Module 3, Lecture 2: Applying Classical Neural Networks

M. Vidyasagar

Distinguished Professor, IIT Hyderabad
Email: m.vidyasagar@iith.ac.in
Website: www.iith.ac.in/∼m_vidyasagar/

Outline

Outline

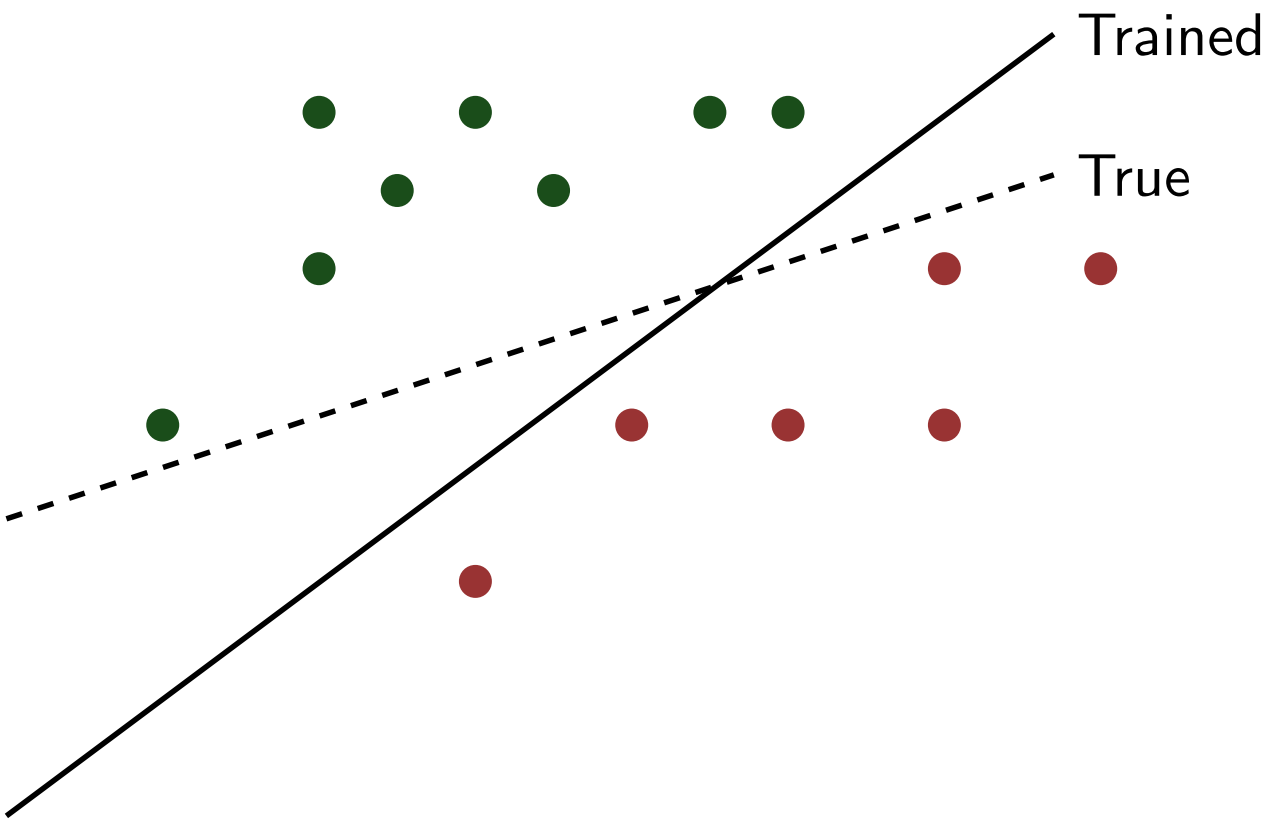
Problem Formulation

Intuitively we feel that

- The generalization error is never *exactly* zero. But as we draw more and more samples, the error *approaches* zero (as in next slide).
- Because the learning samples are drawn at random, the generalization error is also random. Therefore, *with some small probability*, the generalization error could be huge (as in second next slide).
- So, at best, we can aspire to get *low (but not zero)* generalization error with *high probability* (but not probability of one).

