Optimization for ML (2)

CS771: Introduction to Machine Learning

Piyush Rai

The Plan

- Some basic techniques for solving optimization problems
  - First-order optimality
  - Gradient descent
- Dealing with non-differentiable functions
  - Sub-gradients and sub-differential

Optimization Problems in ML

- The general form of an optimization problem in ML will usually be

Usually a sum of the training error + regularizer

$$w_{opt} = \arg\min_{w \in C} L(\boldsymbol{w})$$

- Here $L(\boldsymbol{w})$ denotes the loss function to be optimized
- $\boldsymbol{C}$ is the constraint set that the solution must belong to, e.g.,
  - Non-negativity constraint: All entries in $\boldsymbol{w}_{opt}$ must be non-negative
  - Sparsity constraint: $\boldsymbol{w}_{opt}$ is a sparse vector with atmost K non-zeros
- If no $\boldsymbol{C}$ is specified, it is an unconstrained optimization problem
- Constrained opt. probs can be converted into unconstrained opt. (will see later)
- For now, assume we have an unconstrained optimization problem

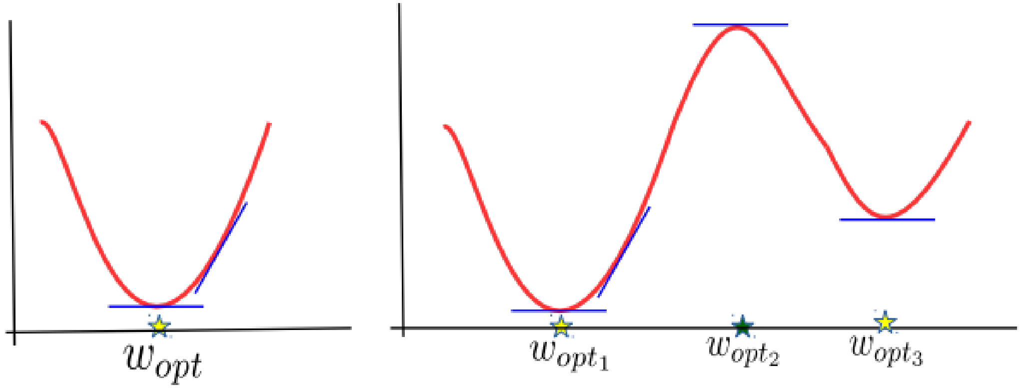
However, possible to have linear/ridge regression where solution has some constraints (e.g., non-neg, sparsity, or even both)

Linear and ridge regression that we saw were unconstrained ($\boldsymbol{w}_{opt}$ was a real-valued vector)

Methods for Solving Optimization Problems

Method 1: Using First-Order Optimality

- Very simple. Already used this approach for linear and ridge regression



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized

The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution

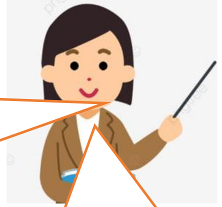
- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like gradient descent

Method 2: Iterative Optimiz. via Gradient Descent

Can I used this approach to solve maximization problems?

For max. problems we can use gradient ascent
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \boldsymbol{g}^{(t)}$$

Iterative since it requires several steps/iterations to find the optimal solution



Fact: Gradient gives the direction of steepest change in function's value

Will move in the direction of the gradient

For convex functions, GD will converge to the global minima

Good initialization needed for non-convex functions

Gradient Descent

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \ldots$ (or until convergence)
  - Calculate the gradient $\boldsymbol{g}^{(t)}$ using the current iterates $\boldsymbol{w}^{(t)}$
  - Set the learning rate η_t
  - Move in the opposite direction of gradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

