MATH 565: Lecture 1 (01/13/2026)

Today: * logistics, syllabus, ...
* problems in ML, optimization for them

This is Optimization for Machine Learning (Math 565)

I'm Bala Krishnamoorthy (call me Bala).

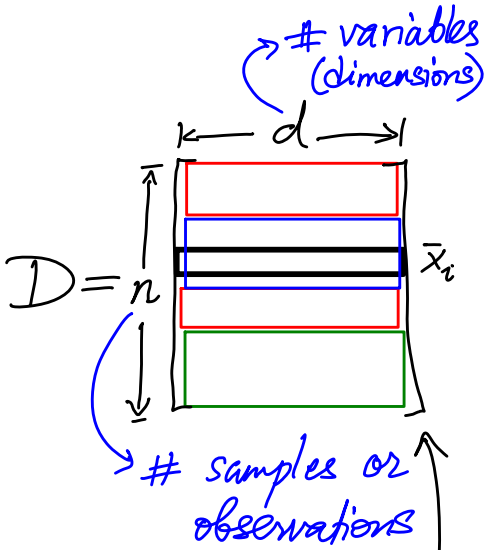
We will **not** use Canvas. All materials for the class will be posted on the course web page: https://bala-krishnamoorthy.github.io/Math565.html

Download the book PDF (accessible via WSU Libraries)!

* Check the syllabus on the course web page.
* homework assignments will be posted at least a week before its due.

Machine Learning Problems

1. **Clustering** We are given a data matrix that is $n \times d$ for n observations (or samples) each having values for d variables (dimension=d). The i^{th} observation is denoted $\bar{x}_i$ (d-vector), which forms the i^{th} row of D (when written as $\bar{x}_i^T$).

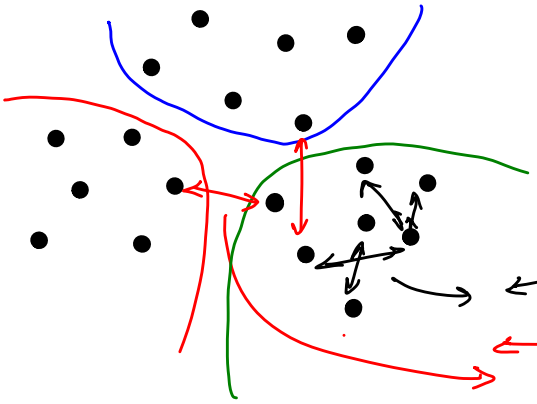


The goal of clustering is to divide the n observations into k disjoint subsets or clusters. The # clusters k may be user-defined, e.g., in k-means clustering, or may be determined as part of the clustering method. Equivalently, the rows of D are to be divided into k disjoint sets (e.g., into red, blue, and green clusters as illustrated here).

Notation: A, X, Y: matrices or sets (uppercase letters)
$\bar{x}, \bar{y}, \bar{\alpha}, \bar{\theta}$: vectors (lower case letters with a bar)
x, β, γ, a: scalars (lower case letters)

Another ("metric") way to think about clustering is illustrated in 2D.

$\bullet \bar{x}_i$



division into 3 clusters (red, blue, green)

$\longleftrightarrow$: within cluster distances
$\longleftrightarrow$: distance to nearest point outside cluster

One objective function used in clustering is to minimize the sum of all within cluster pairwise distances while also maximizing the sum of distances of each point to the nearest point outside its cluster.

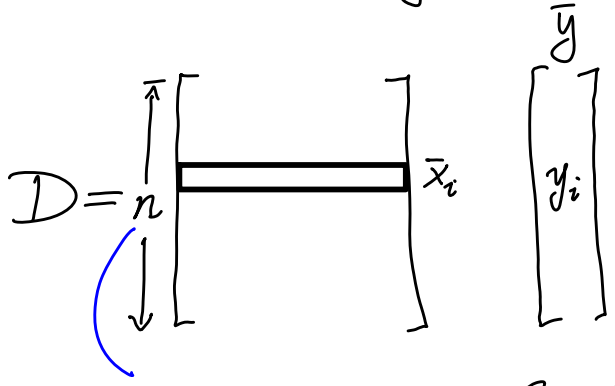
An example: Each $\bar{x}_i$ represents a consumer, and the entries are the $ amounts they spent on fruits, meat, toiletteries, or other items (in a retail or grocery store). The store may be interested in the clusters to target ads for specific products.

Clustering is considered an unsupervised learning problem, since only the data ($\bar{x}_i$) is used without the membership information (of which cluster each $\bar{x}_i$ belongs to).

As opposed to classification/regression problems being introduced now, which are supervised learning problems.

2. Classification

Here, apart from the $n \times d$ data matrix D, we are given also an n-vector $\bar{y}$ of labels for each observation.



For instance, $y_i \in \{-1, 1\}$ (or $\{0, 1\}$) to capture a YES/NO aspect of each observation, e.g., whether customer i is possibly responsive to ads. In another instance, $y_i \in \{R, G, B\}$ (3 colors). Note that such labels are categorical, and it is not obvious how to represent these labels using numerical values. The YES/NO aspect could still be modeled as a continuous numerical value coupled with a cut-off value (to determine YES).

The goal is to build a "model" on D and $\bar{y}$, which is the training set, and use that model to predict the $\bar{y}_t$ values for an external test data set D_t ($n_t \times d$) for which $\bar{y}_t$ is not known.

The model can be described as

$$y_i \approx f(\bar{x}_i).$$

The function to be learned, f, is often parametrized using a weight vector $\bar{w}$, and hence we write

$$y_i \approx f_{\bar{w}}(\bar{x}_i)$$

For instance, in the case of binary classification with $y_i \in \{-1, 1\}$, we could take

$$y_i = \text{sign}\{f_{\bar{w}}(\bar{x}_i)\}$$

Q. How does one choose $\bar{w}$?

We usually pick $\bar{w}$ such that the mismatch between y_i and $f_{\bar{w}}(\bar{x}_i)$, i.e., the "loss", is minimized. The form of f and this loss function are often carefully chosen/constructed to solve

$$\min_{\bar{w}} \sum_{i=1}^{n} \ell\left(y_i - f_{\bar{w}}(\bar{x}_i)\right)$$

ℓ → loss function

Such classification tasks are considered supervised learning, since we are guided by the y_i's (known labels). The goal is to use the built function $f_{\bar{w}}(\bar{x})$ to predict the labels $\bar{y}_t$ for an external or test data set D_t $(n_t \times d)$ (for which the labels are unknown). Since D_t is not used in the training process, such predictions for test sets are called as generalizations.

The step where we want to minimize a loss function makes classification an optimization problem! Before considering details of optimization, we present a simpler ML problem: regression.

3. Regression

We're given D $(n \times d)$ just as in classification, but now, $y_i \in \mathbb{R}$, i.e., it is a numerical value (rather than a label).

The simplest version of regression is linear regression, where the task is to fit a line through the given set of n points (in 2D, and a plane in d-dimensions).

$$y_i = f_{\bar{w}}(\bar{x}_i) = \bar{w}^T \bar{x}_i = \bar{x}_i^T \bar{w}$$

Equivalently, $\bar{y} = D\bar{w}$.

And the loss function usually used is the sum of squared errors:

$$J = \frac{1}{2} \|D\bar{w} - \bar{y}\|^2.$$

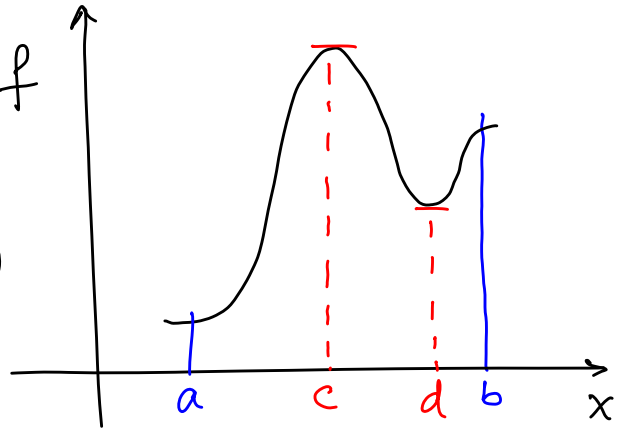
Hence, linear regression becomes the optimization problem:

$$\min_{\bar{w}} \frac{1}{2} \|D\bar{w} - \bar{y}\|.$$

Optimization in 1D Calculus

Find $\min f(x)$ for $a \le x \le b$

* We set $f'(x) = 0$ to find critical points.
* $f''(x) > 0$: local minima (e.g., d)
  $f''(x) < 0$: local maxima (e.g., c)
  $f''(x) = 0$: saddle points



Compare the function values at local minima with those at the end points (a and b) to determine the true ("global") minimum (@ $x = a$ in the figure here).

