Optimization for ML (wrap-up)

CS771: Introduction to Machine Learning

Announcements

- Quiz 2 on Sept 7, 6:15-7:00pm in L20 and L19
    - Class will continue from 7:00pm-8:00pm in L20
- Quiz 1 grading will be completed soon (sorry for the delay)
- Mini-project 1 to be released later this week
- Mid-sem exam on Sept 15, 8:00am-10:00am

Plan today

- Optimization methods
  - Constrained optimization
    - Projected gradient descent
    - Lagrangian based method
  - Optimization of non-differentiable functions
    - Subgradient descent
    - Some examples of subgradient descent
- Large margin classification (Support Vector Machines (SVM))
  - Solving the SVM problem involves aspects of both optimization of non-differentiable functions as well as constrained optimization, and some other aspects of optimization too, so it is also a good "case study" ☺

Constrained Optimization

Projected Gradient Descent

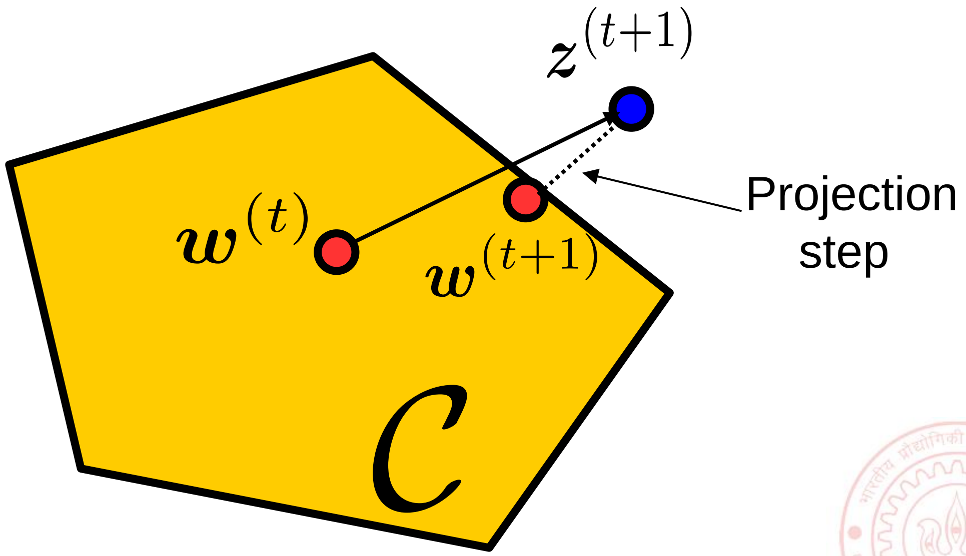
- Consider an optimization problem of the form

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

- Projected GD is very similar to GD with an extra projection step

- Each iteration $\boldsymbol{t}$ will be of the form
  - Perform update: $\boldsymbol{z}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$
  - Check if $\boldsymbol{z}^{(t+1)}$ satisfies constraints
    - If $\boldsymbol{z}^{(t+1)} \in \mathcal{C}$, set $\boldsymbol{w}^{(t+1)} = \boldsymbol{z}^{(t+1)}$
    - If $\boldsymbol{z}^{(t+1)} \notin \mathcal{C}$, project as $\boldsymbol{w}^{(t+1)} = \Pi_{\mathcal{C}}[\boldsymbol{z}^{(t+1)}]$

Projection operator



Projected GD: How to Project?

- Here projecting a point means finding the "closest" point from the constraint set

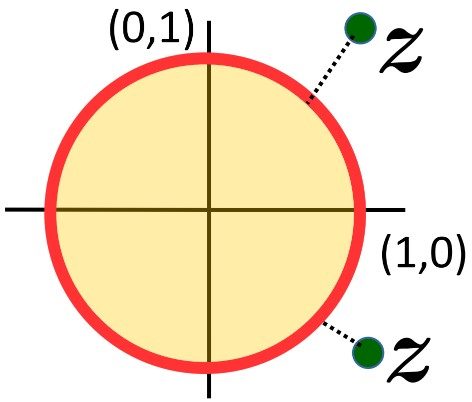
$$\Pi_{\mathcal{C}}[\boldsymbol{z}] = \arg\min_{\boldsymbol{w}\in\mathcal{C}} ||\boldsymbol{z} - \boldsymbol{w}||^2$$

Another constrained optimization problem! But simpler to solve in some special cases! ☺

Projected GD commonly used only when the projection step is simple and efficient to compute

- For some sets $\mathcal{C}$, the projection step is easy

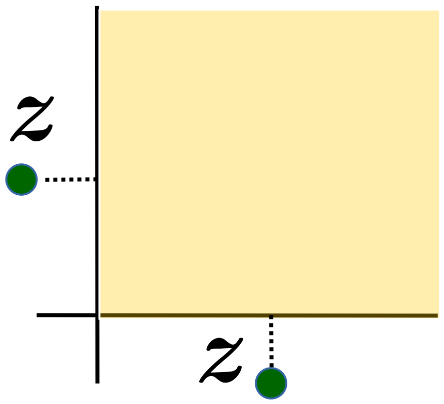
$\mathcal{C}$: Unit radius ℓ_2 ball



Projection = Normalize to unit Euclidean length vector

$$\hat{\boldsymbol{z}} = \begin{cases} \boldsymbol{z} & \text{if } ||\boldsymbol{z}||_2 \leq 1 \\ \dfrac{\boldsymbol{z}}{||\boldsymbol{z}||_2} & \text{otherwise} \end{cases}$$

