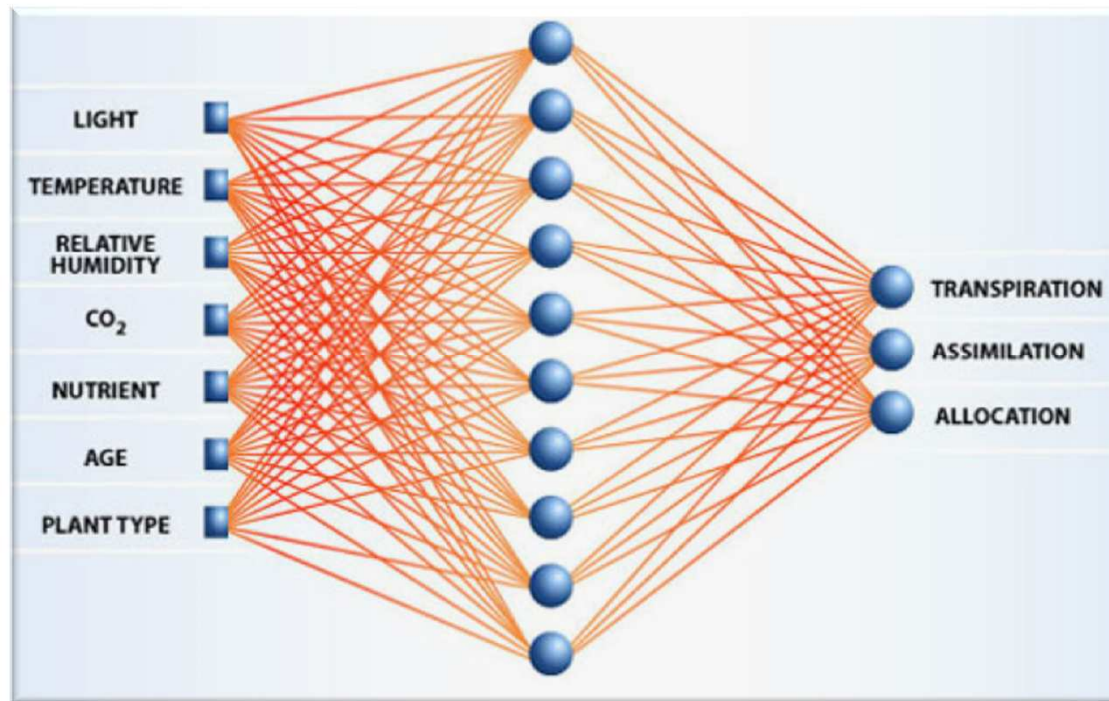


**Neural Networks
and
Support Vector Machines**

NEURAL NETWORKS

Neural Networks

- A **neural network** is a set of connected input/output units (neurons) where each connection has a weight associated with it.



http://aemc.jpl.nasa.gov/activities/bio_regen.cfm

Neural Networks

- During the **learning phase**, the network learns by adjusting the weights that enable it to predict the correct class label of the input samples (the training samples).
 - Knowledge about the learning task is given in the form of examples.
 - Inter neuron connection strengths (**weights**) are used to **store** the acquired **information (the training examples)**.
 - During the **learning process** the weights are modified in order to model the particular learning task correctly on the training examples.

Neural Networks

■ Advantages

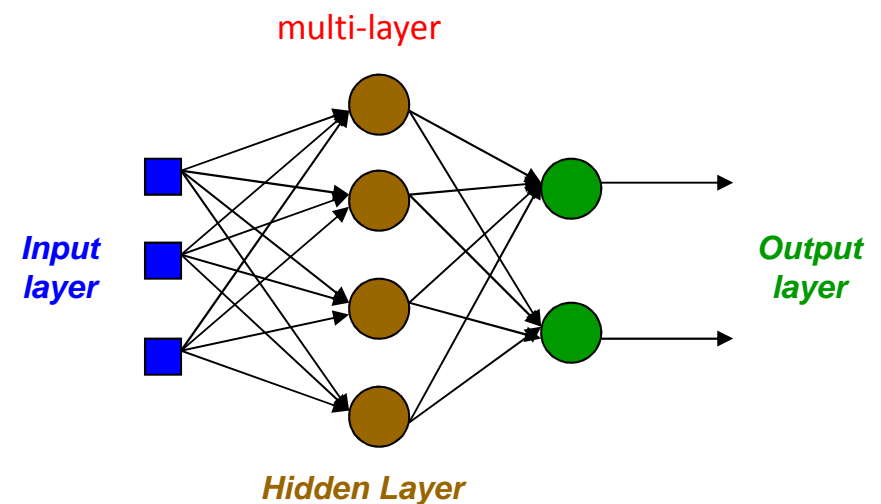
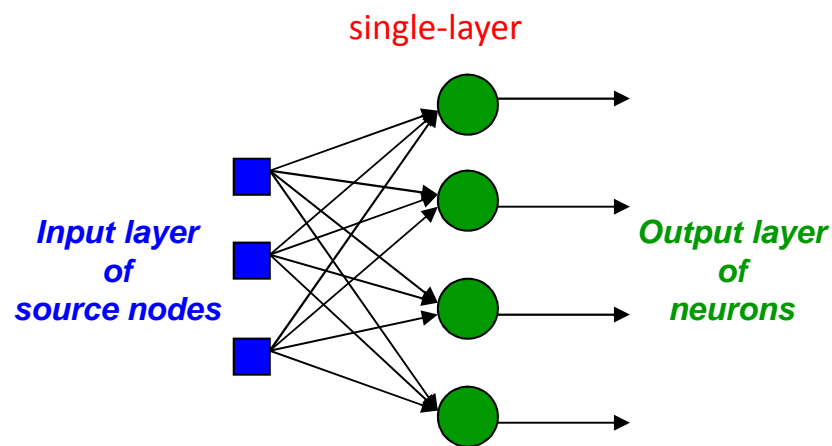
- prediction accuracy is generally high
- robust, works when training examples contain errors or noisy data
- output may be discrete, real-valued, or a vector of several discrete or real-valued attributes
- fast evaluation of the learned target function

■ Criticism

- parameters are best determined empirically, such as the network topology or structure
- long training time
- difficult to understand the learned function (weights)
- not easy to incorporate domain knowledge

Network architectures

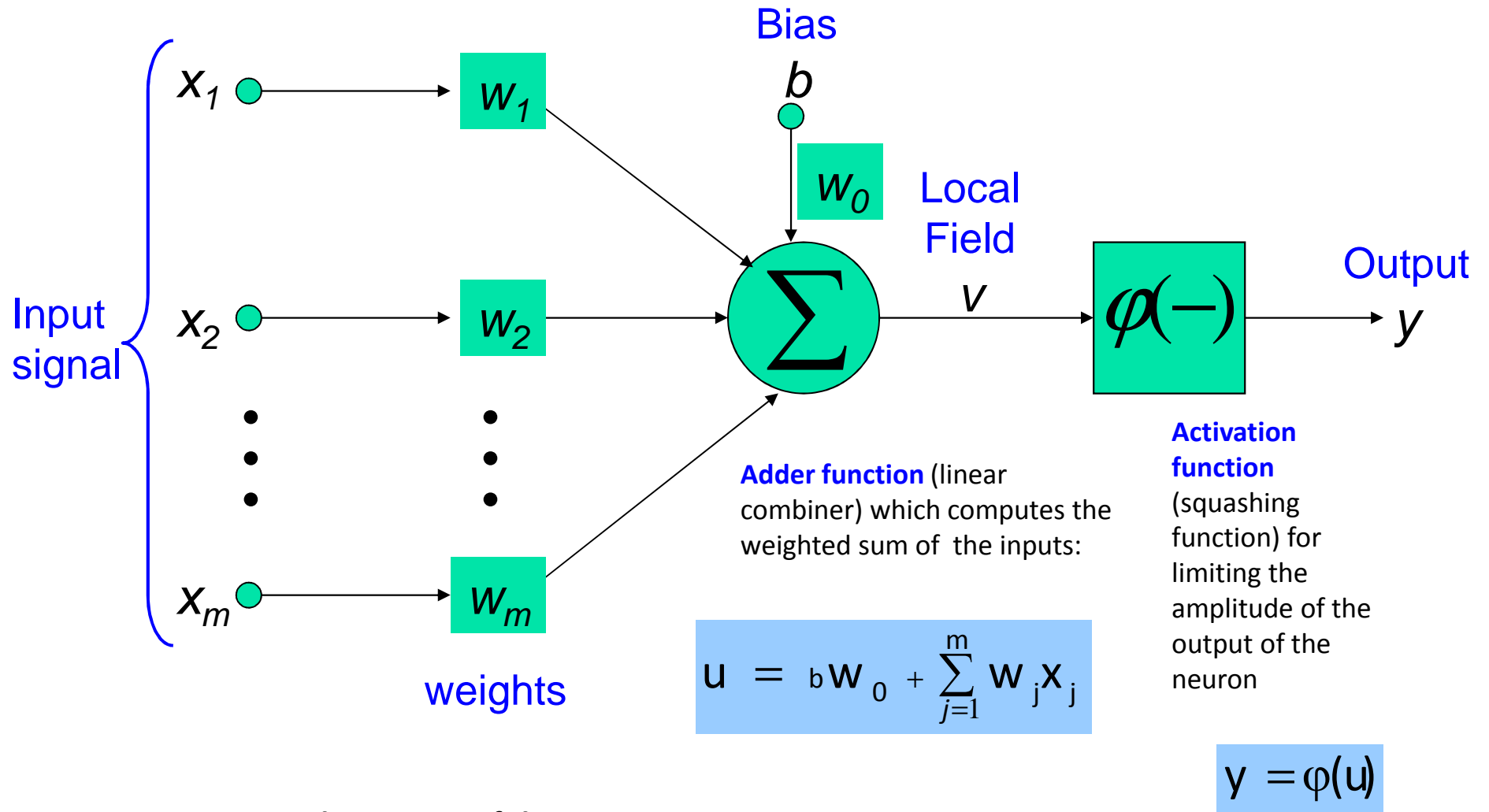
- Three different classes of network architectures
 - single-layer feed-forward
 - multi-layer feed-forward } *neurons are organized in acyclic layers*
 - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train



Neurons

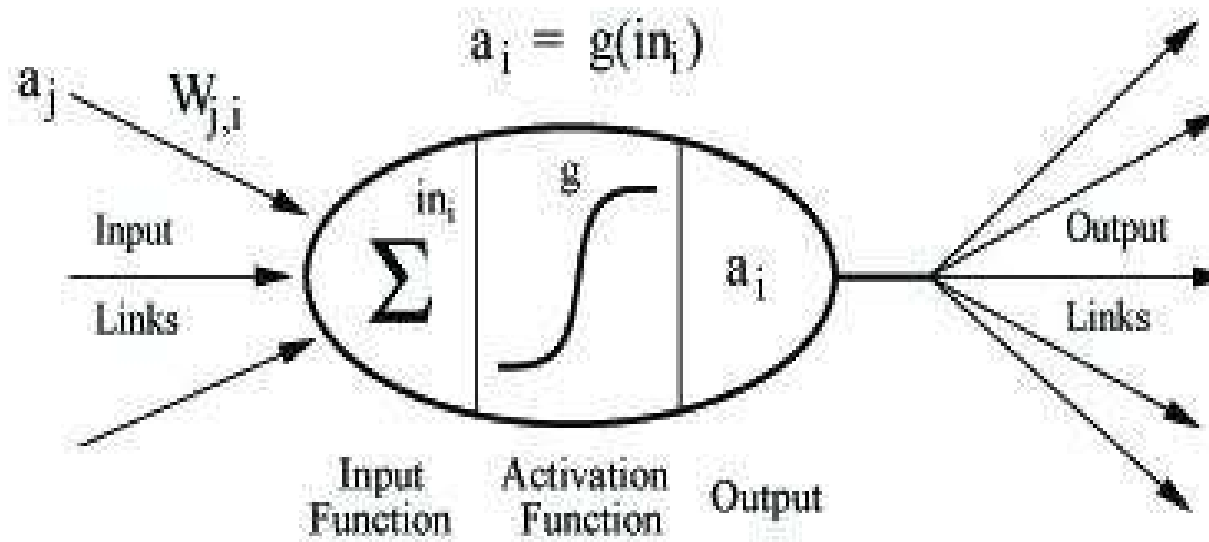
- Neural networks are built out of a densely interconnected set of simple units (**neurons**)
 - Each neuron takes a number of real-valued inputs
 - Produces a single real-valued output
 - Inputs to a neuron may be the outputs of other neurons.
 - A neuron's output may be used as input to many other neurons

The neuron



Bias: serves to vary the activity of the unit

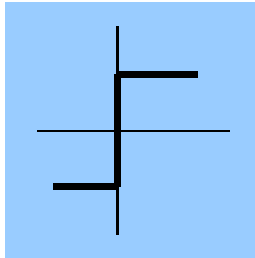
The neuron



<http://www-cse.uta.edu/~cook/ai1/lectures/figures/neuron.jpg>

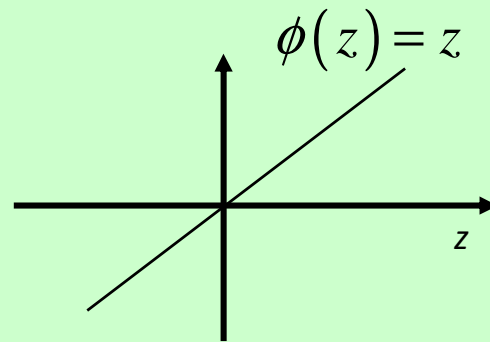
How does it Works?

