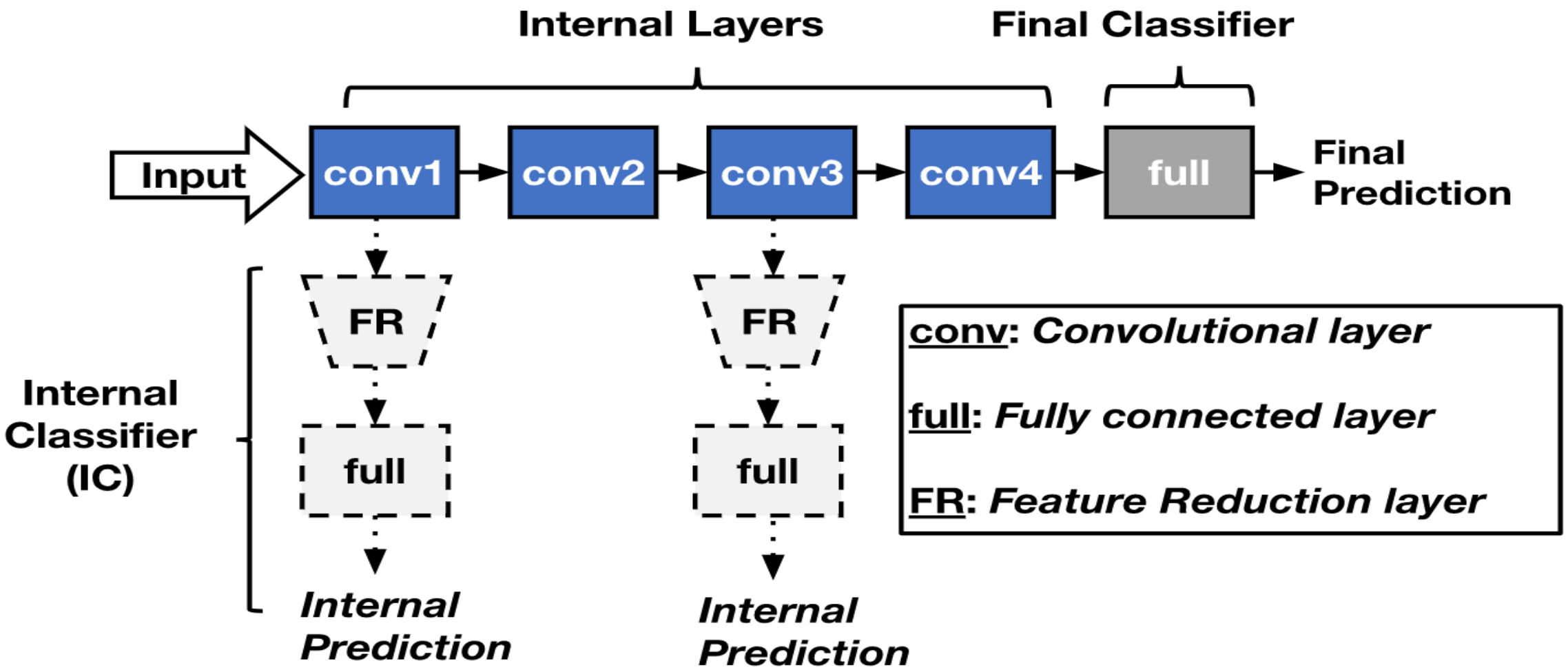
$$Z^{[2]} = W^{[2]T} A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma\left(Z^{[2]}\right)$$

1. The first equation calculates the intermediate output ***Z[1]*** of the first hidden layer.
2. The second equation calculates the final output ***A[1]*** of the first hidden layer.
3. The third equation calculates the intermediate output ***Z[2]*** of the output layer.
4. The fourth equation calculates the final output ***A[2]*** of the output layer which is also the final output of the whole neural network.