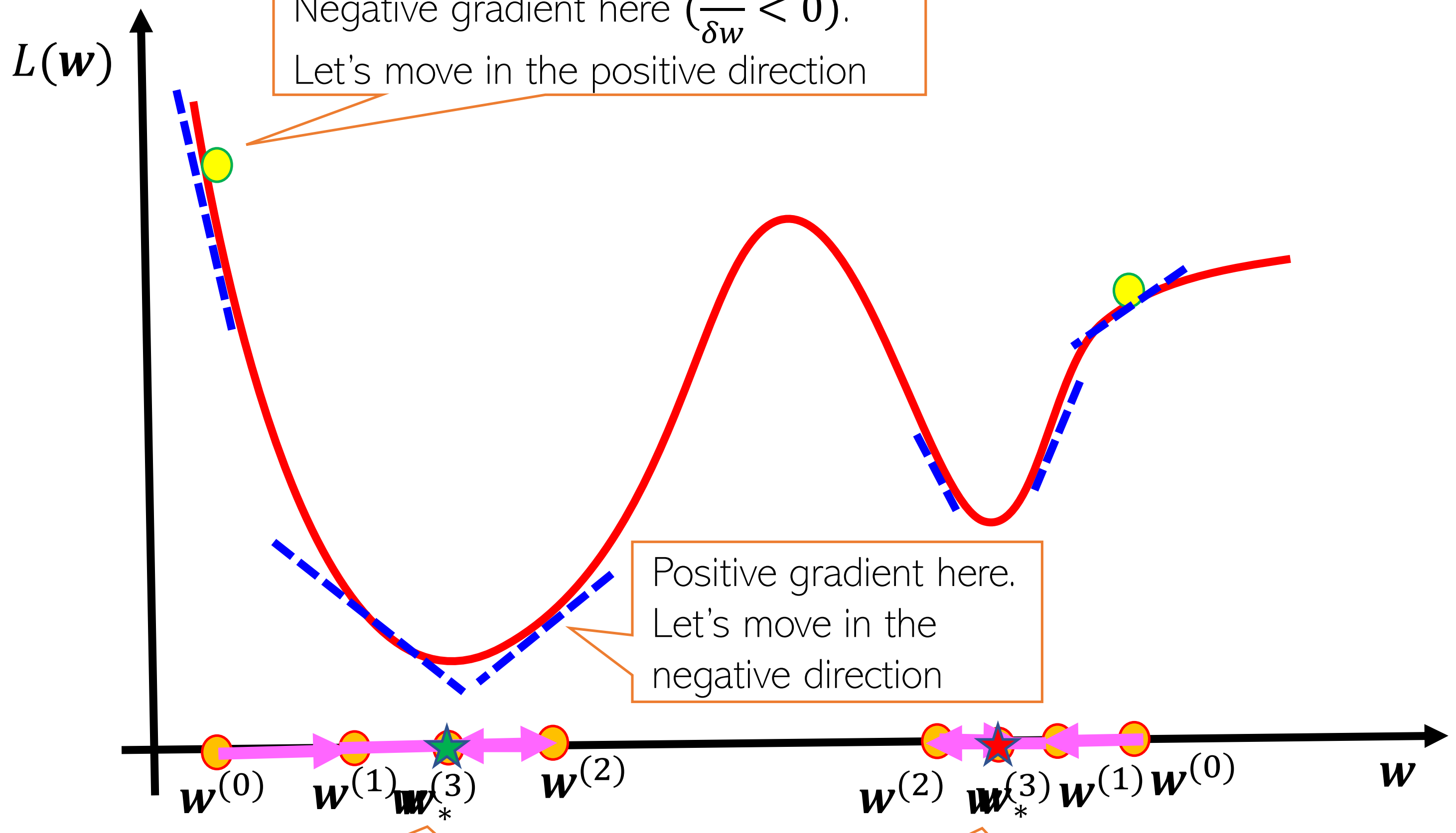
Will see the justification shortly

The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies later

Sometimes may be tricky to to assess convergence? Will see some methods later

Gradient Descent: An Illustration

Negative gradient here ($\frac{\delta L}{\delta w} < 0$). Let's move in the positive direction