- Assign weights to each input-link
- Multiply each weight by the input value (0 or 1)
- Sum all the weight-firing input combinations
- Apply squash function, e.g.:
 - If $\text{sum} > \text{threshold}$ for the Neuron then
 - Output = +1
 - Else Output = -1

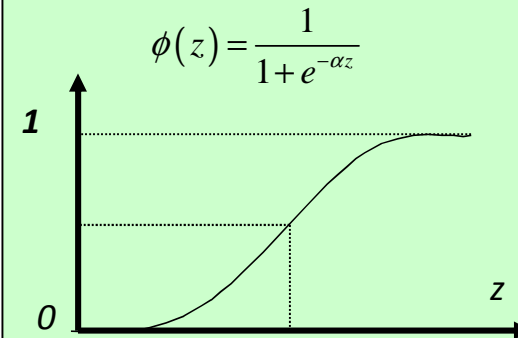


Popular activation functions

Linear activation

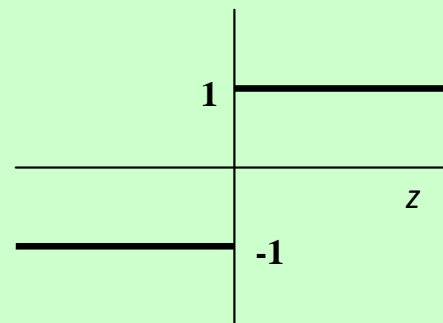


Logistic activation



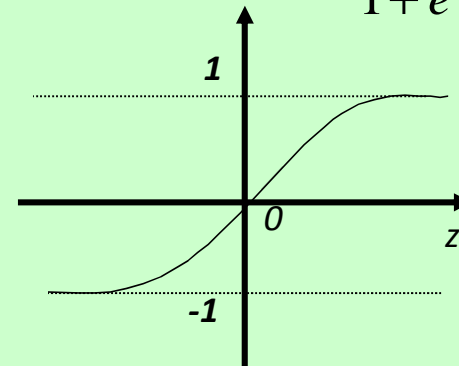
Threshold activation

$$\phi(z) = \text{sign}(z) = \begin{cases} 1, & \text{if } z \geq 0, \\ -1, & \text{if } z < 0. \end{cases}$$



Hyperbolic tangent activation

$$\varphi(u) = \tanh(\gamma u) = \frac{1 - e^{-2\gamma u}}{1 + e^{-2\gamma u}}$$

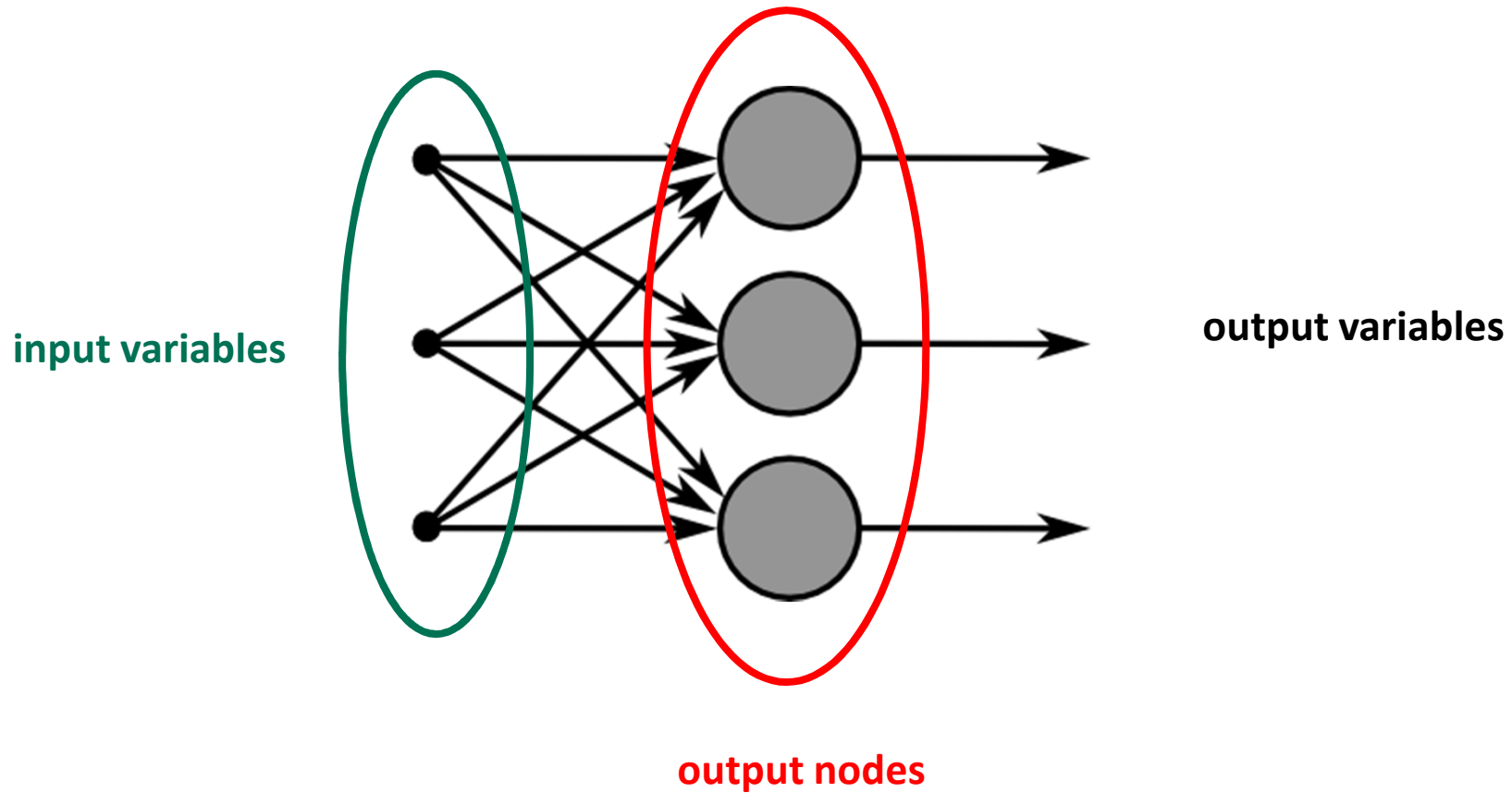


How Are Neural Networks Trained?

- Initially
 - choose small random weights (w_i)
 - Set threshold = 1 (step function)
 - Choose small *learning rate* (r)
- Apply each member of the *training set* to the neural net model using a *training rule* to adjust the weights
 - For each unit
 - Compute the net input to the unit as a linear combination of all the inputs to the unit
 - Compute the output value using the activation function
 - Compute the error
 - Update the weights and the bias

Single Layer Perceptron

Are the simplest form of neural networks



Single layer perceptron: training rule

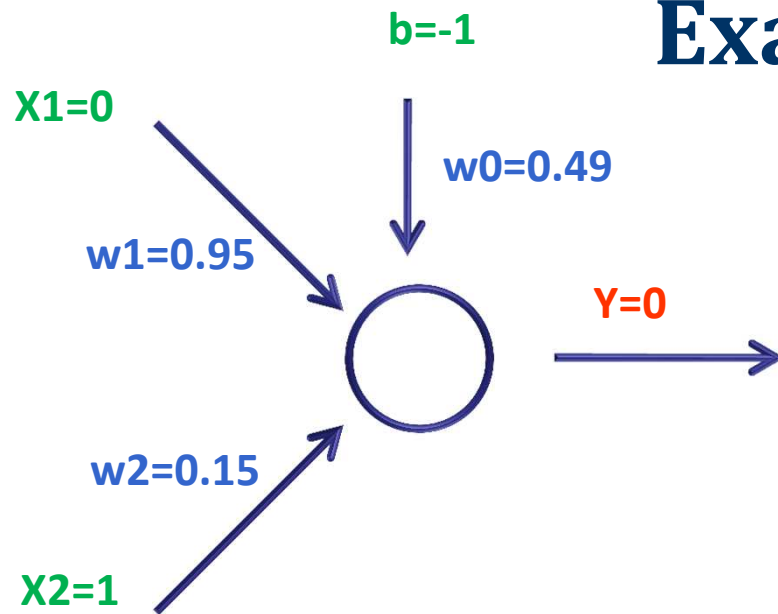
- Modify the *weights* (w_i) according to the *Training Rule*:

$$w_i = w_i + r \cdot (t - a) \cdot x_i$$

- r is the *learning rate* (eg. 0.2)
- t = target output
- a = actual output
- x_i = i -th input value

Learning rate: if too small learning occurs at a small pace, if too large it may stuck in local minimum in the decision space

Example



x1	x2	Y
0	0	0
1	0	1
0	1	1
1	1	1

threshold = 0.5
 $r = 0.05$

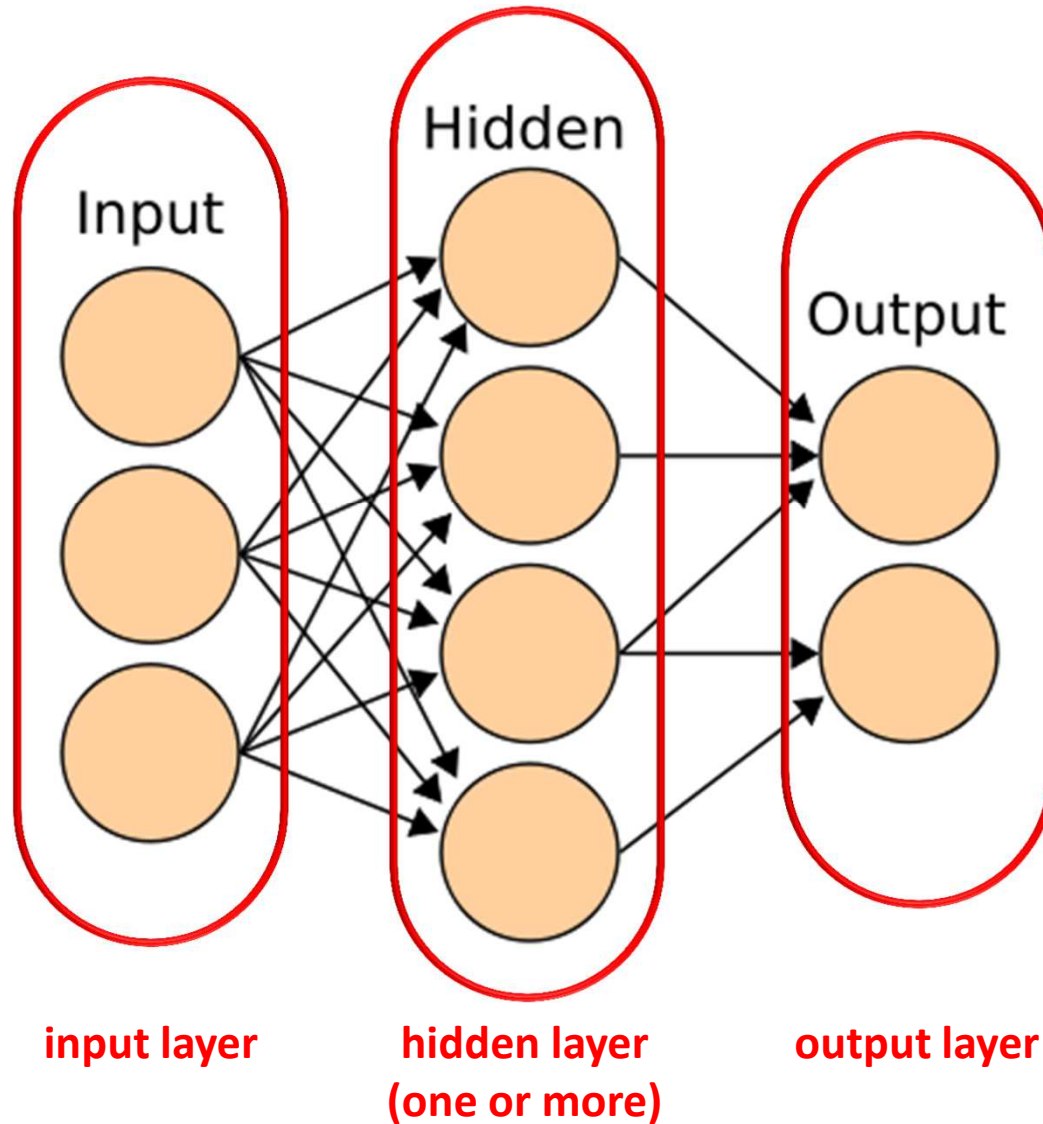
Compute output for the input $u = -1 \times 0.49 + 0 \times 0.95 + 1 \times 0.15 = -0.34 < t$
thus, $y = 0$