Optimization in d-dimensions

In general, we consider problems of the form

$$\begin{aligned} &\min f(\bar{x}) \\ &\text{s.t.} \quad \bar{g}(\bar{x}) \leq \bar{0} \\ &\quad\;\; \bar{h}(\bar{x}) = \bar{0} \end{aligned}$$

Under appropriate assumptions, we can specify optimality conditions (local by default, global when the functions are "nice"), e.g., Karush-Kuhn-Tucker (KKT) conditions.

Corresponding to $f'(x)=0$, we have $\nabla f = \bar{0}$ for

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}, \text{ the gradient of } f.$$

Back to linear regression:

Recall: $\min_{\bar{w}} J = \frac{1}{2} \| D\bar{w} - \bar{y} \|^2$

$$\Rightarrow \nabla J = D^T D \bar{w} - D^T \bar{y} = \bar{0}$$

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

closed form expression!

It turns out that this critical point is the (unique) global minimizer of J. In this sense, linear regression is perhaps the easiest ML problem!

But more generally, we do not get closed form solutions in this form. In fact, we usually use gradient descent to iteratively update $\bar{w}$:

We choose an initial $\bar{w}$ in some way, e.g., randomly. Then, in each iteration, we update

$$\bar{w} \leftarrow \bar{w} - \alpha \nabla J(\bar{w})$$

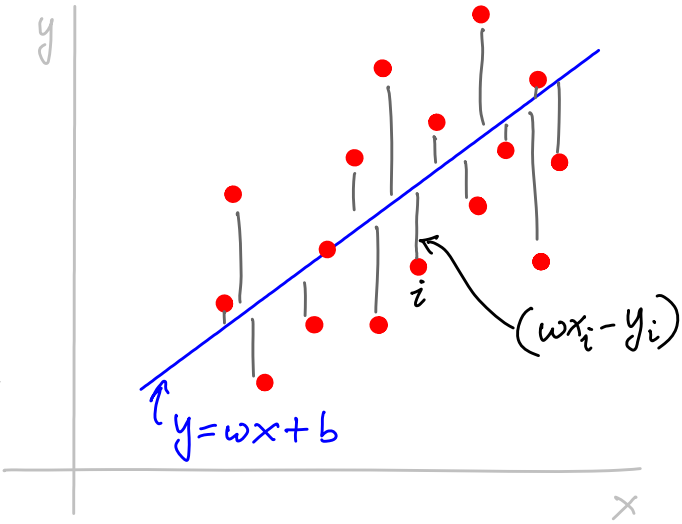
α is the step size, also called the learning rate. We usually need to choose α carefully to ensure the iterations converge. We also have to make assumptions about the function f to guarantee efficient convergence of this gradient-descent method.

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

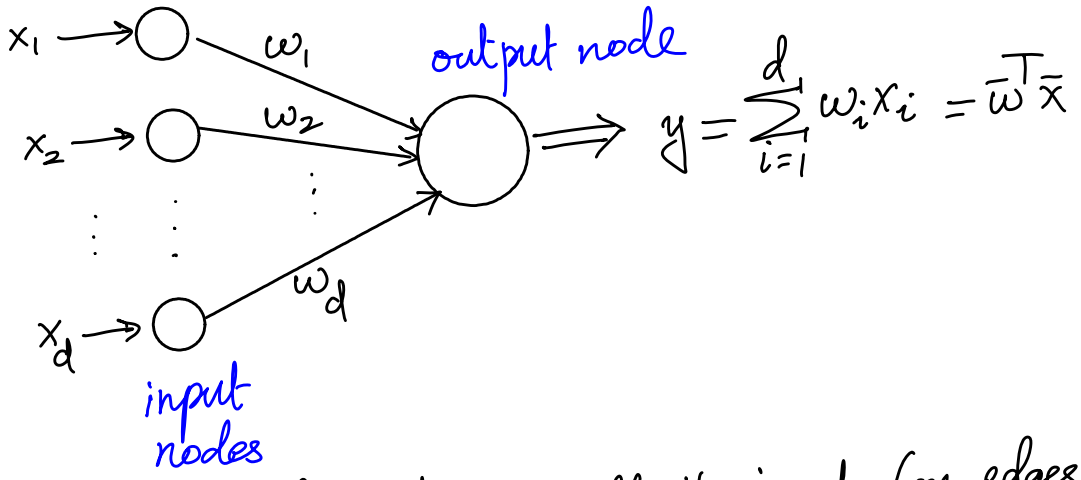
$$\Rightarrow \quad D^TD\bar{w}=D^T\bar{y} \qquad (1)$$

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

A Quick aside: Optimization on Computational Graphs

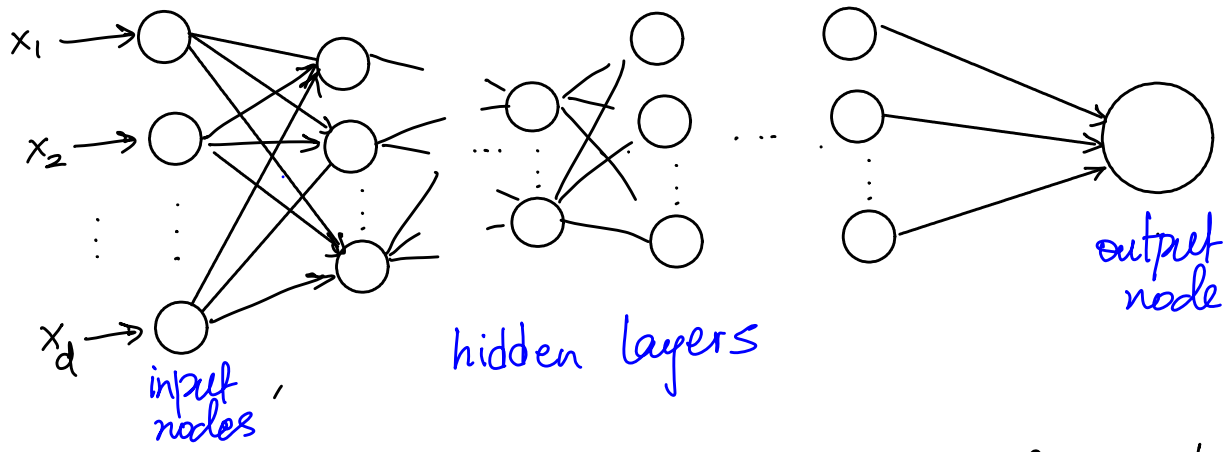
Many machine learning problems can be equivalently posed as problems on computational graphs. Informally, the qualifier "computational" here means the vertices and edges in these graphs are endowed with more general computational steps than, e.g., the weights/costs and edge capacities seen in network 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 $\sum_i w_i x_i = \vec{w}^T \vec{x}$.

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 $\bar{w}$:

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

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

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

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

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

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

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 $w_i = 0$ (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

$\|\bar{w}\|_0 \leq k$ for $k << d$.

(0-norm (# non-zero entries))

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 $w_i = 0$ 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.

($w_i \neq 0$ unless absolutely needed.)

