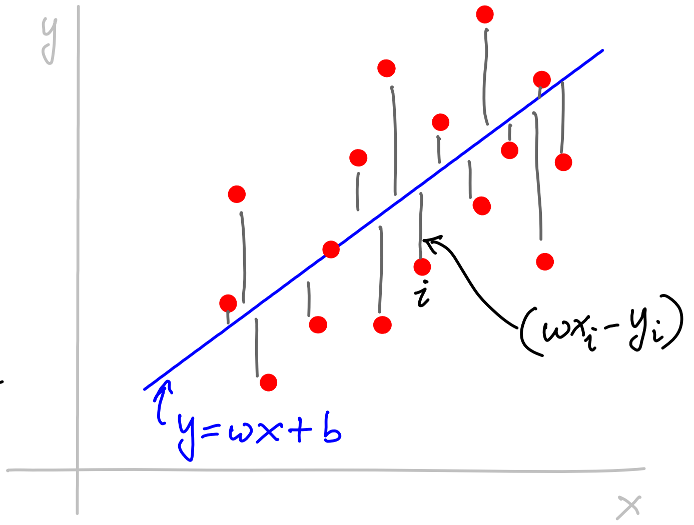
MATH 565: Lecture 2 (01/15/2026)

Today:
* regression and extensions
* Regularization
* Support vector machines (SVM)

Recall: Linear regression: Given $D_{n\times d}$, $\bar{y}\in\mathbb{R}^n$, the goal is to find weight vector $\bar{w}\in\mathbb{R}^d$ such that $\bar{y}\approx D\bar{w}$. In detail, we want to minimize the loss function

$$J=\frac{1}{2}\|D\bar{w}-\bar{y}\|^2$$
$$=\frac{1}{2}\sum_{i=1}^{n}(y_i-\bar{w}^T\bar{x}_i)^2$$



In 1D, linear regression fits a line $y=wx+b$ through a given set of n points (x_i,y_i) such that the sum of the squared "errors" $\sum_{i=1}^{n}(wx_i-y_i)^2$ is minimized.

J is a convex function and hence has a unique minimum. The first order optimality condition (corresponding to $f'(x)=0$ in 1D) is given by $\nabla J=\bar{0}$

$$J=\frac{1}{2}(D\bar{w}-\bar{y})^T(D\bar{w}-\bar{y})=\frac{1}{2}\left[\bar{w}^TD^TD\bar{w}-2\bar{w}^TD^T\bar{y}+\bar{y}^T\bar{y}\right]$$

So, $\nabla J=D^TD\bar{w}-D^T\bar{y}=0$

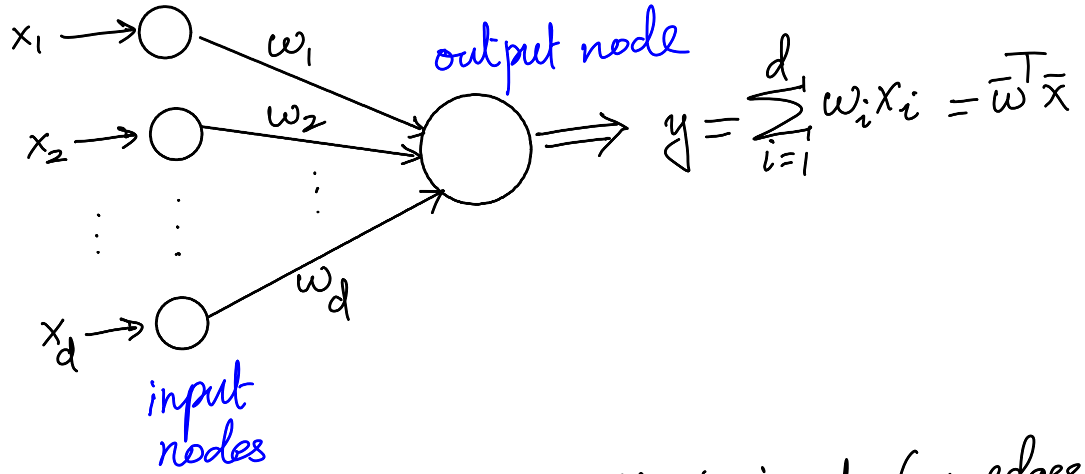
$$\Rightarrow\quad D^TD\bar{w}=D^T\bar{y}\quad\text{———}\ (1)$$