$\mathcal{C}$: Set of non-negative reals



Projection = Set each negative entry in $\boldsymbol{z}$ to be zero

$$\hat{\boldsymbol{z}}_i = \begin{cases} z_i & \text{if } z_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Constrained Opt. via Lagrangian

- Consider the following constrained minimization problem (using f instead of L)

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}), \quad \text{s.t.} \quad g(\boldsymbol{w}) \leq 0$$

- Note: If constraints of the form $g(\boldsymbol{w}) \geq 0$, use $-g(\boldsymbol{w}) \leq 0$
- Can handle multiple inequality and equality constraints too (will see later)
- Can transform the above into the following equivalent unconstrained problem

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}) + c(\boldsymbol{w})$$

$$c(\boldsymbol{w}) = \max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) = \begin{cases} \infty, & \text{if } g(\boldsymbol{w}) > 0 \quad \text{(constraint violated)} \\ 0 & \text{if } g(\boldsymbol{w}) \leq 0 \quad \text{(constraint satisfied)} \end{cases}$$

- Our problem can now be written as

$$\boxed{\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) \right\}}$$

Constrained Opt. via Lagrangian

The Lagrangian: $\mathcal{L}(\boldsymbol{w}, \alpha)$

- Therefore, we can write our original problem as

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \max_{\alpha \geq 0} \alpha g(\boldsymbol{w}) \right\} = \arg\min_{\boldsymbol{w}} \left\{ \max_{\alpha \geq 0} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\}$$

- The Lagrangian is now optimized w.r.t. $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ (Lagrange multiplier)
- We can define Primal and Dual problem as

$$\hat{\boldsymbol{w}}_P = \arg\min_{\boldsymbol{w}} \left\{ \max_{\alpha \geq 0} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\} \quad \text{(Primal Problem)}$$

$$\hat{\boldsymbol{w}}_D = \arg\max_{\alpha \geq 0} \left\{ \min_{\boldsymbol{w}} \{ f(\boldsymbol{w}) + \alpha g(\boldsymbol{w}) \} \right\} \quad \text{(Dual Problem)}$$

Both equal if $f(\boldsymbol{w})$ and the set $g(\boldsymbol{w}) \leq 0$ are convex

$$\alpha_D g(\hat{\boldsymbol{w}}_D) = 0$$

complimentary slackness/Karush-Kuhn-Tucker (KKT) condition

Constrained Opt. with Multiple Constraints

- We can also have multiple inequality and equality constraints

$$
\begin{aligned}
\hat{\boldsymbol{w}} \quad &= \quad \arg\min_{\boldsymbol{w}} f(\boldsymbol{w}) \\
\text{s.t.} \quad & \quad g_i(\boldsymbol{w}) \leq 0, \quad i = 1, \ldots, K \\
& \quad h_j(\boldsymbol{w}) = 0, \quad j =, 1, \ldots, L
\end{aligned}
$$

Note that, unlike $\boldsymbol{\alpha}$, Lagrange variables $\boldsymbol{\beta}$ of the equality constraints are unconstrained

Exercise: Justify why!

- Introduce Lagrange multipliers $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_K]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_L]$
- The Lagrangian based primal and dual problems will be

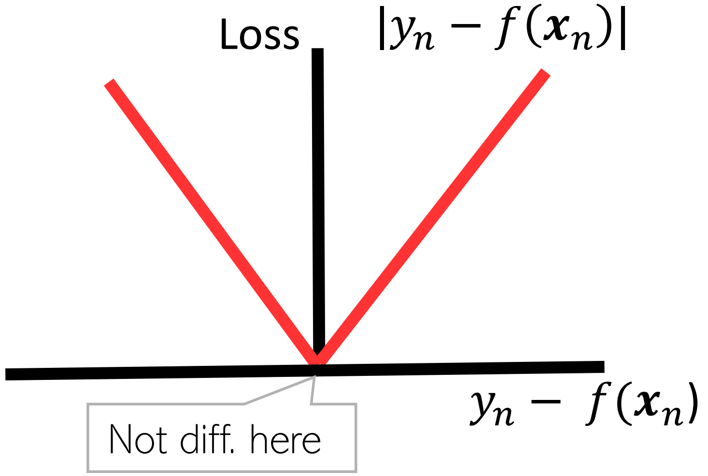
$$\hat{\boldsymbol{w}}_P = \arg\min_{\boldsymbol{w}} \left\{ \max_{\alpha \geq 0, \beta} \left\{ f(\boldsymbol{w}) + \sum_{i=1}^{K} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{L} \beta_j h_j(\boldsymbol{w}) \right\} \right\}$$