Compute the error
target output = 1
actual output (y) = 0
error = $(1 - 0) = 1$
correction factor = error $\times r = 0.05$

Compute the new weights
 $w_0 = 0.49 + 0.05 \times (1 - 0) \times (-1) = 0.44$
 $w_1 = 0.95 + 0.05 \times (1 - 0) \times 0 = 0.95$
 $w_2 = 0.15 + 0.05 \times (1 - 0) \times 1 = 0.20$

Repeat the process with the new weights for a given number of iterations

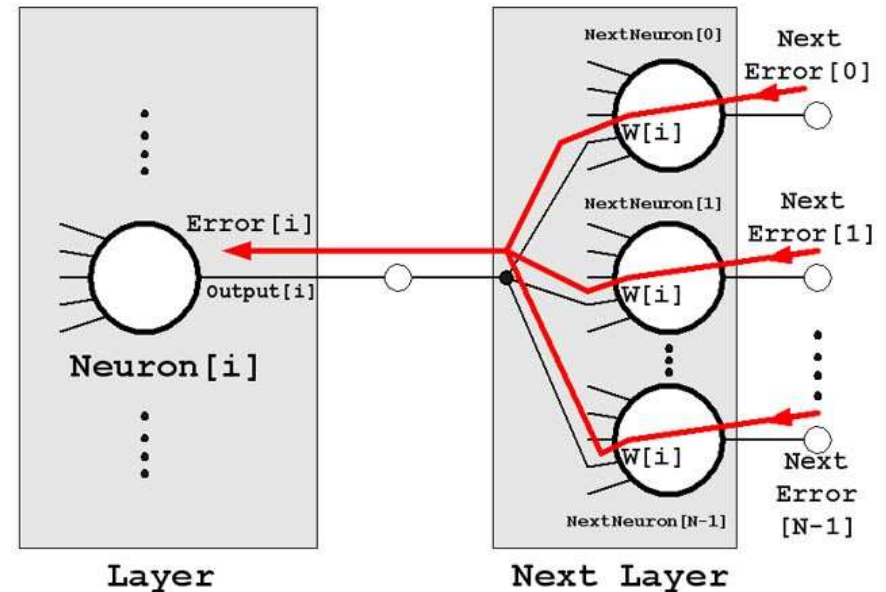
Multi layer network



Training multi layer networks

back-propagation algorithm

- **Phase 1: Propagation**
 - Forward propagation of a training input
 - Back propagation of the propagation's output activations.
- **Phase 2: Weight update**
 - For each weight-synapse:
 - Multiply its output delta and input activation to get the gradient of the weight.
 - Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.
 - This ratio influences the speed and quality of learning. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.
 - Repeat the phase 1 and 2 until the performance of the network is good enough.



Multi-Layer network of sigmoid units

Problem: what is the desired output for a hidden node? => Backpropagation algorithm

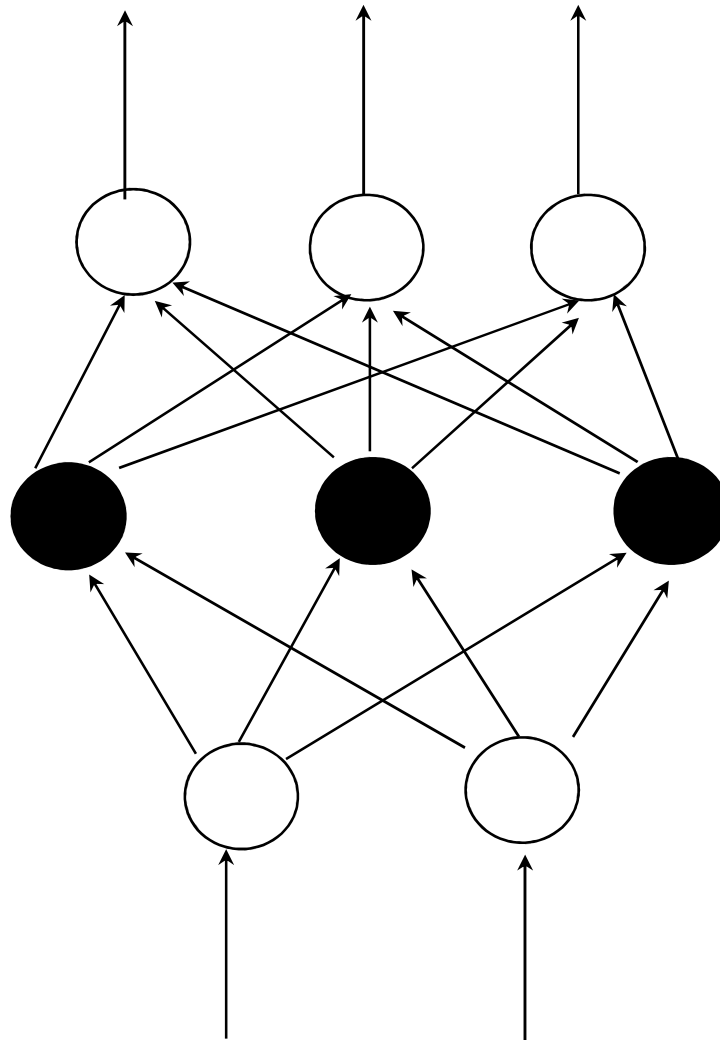
Output vector

Output nodes

Hidden nodes

Input nodes

Input vector: x_i



$$\theta_j = \theta_j + (r)Err_j$$

to update the bias

$$w_{ij} = w_{ij} + (r)Err_j O_i$$

to update the weights

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$$

error for a node in the hidden layer

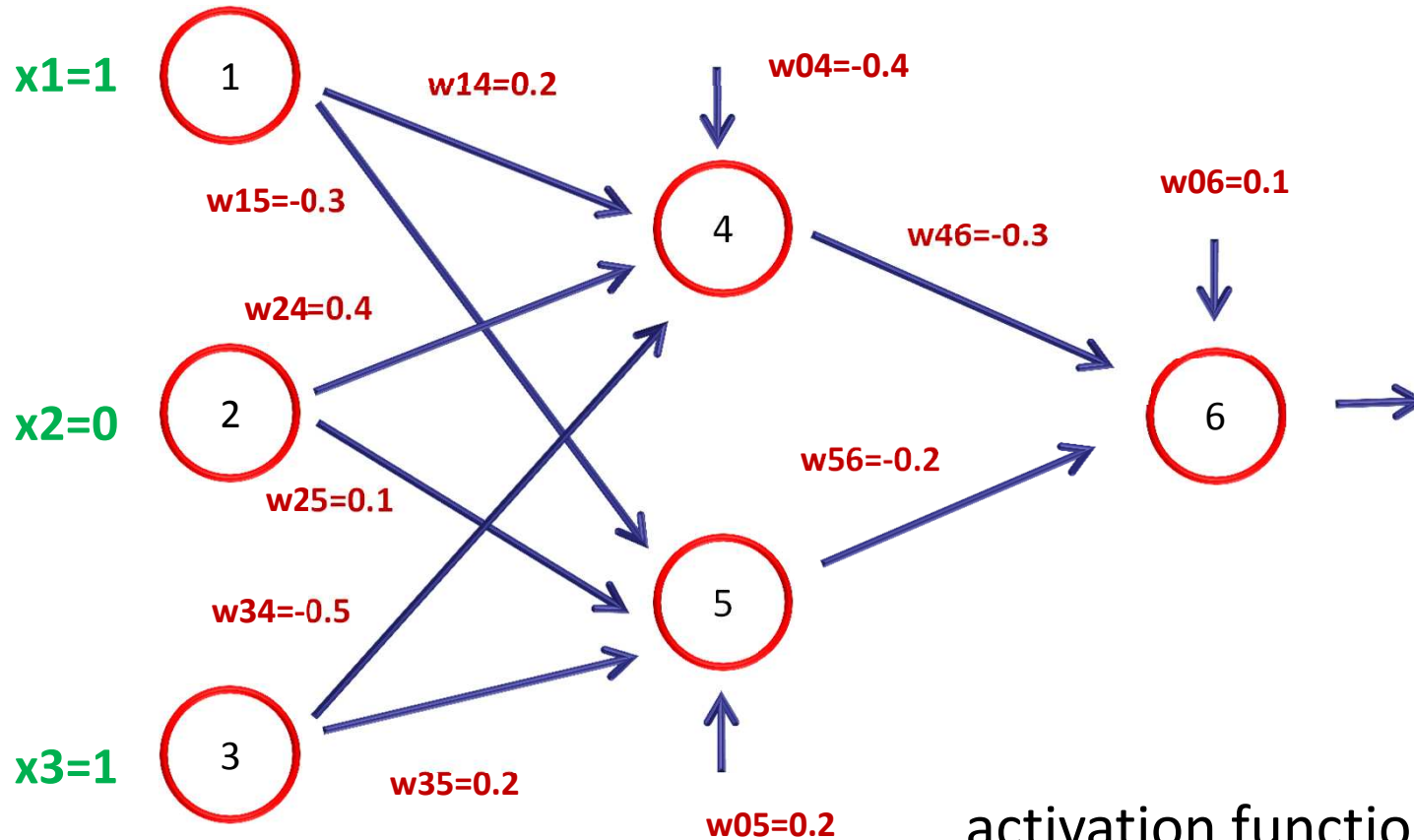
$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

error for a node in the output layer

$$O_j = \frac{1}{1 + e^{-I_j}}$$

$$I_j = \sum_i w_{ij} O_i + \theta_j$$

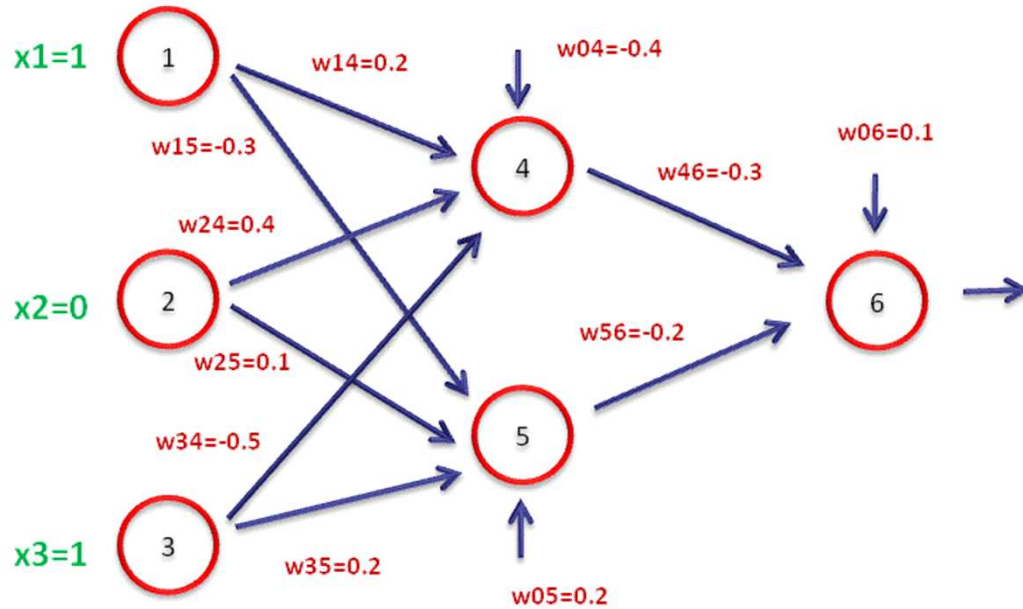
Example



x_i – input variables (1,0,1) whose class is 1
 w_{ij} – randomly assigned weights

