Chapter 8

Support Vector Machines and Radial Basis Function Networks

Neural Networks: A Classroom Approach

Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

Statistical Learning Theory

Neural Networks: A Classroom Approach

Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

Learning from Examples

- Learning from examples is natural in human beings
  - Central to the study and design of artificial neural systems.
- Objective in understanding learning mechanisms
  - Develop software and hardware that can learn from examples and exploit information in impinging data.
  - Examples:
    - bit streams from radio telescopes around the globe
    - 100 TB on the Internet

Generalization

- Supervised systems learn from a training set $T = \{X_k, D_k\}$ $X_k \in \Re^n$, $D_k \in \Re$
- Basic idea: Use the system (network) in predictive mode $y_{predicted} = f(X_{unseen})$
- Another way of stating is that we require that the machine be able to successfully *generalize*.
  - Regression: $y_{predicted}$ is a real random variable
  - Classification: $y_{predicted}$ is either +1, -1

Approximation

- Approximation results discussed in Chapter 6 give us a guarantee that with a *sufficient number of hidden neurons* it should be possible to approximate a given function (as dictated by the input-output training pairs) to any arbitrary level of accuracy.
- Usefulness of the network depends primarily on the accuracy of its predictions of the output for unseen test patterns.

Important Note

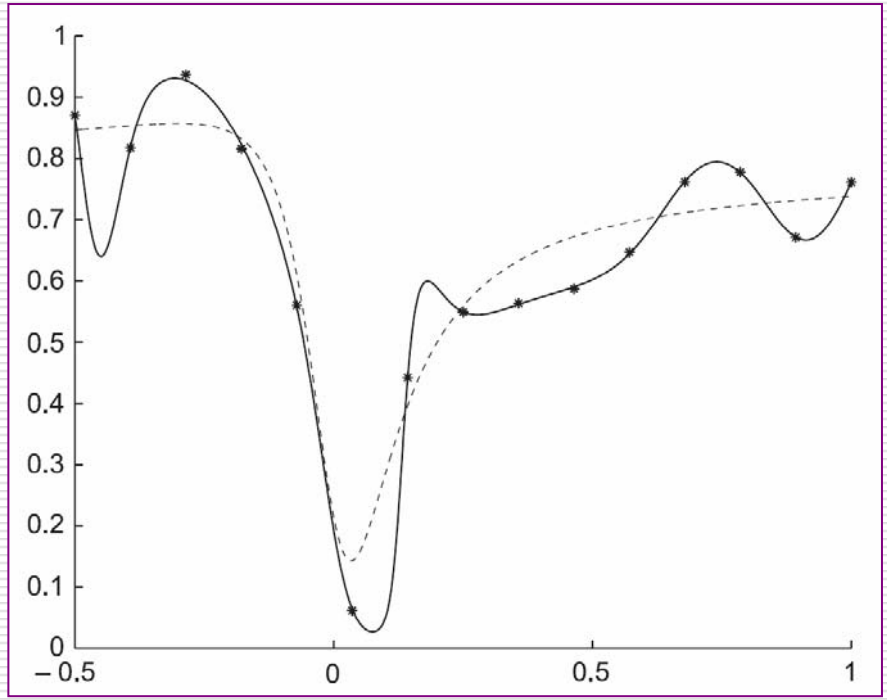
- Reduction of the mean squared error on the training set to a low level does not guarantee good generalization!
- Neural network might predict values based on unseen inputs rather inaccurately, even when the network has been trained to considerably low error tolerances.
- Generalization should be measured using test patterns similar to the training patterns
  - patterns drawn from the same probability distribution as the training patterns.

Broad Objective

- To model the generator function as closely as possible so that the network becomes capable of good generalization.
- Not to fit the model to the training data so accurately that it fails to generalize on unseen data.

Example of Overfitting

- Networks with too many weights (free parameters) overfits training data too accurately and fail to generalize
- Example:
  - 7 hidden node feedforward neural network
  - 15 noisy patterns that describe the deterministic univariate function (dashed line).
  - Error tolerance 0.0001.
  - Network learns each data point extremely accurately
  - Network function develops high curvature and fails to generalize



Occam's Razor Principle

- William Occam, c.1280–1349
  - *No more things should be presumed to exist than are absolutely necessary.*
- Generalization ability of a machine is closely related to
  - the capacity of the machine (functions it can represent)
  - the data set that is used for training.

Statistical Learning Theory

- Proposed by Vapnik
- Essential idea: **Regularization**
  - Given a finite set of training examples, the search for the best approximating function must be restricted to a small space of possible architectures.
  - When the space of representative functions and their capacity is large and the data set small, the models tend to over-fit and generalize poorly.
- Given a finite training data set, achieve the correct balance between accuracy in training on that data set and the capacity of the machine to learn any data set without error.

Optimal Neural Network

- Recall from Chapter 7 the sum of squares error function

$$\mathcal{E} = \frac{1}{2}\int \Big(f(X, W) - E[d|X] \Big)^2 \, p(X)\, dX + \frac{1}{2}\int \sigma^2 \, p(X)\, dX$$

The optimal network function we are in search of minimizes the error by trying to make the first integral zero

Residual error: average training data variance conditioned on the input

- Optimal neural network satisfies

$$\hat{f}(X, W) = E[d|X]$$

Training Dependence on Data

- Network deviation from the desired average is measured by

$$\left(f(X, W) - E[d|X]\right)^2$$

- Deviation depends on a particular instance of a training data set
- Dependence is easily eliminated by averaging over the ensemble of data sets of size Q

$$E_Q\left[(f(X, W) - E[d|X])^2\right]$$

Causes of Error: Bias and Variance

- Bias: network function itself differs from the regression function $E[d|X]$

- Variance: network function is sensitive to the selection of the data set
  - generates large error on some data sets and small errors on others.

Quantification of Bias & Variance

$$\begin{aligned}\big(f(\cdot)-E[d|X]\big)^2 &= \big(f(\cdot)-E_Q[f(\cdot)]+E_Q[f(\cdot)]-E[d|X]\big)^2 \\ &= \big(f(\cdot)-E_Q[f(\cdot)]\big)^2+\big(E_Q[f(\cdot)]-E[d|X]\big)^2 \\ &+2\big(f(\cdot)-E_Q[f(\cdot)]\big)\big(E_Q[f(\cdot)]-E[d|X]\big)\end{aligned}$$

- Consequently,

$$\begin{aligned}E_Q\Big[\big(f(\cdot)-E[d|X]\big)^2\Big] &= E_Q\Big[\big(f(\cdot)-E_Q[f(\cdot)]\big)^2\Big] \\ &+E_Q\Big[\big(E_Q[f(\cdot)]-E[d|X]\big)^2\Big]\end{aligned}$$

Bias-Variance Dilemma

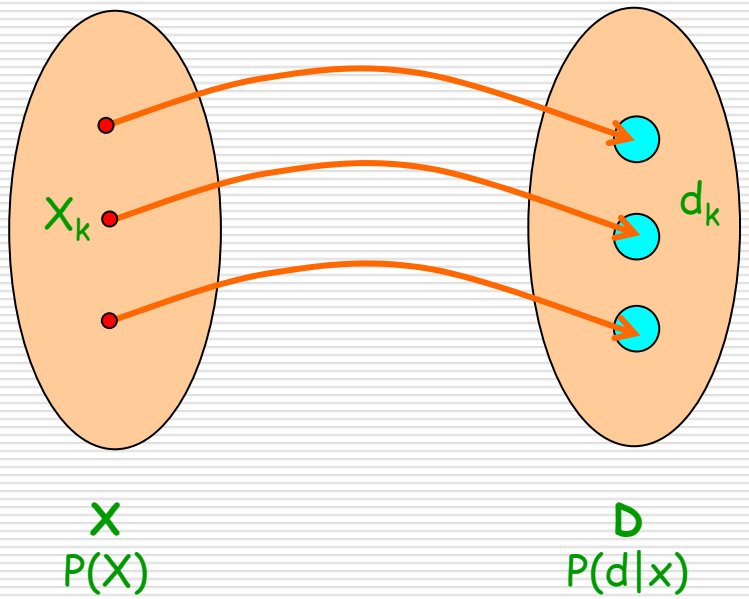
- Separation of the ensemble average into the bias and variance terms:

$$E_Q\left[\left(f(\cdot) - E[d|X]\right)^2\right] = \underbrace{\left(E_Q[f(\cdot)] - E[d|X]\right)^2}_{(\text{bias})^2} + \underbrace{E_Q\left[\left(f(\cdot) - E_Q[f(\cdot)]\right)^2\right]}_{(\text{variance})}$$

- Strike a balance between the ratio of training set size to network complexity such that both bias and variance are minimized.
  - → Good generalization
- Two important factors for valid generalization
  - the number of training patterns used in learning
  - the number of weights in the network

Stochastic Nature of T

- T is sampled stochastically
- $X_k \subset \mathbf{X} \in \Re^n$, $d_k \subset \mathbf{D} \in \Re$
- X_k does not map uniquely to an element, rather a distribution
- Unkown probability distribution p(X,d) defined on $\mathbf{X} \times \mathbf{D}$ determines the probability of observing (X_k, d_k)



Risk Functional

- To successfully solve the regression or classification task, a neural network learns an *approximating* function f(X, W)
- Define the expected risk as

$$R[f] = E\Big[L(d, f(X, W))\Big] = \int L(d, f(X, W))\, dp(X, d)$$

- The risk is a function of functions f drawn from a function space F

Loss Functions

- Square error function

$$L(d, f(X, W)) = (d - f(X, W))^2$$

- Absolute error function

$$L(d, f(X, W)) = |d - f(X, W)|$$

- 0-1 Loss function

$$L(d, f(X, W)) = \begin{cases} 0 & \text{if } f(X, W) = d \\ 1 & \text{otherwise} \end{cases}$$

Optimal Function

- The optimal function f_o minimizes the expected risk R[f]

$$f_o = f(X, W_o) = \arg\min_{\mathcal{F}} R[f]$$

- f_o defined by optimal parameters; W_o is the ideal estimator
- Remember: p(X,d) is unknown, and f_o has to be estimated from finite samples
- f_o cannot be found in practice!

Empirical Risk Minimization (ERM)

- ERM principle is an induction principle that we can use to train the machine using the limited number of data samples at hand
- ERM generates a stochastic approximation of R using T called the empirical risk R_e

$$R_e[f] = \frac{1}{Q} \sum_{k=1}^{Q} L(d_k, f(X_k, W))$$

Empirical Risk Minimization (ERM)

- The best minimizer of the empirical risk replaces the optimal function f_o
- ERM replaces R by R_e and f_o by $\hat{f}$
- Question:
  - Is the minimizer $\hat{f}$ close to f_o ?

Two Important Sequence Limits

- To ensure minimizer $\hat{f}$ close to f_o we need to find the conditions for consistency of the ERM principle.
- Essentially requires specifying the necessary and sufficient conditions for convergence of the following two limits of sequences in a probabilistic sense.

First Limit

- Convergence of the values of expected risks $R[\hat{f}_Q]$ of functions $\hat{f}_Q$, $Q = 1,2,\ldots$ that minimize the empirical risk $R_e[\hat{f}_Q]$ over training sets of size Q, to the minimum of the true risk

$$\lim_{Q \to \infty} R[\hat{f}_Q] \underset{P}{\longrightarrow} R[f_o]$$

- Another way of saying that solutions found using ERM converge to the best possible solution.

Second Limit

- Convergence of the values of empirical risk $R_e[\hat{f}_Q]$ $Q = 1,2,\ldots$ over training sets of size Q, to the minimum of the true risk

$$\lim_{Q\to\infty} R_e[\hat{f}_Q] \underset{P}{\longrightarrow} R[f_o]$$

- This amounts to stating that the empirical risk converges to the value of the smallest risk.
- Leads to the Key Theorem by Vapnik and Chervonenkis

Key Theorem

- Let L(d,f(X,W)) be a set of functions with a bounded loss for probability measure p(X,d) :

$$A \leq \int L(d,\, f(X,\, W))\, dp(X,\, d) \leq B$$

- Then for the ERM principle to be consistent, it is necessary and sufficient that the empirical risk $R_e[f]$ converge uniformly to the expected risk R[f] over the set L(d,f(X,W)) such that

$$\lim_{Q \to \infty} P\left[\sup_{f \in \mathcal{F}} (R[f] - R_e[f]) > \epsilon\right] = 0, \forall \epsilon > 0$$

- This is called *uniform one-sided convergence in probability*

Points to Take Home

- In the context of neural networks, each function is defined by the weights W of the network.
- Uniform convergence Theorem and VC Theory ensure that W which is obtained by minimizing R_e also minimizes R as the number Q of data points increases towards infinity.

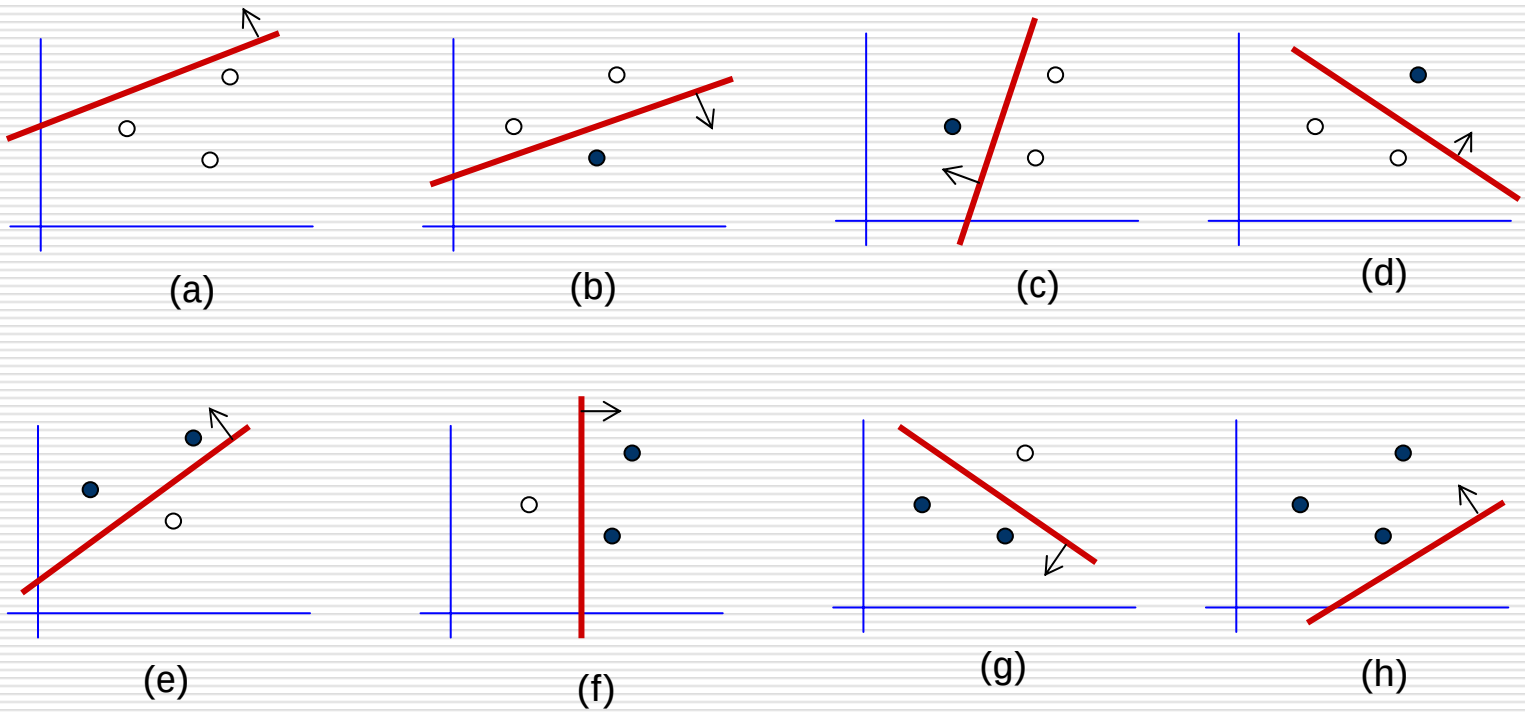
Points to Take Home

- Remember: we have a finite data set to train our machine.
- When any machine is trained on a specific data set (which is finite) the function it generates is a biased approximant which may minimize the empirical risk or approximation error, but not necessarily the expected risk or the generalization error.

Indicator Functions and Labellings

- Consider the set of indicator functions
  - F = {f(X,W)} mapping points in $\Re^n$ into {0,1} or {-1,1}.
- Labelling: An assignment of 0,1 to Q points in $\Re^n$
- Q points can be labelled in 2^Q ways

Labellings in 3-d



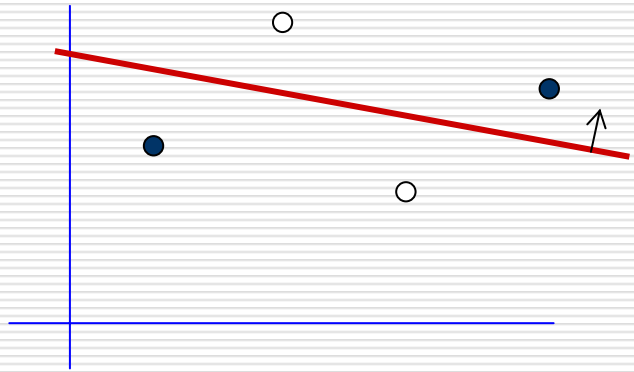
Three points in R^2 can be labelled in eight different ways.
A linear oriented decision boundary can shatter all eight labellings.

Vapnik–Chervonenkis Dimension

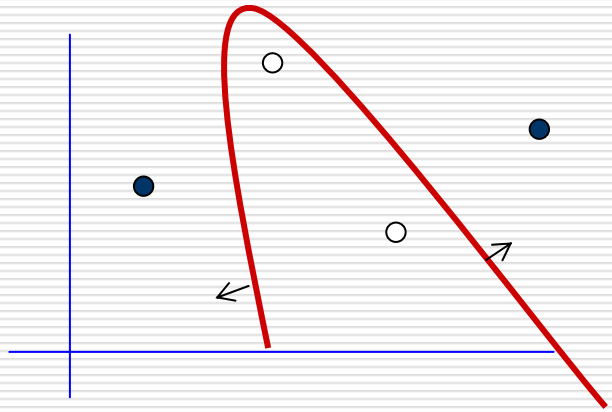
- If the set of indicator functions can correctly classify each of the possible 2^Q labellings, we say the set of points is shattered by F.
- The VC-dimension h of a set of functions F is the largest set of points that can be shattered by the set in question.

VC-Dimension of Linear Decision Functions in $\Re^2$ is 3

- Labelling of four points in $\Re^2$ that cannot be correctly separated by a linear oriented decision boundary



- A quadratic decision boundary can separate this labelling!



VC-Dimension of Linear Decision Functions in $\Re^n$

- At most $n+1$ points can be shattered by oriented hyperplanes in $\Re^n$
- VC-dimension is $n+1$
- Equal to the number of free parameters

Growth Function

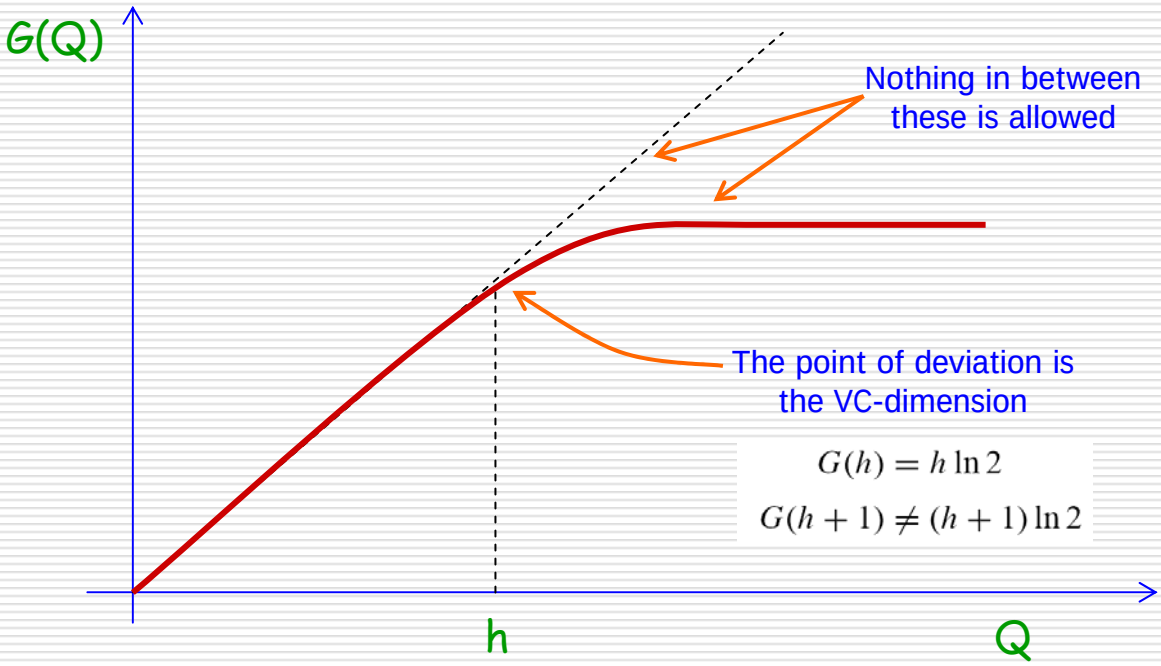
- Consider Q points in $\Re^n$
- N_{XQ} labellings can be shattered by F
- $N_{XQ} \leq 2^Q$
- Growth function

$$G(Q) = \ln\left(\sup_{X_Q}(N_{X_Q})\right) \leq Q \ln 2$$

Growth Function and VC Dimension

$$G(Q) < h\left(\ln\left[\frac{Q}{h}\right] + 1\right)$$

$$G(Q) = Q \ln 2$$



Towards Complexity Control

- In a machine trained on a given training set the appoximants generated are naturally biased towards those data points.
- Necessary to ensure that the model chosen for representation of the underlying function has a complexity (or capacity) that matches the data set in question.
- Solution: *structural risk minimization*
- Consequence of VC-theory
  - The difference between the empirical and expected risk can be bounded in terms of the VC-dimension.