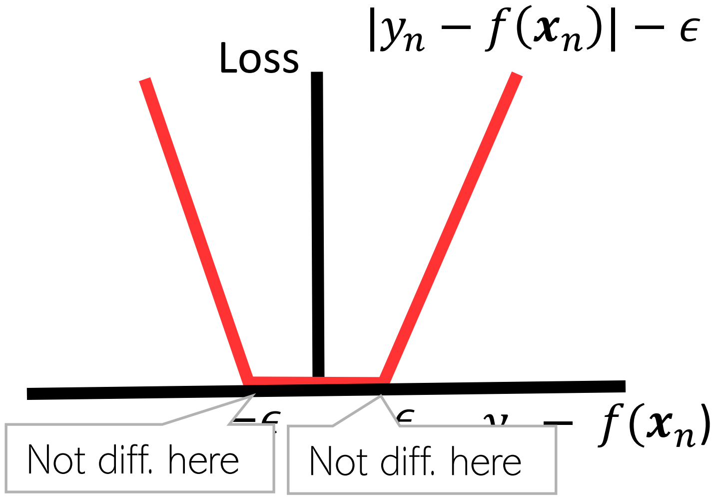
$$\hat{\boldsymbol{w}}_D = \arg\max_{\alpha \geq 0, \beta} \left\{ \min_{\boldsymbol{w}} \left\{ f(\boldsymbol{w}) + \sum_{i=1}^{K} \alpha_i g_i(\boldsymbol{w}) + \sum_{j=1}^{L} \beta_j h_j(\boldsymbol{w}) \right\} \right\}$$

Optimization of Non-differentiable Functions

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 ϵ-insensitive loss; even ℓ_1 norm regularizer is non-diff

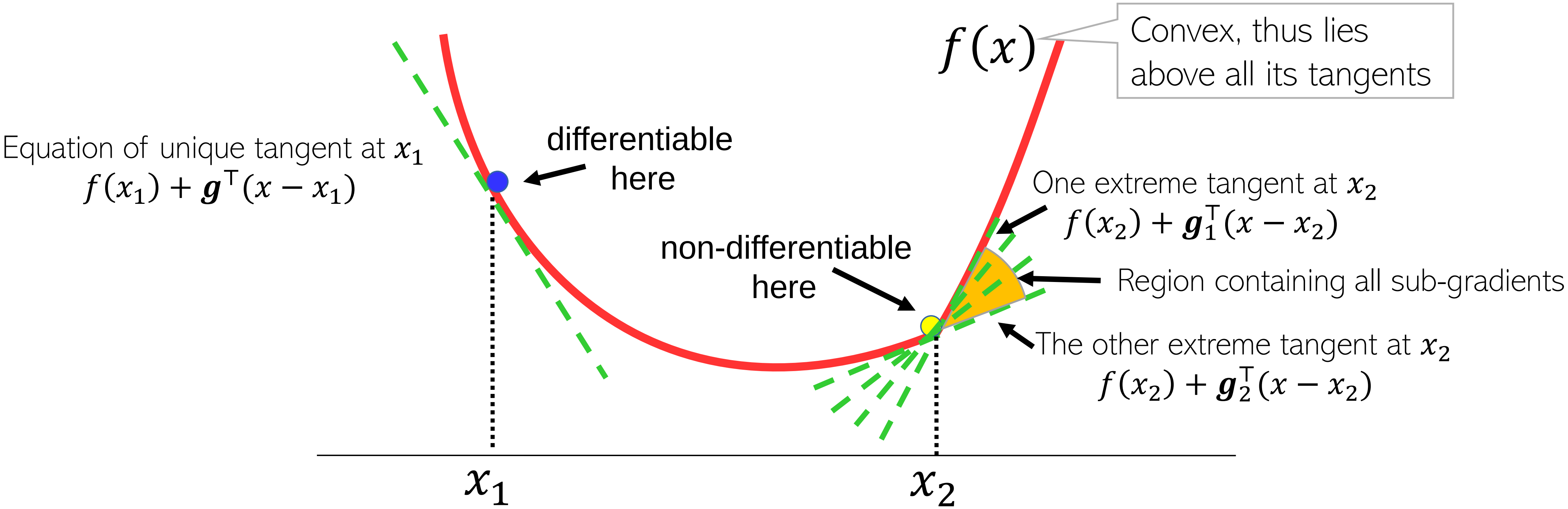




- Basically, any function in which there are points with kink is non-diff
  - At such points, the function is non-differentiable and thus gradients not defined
  - Reason: Can't define a unique tangent at such points

