MATH 565: Lecture 9 (02/10/2026)

Today:
* block coordinate descent (BCD)
* k-means clustering as BCD
* Challenges in GD learning

Block Coordinate Descent (BCD)

* optimize over blocks of variables at a time (as opposed to one dim/var at a time as done in CD)
* each step can be more expensive (than CD), but # steps required can be smaller.
* often used for "multi-convex" problems
  - loss function J is non-convex, but
  - each block of variables gives a convex subproblem;
  - or, subproblem for each block is easy to solve (even if non-convex).

k-means clustering as BCD

As a direct illustration, we describe how k-means clustering can be viewed as a case of BCD.

k-means clustering: Divide $\bar{x}_1, \dots, \bar{x}_n \in \mathbb{R}^d$ into k clusters represented by their "centers" $\underbrace{\bar{z}_1, \dots, \bar{z}_k}_{\text{unknown}} \in \mathbb{R}^d$. Each $\bar{x}_i$ is assigned to one cluster (j, say), so that the sum of squared distances between $\bar{x}_i$ and $\bar{z}_j$ $\forall i, j$ is minimized.

Let $y_{ij} \in \{0, 1\}$ be such that

$$y_{ij} = \begin{cases} 1 & \text{if } \bar{x}_i \text{ is assigned to } \bar{z}_j \text{ (cluster } j) \\ 0 & \text{otherwise.} \end{cases}$$

$$\min_{\bar{z}_j, y_{ij}} J = \sum_{j=1}^{k} \underbrace{\sum_{i=1}^{n} y_{ij} \|\bar{x}_i - \bar{z}_j\|^2}_{O_j}$$

$$\text{s.t.} \quad \sum_{j=1}^{k} y_{ij} = 1, \quad i = 1, .., n$$

$$y_{ij} \in \{0,1\} \quad \forall i, j$$

this is the optimization problem representing k-means clustering. It is a mixed integer nonlinear program (MINLP)

BCD: Alternately fix y_{ij}'s and $\bar{z}_j$'s, and optimize over the others.

one iteration

Step 1 If $\bar{z}_j$'s are fixed, we can choose $y_{ij} = 1$ for which $\|\bar{x}_i - \bar{z}_j\|^2$ is minimal. assign each pt to the nearest cluster center

Step 2 Assume y_{ij}'s are fixed, optimize over $\bar{z}_j$ blocks. BCD over $\bar{z}_j$

With $O_j = \sum_{i=1}^{n} y_{ij} \|\bar{x}_i - \bar{z}_j\|^2$, we get

$$\nabla_{\bar{z}_j} O_j = \left[\frac{\partial O_j}{\partial \bar{z}_j}\right] = -2 \sum_{i=1}^{n} y_{ij} (\bar{x}_i - \bar{z}_j) = \bar{0}$$

$$\Rightarrow \quad \bar{z}_j = \frac{\sum_{i: y_{ij}=1} \bar{x}_i}{\sum_{j=1}^{k} y_{ij}} = \text{mean of the } \bar{x}_i\text{'s assigned to } \bar{z}_j \text{ (cluster } j\text{)}$$

Repeat iterations until convergence.

We get the k-mediods algorithm if we use an L_1-loss (in place of L_2) and apply BCD as described here.

Challenges in Gradient Descent (GD)

* ## Local Minima

Let $J(\bar{w}) = \sum_{i=1}^{d} J_i(w_i)$ → univariate

Result: If each $J_i(w_i)$ has k_i local/global minima, then J has $\prod_{i=1}^{d} k_i$ local/global minima.

→ this # can increase rapidly with d.

* ## Flat Regions

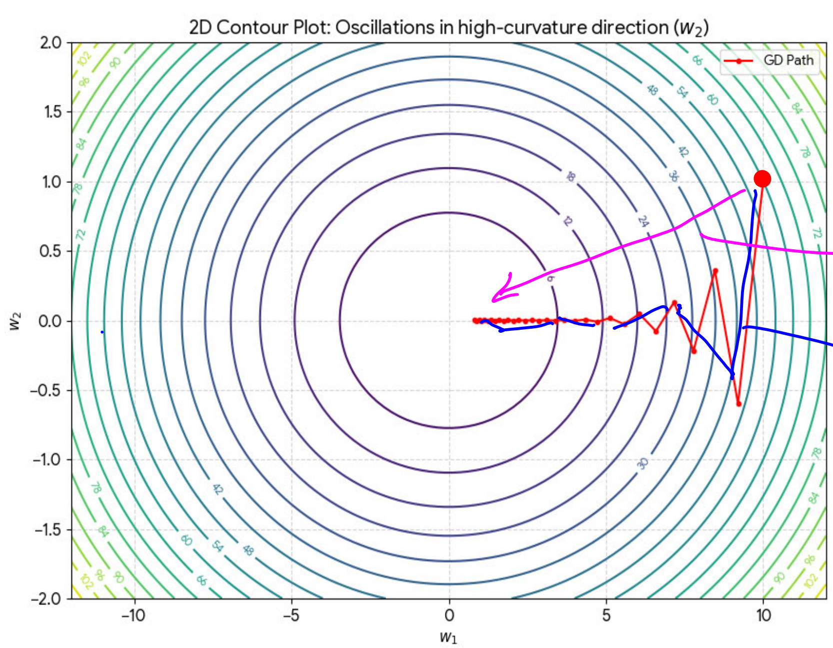
If $\|\nabla J\| \ll 1$ in a large region, then GD can take many iterations (steps) to cross it.

* ## Differential "Curvature" → rate of change of ∇.