VC-Confidence, Confidence Level

- For binary classification loss functions which take on values either 0,1, for some $0 \le \eta \le 1$ the following bound holds with probability at least $1-\eta$:

$$R[f] \le R_e[f] + \sqrt{\frac{h(\ln \frac{2Q}{h} + 1) - \ln(\eta/4)}{Q}}$$

VC-confidence holds with confidence level $1-\eta$

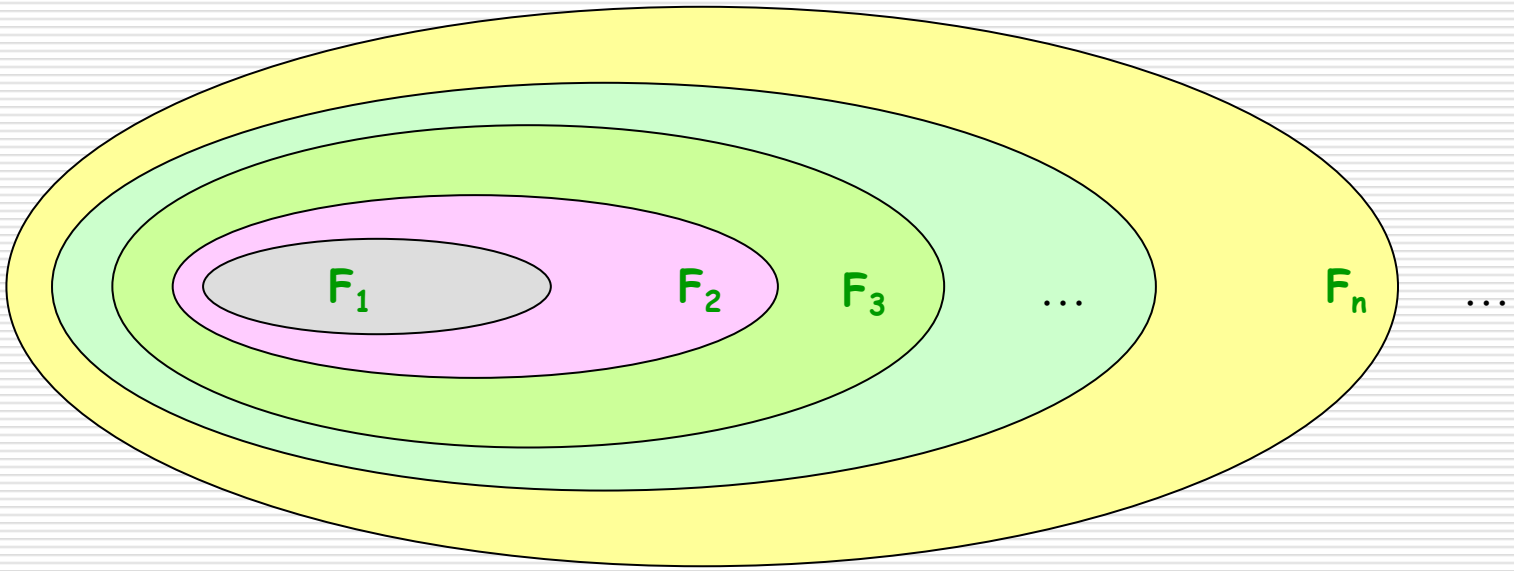
Empirical error

Structural Risk Minimization

- Structural Risk Minimization (SRM):
  - Minimize the combination of the empirical risk *and* the complexity of the hypothesis space.
- Space of functions F is very large, and so restrict the focus of learning to a smaller space called the hypothesis space.
- SRM therefore defines a nested sequence of hypothesis spaces
  - $F_1 \subset F_2 \subset \ldots \subset F_n \subset \ldots$

  - VC-dimensions $h_1 \leq h_2 \leq \ldots \leq h_n \leq \ldots$

Nested Hypothesis Spaces form a Structure



VC-dimensions $h_1 \leq h_2 \leq \ldots \leq h_n \leq \ldots$

Empirical and Expected Risk Minimizers

- $\hat{f}_{i,Q}$ minimizes the empirical error over the Q points in space F_i

$$\hat{f}_{i,Q} = \arg\min_{f \in F_i} R_e[f_{i,Q}]$$

- Is different from $\hat{f}_i$ the true minimizer of the expected risk R in F_i

A Trade-off

$$R[f] \leq R_e[f] + \sqrt{\frac{h(\ln \frac{2Q}{h} + 1) - \ln(\eta/4)}{Q}}$$

- Successive models have greater flexibility such that the empirical error can be pushed down further.
- Increasing i increases the VC-dimension and thus the second term
- Find $F_{n(Q)}$, the minimizer of the r.h.s.
- Goal: select an appropriate hypothesis space to match the training data complexity to the model capacity.
- This gives the best generalization.

Approximation Error: Bias

- Essentially two costs associated with the learning of the underlying function.
- Approximation error, E_A:
  - Introduced by restricting the space of possible functions to be less complex than the target space
  - Measured by the difference in the expected risks associated with the best function and the optimal function that measures R in the target space
  - Does not depend on the training data set; only on the approximation power of the function space

Estimation Error: Variance

- Now introduce the finite training set with which we train the machine.
- Estimation Error, E_E:
  - Learning from finite data minimizes the empirical risk; not the expected risk.
  - The system thus searches a minimizer of the expirical risk; not the expected risk
  - This introduces a second level of error.
- Generalization error = $E_A + E_E$

A Warning on Bound Accuracy

- As the number of training points increase, the difference between the empirical and expected risk decreases.
- As the confidence level increases (η becomes smaller), the VC confidence term becomes increasingly large.
- With a finite set of training data, one cannot increase the confidence level indefinitely:
  - the accuracy provided by the bound decreases!

Support Vector Machines

Neural Networks: A Classroom Approach

Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

Origins

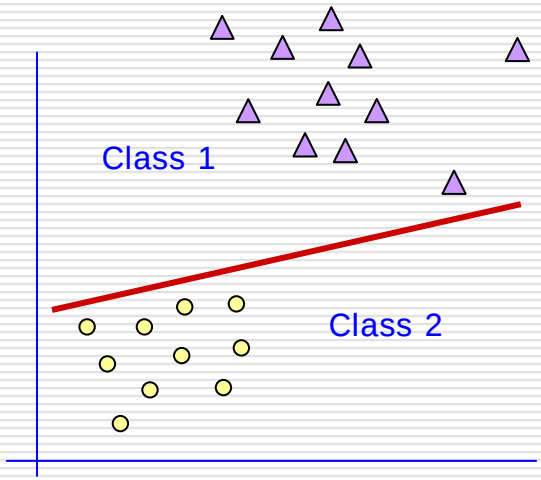
- Support Vector Machines (SVMs) have a firm grounding in the VC theory of statistical learning
- Essentially implements structural risk minimization
- Originated in the work of Vapnik and co-workers at the AT&T Bell Laboratories
- Initial work focussed on
  - optical character recognition
  - object recognition tasks
- Later applications
  - regression and time series prediction tasks

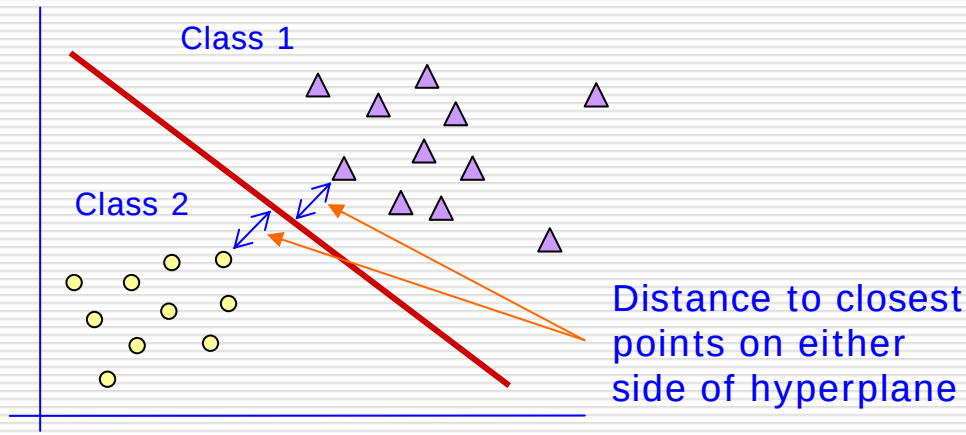
Context

- Consider two sets of data points that are to be classified into one of two classes C_1, C_2
- Linear indicator functions (TLN hyperplane classifiers) which is the bipolar signum function
- Data set is linearly separable
- $T = \{X_k, d_k\}$, $X_k \in \Re^n$, $d_k \in \{-1,1\}$
- C_1: positive samples C_2: negative samples

SVM Design Objective

- Find the hyperplane that maximizes the margin





Hypothesis Space

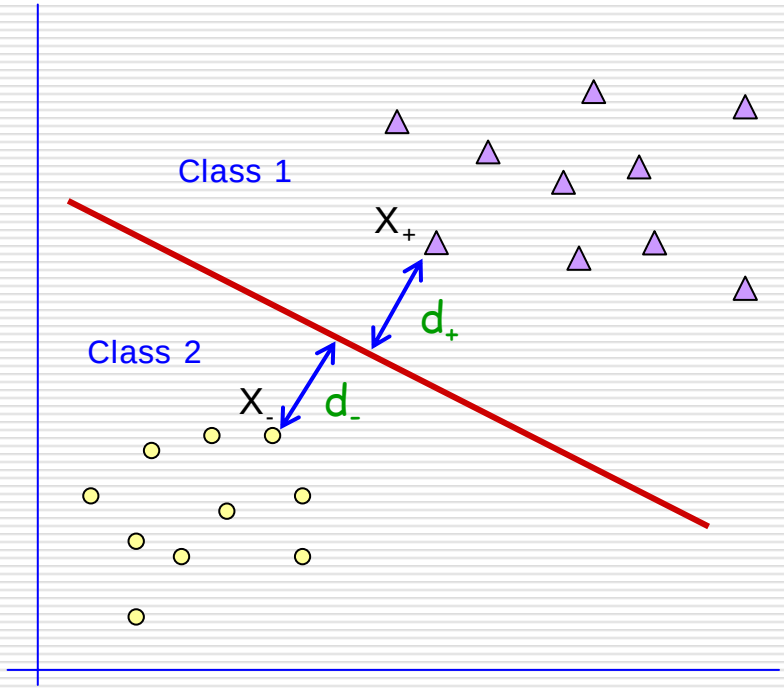
- Our hypothesis space is the space of functions

$$f(X, W, w_0) = \text{sign}(W \cdot X + w_0)$$

- Similar to Perceptron, but now we want to maximize the margins from the separating hyperplane to the nearest positive and negative data points.
- Find the maximum margin hyperplane for the given training set.

Definition of Margin

- The perpendicular distance to the closest positive sample (d_+) or negative sample (d_-) is called the margin



Reformulation of Classification Criteria

- Originally

$$W \cdot X_i + w_0 > 0 \qquad d_i = +1, \quad \text{Class } \mathcal{C}_1$$
$$W \cdot X_i + w_0 < 0 \qquad d_i = -1, \quad \text{Class } \mathcal{C}_2$$

- Reformulated as

$$W \cdot X + w_0 \geq +\Delta \qquad d_i = +1$$
$$W \cdot X + w_0 \leq -\Delta \qquad d_i = -1$$

- Introducing a margin Δ so that the hyperplane satisfies

$$W \cdot X + w_0 = \pm\Delta$$

Canonical Separating Hyperplanes

- Satisfy the constraint $\Delta = 1$
- Then we may write

$$W \cdot X + w_0 \geq +1 \qquad d_i = +1$$
$$W \cdot X + w_0 \leq -1 \qquad d_i = -1$$

or more compactly

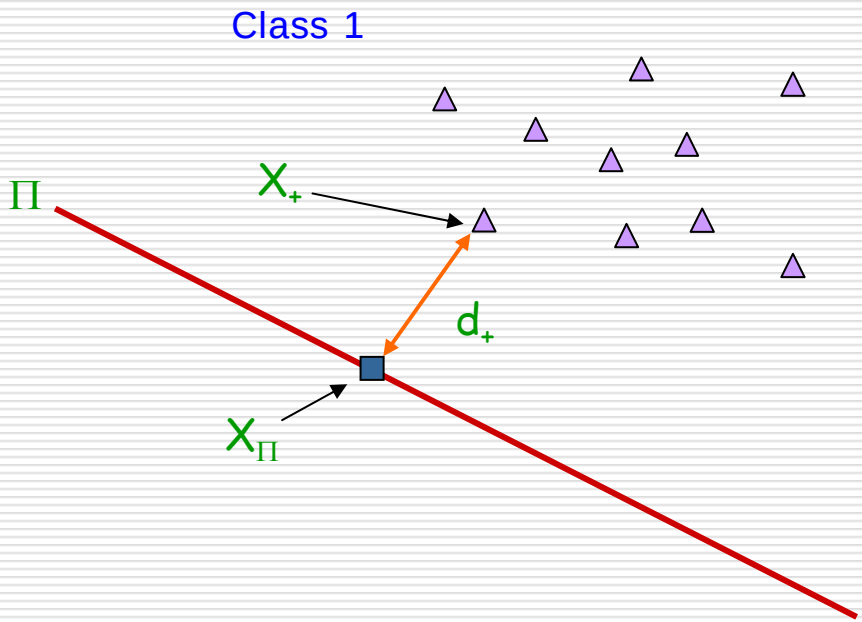
$$d_i(X_i \cdot W + w_0) - 1 \geq 0 \quad \forall i$$

Notation

- X_+ is the data point from C_1 closest to hyperplane Π, and X_Π is the unique point on Π that is closest to X_+
- Maximize d_+
  - $d_+ = || X_+ - X_\Pi ||$
- From the defining equation of hyperplane Π,

$$W \cdot X_+ + w_0 = 1$$

$$W \cdot X_\Pi + w_0 = 0$$



Expression for the Margin

- Defining equations of hyperplane yield

$$W \cdot (X_+ - X_\Pi) = 1$$

- Noting that $X_+ - X_\Pi$ is also perpendicular to Π

$$W \cdot (X_+ - X_\Pi) = W \cdot \left(\|X_+ - X_\Pi\| \frac{W}{\|W\|} \right)$$

$$= \|X_+ - X_\Pi\| \frac{\|W\|^2}{\|W\|}$$

$$= \|X_+ - X_\Pi\| \|W\|$$

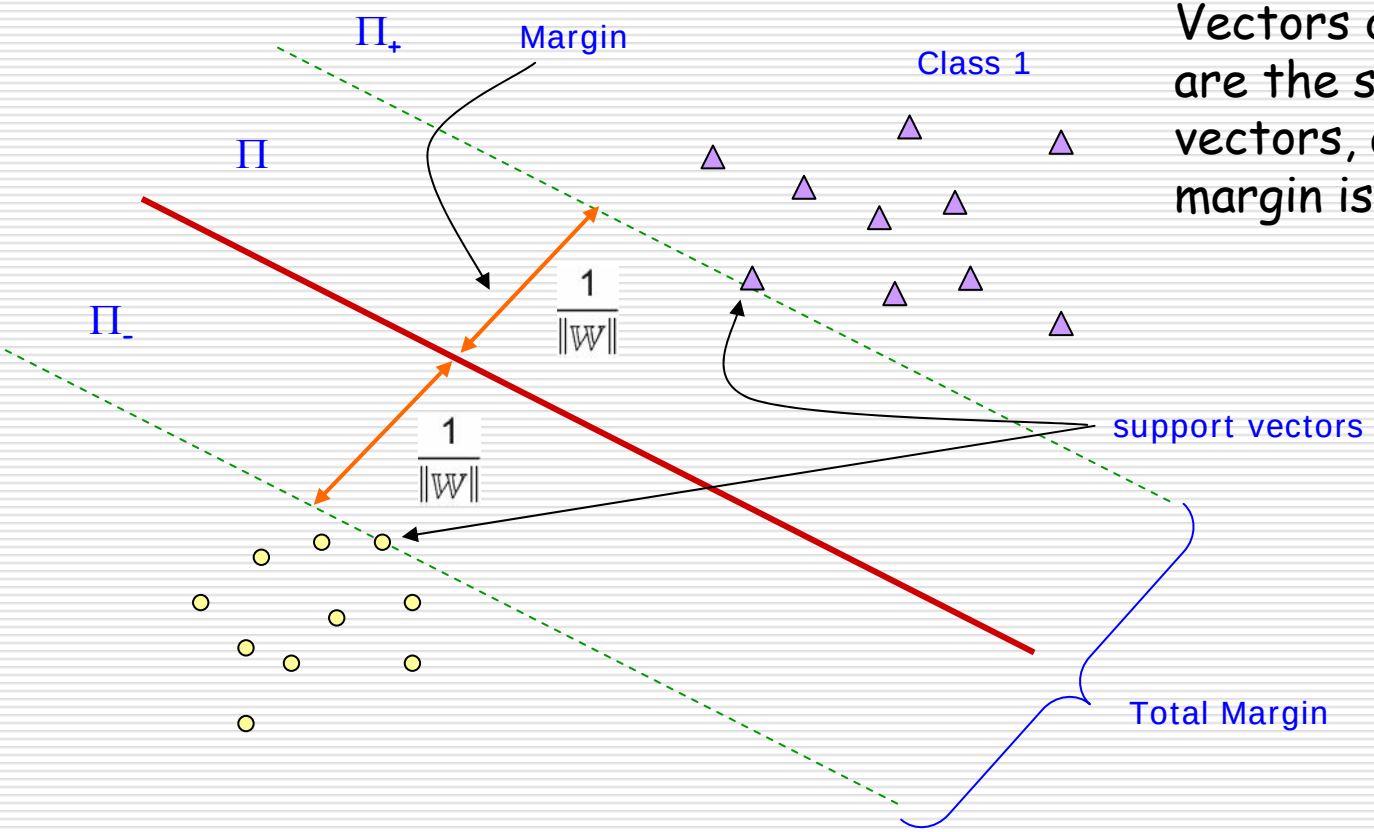
- Eventually yields

$$d_+ = \|X_+ - X_\Pi\| = \frac{1}{\|W\|}$$

- Total margin

$$M = d_+ + d_- = \frac{2}{\|W\|}$$

Support Vectors



Vectors on the margin are the support vectors, and the total margin is 2/||W||

SVM and SRM

- If all data point lie within an n-dimensional hypersphere of radius ρ then the set of indicator functions

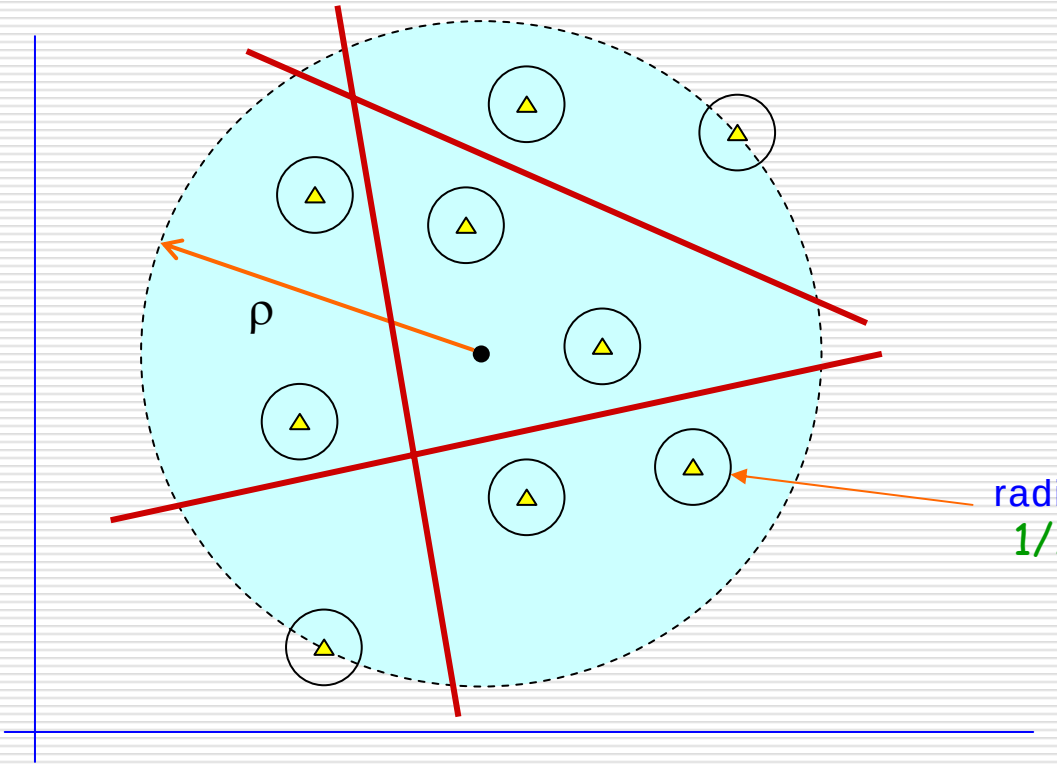
$$F_A = \{ f = \text{sign}(W \cdot X + w_0) \quad | \quad \|W\| \leq A \}$$

- has a VC-dimension that satisfies the following bound

$$h \leq \min\{\rho^2 A^2, n\} + 1$$

- Distance to closest point is 1/||W||
- Constrain ||W|| ≤ A then the distance from the hyperplane to the closest data point must be greater than 1/A. Therefore, Minimize ||W||

SVM Implements SRM



An SVM implements SRM by constraining hyperplanes to lie outside hyperspheres of radius 1/A

$$F_{A_i} = \{ f = \text{sign}(W \cdot X + w_0) \quad | \quad \|W\| \leq A_i \}, \quad A_1 < A_2 < A_3 < \cdots$$

Objective of the Support Vector Machine

- Given $T = \{X_k, d_k\}$, $X_k \in \Re^n$, $d_k \in \{-1,1\}$
- C_1: positive samples C_2: negative samples
- Attempt to classify the data using the smallest possible weight vector norm $||W||$ or $||W||^2$
- Maximize the margin $1/||W||$
- Minimize

$$\Psi(W) = \frac{1}{2}\|W\|^2$$

- subject to the constraints

$$d_i(X_i \cdot W + w_0) - 1 \geq 0 \quad i = 1, \ldots, Q$$

Method of Lagrange Multipliers

- Used for two reasons
  - the constraints on the Lagrangian multipliers are easier to handle;
  - the training data appear in the form of dot products in the final equations a fact that we extensively exploit in the non-linear support vector machine.

Construction of the Lagrangian

- Formulate problem in primal space
- $\Lambda = (\lambda_1, \ldots, \lambda_Q)$, $\lambda_i \geq 0$ is a vector of Lagrange multipliers

$$L_P(W, w_0, \Lambda) = \frac{1}{2}\|W\|^2 - \sum_{i=1}^{Q} \lambda_i [d_i(X_i \cdot W + w_0) - 1]$$

- Saddle point of L_p is the solution to the problem

Shift to Dual Space

- Makes the optimization problem much cleaner in the sense that requires only maximization of λ_i
- Translation to the dual form is possible because both the cost function and the constraints are strictly convex.
- Kuhn-Tucker conditions for the optimum of a constrained optimization problem are invoked to effect the translation of Lp to the dual form

Shift to Dual Space

- Partial derivatives of L_p with respect to the primal variables must vanish at the solution points

$$\frac{\partial L_P(W, w_0, \Lambda)}{\partial W} = W - \sum_{i=1}^{Q} \lambda_i d_i X_i = 0$$

$$\frac{\partial L_P(W, w_0, \Lambda)}{\partial w_0} = \sum_{i=1}^{Q} \lambda_i d_i = \Lambda \cdot D = 0$$

- $D = (d_1, \ldots d_Q)^T$ is the vector of desired values

Kuhn-Tucker Complementarity Conditions

- Constraint $d_i(X_i \cdot W + w_0) - 1 \geq 0 \quad \forall i$
- Must be satisfied with equality

$$\lambda_i \left[d_i(W \cdot X_i + w_0) - 1 \right] = 0, \quad i = 1, \ldots, Q$$

- Yields the dual formulation

$$L_D(\Lambda) = \sum_{i=1}^{Q} \lambda_i - \frac{1}{2} \sum_{i=1}^{Q} \sum_{j=1}^{Q} \lambda_i \lambda_j d_i d_j (X_i \cdot X_j)$$

$$L_D(\Lambda) = \Lambda \cdot \mathbf{1} - \frac{1}{2} \Lambda^T \mathbf{H} \Lambda$$

Final Dual Optimization Problem

- Maximize

$$L_D(\Lambda) = \Lambda \cdot \mathbf{1} - \frac{1}{2}\Lambda^T \mathbf{H} \Lambda$$

- with respect to the Lagrange multipliers, subject to the constraints:

$$\Lambda \cdot D = 0$$

$$\Lambda \geq \mathbf{0}$$

- Quadratic programming optimization problem

Support Vectors

- Numeric optimization yields optimized Lagrange multipliers

$$\hat{\Lambda} = (\lambda_1, \ldots, \lambda_Q)^T$$

- Observation: some Lagrange multipliers go to zero.
- Data vectors for which the Lagrange multipliers are greater than zero are called *support vectors*.
- For all other data points which are not support vectors, $\lambda_i = 0$.

