CS249: ADVANCED DATA MINING

Support Vector Machine and Neural Network

Instructor: Yizhou Sun

yzsun@cs.ucla.edu

April 24, 2017

Announcements

- Homework 1
  - Due end of the day of this Friday (11:59pm)

- Reminder of late submission policy
  - original score * $\mathbf{1}(t <= 24)e^{-(ln(2)/12)*t}$
  - E.g., if you are t = 12 hours late, maximum of half score will be obtained; if you are 24 hours late, 0 score will be given.

Methods to Learn: Last Lecture

	Vector Data	Text Data	Recommender System	Graph & Network
Classification	**Decision Tree**; **Naïve Bayes**; **Logistic Regression** SVM; NN			Label Propagation
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models; kernel k-means	PLSA; LDA	Matrix Factorization	SCAN; Spectral Clustering
Prediction	**Linear Regression** **GLM**		Collaborative Filtering	
Ranking				PageRank
Feature Representation		Word embedding		Network embedding

Methods to Learn

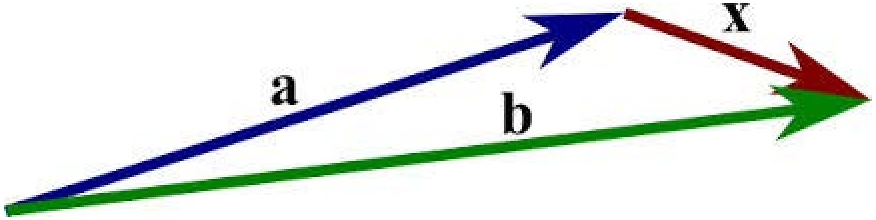
	Vector Data	Text Data	Recommender System	Graph & Network
Classification	**Decision Tree**; **Naïve Bayes**; **Logistic Regression** **SVM; NN**			Label Propagation
Clustering	K-means; hierarchical clustering; DBSCAN; Mixture Models; kernel k-means	PLSA; LDA	Matrix Factorization	SCAN; Spectral Clustering
Prediction	**Linear Regression** **GLM**		Collaborative Filtering	
Ranking				PageRank
Feature Representation		Word embedding		Network embedding

Support Vector Machine and Neural Network

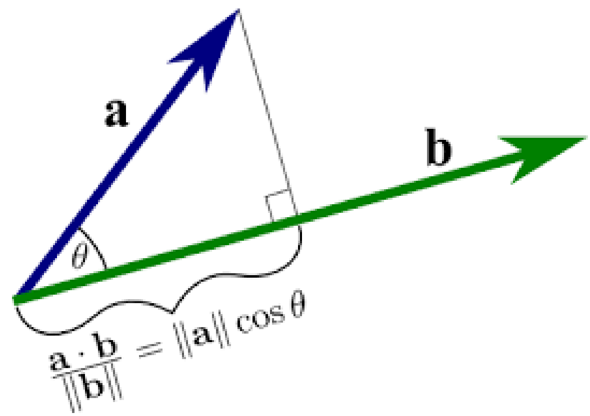
- Support Vector Machine
- Neural Network
- Summary

Math Review

- Vector
  - $\boldsymbol{x} = (\mathrm{x}_1, \mathrm{x}_2, \dots, x_n)$
  - Subtracting two vectors: $\boldsymbol{x} = \boldsymbol{b} - \boldsymbol{a}$



- Dot product
  - $\boldsymbol{a} \cdot \boldsymbol{b} = \sum a_i b_i$
  - Geometric interpretation: projection
  - If $\boldsymbol{a}$ *and* $\boldsymbol{b}$ are orthogonal, $\boldsymbol{a} \cdot \boldsymbol{b} = 0$

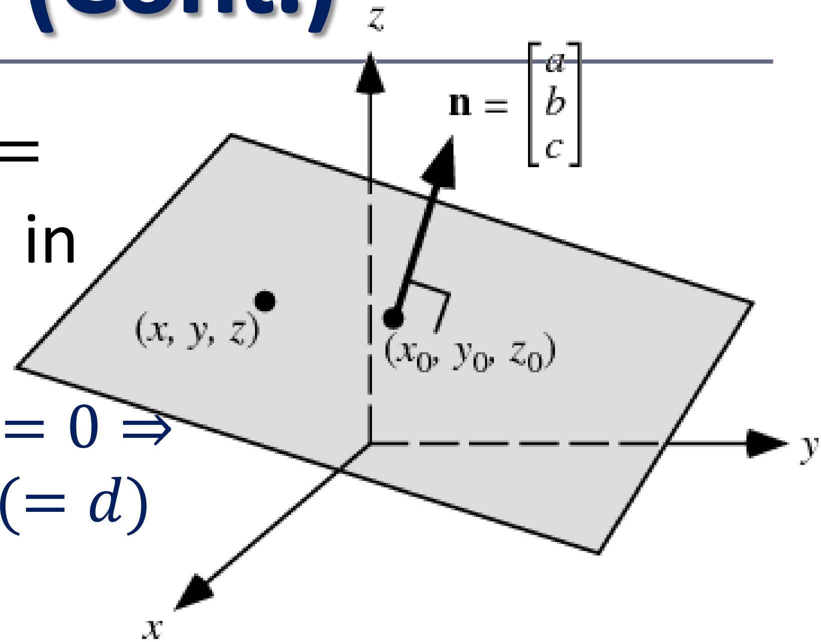


Math Review (Cont.)

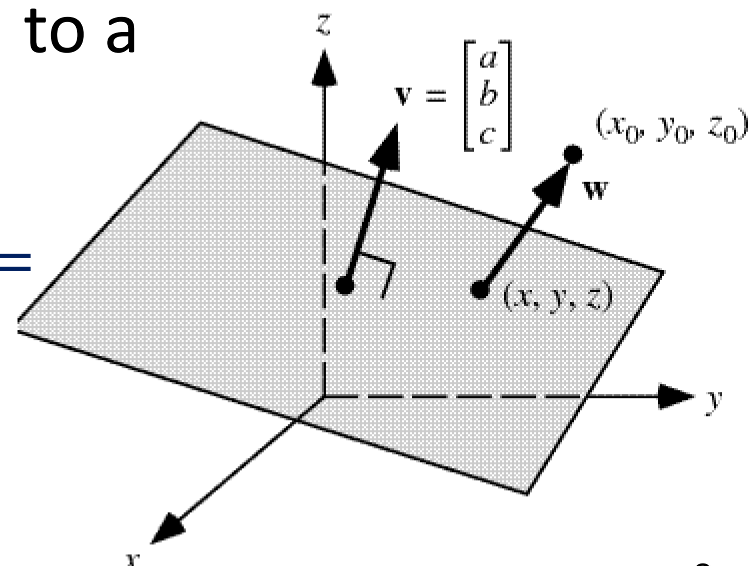
- Plane/Hyperplane
  - $a_1x_1 + a_2x_2 + \cdots + a_nx_n = c$
  - Line (n=2), plane (n=3), hyperplane (higher dimensions)
- Normal of a plane
  - $\boldsymbol{n} = (a_1, a_2, \dots, a_n)$
  - a vector which is perpendicular to the surface

Math Review (Cont.)

- Define a plane using normal $\boldsymbol{n} = (a, b, c)$ and a point (x_0, y_0, z_0) in the plane:
  - $(a, b, c) \cdot (x_0 - x, y_0 - y, z_0 - z) = 0 \Rightarrow ax + by + cz = ax_0 + by_0 + cz_0 (= d)$



- Distance from a point (x_0, y_0, z_0) to a plane $ax + by + cz = \mathrm{d}$
  - $\left|(x_0 - x, y_0 - y, z_0 - z) \cdot \frac{(a,b,c)}{||(a,b,c)||}\right| = \frac{|ax_0+by_0+cz_0-d|}{\sqrt{a^2+b^2+c^2}}$



Linear Classifier

- Given a training dataset $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$
  - A separating hyperplane can be written as a linear combination of attributes

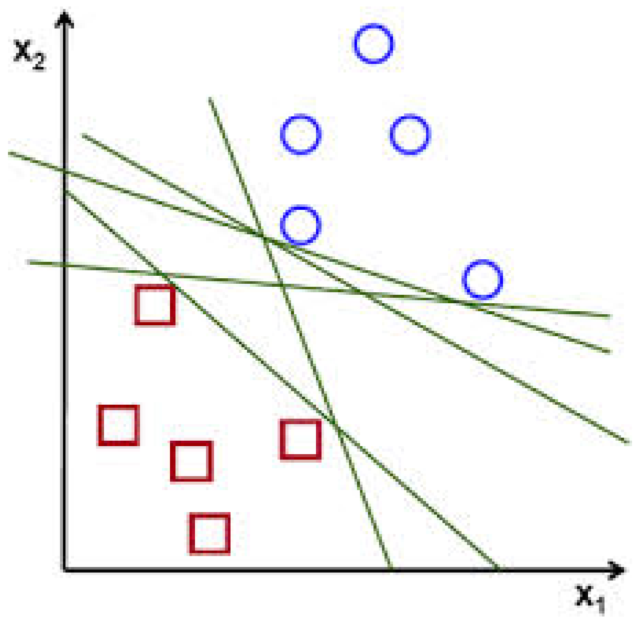
    $\mathbf{W} \bullet \mathbf{X} + b = 0$

    where $\mathbf{W}=\{w_1, w_2, ..., w_n\}$ is a weight vector and b a scalar (bias)
  - For 2-D it can be written as

    $w_0 + w_1 x_1 + w_2 x_2 = 0$
  - Classification:

    $w_0 + w_1 x_1 + w_2 x_2 > 0 \Rightarrow y_i = +1$

    $w_0 + w_1 x_1 + w_2 x_2 \leq 0 \Rightarrow y_i = -1$



Simple Linear Classifier: Perceptron

$\mathbf{x} = (1, x_1, x_2, \ldots, x_d)^T$ $\quad$ $\mathbf{w} = (\omega_0, \omega_1, \omega_2, \ldots, \omega_d)^T$

$y = \{1, -1\}$ $\quad$ $\alpha \in (0, 1]$ (learning rate)

Initialize $\mathbf{w} = \mathbf{0}$ (can be any vector)

Repeat:

- For each training example $(\mathbf{x}_i, y_i)$:
  - Compute $\quad \hat{y}_i = \text{sign}(\mathbf{w}^\mathbf{T}\mathbf{x_i})$
  - if $(y_i \neq \hat{y}_i)$ $\quad \mathbf{w} = \mathbf{w} + \alpha(y_i\mathbf{x_i})$

Until $(y_i = \hat{y}_i \quad \forall i = 1 \ldots N)$

Return $\mathbf{w}$

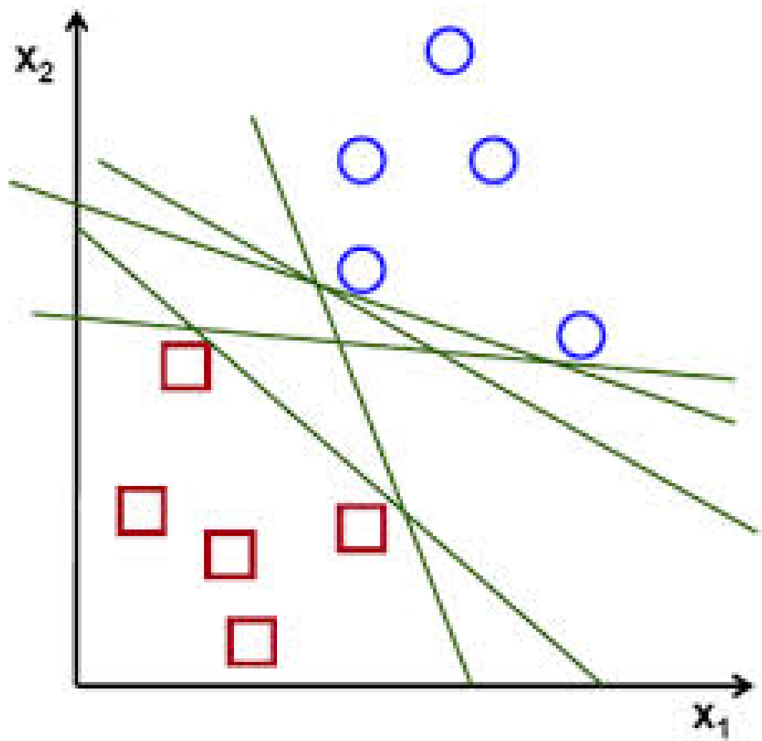
Loss function: $\max\{0, -y_i * w^T x_i\}$

Example

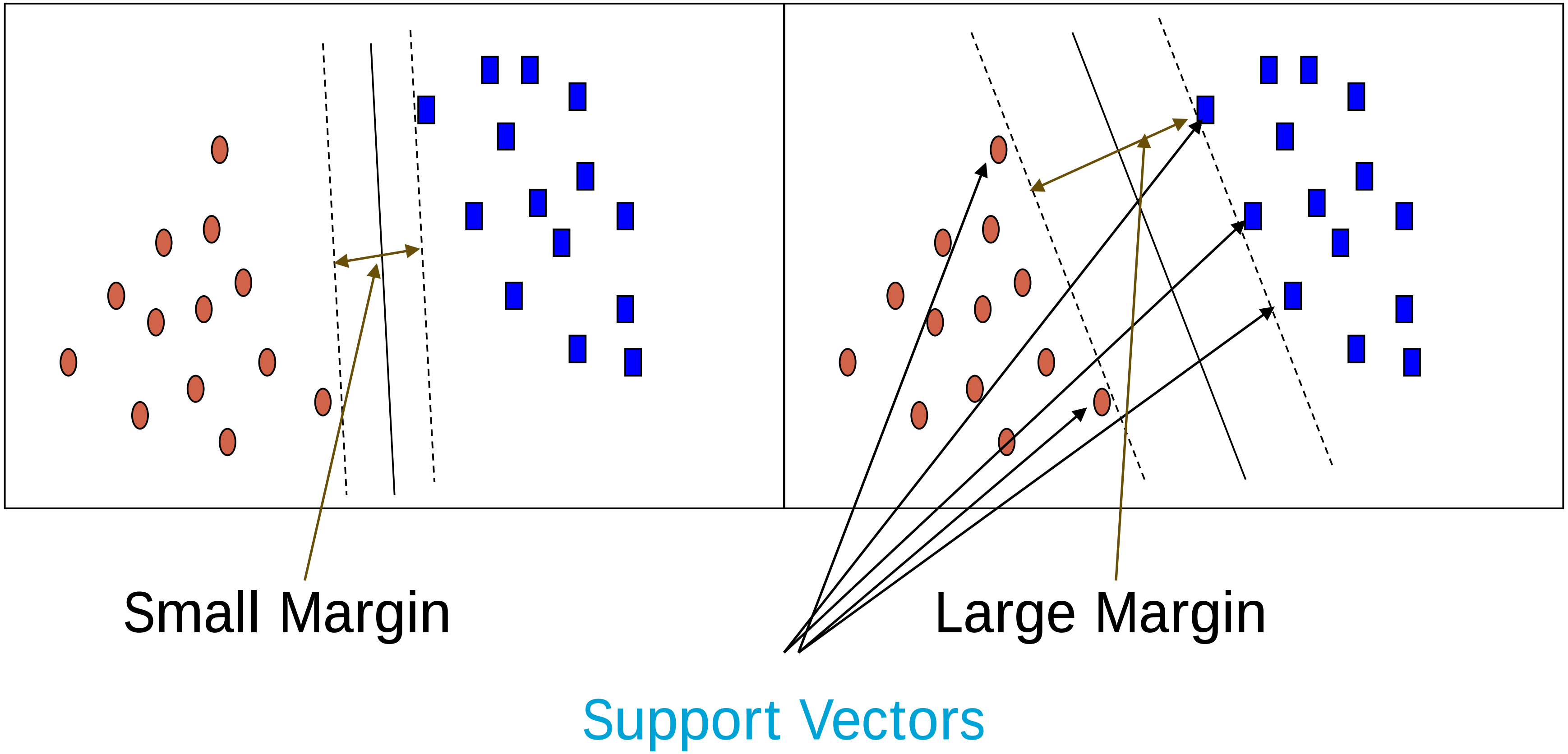
x0	x1	x2	true label	w before update	predicted label	w after update
1	0	1	Y	(0.0, 0.0, 0.0)	N	(0.9, 0.0, 0.9)
1	1	1	N	(0.9, 0.0, 0.9)	Y	(0.0, -0.9, 0.0)
1	0	0	Y	(0.0, -0.9, 0.0)	N	(0.9, -0.9, 0.0)
1	1	0	Y	(0.9, -0.9, 0.0)	N	(1.8, 0.0, 0.0)
1	0	1	Y	(1.8, 0.0, 0.0)	Y	(1.8, 0.0, 0.0)
1	1	1	N	(1.8, 0.0, 0.0)	Y	(0.9, -0.9, -0.9)
1	0	0	Y	(0.9, -0.9, -0.9)	Y	(0.9, -0.9, -0.9)
1	1	0	Y	(0.9, -0.9, -0.9)	N	(1.8, 0.0, -0.9)
1	0	1	Y	(1.8, 0.0, -0.9)	Y	(1.8, 0.0, -0.9)
1	1	1	N	(1.8, 0.0, -0.9)	Y	(0.9, -0.9, -1.8)
1	0	0	Y	(0.9, -0.9, -1.8)	Y	(0.9, -0.9, -1.8)
1	1	0	Y	(0.9, -0.9, -1.8)	N	(1.8, 0.0, -1.8)

Can we do better?

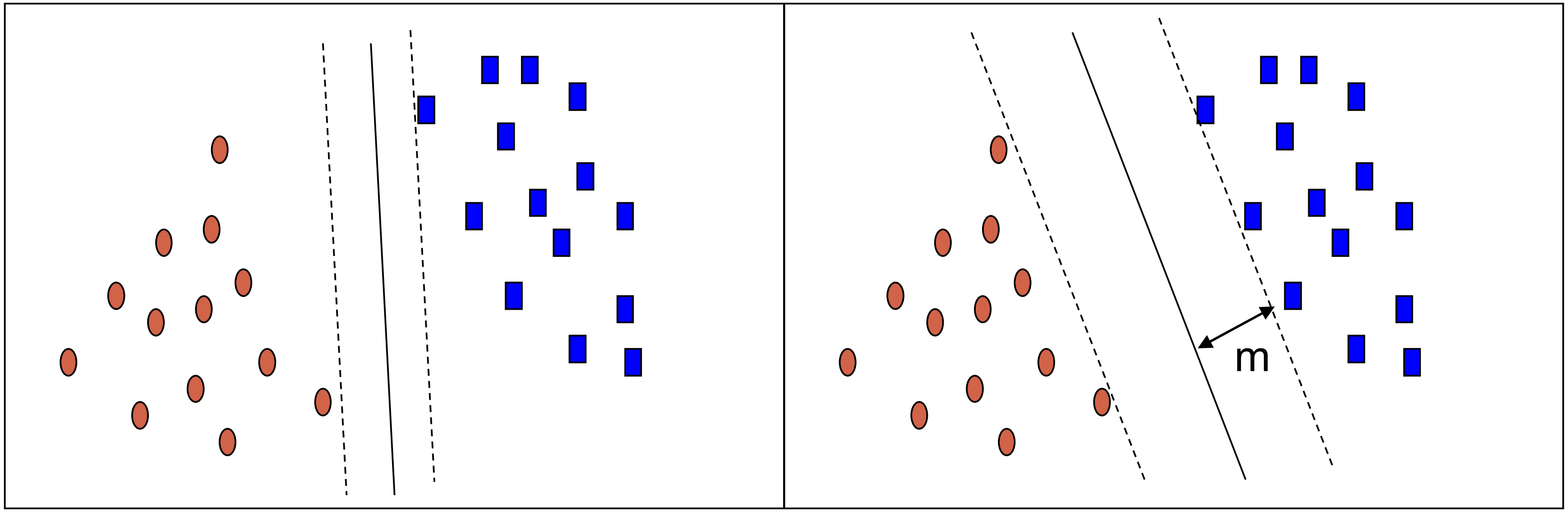
- Which hyperplane to choose?