$$\Rightarrow\quad \bar{w}=(D^TD)^{-1}D^T\bar{y}\quad\text{———}\ (2)$$

A Quick aside: Optimization on Computational Graphs

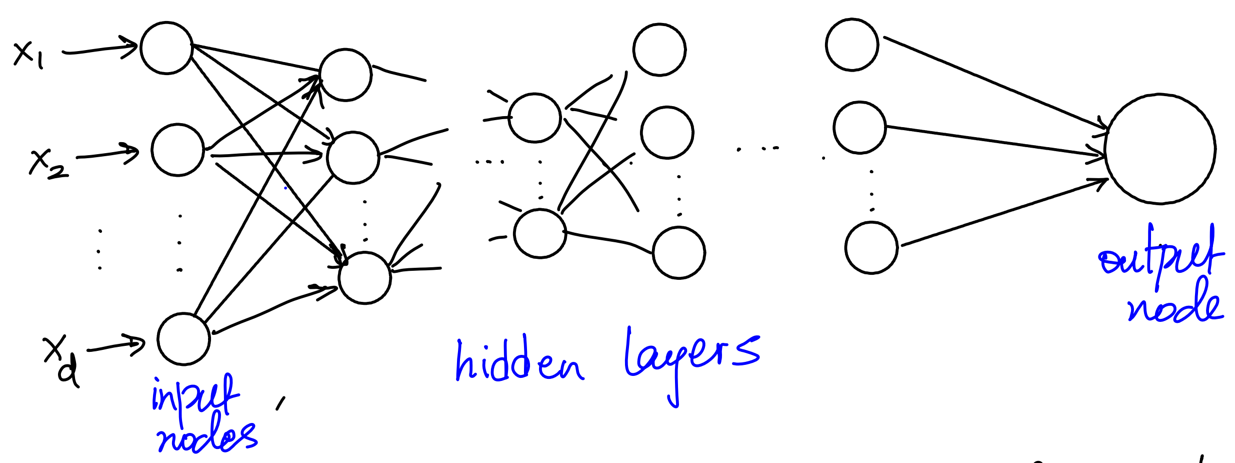
Many machine learning problems can be equivalently posed as problems on computational graphs. Informally, the qualifier "computational" here means the vertices and edges in these graphs are endowed with more general computational steps than, e.g., the weights/costs and edge capacities seen in network flow problems. While we will revisit this topic in detail later on, we give a graph that can be used to model linear regression.

Linear regression graph model



In general, each node takes in all its inputs (on edges coming in to it) and combines them somehow to send output(s) to all nodes to which it is connected. For linear regression, there is one input node for each x_i that uses the weight w_i on its output edge to send $w_i x_i$ to the single output node. And this output node sums up all its inputs to output $\sum_i w_i x_i = \bar{w}^T \bar{x}$.

But the computational graphs could be far more general, with multiple layers of nodes and edges connecting nodes in each layer to many or all nodes in nearby (not necessarily only adjacent) layers.



The functions and data captured on the nodes and edges can also be quite general. As one can imagine, such graphs could capture fairly complicated functions and restrictions.
How does one take gradients on computational graphs?.
We use back propagation. More on this later...

Back to Regression...

Solving a system of linear equations can be considered as a version of linear regression. If there is a solution, then we get $J=0$ (zero loss). But if there is no (exact) solution, we could still get a "best fit" solution (in the least squares sense).

Recall how we found $\bar{w}$:

$$D^T D \bar{w} = D^T \bar{y} \quad \text{——} \quad (1)$$

$$\bar{w} = (D^T D)^{-1} D^T \bar{y} \quad \text{——} \quad (2)$$

How do we compute $\bar{w}$ efficiently (for large instances)?
Computing $(D^T D)^{-1}$ can be costly, especially when n and d are huge (or even large). We usually use QR decomposition of the data matrix D.

$$D = QR \quad \text{——— (3)}$$

where Q is $n \times d$ with orthonormal columns and R is $d \times d$ upper triangular.

Hence, $Q^T Q = I_d$ (identity matrix).