Optimal Weights and Bias

- n_s is the number of support vectors

$$\hat{W} = \sum_{i=1}^{Q} \hat{\lambda}_i d_i X_i = \sum_{k=1}^{n_s} \hat{\lambda}_k d_k X_k$$

- Optimal bias computed from the complementarity conditions

$$\hat{w_0} = \frac{1}{d_s} - \hat{W} \cdot X_s$$

- Usually averaged over all support vectors and uses Hessian

$$\hat{w_0} = \frac{1}{n_s}\left[\sum_{l=1}^{n_s}\left(\frac{1}{d_l} - \hat{W} \cdot X_l\right)\right] = \frac{1}{n_s}\left[\sum_{l=1}^{n_s} d_l\left(1 - \sum_{k=1}^{n_s} \hat{\lambda}_k H_{kl}\right)\right]$$

Classifying an Unknown Data Point

- Use a linear indicator function:

$$y(X) = \text{sign}\left(\sum_{i=1}^{n_s} d_i \hat{\lambda}_i (X \cdot X_i) + \hat{w}_0\right)$$

MATLAB Code: Linear SVM Linearly Separable Case

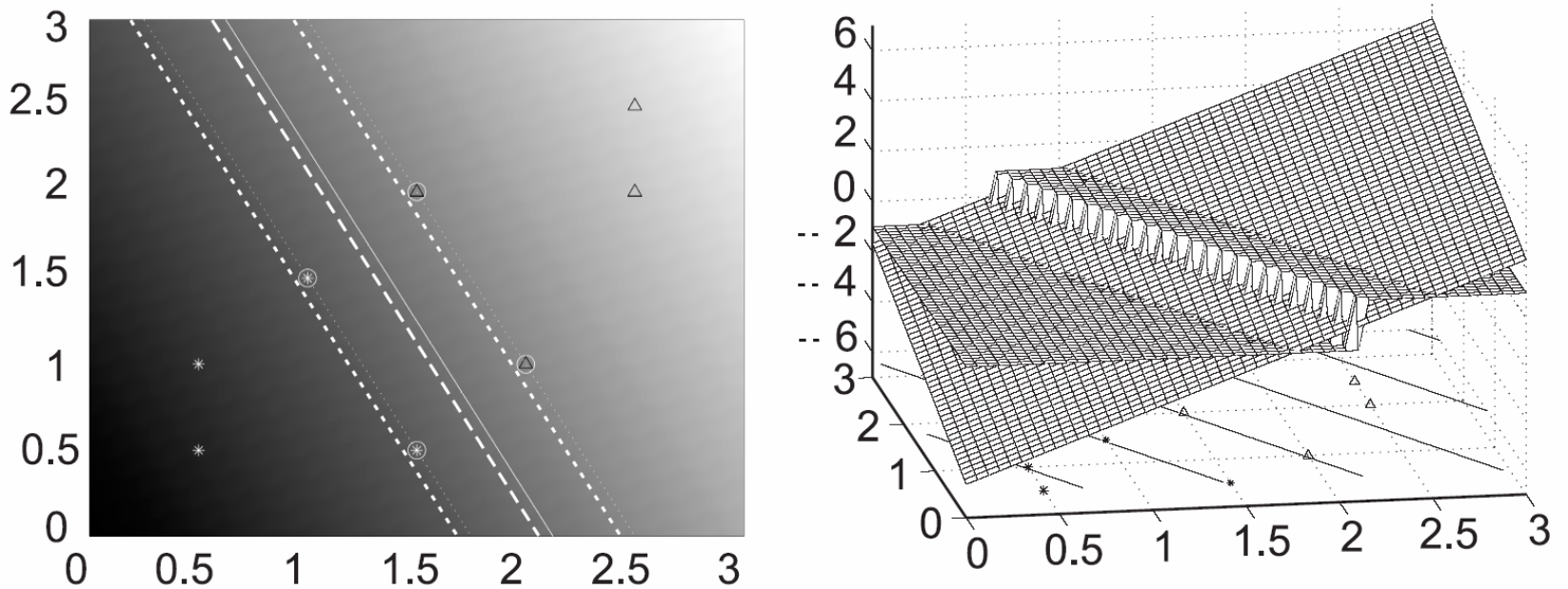
```
X=[0.5 0.5 % Data points
0.5 1.0
1. 1.5
1.5 0.5
1.5 2.0
2.0 1.0
2.5 2.0
2.5 2.5];
D=[-1 -1 -1 -1 1 1 1 1]';% Corresponding
      classes
q = size(X,1); % Size of data set
epsilon = 1e-5; % threshold for checking
      support vectors
H = zeros(q,q); % Initialize Hessian matrix
for i = 1:q % Set up the Hessian
for j = 1:q
H(i,j) = D(i)*D(j)*X(i,:)*X(j,:)';
end
end
...
f = -ones(q,1); %Vectors of ones
% Parameters for the Optimization problem
numeqconstraints = 1; % Number of equality
      constraints = 1
A = D'; % Set up the equality constraint
b = 0;
vlb = zeros(q,1); % Lower bound of lambdas = 0
vub = Inf*ones(q,1); % No upper bound
x0 = zeros(q,1); % Initial point is 0
%Invoke MATLAB's standard qp function for
      quadratic optimization...
[lambda alpha how] = qp(H, f, A, b, vlb, vub, x0,
      numeqconstraints);
svindex= find(lambda > epsilon);% Support vector
      indices
ns = length(svindex); % Number of support
      vectors
w_0 = (1/ns)*sum(D(svindex) - ...% Optimal bias
H(svindex,svindex)*lambda(svindex).*D(svindex));
```

Simulation Result

- Details of linearly separable data, support vectors and Lagrange multipliers values after optimization

Data point	Class	Support Vector	λ_i
0.5 0.5	−1	No	0.0
0.5 1.0	−1	No	0.0
1.0 1.5	−1	Yes	2.664
1.5 0.5	−1	Yes	1.780
1.5 2.0	1	Yes	1.775
2.0 1.0	1	Yes	2.669
2.5 2.0	1	No	0.0
2.5 2.5	1	No	0.0

Simulation Result

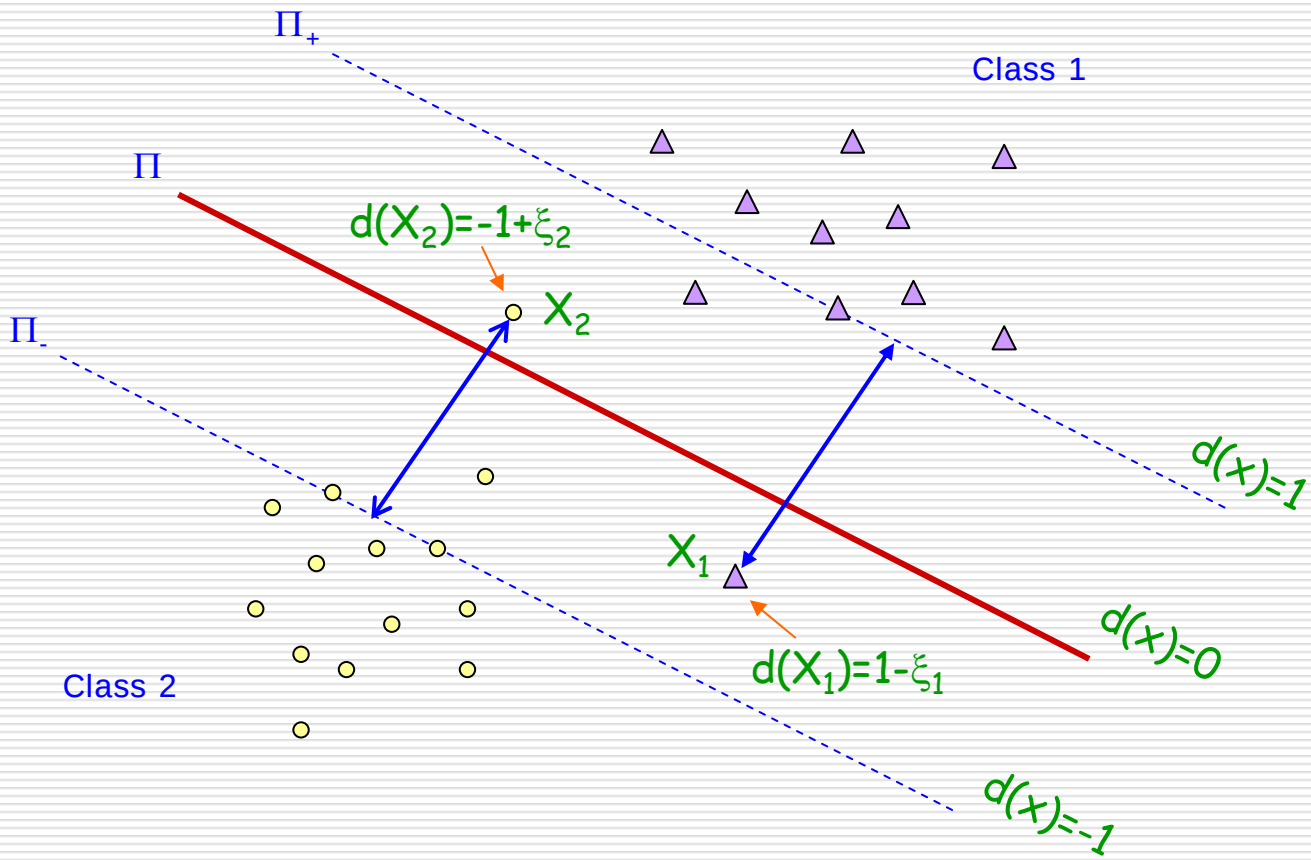


(a) Class 1 data (triangles) and Class 2 data (stars) plotted against a shaded background indicating the magnitude of the hyperplane: large negative values are black; large positive values are white. The class separating boundary (solid line) is shown along with the margins (dotted lines). Four support vectors are visible. (b) Intersection of the hyperplane and the indicator function gives the class separating boundaries and the margins. These are also indicated by the contour lines

Soft Margin Hyperplane Classifier

- For non-linearly separable data classes overlap
- Constraint $d_i(X_i \cdot W + w_0) - 1 \geq 0 \quad \forall i$ cannot be satisfied for all data points
- Optimization procedure will go on increasing the dual Lagrangian to arbitrarily large values
- Solution: Permit the algorithm to misclassify some of the data points albeit with an increased cost
- A *soft margin* is generated within which all the misclassified data lie

Soft Margin Classifier



Slack Variables

- Introduce Q slack variables ξ_i

$$W \cdot X + w_0 \geq +1 - \xi_i \qquad d_i = +1$$

$$W \cdot X + w_0 \leq -1 + \xi_i \qquad d_i = -1$$

- Data point is misclassified if the corresponding slack variable exceeds unity
  - $\Sigma\, \xi_i$ represents an upper bound on the number of misclassifications

Cost Function

- Optimization problem is modified as:
  - Minimize

$$\frac{1}{2}\|W\|^2 + C\left(\sum_{i=1}^{Q}\xi_i\right)^k$$

  - subject to the constraints

$$d_i(X_i \cdot W + w_0) - 1 + \xi_i \geq 0 \quad i = 1, \ldots, Q$$

$$\xi_i \geq 0$$

Notes

- C is a parameter that assigns a penalty to the misclassifications
- Choose $k = 1$ to make the problem quadratic
- Minimizing $||W||^2$ minimizes the VC-dimension while maximizing the margin
- C provides a trade-off between the VC-dimension and the *empirical risk* by changing the relative weights of the two terms in the objective function

Lagrangian in Primal Variables

- For the re-formulated optimization problem

$$L_P(W, w_0, \Lambda, \Xi, \Gamma) = \frac{1}{2}\|W\|^2 + C\left(\sum_{i=1}^{Q} \xi_i\right)$$

$$- \sum_{i=1}^{Q} \lambda_i [d_i(X_i \cdot W + w_0)$$

$$-1 + \xi_i] - \sum_{i=1}^{Q} \gamma_i \xi_i$$

Definitions

- $\Lambda = (\lambda_1,\ldots,\lambda_Q)^T \; \lambda_i \geq 0 \quad \Gamma = (\gamma_1,\ldots,\gamma_Q)^T \; \gamma \geq 0$
- $\Xi = (\xi_1,\ldots,\xi_Q)^T \; \xi \geq 0$
- Reformulate the optimization problem in the dual space
- Invoke Kuhn-Tucker conditions for the optimum

Intermediate Result

- Partial derivatives with respect to the primal variables must vanish at the saddle point.

$$\frac{\partial L_P(W, w_0, \Lambda, \Xi, \Gamma)}{\partial W} = W - \sum_{i=1}^{Q} \lambda_i d_i X_i = 0$$

$$\frac{\partial L_P(W, w_0, \Lambda, \Xi, \Gamma)}{\partial w_0} = \sum_{i=1}^{Q} \lambda_i d_i = \Lambda \cdot D = 0$$

$$\frac{\partial L_P(W, w_0, \Lambda, \Xi, \Gamma)}{\partial \xi_i} = C - \lambda_i - \gamma_i = 0$$

Kuhn-Tucker Complementarity Conditions

- These are $\lambda_i \left[d_i (W \cdot X_i + w_0) - 1 + \xi_i \right] = 0, \quad i = 1, \ldots, Q$
- Which finally yields the dual formulation:

$$L_D(\Lambda) = \sum_{i=1}^{Q} \lambda_i - \frac{1}{2} \sum_{i=1}^{Q} \sum_{j=1}^{Q} \lambda_i \lambda_j d_i d_j (X_i \cdot X_j)$$

- Recast into matrix form

$$L_D(\Lambda) = \Lambda \cdot \mathbf{1} - \frac{1}{2} \Lambda^T \mathbf{H} \Lambda$$

- Hessian matrix has elements $H_{ij} = d_i\, d_j (X_i \cdot Xj)$

Dual Optimization Problem

- Maximize

$$L_D(\Lambda) = \Lambda \cdot \mathbf{1} - \frac{1}{2}\Lambda^T \mathbf{H} \Lambda$$

- Subject to the constraints

$$\Lambda \cdot D = 0$$

$$\Lambda \geq \mathbf{0}$$

$$\Lambda \leq C\mathbf{1}$$

Optimal Weight Vector

- Lagrange dual for the non-separable case is identical to that of the separable case
  - No slack variables or their Lagrange multipliers appear
  - Difference: Lagrange multipliers now have an upper bound: *C*
- Compute the optimal weight vector

$$\hat{W} = \sum_{i=1}^{Q} \hat{\lambda}_i d_i X_i$$

$$= \sum_{k=1}^{n_s} \hat{\lambda}_k d_k X_k$$

Kuhn-Tucker Complementarity

- Application yields

$$\lambda_i \left[d_i (W \cdot X_i + w_0) - 1 + \xi_i \right] = 0, \quad i = 1, \ldots, Q$$
$$\gamma_i \xi_i = 0, \quad i = 1, \ldots, Q$$

- And we know for support vectors, $\lambda_i \geq 0$ and,

$$\lambda_i + \gamma_i = C$$

- Implies that the following constraint is satisfied exactly:

$$d_i (W \cdot X_i + w_0) - 1 + \xi_i = 0$$

Bounded and Unbounded Support Vectors

- Option 1
  - $\xi_i = 0 \Rightarrow$ the support vector is on the margin
  - $\Rightarrow \gamma_i > 0 \Rightarrow \lambda_i < C$
  - For support vectors on the margin $0 < \lambda_i < C$
  - These are called unbounded support vectors
- Option 2
  - $\xi_i > 0 \Rightarrow \gamma_i = 0 \Rightarrow \lambda_i = C$
  - Support vectors between the margins have their Lagrange multipliers equal to the bound C
  - These are called bounded support vectors

Computation of the Bias

- By averaging over unbounded support vectors

$$\hat{w}_0 = \frac{1}{n_{s_u}} \left[\sum_{l=1}^{n_{su}} d_l \left(1 - \sum_{k=1}^{n_{su}} \hat{\lambda}_k H_{kl} \right) \right]$$

- Unknown data point classified using the indicator function

$$y(X) = \text{sign}\left(\sum_{i=1}^{n_s} d_i \hat{\lambda}_i (X \cdot X_i) + \hat{w}_0 \right)$$

MATLAB Code: Non-separable Classes, Linear SVM

```
clear all;
X=[0.5 0.5 % Data points
0.5 1.0
1. 1.5
1.5 0.5
1.5 2.0
2.0 1.0
2.5 2.0
2.5 2.5];
% Corresponding classes
D=[ -1 -1 1 -1 -1 1 1 1]';
q = size(X,1); % size of data set
epsilon = 1e-5; % threshold to check support
        vectors
C = 5; % Control parameter
H = zeros(q,q); % Initialize Hessian matrix
for i = 1:q % Set up the Hessian
for j = 1:q
H(i,j) = D(i)*D(j)*X(i,:)*X(j,:)';
end
end
```

```
f = -ones(q,1); %Vectors of ones
% Parameters for the Optimization problem
vlb = zeros(q,1); % Lower bound of lambdas = 0
vub = C*ones(q,1); % Upper bound C
x0 = zeros(q,1); % Initial point is 0
numconstraints = 1; % Number of equality
        constraints = 1
A = D'; % Set up the equality constraint
b = 0;
%Invoke MATLAB's standard qp function for
        quadratic optimization...
[lambda alpha how] = qp(H, f, A, b, vlb, vub, x0,
        numconstraints);
svindex= find( lambda > epsilon); %Find support
        vectors
%Find unbounded support vectors
usvindex= find( lambda > epsilon & lambda < C -
        epsilon);
ns = length(usvindex);% Number of unbounded
        support vectors
w_0 = (1/ns)*sum(D(usvindex) - ... % Optimal bias
H(usvindex,svindex)*lambda(svindex).*D(usvindex
        ));
...
```

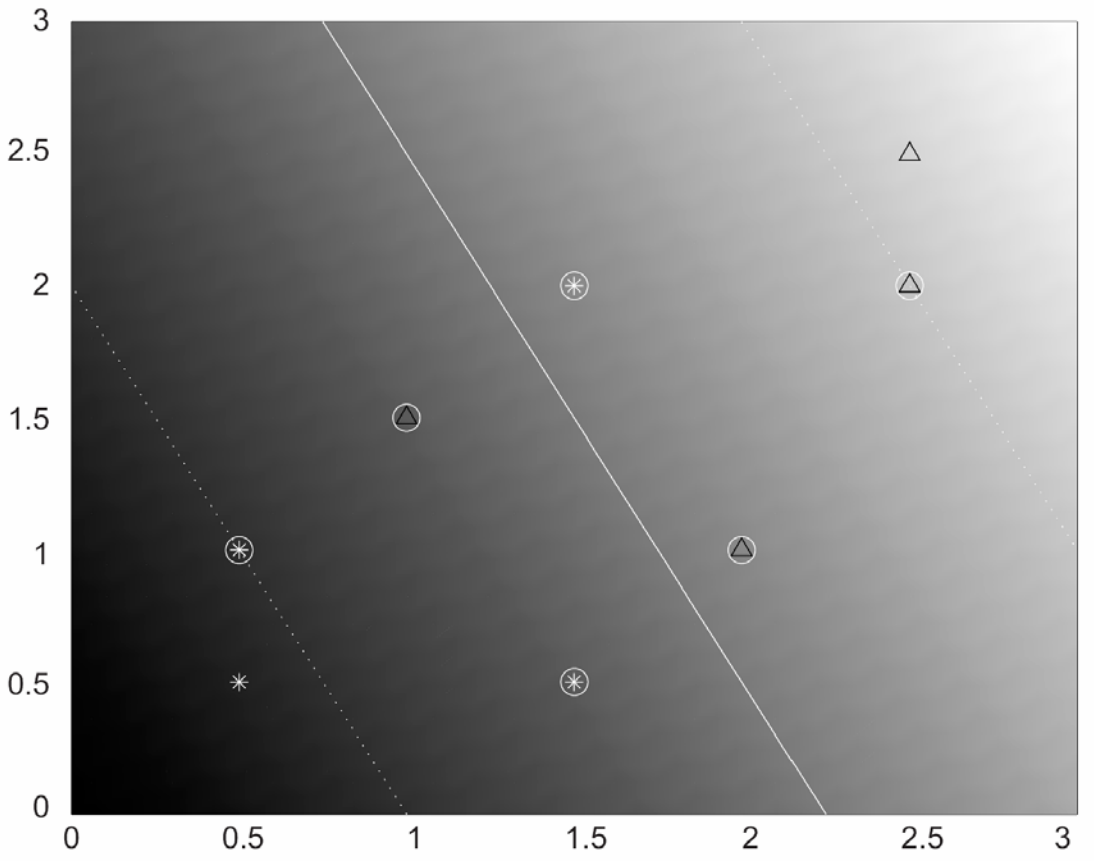
Simulation

- Linearly non-separable data set, Lagrange multiplier values after optimization, and type of support/non-support vector

Data point	Class	Support Vector	λ_i	Type
0.5 0.5	−1	No	0	Non-support
0.5 1.0	−1	Yes	0.4	Unbounded
1.0 1.5	1	Yes	5.0	Bounded
1.5 0.5	−1	Yes	5.0	Bounded
1.5 2.0	−1	Yes	5.0	Bounded
2.0 1.0	1	Yes	5.0	Bounded
2.5 2.0	1	Yes	0.4	Unbounded
2.5 2.5	1	No	0	Non-support

Simulation

- Same data set as in the linearly separable case, except for
  - classes of data points 3 and 5 interchanged to make the problem non-separable
- Number of support vectors is now six
- Two are unbounded, and four being bounded



Towards the Non-linear SVM

- Next:
  - Lay down the method of designing a support vector machine that has a non-linear decision boundary.
- Ideas about the linear SVM are directly extendable to the non-linear SVM using an amazingly simple technique that is based on the notion of an inner product kernel.

Feature Space Maps

- Basic idea:
  - Map the data points using a feature space map into a very high dimensional feature space **H**

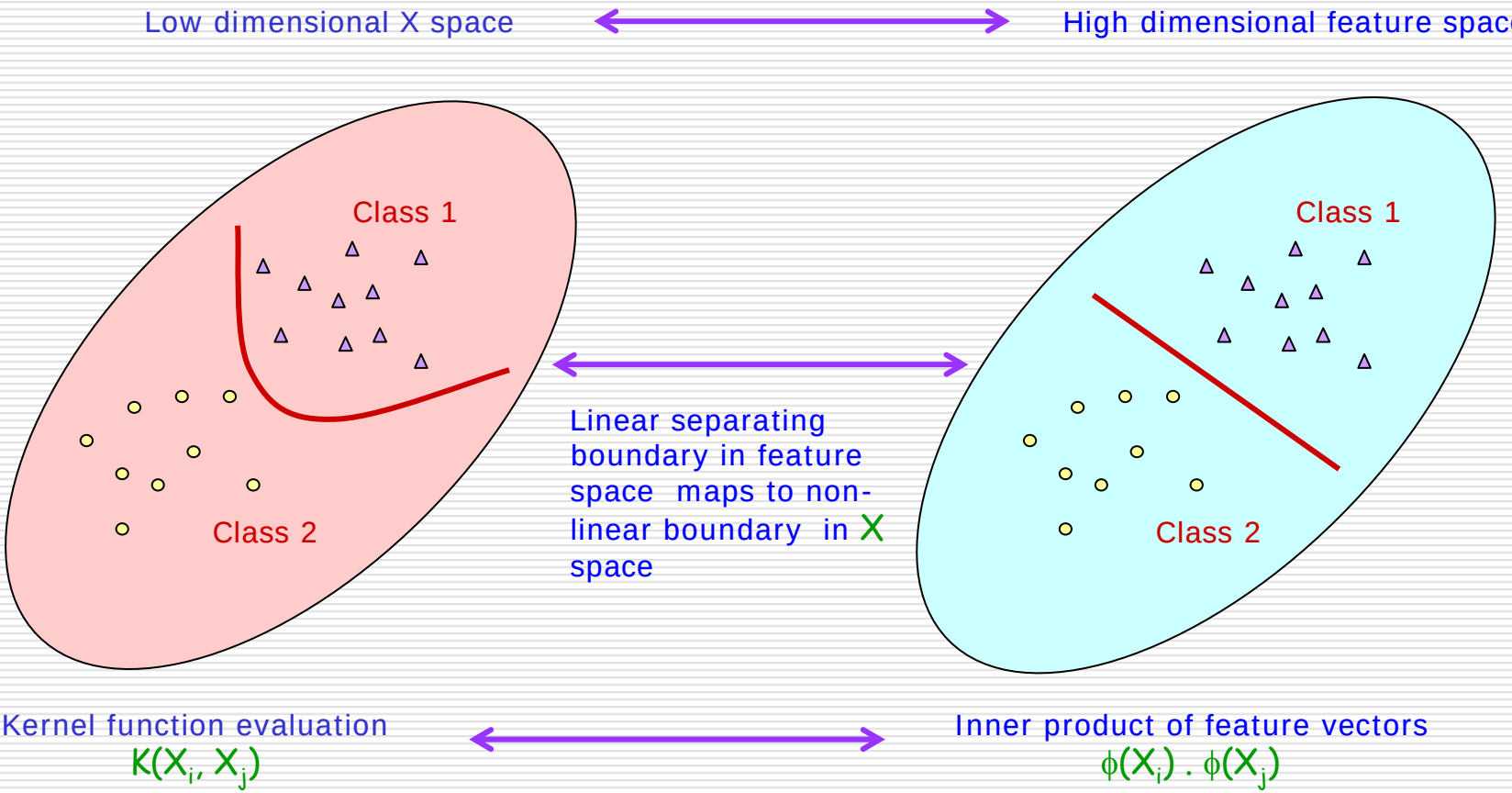
$$X \rightarrow \Phi(X) = (a_1\phi_1(X), a_2\phi_2(X), \ldots, a_n\phi_n(X), \ldots)$$

  - Non-separable data become linearly separable
  - Work on a linear decision boundary in that space

$$y(X) = \text{sign}\left(\sum_{i=1}^{n_s} d_i\hat{\lambda}_i\Phi(X)\cdot\Phi(X_i) + \hat{w}_0\right)$$

  - Map everything back to the original pattern space

Pictorial Representation of Non-linear SVM Design Philosophy



Kernel Function

- Note: X values of the training data appear in the Hessian term H_{ij} only in the form of dot products
- Search for a "Kernel Function" that satisfies

$$K(X_i, X_j) \equiv \Phi(X_i) \cdot \Phi(X_j) = \sum_{l=1}^{\infty} a_l^2 \phi_l(X_i) \phi_l(X_j)$$

- Allows us to use $K(X_i, X_j)$ directly in our equations without knowledge of the map!