SVM—Margins and Support Vectors



SVM—When Data Is Linearly Separable



Let data D be ($\mathbf{X}_1$, y_1), ..., ($\mathbf{X}_{|D|}$, $y_{|D|}$), 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

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

- The hyperplane defining the sides of the margin, e.g.,:

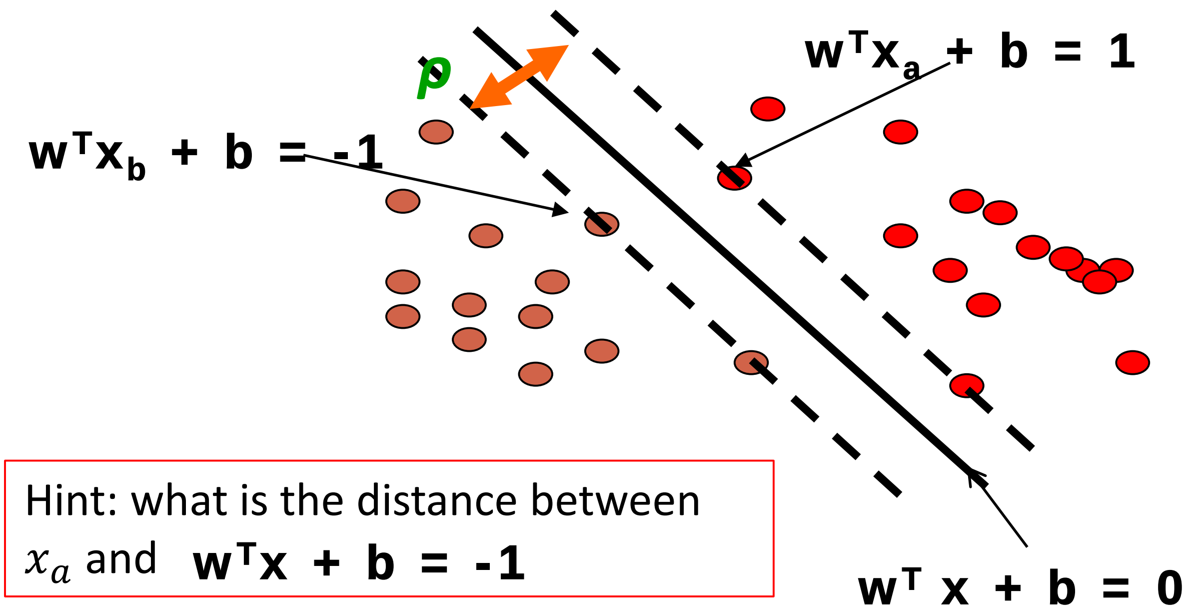
  H_1: $w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

  H_2: $w_0 + w_1 x_1 + w_2 x_2 \leq -1$ 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 → *Quadratic Programming (QP)* → Lagrangian multipliers

Maximum Margin Calculation

- **w**: decision hyperplane normal vector
- $\mathbf{x}_i$: data point *i*
- y_i: class of data point *i* (+1 or -1)



$$margin: \rho = \frac{2}{||\boldsymbol{w}||}$$

Hint: what is the distance between x_a and $\mathbf{w}^T\mathbf{x} + \mathbf{b} = \mathbf{-1}$

SVM as a Quadratic Programming

- QP

Objective: Find $\mathbf{w}$ and b such that $\rho = \frac{2}{||\boldsymbol{w}||}$ is maximized;

Constraints: For all $\{(\mathbf{x_i}, y_i)\}$

$\mathbf{w^T x_i} + b \geq 1$ if $y_i=1$;

$\mathbf{w^T x_i} + b \leq -1$ if $y_i = -1$

- A better form

Objective: Find $\mathbf{w}$ and b such that $\boldsymbol{\Phi}(\mathbf{w}) = \frac{1}{2}\, \mathbf{w^T w}$ is minimized;

Constraints: for all $\{(\mathbf{x_i}, y_i)\}$: $y_i (\mathbf{w^T x_i} + b) \geq 1$

Solve QP

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

Lagrange Formulation

Minimize

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{i=1}^{N} \alpha_i (y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - 1)$$

Take the partial derivatives w.r.t $\mathbf{w}$, b:

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0$$

Primal Form and Dual Form

Primal

Objective: Find $\mathbf{w}$ and b such that $\boldsymbol{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^T\mathbf{w}$ is minimized;

Constraints: for all $\{(\mathbf{x_i}, y_i)\}$: $y_i (\mathbf{w}^T\mathbf{x_i} + b) \geq 1$

Equivalent under some conditions: KKT conditions

Dual

Objective: Find $\alpha_1 \ldots \alpha_n$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and

Constraints
(1) $\Sigma\alpha_i y_i = 0$
(2) $\alpha_i \geq 0$ for all α_i

- More derivations: http://cs229.stanford.edu/notes/cs229-notes3.pdf

The Optimization Problem Solution

- The solution has the form:

$$\mathbf{w} = \Sigma \alpha_i y_i \mathbf{x_i} \qquad b = y_k - \mathbf{w}^T\mathbf{x_k} \text{ for any } \mathbf{x_k} \text{ such that } \alpha_k \neq 0$$

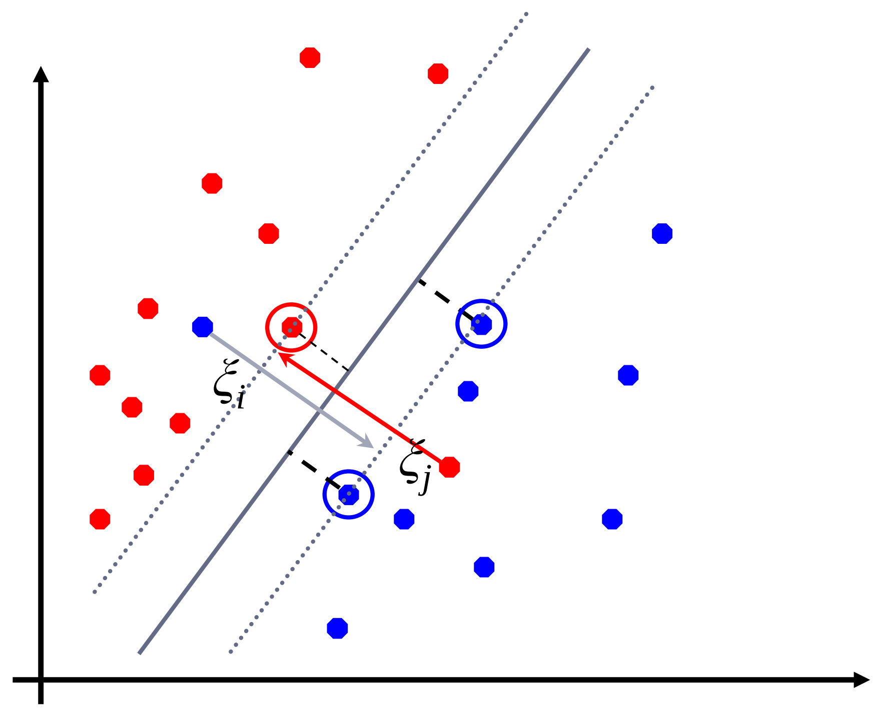
- Each non-zero α_i indicates that corresponding $\mathbf{x_i}$ is a support vector.
- Then the classifying function will have the form:

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

- Notice that it relies on an *inner product* between the test point **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.

Soft Margin Classification

- If the training data is not linearly separable, *slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples.
- Allow some errors
  - Let some points be moved to where they belong, at a cost
- Still, try to minimize training set errors, and to place hyperplane “far” from each class (large margin)



Soft Margin Classification Mathematically

- The old formulation:

> Find $\mathbf{w}$ and b such that
> $\mathbf{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^{T}\mathbf{w}$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
> $y_i\, (\mathbf{w}^{\mathbf{T}}\mathbf{x_i} + \mathrm{b}) \geq 1$

- The new formulation incorporating slack variables:

> Find $\mathbf{w}$ and b such that
> $\mathbf{\Phi}(\mathbf{w}) = ½\, \mathbf{w}^{T}\mathbf{w} + C\Sigma\xi_i$ is minimized and for all $\{(\mathbf{x_i}, y_i)\}$
> $y_i\, (\mathbf{w}^{\mathbf{T}}\mathbf{x_i} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all i

- Parameter *C* can be viewed as a way to control overfitting
  - A regularization term (L1 regularization)

Soft Margin Classification – Solution

- The dual problem for soft margin classification:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

- Neither slack variables ξ_i nor their Lagrange multipliers appear in the dual problem!
- Again, $\mathbf{x_i}$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x_i}$
$b = y_k(1 - \xi_k) - \mathbf{w}^T\mathbf{x}_k$ where $k = \underset{k'}{\text{argmax}}\ \alpha_{k'}$

w is not needed explicitly for classification!