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. $\forall \boldsymbol{x}$

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

Sub-gradients, Sub-differential, and Some Rules

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

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

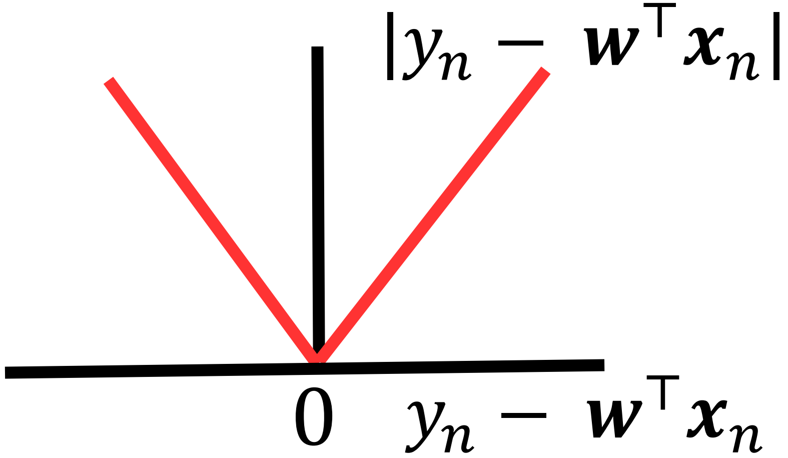
- Some basic rules of sub-diff calculus to keep in mind

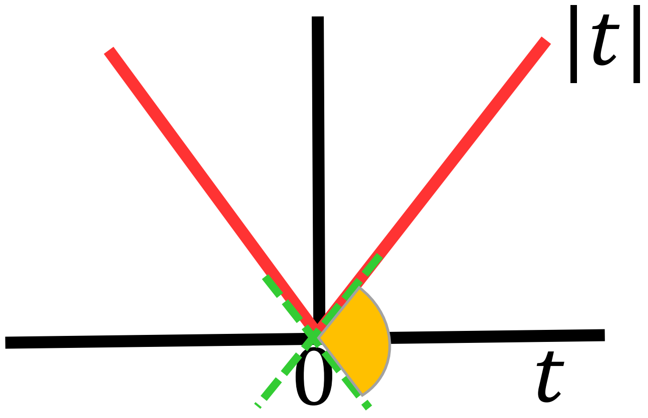
  The affine transform rule is a special case of the more general chain rule

  - 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: $\partial f(\mathbf{a}^\top \mathbf{x} + b) = \mathbf{a} \cdot \partial f(t) = \{\mathbf{a} \cdot c : c \in \partial f(t)\}$, 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 $\boldsymbol{x}_*$, i.e., $\mathbf{0} \in \partial f(\boldsymbol{x}_*)$

Sub-Gradient For Absolute Loss Regression





Using max rule of sub-differentials and using $|t| = \max\{t, -t\}$

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

- The loss function for linear reg. with absolute loss: $\boldsymbol{L(w)} = |y_n - \boldsymbol{w}^\top \boldsymbol{x}_n|$
- Non-differentiable at $y_n - \boldsymbol{w}^\top \boldsymbol{x}_n = 0$
- Can use the affine transform and max rule of sub-diff calculus
- Assume $\boldsymbol{t} = y_n - \boldsymbol{w}^\top \boldsymbol{x}_n$. Then $\boldsymbol{\partial L(w)} = -\boldsymbol{x}_n \partial|t|$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \times 1 = -\boldsymbol{x}_n$ if $t > 0$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \times -1 = \boldsymbol{x}_n$ if $t < 0$
  - $\partial L(\boldsymbol{w}) = -\boldsymbol{x}_n \times c = -c\boldsymbol{x}_n$ 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

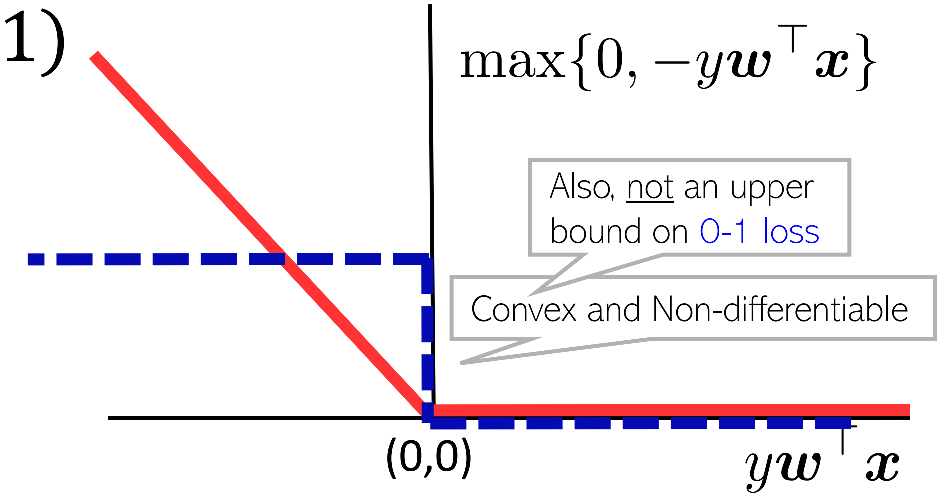
- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2,\ldots$ (or until convergence)
  - Calculate the sub-gradient $\boldsymbol{g}^{(t)} \in \partial L(\boldsymbol{w}^{(t)})$
  - Set the learning rate η_t
  - Move in the opposite direction of subgradient

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

Detour: Other Loss Functions for Binary Classifn

- For an ideal loss function, assuming $y_n \in (-1, +1)$
  - Large positive $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \Rightarrow$ small/zero loss
  - Large negative $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \Rightarrow$ large/non-zero loss
  - Small (large) loss if predicted probability of the the true label is large (small)