activation function
 $O_j = 1 / (1 + e^{-I_j})$
and
learning rate = 0.9

Propagation

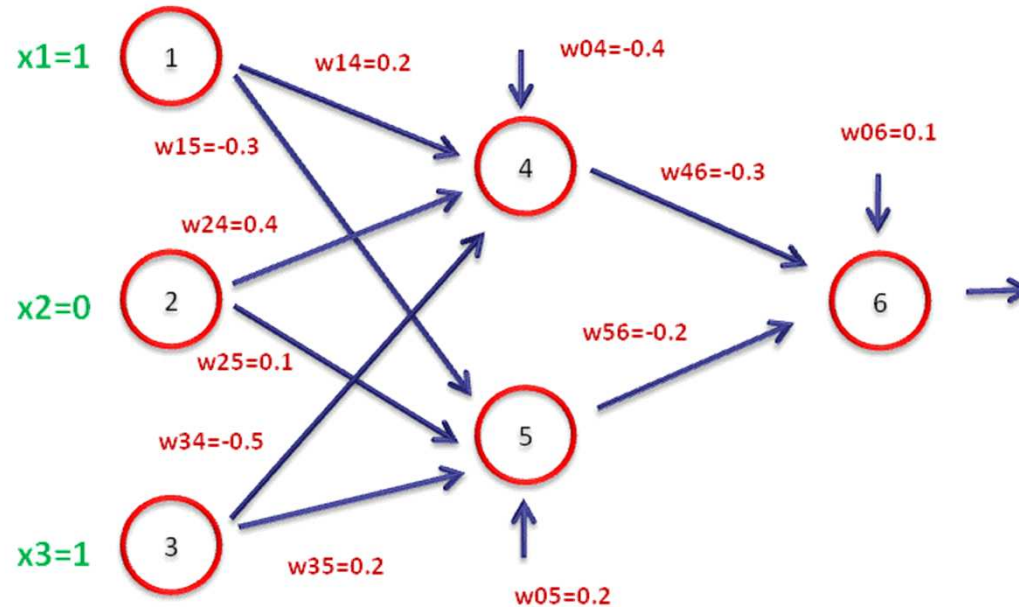


$$I_j = \sum_i w_{ij} O_i + \theta_j$$

$$O_j = \frac{1}{1 + e^{-I_j}}$$

neuron	input	output
4	$0.2x_1 + 0.4x_2 - 0.5x_3 - 0.4 = -0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3x_1 + 0.1x_2 + 0.2x_3 + 0.2 = 0.1$	$1/(1+e^{-0.1})=0.525$
6	$-0.3 \times 0.332 - 0.2 \times 0.525 + 0.1 = -0.105$	$1/(1+e^{0.105})=0.474$

Calculation of the neuron'



error for a node in the output layer

$$Err_j = O_j(1 - O_j)(T_j - O_j)$$

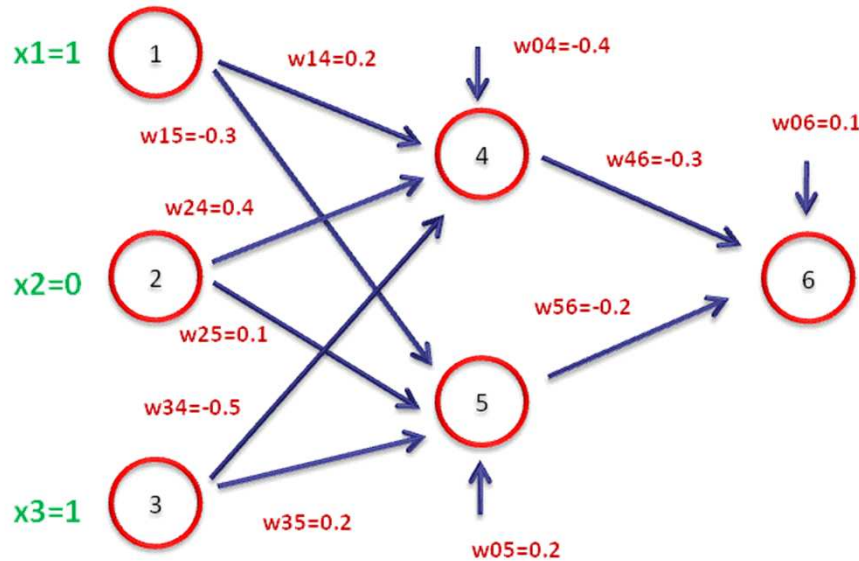
error for a node in the hidden layer

$$Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$$

neuron	output
4	0.332
5	0.525
6	0.474

neuron	error
6	$0.474 \times (1 - 0.474) \times (1 - 0.474) = 0.1311$
5	$0.525 \times (1 - 0.525) \times (-0.2) \times 0.1311 = -0.0065$
4	$0.332 \times (1 - 0.332) \times (-0.3) \times 0.1311 = -0.0087$

Updating weights



to update the weights

$$w_{ij} = w_{ij} + (r)Err_j O_i$$

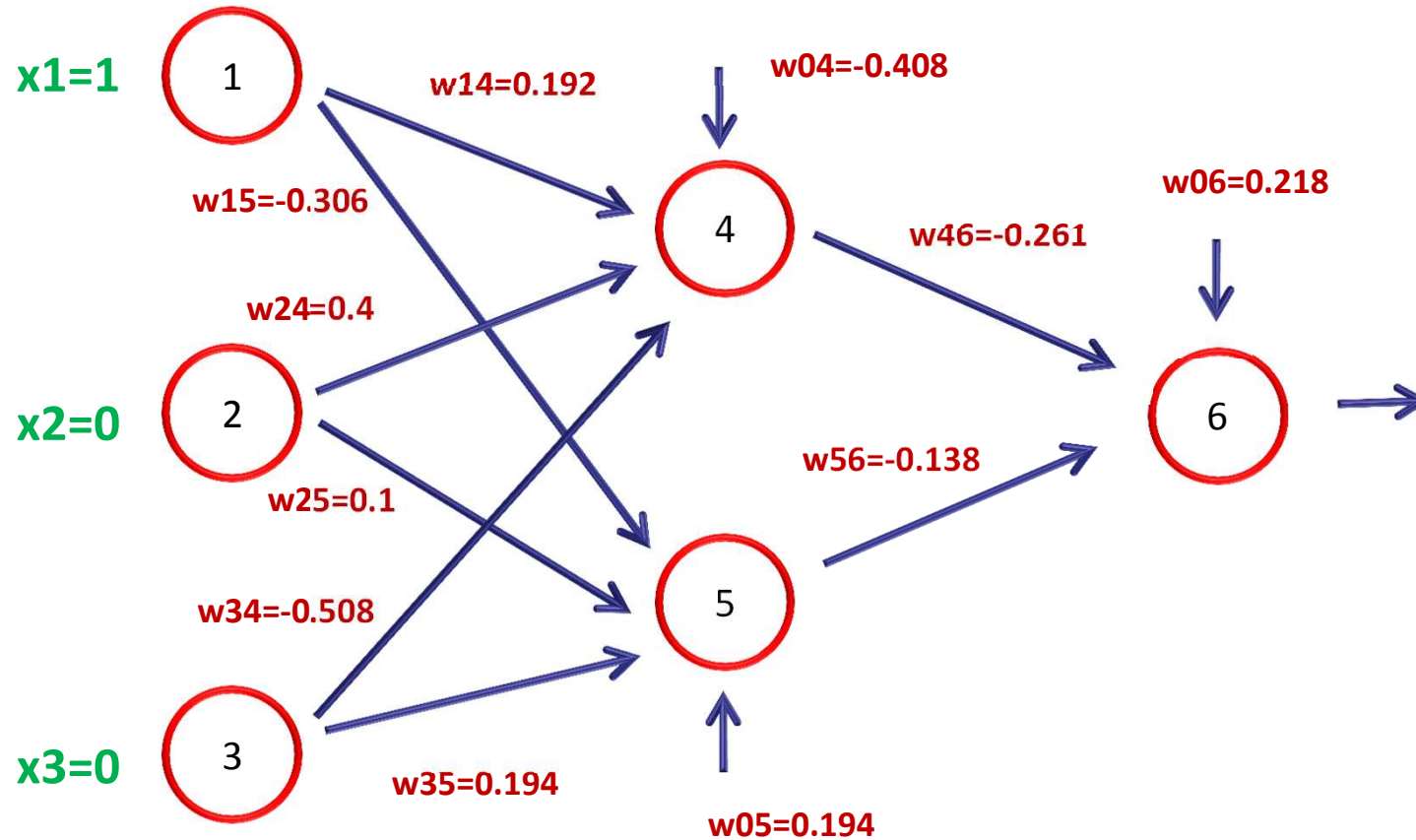
to update the bias

$$\theta_j = \theta_j + (r)Err_j$$

neuron	output	error
4	0.332	-0.0087
5	0.525	-0.0065
6	0.474	0.1311

weight	New value
w46	$-0.3 + 0.9 \times 0.1311 \times 0.332 = -0.261$
w56	$-0.2 + 0.9 \times 0.1311 \times 0.525 = -0.138$
w14	$0.2 + 0.9 \times -0.0087 \times 1 = 0.192$
w15	$-0.3 + 0.9 \times -0.0065 \times 1 = -0.306$
w24	$0.4 + 0.9 \times -0.0087 \times 0 = 0.4$
w25	$0.1 + 0.9 \times -0.0065 \times 0 = 0.1$
w34	$-0.5 + 0.9 \times -0.0087 \times 1 = -0.508$
w35	$0.2 + 0.9 \times -0.0065 \times 1 = 0.194$
w06	$0.1 + 0.9 \times 0.1311 = 0.218$
w05	$0.2 + 0.9 \times -0.0065 = 0.194$
w04	$-0.4 + 0.9 \times -0.0087 = -0.408$

Example



This is the resulting network after the first iteration. We now have to process another training example until the overall error is low or we run out of examples.

Neural Network as a Classifier

- Weakness
 - Long training time
 - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
 - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
 - High tolerance to noisy data
 - Ability to classify untrained patterns
 - Well-suited for continuous-valued inputs and outputs
 - Successful on a wide array of real-world data
 - Algorithms are inherently parallel

SUPPORT VECTOR MACHINES

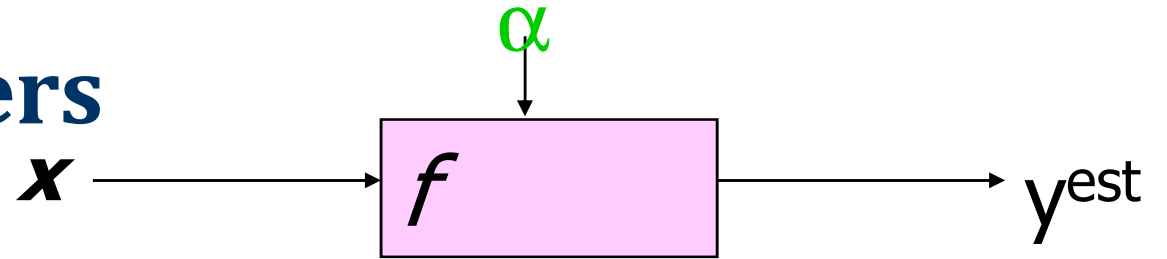
SVM—Support Vector Machines

- A new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating hyperplane (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using support vectors (“essential” training tuples) and margins (defined by the support vectors)