Positive gradient here. Let's move in the negative direction

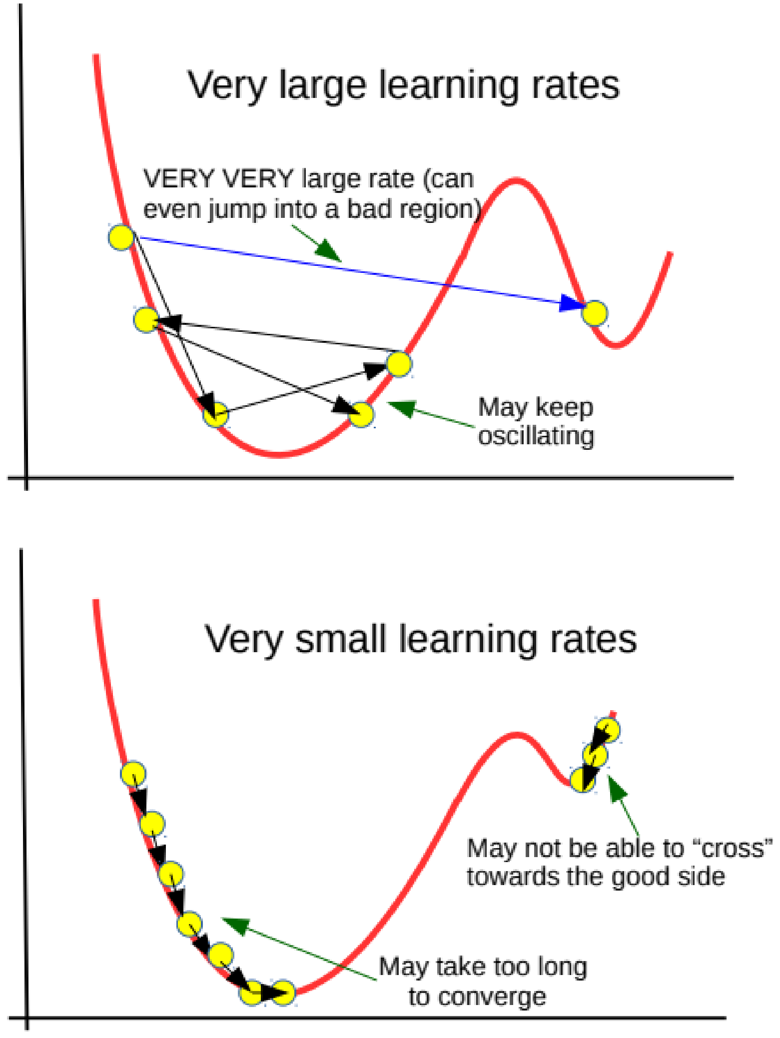
Woohoo! ☺ Global minima found!!!

Stuck at a local minima ☹

GD thanks you for the good initialization ☺

Good initialization is very important

Learning rate is very important



GD: An Example

- Let's apply GD for least squares linear regression

$$\boldsymbol{w}_{ridge} = \arg\min_{\boldsymbol{w}} L_{reg}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$$

- The gradient: $\boldsymbol{g} = -\sum_{n=1}^{N} 2(y_n - \boldsymbol{w}^\top \boldsymbol{x}_n)\boldsymbol{x}_n$

- Each GD update will be of the form

Prediction error of current model $\boldsymbol{w}^{(t)}$ on the n^{th} training example