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

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

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

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

Modifications of J_w

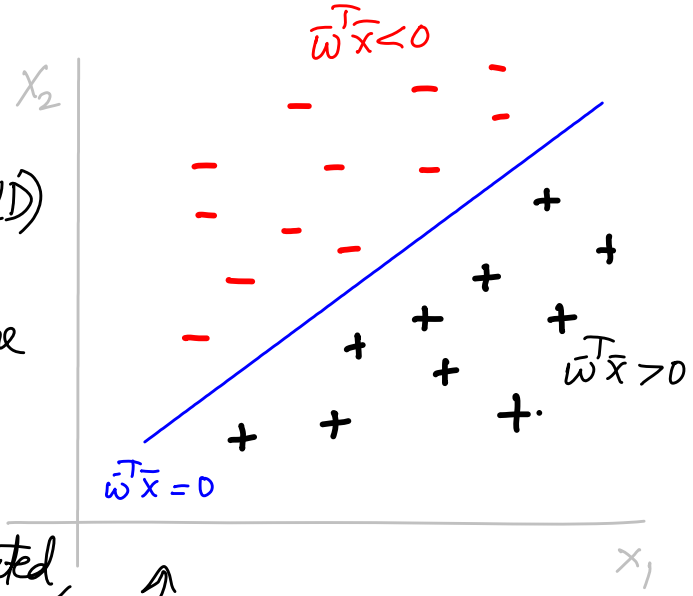
In $J_w = \frac{1}{2}\|D\bar{w} - \bar{y}\|^{\frac{2}{2}} + \frac{1}{2}\lambda\|\bar{w}\|^{\frac{2}{2}}$, also called "fit"

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.

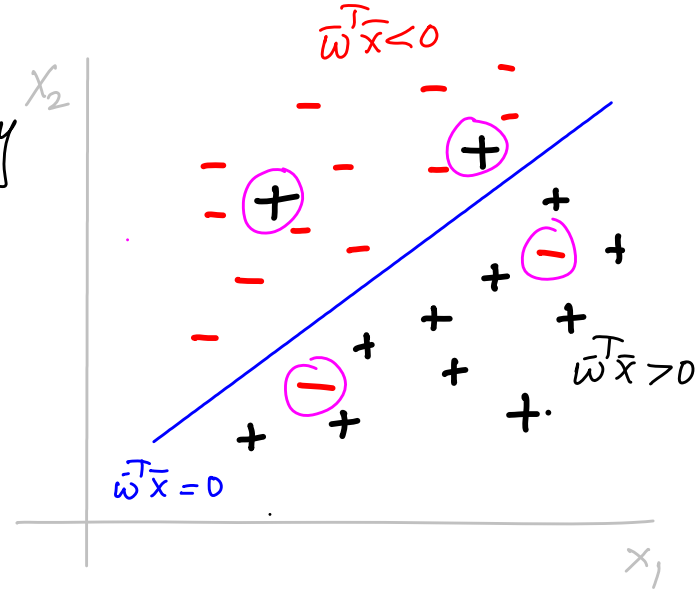
Binary Classification

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

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

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

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

(again, using $y_i^2 = 1$).

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

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

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.

$$\min_{\bar{w}, \bar{\xi}, b} \; J_{SVM} = C\sum_{i=1}^{n}\bar{\xi}_i + \frac{1}{2}\|\bar{w}\|^2$$

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

$$\bar{\xi}_i \geq 0$$

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

MATH 565: Lecture 3 (01/20/2026)

Today:
* more on SVM
* Taylor expansion
* local optimality conditions

Recall: SVM model:

$$\min_{\bar{w},b,\bar{\xi}} \; J = C\sum_{i=1}^{n}\xi_i + \frac{1}{2}\|\bar{w}\|^2 \qquad (C>0)$$

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

$$\xi_i \geq 0 \quad \forall i$$

Let's examine the unified (main) constraints in detail:

Recall: the model tries to get $\bar{w}^T\bar{x}_i+b\geq 1$ when $y_i=+1$ and $\bar{w}^T\bar{x}_i+b\leq -1$ when $y_i=-1$.

ξ_i: measures by how much the i^{th} sample violates well-separatedness

When $y_i=+1$, we get $\bar{w}^T\bar{x}_i+b\geq 1-\xi_i$

For instance, if $\bar{w}^T\bar{x}_i+b=0.7$, then $\xi_i\geq 0.3$ is needed;
but if $\bar{w}^T\bar{x}_i+b=2$, then $\xi_i=0$ works, and will be set to this value because of the $C\xi_i$ term in the objective function (recall, $\xi_i\geq 0$).

On the other hand, if $y_i=-1$, the constraint becomes

$$\bar{w}^T\bar{x}_i+b\leq -1+\xi_i.$$

For instance, if $\bar{w}^T\bar{x}_i+b=-3$, $\xi_i=0$ in the optimal solution.
But if $\bar{w}^T\bar{x}_i+b=0.5$, we need $\xi_i\geq 1.5$ for the constraint to hold.

Technically, we should be including the affine variable b in the regularizing term:

affine variable = or intercept

$$\min_{\bar{w}, b, \bar{\xi}} \; J = C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \left\| \begin{bmatrix} \bar{w} \\ b \end{bmatrix} \right\|^2$$

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

$$\xi_i \geq 0 \;\; \forall i$$

Another equivalent notation is to write $\bar{w} = [w_0 \; w_1 \ldots w_d]^T$, and write the model as

$$\min_{\bar{w}, \bar{\xi}} \; J = C \sum_{i=1}^{n} \xi_i + \frac{1}{2} \|\bar{w}\|^2$$

$$\text{s.t.} \quad y_i \left(\bar{w}^T \begin{bmatrix} 1 \\ \bar{x}_i \end{bmatrix} \right) \geq 1 - \xi_i, \quad i = 1, \ldots, n,$$

$$\xi_i \geq 0 \;\; \forall i.$$

Note how the well-separated instances do not contribute to the objective function — irrespective of the extent of their well-separatedness. At the same time, instances that are not well-separated do incur a loss, i.e., they add to the loss function. Furthermore, the amount of this loss is more when the instance violates well-separatedness more.

We will study SVM models in detail later on...

Taylor Expansion

In one dimension, the Taylor expansion of $f(x)$ at $x=a$ is given by

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \cdots + \frac{(x-a)^r}{r!}\frac{d^r f(x)}{dx^r}\bigg|_{x=a} + \cdots$$

When $|x-a|$ is small, we can take

$$f(x) \approx f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a)$$

as a reasonable and accurate representation of $f(x)$.

In d-dimensions, the Taylor expansion of $f(\bar{x})$ at $\bar{x}=\bar{a}$ is

$$f(\bar{x}) = f(\bar{a}) + \sum_{i=1}^{d}(x_i - a_i)\left[\frac{\partial f}{\partial x_i}\right]\bigg|_{\bar{x}=\bar{a}} + \sum_i\sum_j \frac{(x_i-a_i)(x_j-a_j)}{2!}\left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]\bigg|_{\bar{x}=\bar{a}} + \cdots$$

Equivalently,

$$f(\bar{x}) = f(\bar{a}) + [\bar{x}-\bar{a}]^T \nabla f(\bar{a}) + [\bar{x}-\bar{a}]^T Hf(\bar{a})[\bar{x}-\bar{a}] + \cdots$$

for gradient $\nabla f(\bar{a})$ and Hessian $Hf(\bar{a})$ of $f(\bar{x})$ at $\bar{x}=\bar{a}$.

We will employ Taylor series approximations of functions for multiple purposes in this class – both for devising efficient algorithms and for deriving theoretical results, i.e., proofs. Despite its somewhat simple form, the Taylor series expansion proves to be a powerful tool!

We now present local optimality conditions first in 1D and then in d-dimensions in general. We will use the Taylor series expansion to justify them.

Local Optimality Conditions in 1D

Lemma 1 Let $f: \mathbb{R} \to \mathbb{R}$ be a function. Then $f(x)$ is a minimum value at $x = x_0$ with respect to its immediate locality if

$f'(x_0) = 0$ and → (first order optimality condition)

$f''(x_0) > 0$ → (second order optimality condition)

in a small enough neighborhood of $x = x_0$

Proof Consider the Taylor expansion of f at x_0 for $x = x_0 + \Delta$: $(\Delta \in \mathbb{R})$

$$f(x_0+\Delta) \approx f(x_0) + \Delta f'(x_0) + \frac{\Delta^2}{2} f''(x_0)$$

($\Delta f'(x_0) = 0$, $\frac{\Delta^2}{2} f''(x_0) > 0$)

We can take $|\Delta| \ll 1$ small enough such that

$$\left|\frac{\Delta^2}{2} f''(x_0)\right| \gg \left|\sum \frac{\Delta^r}{r!} f^{(r)}(x_0)\right|$$

→ the second order term dominates all other higher order terms taken together

$\Rightarrow$ under first and second order optimality conditions,

$$f(x_0+\Delta) > f(x_0) \quad \text{for } |\Delta| \text{ small enough} \qquad \square$$

The first order condition ($f'(x) = 0$) is typically solved using **gradient descent**.