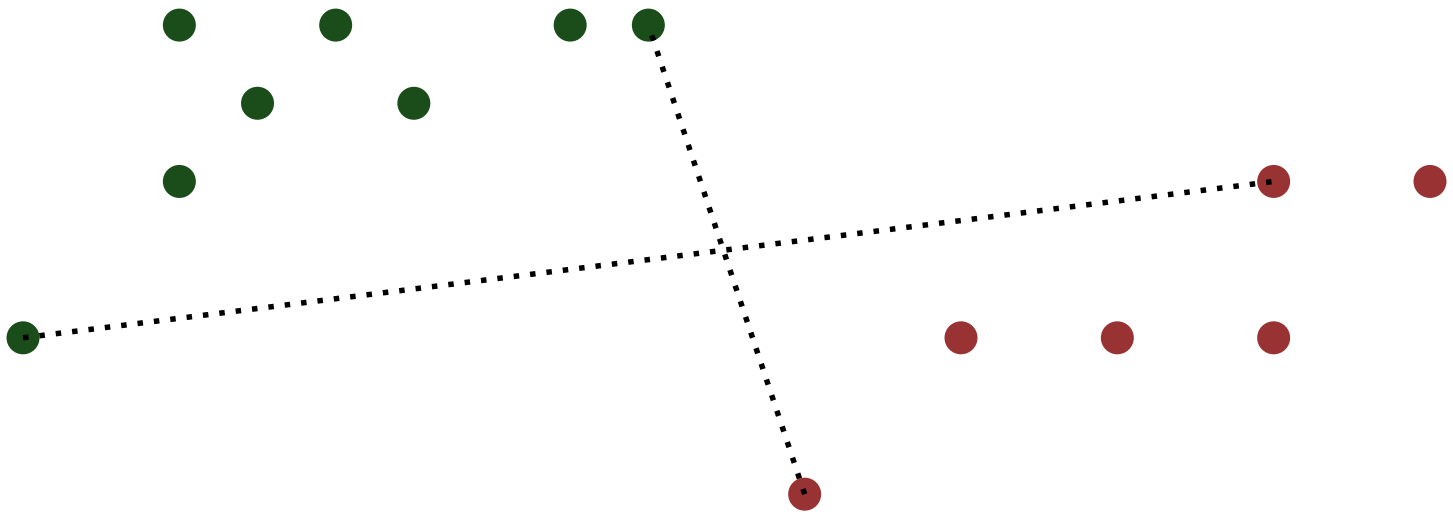
Can we make these statements precise?

Bounding the Generalization Error



The trained NN has zero training error. Can we *bound* the generalization error?

Possibility of Getting Bad Training Samples



Because training samples are randomly drawn, occasionally we might get *huge* generalization error.

Precise Formulation

Let us specify two parameters: **accuracy** ϵ, and **confidence** δ. Both numbers should be small.

Problem: Given ϵ, δ, can we find a "sample complexity" $m_0 = m_0(\epsilon, \delta)$ with the following property?

For *every* unknown perceptron, and *every* candidate perceptron that correctly classifies the training data, the generalization error is $\leq \epsilon$ with probability $\geq 1 - \delta$.

Is this even possible?

Solution to the Formulated Problem

For a perceptron with n inputs, define $d = n + 1$. Then

$$m_0(\epsilon, \delta) = \max\left\{ \frac{8d}{\epsilon} \log_2 \frac{8e}{\epsilon}, \frac{4}{\epsilon} \log_2 \frac{2}{\delta} \right\}.$$

Example: Taking $n = 2, \epsilon = 0.01, \delta = 10^{-6}$ gives $m_0 = 26,608$.

So, irrespective of what the unknown perceptron is, and what probability distribution is used to draw the samples, if we draw m_0 samples and find *any* separating line, then the next test sample will be correctly classified 99% of the time, with probability 0.999999.

Observation: m_0 increases linearly with d, like $(1/\epsilon)\log(1/\epsilon)$, and like $\log(1/\delta)$.