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

Classification with SVMs

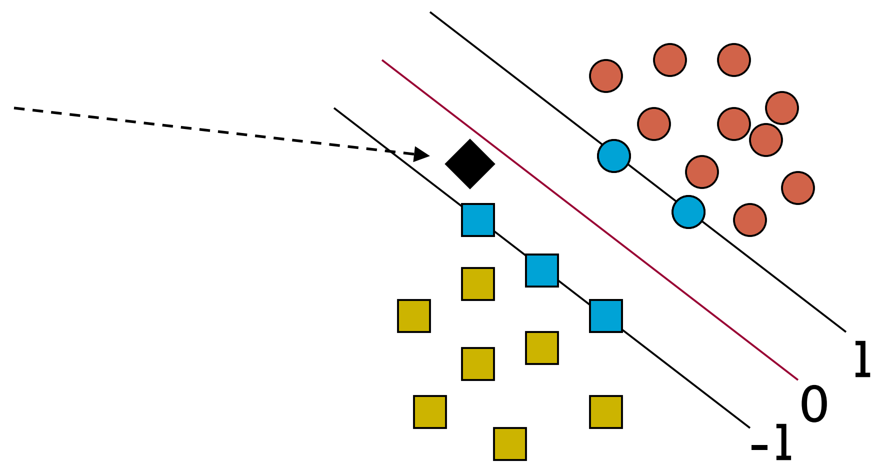
- Given a new point **x**, we can score its projection onto the hyperplane normal:
  - I.e., compute score: $\mathbf{w}^T\mathbf{x} + b = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$
    - Decide class based on whether < or > 0

  - Can set confidence threshold *t*.

Score > *t*: yes

Score < -*t*: no

Else: don't know



Linear SVMs: Summary

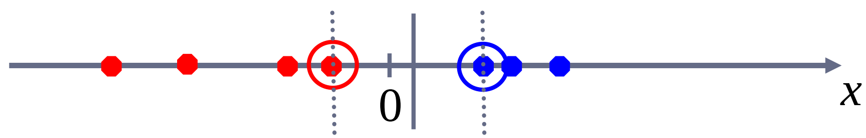
- The classifier is a *separating hyperplane.*
- The most "important" training points are the support vectors; they define the hyperplane.
- Quadratic optimization algorithms can identify which training points $\mathbf{x_i}$ are support vectors with non-zero Lagrangian multipliers α_i.
- Both in the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

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x_i}^T\mathbf{x_j}$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

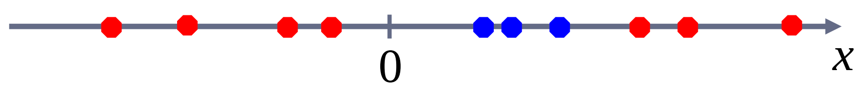
$$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x_i}^T\mathbf{x} + b$$

Non-linear SVMs

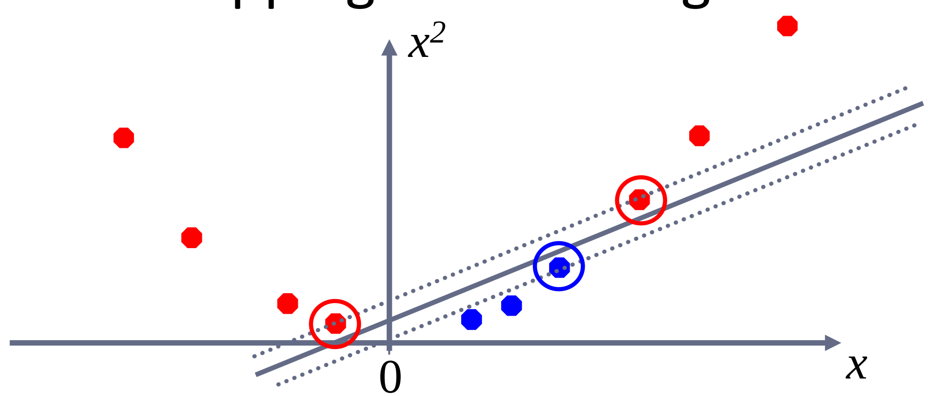
- Datasets that are linearly separable (with some noise) work out great:



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

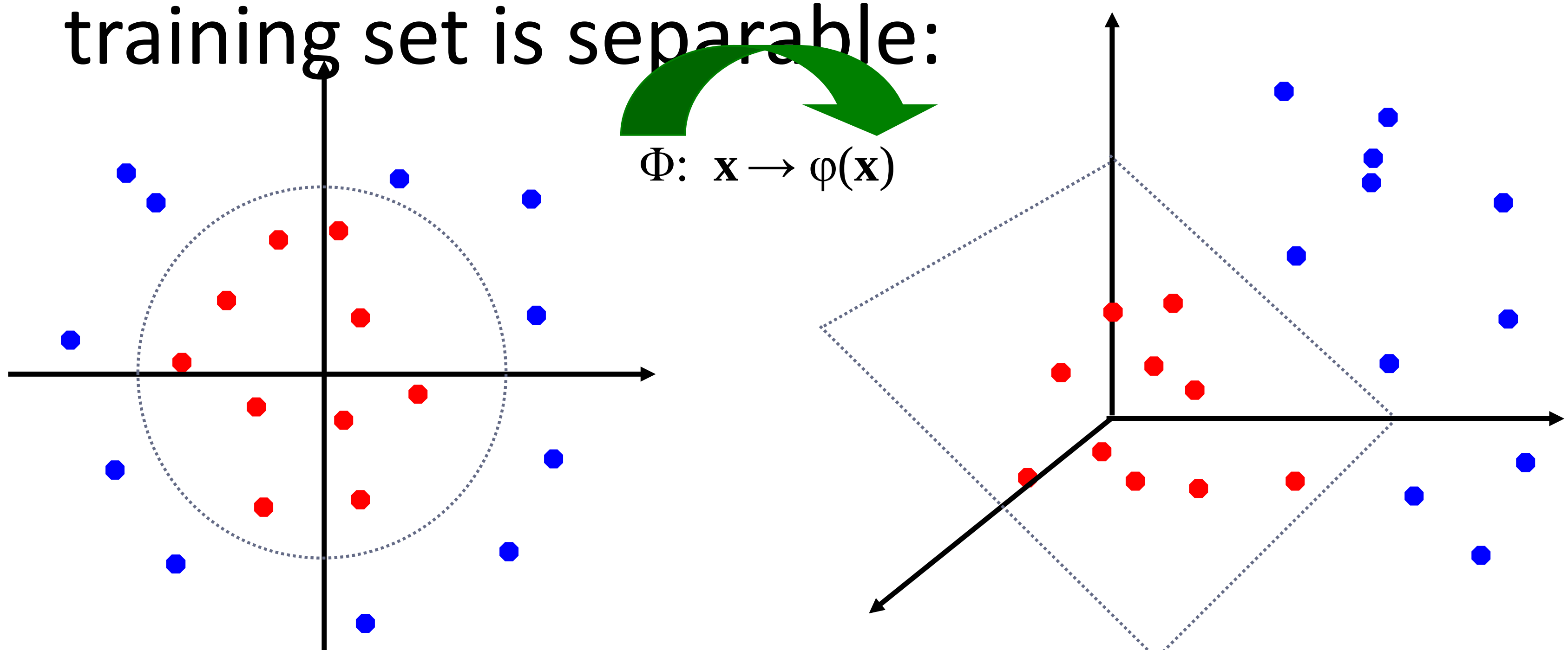


- How about … mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

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



The "Kernel Trick"

- The linear classifier relies on an inner 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 Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner 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 $K(\mathbf{x}_i,\mathbf{x}_j)= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$:

$K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2{}_{,}= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}=$

$= [1\ \ x_{i1}^2\ \ \sqrt{2}\, x_{i1}x_{i2}\ \ x_{i2}^2\ \ \sqrt{2}x_{i1}\ \ \sqrt{2}x_{i2}]^T\, [1\ \ x_{j1}^2\ \ \sqrt{2}\, x_{j1}x_{j2}\ \ x_{j2}^2\ \ \sqrt{2}x_{j1}\ \ \sqrt{2}x_{j2}]$

$= \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ where $\phi(\mathbf{x}) = [1\ \ x_1^2\ \ \sqrt{2}\, x_1x_2\ \ x_2^2\ \ \sqrt{2}x_1\ \ \sqrt{2}x_2]$

SVM: Different Kernel functions

- Instead of computing the dot product on the transformed data, it is math. equivalent to applying a kernel function $K(\mathbf{X_i}, \mathbf{X_j})$ to the original data, i.e., $K(\mathbf{X_i}, \mathbf{X_j}) = \Phi(\mathbf{X_i})^T\Phi(\mathbf{X_j})$
- Typical Kernel Functions

$$\textbf{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

$$\textbf{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

$$\textbf{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

- *SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)

Non-linear SVM

- Replace inner-product with kernel functions
  - Optimization problem

Find $\alpha_1 \ldots \alpha_N$ such that

$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - ½\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{K}(\mathbf{x_i},\mathbf{x_j})$ is maximized and