Shallow Networks

Shallow-Deep Networks: A Generic Modification to Deep Neural Networks

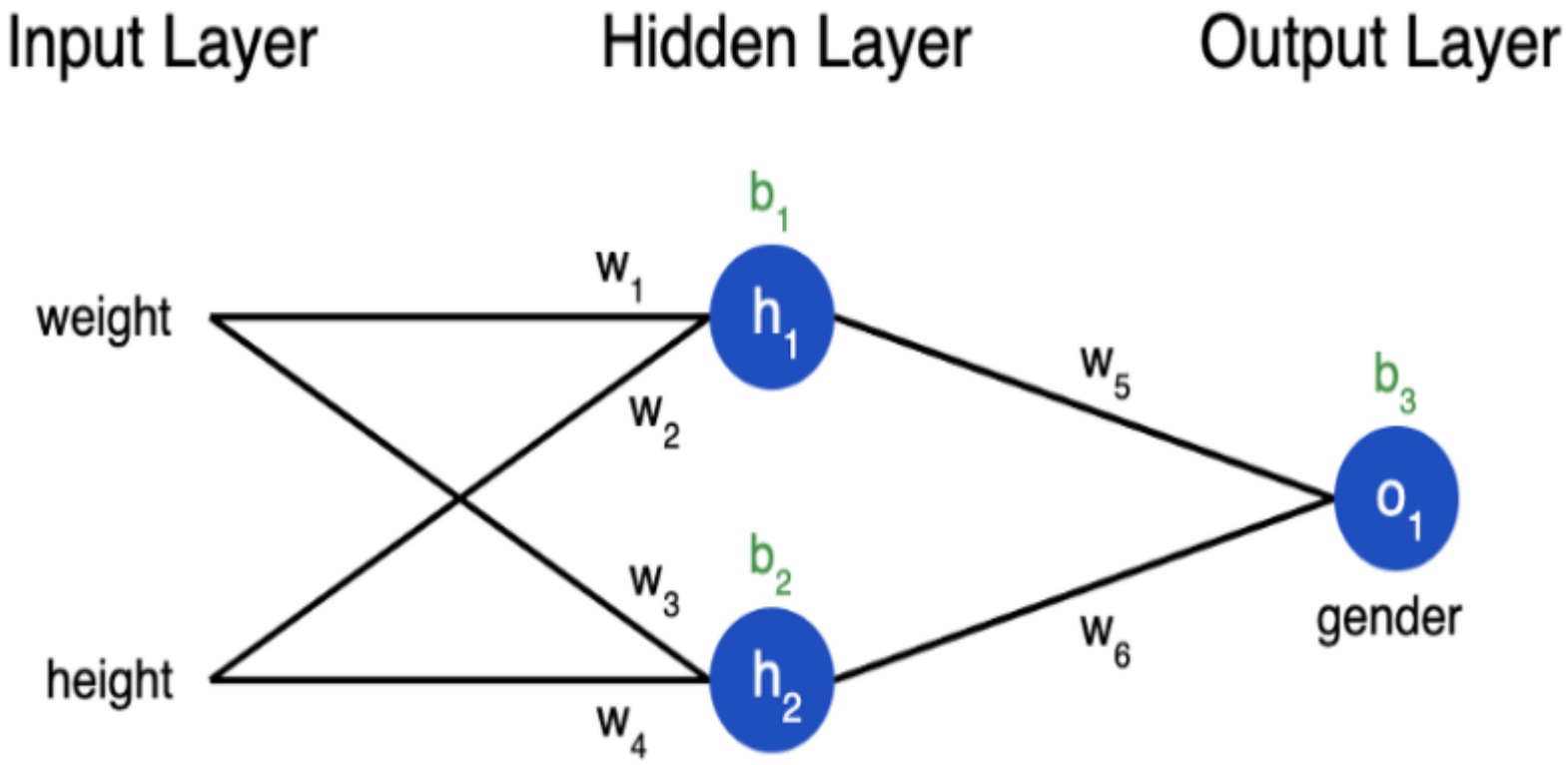


Shallow Networks

Difference Between a Shallow Net & Deep Learning Net:

Sl.No	Shallow Net's	Deep Learning Net's
1	One Hidden layer(or very less no. of Hidden Layers)	Deep Net's has many layers of Hidden layers with more no. of neurons in each layers
2	Takes input only as VECTORS	DL can have raw data like image, text as inputs
3	Shallow net's needs more parameters to have better fit	DL can fit functions better with less parameters than a shallow network
4	Shallow networks with one Hidden layer (same no of neurons as DL) cannot place complex functions over the input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially with task complexity.	DL don't need to increase it size(neurons) for complex problems
6	Shallow network is more difficult to train with our current algorithms (e.g. it has issues of local minima etc)	Training in DL is easy and no issue of local minima in DL

Neural Networks As Universal Function Approximation



The output to be predicted is GENDER from the inputs Heights & width.

Without Activation part

h_1 is a linear function of both weight and height with parameters w_1, w_2, and the bias term b_1. Therefore mathematically,

$h_1 = w_1$*weight + w_2*height + b_1

Similarly

h2 = w_3*weight + w_4*height + b_2

we realize that o1 is also a linear function of h_1 and h2, and therefore depends linearly on input attributes weight and height as well.

Neural Networks - Universal Function Approximation - Continued

- The above equations (in previous slide) seems to be Linear, but we need a Universal Approximator which approximates non linear data's also

This is where activation layers come into play.

An activation layer is applied right after a linear layer in the Neural Network to provide non-linearities.

Non-linearities help Neural Networks perform more complex tasks.

An activation layer operates on activations (h_1, h_2 in this case) and modifies them according to the activation function provided for that particular activation layer.

Activation functions are generally non-linear except for the identity function.

Some commonly used activation functions are ReLu, sigmoid, softmax, etc.

With the introduction of non-linearities along with linear terms, it becomes possible for a neural network to model any given function approximately on having appropriate parameters(w_1, w_2, b_1, etc in this case). The parameters converge to *appropriateness* on training suitably.

Points to Remember – Unit 1

- Why we need ANN?
- Types of Learning/Training
- Linear models
  - Perceptron, Regression, SVM
- Issues in Linear models
  - Need for Multi Layer networks (Linear inseparable issues)
- Back Propagation Network
  - Gradient descent
  - Stochastic Gradient Descent
- Shallow Networks
- Universal Function Approximation

Dr. V. Vedanarayanan B.E., M.E., PhD
Assistant Professor, Department of ECE,
School of Electrical and Electronics
Sathyabama Institute of Science and technology
Vedanarayanan.etc@sathyabama.ac.in