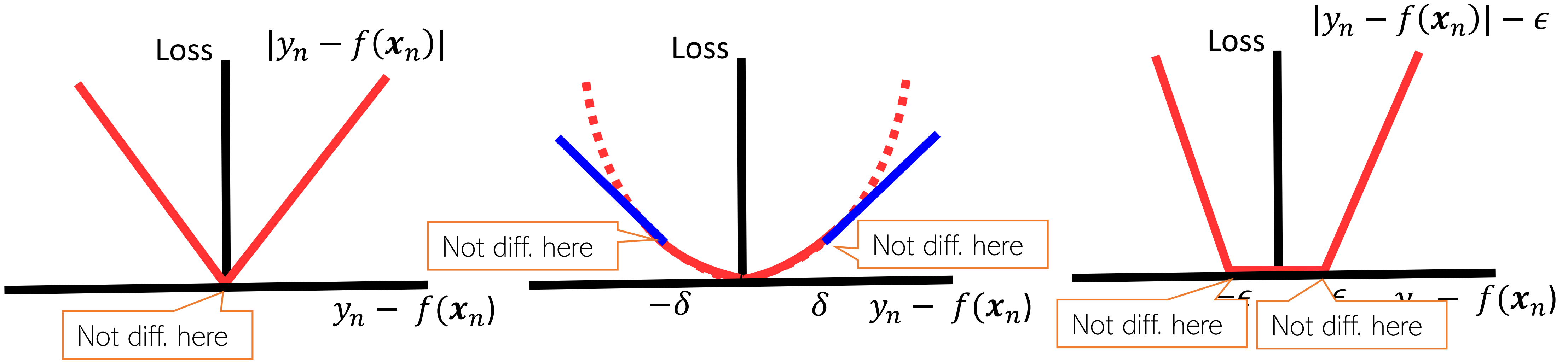
$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t \sum_{n=1}^{N} 2\left(y_n - \boldsymbol{w}^{(t)^\top} \boldsymbol{x}_n\right) \boldsymbol{x}_n$$

Training examples on which the current model's error is large contribute more to the update

- Exercise: Assume $N = 1$, and show that GD update improves prediction on the training input $(\boldsymbol{x}_n, y_n)$, i.e, y_n is closer to $\boldsymbol{w}^{(t+1)^\top} \boldsymbol{x}_n$ than to $\boldsymbol{w}^{(t)^\top} \boldsymbol{x}_n$
  - This is sort of a proof that GD updates are "corrective" in nature (and it actually is true not just for linear regression but can also be shown for various other ML models)

Dealing with Non-differentiable Functions

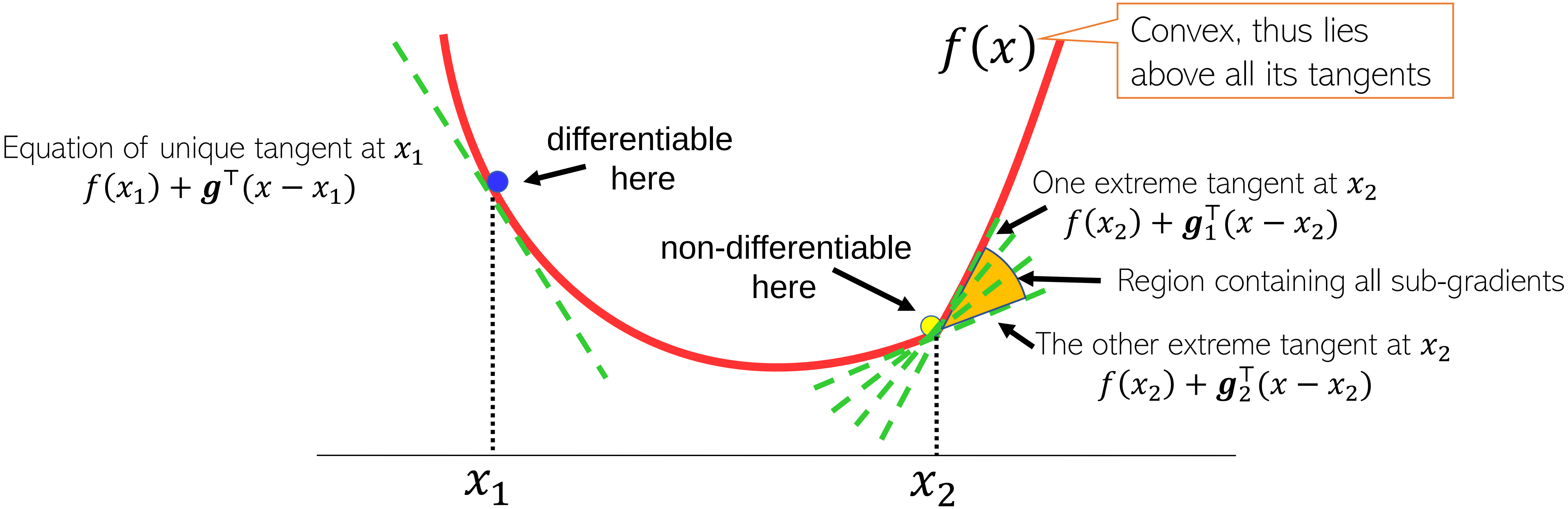
- In many ML problems, the objective function will be non-differentiable
- Some examples that we have already seen: Linear regression with absolute loss, or Huber loss, or $\boldsymbol{\epsilon}$-insensitive loss; even $\boldsymbol{\ell_1}$ norm regularizer is non-diff