↳ rate of change of gradient can be vastly different in different dimensions.

e.g., $J(\bar{w}) = \frac{1}{2} w_1^2 + 10 w_2^2$ → curvature in w_2 is 20x larger than in w_1

high curvature direction ↑

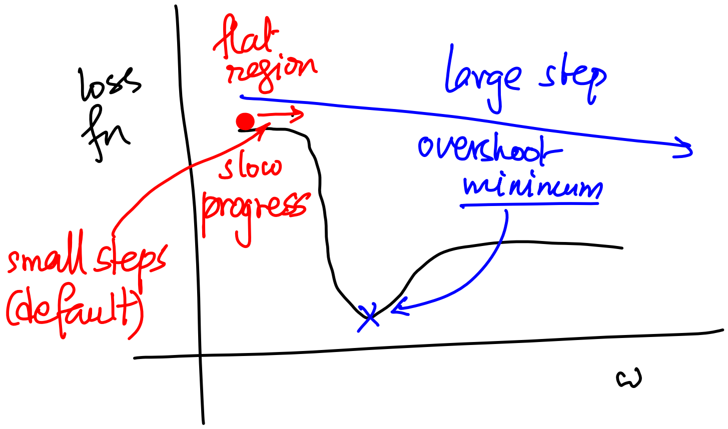


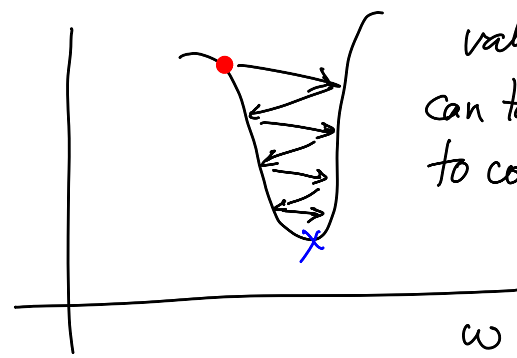
ideal descent direction

with momentum

- "vanishing and exploding gradients" problem in NNs.
- In typical ML applications (e.g., regression), standardizing or normalizing data ($\bar{x}_i$ columns) often helps.

* <u>Difficult "Topologies"</u> → cliffs or valleys





Methods to Address These Issues

<u>Ideas</u>

* use second order info by considering curvature when updating ∇.

e.g., use distinct α_i (learning rate) for each dimension $i = 1, \dots, d$.

* May not want to compute full second order details, e.g., HJ (Hessian), as that could be quite expensive computationally.

We consider several approaches along the line of these ideas...

Momentum-Based Learning

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

We rewrite this step with $\bar{v} \leftarrow -\alpha \nabla J$ as $\bar{w} \leftarrow \bar{w} + \bar{v}$.

Now, we update $\bar{v}$ instead as

$$\bar{v} \leftarrow \beta \bar{v} - \alpha \nabla J \quad \text{for } \beta \in (0,1).$$

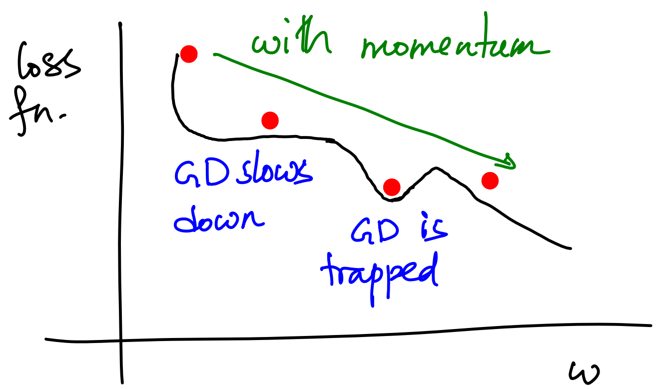
More precisely, for $k \geq 1$, we set iteration #

$$\bar{v}^{k+1} = \beta \bar{v}^{k} - \alpha \nabla J(\bar{w}_k), \text{ and}$$

$$\bar{w}^{k+1} = \bar{w}^{k} + \bar{v}^{k+1}.$$

β : momentum parameter (also called the friction parameter or damping parameter) analogy to (classical) mechanics

Analogy

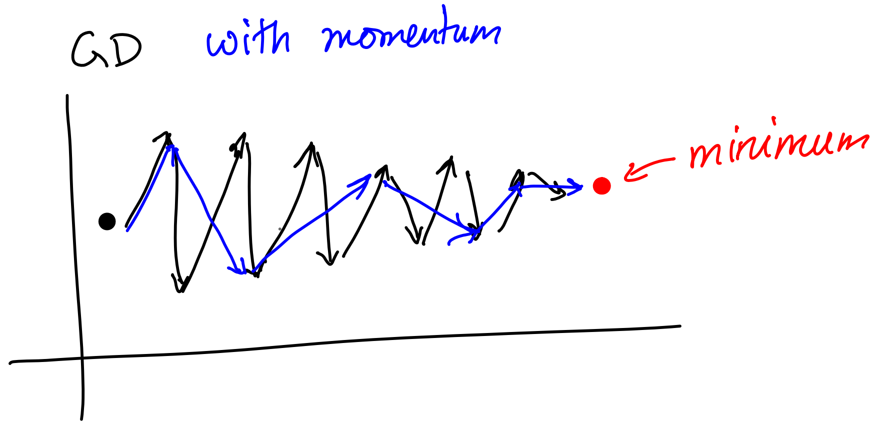


* "Standard" GD: "ball" has no mass. it stops when $\nabla J = \bar{0}$.

* adding momentum gives it "inertia" — ball keeps rolling — can coast through flat regions and avoid small bumps (local minima)

* But without "friction", the ball could oscillate a lot before settling at the global minimum.

Here is a schematic of how GD behaves vs how it does with momentum



Intuitively, momentum-based learning dampens oscillations in unwanted directions.