Using (3) in (1) gives

$$R^T \underbrace{Q^T Q}_{I} R \bar{w} = R^T Q \bar{y}$$

$$\Rightarrow \quad (R^T)^{-1} \left(R^T R \bar{w} = R^T Q \bar{y} \right) \quad \text{——— (4)}$$

$$\Rightarrow \quad R \bar{w} = Q^T \bar{y}$$

This is a triangular system and can be solved by back substitution.
↳ as R is upper triangular

This computation assumes that $D^T D$ is invertible, which may not always hold — e.g., when $n < d$ (we do not have enough samples for the given number of dimensions). This is similar to the setting in which a linear system $A\bar{x} = \bar{b}$ has infinitely many solutions. In this setting, we run the danger of overfitting the (training) data. And when that happens, the trained model may not generalize as well to external test data.

To resolve this situation, we need to do regularization. Intuitively, we need to limit the # w_j's that are non-zero . . .

Tikhonov Regularization

We modify the loss function to

$$J_W = \frac{1}{2}\|D\bar{w} - \bar{y}\|^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

for the regularization parameter $\lambda > 0$.

Our goal is to somehow enforce several or many of the $w_j = 0$ (i.e., only a few of them are non-zero). One approach to achieve this goal is to add an explicit constraint of the form

0-norm (# non-zero entries) $\longrightarrow$ $\|\bar{w}\|_0 \leq k$ for $k << d$.

But this constraint is hard to enforce, and also destroy the nice structure of J (convexity). Instead, we add the squared norm penalty $\frac{\lambda}{2}\|\bar{w}\|^2$ as a regularization term. It achieves the same goal — it forces many $w_j = 0$ in the optimal solution. But it is also a strongly convex function ($\lambda > 0$), and hence J_W is so as well ($\frac{1}{2}\|D\bar{w} - \bar{y}\|^2$ term is convex), implying that J_W has a unique optimal solution.

$w_j \neq 0$ unless absolutely needed.

Setting $\nabla J_W = \bar{0}$ for optimality gives

$$(D^T D + \lambda I)\bar{w} = D^T \bar{y}$$

$$\Rightarrow \quad \bar{w} = (D^T D + \lambda I)^{-1} D^T \bar{y}$$

We'll talk about more details of this regularized model later on...

Modifications of J_W

In $J_W = \frac{1}{2}\|D\bar{w}-\bar{y}\|^{\cancel{2}\,1} + \frac{1}{2}\lambda\|\bar{w}\|^{\cancel{2}\,1}$,

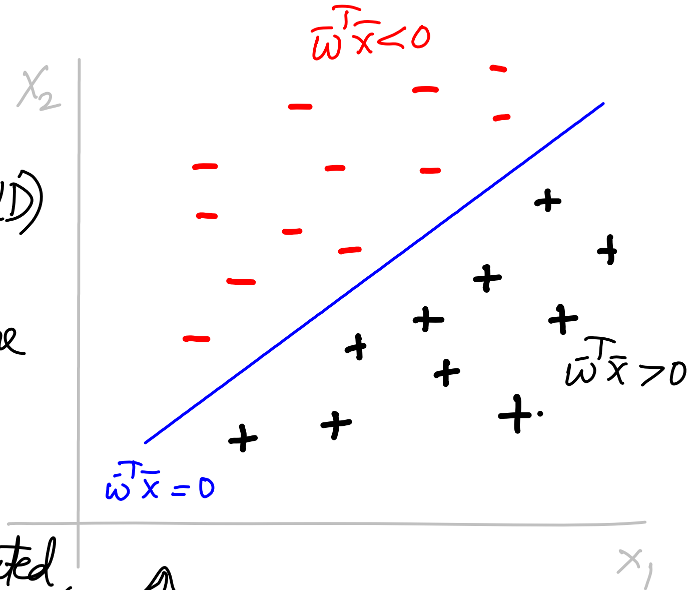
also called "fit"

one could consider modifying either (loss/error or regularization) term to an L_1-norm term (instead of L_2). There are advantages and disadvantages for using L_1 terms. L_1-norms are piecewise linear, and when they appear in minimization objective functions as here, they can be easily linearized, making computations more efficient. At the same time, the L_2-terms usually have smoother behavior than their L_1-counterparts.