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

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

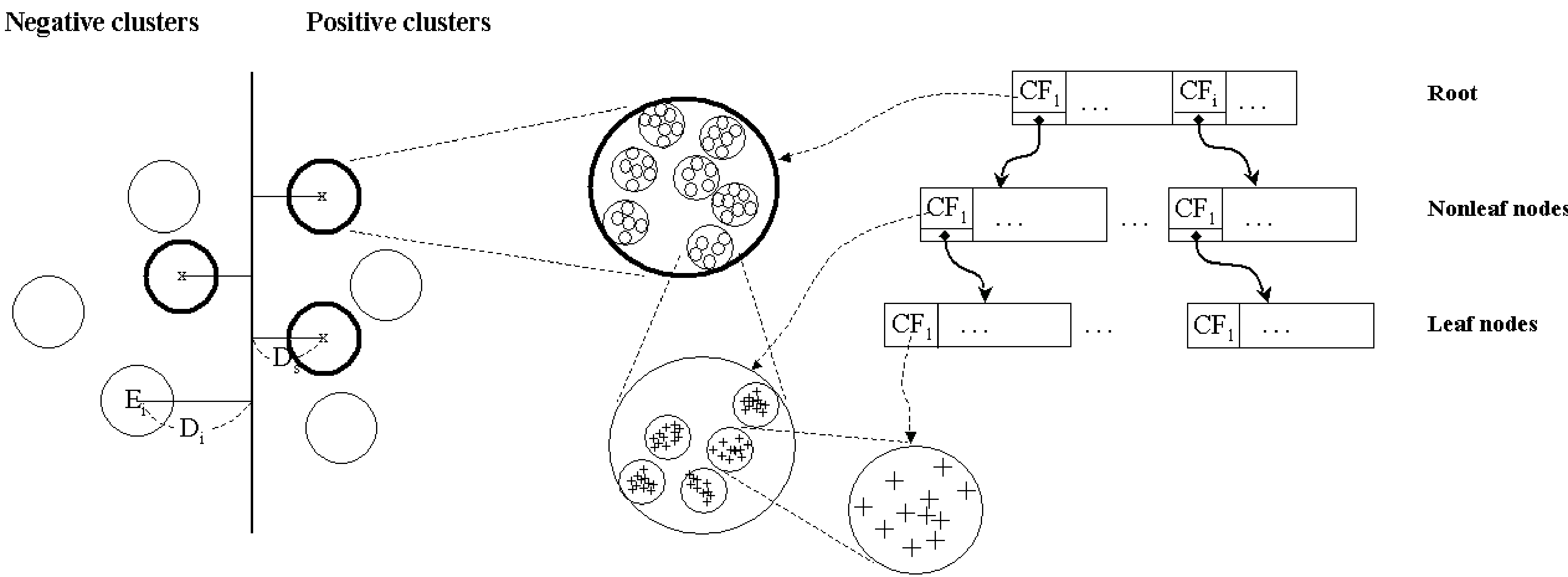
  - Decision boundary

$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{K}(\mathbf{x_i},\mathbf{x_j}) + b$

*Scaling SVM by Hierarchical Micro-Clustering

- SVM is not scalable to the number of data objects in terms of training time and memory usage
- H. Yu, J. Yang, and J. Han, "Classifying Large Data Sets Using SVM with Hierarchical Clusters", KDD'03)
- CB-SVM (Clustering-Based SVM)
  - Given limited amount of system resources (e.g., memory), maximize the SVM performance in terms of accuracy and the training speed
  - Use micro-clustering to effectively reduce the number of points to be considered
  - At deriving support vectors, de-cluster micro-clusters near "candidate vector" to ensure high classification accuracy

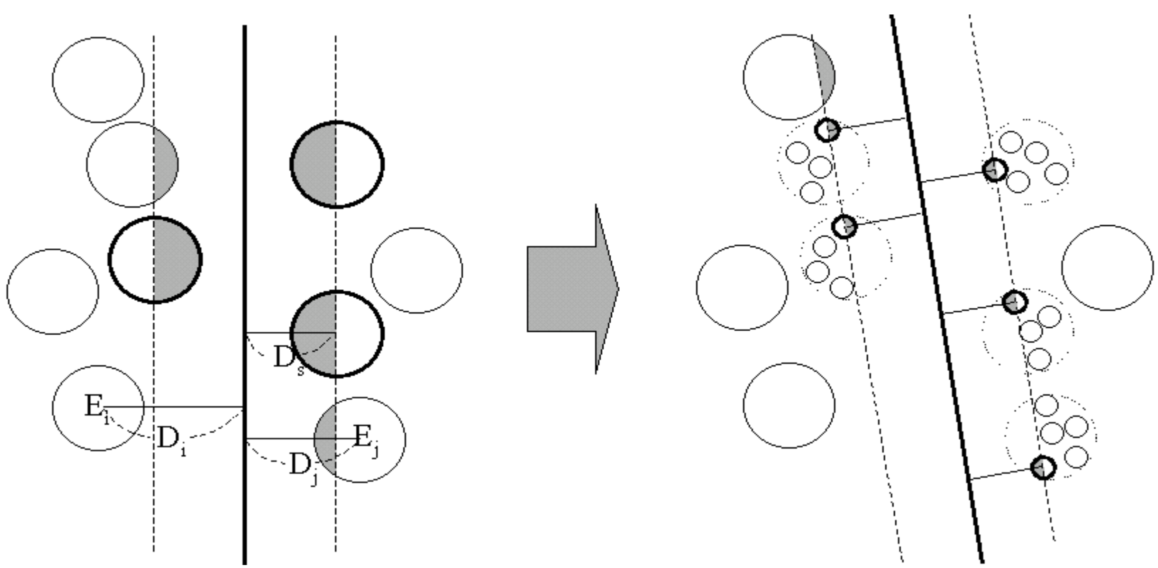
*CF-Tree: Hierarchical Micro-cluster



- Read the data set once, construct a statistical summary of the data (i.e., hierarchical clusters) given a limited amount of memory
- Micro-clustering: Hierarchical indexing structure
  - provide finer samples closer to the boundary and coarser samples farther from the boundary

*Selective Declustering: Ensure High Accuracy

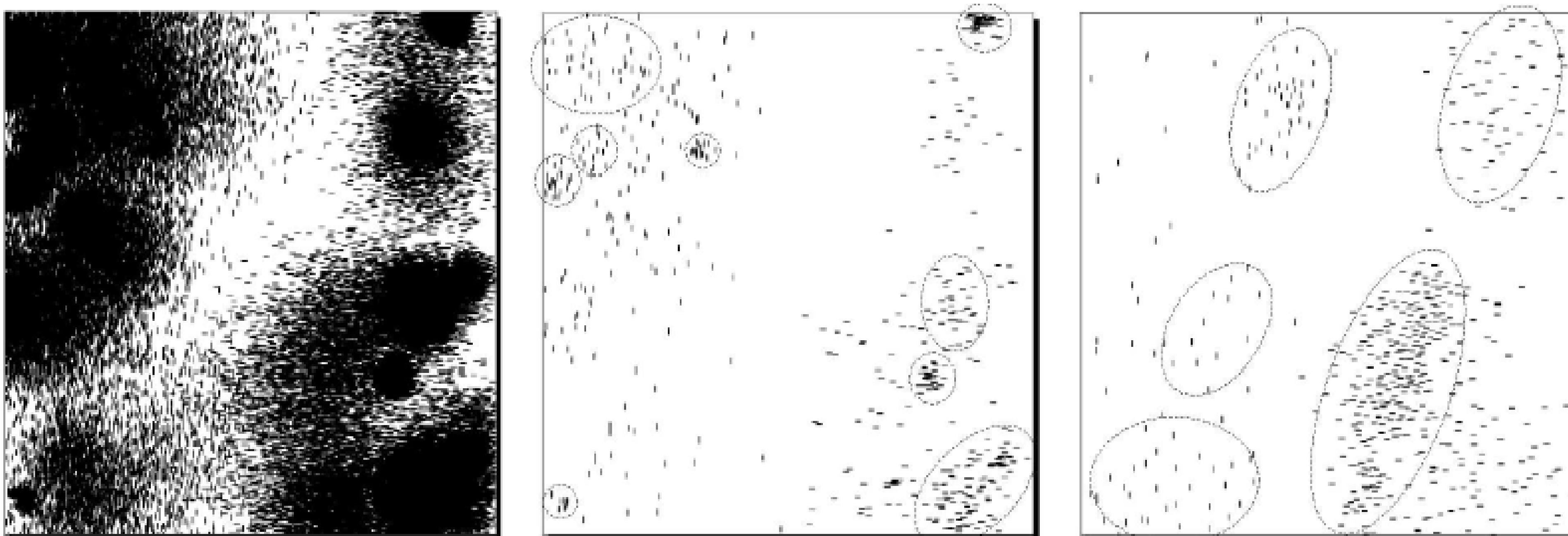
- CF tree is a suitable base structure for selective declustering
- De-cluster only the cluster E_i such that
  - $D_i - R_i < D_s$, where D_i is the distance from the boundary to the center point of E_i and R_i is the radius of E_i
  - Decluster only the cluster whose subclusters have possibilities to be the support cluster of the boundary
    - "Support cluster": The cluster whose centroid is a support vector



*CB-SVM Algorithm: Outline

- Construct two CF-trees from positive and negative data sets independently
  - Need one scan of the data set
- Train an SVM from the centroids of the root entries
- De-cluster the entries near the boundary into the next level
  - The children entries de-clustered from the parent entries are accumulated into the training set with the non-declustered parent entries
- Train an SVM again from the centroids of the entries in the training set
- Repeat until nothing is accumulated

*Accuracy and Scalability on Synthetic Dataset



(a) original data set ($N = 113601$)

(b) 0.5% randomly sampled data ($N = 603$)

(c) data distribution at the last iteration in CB-SVM ($N = 597$)

Figure 6: Synthetic data set in a two-dimensional space. '|': positive data; '−': negative data

- Experiments on large synthetic data sets shows better accuracy than random sampling approaches and far more scalable than the original SVM algorithm

SVM Related Links

- SVM Website: http://www.kernel-machines.org/
- 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 in C
  - **SVM-torch**: another recent implementation also written in C
- From classification to regression and ranking:
  - http://www.dainf.ct.utfpr.edu.br/~kaestner/Mineracao/hwanjoyu-svmtutorial.pdf