(initialization) Step 0. Start $x = x_0$ (can be chosen randomly)

do

Step k. $x_k \leftarrow x_{k-1} - \alpha f'(x_{k-1})$ (in general, $x \leftarrow x - \alpha f'(x)$)

while $|x_k - x_{k-1}| > \epsilon$.

Here $\alpha > 0$ is the learning rate (also called the step size) and $\epsilon \geq 0$ is a convergence tolerance.

Note that we are changing x by $\delta x = -\alpha f'(x)$. In fact, we are changing x along the "steepest descent" direction — which is trivial in 1D as we have only two options (↑ or ↓), but is nontrivial in d dimensions ($d \geq 2$) when we could have infinitely many options.

Our goal is to decrease $f(x)$ (we are minimizing it). Each step of gradient descent is guaranteed to decrease $f(x)$ for small values of α, since we have by Taylor expansion:

$$\begin{aligned} f(x+\delta x) &\approx f(x) + \delta x f'(x) \qquad \text{for } |\delta x| << 1. \\ &= f(x) - \alpha f'(x) \cdot f'(x) \\ &= f(x) - \alpha [f'(x)]^2 \\ &< f(x) \end{aligned}$$

By the same argument, we get that $f(x+\delta x) \approx f(x)$ when $f'(x) = 0$, which indicates we have converged to a local optimum.

Example

Consider $f(x) = x^2 \sin x + x$.

$$\Rightarrow \quad f'(x) = 2x\sin(x) + x^2\cos(x) + 1$$

and $$f''(x) = (2 - x^2)\sin(x) + 4x\cos(x)$$

We explore steps of gradient descent starting at $x_0 = 2$ and then at $x_0 = 5$ using $\alpha = 0.05$.

We get faster convergence for $x_0 = 5$ to a local minimum at $x^* = 5.05$.

See course web page for Python notebook...

Local Optimality Conditions in d Dimensions

Notation We use $\bar{w}, \bar{\xi}$ as variables ($\bar{w} = [w_1, \dots w_d]^T$ or $\bar{w} = [w_0, w_1, \dots w_d]^T$) while $\bar{x}, \bar{y}$ are data in our settings of optimization for ML. The objective function is a loss function typically denoted J, e.g., $J = \frac{1}{2}\|D\bar{w} - \bar{y}\|^2 + \frac{1}{2}\|\bar{w}\|^2$ for regularized regression.

Lemma 2 Let $J: \mathbb{R}^d \to \mathbb{R}$ be a loss function. Then $J(\bar{w})$ is a minimum value at $\bar{w} = \bar{w}_0$ with respect to its immediate locality if

$$\nabla J(\bar{w}_0) = \bar{0}, \text{ i.e., } \left[\frac{\partial J}{\partial w_1} \cdots \frac{\partial J}{\partial w_d}\right]^T \Bigg|_{\bar{w}=\bar{w}_0} = \bar{0}, \text{ and}$$

(first order optimality condition)

and $HJ(\bar{w}_0) \succ 0$, i.e., the Hessian at $\bar{w} = \bar{w}_0$ is positive definite,

(i.e., $\bar{w}^T H \bar{w} \geq 0 \;\; \forall \bar{w} \in \mathbb{R}^d \setminus \{\bar{0}\}$.

(second order optimality condition)

Similar to the 1D case, we can understand these conditions using the Taylor expansion of J at $\bar{w}_0$ with $\bar{w} = \bar{w}_0 + \epsilon\bar{v}$ for $\epsilon > 0$:

$$J(\bar{w}_0 + \epsilon\bar{v}) \approx J(\bar{w}_0) + \epsilon \bar{v}^T \underbrace{\nabla J(\bar{w}_0)}_{=\bar{0}} + \frac{\epsilon^2}{2} \bar{v}^T \underbrace{[HJ(\bar{w}_0)]\bar{v}}_{>0 \;\forall \bar{v} \neq \bar{0}}$$

Under the first and second order optimality conditions, we get $J(\bar{w}_0 + \epsilon\bar{v}) > J(\bar{w}_0) \;\; \forall \bar{v} \neq \bar{0}$ and $\epsilon > 0$.

MATH 565: Lecture 4 (01/22/2026)

Today:
* illustration of gradient descent
* local optimality in d-dimensions
* convex functions

We saw gradient descent for $f(x) = x^2\sin(x) + x$, with

$f'(x) = 2x\sin x + x^2\cos(x) + 1$,

and $f''(x) = (2-x^2)\sin(x) + 4x\cos(x)$

A 2D Extension

Consider $G(x,y) = f(x) + f(y)$, which is a separable function, i.e., it has no terms involving both x and y.

$$\Rightarrow \nabla G = \begin{bmatrix} \frac{\partial G}{\partial x} \\ \frac{\partial G}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x\sin(x) + x^2\cos(x) + 1 \\ 2y\sin(y) + y^2\cos(y) + 1 \end{bmatrix}$$

Gradient descent now implements $\bar{w} \leftarrow \bar{w} - \alpha \nabla G(\bar{w})$.

Check Python notebook on the course web page...

We consider one final example with a nonseparable function:

Let $d(x,y) = x^2\sin(y)$, which gives

$$\nabla d = \begin{bmatrix} 2x\sin y \\ x^2\cos y \end{bmatrix}.$$

The point of these exercises was to demonstrate that optimization using gradient descent in high dimensions could become quite complex quickly...

More on local optimality in d dimensions

The sufficient conditions (first- and second order) for local optimality in d dimensions (given in Lemma 2 in lecture 3) can be qualified further to specify more nuanced cases for the second order condition.

Recall: $\nabla J = \bar{0}$ (first order optimality condition)

$HJ \succ 0$, i.e., H is PD (positive definite)

→ (second order optimality condition)

(Result: a real symmetric matrix H is PD $\Leftrightarrow$ all its eigenvalues are > 0).

Second order optimality conditions

Let $\nabla J(\bar{w}_0) = \bar{0}$.

1. If $HJ(\bar{w}_0) \succ 0$, then $\bar{w}_0$ is a local minimum (Lemma 2)
   (H is PD)
2. If $HJ(\bar{w}_0) \prec 0$, then $\bar{w}_0$ is a local maximum.
   (H is ND, negative definite)
3. If $HJ(\bar{w}_0)$ is indefinite, then $\bar{w}_0$ is a saddle point.
4. If $HJ(\bar{w}_0) \succeq 0$ or $HJ(\bar{w}_0) \preceq 0$, then the test is inconclusive.
   ↖ positive/negative semidefinite

The presence of local optima can make the task of finding a global optimum hard. We now study a class of functions for which any local optima are also guaranteed to be global optima — convex functions.

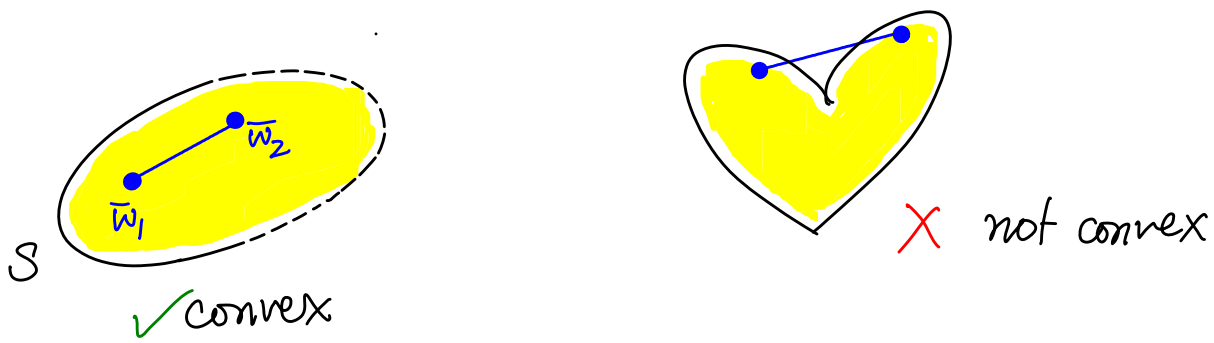
Convex Sets and Functions

We first define convex sets.