Outline

Outline

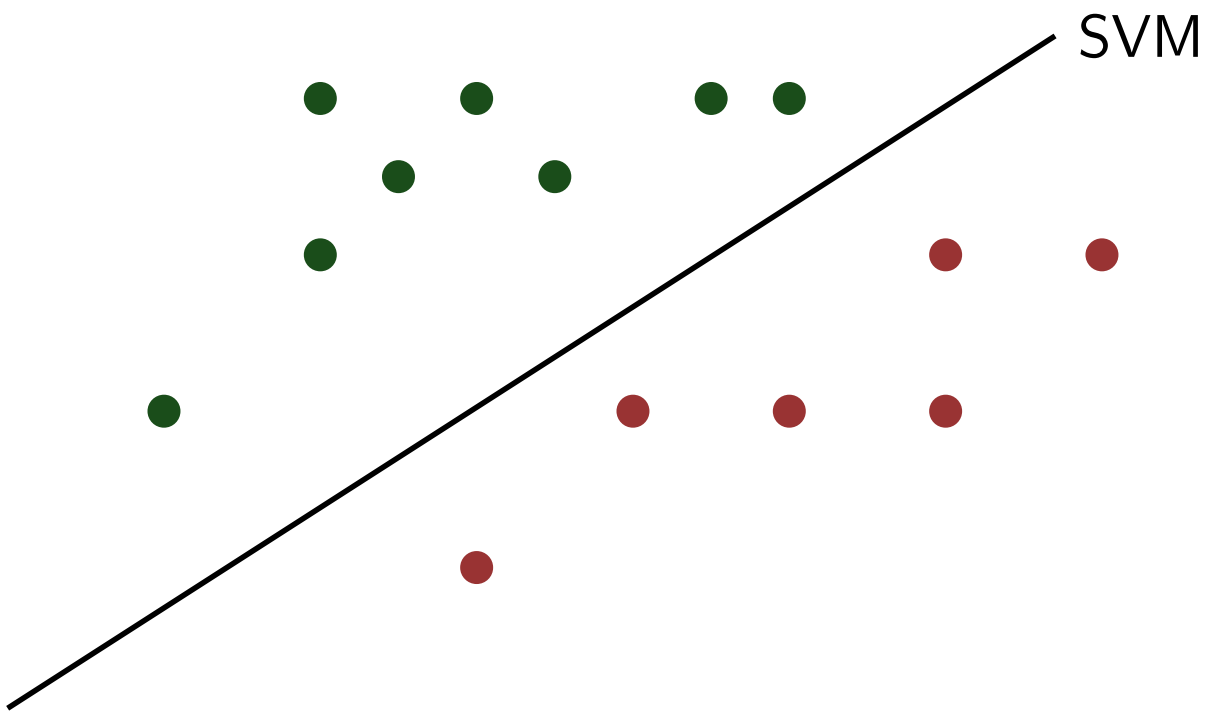
Support Vector Machines

Note: The original paper by Cortes and Vapnik Link had the title "Support Vector *Networks*," not "Support Vector *Machines*." But afterwards "machine" has been used universally, and the abbreviation SVM took root.

Question: Given a set of linearly separable data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ where $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}$, by definition there exist *infinitely many* separating hyperplanes. Which one is *the best*?

Answer: The hyperplane such that the closest point is as far as possible from the hyperplane.

Depiction of a Support Vector Machine



Construction of a Support Vector Machine

If $(\mathbf{w}, \theta)$ defines the optimal hyperplane, so does $(\alpha\mathbf{w}, \alpha\theta)$ for every $\alpha > 0$. So we need to normalize.

Define the **Euclidean norm** (denoted by $\|\cdot\|_2$ by

$$\|\mathbf{w}\|_2 = \left(\sum_{i=1}^{n} w_i^2 \right)^{1/2}.$$

Then the SVM can be found by solving this problem:

$$\min_{\mathbf{w},\theta} \|\mathbf{w}\|_2^2 \text{ s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1, i = 1, \ldots, m.$$

This is known as a "quadratic programming (QP) problem" and can be solved for very large-sized problems with limited resources.

Construction of a Support Vector Machine (Cont'd)

In the problem formulation, the constraints

$$y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1, i = 1, \ldots, m$$

take care of the "scaling" of $\mathbf{w}, \theta$. Also, the number 1 on the right side is rather arbitrary, and is called the "margin" of the classifier.

(Look ahead to "soft-margin" classifiers.)

Why is it Called a Support Vector Machine?

The optimal solution depends *only on the closest vectors* to the optimal separating hyperplane.