"Perceptron" Loss

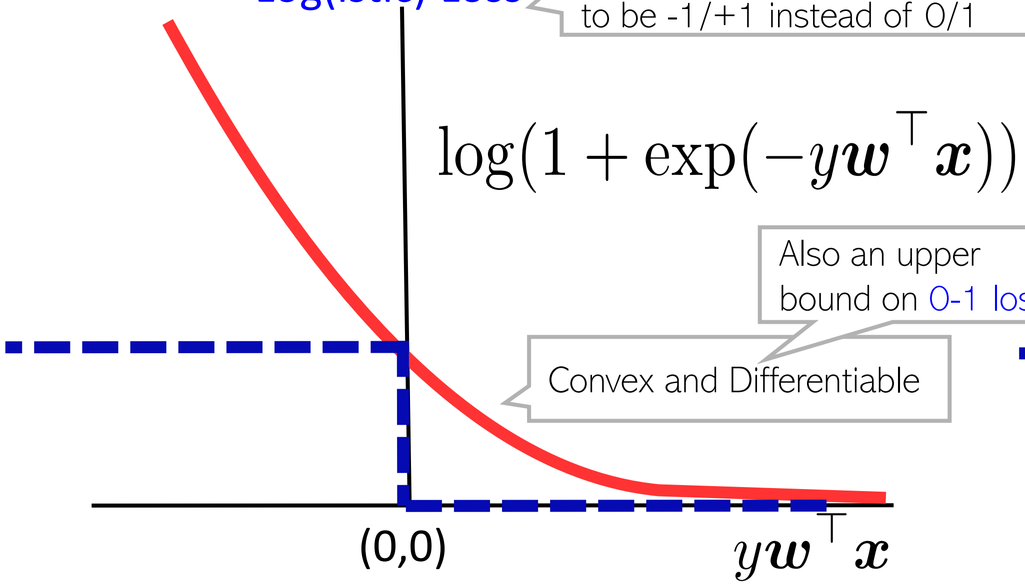


Also, not an upper bound on 0-1 loss

Convex and Non-differentiable

Log(istic) Loss

Same as cross-entropy loss (logistic reg.) if we assume labels to be -1/+1 instead of 0/1

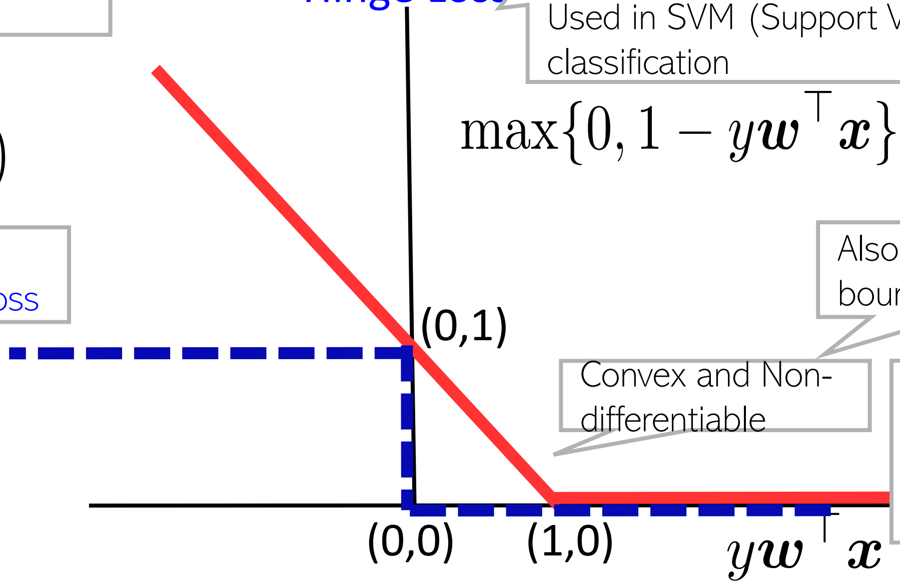


Also an upper bound on 0-1 loss

Convex and Differentiable

Hinge Loss

Very popular like cross-entropy loss. Used in SVM (Support Vector Machine) classification



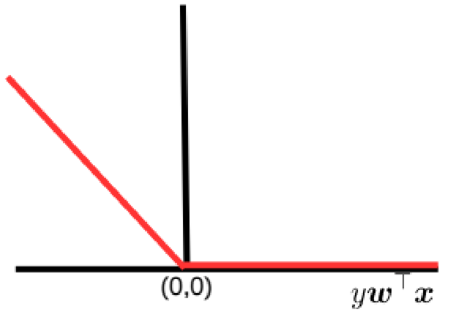
Also an upper bound on 0-1 loss

Convex and Non-differentiable

There are also multi-class variants of Perceptron and Hinge loss. Will see later

Subgradient Descent for Perceptron Loss

- Perceptron loss for binary classification: $L(w) = \sum_{n=1}^{N} \max\{0, -y_n \boldsymbol{w}^\top \boldsymbol{x}_n\}$