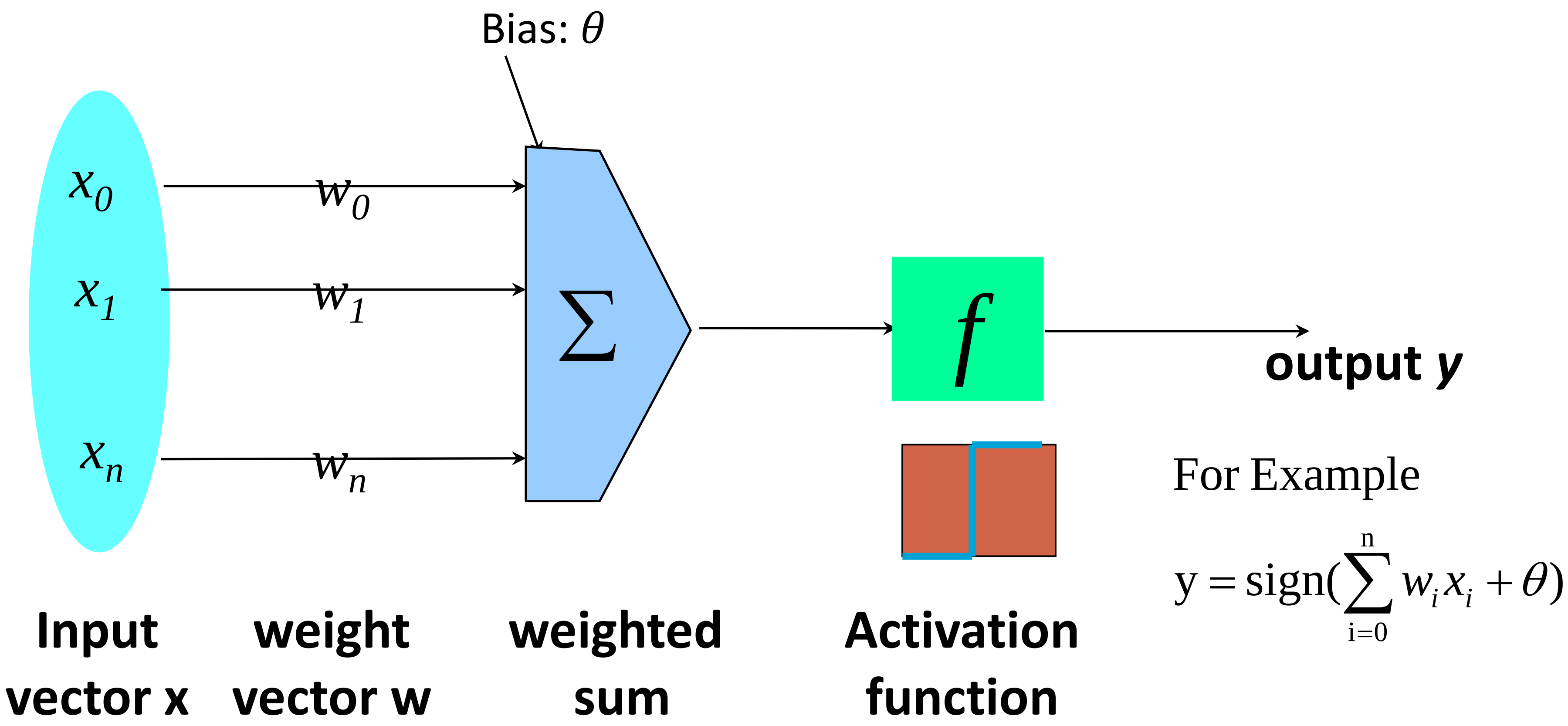
Support Vector Machine and Neural Network

- Support Vector Machine
- Neural Network
- Summary

Artificial Neural Networks

- Consider humans:
  - Neuron switching time ~.001 second
  - Number of neurons ~10^{10}
  - Connections per neuron ~10^{4-5}
  - Scene recognition time ~.1 second
  - 100 inference steps doesn't seem like enough -> parallel computation
- Artificial neural networks
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed process
  - Emphasis on tuning weights automatically

Single Unit: Perceptron



- An *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

Perceptron Training Rule

For each training data point:

$$w_i \leftarrow w_i + \Delta w_i$$

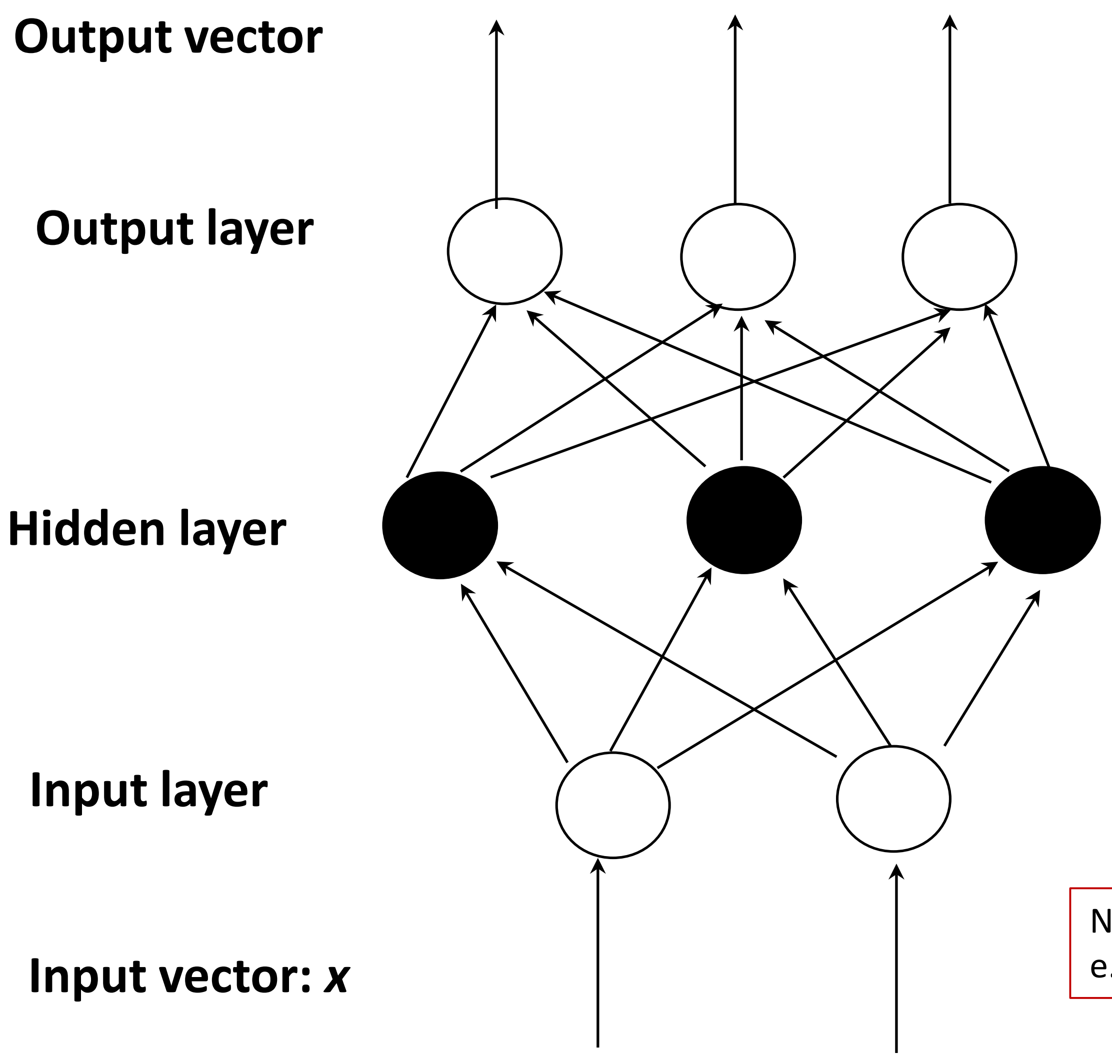
where

$$\Delta w_i = \eta(t - o)x_i$$

- t: target value (true value)
- o: output value
- η: learning rate (small constant)

A Multi-Layer Feed-Forward Neural Network

A **two-layer** network



$$\boldsymbol{y} = g(W^{(2)}\boldsymbol{h} + b^{(2)})$$

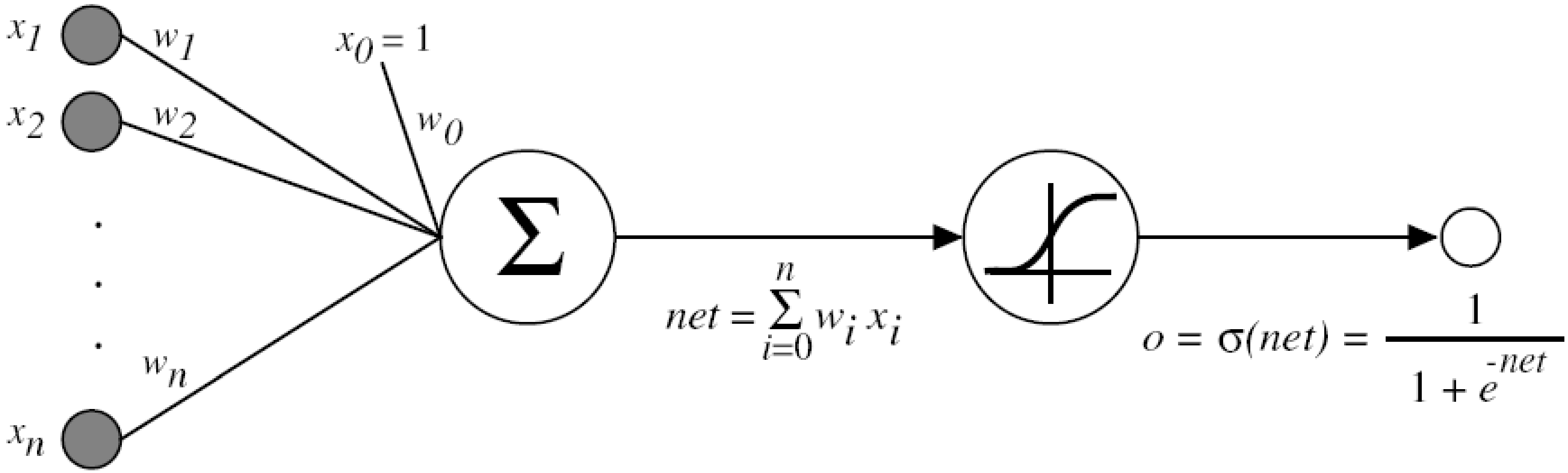
$$\boldsymbol{h} = f(W^{(1)}\boldsymbol{x} + b^{(1)})$$