Specifically, the optimal weight vector $\mathbf{w}$ is of the form

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i,$$

where $\alpha_i = 0$ unless $\mathbf{x}_i$ is one of the closest samples to the optimal separating hyperplane.

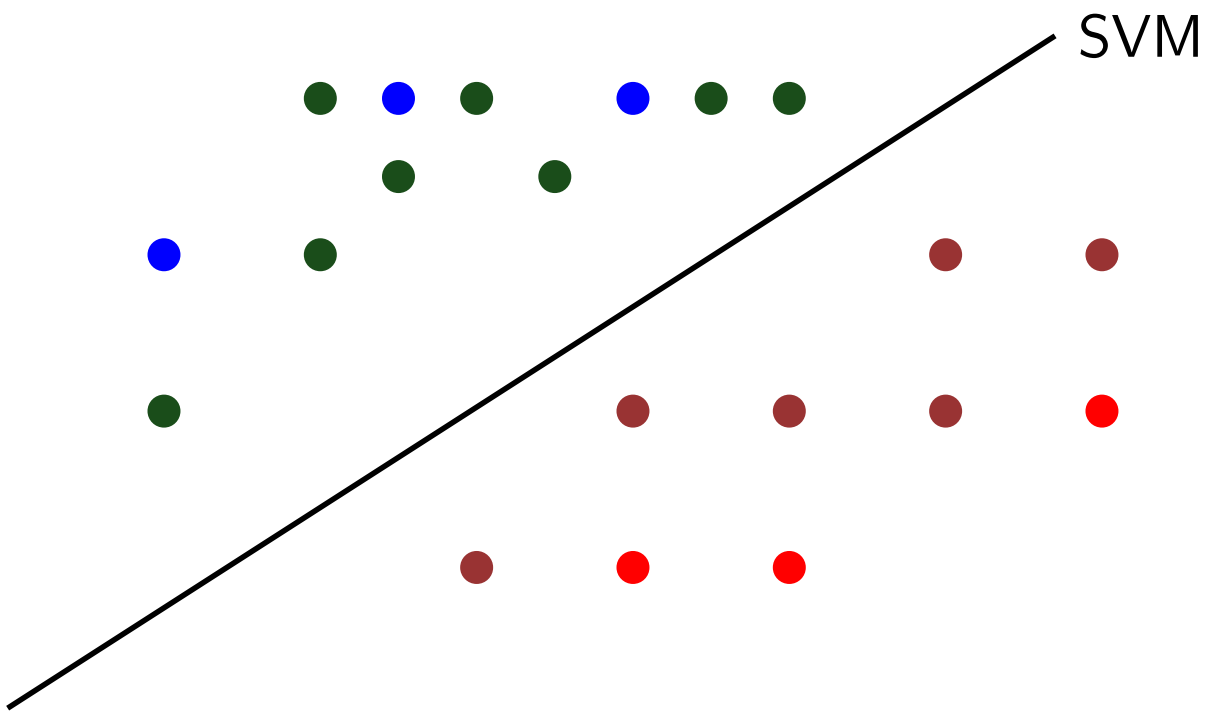
These closest samples are said to "support" the hyperplane. Consequently, other samples that are *not* the closest to the optimal hyperplane *d not affect* the choice of $\mathbf{w}$.

IIT Hyderabad

Insensitivity of the Optimal Separating Hyperplane

Consequence of the "support" property: If new data is added that is farther from the optimal separating hyperplane than the closest point, *the optimal separating hyperplane need not be recomputed!* (Next slide.)

Insensitivity of the Optimal Separating Hyperplane



New data does not change the optimal separating hyperplane.

Outline

Soft-Margin SVMS

The traditional SVM makes sense only when the data is linearly separable. In the general case we have two choices:

- First test whether the data is linearly separable. If it is, solve the "standard" SVM formulation. If not, try something else.
- Come up with a more general formulation that includes the "standard" SVM as a special case if the data is linearly separable.

Soft-margin classifiers follow the second approach.