Subgradients w.r.t. $\boldsymbol{w}$

Using max rule of sub-differentials

$$\boldsymbol{g}_n = \begin{cases} 0 & \text{for} \quad y_n \boldsymbol{w}^\top \boldsymbol{x}_n > 0 \\ -y_n \boldsymbol{x}_n & \text{for} \quad y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0 \\ c y_n \boldsymbol{x}_n & \text{for} \quad y_n \boldsymbol{w}^\top \boldsymbol{x}_n = 0 \end{cases} \quad (\text{where } c \in [-1,0])$$

- If we pick $c = -1$ then $\boldsymbol{g}_n = -\mathbb{I}[y_n \boldsymbol{w}^\top \boldsymbol{x}_n \leq 0] y_n \boldsymbol{x}_n$
- Stochastic subgradient descent for Perceptron loss will have updates of the form

$$\text{if} \quad y_n \boldsymbol{w}^{(t)^\top} \boldsymbol{x}_n \leq 0$$

Meaning: The current weight vector does not correctly predict the label of $\boldsymbol{x}_n$

"mistake-driven" update (update only when the current weights make a mistake)

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \eta_t y_n \boldsymbol{x}_n$$

Perceptron Algorithm for Binary Classification

- Stochastic sub-grad desc on Perceptron loss is also known as the Perceptron algorithm

Note: An example may get chosen several times during the entire run

- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$, set $\eta_t = 1$
- For iteration $t = 0,1,2,\ldots$ (or until convergence)
  - Pick a training example $(\boldsymbol{x}_n, y_n)$ randomly
  - If $y_n {\boldsymbol{w}^{(t)}}^\top \boldsymbol{x}_n \leq 0$ (i.e., mistake) then update

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + y_n \boldsymbol{x}_n$$

Updates are "corrective": If $y_n = +1$ and $\boldsymbol{w}^\top \boldsymbol{x}_n < 0$, after the update $\boldsymbol{w}^\top \boldsymbol{x}_n$ will be less negative. Likewise, if $y_n = -1$ and $\boldsymbol{w}^\top \boldsymbol{x}_n > 0$, after the update $\boldsymbol{w}^\top \boldsymbol{x}_n$ will be less positive

If training data is linearly separable, the Perceptron algo will converge in a finite number of iterations (Block & Novikoff theorem)

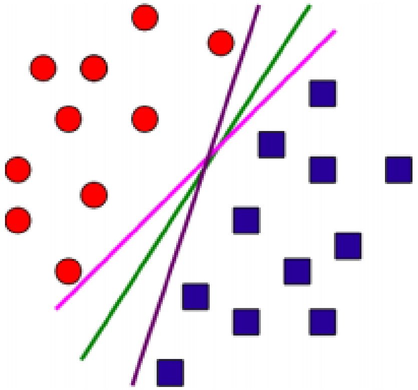
- An example of an online learning algorithm (processes one training ex. at a time)
- Assuming $\boldsymbol{w}^{(0)} = \boldsymbol{0}$, easy to see that the final $\boldsymbol{w}$ has the form $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$
  - α_n is total number of mistakes made by the algorithm on example $(\boldsymbol{x}_n, y_n)$
  - Important: $\boldsymbol{w}$ in many classification/regression models can be written as $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$

Thus $\boldsymbol{w}$ is In the span of training inputs

Meaning of α_n may be different depending on the problem

Perceptron and (lack of) Margins

- Perceptron would learn a hyperplane (of many possible) that separates the classes



Basically, it will learn the hyperplane which corresponds to the $\boldsymbol{w}$ that minimizes the Perceptron loss

Kind of an "unsafe" situation to have – ideally would like it to be reasonably away from closest training examples from either class

- Doesn't guarantee any "margin" around the hyperplane
  - The hyperplane can get arbitrarily close to some training example(s) on either side
  - This may not be good for generalization performance

$\gamma > 0$ is some pre-specified margin

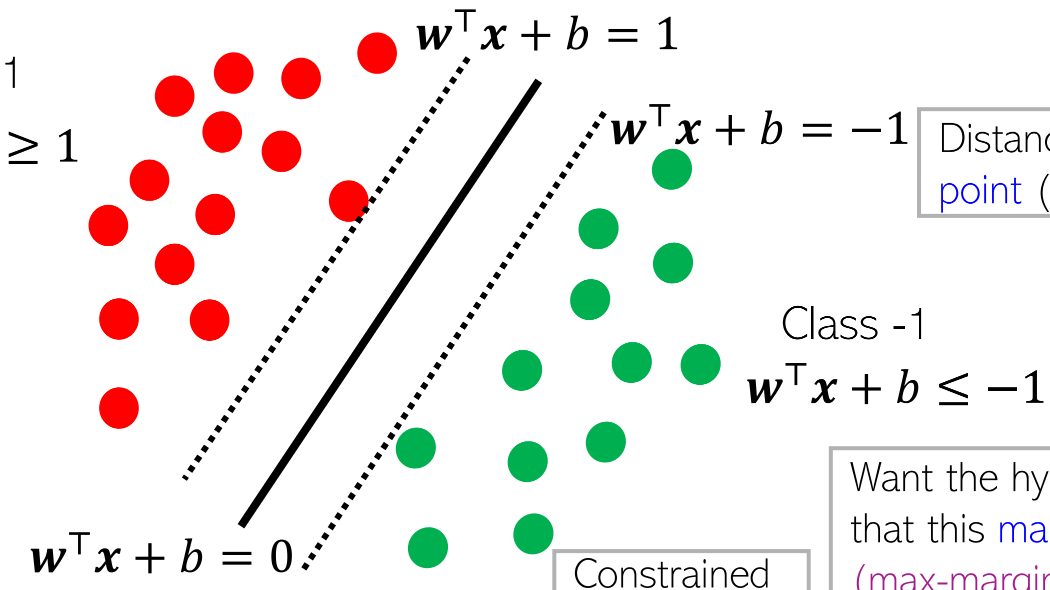
- Can artificially introduce margin by changing the mistake condition to $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \leq \gamma$
- Methods like logistic regression also do not guarantee large margins
- Support Vector Machine (SVM) does it directly by learning the max. margin hyperplane

