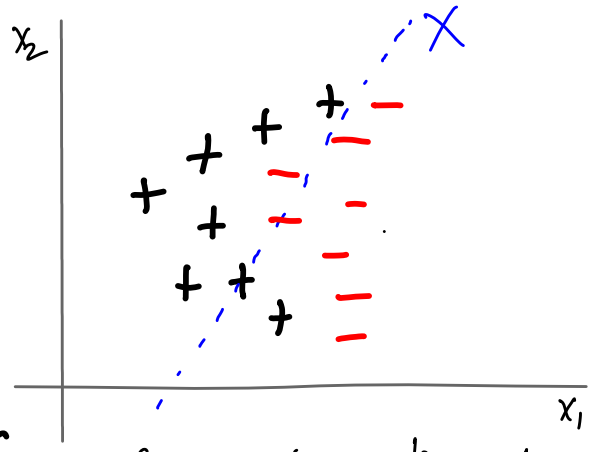
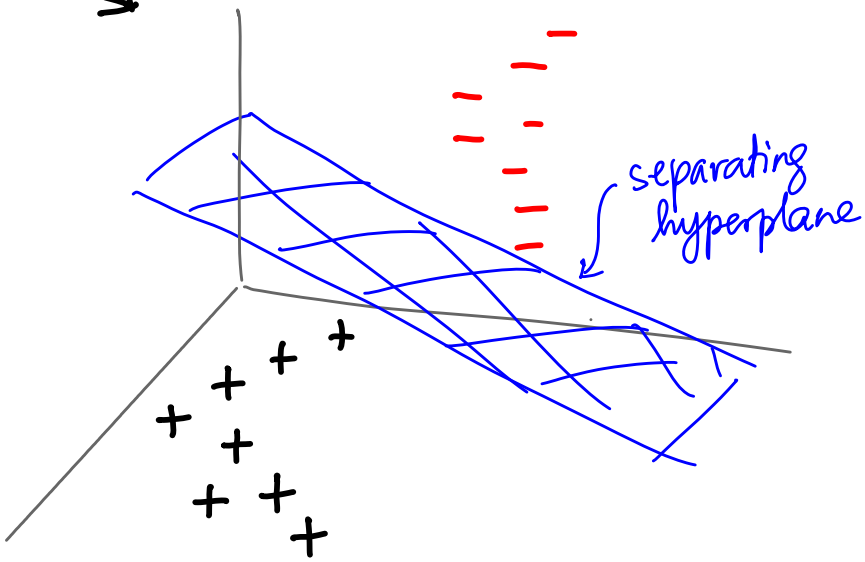
MATH 565: Lecture 23 (04/07/2026)

Today: * kernel SVM
* Computational graphs

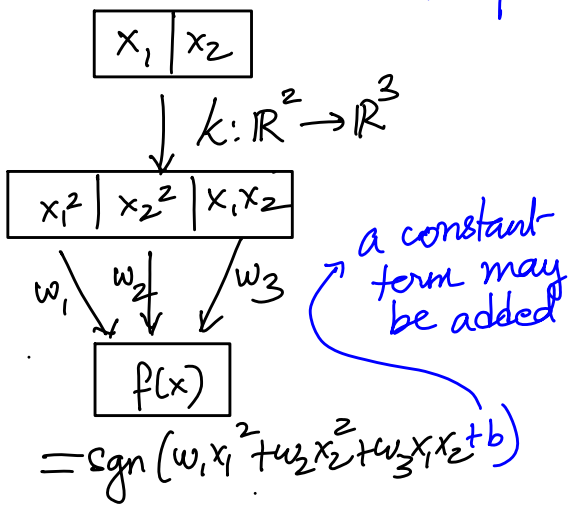
Recall Illustration of kernel SVM

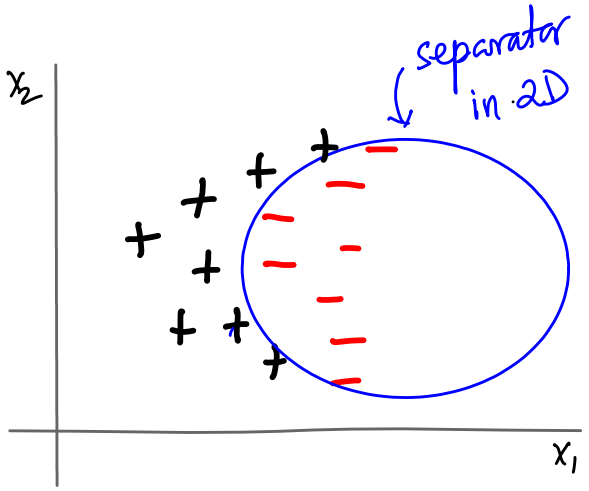


kernel transformation: $k(x_1, x_2)$



The kernel $k: \mathbb{R}^2 \to \mathbb{R}^3$ may have the following structure: (just for an example...)