Example: Computing Feature Space Inner Products via Kernel Functions

- Assume $X = x$, $\Phi(x) = (1, x, x^2, \ldots, x^m)$
- Choose $a_l = 1$, $l = 1, \ldots, m$, and the decision surface is a polynomial in x
- The inner product $\Phi(x) \cdot \Phi(y) = (1, x, x^2, \ldots, x^m)^T(1, y, y^2, \ldots, y^m) = 1 + xy + (xy)^2 + (xy)^m$ is polynomial of degree m
- Computing in high dimensions can become computationally very expensive…

An Amazing Trick

- Careful choice of the coefficients can change everything!
- Example: Choosing $a_l = \binom{m}{l}$
- Yields

$$\Phi(x) \cdot \Phi(y) = \sum_{l=0}^{m} \binom{m}{l} (xy)^l = (1 + xy)^m$$

- A "kernel" function evaluation equals the inner product, making computation simple.

Example

- $X = (x_1, x_2)^T$
- Input space $\Re^2$
- Feature space $\Re^6$:

$$\Phi(X) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- Admits the kernel function

$$K(X, Y) = \Phi(X) \cdot \Phi(Y) = (1 + XY)^2$$

Non-Linear SVM Discriminant with Polynomial Kernel Functions

- Using kernel functions for inner products the SVM discriminant becomes

$$d(X) = \text{sign}\left(\sum_{i=1}^{n_s} d_i \hat{\lambda}_i K(X, X_i) + \hat{w}_0\right)$$

- This is a non-linear decision boundary in input space generated from a linear superposition of n_s kernel function terms
- This requires the identification of suitable kernel functions that can be employed

Mercer's Condition

- There exists a mapping Φ and an expansion of a symmetric kernel function

$$K(X, Y) = \sum_i \phi_i(X)\phi_i(Y)$$

iff

$$\iint K(X, Y)\, g(X)\, g(Y)\, dX\, dY \geq 0$$

such that

$$\int g^2(X)\, dX < \infty$$

Inner Product Kernels (1)

- Polynomial discriminant functions

$$d(X) = \text{sign}\left(\sum_{i=1}^{n_s} d_i \hat{\lambda}_i K(X, X_i) + \hat{w}_0 \right)$$

- admit the kernel function

$$K(X, Y) = (1 + X \cdot Y)^m$$

Inner Product Kernels (2)

- Radial basis indicator functions of the form

$$y(X) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i \exp\left[-\frac{\|X - X_i\|^2}{\sigma^2}\right]\right)$$

- admit the kernel function

$$K(X, Y) = \exp\left[-\frac{\|X - Y\|^2}{\sigma^2}\right]$$

Inner Product Kernels (3)

- Neural network indicator functions of the form

$$y(X) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i \tanh[a(X \cdot W) + b] + w_0\right)$$

- admit the kernel function

$$K(X, Y) = \tanh(a(X \cdot Y) + b)$$

Operational Summary of SVM Learning Algorithm

Given	A training set $\mathcal{T}$ comprising vectors $X_k \in \mathbb{R}^n$ and desired output vectors $d_k \in \{-1, +1\}$
Initialize	$\hookrightarrow$ Choose a kernel function $K(\cdot, \cdot)$ $\hookrightarrow$ Set up the Hessian matrix: $H_{ij} = d_i d_j K(X_i, X_j)$ $\hookrightarrow$ Set C

Operational Summary of SVM Learning Algorithm

Maximize $\quad \Lambda^T \mathbf{1} - \frac{1}{2}\Lambda^T \mathbf{H} \Lambda$ subject to the constraints:

$\Lambda \cdot D = 0$

$\Lambda \geq \mathbf{0}$

$\Lambda \leq C\mathbf{1}$

using any quadratic program optimizer.
Obtain optimized Lagrange multipliers.

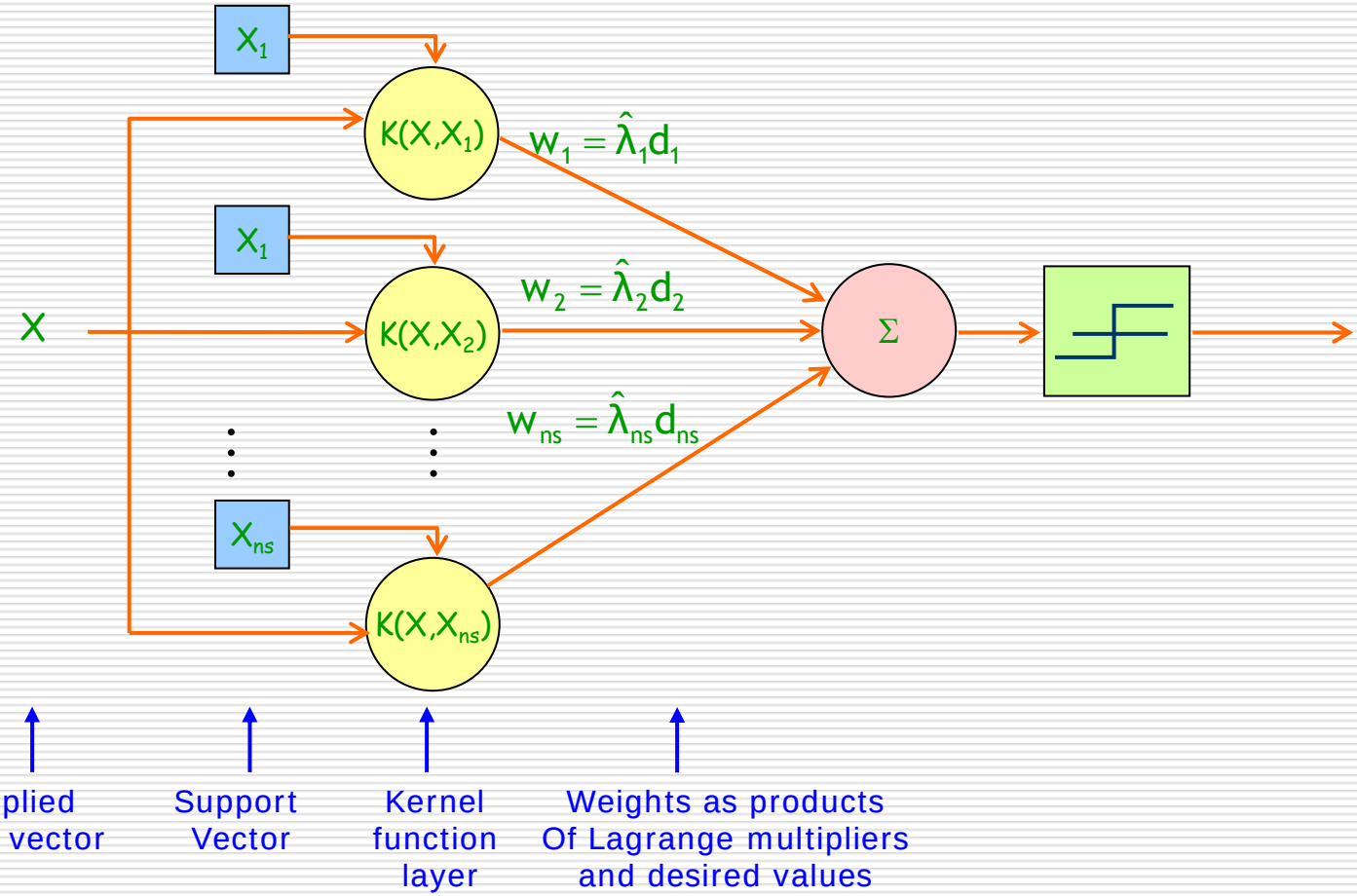
Predict Class $\quad$ Given any test input X, set the class:

$$y(X) = \text{sign}\left(\sum_{i=1}^{n_s} d_i \hat{\lambda}_i K(X, X_i) + \hat{w}_0 \right)$$

MATLAB Code Segment for Hessian Computation

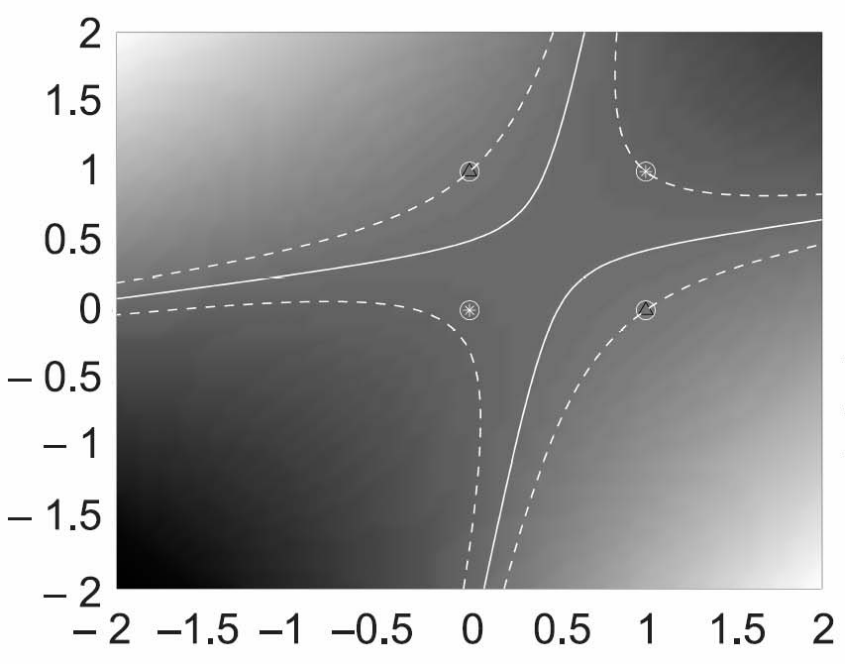
```
% All code same as for linear SVM non-separable data
% Code snippet shown for polynomial kernel

ord = 2; % Order of polynomial kernel
H = zeros(q,q); % Initialize Hessian matrix

for i = 1:q % Set up the Hessian
  for j = 1:q
    H(i,j) = D(i)*D(j)*(X(i,:)*X(j,:)' + 1)^ ord;
  end
end
```

SVM Computations Portrayed as a Feedforward Neural Network!

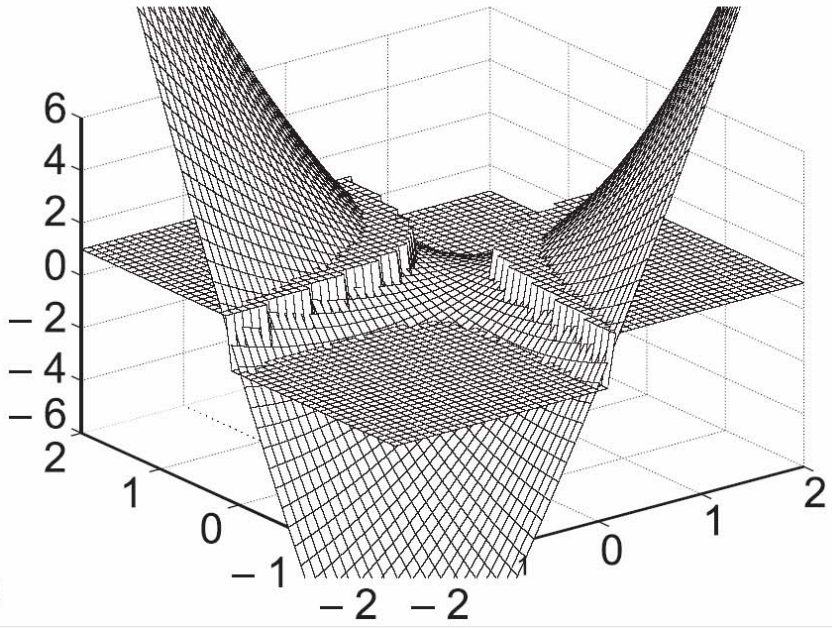


XOR Simulation

- XOR Classification, $C = 3$, Polynomial kernel $(X_i^T X_j + 1)^2$



Margins and class separating boundary using a second order polynomial kernel



Intersection of the signum indicator function and non-linear polynomial surface

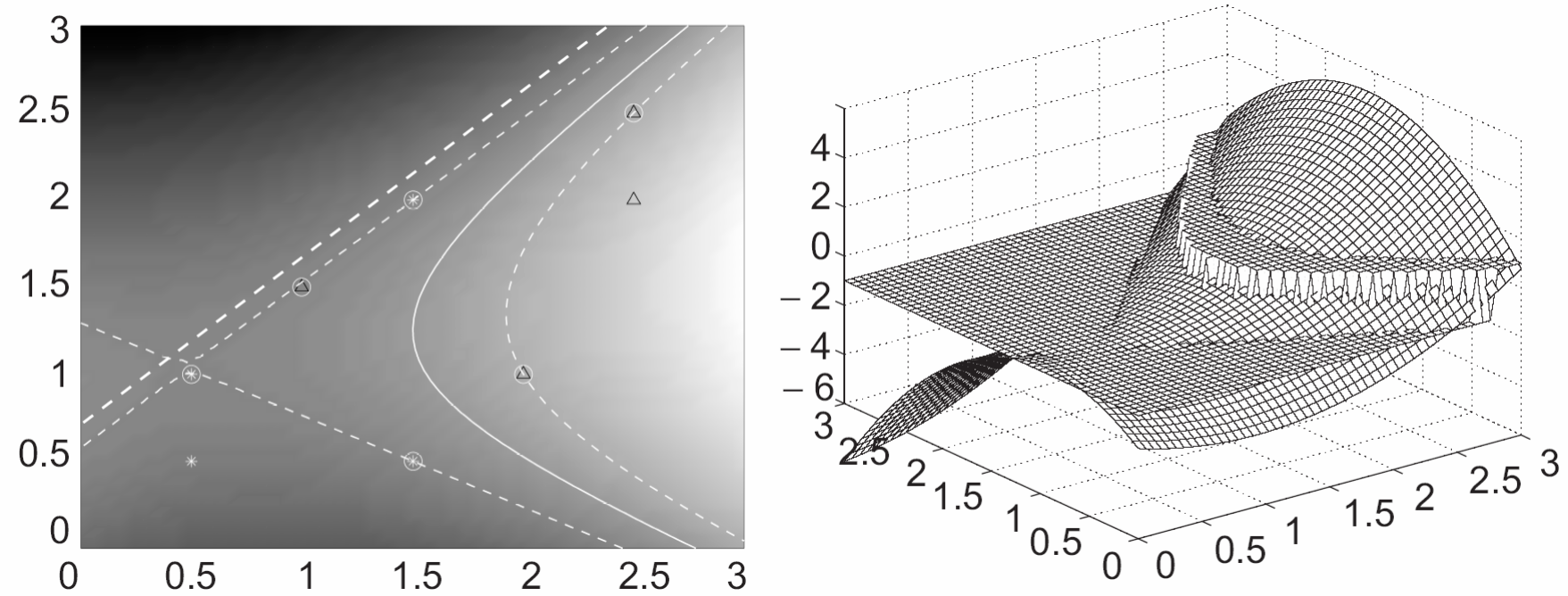
XOR Simulation

- Data specification for the XOR classification problem with Lagrange multiplier values after optimization

Data point	Class	Support Vector	λ_i	Type
0 0	−1	Yes	3.0	Bounded
0 1	1	Yes	2.42	Unbounded
1 0	1	Yes	2.42	Unbounded
1 1	−1	Yes	1.85	Unbounded

Non-linearly Separable Data Scatter Simulation

- $C = 10$: Stress on large margin sacrifice classification accuracy



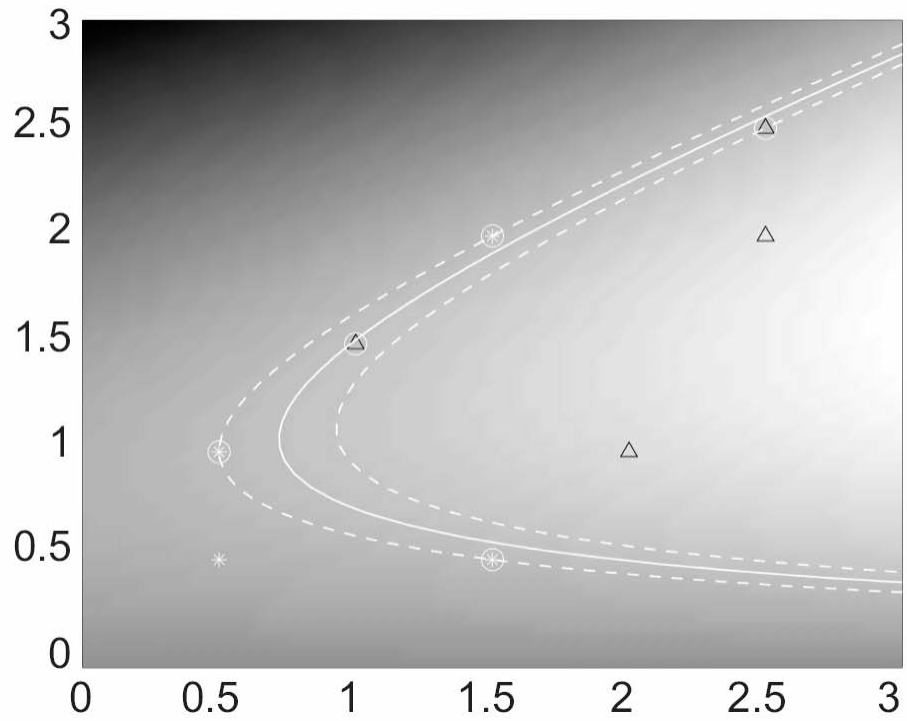
Non-linearly Separable Data Scatter Simulation

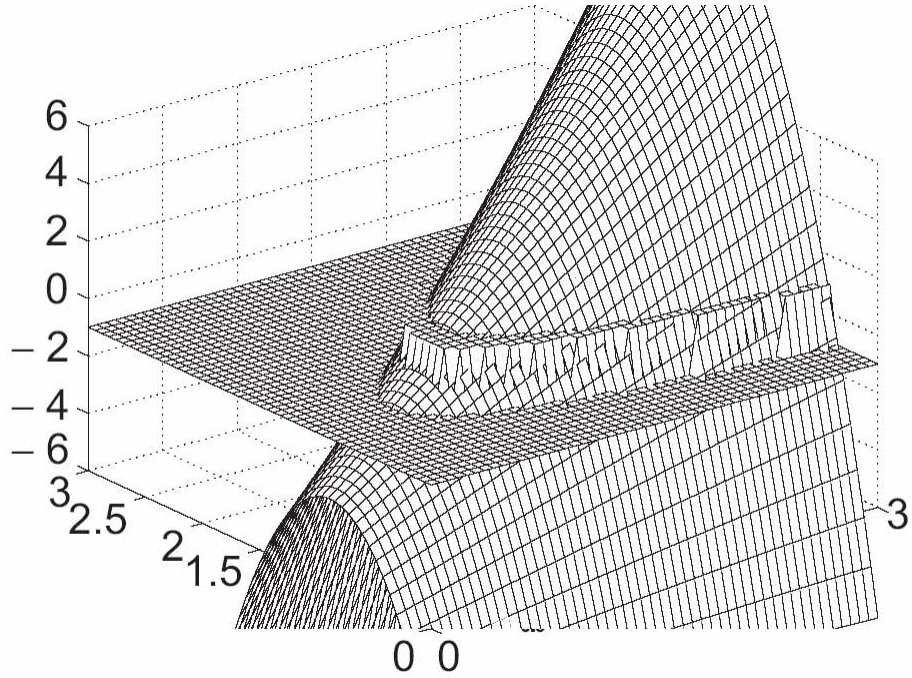
- $C = 10$

Data point	Class	Support Vector	λ_i	Type
		$C = 10$		
0.5 0.5	−1	No	0.0	Non-support
0.5 1.0	−1	Yes	1.1	Unbounded
1.0 1.5	1	Yes	10.0	Bounded
1.5 0.5	−1	Yes	6.49	Unbounded
1.5 2.0	−1	Yes	7.76	Unbounded
2.0 1.0	1	Yes	4.57	Unbounded
2.5 2.0	1	Yes	0.0	Non-support
2.5 2.5	1	No	0.79	Unbounded

Non-linearly Separable Data Scatter Simulation

- C= 150: Small margin, high classification accuracy





Non-linearly Separable Data Scatter Simulation

- $C = 150$

Data point	Class	Support Vector	λ_i	Type
		$C = 150$		
0.5 0.5	−1	No	0.0	Non-support
0.5 1.0	−1	Yes	56.4	Unbounded
1.0 1.5	1	Yes	150.0	Bounded
1.5 0.5	−1	Yes	11.32	Unbounded
1.5 2.0	−1	Yes	100.89	Unbounded
2.0 1.0	1	No	0.0	Non-support
2.5 2.0	1	No	0.0	Non-support
2.5 2.5	1	Yes	18.64	Unbounded

Support Vector Machines for Regression

- The outputs can take on real values, and thus the training data now take on the form $T = \{X_k, d_k\}$ $X_k \in \Re^n$, $d_k \in \Re$
- Find the functional that models the dependence of d on X in a probabilistic sense
- Support vector machines for regression approximate functions of the form

$$f(X, W) = \sum_{i=1}^{m} w_i \phi_i(X) + w_0 = W^T \Phi(X) + w_0$$

High dimensional feature vector

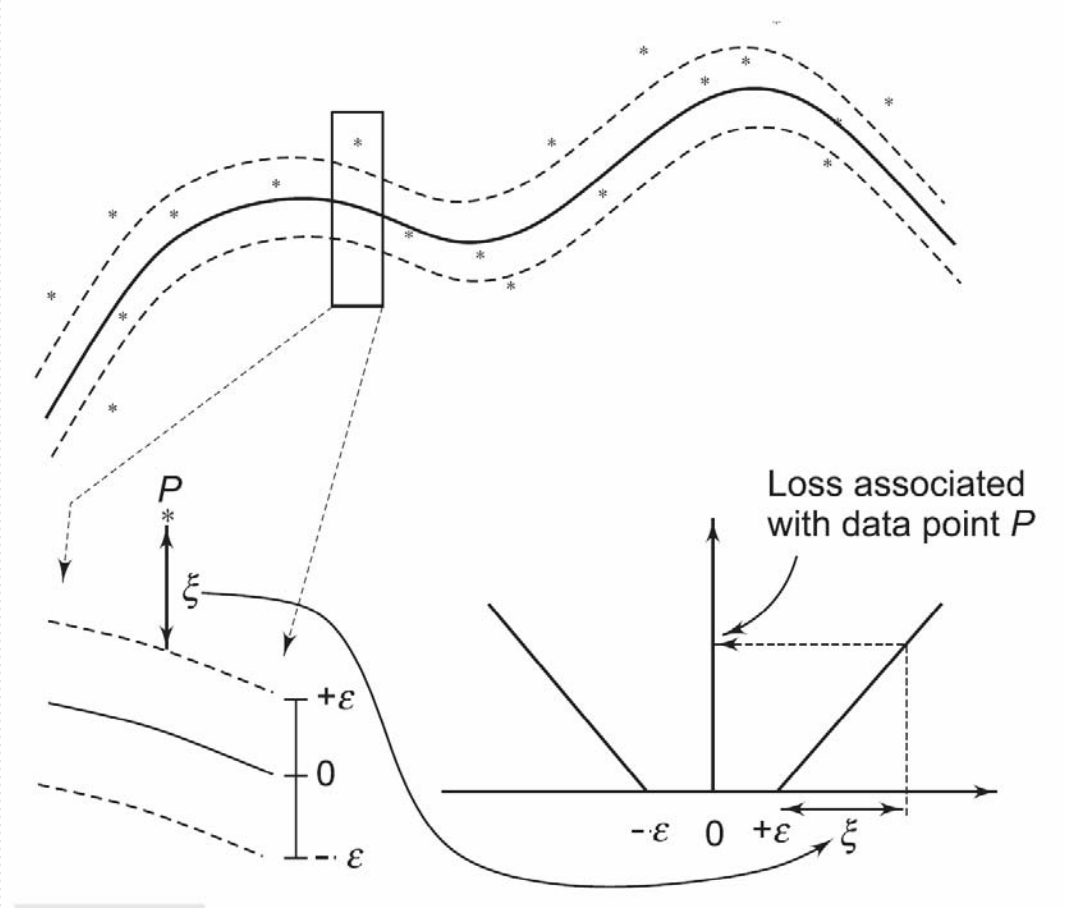
Measure of the Approximation Error

- Vapnik introduced a more general error function called the ε-insensitive loss function

$$L_\epsilon(d, f(X, W)) = \begin{cases} 0 & \text{if } |d - f(X, W)| \leq \epsilon \\ |d - f(X, W)| - \epsilon & \text{otherwise} \end{cases}$$