Learning Large-Margin Hyperplanes

Support Vector Machine (SVM)

SVM originally proposed by Vapnik and colleagues in early 90s

- Hyperplane based classifier. Ensures a large margin around the hyperplane
- Will assume a linear hyperplane to be of the form $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ (nonlinear ext. later)



$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1$ if $y_n = +1$

$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1$ if $y_n = -1$

$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad \forall n$

Distance from the closest point (on either side)

"Margin" of the hyperplane

Distance of an input $\boldsymbol{x}_n$ from the h.p.

$$\gamma = \min_{1 \leq n \leq N} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_n + b|}{||\boldsymbol{w}||} = \frac{1}{||\boldsymbol{w}||}$$

Want the hyperplane $(\boldsymbol{w}, b)$ such that this margin is maximized (max-margin hyperplane) and

$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad \forall n$

Constrained optimization problem

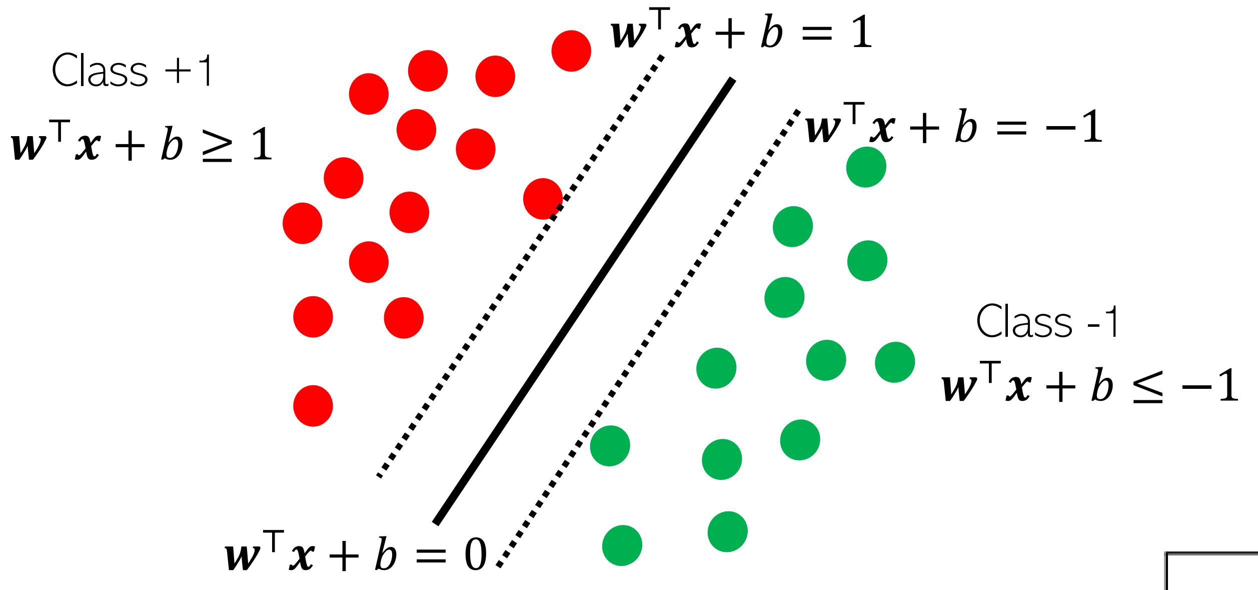
$$\text{Total margin} = \frac{2}{||\boldsymbol{w}||}$$

- Two other "supporting" hyperplanes defining a "no man's land"
  - Ensure that zero training examples fall in this region (will relax later)
  - The SVM idea: Position the hyperplane s.t. this region is as "wide" as possible

The 1/-1 in supp. h.p. equations is arbitrary; can replace by any scalar m/-m and solution won't change, except a simple scaling of $\boldsymbol{w}$

Hard-Margin SVM

- Hard-Margin: Every training example must fulfil margin condition $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$
- Meaning: Must not have any example in the no-man's land