Soft-Margin SVMs: Mathematical Formulation

Recall the standard SVM formulation:

$$\min_{\mathbf{w},\theta} \|\mathbf{w}\|_2^2 \text{ s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1, i = 1, \ldots, m.$$

Now we change this to

$$\min_{\mathbf{w},\theta,\xi_i} \|\mathbf{w}\|_2^2 + C\|\boldsymbol{\xi}\|_2^2 = \sum_{i=1}^{n} w_i^2 + C\sum_{i=1}^{m} \xi_i^2$$

subject to the constraints

$$y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, m.$$

Soft-Margin SVMs: Mathematical Formulation (Cont'd)

Compared to the standard SVM formulation, the soft-margin classifier involves two changes.

First, the constraints

$$y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1, i = 1, \ldots, m$$

are replaced by

$$y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, m.$$

If $0 \leq \xi_i < 1$, then the inner product still has the correct sign, but with a reduced margin. But if $\xi_i > 1$, then the sample $\mathbf{x}_i$ can be misclassified.

Soft-Margin SVMs: Mathematical Formulation (Cont'd)

Second, the original cost function $\|\mathbf{w}\|_2^2$ is replaced by

$$\|\mathbf{w}\|_2^2 + C\|\boldsymbol{\xi}\|_2^2 = \sum_{i=1}^{n} w_i^2 + C\sum_{i=1}^{m} \xi_i^2$$

The added term $C\|\boldsymbol{\xi}\|_2^2$ is called the "regularizer." It prevents the additional constants ξ_i from becoming too large.

Interpretation of the Soft-Margin SVM

If the original data is linearly separable, then the constraints can be satisfied with $\xi_i = 0$ for all i. Hence, if C is large enough, then this will be the optimal choice.

If the data is not linearly separable, then some ξ_i won't be zero. So we get a trade-off between the margin and linear separability. In effect, some of the training samples $(\mathbf{x}_i, y_i)$ will be *incorrectly classified* by the optimal $(\mathbf{w}, \theta)$ pair.

This is still a quadratic programming (QP) problem, and can be solved efficiently.

Feature Selection SVMs

In most typical engineering applications, the dimension of the samples (n) is much smaller than the number of samples (m).

However, in applications such as cancer biology, exactly the opposite is true. The integer n equals the number of genes in the human body, around 20,000, whereas the integer m equals the number of cancer tumors, a few hundred at best.

In such applications, the linear separability of the data is a non-issue (discussed later). Rather, the key is to identify *the most predictive features*.

ℓ_1-Norm SVM for Feature Selection

An easy way to achieve feature selection is to modify the SVM formulation to

$$\min_{\mathbf{w},\theta} \|\mathbf{w}\|_1 \text{ s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1, i = 1, \ldots, m,$$

where

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{n} |w_i|.$$

So the objective function $\|\mathbf{w}\|_2^2$ is replaced by $\|\mathbf{w}\|_1$.

ℓ_1-Norm SVM for Feature Selection (Cont'd)

The problem can be converted to a *linear programming (LP)* problem, by noting that every real number w is the difference of two nonnegative numbers α, β, and $|w| = \min\{\alpha + \beta\}$.

So the ℓ_1-norm SVM can be rewritten as

$$\min_{\alpha_l, \beta_l} \sum_{l=1}^{n} \alpha_l + \beta_l \text{ s.t. } \sum_{l=1}^{n} (\alpha_l - \beta_l) y_i (x_{il} - \theta) \geq 1, i = 1, \ldots, m,$$

Fact: The optimal solution $\mathbf{w}$ has *no more than m nonzero components*, though it is an n-dimensional vector. Thus feature selection happens automatically.

Asymmetric Penalties for Misclassification

Recall the problem statement: We are given labelled data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ where $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \{-1, 1\}$. We wish to find an SVM that assigns the correct label to most of the vectors $\mathbf{x}_i$.

In some applications, the consqeuences of misclassification are *not symmetric.* Examples:

- Tumor classification: Classifying a malignant tumor as benign is a lot worse than the other way around.
- At an airport security checkpoint, failing to detect someone carrying weapons is much more serious than frisking some unarmed people.

SVM with Asymmetric Penalties for Misclassification

Divide the training samples into two groups: Renumber so that $y_i = 1$ for $i = 1, \ldots, m_1$ (the first m_1 samples are assigned a label of $+1$), and the next m_2 are assigned a label of -1.