Def A set $S \subseteq \mathbb{R}^d$ is **convex** if $\forall\ \bar{w}_1, \bar{w}_2 \in S$,

$$\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2 \in S \quad \forall\ \lambda \in [0,1].$$

In words, the line segment connecting $\bar{w}_1$ and $\bar{w}_2$ lies in S.

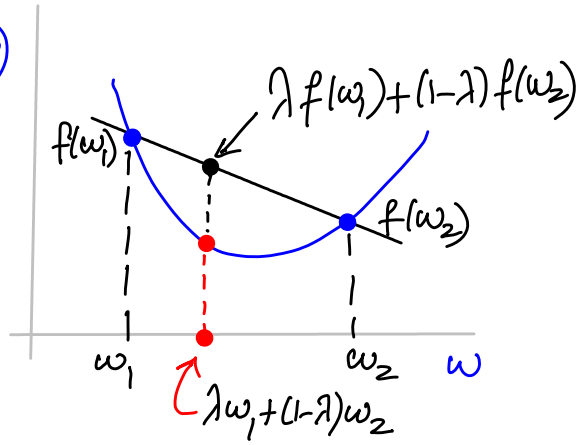


Def A function $f: \Omega \to \mathbb{R}$ is a **convex function** for convex set $\Omega \subseteq \mathbb{R}^d$ if

$$f\left(\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2\right) \leq \lambda f(\bar{w}_1) + (1-\lambda) f(\bar{w}_2) \qquad (*)$$

$$\forall\ \bar{w}_1, \bar{w}_2 \in \Omega,\ \forall\ \lambda \in [0,1].$$

In 1D, the value of the function at a convex combination of two points lies not above the corresponding convex combination of the function values at the two points.



Proposition 3 If $f: \Omega \to \mathbb{R}$ is convex, then

$$f\left(\sum_{i=1}^{k} \lambda_i \bar{\omega}_i\right) \leq \sum_{i=1}^{k} \lambda_i f(\omega_i) \text{ for } \omega_i \in \Omega,\ 0 \leq \lambda_i \leq 1,\ \sum_{i=1}^{k} \lambda_i = 1.$$

Proof We use induction on k. → $k=1$ is trivial: $\lambda_1 = 1$ and $f(1\bar{\omega}_1) = 1 f(\bar{\omega}_1)$

base case $k=2$ is (*) from the definition of a convex function.

Assume result holds for k, and consider statement for $k+1$:

We want to show that

$$f\left(\sum_{i=1}^{k+1} \lambda_i \bar{\omega}_i\right) \leq \sum_{i=1}^{k+1} \lambda_i f(\bar{\omega}_i) \quad \text{holds.}$$

Assume $0 < \lambda_{k+1} < 1$ ($\lambda_{k+1} = 0$ or $1 \Rightarrow$ trivial)

note: $\sum_{i=1}^{k} \lambda_i = 1 - \lambda_{k+1}$

$$\Rightarrow f\left(\sum_{i=1}^{k} \lambda_i \bar{\omega}_i + \lambda_{k+1} \bar{\omega}_{k+1}\right) = f\left(\frac{1-\lambda_{k+1}}{(1-\lambda_{k+1})} \sum_{i=1}^{k} \lambda_i \bar{\omega}_i + \lambda_{k+1} \bar{\omega}_{k+1}\right)$$

$$= f\left((1-\lambda_{k+1})\left[\sum_{i=1}^{k} \lambda_i' \bar{\omega}_i\right] + \lambda_{k+1} \bar{\omega}_{k+1}\right) \quad \text{for } \lambda_i' = \frac{\lambda_i}{1-\lambda_{k+1}}$$

($0 \leq \lambda_i' \leq 1$)

$$\leq (1-\lambda_{k+1}) f\left(\sum_{i=1}^{k} \lambda_i' \bar{\omega}_i\right) + \lambda_{k+1} f(\bar{\omega}_{k+1})$$

by result for $k=2$ (base case)

But $\sum_{i=1}^{k} \lambda_i' = 1$, which gives that the expression is

$$\leq (1-\lambda_{k+1}) \sum_{i=1}^{k} \lambda_i' f(\bar{\omega}_i) + \lambda_{k+1} f(\bar{\omega}_{k+1})$$

by induction assumption

$$= \sum_{i=1}^{k+1} \lambda_i f(\bar{\omega}_i)$$

as $(1-\lambda_{k+1})\lambda_i' = \lambda_i$

□

Useful Properties of Convex Functions

1. If f_1, f_2 are convex functions, then $g = f_1 + f_2$ is convex.
   (sum of convex functions is convex)

2. If f_1, f_2 are convex functions, then $g = \max(f_1, f_2)$ is convex.
   (max of convex functions is convex)

3. If $f: \Omega \to \mathbb{R}$ is convex and $f(\bar{w}) \geq 0$, then $g = f^2 = [f(\bar{w})]^2$ is convex.

4. If $f: \mathbb{R} \to \mathbb{R}$ is convex and $g: \mathbb{R}^d \to \mathbb{R}$ is linear, then $h: \mathbb{R}^d \to \mathbb{R}$ defined as $h = f(g(\bar{w}))$ is convex.

Proof $f(\lambda w_1 + (1-\lambda) w_2) \leq \lambda f(w_1) + (1-\lambda) f(w_2)$ ——— (1)

Let $g(\bar{w}) = \bar{w}^T \bar{x} + b$ linear function.

Then, $h(\bar{w}) = f(g(\bar{w})) = f(\bar{w}^T \bar{x} + b)$

$$\Rightarrow h(\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2) = f(g(\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2))$$

$$= f(\bar{x}^T(\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2) + b) \quad \to (\lambda + (1-\lambda))b$$

$$= f(\lambda \underbrace{(\bar{x}^T \bar{w}_1 + b)}_{w_1} + (1-\lambda) \underbrace{(\bar{x}^T \bar{w}_2 + b)}_{w_2})$$

$$\leq \lambda f(\bar{x}^T \bar{w}_1 + b) + (1-\lambda) f(\bar{x}^T \bar{w}_2 + b) \quad \to \text{by (1)}$$

$$= \lambda h(\bar{w}_1) + (1-\lambda) h(\bar{w}_2)$$

□

This form of composition of functions is used in many machine learning contexts, e.g., in deep learning networks, a node may take the input from k other nodes, combine them in a linear or affine format, and then apply a convex transformation to that combination.

The order in which the composition is taken in the above statement is important! For instance, if we consider $f(x)=x^2$, which is convex, and $g(x)=-x$, which is linear, and then consider

$$j(x)=g(f(x))=-x^2,$$

which is concave!

MATH 565: Lecture 5 (01/27/2026)

Today: * More results on convexity
* Details of gradient descent

Recall: $f: \Omega \to \mathbb{R}$ for convex $\Omega \subseteq \mathbb{R}^d$ is convex if $\forall \bar{w}_1, \bar{w}_2 \in \Omega$, $\lambda \in [0,1]$,

$$f(\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2) \leq \lambda f(\bar{w}_1) + (1-\lambda) f(\bar{w}_2) \quad \text{——— } (*)$$

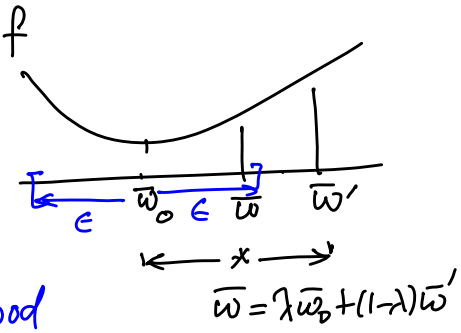
We can use convexity to show that every local minimum of a convex function must be a global minimum!