The hyperplane separator in the kernel projected space ($\mathbb{R}^3$) may correspond to a non-linear separator in the input space ($\mathbb{R}^2$).

We "learn" w_1, w_2, w_3 (and b) so that the evaluated "output" function $\text{sgn}(w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + b)$ evaluates to the correct ± 1 values for most/all input points $(+, -)$.

Q: Why would going to a higher dimension work here?

VC Dimension (Vapnik-Chervonenkis)

For binary classification, the VC dimension measures the capacity of a hypothesis space $\mathcal{H}$ by counting how many (sets of) points it can "shatter".

Def A set of points $S = \{\bar{x}_i\}_{i=1}^{n}$ is shattered by a hypothesis space $\mathcal{H}$ if, no matter how you assign the ± 1 labels to S, there exists a $h \in \mathcal{H}$ that perfectly separates S.

When $|S| = n$, there are 2^n ways to assign ± 1 labels to S. Hence, to shatter S, the hypothesis space $\mathcal{H}$ must be flexible enough to realize all these configurations, i.e., be able to separate all of them.

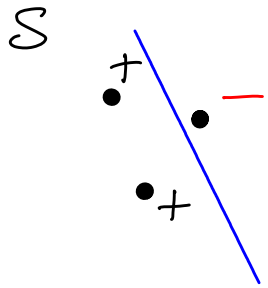
Def The VC dimension of a hypothesis space $\mathcal{H}$ is given as

$VC(\mathcal{H}) =$ maximum # points that can be shattered by $\mathcal{H}$.

If $VC(\mathcal{H}) = d$, then there is at least one set S of d points that can be shattered by $\mathcal{H}$. Also, no set of $d+1$ (or more) points can be shattered by $\mathcal{H}$.

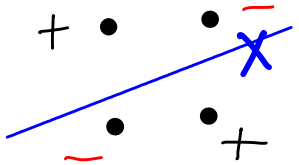
Example $\mathcal{H}$ = linear classifiers in 2D → separating lines

1. 3 points (non-collinear, in general position)



We can always draw a line separating the +'s and −'s.

2. 4 points (in general position)



$\mathcal{H}$ cannot shatter S: assign +'s to one diagonal pair and −'s to the other pair.

$\Rightarrow VC(\mathcal{H}) = 3.$

In general, in d dimensions, $VC(\mathcal{H}) = d+1$.

If $n > d+1$, then Cover's theorem (1965) states that the number of linear separators is

$$2\sum_{i=0}^{d}\binom{n-1}{i}.$$

$\Rightarrow$ The probability of linear separability $=$

$$\frac{1}{2^n}\cdot 2\sum_{i=0}^{d}\binom{n-1}{i}.$$

This probability $= 1$ if $n \le d+1$, and the probability goes to 0 rapidly when $n > 2(d+1)$ (as n increases).

In between $(d+1 < n \le 2(d+1))$, this probability increases with increasing d (for a given n) up to a certain point.

Hence the motivation to apply kernel tricks!

And the good part is that all properties of the linear (hinge) SVM apply to the nonlinear case obtained using a suitable kernel, i.e., one that has nice properties to start with.

Instead of $\bar{x}_i^T \bar{x}_j$ in (D), we can use $k(\bar{x}_i, \bar{x}_j)$. The commonly used kernels include the following.

1. polynomial:
$$k(\bar{x}, \bar{x}_i) = \alpha(\bar{x}^T \bar{x}_i)^d + \beta$$
degree d and additional parameters α, β to be chosen by user

2. Radial Basis function (RBF) with Gaussian kernel:
$$k(\bar{x}, \bar{x}_i) = e^{-(\|\bar{x} - \bar{x}_i\|^2 / c)} \quad \text{for } c > 0.$$

3. Neural networks with tanh activation: (hyperbolic tangent)
$$k(\bar{x}, \bar{x}_i) = \tanh(\alpha \bar{x}^T \bar{x}_i + \beta) \quad \alpha > 0, \ \beta \in \mathbb{R}.$$

The H-SVM dual problem (D) is

$$\min_{0 \le \bar{\alpha} \le \bar{C}} L_D = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{\bar{x}_i^T \bar{x}_j}_{k} - \sum_{i=1}^{n} \alpha_i$$

When using a kernel k, we simply solve

$$\min_{0 \le \bar{\alpha} \le \bar{C}} L_D(k) = \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \underline{k(\bar{x}_i, x_j)} - \sum \alpha_i$$

If kernel k is PD, then $Q_k = \left[y_i y_j k(\bar{x}_i, \bar{x}_j) \right]_{ij}$ is PD, and (D) is a convex minimization problem.