- No loss if error range within ±ε
- Loss equal to linear error - ε if error greater than ±ε

ε-Insensitive Loss Function



Minimization Problem

- Assume the empirical risk

$$R_e[f] = \frac{1}{Q} \sum_{k=1}^{Q} L_\epsilon(d_k, f(X_k, W))$$

- subject to

$$\|W\|^2 \leq A$$

- Introduce two sets of slack variables ξ_i, ξ_i' for each of Q input patterns

Cost Functional

- Define

$$
\begin{aligned}
d_i - W^T \Phi(X) - w_0 - \epsilon &\le \xi_i \quad i = 1, \ldots, Q \\
W^T \Phi(X) + w_0 - d_i - \epsilon &\le \xi_i' \quad i = 1, \ldots, Q \\
\xi_i &\ge 0 \quad i = 1, \ldots, Q \\
\xi_i' &\ge 0 \quad i = 1, \ldots, Q
\end{aligned}
$$

- The empirical risk minimization problem is then equivalent to minimizing the functional

$$
\frac{1}{2} \| W \|^2 + C \left(\sum_{i=1}^{Q} \xi_i + \sum_{i=1}^{Q} \xi_i' \right)
$$

Primal Variable Lagrangian

- Slack variables $\Xi = (\xi_1, \ldots, \xi_Q)^T$ $\Xi = (\xi_1', \ldots, \xi_Q')^T$
- Lagrange multipliers $\Gamma = (\gamma_1, \ldots \gamma_Q)^T$ $\Lambda = (\lambda_1, \ldots \lambda_Q)^T$ $\Gamma' = (\gamma_1', \ldots \gamma_Q')^T$ $\Lambda = (\lambda_1', \ldots \lambda_Q')^T$

$$L_P(W, w_0, \Xi, \Xi', \Lambda, \Lambda', \Gamma, \Gamma') = \frac{1}{2} W^T W + C \sum_{i=1}^{Q} (\xi_i + \xi_i')$$

$$- \sum_{i=1}^{Q} \lambda_i (W^T \Phi(X_i) + w_0 - d_i + \epsilon + \xi_i)$$

$$- \sum_{i=1}^{Q} \lambda_i' (d_i - W^T \Phi(X_i) - w_0 + \epsilon + \xi_i') - \sum_{i=1}^{Q} (\gamma_i \xi_i + \gamma_i' \xi_i')$$

Saddle Point Behaviour

$$\frac{\partial(L_P(W, w_0, \Xi, \Xi', \Lambda, \Lambda', \Gamma, \Gamma'))}{\partial W} = 0 \implies W = \sum_{i=1}^{Q}(\lambda_i - \lambda_i')\Phi(X_i)$$

$$\frac{\partial(L_P(W, w_0, \Xi, \Xi', \Lambda, \Lambda', \Gamma, \Gamma'))}{\partial w_0} = 0 \implies \sum_{i=1}^{Q}\lambda_i - \lambda_i' = 0 \quad 0 = 0$$

$$\frac{\partial(L_P(W, w_0, \Xi, \Xi', \Lambda, \Lambda', \Gamma, \Gamma'))}{\partial \xi_i} = 0 \implies \gamma_i = C - \lambda_i$$

$$\frac{\partial(L_P(W, w_0, \Xi, \Xi', \Lambda, \Lambda', \Gamma, \Gamma'))}{\partial \xi_i'} = 0 \implies \gamma_i' = C - \lambda_i'.$$

Simplified Dual Form

- Substitution results in the dual form

$$L_D(\Lambda, \Lambda') = \left(\sum_{i=1}^{Q} \lambda_i (d_i - \epsilon) - \lambda_i' (d_i + \epsilon) \right)$$

$$-\frac{1}{2} \sum_{i=1}^{Q} \sum_{j=1}^{Q} (\lambda_i \lambda_j - \lambda_i \lambda_j' - \lambda_i' \lambda_j + \lambda_i' \lambda_j') K(X_i, X_j)$$

Dual Lagrangian in Vector Form

- Maximize

$$L_D(\Lambda, \Lambda') = (\Lambda^T \Lambda'^T) \begin{pmatrix} D - \epsilon \mathbf{1} \\ -D - \epsilon \mathbf{1} \end{pmatrix} - \frac{1}{2} (\Lambda^T \Lambda'^T) \begin{pmatrix} \mathbf{H} & -\mathbf{H} \\ -\mathbf{H} & \mathbf{H} \end{pmatrix} \begin{pmatrix} \Lambda \\ \Lambda' \end{pmatrix}$$

- subject to the constraints

$$\sum_{i=1}^{Q} \lambda_i - \lambda_i' = 0$$

$$0 \le \lambda_i \le C \quad i = 1, \ldots, Q$$

$$0 \le \lambda_i' \le C \quad i = 1, \ldots, Q$$

$H_{ij} = K(X_i, X_j)$
$D = (d_1, \ldots, d_Q)^T$
$\mathbf{1} = (1, \ldots, 1)^T$

Optimal Weight Vector

- For n_s support vectors

$$\hat{W} = \sum_{i=1}^{Q} (\hat{\lambda}_i - \hat{\lambda}'_i)\Phi(X_i)$$

$$= \sum_{k=1}^{n_s} (\hat{\lambda}_k - \hat{\lambda}'_k)\Phi(X_k)$$

Computing the Optimal Bias

- Invoke Kuhn-Tucker complementarity

$$w_0 = d_i - W^T \Phi(X_i) - \epsilon \ \text{sign}(\beta_i)$$

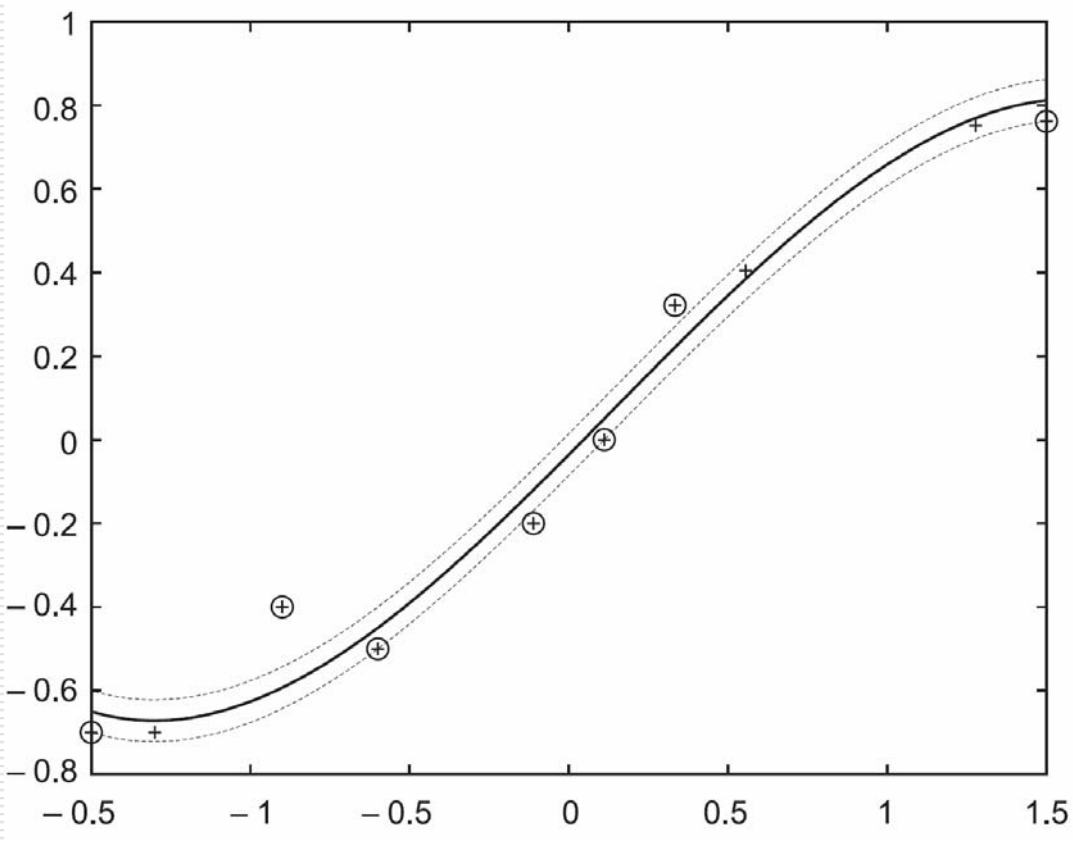
- Substitution of the optimal weight vector yields

$$\hat{w}_0 = \frac{1}{n_{s_u}} \sum_{i=1}^{n_{s_u}} \left(d_i - \sum_{k=1}^{n_s} \beta_k \Phi^T(X_k) \Phi(X_i) - \epsilon \ \text{sign}(\beta_i) \right)$$

$$= \frac{1}{n_{s_u}} \sum_{i=1}^{n_{s_u}} \left(d_i - \sum_{k=1}^{n_s} \beta_k H_{ki} - \epsilon \ \text{sign}(\beta_i) \right)$$

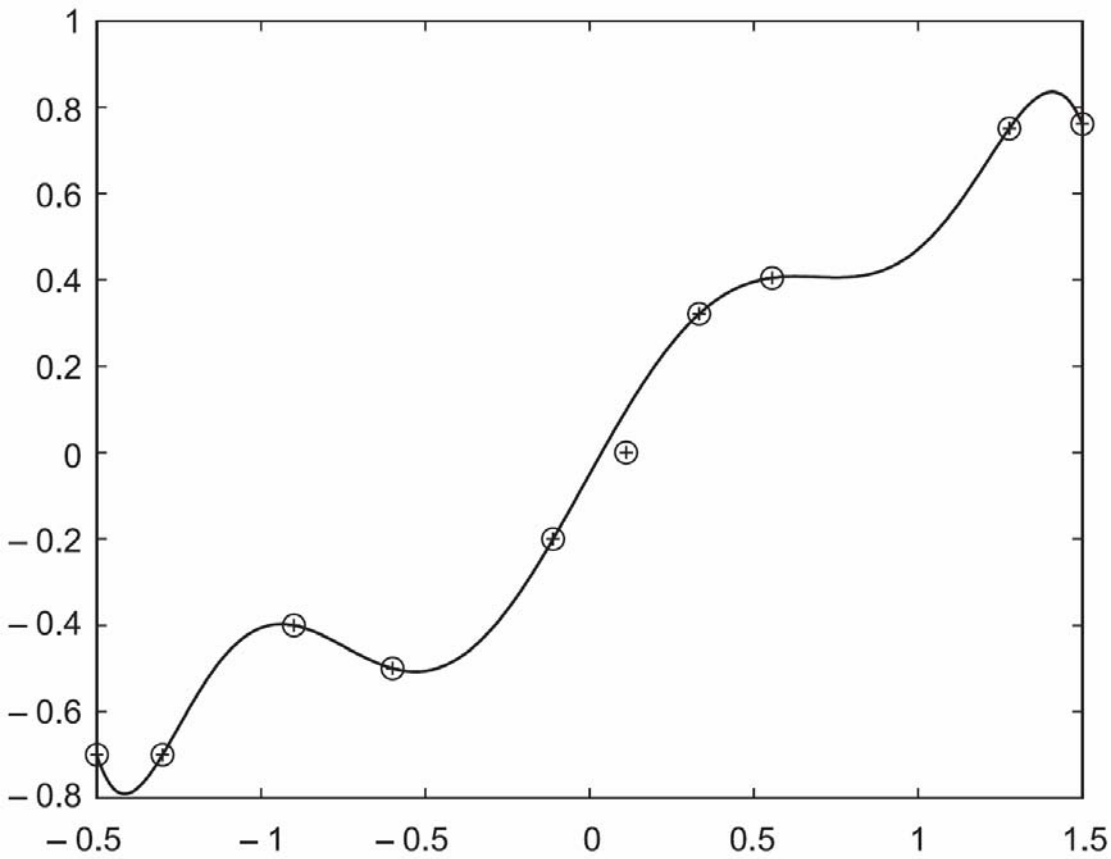
Simulation

- Regression on noisy hyperbolic tangent data scatter
- Third order polynomial kernel
- ε=0.05, C=10



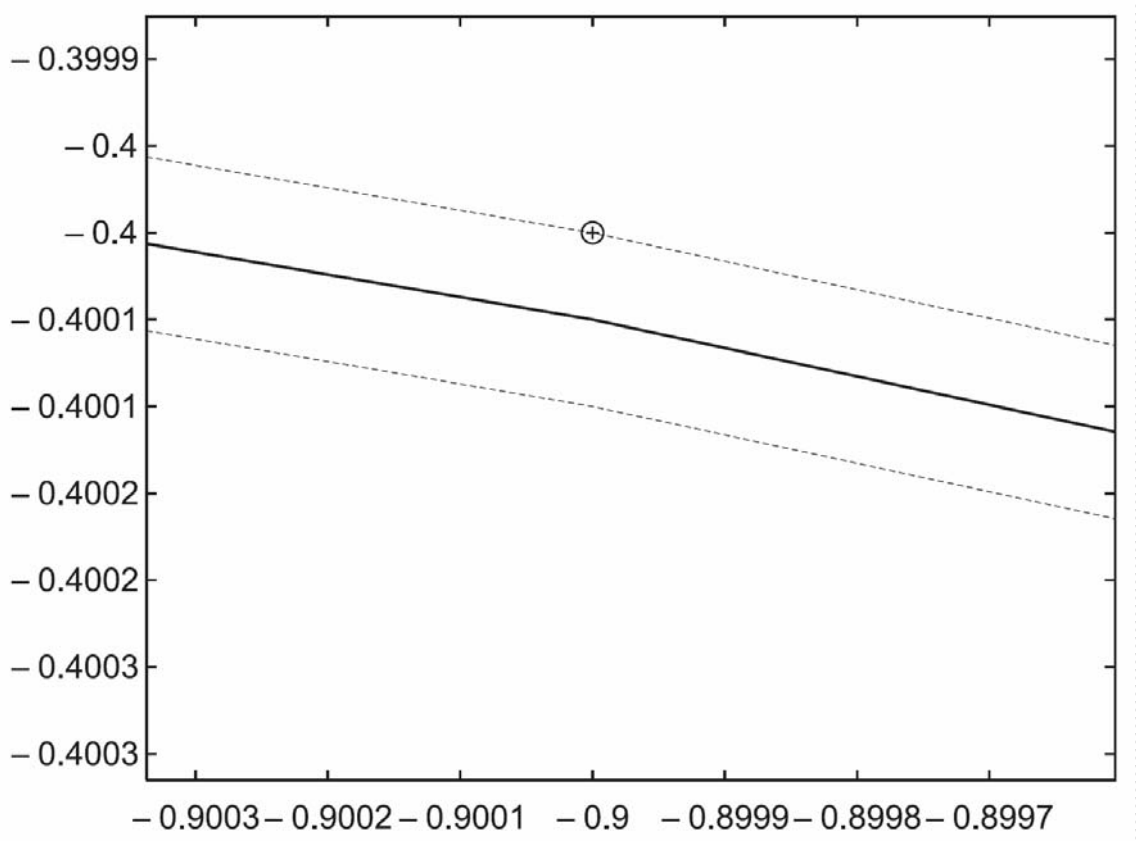
Simulation

- Regression on noisy hyperbolic tangent data scatter
- Eight order polynomial kernel
- ε=0.00005, C=10



Simulation: Zoom Plot

- Eight order polynomial kernel
- ε=0.00005, C=10
- Shows the fine margin, and the support vector



Radial Basis Function Networks

Neural Networks: A Classroom Approach

Satish Kumar
Department of Physics & Computer Science
Dayalbagh Educational Institute (Deemed University)

Radial Basis Function Networks

- Feedforward neural networks
  - compute activations at the hidden neurons using an exponential of a [Euclidean] distance measure between the input vector and a prototype vector that characterizes the signal function at a hidden neuron.
- Originally introduced into the literature for the purpose of interpolation of data points on a finite training set

Interpolation Problem

- Given $T = \{X_k, d_k\}$ $X_k \in \Re^n$, $d_k \in \Re$
- Solving the interpolation problem means finding the map $f(X_k) = d_k$, $k = 1,\ldots,Q$ (target points are scalars for simplicity of exposition)
- RBFN assumes a set of exactly Q non-linear basis functions $\varphi(||X - X_i||)$
- Map is generated using a superposition of these

$$f(X) = \sum_{i=1}^{Q} w_i \phi(\|X - X_i\|)$$

Exact Interpolation Equation

- Interpolation conditions

$$\sum_{i=1}^{Q} w_i \phi(\|X_k - X_i\|) = d_k \quad k = 1, \ldots, Q$$

- Matrix definitions

$$D = (d_1, \ldots, d_Q)^T$$

$$W = (w_1, \ldots, w_Q)^T$$

$$\Phi = \begin{pmatrix} \phi(\|X_1 - X_1\|) & \cdots & \phi(\|X_1 - X_Q\|) \\ \vdots & \ddots & \vdots \\ \phi(\|X_Q - X_1\|) & \cdots & \phi(\|X_Q - X_Q\|) \end{pmatrix}$$

- Yields a compact matrix equation

$$D = \Phi^T W = \Phi W$$

Michelli Functions

- Gaussian functions

$$\phi(x) = \exp\left(-\frac{(x-c)^2}{2\sigma^2}\right), \quad \sigma > 0; x, c \in \mathbb{R}$$

- Multiquadrics

$$\phi(x) = (x^2 + c^2)^{1/2}, \quad c > 0; x, c \in \mathbb{R}$$

- Inverse multiquadrics

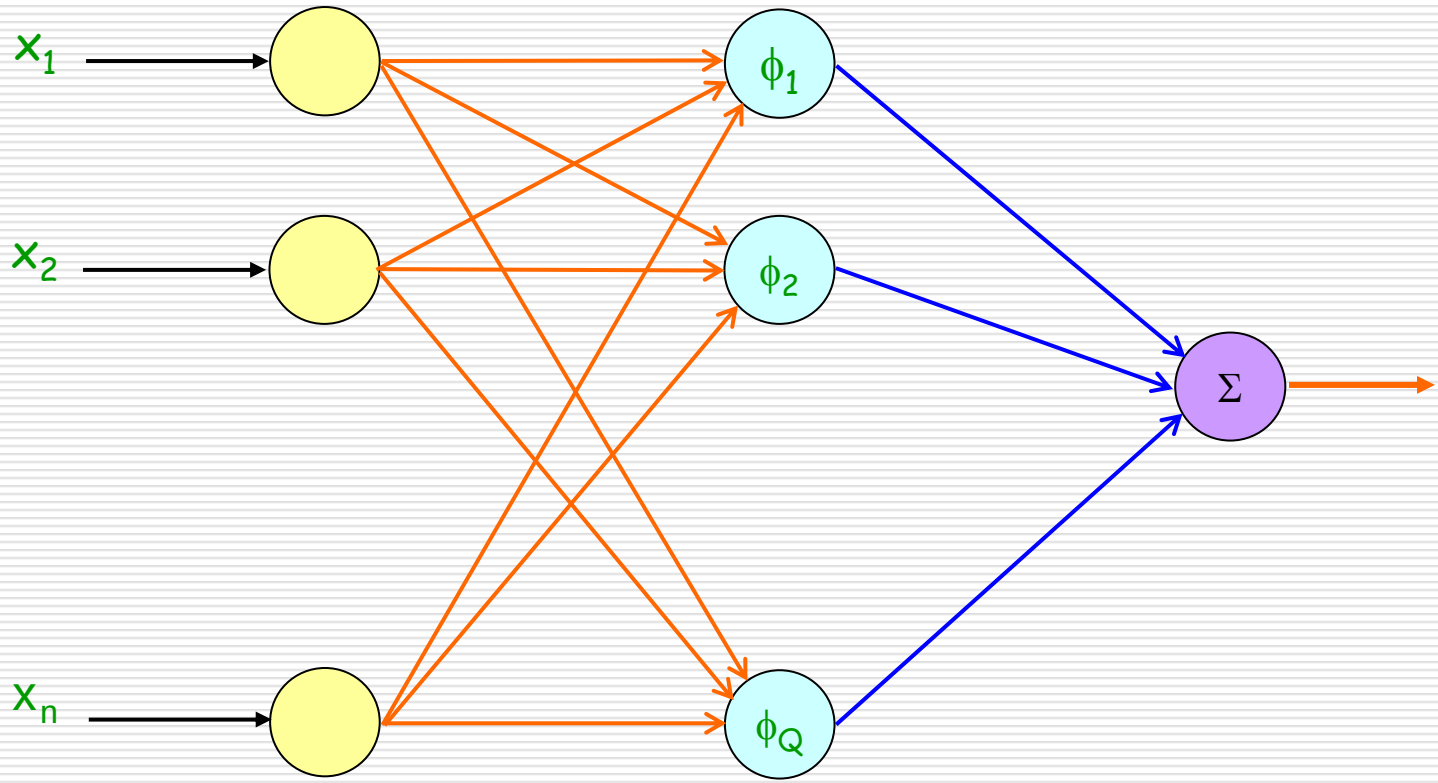
$$\phi(x) = \frac{1}{(x^2 + c^2)^{1/2}}, \quad c > 0, x, c \in \mathbb{R}$$