Choose weights γ for misclassifying first group of samples, and δ for misclassifying the second group of samples.

Solve

$$\min_{\mathbf{w},\theta,\xi_i} \sum_{i=1}^{n} w_i^2 + C \left(\sum_{i=1}^{m_1} \gamma \xi_i^2 + \sum_{i=m_1+1}^{m_2} \delta \xi_i^2 \right)$$

subject to the constraints

$$y_i(\mathbf{w}^\top \mathbf{x}_i - \theta) \geq 1 - \xi_i, \xi_i \geq 0, i = 1, \ldots, m.$$

Higher-Order SVMs

The SVM is inherently a *linear* classifier. The standard SVM uses a *linear discriminant function* of the form

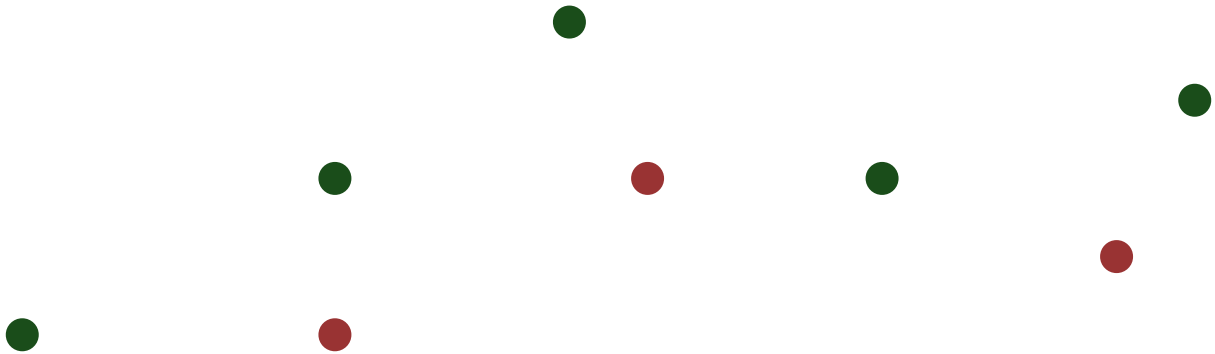
$$\Delta(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} - \theta$$

to classify the vectors.

If $\Delta(\mathbf{x}_i) \geq 0$, the sample $\mathbf{x}_i$ is put into the $+$ bucket, otherwise into the $-$ bucket.

Higher-Order SVMs (Cont'd)

But not all data is linearly separable! Especially so when the number of features n is smaller than the number of features m.



Higher-Order SVMs (Cont'd)

If the data is not linearly separable, each sample vector $\mathbf{x}_i \in \mathbb{R}^n$ is mapped into a *higher-dimensional space* called the **enlarged feature space**.

If the enlarged feature space has higher dimension than m, the number of samples, this ensures that *random labellings* of the samples become linearly separable. (More on this later.)

Common Feature Maps

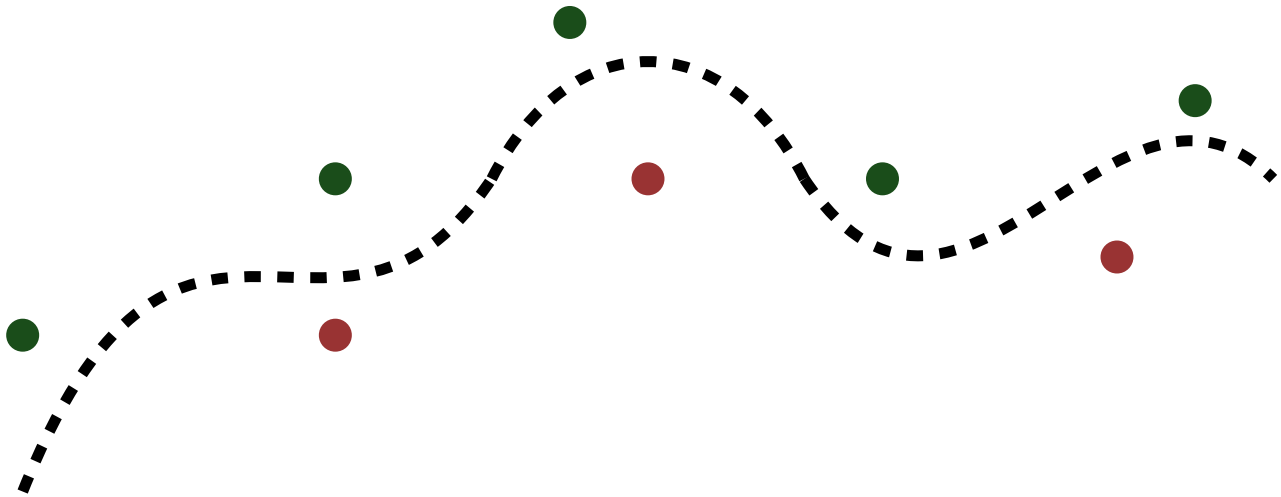
The simplest set of features is to map the components of $\mathbf{x}$ into higher-degree polynomials.