Since we're increasing the dimension, we are allowing the possibility of overfitting. Hence, we usually apply kernel SVMs under the setting of cross validation.

Optimization on Computational Graphs (CGs)

A computational graph $G_c = (\mathcal{N}, \mathcal{E})$ has node set $\mathcal{N}$ and edge set $\mathcal{E}$.

node ≡ unit of computation

edge ≡ relation between nodes
An edge can directed or undirected

Many practical CGs are directed acyclic graphs (DAGs), but CGs with cycles are possible.

Each node in a CG is associated with a variable (can be a vector), which can be continuous, discrete, or probabilistic (is assumed to be continuous by default).

Edges can have parameters to be learned from data.

DACG: Directed Acyclic Computational Graph. $G_c(\mathcal{N}, \mathcal{E})$

$\mathcal{N}$ = set of computational nodes, each associated with a variable

$\mathcal{E}$ = set of directed edges connecting nodes: $(i,j) \in \mathcal{E}$ for $i,j \in \mathcal{N}$. May have a learnable parameter w_{ij}.

The variable value at node $i \in \mathcal{N}$, $v(i)$, is computed as follows:

$v(i)$ is { fixed externally, if i is an input node;
computed as a function of $v(j)$'s for $(j,i) \in \mathcal{E}$ with w_{ji}'s when relevant.

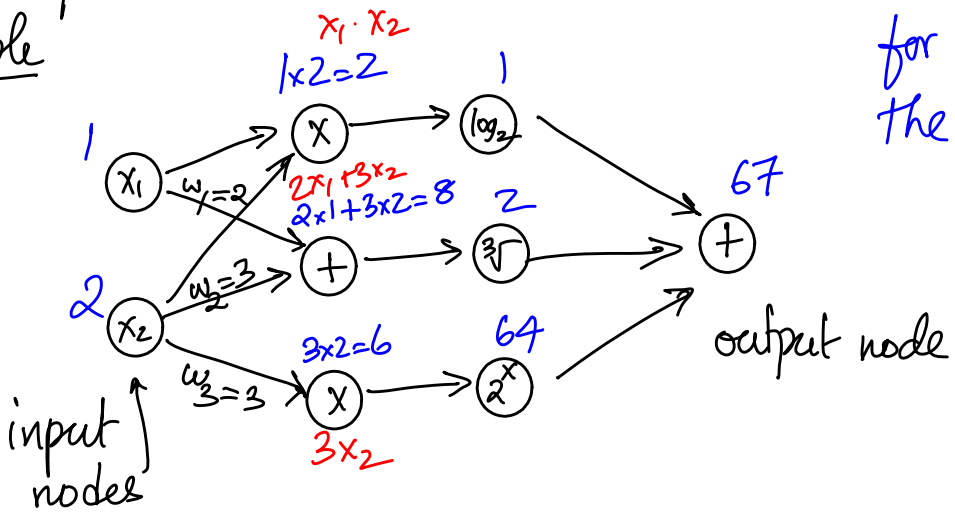
N consists of input, hidden, and output nodes

- input: external inputs to G_C
- hidden: intermediate nodes
- output: final outputs

Hidden and output nodes (i) compute $v(i)$ as a relatively simple local function of incoming variables $(v(j)$ for $(j,i) \in E)$ and weights w_{ji}.

G_C defines a global function $f_G : \mathbb{R}^n \to \mathbb{R}^m$ for n input nodes and ouput of dimension m.

Example



for $x_1 = 1$, $x_2 = 2$ (input) the output is 67

What is the global f_G here?

$$f_G(\bar{x}) = f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \log_2(x_1 \cdot x_2) + \sqrt[3]{2x_1 + 3x_2} + 2^{3x_2}$$

for $\bar{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $f_G(\bar{x}) = 67$.

But if training data has $\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}, 50\right)$ ($\neq 67$), we can adjust $\bar{w}$ to get $f\left(\begin{bmatrix} 1 \\ 2 \end{bmatrix}\right) \to 50$.

Finding the $\bar{w}$ that matches $f(\bar{x})$ for each input $\bar{x}$ is posed as an optimization problem.