Solving the Interpolation Problem

- Choosing Φ correctly ensures invertibility: $W = \Phi^{-1} D$
- Solution is a set of weights such that the interpolating surface generated passes through exactly every data point
- Common form of Φ is the localized Gaussian basis function with center μ and spread σ

$$\phi(X) = \exp\left(-\frac{\|X-\mu\|^2}{2\sigma^2}\right)$$

Radial Basis Function Network

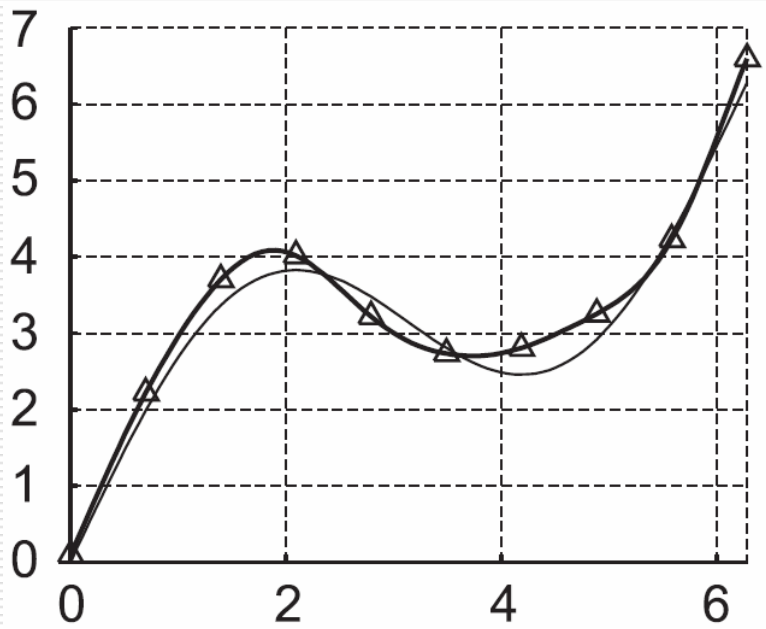


Interpolation Example

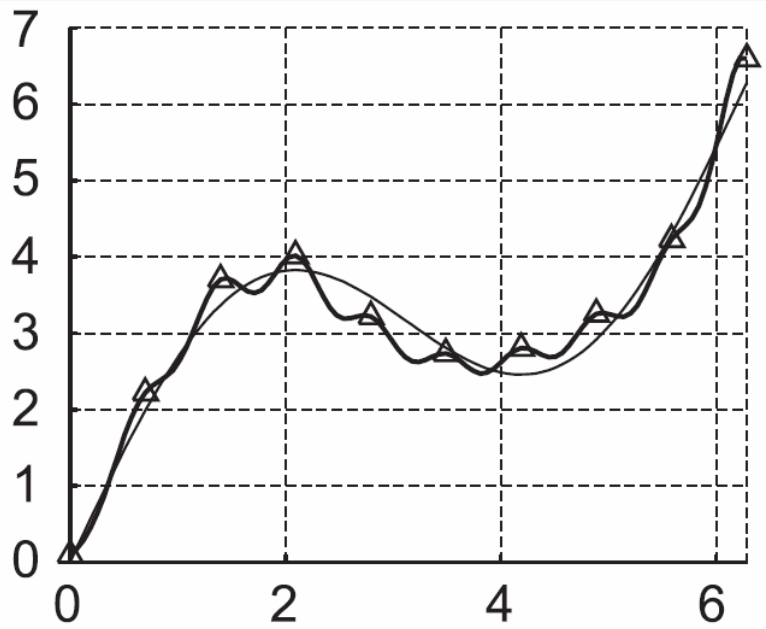
- Assume a noisy data scatter of $Q = 10$ data points
- Generator: $2 \sin(x) + x$
- In the graphs that follow:
  - data scatter (indicated by small triangles) is shown along the generating function (the fine line)
  - interpolation shown by the thick line

Interpolant: Smoothness-Accuracy

- $\sigma = 1$

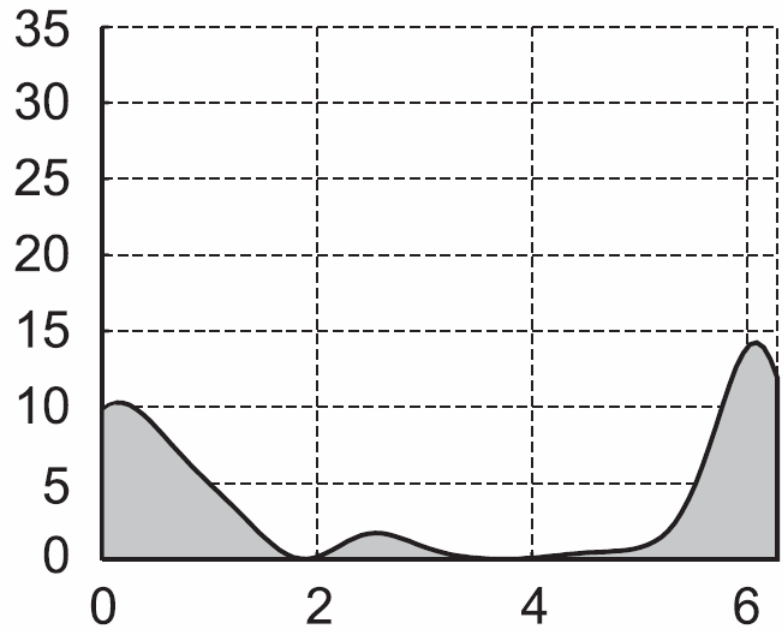


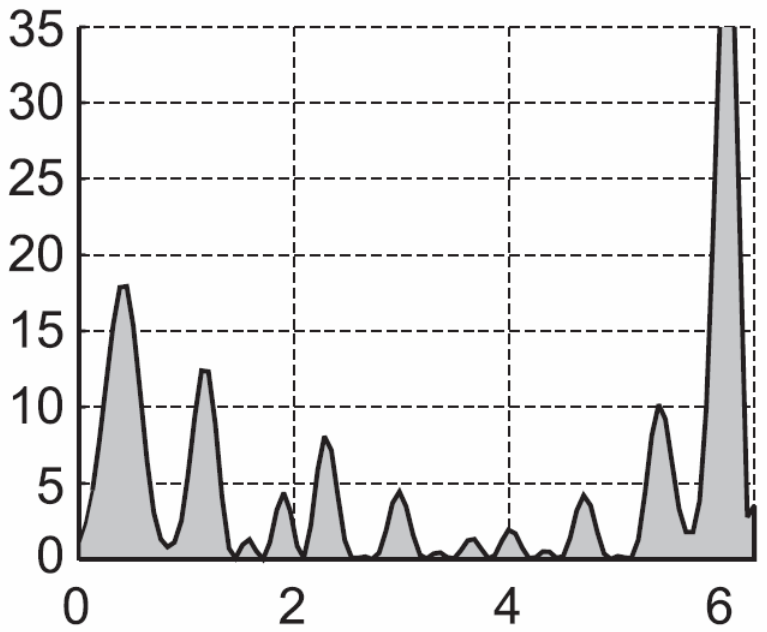
- $\sigma = 0.3$



Derivative Square Function

- $\sigma = 1$
- $\sigma = 0.3$





Notes

- Making the spread factor smaller
  - makes the function increasingly non-smooth
  - being able to achieve a 100 per cent mapping accuracy on the ten data points rather than smoothness of the interpolation
- Quantify the oscillatory behaviour of the interpolants by considering their derivatives
  - Taking the derivative of the function
  - Square it (to make it positive everywhere)
  - Measure the areas under the curves
- Provides a nice measure of the non-smoothness—the greater the area, the more non-smooth the function!

Problems…

- Oscillatory behaviour is highly undesirable for proper generalization
- Better generalization is achievable with smoother functions which are fitted to noisy data
- Number of basis functions in the expansion is equal to the number of data points!
  - Not possible to have for real world data sets can be extremely large
  - Computational and storage requirements for can explode very quickly

The RBFN Solution

- Choose the number of basis functions to be some number $q < Q$
- No longer restrict the centers of the basis functions to be fixed to the data point values.
  - Now made trainable parameters of the model
- Spreads of each of the basis functions is permitted to be different and trainable.
  - Learning can be done either by supervised or unsupervised techniques
- A bias is included in the final linear superposition

Interpolation with Fewer than Q Basis Functions

- Assume centers and spreads of the basis functions are optimized and fixed
- Proceed to determine the hidden-output neuron weights using the procedure adopted in the interpolation case

Solving the Problem in a Least Squares Sense

- To formalize this, consider interpolating a set of data points with a number $q < Q$
- Then,

$$f(X) = \sum_{i=1}^{q} w_i \phi(\| X - X_i \|)$$

- Introduce the notion of error since the interpolation is not exact

$$\mathcal{E} = \frac{1}{2} \sum_{k=1}^{Q} \left(d_k - \sum_{i=1}^{q} w_i \phi(\| X_k - X_i \|) \right)^2$$

Compute the Optimal Weights

- Differentiating w.r.t. w_i and setting it equal to zero

$$\sum_{k=1}^{Q} \phi_{ki} \left(\sum_{l=1}^{q} w_l \phi_{kl} \right) = \sum_{k=1}^{Q} d_k \phi_{ki}, \quad i = 1, \ldots, q$$

- Then

$$\left(\Phi^T \Phi\right) W = \Phi^T D$$

$$\Phi = \begin{pmatrix} \phi_{11} & \cdots & \phi_{1q} \\ \phi_{21} & \cdots & \phi_{2q} \\ \vdots & \ddots & \vdots \\ \phi_{Q1} & \cdots & \phi_{Qq} \end{pmatrix}$$

Pseudo-Inverse

- This yields

$$W = \left(\Phi^T \Phi\right)^{-1} \Phi^T D$$
$$= \Phi^{\dagger} D$$

Equation solved using singular value decomposition

- where

$$\Phi^{\dagger} \equiv \left(\Phi^T \Phi\right)^{-1} \Phi^T$$

Pseudo-inverse (is not square: q × Q)

Two Observations

- Straightforward to include a bias term w_0 into the approximation equation

$$f(X) = \sum_{i=1}^{q} w_i \phi(\| X - X_i \|) + w_0, \qquad q < Q$$

- Basis function is generally chosen to be the Gaussian

$$\phi(\| X - X_i \|) = \exp\left(-\frac{\| X - X_i \|^2}{2\sigma_i^2} \right)$$

Generalizing Further

- RBFs can be generalized to include arbitrary covariance matrices K_i

$$\phi(\|X - X_i\|) = \exp\left(-\frac{1}{2}(X - X_i)^T \mathbf{K}_i^{-1}(X - X_i)\right)$$

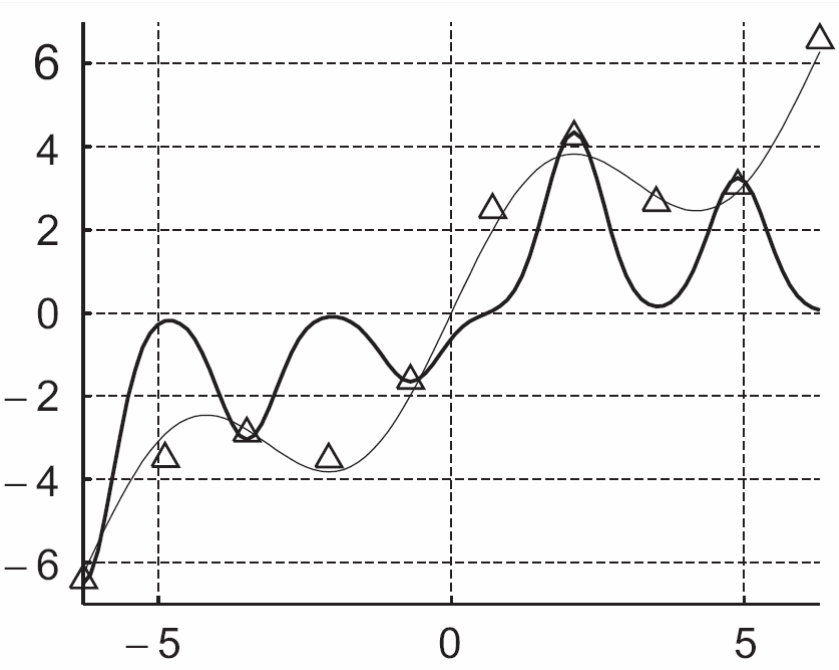
- Universal approximator
- RBFNs have the *best approximation* property
  - The set of approximating functions that RBFNs are capable of generating, there is one function that has the minimum approximation error *for any given function* which has to be approximated

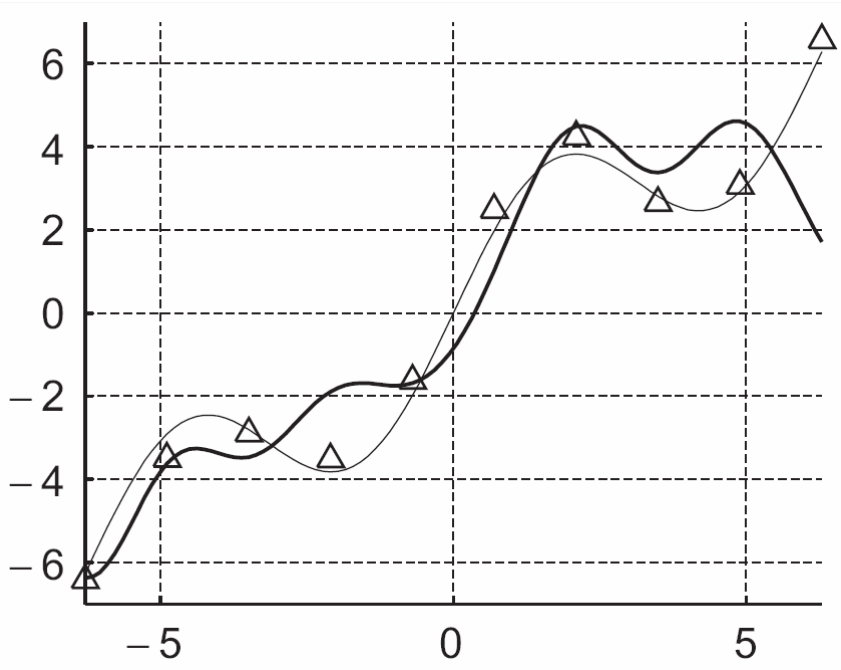
Simulation Example

- Consider approximating the ten noisy data points with fewer than ten basis functions
- $f(x) = 2 \sin(x) + x$
- Five basis functions chosen for approximation
  - half the number of data points.
- Selected to be centered at data points 1, 3, 5, 7 and 9 (data points numbered from 1 through 10 from left to right on the graph [next slide])

Simulation Example

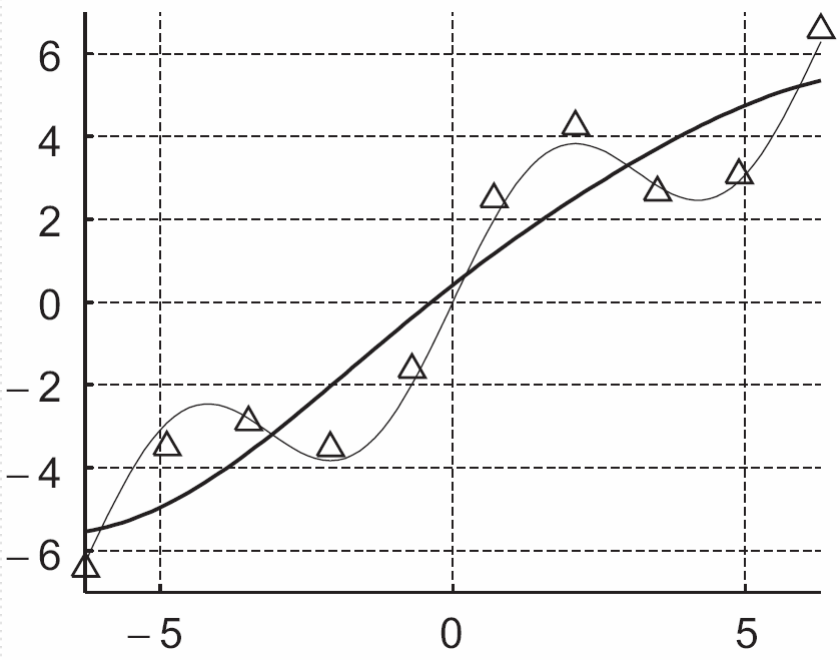
- $\sigma = 0.5$
- $\sigma = 1$

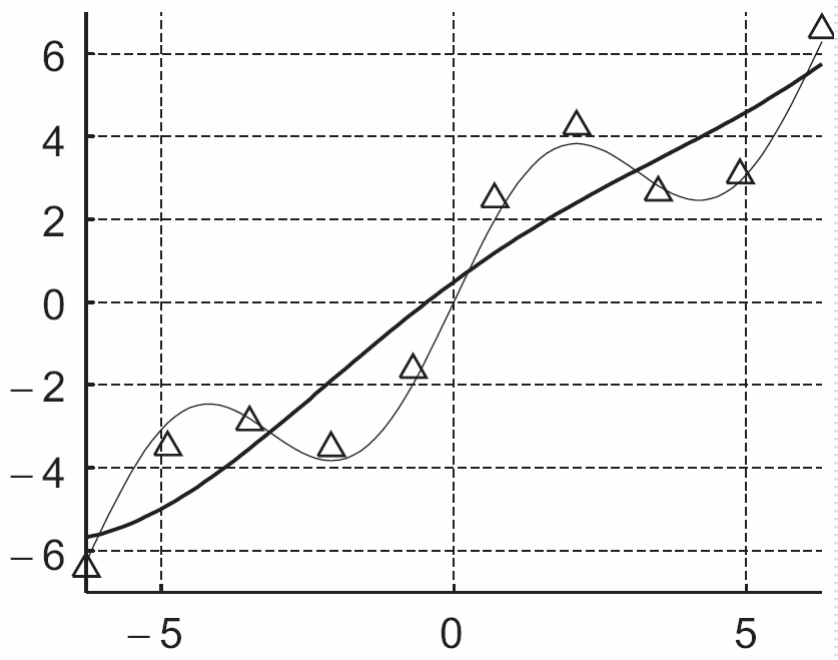




Simulation Example

- $\sigma = 5$
- $\sigma = 10$





MATLAB Code for RBFN

```
Q = 10; % 10 data points
noise = 0.6; % additive noise
x= linspace(-2*pi,2*pi,Q); % X samples
scatter = (2*rand(1,Q) - 1)*noise;
d = (2*sin(x) + x + scatter)'; % Y data
testpts = 100; % Number of test data
testx= linspace(-2*pi, 2*pi, testpts);
testy = (2*sin(testx) + testx)';
sigma = .5;

for i = 1:Q
  k=1;
  for j = 1:Q/2
    phi(i,j) = exp(-(x(i)-
        x(k))^2/(2*sigma^2));
    k=k+2;
  end
end
```

```
% Compute pseudo inverse
pseudoinv = inv(phi' * phi) * phi';
W = pseudoinv * d; % Compute weights
% Generate phi matrixfor test data

for i = 1:testpts
  k=1;
  for j = 1:Q/2
    testphi(i,j) = exp(-(testx(i)-
        x(k))^2/(2*sigma^2));
    k=k+2;
  end
end

% Generate approximant
f = testphi* W;
...
```

RBFN Classifier to Solve the XOR Problem

- Will serve to show how a bias term is included at the output linear neuron

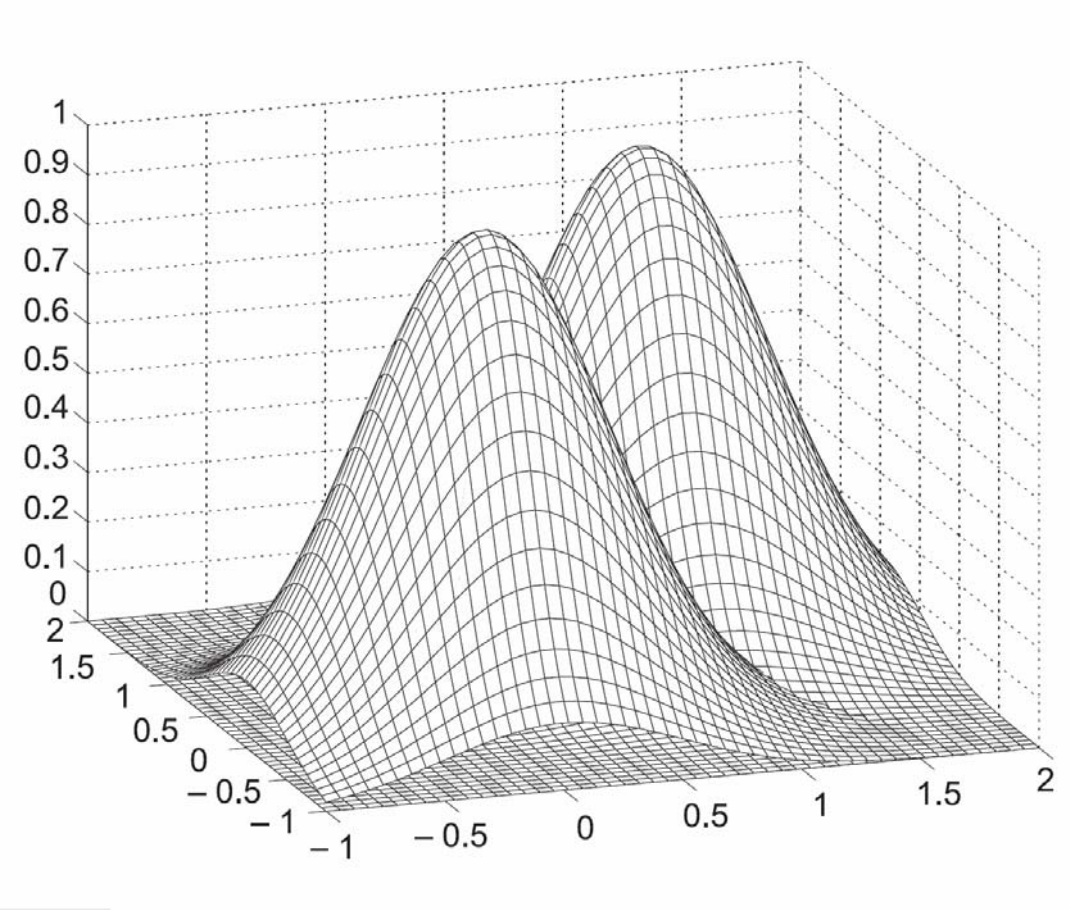
Point # k	x_1	x_2	d
X_1	0	0	0
X_2	0	1	1
X_3	1	0	1
X_4	1	1	0

- RBFN classifier is assumed to have two basis functions centered at data points 1 and 4

$$\phi_1(\|X - X_1\|) = \exp\left(-\frac{\|X - X_1\|^2}{2\sigma^2}\right)$$

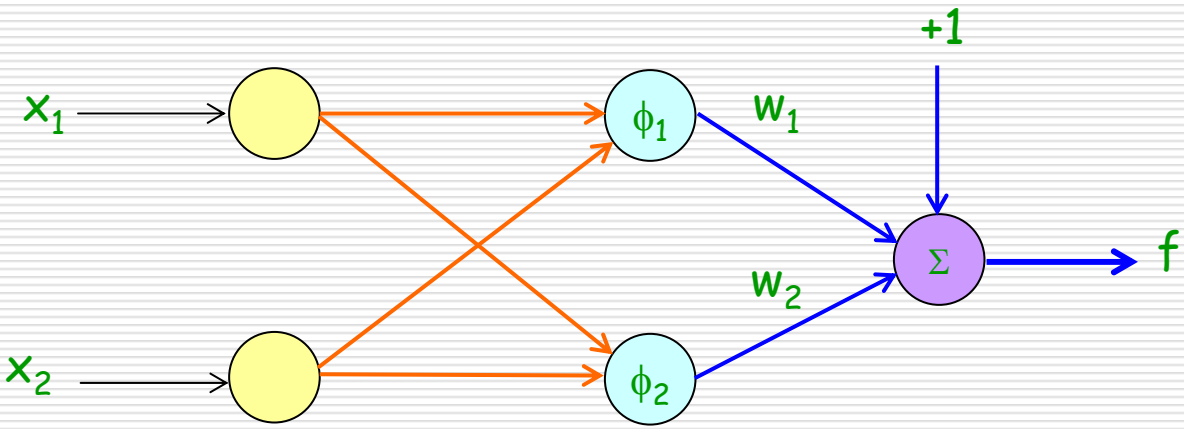
$$\phi_2(\|X - X_4\|) = \exp\left(-\frac{\|X - X_4\|^2}{2\sigma^2}\right)$$

Visualizing the Basis Functions



RBFN Architecture

$$f(X) = w_0 + w_1 \exp\left(-\frac{\|X - X_1\|^2}{2}\right) + w_2 \exp\left(-\frac{\|X - X_4\|^2}{2}\right)$$



Basis functions centered
at data points 1 and 4

Finding the Solution

- We have the D, W, Φ vectors and matrices as shown alongside

$$D = (0\,1\,1\,0)^T$$

$$W = (w_0\ w_1\ w_2)^T$$

$$\Phi = \begin{pmatrix} 1.0000 & 1.0000 & 0.0183 \\ 1.0000 & 0.1353 & 0.1353 \\ 1.0000 & 0.1353 & 0.1353 \\ 1.0000 & 0.0183 & 1.0000 \end{pmatrix}$$