Bias term

Weight matrix

Nonlinear transformation,
e.g. sigmoid transformation

Sigmoid Unit

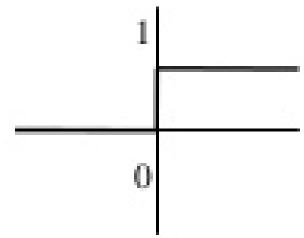


- $\sigma(x) = \frac{1}{1+e^{-x}}$ is a sigmoid function
  - Property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$
  - Will be used in learning

Activation functions

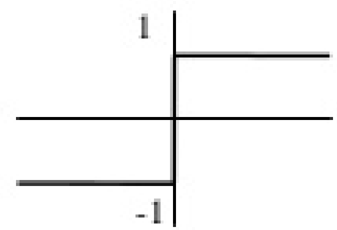
- **Step** function

$$step_t(x) = \begin{cases} 1 & x > t \\ 0 & otherwise \end{cases}$$



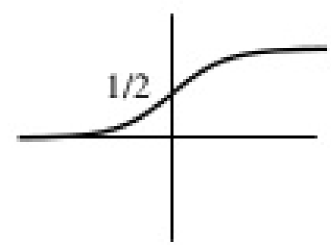
- **Sign** function

$$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & altrimenti \end{cases}$$



- **Sigmoid** function

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



How A Multi-Layer Neural Network Works

- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward**: None of the weights cycles back to an input unit or to an output unit of a previous layer
- From a math point of view, networks perform **nonlinear regression**: **Given enough hidden units and enough training samples, they can closely approximate any continuous function**

Defining a Network Topology

- Decide the **network topology:** Specify # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalize the **input** values for each attribute measured in the training tuples to [0.0—1.0]
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a different network topology or a different set of initial weights

Learning by Backpropagation

- Backpropagation: A **neural network** learning algorithm
- Started by psychologists and neurobiologists to develop and test computational analogues of neurons
- During the learning phase, the **network learns by adjusting the weights** so as to be able to predict the correct class label of the input tuples
- Also referred to as **connectionist learning** due to the connections between units

Backpropagation

- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the loss function** between the network's prediction and the actual target value, say **mean squared error**
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"

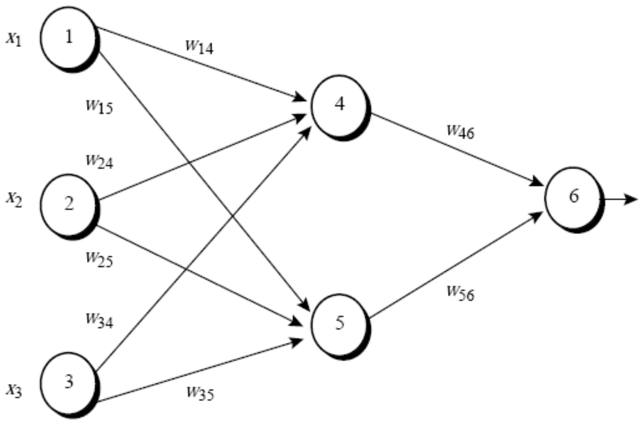
Example of Loss Functions

- Hinge loss
- Logistic loss
- Cross-entropy loss
- Mean square error loss
- Mean absolute error loss

A Special Case

- Activation function: Sigmoid

$$O_j = \sigma(\sum_i w_{ij} O_i)$$



- Loss function: mean square error

$$J = \frac{1}{2}\sum_j (T_j - O_j)^2,$$

T_j: *true value of output unit* j;

O_j: output value

Backpropagation Steps to Learn Weights

- Initialize weights to small random numbers, associated with biases
- Repeat until terminating condition meets
  - For each training example
    - **Propagate the inputs forward** (by applying activation function)
      - For a hidden or output layer unit j
        - Calculate net input: $I_j = \sum_i w_{ij} O_i + \theta_j$
        - Calculate output of unit j: $O_j = \sigma(I_j) = \frac{1}{1+e^{-I_j}}$
    - **Backpropagate the error** (by updating weights and biases)
      - For unit j in output layer: $Err_j = O_j(1 - O_j)(T_j - O_j)$
      - For unit j in a hidden layer: : $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$
      - Update weights: $w_{ij} = w_{ij} + \eta Err_j O_i$
      - Update bias: $\theta_j = \theta_j + \eta Err_j$
- Terminating condition (when error is very small, etc.)

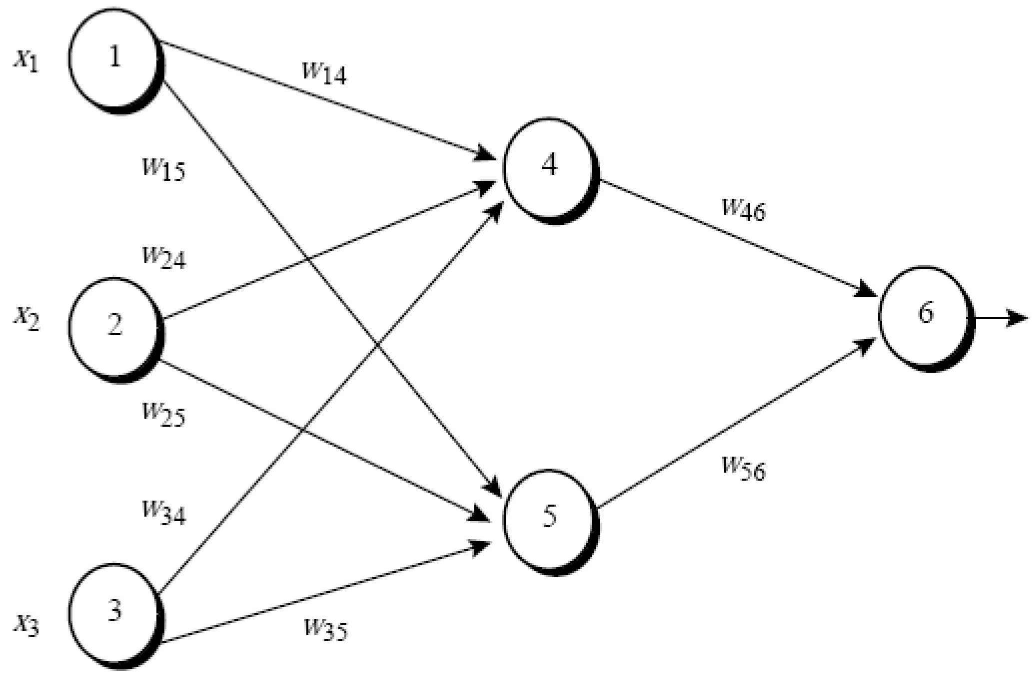
More on the hidden layer j

- Chain rule of first derivation

$$\frac{\partial J}{\partial w_{ij}} = \sum_k \frac{\partial J}{\partial O_k} \frac{\partial O_k}{\partial O_j} \frac{\partial O_j}{\partial w_{ij}}$$

$$\frac{\partial J}{\partial \theta_j} = \sum_k \frac{\partial J}{\partial O_k} \frac{\partial O_k}{\partial O_j} \frac{\partial O_j}{\partial \theta_j}$$

Example



A multilayer feed-forward neural network

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Initial Input, weight, and bias values

Example

- Input forward:

Table 9.2: The net input and output calculations.

Unit j	*Net input, I_j*	*Output, O_j*
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

- Error backpropagation and weight update:

Table 9.3: Calculation of the error at each node.

Unit j	*Err_j*
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Table 9.4: Calculations for weight and bias updating.