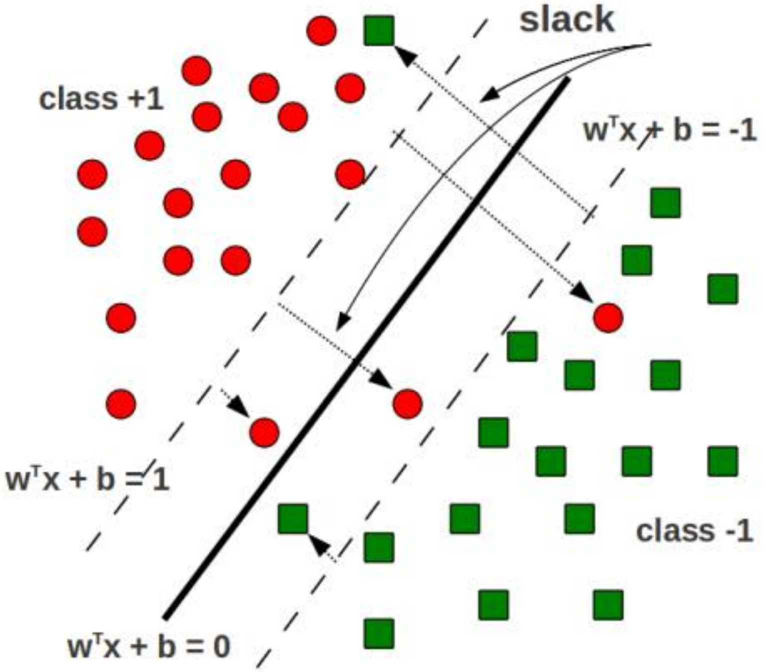
- Also want to maximize margin $2\gamma = \frac{2}{\|\boldsymbol{w}\|}$
- Equivalent to <u>minimizing</u> $\|\boldsymbol{w}\|^2$ or $\frac{\|\boldsymbol{w}\|^2}{2}$
- The objective func. for hard-margin SVM

$$\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w}, b) = \frac{||\boldsymbol{w}||^2}{2}$$
$$\text{subject to} \quad y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

Constrained optimization problem with N inequality constraints. Objective and constraints both are convex

Lagrange based optimization can be used to solve it

Soft-Margin SVM (More Commonly Used)



- Allow some training examples to fall within the no-man's land (margin region)

Helps in getting a wider margin (and better generalization)

- Even okay for some training examples to fall totally on the wrong side of h.p.
- Extent of "violation" by a training input $(\boldsymbol{x}_n, y_n)$ is known as slack $\xi_n \geq 0$
- $\xi_n > 1$ means totally on the wrong side

Note/verify that the slack for each training example is just the hinge loss

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 - \xi_n \quad \text{if} \quad y_n = +1$$

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 + \xi_n \quad \text{if} \quad y_n = -1$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \quad \forall n$$

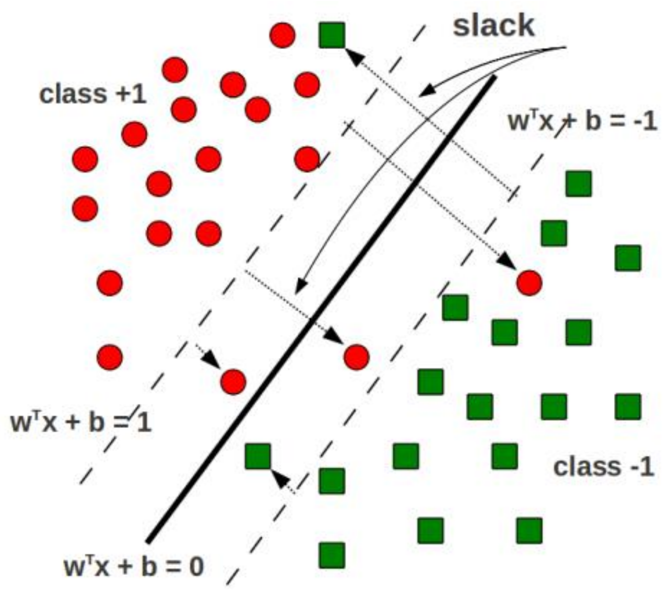
Soft-margin constraint

$$\xi_n = \max\{0, 1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)\}$$

Soft-Margin SVM (Contd)

- Goal: Still want to maximize the margin such that
  - Soft-margin constraints $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$ are satisfied for all training ex.
  - Do not have too many margin violations (sum of slacks $\sum_{n=1}^{N} \xi_n$ should be small)

Sum of slacks is like the training error



- The objective func. for soft-margin SVM

Lagrange based optimization can be used to solve it

Inversely prop. to margin

Trade-off hyperparam

training error

Constrained optimization problem with $2N$ inequality constraints. Objective and constraints both are convex

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad f(\boldsymbol{w},b,\boldsymbol{\xi}) = \frac{||\boldsymbol{w}||^2}{2} + C\sum_{n=1}^{N}\xi_n$$
$$\text{subject to} \quad y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N$$

- Hyperparameter C controls the trade off between large margin and small training error (need to tune)
  - Too large C: small training error but also small margin (bad)
  - Too small C: large margin but large training error (bad)