Proof

Let $\bar{w}_0$ be a local minimum of $f(\bar{w})$.

$$\Rightarrow f(\bar{w}) \geq f(\bar{w}_0) \quad \forall \|\bar{w} - \bar{w}_0\| \leq \epsilon \quad \text{——— (1)}$$

↘ small enough neighborhood of $\bar{w}_0$



Consider $\bar{w}'$ such that $\|\bar{w}' - \bar{w}_0\| > \epsilon$.

Let $\lambda \in (0,1)$ be such that

$$\bar{w} = \lambda \bar{w}_0 + (1-\lambda)\bar{w}' \text{ has } \|\bar{w} - \bar{w}_0\| \leq \epsilon.$$

↘ $\bar{w}'$ is outside the local neighborhood of $\bar{w}_0$, but $\bar{w}$ is within the same.

f is convex $\Rightarrow$

$$f(\underbrace{\lambda \bar{w}_0 + (1-\lambda)\bar{w}'}_{\bar{w}}) \leq \lambda f(\bar{w}_0) + (1-\lambda) f(\bar{w}') \quad \text{——— (2)}$$

↘ by $(*)$

Combining (1) and (2) $\Rightarrow$

$$f(\bar{w}_0) \leq f(\bar{w}) \leq \lambda f(\bar{w}_0) + (1-\lambda) f(\bar{w}')$$

$$\Rightarrow (1-\lambda) f(\bar{w}_0) \leq (1-\lambda) f(\bar{w}')$$

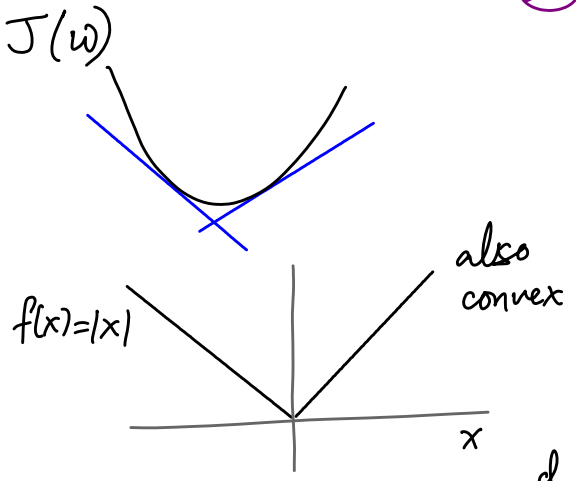
$$\Rightarrow f(\bar{w}') \geq f(\bar{w}_0)$$

$\Rightarrow \bar{w}_0$ is a global minimum.

□

An Observation

A convex function f lies above any tangent to f. We can use this observation to derive a characterization of convex functions in terms of ∇f.



Lemma 4 A differentiable function $f(\bar{w}): \Omega \to \mathbb{R}$ for convex $\Omega \subseteq \mathbb{R}^d$ is a convex function if and only if

$$f(\bar{w}) \geq f(\bar{w}_0) + [\nabla f(\bar{w}_0)]^T (\bar{w} - \bar{w}_0) \quad \text{——— (I)}$$

$\forall\, \bar{w}, \bar{w}_0 \in \Omega$

↪ first derivative condition of convexity

Proof ($\Rightarrow$) Assume f is convex $\Rightarrow$

$\forall\, \bar{w}, \bar{w}_0 \in \Omega$, $\lambda \in [0,1]$

$$f((1-\lambda)\bar{w}_0 + \lambda\bar{w}) \leq (1-\lambda) f(\bar{w}_0) + \lambda f(\bar{w})$$

↪ same as (*), but with λ and $1-\lambda$ flipped

$$\Rightarrow f(\bar{w}_0 + \lambda(\bar{w} - \bar{w}_0)) - f(\bar{w}_0) \leq \lambda\,(f(\bar{w}) - f(\bar{w}_0))$$

Hence, for $\lambda > 0$, we get

$$\frac{f(\bar{w}_0 + \lambda(\bar{w} - \bar{w}_0)) - f(\bar{w}_0)}{\lambda} \leq f(\bar{w}) - f(\bar{w}_0).$$

Taking the limit of LHS as $\lambda \to 0^+$ gives the directional derivative of f at $\bar{w}_0$ in the direction of $\bar{w} - \bar{w}_0$.

$$\Rightarrow [\nabla f(\bar{w}_0)]^T (\bar{w} - \bar{w}_0) \leq f(\bar{w}) - f(\bar{w}_0)$$

$$\Rightarrow f(\bar{w}) \geq f(\bar{w}_0) + \nabla f(\bar{w}_0)^T (\bar{w} - \bar{w}_0)$$

($\Leftarrow$) proof: you can do it yourselves!

□

Note: This definition assumes that f is differentiable. → $f(x)=|x|$ is not differentiable, but is convex!

But still, going one step further, we can give a characterization based on the second derivative of f.

Lemma 5 A twice differentiable function $f: \Omega \to \mathbb{R}$ for convex $\Omega \subseteq \mathbb{R}^d$ is convex iff

$$Hf(\bar{w}) \succeq 0 \quad \forall \bar{w} \in \Omega.$$

→ second derivative condition of convexity

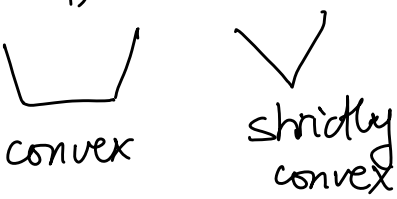
Note: Lemma 4 says that if $\nabla f(\bar{w}_0) = \bar{0}$, then $f(\bar{w}) \geq f(\bar{w}_0) \; \forall \bar{w}$, directly implying that $\bar{w}_0$ is a global optimum.

Strict Convexity

→ a stronger notion of convexity

Def $f: \Omega \to \mathbb{R}$ for convex $\Omega \subseteq \mathbb{R}^d$ is strictly convex if

$$f(\lambda \bar{w}_1 + (1-\lambda)\bar{w}_2) < \lambda f(\bar{w}_1) + (1-\lambda) f(\bar{w}_2) \quad \forall \bar{w}_1, \bar{w}_2 \in \Omega, \lambda \in [0,1].$$



Lemma 6 A strictly convex function can contain at most one critical point. If such a critical point exists, it is the global minimum of the function.

$y = e^x$: strictly convex but has no critical point.

Lemma 7 Let $f: \Omega \to \mathbb{R}$ be convex and $g: \Omega \to \mathbb{R}$ be strictly convex for convex $\Omega \subseteq \mathbb{R}^d$. Then $h: \Omega \to \mathbb{R}$ defined as $h = f + g$ is strictly convex.

This property is used in many ML settings — we add a strictly convex term to convex loss J to make it strictly convex.

For instance, Tikhonov regularization:

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

($\|D\bar{w} - \bar{y}\|^2$: convex; $\|\bar{w}\|^2$: strictly convex)

→ J is strictly convex.

Details of Gradient descent

$$\bar{w} \leftarrow \bar{w} - \alpha \nabla J(\bar{w})$$

α: learning rate; $\nabla J(\bar{w})$: gradient

We consider both these components in detail.

How to compute ∇J?

* analytically (when simple/possible) and hard code it into the computation
  - automatic differentiation (in NNs)..
    → we'll talk about it later...

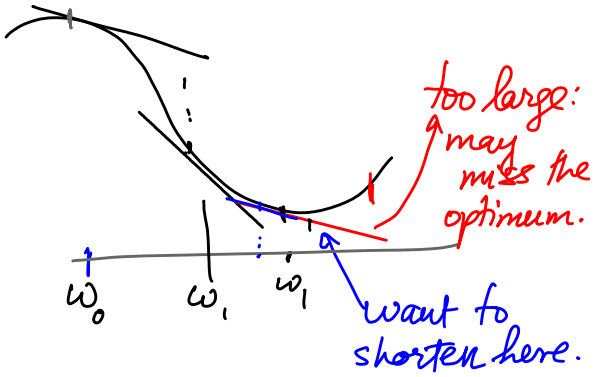
* finite difference approximation: $\nabla J = \begin{bmatrix} \vdots \\ \frac{\partial J}{\partial w_i} \\ \vdots \end{bmatrix}$

We compute $\frac{\partial J}{\partial w_i} \approx \frac{J(w_1, \ldots, w_i + \Delta, \ldots, w_d) - J(\bar{w})}{\Delta_i}$ for "small" Δ

  - can do this calculation for only a subset of $i \in \{1, \ldots, d\}$, especially when $d >> 1$.
    ↪ when d is in the billions, we cannot afford to take the partial derivative in each dimension.
  - We could also use different Δ for each i, (as long as Δ_i is "small").