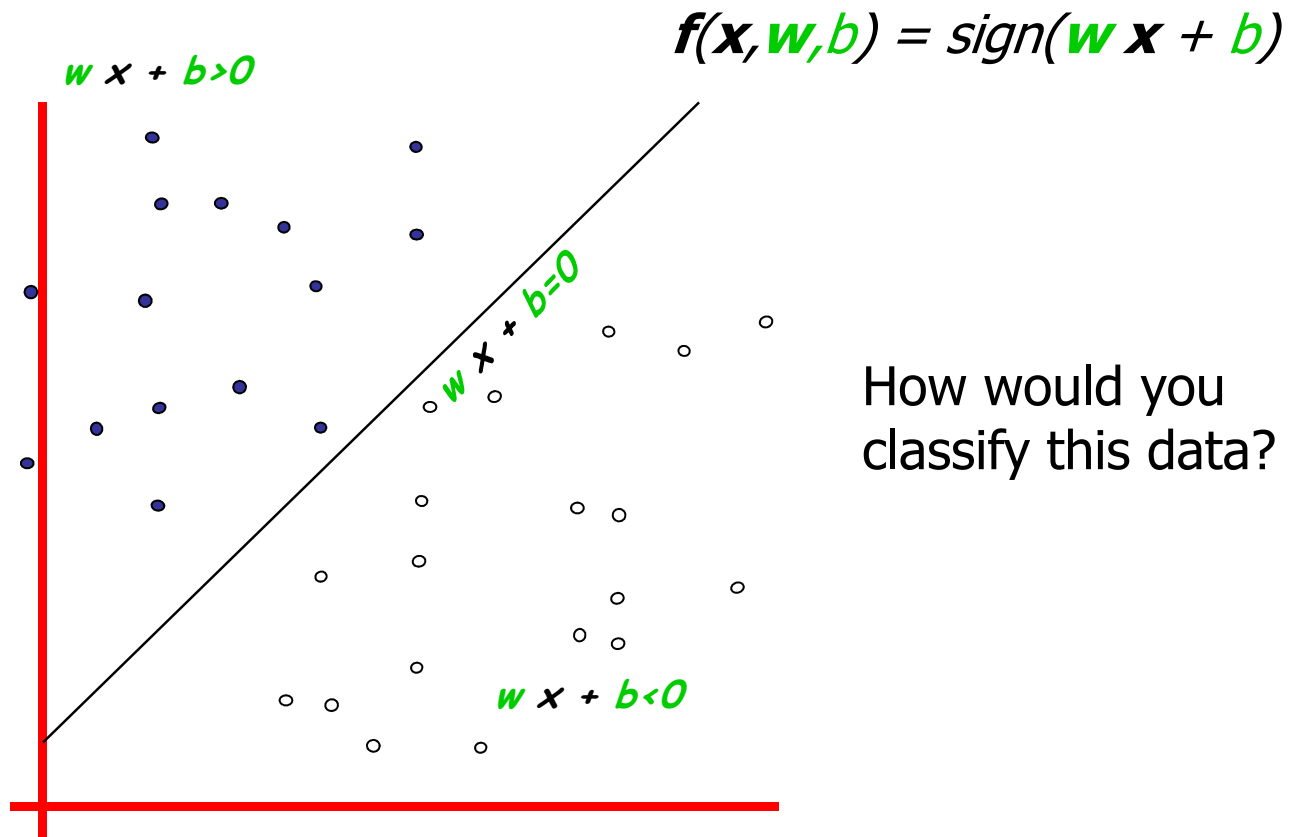
SVM—History and Applications

- Vapnik and colleagues (1992)—groundwork from Vapnik & Chervonenkis' statistical learning theory in 1960s
- Features: training can be slow but accuracy is high owing to their ability to model complex nonlinear decision boundaries (margin maximization)
- Used both for classification and regression
- Applications:
 - handwritten digit recognition, object recognition, speaker identification, benchmarking time-series prediction tests

Linear Classifiers

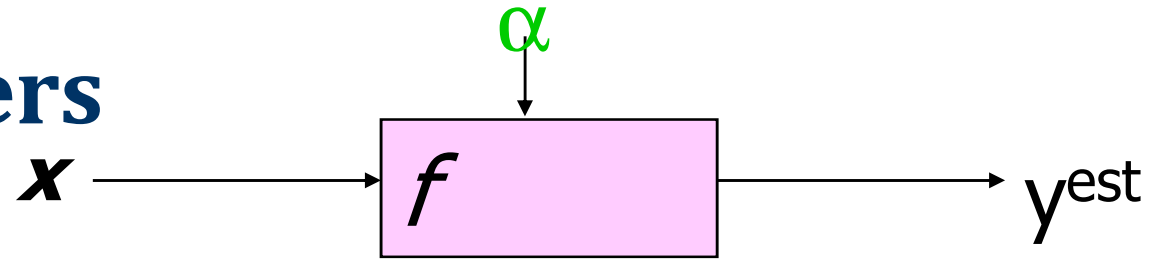


- denotes +1
- denotes -1



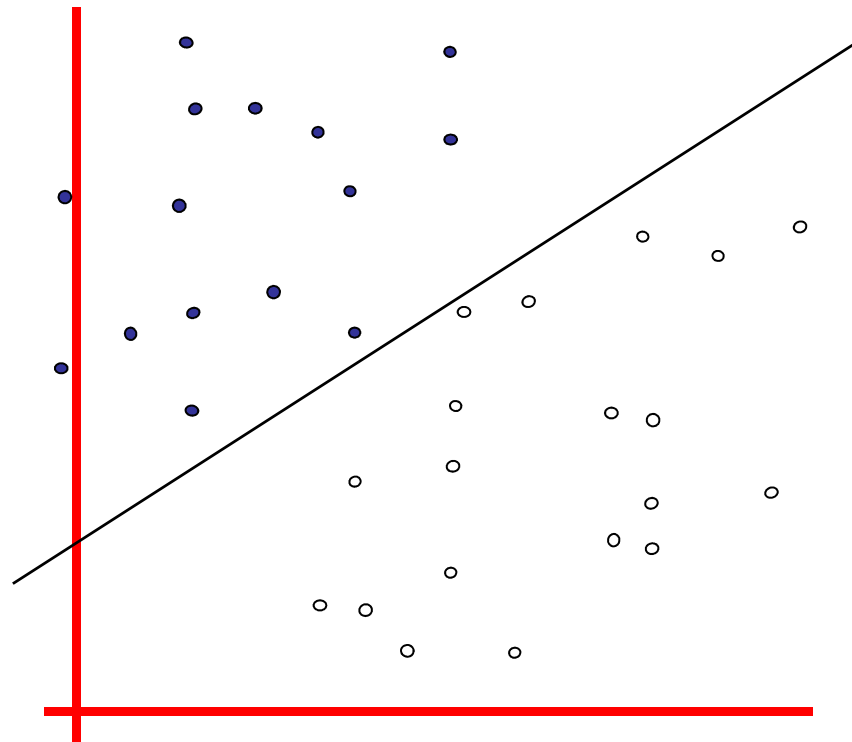
How would you classify this data?

Linear Classifiers



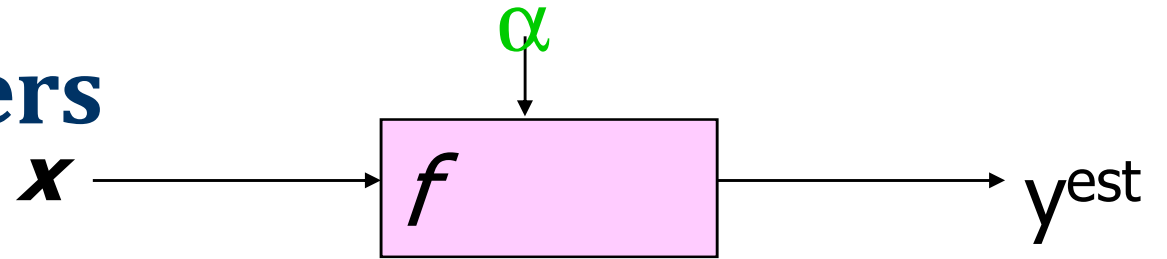
$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

- denotes +1
- denotes -1

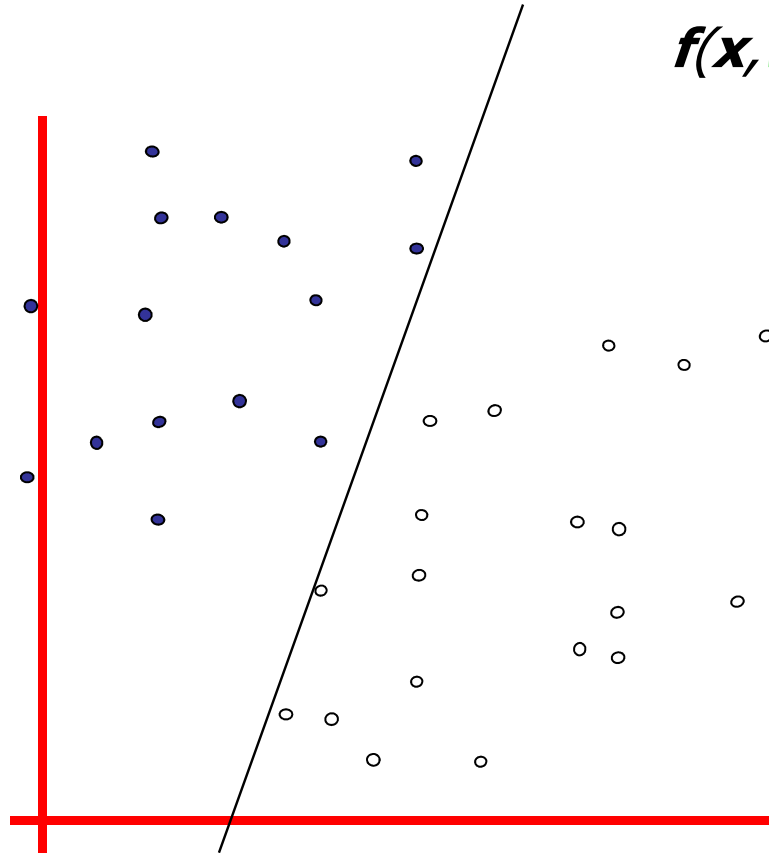


How would you classify this data?

Linear Classifiers



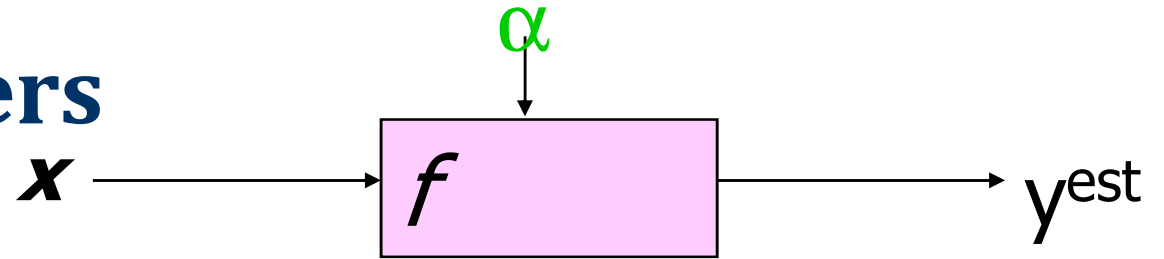
- denotes +1
- denotes -1



$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

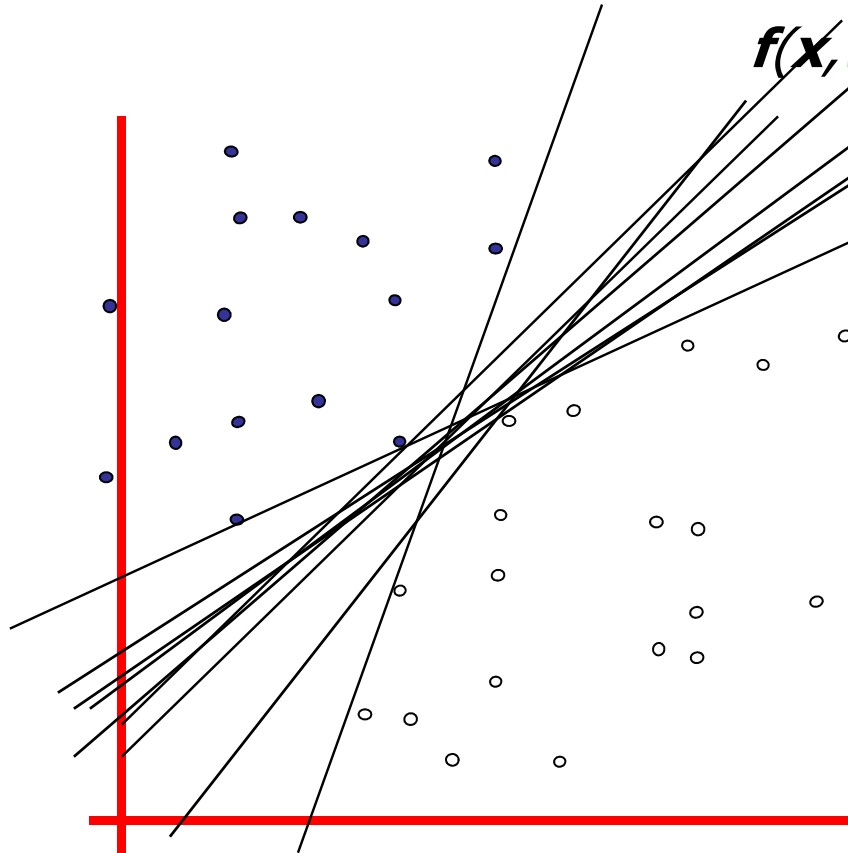
How would you classify this data?