So if $\mathbf{x} = [x_1 \ldots x_n]^\top$, we define $n(n+1)/2$ quadratic terms: n terms of the form x_i^2, and $(n(n-1))/2$ terms of the form $x_i x_j, i \neq j$. Together with the n linear terms x_i, this gives rise to a total of $(n(n+3))/2$ features for each of the original n-dimensional feature vectors. Then we run an SVM using this enlarged feature space.

This leads to a *nonlinear* discriminant function of the form

$$\Delta(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} x_i x_j - \theta.$$

Nonlinear Discriminant Functions Illustrated



Kernel-Based Learning

The idea of using an enllarged feature set for each sample leads naturally to **kernel-based learning**.

Given an n-dimensional "original" feature vector $\mathbf{x}$, define a map $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^l$ where $l \ll n$, so that $\boldsymbol{\phi}(\mathbf{x})$ is the enlarged fature vector corresponding to $\mathbf{x}$. Also, let $\mathbf{v}$ denote the weight vector in the l-dimensional space of enlarged features.

Then standard SVM theory tells us that the optimal (enlarged) weight vector $\mathbf{v}$ looks like

$$\mathbf{v} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{v}_i \phi(\mathbf{x}_i),$$

where $\alpha_i = 0$ unless the vector $\phi(\mathbf{x}_i)$ is a support vector.

Kernel-Based Learning (Cont'd)

So the optimal discriminant function now looks like

$$\Delta(\mathbf{x}) = \mathbf{v}^\top \boldsymbol{\phi}(\mathbf{x}) - \theta = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{\phi}^\top(\mathbf{x}_i) b \phi(\mathbf{x}) - \theta.$$

The function

$$\boldsymbol{\phi}^\top(\mathbf{u})\boldsymbol{\phi}(x) = K(\mathbf{u}, \mathbf{x})$$

is called the **kernel**. Moreover, the discriminant Δ makes use of this kernel function.

In kernel-based learning, the kernel function (or enlarged feature set) is specified by the user.

One can also "over-specify" and use the feature-selection SVM to determine the most appropriate set of (enlarged) features.

SVMs for Recognizing Handwritten Characters

During the 1990s, recognizing handwritten characters was seen as a difficult test case.

The Cortex-Vapnik paper gave a successful solution to this problem using SVMs. ▸ Link

SVMs for Recognizing Handwritten Characters (Cont'd)

Cortes-Vapnik used a training set of $60,000$ 28×28 binary images, and a test set of $10,000$ images.

Therefore $n = 28^2 = 784$. They then used a fourth-order polynomial to generate an enlarged feature space.

They trained one SVM for each of the ten digits (ten SVMs in all) and had a majority-polling method.

Outline

The Vapnik-Chervonenkis (VC)-Dimension

With every neural network *architecture*, we can associate an intege, call it d, referred to as the **Vapnik-Chervonenkis (VC)-Dimension**. In its simplest form it applies to *classification* NNs.

It is a measure of the "richness" of the architecture, and it is *monotonic*.

So if we take a NN architecture and add a few more layers and/or neurons, the VC-dimension can only go up.

Before we define this concept, we answer the question: What is it good for?

VC-Dimension and the Learning Rate

Suppose d is the VC-dimension of a NN architecture. Given a set of labelled samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ where $y_i \in \{-1, 1\}$.

Train the NN to achieve the minimum classification error on the training samples, by adjusting the weights to minimize the *fraction of misclassified samples*.