Changing α

Intuitively, we want to start with a large α so that J can decrease rapidly at start. And then make α smaller when we are "close to" a critical point.
(so, we "decay" α)



$$\bar{w}_{t+1} = \bar{w}_t - \alpha_t \nabla J(\bar{w}_t)$$

t: time ($\equiv$ iteration k as used before)

1. Exponential decay

$$\alpha_t = \alpha_0 e^{(-\mu t)}$$ → controls the rate of decay

2. inverse decay:

$$\alpha_t = \frac{\alpha_0}{1+\mu t}$$

3. Step decay: → decrease α_t every m steps

$$\alpha_t = \frac{\alpha_{t-m+1}}{\mu} \quad \text{when} \quad t = zm \text{ for } z \in \mathbb{N}.$$

4. Bold driver algorithm:

α_0: (initial value)

step t: $\alpha_t = \alpha_{t-1} \times 1.05$ → increase α_t by 5% to keep the decrease of J going

compute J_t

if $J_t < J_{t-1}$
  proceed;
else
  $\alpha_t = \alpha_t/2$
  compute J_t again
end

MATH 565: Lecture 6 (01/29/2026)

Today: * 2nd order optimality examples
* gradient descent: line search
* stochastic gradient descent (SGD)

Second order local optimality conditions (from Lecture 4): With $\nabla J(\bar{w}_0)=\bar{0}$,

#4. If $HJ(\bar{w}_0) \succcurlyeq 0$, test is inconclusive.

Consider $f(x,y)=x^2+y^4$
$g(x,y)=x^2-y^4$
$h(x,y)=x^2+y^3$

$\Rightarrow \nabla f=\begin{bmatrix}2x\\4y^3\end{bmatrix}, \nabla g=\begin{bmatrix}2x\\-4y^3\end{bmatrix}, \nabla h=\begin{bmatrix}2x\\3y^2\end{bmatrix}; \Rightarrow \nabla=\bar{0}$ for all three functions at $\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$

Also, $Hf=\begin{bmatrix}2&0\\0&12y^2\end{bmatrix}, Hg=\begin{bmatrix}2&0\\0&-12y^2\end{bmatrix}, Hh=\begin{bmatrix}2&0\\0&6y\end{bmatrix}$

$\Rightarrow H=\begin{bmatrix}2&0\\0&0\end{bmatrix}$ at $\bar{0}$ for all cases, and $H \succcurlyeq 0$ (PSD).
→ eigenvalues are 2, 0.

But $\begin{bmatrix}x\\y\end{bmatrix}=\bar{0}$ is a local minimum of f, a saddle point of g, and is neither of h!

Consider the Taylor series expansion:

$$J(\bar{w}) \approx J(\bar{w}_0) + \underbrace{\nabla J^T}_{=0}(\bar{w}-\bar{w}_0) + \underbrace{(\bar{w}-\bar{w}_0)^T HJ(\bar{w}_0)}_{=0?}(\bar{w}-\bar{w}_0) + \text{higher order terms}$$

If $H=0$ at $\bar{w}_0$, then behavior of J depends on the higher order terms, and hence the second order test is inconclusive when determining local optima.

Back to gradient descent

$$\bar{w}_{t+1} = \bar{w}_t - \alpha_t \nabla J(\bar{w}_t)$$

Choosing α_t (continued...)

→ We saw decay strategies for changing α_t last time...

5. **Line search**

Instead of decaying, choose α_t optimally!

Let $\bar{g}_t$ be a descent direction in step t. We can set $\bar{g}_t = -\nabla J(\bar{w}_t)$ by default, but can pick it differently.

In $\bar{w}_{t+1} \leftarrow \bar{w}_t + \alpha_t \bar{g}_t$, we pick α_t as

$$\alpha_t = \underset{\alpha}{\operatorname{argmin}}\ J(\bar{w}_t + \alpha \bar{g}_t) \qquad (\ell)$$

(ℓ for line search)

Since we are trying to minimize J, we may as well go all the way in each iteration. Also, we get the following result.

If we choose α_t by solving (ℓ), we can guarantee that

$$\nabla J(\bar{w}_{t+1}) \perp \bar{g}_t$$

else, α_t is not optimal for (ℓ)!

Consider moving a small amount ($\pm\delta$) further along $\bar{g}_t$ (beyond α_t):

$$J(\bar{w}_t + \alpha_t \bar{g}_t \pm \delta \bar{g}_t) \approx J(\bar{w}_t + \alpha_t \bar{g}_t) \pm \delta \bar{g}_t^T \nabla J(\bar{w}_t + \alpha_t \bar{g}_t)$$

$\neq 0 \Rightarrow$ J can be improved further.

If $\nabla J(\bar{w}_{t+1}) \neq \bar{0}$, we can decrease J further by moving $\pm\delta$ units along $\bar{g}_t$.

Choosing descent directions in this orthogonal fashion could be useful in general...

How to solve (1)?

Pick some α_{max} and consider $\alpha \in [0, \alpha_{max}]$, and search this interval using

- binary search → halve the interval in each iteration, keeping the half that is beneficial
- golden section search
  let current interval is $[a, b]$
  check $a < m_1 < m_2 < b$ → unlike in binary search, evaluate only (up to) 4 points.
  can drop $[a, m_1]$ or $[m_2, b]$ or
  can exclude $[a, m_2)$ or $(m_1, b]$.
- Armijo rule (check L04ML). → another inexact but efficient method

---

Properties of Optimization in ML

* objective functions are loss functions that are usually additively separable — contribution to loss from each sample or point is added up to get the total loss.

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

Here is another common loss function in ML:

- product of probabilities $P(\bar{x}_i, y_i, \bar{w})$ that the prediction for sample i using $\bar{w}$ matches true value y_i for each sample:

$$\prod_{i=1}^{n} P(\bar{x}_i, y_i, \bar{w}).$$

But instead of the product, we use → additively separable

$$J(\bar{w}) = -\log\left(\prod_{i=1}^{n} P(\bar{x}_i, y_i, \bar{w})\right) = -\sum_{i=1}^{n} \log\left(P(\bar{x}_i, y_i, \bar{w})\right).$$

In general, we have

$$J(\bar{w}) = \sum_{i=1}^{n} J_i(\bar{w}).$$

$J_i(\bar{w})$: loss from sample/point i.

i: sample from the n points

This structure makes it possible to use stochastic gradient descent (SGD):

Let S be a sample of the n points (observations) $S \subset \{1,\dots,n\}$. Consider $J_S(\bar{w}) = \sum_{i \in S} J_i(\bar{w})$. For instance, in regression, $J_S(\bar{w}) = \sum_{i \in S} \|\bar{x}_i^T \bar{w} - y_i\|^2$.

S is called a **minibatch**, and the method is called the minibatch SGD, where we compute

$$\bar{w} \leftarrow \bar{w} - \alpha \nabla J_S(\bar{w})$$

J_S instead of J

Often, $|S| \ll n$, so minibatch SGD is typically more efficient than GD.

The special case with $|S|=1$ is called SGD. (without the minibatch qualifier)

→ in each iteration, choose one random observation.

SGD (with $|S|=1$) is used when n is huge (and d also is large).

One typically uses minibatch SGD with $|S| = 2^r$ (e.g., 64, 256, ...) for effective use of GPUs (or other processors/cores).

The minibatch gradients are often close to the (full) gradient at start — when $\bar{w}$ is far from a (local) minimum. But they are not as close when we get closer to the minimum — but regularization usually helps out.