Linear Classifiers



- denotes +1
- denotes -1

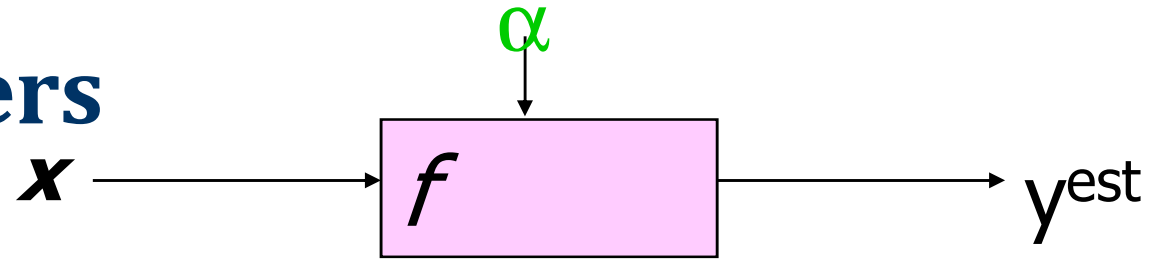
$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$



Any of these
would be fine..

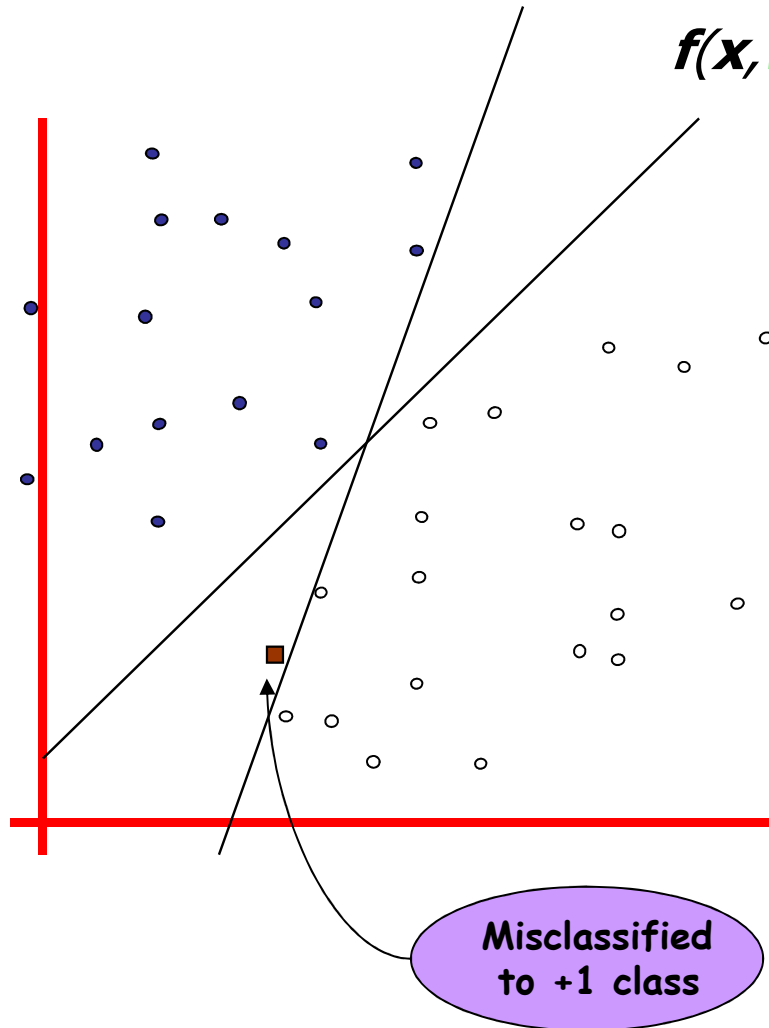
..but which is
best?

Linear Classifiers

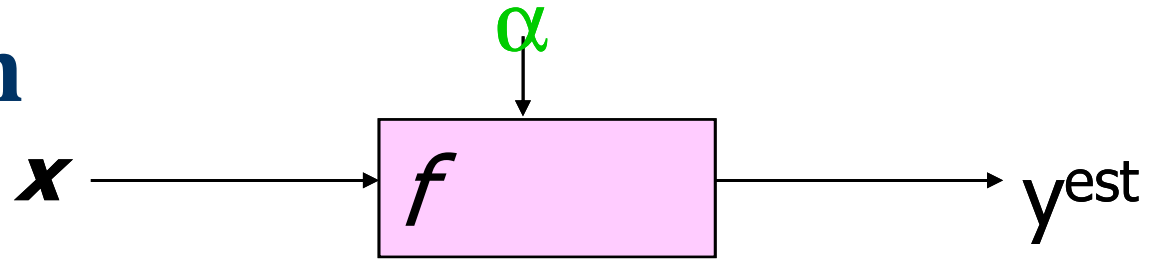


- denotes +1
- denotes -1

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$

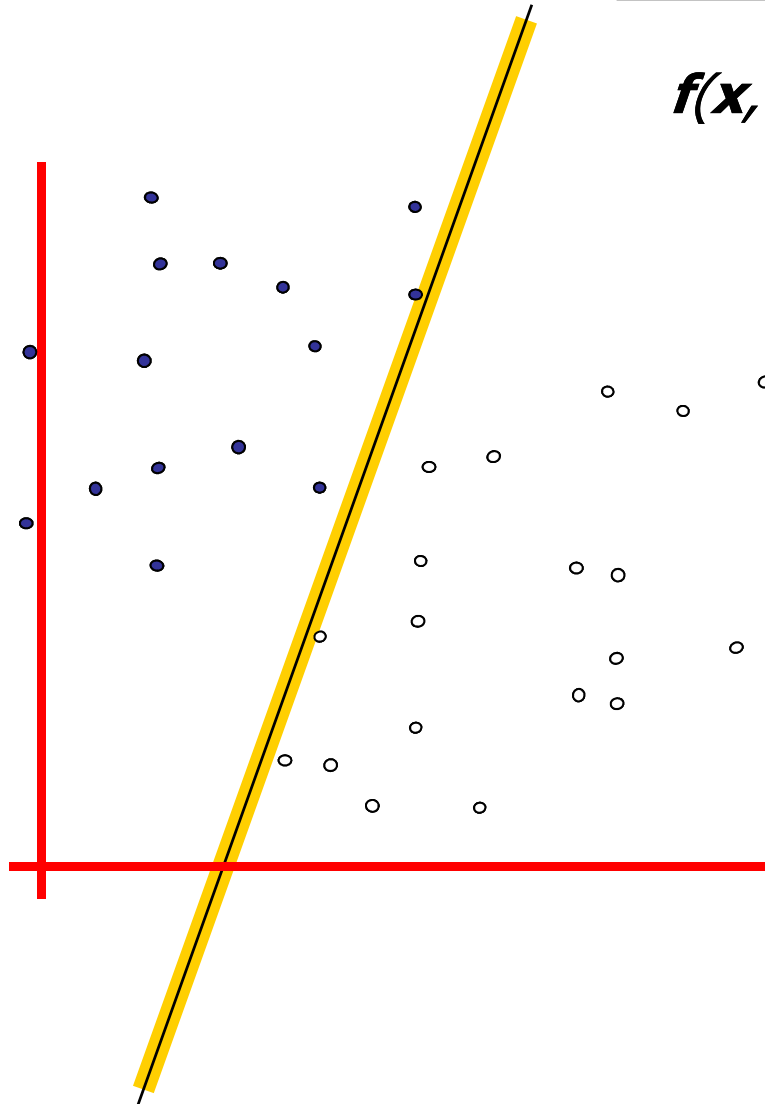


Classifier Margin



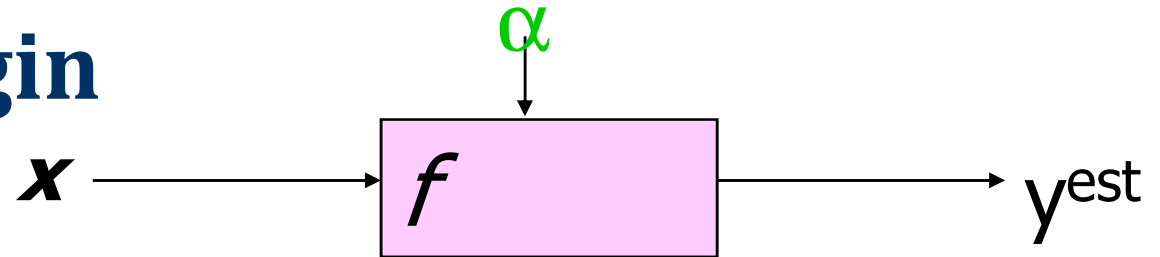
- denotes +1
- denotes -1

$$f(\mathbf{x}, \mathbf{w}, b) = \text{sign}(\mathbf{w} \mathbf{x} + b)$$



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

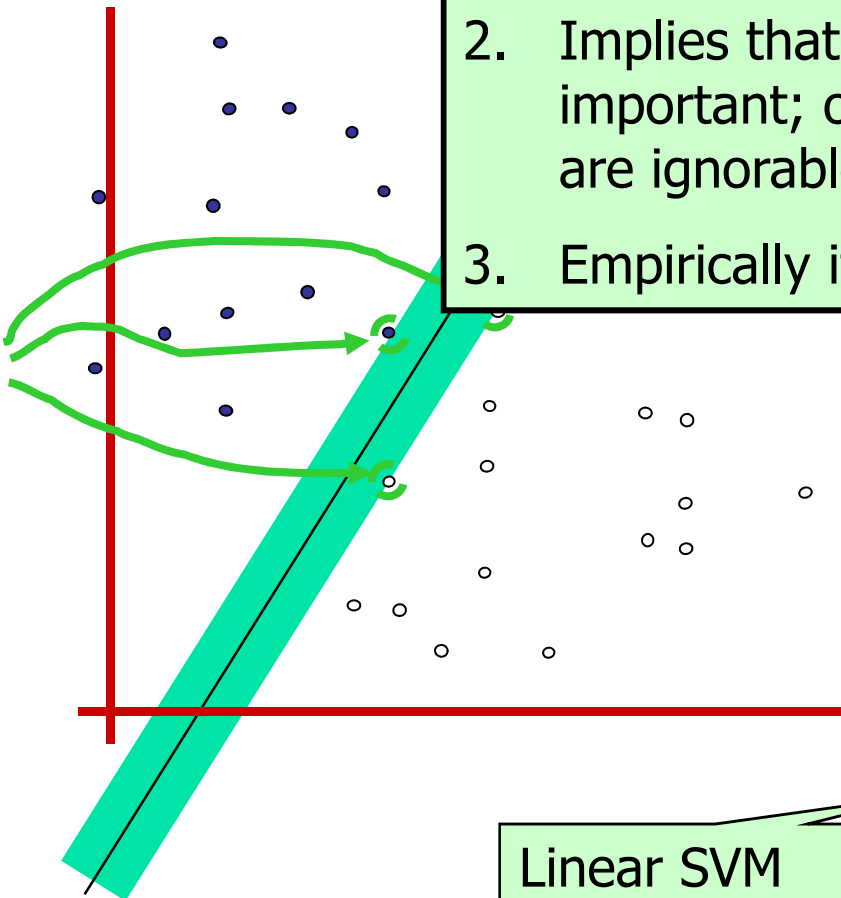
Maximum Margin



- denotes +1
- denotes -1

1. Maximizing the margin is good according to intuition and PAC theory
2. Implies that only support vectors are important; other training examples are ignorable.
3. Empirically it works very very well.