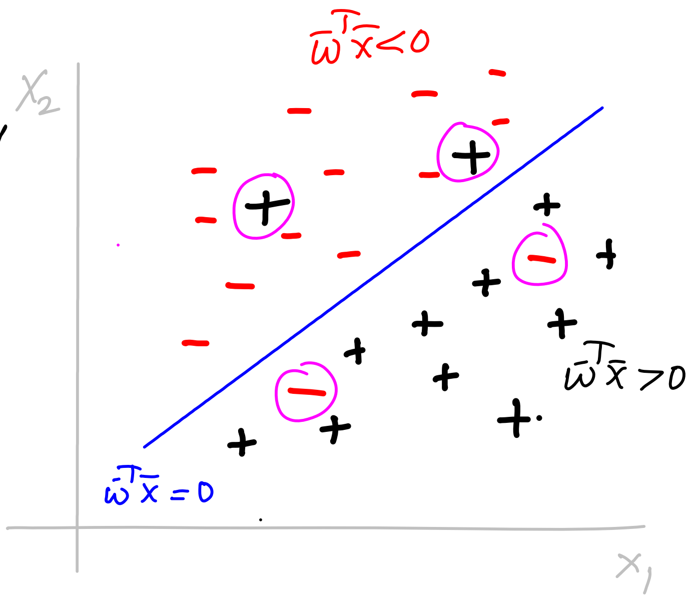
Binary Classification

We now consider the binary classification problem in a bit more detail in the context of regression. Here, we have $y_i \in \{-1, 1\}$, and the goal is to "separate" the +1 instances "as best as possible" from the −1 instances.

A direct generalization of linear regression is to find $\bar{w}$ s.t. $\bar{w}^T\bar{x} = 0$ is the separating line (in 2D) or hyperplane (in d-dim), with the $\bar{w}^T\bar{x} > 0$ side capturing all the +1 instances and the other side containing all the −1 instances. This may work well in well-separated, i.e., easy, instances as illustrated here.



But when the +1 and −1 instances are not well-separated (as seen here), we would want to modify J_w (loss function) to capture the violations of good separations. Here is one approach. We write



$$J_w = \frac{1}{2}\sum_{i=1}^{n}(y_i - \bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2.$$

Using the fact that $y_i^2 = 1$ (as $y_i \in \{-1, 1\}$), we write

$$J_w = \frac{1}{2}\sum_{i=1}^{n} y_i^2(y_i - \bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}(y_i^2 - y_i\bar{w}^T\bar{x}_i)^2 + \frac{\lambda}{2}\|\bar{w}\|^2$$

$$= \frac{1}{2}\underbrace{\sum_{i=1}^{n}(1 - y_i\bar{w}^T\bar{x}_i)^2}_{} + \frac{\lambda}{2}\|\bar{w}\|^2 \quad \text{(again, using } y_i^2 = 1\text{)}.$$

The first term will penalize violations of good separation. At the same time, it will also penalize cases that are "extremely" well-separated! For instance, if $y_i = +1$ and $\bar{w}^T\bar{x}_i = 100$, which is a very good separation, the term in J_w will be $(1 - 1\cdot 100)^2 = (99)^2 = 9801$, which is huge!

Hence, we want to set the contribution of well-separated instances (to J_w) as zero — this is the idea used in support vector machines (SVMs)!

Support Vector Machines (SVM)

We set

$$J_{L_2\text{-svm}} = \frac{1}{2}\sum_{i=1}^{n}\left(\max\{0, [1-y_i(\bar{w}^T\bar{x}_i)]\}\right)^2 + \frac{1}{2}\|\bar{w}\|^2$$

A (computationally) better option is to consider an L_1-SVM loss. We rewrite this L_1-SVM problem as a slightly modified optimization problem, as presented below. For many people working in optimization, this SVM model is a natural way to start exploring machine learning problems, as it is a convex optimization problem.

$$\min_{\bar{w}, \bar{\varepsilon}, b} J_{svm} = C\sum_{i=1}^{n}\varepsilon_i + \frac{1}{2}\|\bar{w}\|^2$$

$$\text{s.t.} \quad y_i(\bar{w}^T\bar{x}_i + b) \geq 1 - \varepsilon_i, \quad i=1,\dots,n$$

$$\varepsilon_i \geq 0$$

We will check the details of this model in the next lecture...