Minibatch SGD usually tries to use as much of the data (points) as the iterations proceed. Here is one approach. We use

$R[n]$ = a random permutation of $\{1,\dots,n\} = \{r_1,\dots,r_n\}$.

(each r_i is one of $\{1,2,\dots,n\}$.)

Let $|S| = k$ (minibatch size).

Set $S_i = \{r_{(i-1)k+1},\dots,r_{ik}\}$, for $i = 1,\dots,\lceil \frac{n}{k} \rceil$.

A single S_i is referred to as an epoch. Each epoch uses k points (observations), except maybe the last epoch. There are a total of $\lceil \frac{n}{k} \rceil$ epochs.

In a typical minibatch SGD run,

- ∇J_S is a good approximation of ∇J
- Choice of S becomes more important as $\bar{w}$ gets closer to a local minimum.
  → (regularization helps).

MATH 565: Lecture 7 (02/03/2026)

Today: * optimization in general v/s in ML
* cross validation
* SGD details

Optimization in general v/s Optimization in ML

general

1. objective function need not be separable

2. Optimize as much as possible

3. Worst performance on outside/test-data; as the model is overfitted to training data

4. Work with the original objective function.

ML

1. loss functions are usually additively separable, e.g.,
$$J(\bar{w}) = \frac{1}{2}\|D\bar{w}-\bar{y}\|^2 = \sum_{i=1}^{n}(x_i^T\bar{w}-y_i)^2$$

2. Avoid overfitting — stop when "performance" on test-data is high enough (or use cross validation)

3. better performance on test-data; because of stochasticity (from SGD), regularization, etc.

4. Loss function J usually modified by adding regularization term(s)
e.g., Tikhonov regularization
$$J(\bar{w}) = \frac{1}{2}\|D\bar{w}-\bar{y}\|^2 + \frac{1}{2}\|\bar{w}\|^2$$

Hyperparameter Tuning

e.g. SVM

$$J_{L_1\text{-SVM}} = C\sum_{i=1}^{n}\xi_i + \frac{\lambda}{2}\|\vec{w}\|^2$$

variables are ξ_i, w_i; but also called as parameters in some settings.

hyperparameters $(C>0, \lambda>0)$

We use grid search to identify best C, λ values.

e.g., $C = \{0.01, 0.1, 1, 10, 100, 1000\}$ → 6 values

$\lambda = \{0.001, 0.01, 0.1, 1, 10\}$ → 5 values

For each (C_i, λ_j) choice, solve the optimization problem $\min_{\vec{w}} J$.

The only requirements are that $C>0$ and $\lambda>0$. Hence, we naturally do not want to try too many values. Here is a typical strategy.

* Start with a coarse grid, identify one pair of (C_i, λ_j) values that "perform best"
* do a finer grid search around this (C_i, λ_j) pair.

e.g., Let $C=10$, $\lambda=0.1$ be identified in the coarse grid search. Then, we explore $C=\{6.5, 7, \ldots 10, \ldots 13.5\}$ and $\lambda = \{0.05, 0.06, .., 0.1, \ldots, 0.15\}$, for instance, and identify a (possibly) new optimal pair.

Can use k-fold cross validation (see next page...)

Note: We could also do sampling of C, λ values rather than run through grids of candidate values.

k-fold Cross Validation (CV)

$k=10$ is a widely used option

Let $R[n]$ = random permutation of $\{1,\dots,n\}$.
$= \{r_1,\dots,r_n\}$, where each $r_i \in \{1,\dots,n\}$.

Form k folds (subsets) $\{F_i\}_{i=1}^{k}$ of $R[n]$, each with $p \approx \lfloor \frac{n}{k} \rfloor$ points.

e.g., $F_i = \{ r_{(i-1)p+1}, \dots, r_{ip} \}$, $i = 1, \dots, k-1$, and

$F_k = \{ r_{(k-1)p+1}, \dots, r_n \}$. → the last fold may have a slightly different # entries.

e.g., $n = 256$, $k = 10$ ⇒ $p = 26$, but last fold has 22 pts.

for $i = 1 .. k$

$T_i = R[n] \setminus F_i$ (training set for iteration i)

$\bar{w}_i = \text{argmin } J_i \left[\text{using } (D_{T_i}, \bar{y}_{T_i}) \right]$ → minimize loss function on the training set

prediction → $\hat{\bar{y}}_i \leftarrow (D_{F_i}, \bar{w}_i)$ use $\bar{w}_i$ to predict for instances in F_i

end

Evaluate $(\hat{\bar{y}}, \bar{y})$

$\hat{\bar{y}}$: vector of predicted values
$\bar{y}$: true values.

For instance, with $|C|=6$, $|\lambda|=5$ values, we get 30 $\hat{\bar{y}}$ predictions. Pick the (C_i, λ_j) pair that gives the least error.

We can use SGD inside each iteration (for solving $\min_{\bar{w}} J_i$).

Note that depending on how many hyperparameters are there and on the # folds (k), we may end up solving large numbers of (smaller) optimization problems here ($\min J_i$)! or huge!

Details of Stochastic Gradient Descent (SGD)

Recall: linear regression with Tikhonov regularization:

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

$$\Rightarrow \nabla J = \underbrace{D^TD\bar{w} - D^T\bar{y} + \lambda\bar{w}}$$

GD: $\bar{w} \leftarrow \bar{w} - \alpha\nabla J$

$$\Rightarrow \bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha D^T(\underbrace{D\bar{w}-\bar{y}}_{\bar{e}\text{ error}})$$

We compute the error vector $\bar{e}$ first, and then $D^T\bar{e}$ (and hence avoid computing D^TD). → computationally expensive to evaluate matrix-matrix product; matrix-vector products are more efficient

Rewriting the GD step:

$$\bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha\sum_{i=1}^{n}\bar{x}_i\underbrace{(\bar{w}^T\bar{x}_i - y_i)}_{e_i}$$

We get the SGD update by not using all n points:

$$\bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha\sum_{i\in S}\bar{x}_i\underbrace{(\bar{w}^T\bar{x}_i - y_i)}_{e_i}$$

If setting $y = \bar{w}^T\bar{x}+b$ (or $\bar{w}^T\bar{x}+w_0$), the SGD updates are made in a similar fashion: → b: "bias"

$$\bar{w} \leftarrow \bar{w}(1-\alpha\lambda) - \alpha\sum_{i\in S}\bar{x}_i(\bar{w}^T\bar{x}_i + b - y_i)$$

$$b \leftarrow b(1-\alpha\lambda) - \alpha\sum_{i\in S}(\bar{w}^T\bar{x}_i + b - y_i)$$

SGD for binary classification

We have $y_i = \pm 1$ now.

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

$y_i^2 = 1$

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

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

SGD: $\bar{w} \leftarrow \bar{w} - \alpha \nabla J$

$$\bar{w} \leftarrow \bar{w}(1 - \alpha\lambda) + \alpha \sum_{\cancel{i=1}\ i \in S}^{n} y_i \bar{x}_i \underbrace{(1 - y_i(\bar{w}^T \bar{x}_i))}_{\text{error}}$$

Recall that this loss function penalizes truly well-separated instances.

Better loss functions: hinge loss and L_2-SVM loss:

$$J_{H\text{-SVM}} = \sum_{i=1}^{n} \max\{0, 1 - y_i(\bar{w}^T \bar{x}_i)\} + \frac{\lambda}{2}\|\bar{w}\|^2 \quad \text{(hinge)}$$

$$J_{L_2\text{-SVM}} = \frac{1}{2}\sum_{i=1}^{n} \max\{0, (1 - y_i(\bar{w}^T \bar{x}_i))\}^2 + \frac{\lambda}{2}\|\bar{w}\|^2 \quad (L_2\text{-SVM})$$

Here are the gradients of these loss functions:

$$\nabla J_{H\text{-SVM}} = -y_i \bar{x}_i \, \delta([1 - y_i(\bar{w}^T \bar{x}_i)] > 0) + \lambda \bar{w}$$

(indicator)

$$\nabla J_{L_2\text{-SVM}} = -y_i \bar{x}_i \max\{0, 1 - y_i(\bar{w}^T \bar{x}_i)\} + \lambda \bar{w}$$