Weight or bias	*New value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Efficiency and Interpretability

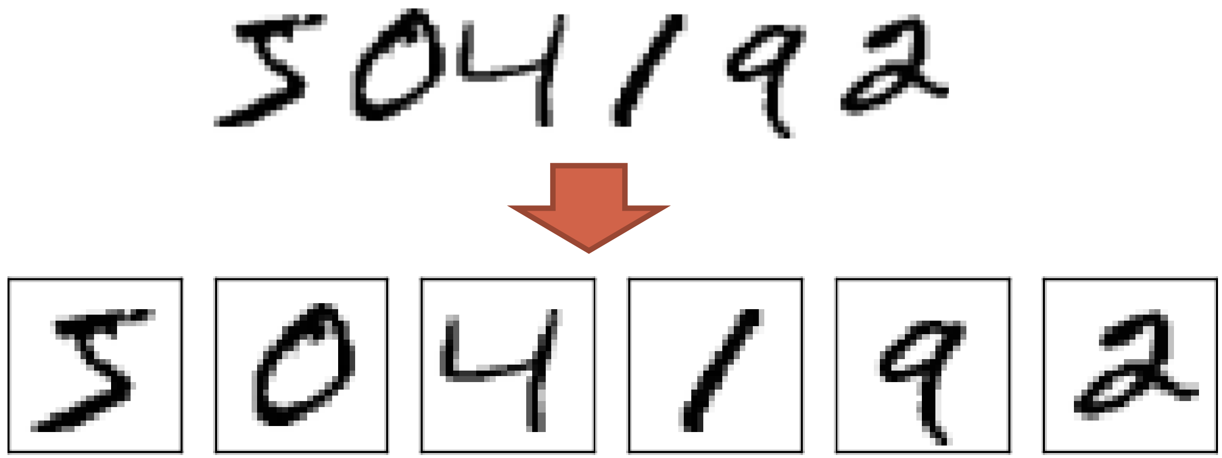
- **Efficiency** of backpropagation: Each iteration through the training set takes O(|D| * *w*), with |D| tuples and *w* weights, but # of iterations can be exponential to n, the number of inputs, in worst case
- For easier comprehension: **Rule extraction** by network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- **Sensitivity analysis**: assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules
  - E.g., If x decreases 5% then y increases 8%

Neural Network as a Classifier

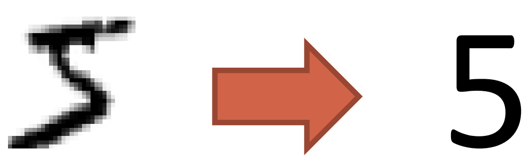
- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or "structure."
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of "hidden units" in the network
- Strength
  - High tolerance to noisy data
  - Successful on an array of real-world data, e.g., hand-written letters
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks
  - Deep neural network is powerful

Digits Recognition Example

- Obtain sequence of digits by segmentation

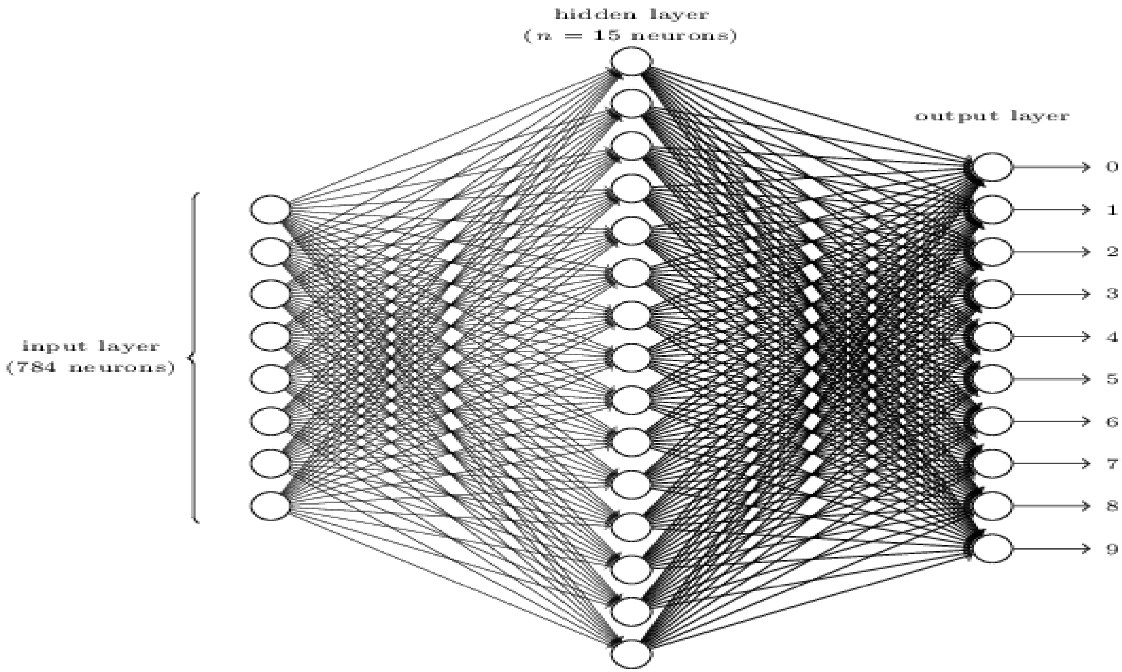


- Recognition (our focus)

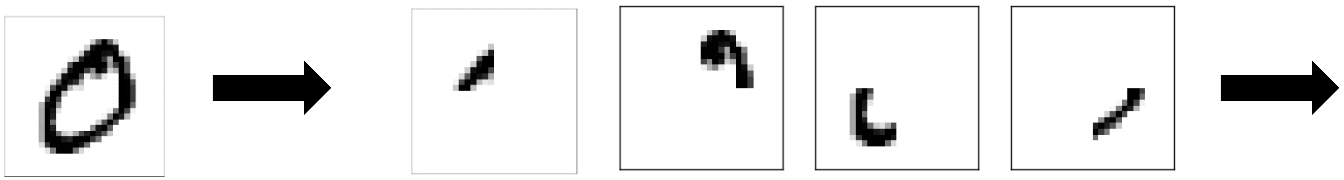


Digits Recognition Example

- The architecture of the used neural network



- What each neurons are doing?



0

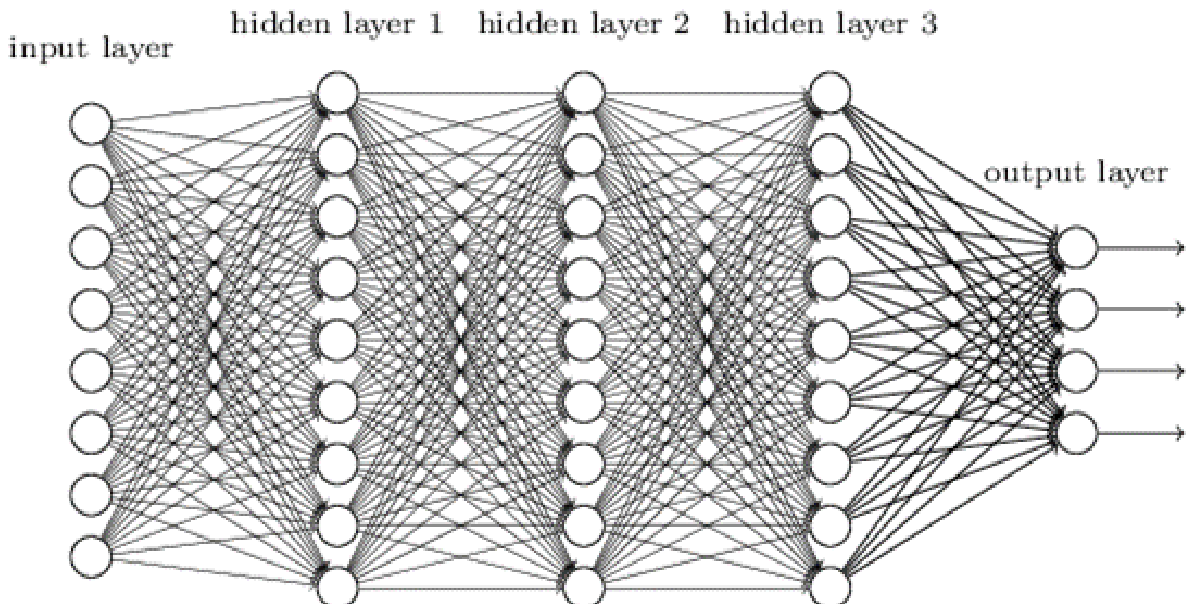
Input image

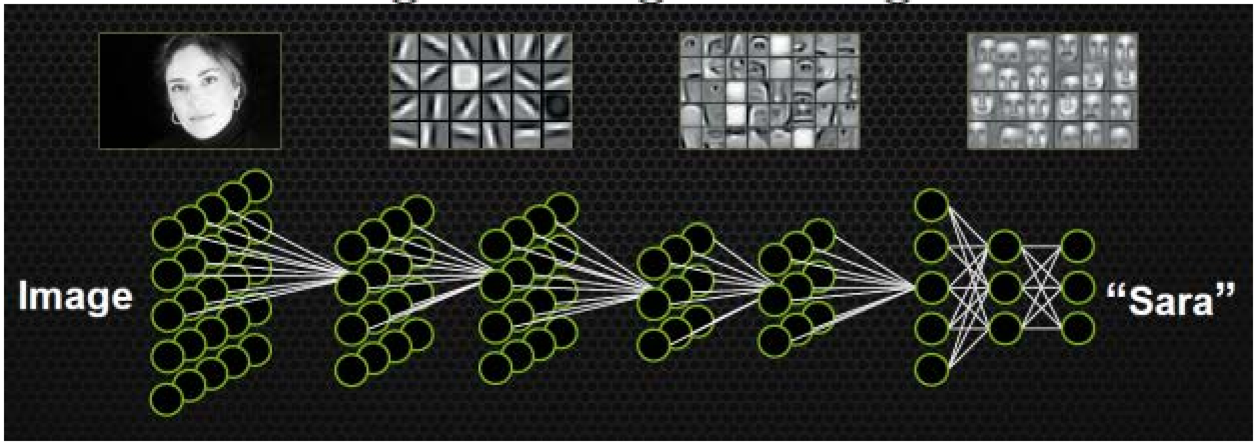
Activated neurons detecting image parts

Predicted number

Towards Deep Learning

Deep neural network





Deep Learning References

- http://neuralnetworksanddeeplearning.com/
- http://www.deeplearningbook.org/

Support Vector Machine and Neural Network

- Support Vector Machine
- Neural Network
- Summary

Summary

- Support Vector Machine
  - Linear classifier; support vectors; kernel SVM
- Neural Network
  - Feed-forward neural networks; activation function; loss function; backpropagation