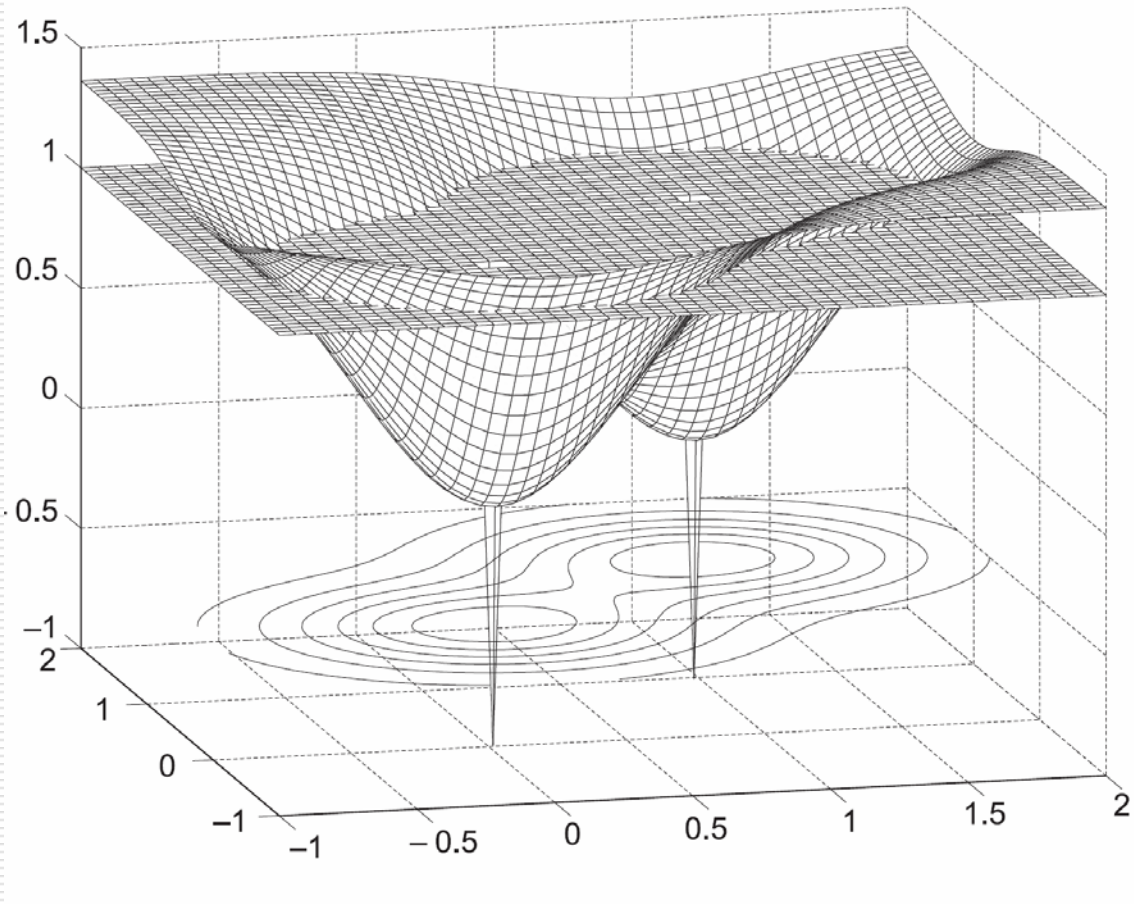
- Pseudo inverse

$$\Phi^{\dagger} = \begin{pmatrix} -0.1810 & 0.6810 & 0.6810 & -0.1810 \\ 1.1781 & -0.6688 & -0.6688 & 0.1594 \\ 0.1594 & -0.6688 & -0.6688 & 1.1781 \end{pmatrix}$$

- Weight vector

$$W = \begin{pmatrix} 1.3620 \\ -1.3375 \\ -1.3375 \end{pmatrix}$$

Visualization of Solution



Ill Posed, Well Posed Problems

- Ill-posed problems originally identified by Hadamard in the context of partial differential equations.
- Problems are well-posed if their solutions satisfy three conditions:
  - they exist
  - they are unique
  - they depend continuously on the data set
- Problems that are not well posed are ill-posed
- Example
  - differentiation is an ill-posed problem because some solutions need not depend continuously on the data
  - inverse kinematics problem which maps external real world movements into an internal coordinate system

Approximation Problem is Ill Posed

- The solution to the problem is not unique
- Sufficient data is not available to reconstruct the mapping uniquely
- Data points are generally noisy
- The solution to the ill-posed approximation problem lies in regularization
  - essentially requires the introduction of certain constraints that impose a restriction on the solution space
- Necessarily problem dependent
- Regularization techniques impose smoothness constraints on the approximating set of functions.
- Some degree of smoothness is necessary for the representative function since it has to be robust against noise.

Regularization Risk Functional

- Assume training data T generated by random sampling of the function
- Regularization techniques replace the standard error minimization problem with minimization of a regularization risk functional

Tikhonov Functional

- Regularization risk functional comprises two terms

$$R_r[f] = \sum_{i=1}^{Q}(d_i - f(X_i))^2 + \lambda\zeta[f]$$

error function

smoothness functional

intuitively appealing to consider using function derivatives to characterize smoothness

Regularization Parameter

- The smoothness functional is expressed as

$$\zeta[f] = \|\mathbf{P}f\|^2$$

  - P is a linear differential operator, ||·|| is a norm defined on the function space (Hilbert space)
- The regularization risk functional to be minimized is

regularization parameter

$$R_r[f] = \underbrace{\sum_{i=1}^{Q}(d_i - f(X_i))^2}_{\text{closeness to data}} + \underbrace{\lambda\|\mathbf{P}f\|^2}_{\text{smoothness}}$$

Euler-Lagrange Equations

- We need to calculate the functional derivative of R_r called the Frechet differential, and set it equal to zero

$$dR_r[f, g] \equiv \frac{d}{d\gamma} R_r[f + \gamma g]\Big|_{\gamma=0}$$

- A series of algebraic steps (see text) yields the Euler-Lagrange equations for the Tikhonov functional

$$\tilde{\mathbf{P}}\mathbf{P} f = \frac{1}{\lambda}\left(\sum_{i=1}^{Q}(d_i - f(X_i))\delta(X - X_i)\right)$$

Solving the Euler-Lagrange System

- Requires the use of the Green's function for the linear differential operator

$$Q = \tilde{P}P$$

- Green's function for a linear differential operator **Q** satisfies prescribed boundary conditions and has continuous partial derivatives with respect to X everywhere except at $X = X_i$ where there is a singularity.
- Satisfies the differential equation $QG(X,Y) = 0$

Solving the Euler-Lagrange System

- See algebra in the text
- Yields the final solution

$$\hat{f}(X) = \frac{1}{\lambda}\sum_{i=1}^{Q}(d_i - f(X_i))G(X, X_i)$$

$$= \sum_{i=1}^{Q} w_i G(X, X_i) \qquad w_i = \tfrac{1}{\lambda}(d_i - f(X_i))$$

- Linear weighted sum of Q Greens functions centered at the data points X_i

Quick Summary

- The regularization solution uses Q Green's functions in a weighted summation
- The nature of the chosen Green's function depends on the kind of differential operator P chosen for the regularization term of R_r

Solving for Weights

- Starting point

$$\hat{f}(X) = \sum_{i=1}^{Q} w_i G(X, X_i)$$

- Evaluate the equation at each data point

$$\hat{f}(X_k) = \sum_{i=1}^{Q} w_i G(X_k, X_i) \quad k = 1, \ldots, Q$$

$$w_i = \frac{1}{\lambda}(d_i - f(X_i)) \quad k = 1, \ldots, Q$$

Solving for Weights

- Introduce matrix notation

$$\hat{F} = (\hat{f}(X_1), \ldots, \hat{f}(X_Q))^T$$

$$W = (w_1, \ldots, w_Q)^T$$

$$D = (d_1, \ldots, d_Q)^T$$

$$\mathbf{G} = \begin{pmatrix} G(X_1, X_1) & \cdots & G(X_1, X_Q) \\ G(X_2, X_1) & \cdots & G(X_2, X_Q) \\ \vdots & \ddots & \vdots \\ G(X_Q, X_1) & \cdots & G(X_Q, X_Q) \end{pmatrix}$$

Solving for Weights

- With these matrix definitions

$$W = \frac{1}{\lambda}(D - \hat{F})$$

$$\hat{F} = \mathbf{G}W$$

- and

$$(\mathbf{G} + \lambda\mathbf{I})W = D$$

- Finally (!)

$$W = (\mathbf{G} + \lambda\mathbf{I})^{-1} D$$

Euclidean Norm Dependence

- If the differential operator P is
  - rotationally invariant
  - translationally invariant
- Then the Green's function G(X,Y) depends only on the Euclidean norm of the difference of the vectors

$$G(X, Y) = G(\| X - Y \|)$$

- Then

$$\hat{f}(X) = \sum_{i=1}^{Q} w_i G(\| X - X_i \|)$$

Multivariate Gaussian is a Green's Function

- Gaussian function defined by

$$G(X, Y) = \exp\left(-\frac{\|X - Y\|^2}{2\sigma_i^2}\right)$$

- The final minimizer is then

$$\hat{f}(X) = \sum_{i=1}^{Q} w_i \exp\left(-\frac{\|X - X_i\|^2}{2\sigma_i^2}\right)$$

- is a Green's function defined by the self-adjoint differential operator

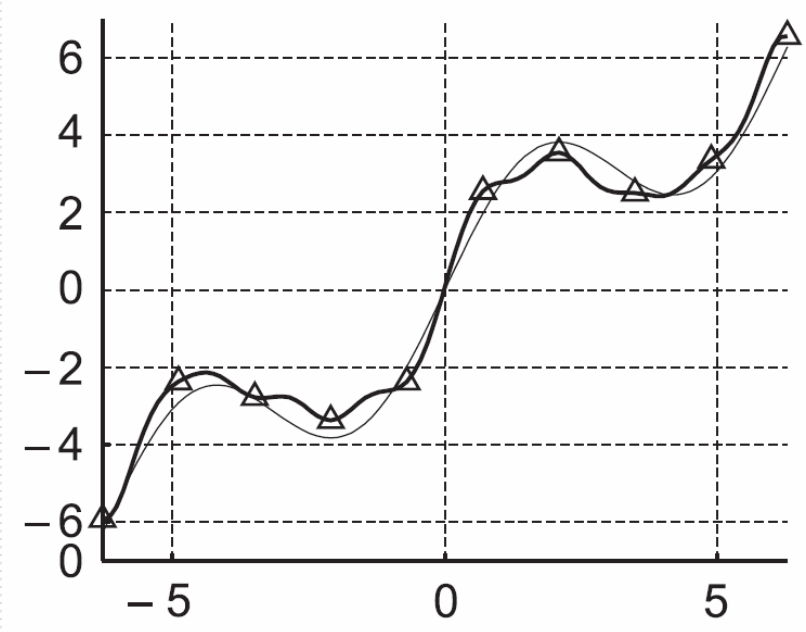
$$\mathbf{Q} = \tilde{\mathbf{P}}\mathbf{P} = \sum_{i=0}^{\infty} (-1)^i \frac{\sigma_i^{2i}}{n!\ 2^i} \nabla^{2i} \qquad \nabla^2 = \frac{\partial^2}{\partial x_1^2} + \cdots + \frac{\partial^2}{\partial x_n^2}$$

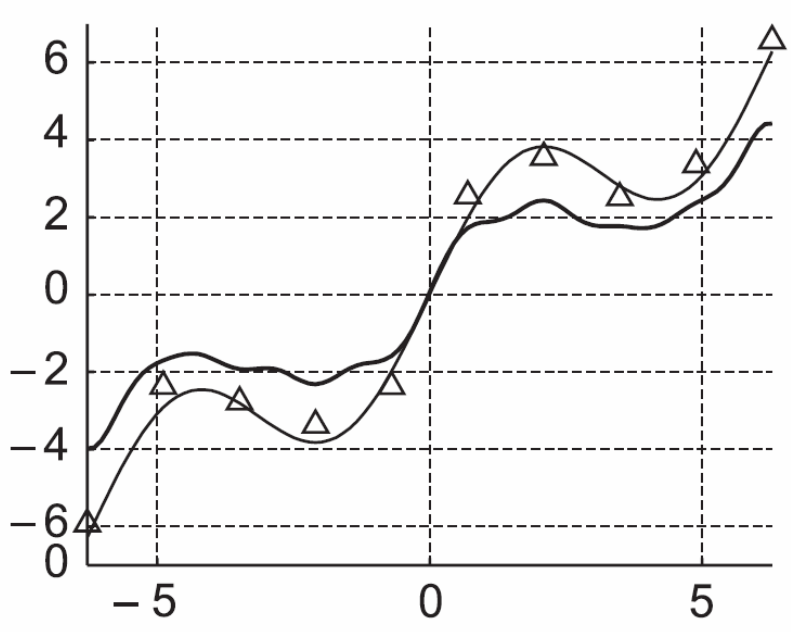
MATLAB Code Segment for RBFN Regularized Interpolation

```
% Code segment for Regularized Interpolation
lambda = 0.5;
for i = 1:Q
for j = 1:Q
phi(i,j) = exp(-(x(i)-x(j))^2/(2*sigma^2));
end
end
Wreg = inv(phi + (lambda * eye(Q))) * d';
for k = 1:testpts
for i = 1:Q
phitest(k,i) = exp(-(testx(k)-x(i))^2/(2*sigma^2));
end
f(k) = phitest(k,:)*W_reg;
end
...
```

Comparing Regularized and Non-regularized Interpolations

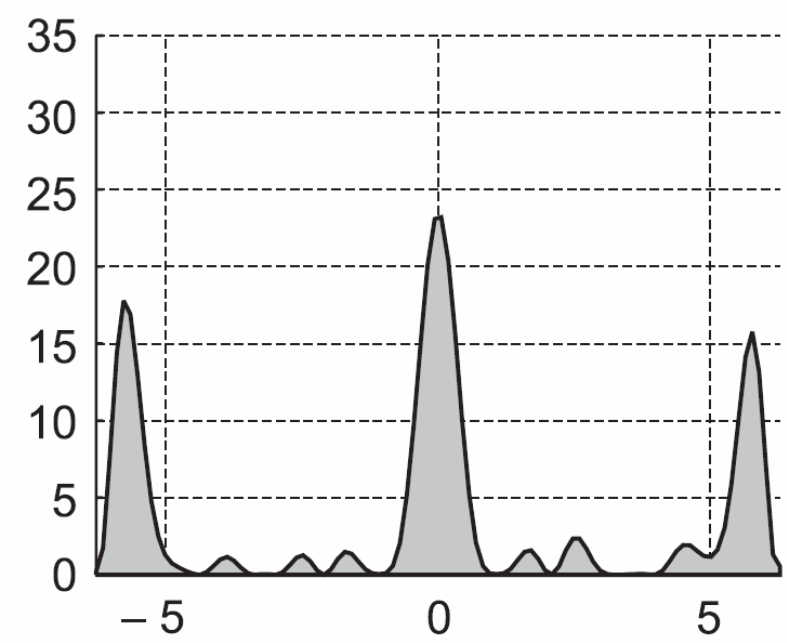
- No regularizing term $\lambda = 0$
- Regularizing term $\lambda = 0.5$

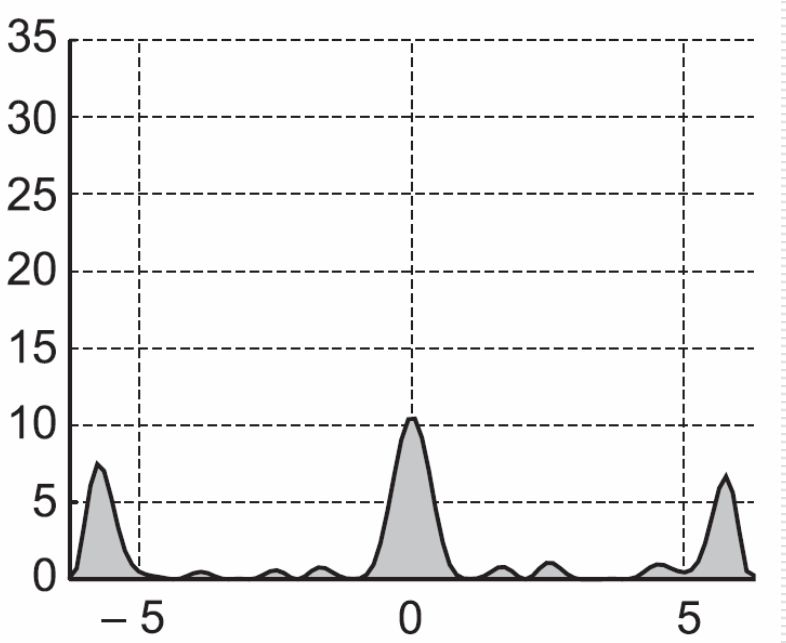




Comparing Regularized and Non-regularized Interpolations

- No regularizing term $\lambda = 0$
- Regularizing term $\lambda = 0.5$





Generalized Radial Basis Function Network

- We now proceed to generalize the RBFN in two steps
  - **Reduce the Number of Basis Functions, Use Non-Data Centers**
  - **Use a Weighted Norm**

Reduce the Number of Basis Functions, Use Non-Data Centers

- The approximating function is,

$$f_a(X) = \sum_{i=1}^{q} w_i G(\|X - \mu_i\|)$$

- Interested in minimizing the regualrized risk

$$R_r[f_a] = \sum_{k=1}^{Q} \left(d_k - \sum_{i=1}^{q} w_i G(\|X_k - \mu_i\|) \right)^2 + \lambda \|\mathbf{P} f_a\|^2$$

Simplifying the First Term

- Using the matrix substitutions

$$\mathbf{G} = \begin{pmatrix} G(\|X_1 - \mu_1\|) & \cdots & G(\|X_1 - \mu_q\|) \\ G(\|X_2 - \mu_1\|) & \cdots & G(\|X_2 - \mu_q\|) \\ \vdots & \ddots & \vdots \\ G(\|X_Q - \mu_1\|) & \cdots & G(\|X_Q - \mu_q\|) \end{pmatrix}$$

$$D = (d_1, \ldots, d_q)^T$$

$$W = (w_1, \ldots, w_p)^T$$

- yields

$$\sum_{k=1}^{Q} \left(d_k - \sum_{i=1}^{q} w_i G(\|X_k - \mu_i\|) \right)^2 = \|D - \mathbf{G}W\|^2$$

$$= (D - \mathbf{G}W)^T (D - \mathbf{G}W)$$

Simplifying the Second Term

- Use the properties of the adjoint of the differential operator and Green's function

$$\|\mathbf{P} f_a\|^2 = W^T \tilde{\mathbf{G}} W$$

- where

$$\tilde{\mathbf{G}} = \begin{pmatrix} G(\|\mu_1 - \mu_1\|) \cdots G(\|\mu_1 - \mu_q\|) \\ G(\|\mu_2 - \mu_1\|) \cdots G(\|\mu_2 - \mu_q\|) \\ \vdots \quad \ddots \quad \vdots \\ G(\|\mu_q - \mu_1\|) \cdots G(\|\mu_q - \mu_q\|) \end{pmatrix}$$

- Finally

$$R_r[f_a] = (D - \mathbf{G}W)^T (D - \mathbf{G}W) + \lambda W^T \tilde{\mathbf{G}} W$$

Using a Weighted Norm

- Replace the standard Euclidean norm by

$$\|X-\mu\|_S^2 = (X-\mu)^T \mathbf{S}^T \mathbf{S}(X-\mu)$$

- S is a norm-weighting matrix of dimension n×n
- Substituting into the Gaussian yields

$$G(\|X-\mu_i\|_S) = \exp\left(-(X-\mu)^T \mathbf{S}^T \mathbf{S}(X-\mu)\right)$$

$$= \exp\left(-\frac{1}{2}(X-\mu)^T \mathbf{K}^{-1}(X-\mu)\right)$$

  - where K is the covariance matrix
- With $K = \sigma^2 I$ is a restricted form

Generalized Radial Basis Function Network

- Some properties
  - Fewer than Q basis functions
  - A weighted norm to compute distances, which manifests itself as a general covariance matrix
  - A bias weight at the output neuron
  - Tunable weights, centers, and covariance matrices

Learning in RBFNs

- **Random Subset Selection**
  - Out of Q data points, q of them are selected at random
  - Centers of the Gaussian basis functions are set to those data points.
- **Semi-random selection**
  - A basis function is placed at every r^{th} data point

Random, Semi-random Selection

- Spreads are a function of the maximum distance between chosen centers and q

$$\alpha \equiv \max_{1 \le i,j \le q} (\| \mu_i - \mu_j \|)$$

- Gaussians are then defined

$$G(\| X - \mu_i \|) = \exp\left(-\frac{\| X - \mu_i \|^2}{2\sigma^2} \right)$$

- such that

$$\sigma = \alpha / \sqrt{2q}$$

Operational Summary of Radial Basis Function Network

- Design assuming random placement of centers and fixed spreads

Given	A training set $\mathcal{T}$ comprising vectors $X_k \in \mathbb{R}^n$ and desired output vectors $d_k \in \mathbb{R}$
Initialize	$\hookrightarrow$ Choose the number of basis functions q $\hookrightarrow$ Set the regularization parameter λ
Design RBFN	$\hookrightarrow$ Set the values of the centers $\{\mu_i\}_{i=1}^{q}$ to random data points using a uniform distribution $\hookrightarrow$ Compute maximum distance between chosen centers α $\hookrightarrow$ Set the spreads $\sigma = \alpha/\sqrt{2q}$ $\hookrightarrow$ Compute $\mathbf{G}$, $\tilde{\mathbf{G}}$ $\hookrightarrow$ Compute pseudo-inverse $(\mathbf{G}^T\mathbf{G} + \lambda\tilde{\mathbf{G}})^{-1}\mathbf{G}^T$ using SVD $\hookrightarrow$ Find the optimal set of weights W $W = \mathbf{G}^{\dagger}D$

Hybrid Learning Procedure

- Determine the centers of the basis functions using a clustering algorithm such as the k-means clustering algorithm
- Tune the hidden to output weights using the LMS procedure

k-Means Clustering

Given	A training set $\mathcal{T}$ comprising vectors $X_k \in \mathbb{R}^n$ and desired output vectors $d_k \in \mathbb{R}$
Initialize	↬ Choose the number of clusters q ↬ Randomize the values of the centers $\{\mu_i^0\}_{i=1}^q$ ↬ Set the learning rate $0 < \eta < 1$
Iterate	↻ ↻ Repeat { ↬ Pick a data point X_k ↬ Find the index J of the closest center: $J = \arg\min_{1 \le i \le q} \|X_k - \mu_i^k\|$ ↬ Update only the closest center: $\mu_J^{k+1} = \mu_J^k + \eta(X_k - \mu_J^k)$ }(until there is no perceptible change in the centers.)