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \#(y_i \neq f(\mathbf{w}, \mathbf{x}_i)).$$

Let J^* denote this minimum (which could be zero if we achieve perfect classification of the training set).

VC-Dimension and the Learning Rate (Cont'd)

Given an accuracy ϵ, δ, define

$$m_0(\epsilon, \delta) = \max\left\{\frac{8d}{\epsilon}\log_2\frac{8e}{\epsilon}, \frac{4}{\epsilon}\log_2\frac{2}{\delta}\right\},$$

where d is the VC-dimension of the NN architecture (or an upper bound).

If the network is trained with at least m_0 samples, then a randomly selected *test input* is correctly classified with accuracy $J^* + \epsilon$, with probability $\geq 1 - \epsilon$.

The VC-dimension of a perceptron with n inputs is $n + 1$, and $J^* = 0$ if the data is linearly separable.

Training and Testing Error

The overall error is a sum of the training error and the "generalization error."

The training error depends on the quality of the algorithm used to minimize it.

The generalization error depends on the NN architecture.

A lot of analysis of the VC-dimension of NN architectures.

VC-Dimension of Neural Networks

What is the VC-dimension?

Suppose that we have a neural network architecture, and a collection of m vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

There are 2^m different ways to assign labels $y_i \in \{-1, 1\}$ for each $\mathbf{x}_i$.

Is the NN architectur rich enough that, for *each* of these 2^m possible labellings, the training error can be made zero by suitably adjusting the weights?

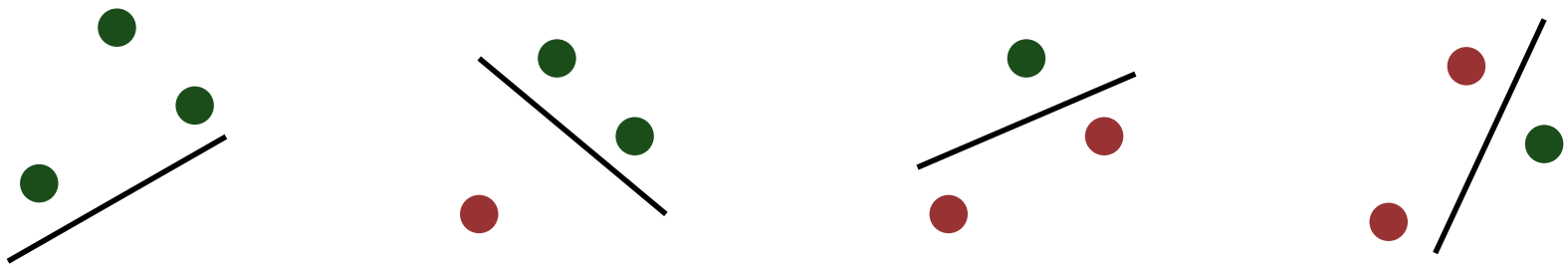
As m is increased, this becomes more and more difficult.

The VC-dimension is the **largest value** of m such that, for some collection of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$, everyone of the 2^m labellings can be achieved.

IIT Hyderabad

VC-Dimension of a Perceptron with Two Inputs

Take 3 points, which do not lie on a straight line. All $2^3 = 8$ labellings can be achieved by a suitably chosen perceptron.



Other four labellings are just mirror images of these. So the VC-dimension is ≥ 3.

VC-Dimension of a Perceptron with Two Inputs (Cont'd)

Now suppose we have four vectors in $\mathbb{R}^2$. There are two cases: (a) One of the vectors is inside the triangle formed by the other three. (b) None of the vectors is inside the triangle formed by the other three.

(a) Every half-plane that contains all three green dots must also contain the maroon dot. (b) This is just the XOR example.

VC-Dimension of a Perceptron with n Inputs

So the VC-dimension of a perceptron on $\mathbb{R}^2$ is three.

On $\mathbb{R}^n$, it is $n+1$. In fact, given *any* set of $n+1$ points that do not lie in a lower-dimensional subspace, *all* 2^{n+1} assignments of labels can be realized by a SVM.

This is the rationale behind higher-order SVMs – By enlarging the number of features, we ensure that the data is linearly separable (in the higher-dimensional space).