Support Vectors are those datapoints that the margin pushes up against

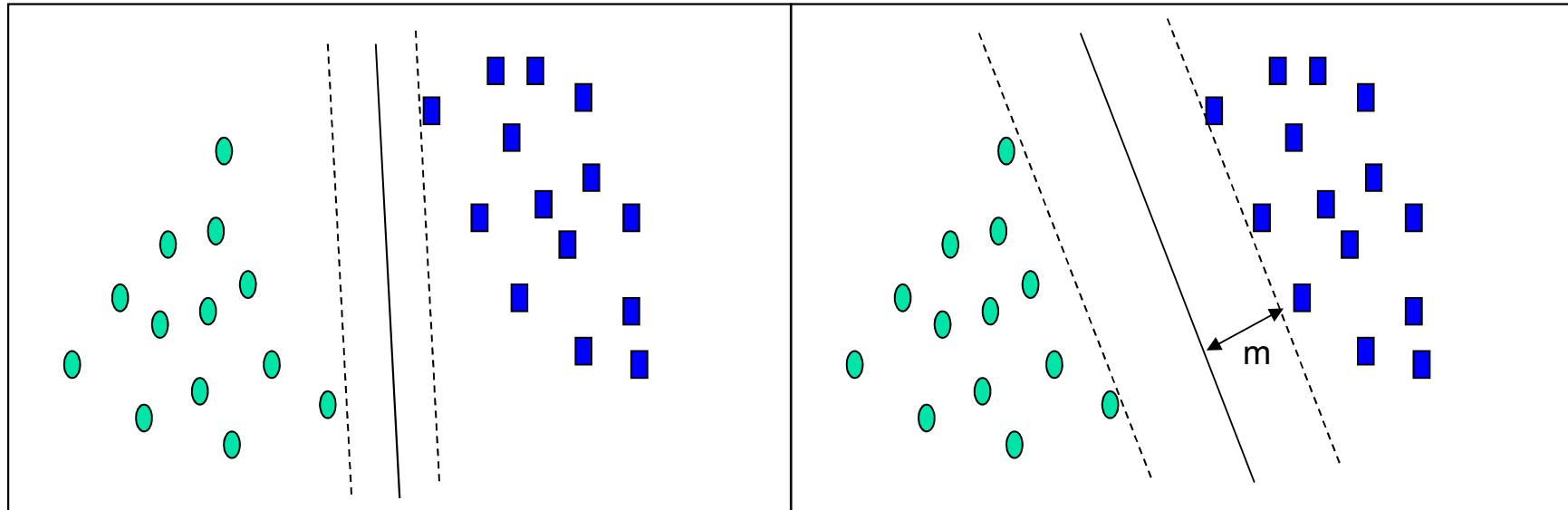


linear classifier with the, um, maximum margin.

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

Linear SVM

SVM—When Data Is Linearly Separable



Let data D be $(\mathbf{X}_1, y_1), \dots, (\mathbf{X}_{|D|}, y_{|D|})$, where \mathbf{X}_i is the set of training tuples associated with the class labels y_i

There are infinite lines (hyperplanes) separating the two classes but we want to find the best one (the one that minimizes classification error on unseen data)

SVM searches for the hyperplane with the largest margin, i.e., **maximum marginal hyperplane (MMH)**

SVM—Linearly Separable

- A separating hyperplane can be written as

$$\mathbf{W} \cdot \mathbf{X} + b = 0$$

where $\mathbf{W} = \{w_1, w_2, \dots, w_n\}$ is a weight vector and b a scalar (bias)

- For 2-D it can be written as

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

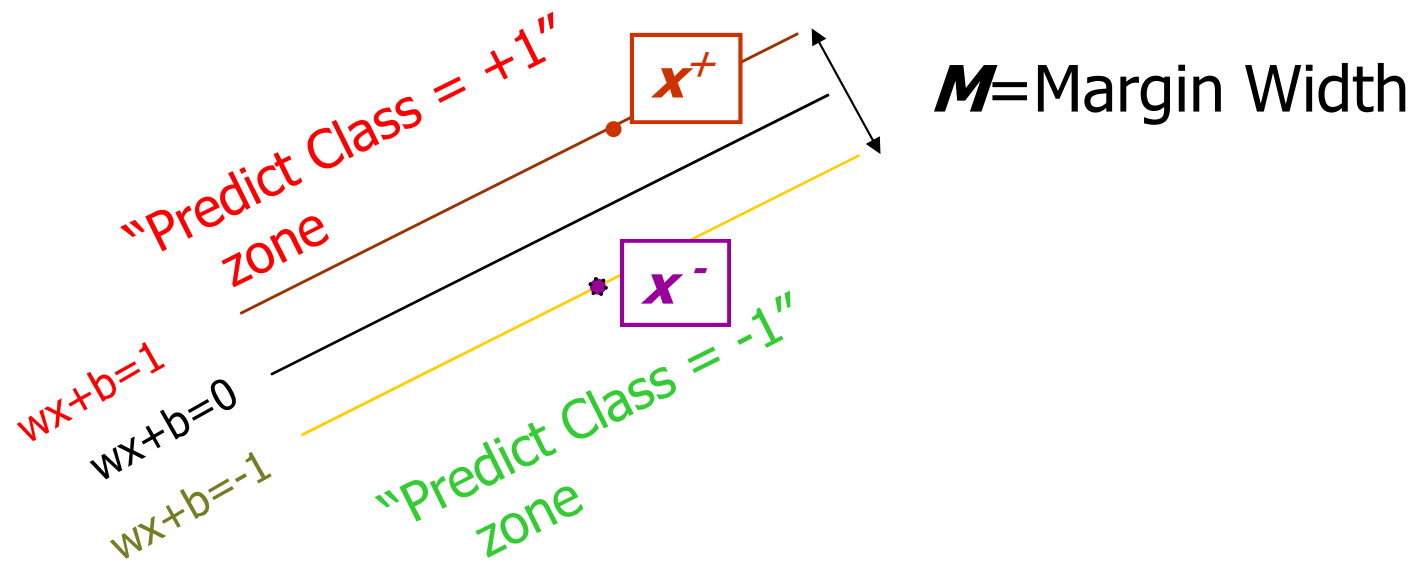
- The hyperplane defining the sides of the margin:

$$H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1 \quad \text{for } y_i = +1, \text{ and}$$

$$H_2: w_0 + w_1 x_1 + w_2 x_2 \leq -1 \quad \text{for } y_i = -1$$

- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints \rightarrow *Quadratic Programming (QP)* \rightarrow Lagrangian multipliers

Linear SVM Mathematically



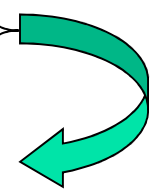
What we know:

- $w \cdot x^+ + b = +1$
- $w \cdot x^- + b = -1$
- $w \cdot (x^+ - x^-) = 2$

$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

Linear SVM Mathematically

- Goal: 1) Correctly classify all training data

$$\begin{array}{ll} wx_i + b \geq 1 & \text{if } y_i = +1 \\ wx_i + b \leq 1 & \text{if } y_i = -1 \\ y_i(wx_i + b) \geq 1 & \text{for all } i \end{array}$$


2) Maximize the Margin $M = \frac{2}{|w|}$
same as minimize $\frac{1}{2} w^t w$

- We can formulate a Quadratic Optimization Problem and solve for w and b

- Minimize $\Phi(w) = \frac{1}{2} w^t w$

subject to $y_i(wx_i + b) \geq 1 \quad \forall i$

Solving the Optimization Problem

Find \mathbf{w} and b such that
 $\Phi(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w}$ is minimized;
and for all $\{(\mathbf{x}_i, y_i)\}$: $y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

- Need to optimize a *quadratic* function subject to *linear* constraints.
- Quadratic optimization problems are a well-known class of mathematical programming problems, and many (rather intricate) algorithms exist for solving them.
- The solution involves constructing a *dual problem* where a *Lagrange multiplier* α_i is associated with every constraint in the primary problem:

Find $\alpha_1 \dots \alpha_N$ such that
 $\mathbf{Q}(\boldsymbol{\alpha}) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and
(1) $\sum \alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

The Optimization Problem Solution

- The solution has the form:

$$\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i \quad b = y_k - \mathbf{w}^T \mathbf{x}_k \text{ for any } \mathbf{x}_k \text{ such that } \alpha_k \neq 0$$

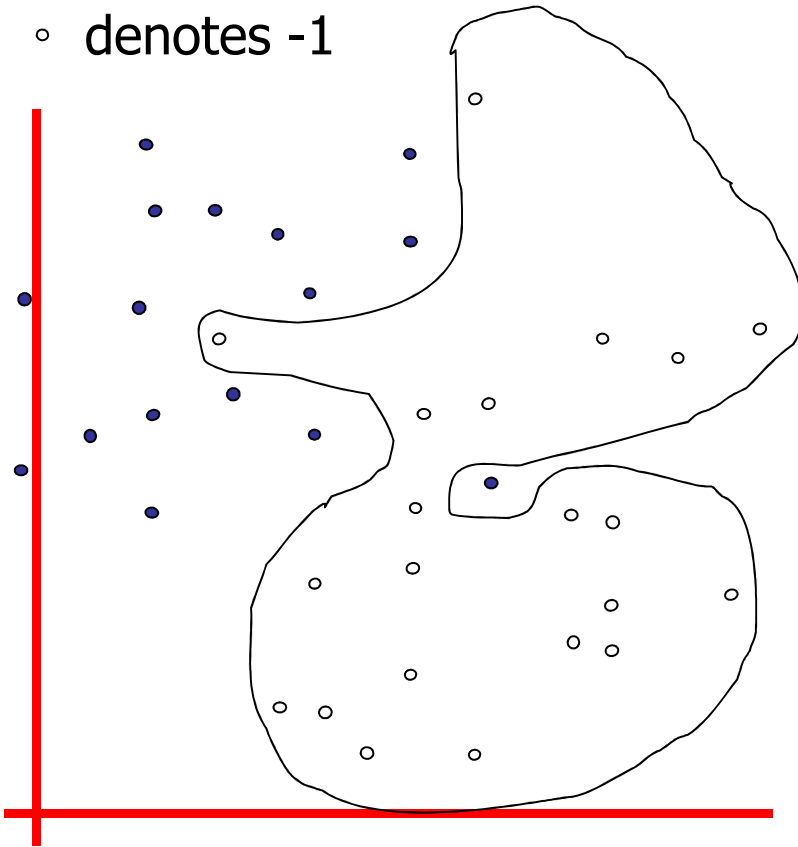
- Each non-zero α_i indicates that corresponding \mathbf{x}_i is a support vector.
- Then the classifying function will have the form:

$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

- Notice that it relies on an *inner product* between the test point \mathbf{x} and the support vectors \mathbf{x}_i – we will return to this later.
- Also keep in mind that solving the optimization problem involved computing the inner products $\mathbf{x}_i^T \mathbf{x}_j$ between all pairs of training points.

Dataset with noise

- denotes +1
- denotes -1

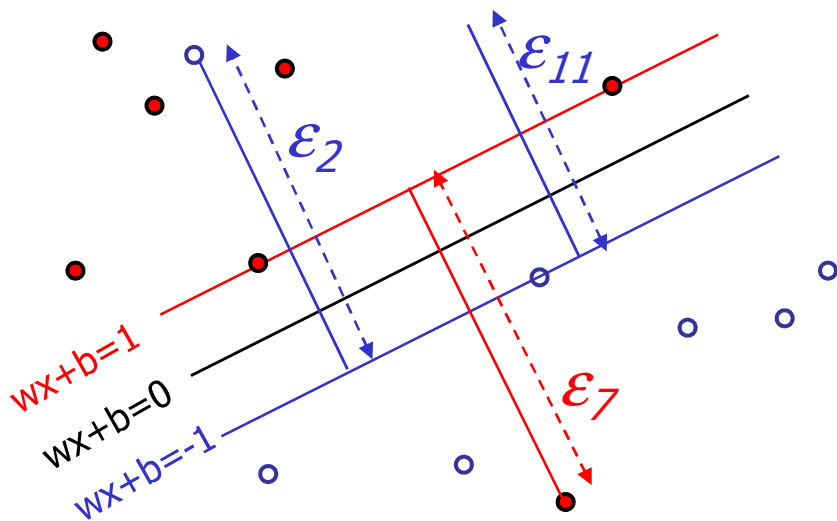


- **Hard Margin:** So far we require all data points be classified correctly
 - No training error
- **What if the training set is noisy?**
 - **Solution 1:** use very powerful kernels

OVERFITTING!

Soft Margin Classification

Slack variables ξ_i can be added to allow misclassification of difficult or noisy examples.



What should our quadratic optimization criterion be?