Supervised Learning of Centers

- All the parameters being free and subject to a standard supervised learning procedure such as gradient descent
- Define an error function

$$\mathcal{E} = \frac{1}{2}\sum_{k=1}^{Q}\left(d_k - \sum_{i=1}^{q} w_i G(\|X_k - \mu_i\|_S^2)\right)^2 = \frac{1}{2}\sum_{k=1}^{Q} e_k^2$$

- Free parameters: centers, spreads (covariance matrices), weights

Partial Derivatives

$$\frac{\partial \mathcal{E}_k}{\partial w_i^k} = \sum_{j=1}^{Q} e_j^k G(\|X_j - \mu_i^k\|_S)$$

$$\frac{\partial \mathcal{E}_k}{\partial \mu_i^k} = 2w_i^k \sum_{j=1}^{Q} e_j^k G'(\|X_j - \mu_i^k\|_S)\mathbf{K}_i^{-1|k}(X_j - \mu_i^k)$$

$$\frac{\partial \mathcal{E}_k}{\partial \mathbf{K}_i^{-1|k}} = -w_i^k \sum_{j=1}^{Q} e_j^k G'(\|X_j - \mu_i^k\|_S)(X_j - \mu_i^k)(X_j - \mu_i^k)^T$$

Update Equations

$$w_i^{k+1} = w_i^k - \eta_1 \frac{\partial \mathcal{E}_k}{\partial w_i^k}, \quad i = 1, \ldots, q$$

$$\mu_i^{k+1} = \mu_i^k - \eta_2 \frac{\partial \mathcal{E}_k}{\partial \mu_i^k}, \quad i = 1, \ldots, q$$

$$\mathbf{K}_i^{-1|k+1} = \mathbf{K}_i^{-1|k} - \eta_3 \frac{\partial \mathcal{E}_k}{\partial \mathbf{K}_i^{-1|k}}, \quad i = 1, \ldots, q$$

Image Classification Application

- High dimensional feature space leads to poor generalization performance of image classification algorithms
- Indexing and retrieval of image collections in the World Wide Web is a major challenge
- Support vector machines provide much promise in such applications.
- We now describe the application of support vector machines to the problem of image classification

Extending SVMs to the Multi-class Case

- "One against the others"
- C hyperplanes for C classes

$$y_j(X) = \text{sign}\left(\sum_{i=1}^{n_s} \hat{\lambda}_i d_i K(X, X_i) + b\right)$$

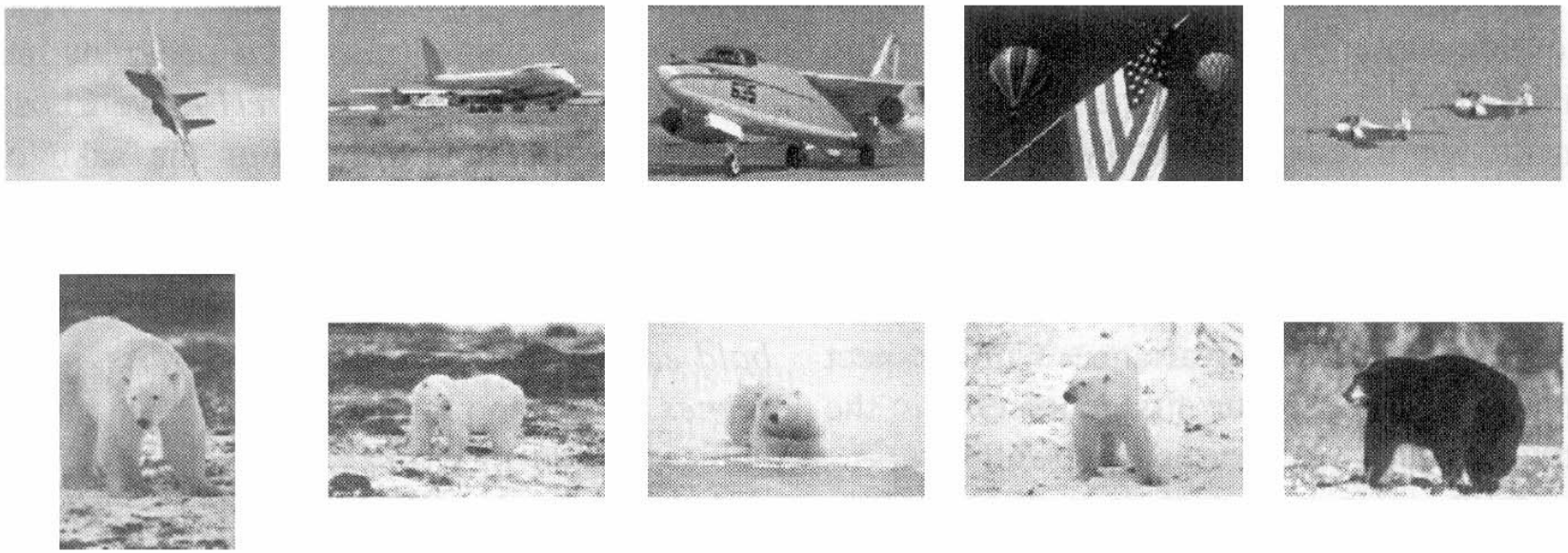
- Class C_J is assigned to point X if

$$J = \arg\max_j \; y_j(X)$$

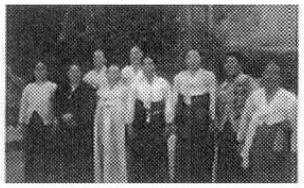
Description of Image Data Set

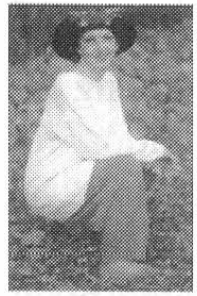
- Corel Stock Photo collection: 200 classes each with 100 images
- Two databases derived from the original collection as follows:
  - Corel14
    - 14 classes and 1400 images (100 images per category)
  - Classes were from the original Corel classification:
    - air shows, bears, elephants, tigers, Arabian horses, polar bears, African specialty animals, cheetahs-leopards-jaguars, bald eagles, mountains, fields, deserts, sunrises-sunsets, night scenes
  - This database has many outliers, deliberately retained
- Corel7
  - Newly designed categories
  - 7 classes and 2670 images
    - airplanes, birds, boats, buildings, fish, people, vehicles

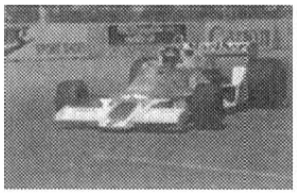
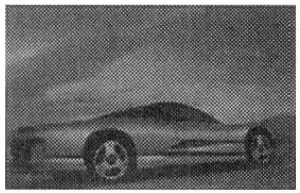
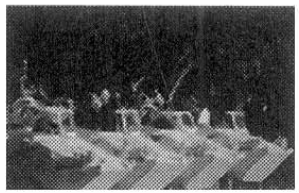
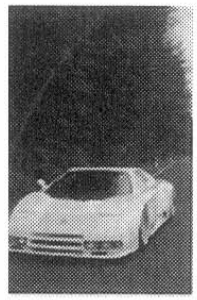
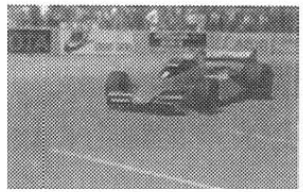
Corel14



Corel7



Colour Histogram

- Colour is represented by a point in a three dimensional colour space:
  - Hue-saturation-luminance value (HSV)
  - Is in direct correspondence with the RGB space.
- Sixteen bins per colour component are selected yielding a dimension of 4096

Selection of Kernel

- Polynomial

$$K_p(X, Y) = (X \cdot Y + 1)^p$$

- Gaussian

$$K_g(X \cdot Y) = \exp(-\rho \| X - Y \|^2)$$

- General kernels

$$K_{d_{RBF}}(X, Y) = \exp(-\rho\, d(X, Y))$$

$$d_{\chi^2}(X, Y) = \sum_i \frac{(x_i - y_i)^2}{x_i + y_i}$$

$$d_{L_1}(X, Y) = \sum_i |x_i - y_i|$$

Gaussian Radial Basis Function Classifiers and SVMs

- Support vector machine is indeed a radial basis function network where
  - the centers correspond to the support vectors
  - the number of centers is the number of support vectors
  - the weights and bias are all chosen automatically using the SVM learning procedure
- This procedure gives excellent results when compared with Gaussian radial basis function networks trained with non-SVM methods.

Experiment 1

- For the preliminary experiment, 1400 *Corel14* samples were divided into 924 training and 476 test samples
- For *Corel7* the 2670 samples were divided into 1375 training and test samples each
- Error Rates

Database	Linear	Poly 2	Gaussian RBF	χ^2 RBF	Laplacian RBF
Corel14	36.3	35.3	30.5	14.7	14.5
Corel7	42.7	38.9	32.2	21.6	20.5

Experiment 2

- Introducing Non-Gaussian Kernels

$$K(X, Y) = \exp(-\rho\, d_{a,b}(X, Y))$$

$$d_{a,b}(X, Y) = \sum_i |x_i^a - y_i^a|^b$$

- In addition to a linear SVM, the authors employed three kernels: Gaussian, Laplacian, sub-linear

Corel14

Kernel	Linear	Gaussian RBF $b = 2$	Laplacian RBF $b = 1$	Sublinear RBF $b = 0.5$
N_{SV}	2023	2307	3339	3352
$a = 1$	36.4	32.7	17.4	13.7
$a = 0.5$	22.2	17.1	12.7	12.6
$a = 0.25$	15.2	13.0	11.0	12.4
$a = 0.125$	14.3	12.0	11.5	12.5
$a = 0.0$	18.4	16.5	16.5	16.5

Corel7

Kernel	Linear	Gaussian RBF $b = 2$	Laplacian RBF $b = 1$	Sublinear RBF $b = 0.5$
N_{SV}	3567	4137	5127	5418
$a = 1$	42.5	40.4	22.8	19.2
$a = 0.5$	28.4	21.2	17.4	18.5
$a = 0.25$	23.6	17.6	16.3	18.8
$a = 0.125$	26.9	28.6	19.0	19.3
$a = 0.0$	33.2	24.2	24.2	24.2

Weight Regularization

- Regularization is a technique that builds a penalty function into the error function itself
  - increases the error on poorly generalizing networks
- Feedforward neural networks with large number and magnitude of weights generate over-fitted network mappings that have high curvature in pattern space
- Weight regularization: Reduce the curvature by penalizing networks that have large weight values

Introducing a Regularizer

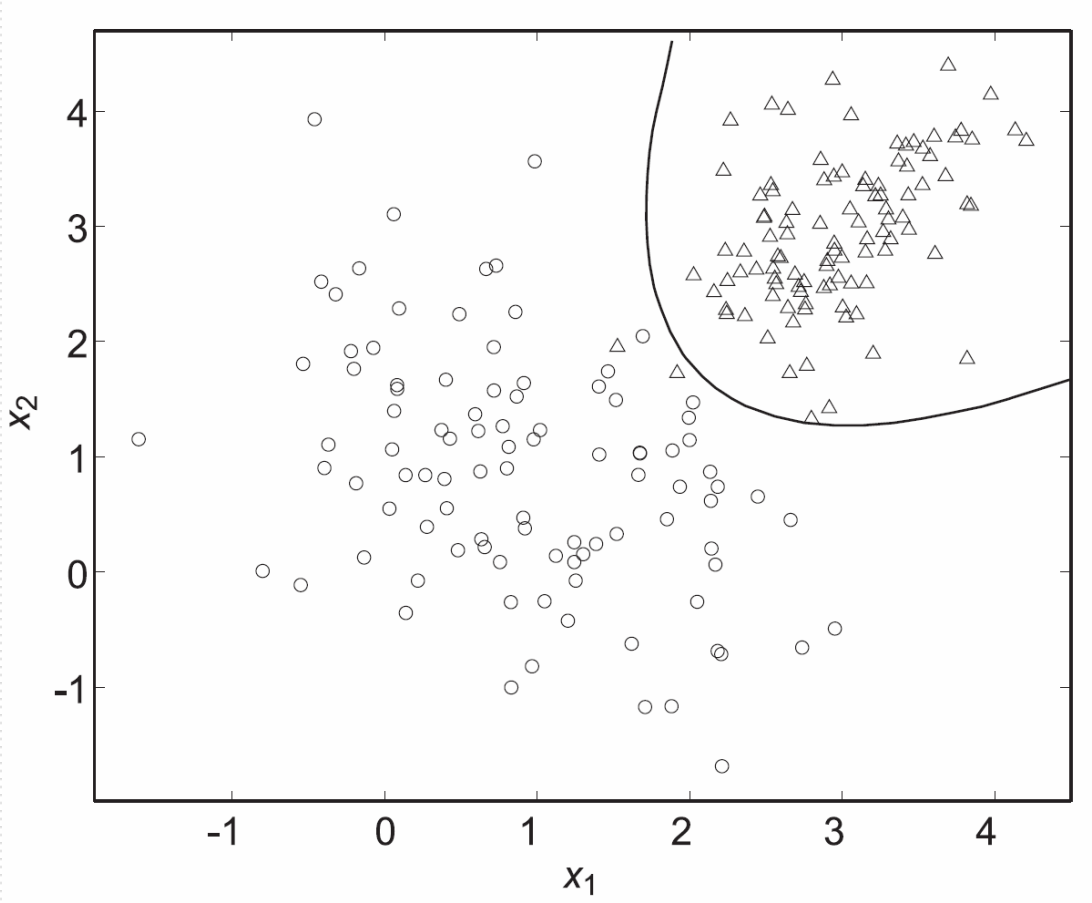
- Basic idea: add a "sum of weight squares" term over all weights in the network presently being optimized
- α is a weight regularization parameter
- A weight decay regularizer needs to treat both input-hidden and hidden-output weights differently in order to work well

$$\tilde{\mathcal{E}} = \mathcal{E} + \alpha\Omega$$
$$= \mathcal{E} + \alpha\left(\frac{1}{2}\sum_i w_i^2\right)$$
$$= \mathcal{E} + \frac{\alpha}{2} W^T W$$

$$\Omega = \frac{\alpha_1}{2}\sum_i\sum_h w_{ih}^2 + \frac{\alpha_2}{2}\sum_h\sum_j w_{hj}^2$$

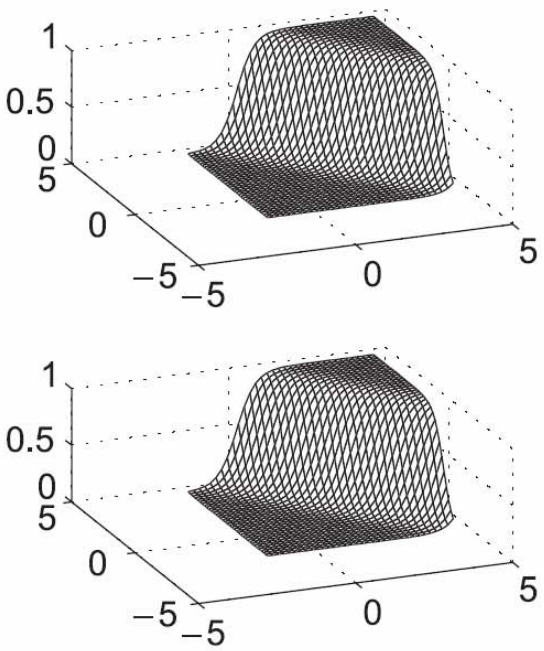
MATLAB Simulation

- Two-class data for weight regularization example

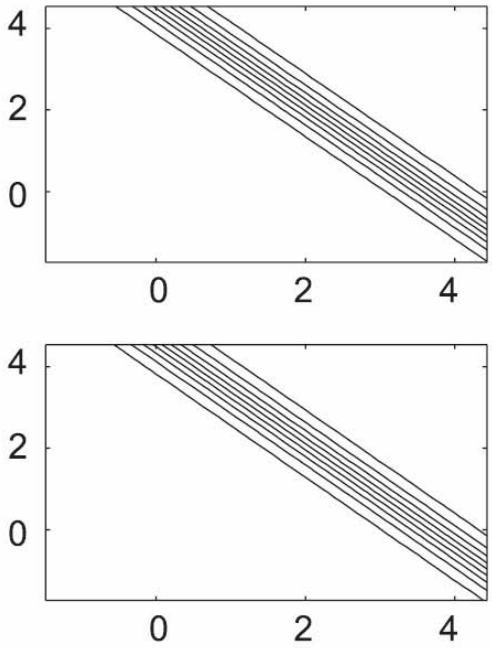


MATLAB Simulation α = 0, 0.01

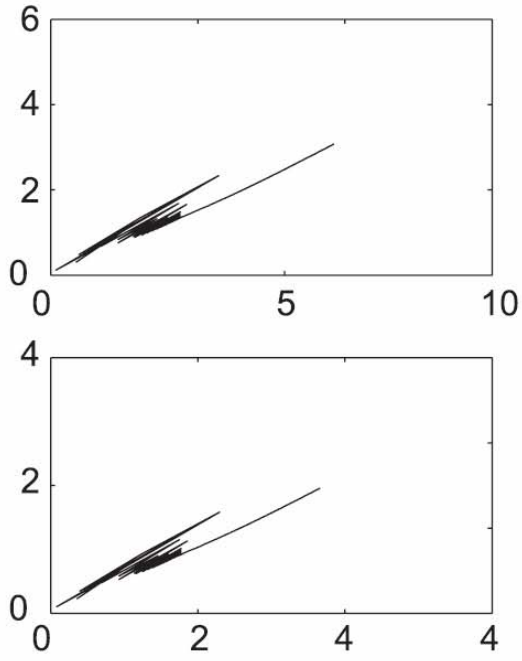
Signal function



Contours



Weight space trajectories

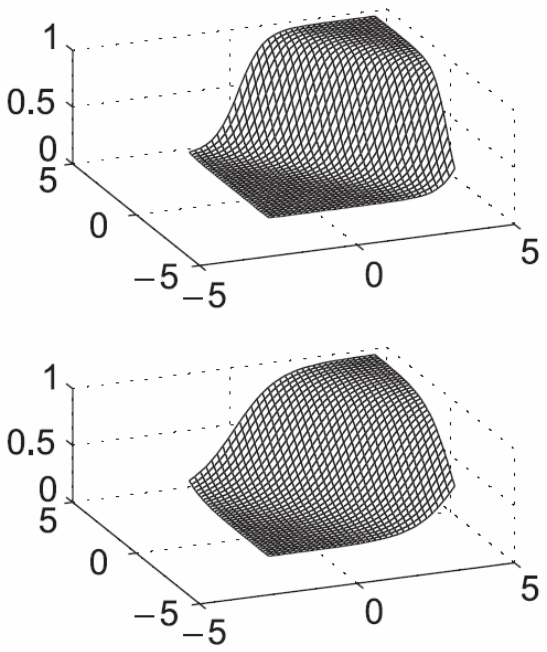


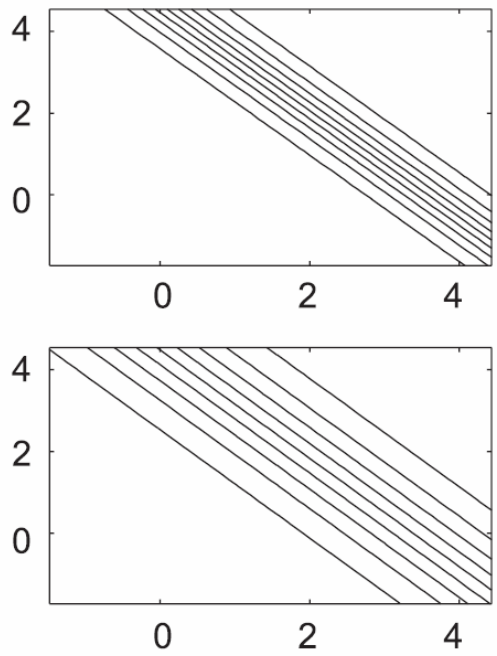
MATLAB Simulation α = 0.1, 1

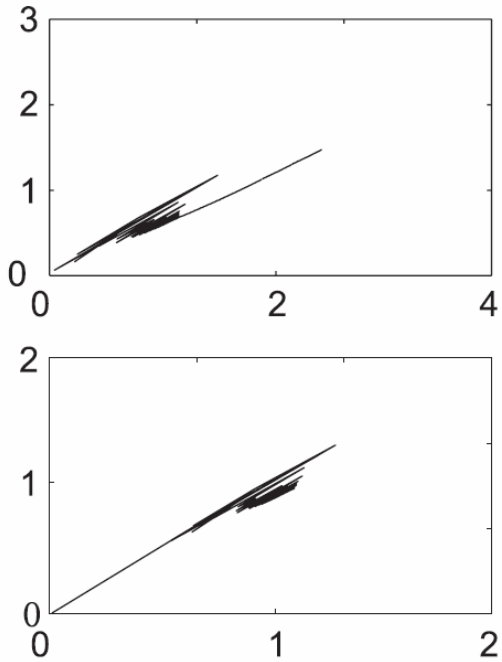
Signal function

Contours

Weight space trajectories



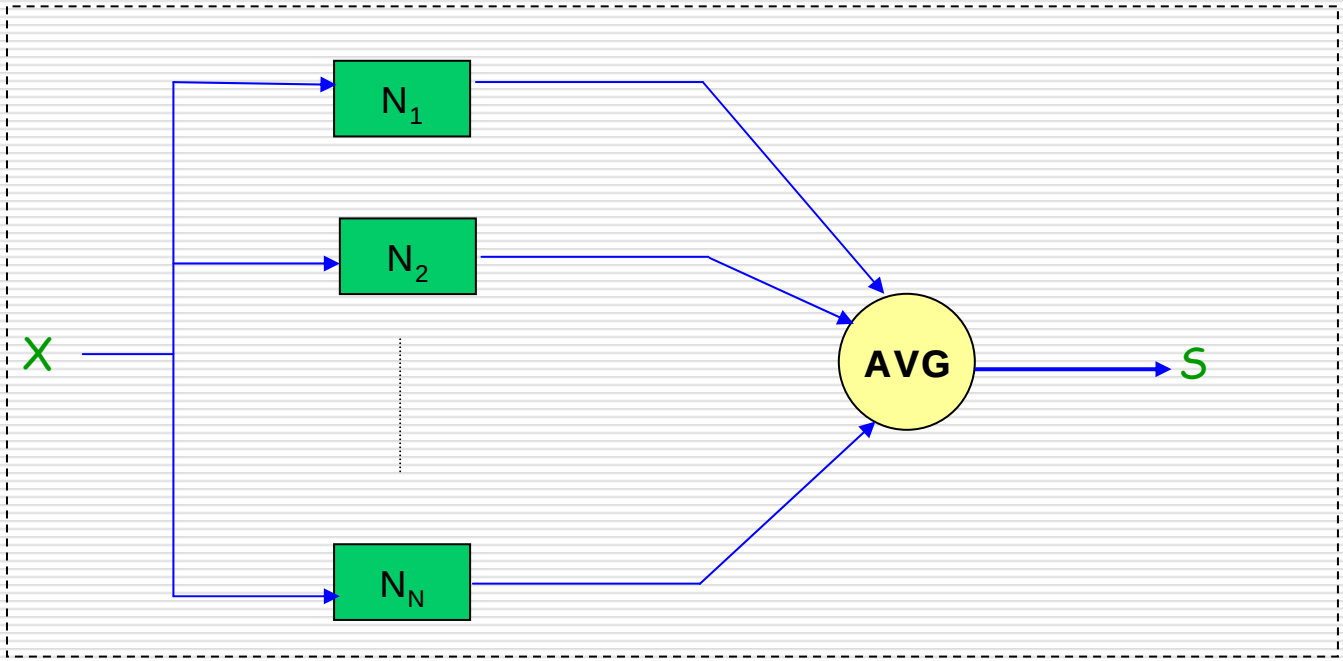




Committees of Networks

- A set of different neural network architectures that work together to generate an estimate of the underlying function f(X)
- Each network is assumed to have been trained on the same data distribution although not necessarily the same data set
- An averaging out of noise components reduces the overall noise in prediction
- Performance can actually improve at a minimal computational cost when using a committee of networks

Architecture of Committee Network



Averaging Reduces the Error

- Analysis shows that the error can only reduce on averaging
- Assume

$$\mathcal{E}_{\text{av}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{E}_i$$

$$= \frac{1}{N}\sum_{i=1}^{N}E\left[\epsilon_i^2(X)\right]$$

$$\mathcal{E}_C = \frac{1}{N^2}E\left[\sum_i \epsilon_i^2 + \sum_i\sum_{\substack{j \\ j\neq i}}\epsilon_i\epsilon_j\right]$$

$$= \frac{1}{N^2}\left(\sum_i E[\epsilon_i^2] + \sum_i\sum_{\substack{j \\ j\neq i}}E[\epsilon_i\epsilon_j]\right)$$

$$= \frac{1}{N^2}\sum_{i=1}^{N}E[\epsilon_i^2]$$

$$= \frac{1}{N}\mathcal{E}_{\text{av}}$$

Mixtures of Experts

- Learning a map is decomposed into the problem of learning mappings over different regions of the pattern space
- Different networks are trained over those regions
- Outputs of these individual networks can then be employed to generate an output for the entire pattern space by appropriately selecting the correct networks' output
- Latter task can be done by a separate gating network
- The entire collection of individual networks together with the gating network is called the mixture of experts model