- Basically, any function in which there are points with kink is non-diff
  - At such points, the function is non-differentiable and thus gradients not defined
  - Reason: Can't define a unique tangent at such points

Sub-gradients

- For convex non-diff fn, can define sub-gradients at point(s) of non-differentiabilty



- For a convex, non-diff function $f(\boldsymbol{x})$, sub-gradient at $\boldsymbol{x}_*$ is <u>any</u> vector $\boldsymbol{g}$ s.t. $\forall \boldsymbol{x}$

$$f(x) \geq f(\boldsymbol{x}_*) + \boldsymbol{g}^\top(\boldsymbol{x} - \boldsymbol{x}_*)$$

Sub-gradients, Sub-differential, and Some Rules

- Set of all sub-gradient at a non-diff point $\boldsymbol{x}_*$ is called the sub-differential

$$\boxed{\partial f(\boldsymbol{x}_*) \triangleq \{\boldsymbol{g} : f(\mathbf{x}) \geq f(\boldsymbol{x}_*) + \boldsymbol{g}^\top(\boldsymbol{x} - \boldsymbol{x}_*) \quad \forall \mathbf{x}\}}$$

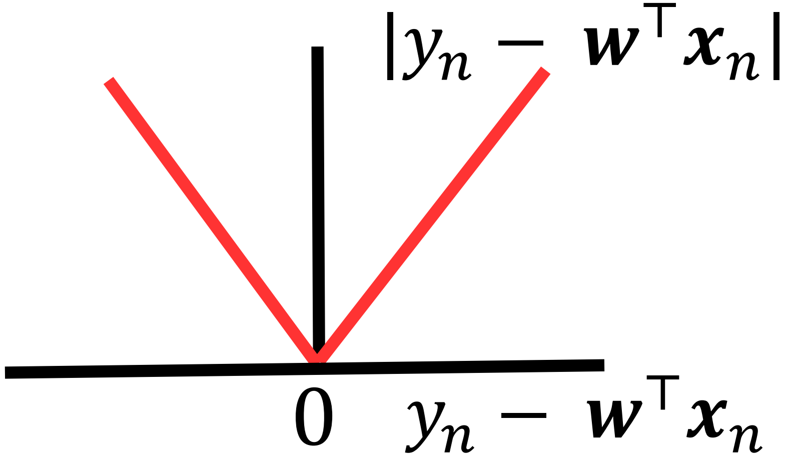
- Some basic rules of sub-diff calculus to keep in mind

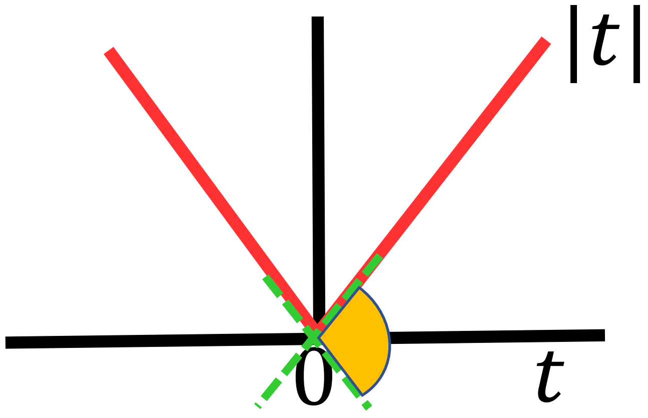
The affine transform rule is a special case of the more general chain rule