Minimize

$$\frac{1}{2} \mathbf{w}^T \cdot \mathbf{w} + C \sum_{k=1}^R \xi_k$$

Hard Margin v.s. Soft Margin

- The old formulation:

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = 1/2 \mathbf{w}^T \mathbf{w}$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1$$

- The new formulation incorporating slack variables:

Find \mathbf{w} and b such that

$\Phi(\mathbf{w}) = 1/2 \mathbf{w}^T \mathbf{w} + C \sum \xi_i$ is minimized and for all $\{(\mathbf{x}_i, y_i)\}$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0 \text{ for all } i$$

- Parameter C can be viewed as a way to control overfitting.

Linear SVMs: Overview

- The classifier is a *separating hyperplane*.
- Most “important” training points are support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points \mathbf{x}_i are support vectors with non-zero Lagrangian multipliers α_i .
- Both in the dual formulation of the problem and in the solution training points appear only inside dot products:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $0 \leq \alpha_i \leq C$ for all α_i

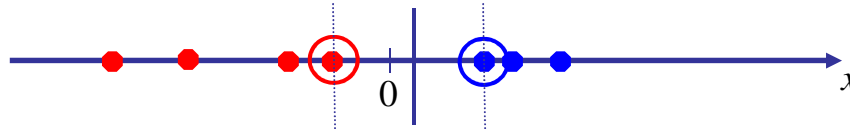
$$f(\mathbf{x}) = \sum \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b$$

Why Is SVM Effective on High Dimensional Data?

- The complexity of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The support vectors are the essential or critical training examples —they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

SVM—Linearly Inseparable

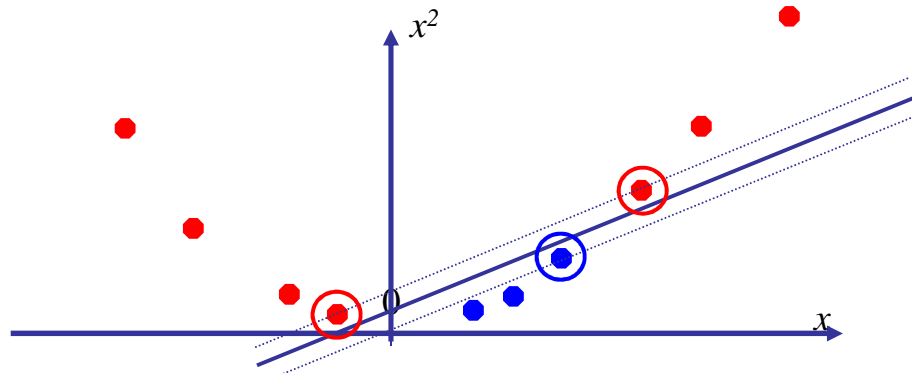
- Datasets that are linearly separable with some noise work out great:



- But what are we going to do if the dataset is just too hard?

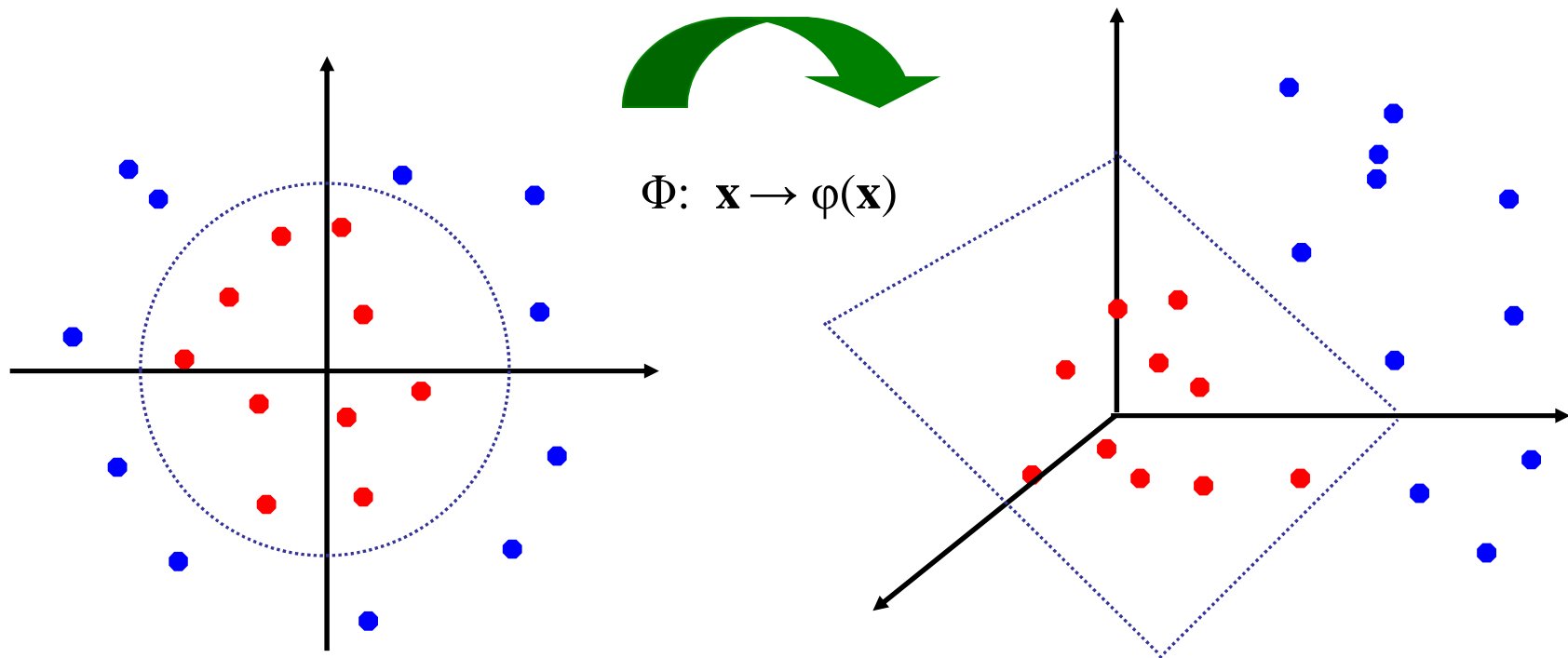


- How about... mapping data to a higher-dimensional space:

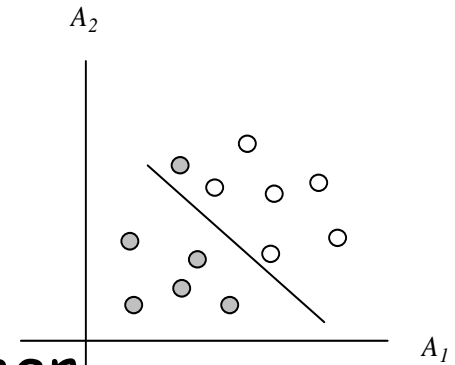


SVM—Linearly Inseparable

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



SVM—Linearly Inseparable



- Transform the original input data into a higher dimensional space

Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(\mathbf{X}) = x_1$, $\phi_2(\mathbf{X}) = x_2$, $\phi_3(\mathbf{X}) = x_3$, $\phi_4(\mathbf{X}) = (x_1)^2$, $\phi_5(\mathbf{X}) = x_1x_2$, and $\phi_6(\mathbf{X}) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where \mathbf{W} and \mathbf{Z} are vectors. This is linear. We solve for \mathbf{W} and b and then substitute back so that we see that the linear decision hyperplane in the new (\mathbf{Z}) space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(\mathbf{Z}) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \quad \blacksquare \end{aligned}$$

- Search for a linear separating hyperplane in the new space

The “Kernel Trick”

- The linear classifier relies on dot product between vectors $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- If every data point is mapped into high-dimensional space via some transformation $\Phi: \mathbf{x} \rightarrow \phi(\mathbf{x})$, the dot product becomes:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

- A *kernel function* is some function that corresponds to an inner product in some expanded feature space.
- Example:

2-dimensional vectors $\mathbf{x} = [x_1 \ x_2]$; let $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$,

Need to show that $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$:

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \quad \text{where } \phi(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$

What Functions are Kernels?

- For some functions $K(\mathbf{x}_i, \mathbf{x}_j)$ checking that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j) \text{ can be cumbersome.}$$

- Mercer's theorem:

Every semi-positive definite symmetric function is a kernel

- Semi-positive definite symmetric functions correspond to a semi-positive definite symmetric Gram matrix:

$K =$

$K(\mathbf{x}_1, \mathbf{x}_1)$	$K(\mathbf{x}_1, \mathbf{x}_2)$	$K(\mathbf{x}_1, \mathbf{x}_3)$...	$K(\mathbf{x}_1, \mathbf{x}_N)$
$K(\mathbf{x}_2, \mathbf{x}_1)$	$K(\mathbf{x}_2, \mathbf{x}_2)$	$K(\mathbf{x}_2, \mathbf{x}_3)$		$K(\mathbf{x}_2, \mathbf{x}_N)$
...
$K(\mathbf{x}_N, \mathbf{x}_1)$	$K(\mathbf{x}_N, \mathbf{x}_2)$	$K(\mathbf{x}_N, \mathbf{x}_3)$...	$K(\mathbf{x}_N, \mathbf{x}_N)$

Examples of Kernel Functions

- Linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- Polynomial of power p : $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$
- Gaussian (radial-basis function network):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

- Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\beta_0 \mathbf{x}_i^T \mathbf{x}_j + \beta_1)$

Non-linear SVMs Mathematically

- Dual problem formulation:

Find $\alpha_1 \dots \alpha_N$ such that

$Q(\alpha) = \sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ is maximized and

(1) $\sum \alpha_i y_i = 0$

(2) $\alpha_i \geq 0$ for all α_i

- The solution is:

$$f(\mathbf{x}) = \sum \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}_j) + b$$

- Optimization techniques for finding α_i 's remain the same!

Nonlinear SVM - Overview

- SVM locates a separating hyperplane in the feature space and classifies points in that space
- It does not need to represent the space explicitly, simply by defining a kernel function
- The kernel function plays the role of the dot product in the feature space.

Weakness of SVM

- It is sensitive to noise

- A relatively small number of mislabeled examples can dramatically decrease the performance

- It only considers two classes

- how to do multi-class classification with SVM?

- Answer:

- 1) with output arity m , learn m SVM's

- SVM 1 learns "Output==1" vs "Output != 1"

- SVM 2 learns "Output==2" vs "Output != 2"

- :

- SVM m learns "Output== m " vs "Output != m "

- (This strategy for prediction of multi-class problems using binary classifiers is known as One-against-all)

- 2) To predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.

Some Issues

- **Choice of kernel**

- Gaussian or polynomial kernel is default
- if ineffective, more elaborated kernels are needed
- domain experts can give assistance in formulating appropriate similarity measures

- **Choice of kernel parameters**

- e.g. σ in Gaussian kernel
- σ is the distance between closest points with different classifications
- In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

- **Optimization criterion** – Hard margin v.s. Soft margin

- a lengthy series of experiments in which various parameters are tested

SVM vs. Neural Network

■ SVM

- Relatively new concept
- Deterministic algorithm
- Nice Generalization properties
- Hard to learn – learned in batch mode using quadratic programming techniques
- Using kernels can learn very complex functions

■ Neural Network

- Relatively old
- Nondeterministic algorithm
- Generalizes well but doesn't have strong mathematical foundation
- Can easily be learned in incremental fashion
- To learn complex functions—use multilayer perceptron (not that trivial)

SVM—Introduction Literature

- *'Data Mining: Practical Machine Learning Tools and Techniques second edition', Ian H. Witten and Eibe Frank, 2005*
- “Statistical Learning Theory” by Vapnik: the big reference but extremely hard to understand, containing many errors too.
- C. J. C. Burges. [A Tutorial on Support Vector Machines for Pattern Recognition](#). *Knowledge Discovery and Data Mining*, 2(2), 1998.
 - Better than the Vapnik’s book, but still written too hard for introduction, and the examples are so not-intuitive
- The book “An Introduction to Support Vector Machines” by N. Cristianini and J. Shawe-Taylor
 - Also written hard for introduction, but the explanation about the mercer’s theorem is better than above literatures
- The neural network book by Haykins
 - Contains one nice chapter of SVM introduction

SVM Related Links

- SVM Website
 - <http://www.kernel-machines.org/>
 - <http://www.svms.org/> (see the tutorials)
- Representative implementations
 - LIBSVM: an efficient implementation of SVM, multi-class classifications, nu-SVM, one-class SVM, including also various interfaces with java, python, etc.
 - SVM-light: simpler but performance is not better than LIBSVM, support only binary classification and only C language
 - SVM-torch: another recent implementation also written in C.



Thank you !!!