  - Scaling rule: $\partial\big(c \cdot f(\mathbf{x})\big) = c \cdot \partial f(\mathbf{x}) = \{c \cdot \mathbf{v} : \mathbf{v} \in \partial f(\mathbf{x})\}$
  - Sum rule: $\partial\big(f(\mathbf{x}) + g(\mathbf{x})\big) = \partial f(\mathbf{x}) + \partial g(\mathbf{x}) = \{\mathbf{u} + \mathbf{v} : \mathbf{u} \in \partial f(\mathbf{x}), \mathbf{v} \in \partial g(\mathbf{x})\}$
  - Affine trans: $\partial f(\mathbf{a}^\top \mathbf{x} + b) = \partial f(t) \cdot \mathbf{a} = \{c \cdot \mathbf{a} : c \in \partial f(t)\}$, where $t = \mathbf{a}^\top \mathbf{x} + b$
  - Max rule: If $h(\boldsymbol{x}) = \max\{f(\boldsymbol{x}), g(\boldsymbol{x})\}$ then we calculate $\partial h(\boldsymbol{x})$ at $\boldsymbol{x}_*$ as
    - If $f(\boldsymbol{x}_*) > g(\boldsymbol{x}_*), \partial h(\boldsymbol{x}_*) = \partial f(\boldsymbol{x}_*)$, If $g(\boldsymbol{x}_*) > f(\boldsymbol{x}_*), \partial h(\boldsymbol{x}_*) = \partial g(\boldsymbol{x}_*)$
    - If $f(\boldsymbol{x}_*) = g(\boldsymbol{x}_*), \partial h(\boldsymbol{x}_*) = \{\alpha\mathbf{a} + (1-\alpha)\mathbf{b} : \mathbf{a} \in \partial f(\boldsymbol{x}_*), \mathbf{b} \in \partial g(\boldsymbol{x}_*), \alpha \in [0,1]\}$

- $\boldsymbol{x}_*$ is a stationary point for a non-diff function $f(\boldsymbol{x})$ if the zero vector belongs to the sub-differential at x_*, i.e., $\mathbf{0} \in \partial f(\boldsymbol{x}_*)$

Sub-Gradient For Absolute Loss Regression





$$\partial|t| = \begin{cases} 1 & \text{if } t > 0 \\ -1 & \text{if } t < 0 \\ [-1,+1] & \text{if } t = 0 \end{cases}$$

- The loss function for linear reg. with absolute loss: $L(\boldsymbol{w}) = |y_n - \boldsymbol{w}^\top \boldsymbol{x}_n|$
- Non-differentiable at $y_n - \boldsymbol{w}^\top \boldsymbol{x}_n = 0$
- Can use the affine transform rule of sub-diff calculus
- Assume $t = y_n - \boldsymbol{w}^\top \boldsymbol{x}_n$. Then $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \partial|t|$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_\mathrm{n} \times 1 = -x_n$ if $t > 0$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_\mathrm{n} \times -1 = x_n$ if $t < 0$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_\mathrm{n} \times c = -cx_n$ where $c \in [-1,+1]$ if $t = 0$

Sub-Gradient Descent

- Suppose we have a non-differentiable function $L(\boldsymbol{w})$
- Sub-gradient descent is almost identical to GD except we use subgradients

Sub-Gradient Descent

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \ldots$ (or until convergence)
  - Calculate the sub-gradient $\boldsymbol{g}^{(t)} \in \partial L(\boldsymbol{w}^{(t)})$
  - Set the learning rate η_t
  - Move in the opposite direction of subgradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

Coming up next

- Making GD faster: Stochastic gradient descent
- Constrained optimization
- Co-ordinate descent
- Alternating optimization
- Practical issue in optimization for ML