Slides adapted from Prof Carpuat and Duraiswami

CMSC 422 Introduction to Machine Learning

Lecture 15 (Sub)gradient Descent

Furong Huang / furongh@umd.edu

UNIVERSITY OF MARYLAND

Topics

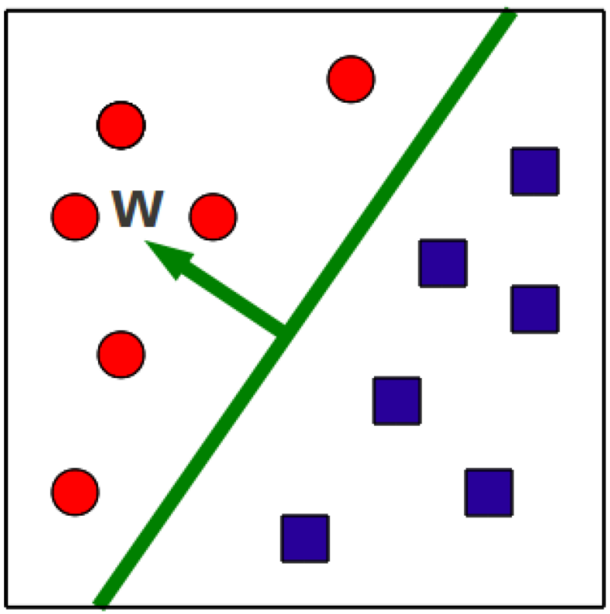
Linear Models
- Loss functions
- Regularization

Gradient Descent

Calculus refresher
- Convexity
- Gradients

[CIML Chapter 6]

Binary classification via hyperplanes



A classifier is a hyperplane (w,b)

At test time, we check on what side of the hyperplane examples fall

$$\hat{y} = sign(w^T x + b)$$

This is a **linear classifier**

Because the prediction is a linear combination of feature values x

Task: Binary Classification

Given:

1. An input space $\mathcal{X}$
2. An unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \{-1, +1\}$

Compute: A function f minimizing: $\mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}\left[f(\boldsymbol{x}) \neq y\right]$

Learning a Linear Classifier as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

$\mathbb{I}(.)$ Indicator function: 1 if (.) is true, 0 otherwise

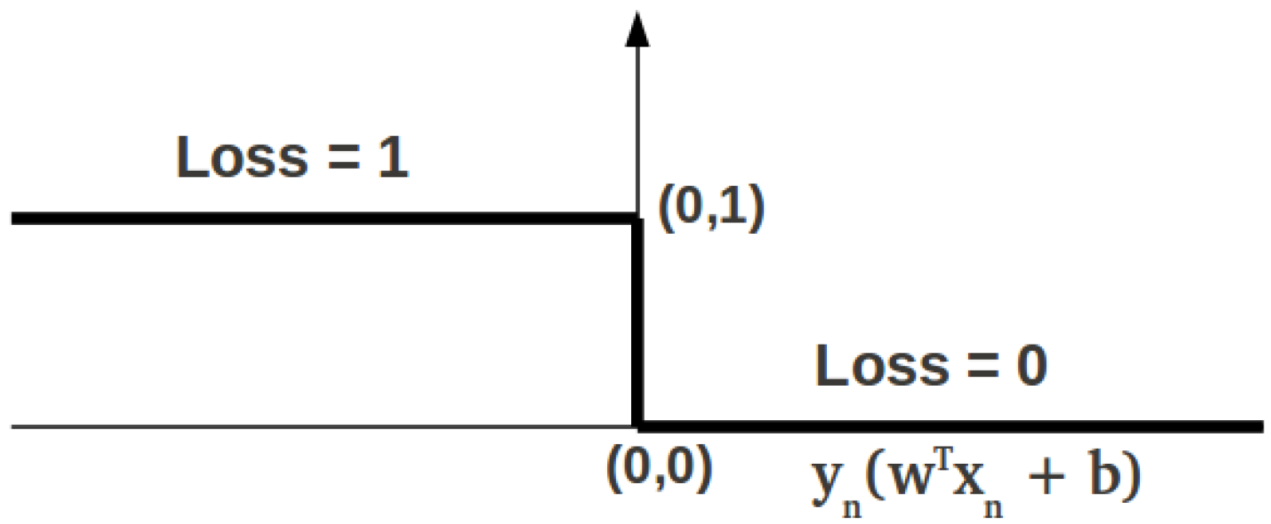
The loss function above is called the 0-1 loss

Learning a Linear Classifier as an Optimization Problem

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

- **Problem:** The 0-1 loss above is NP-hard to optimize exactly/approximately in general
- **Solution:** Different loss function approximations and regularizers lead to specific algorithms
  (e.g., perceptron, support vector machines, logistic regression, etc.)

The 0-1 Loss



Small changes in w,b can lead to big changes in the loss value

0-1 loss is non-smooth, non-convex

Calculus refresher:

Smooth functions, convex functions

Approximating the 0-1 loss with surrogate loss functions

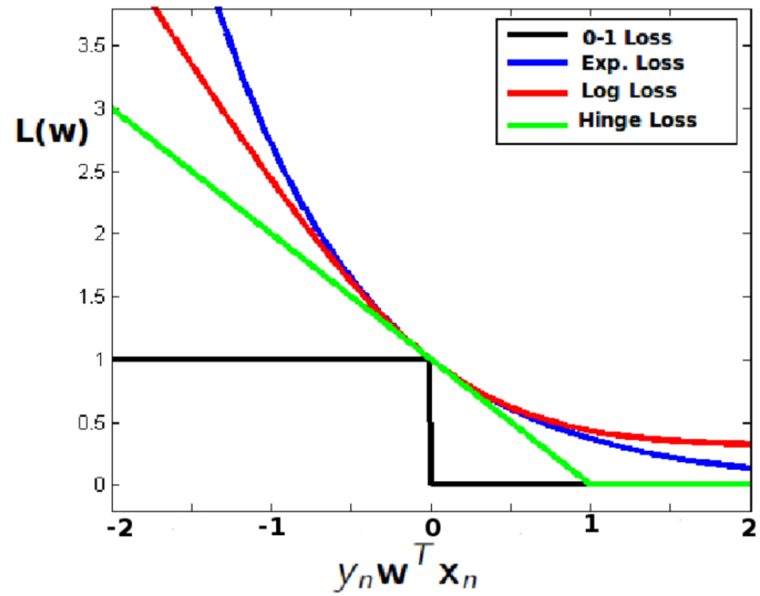
Examples (with b = 0)

Hinge loss $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$

Log loss $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$

Exponential loss $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

All are convex upper-bounds on the 0-1 loss



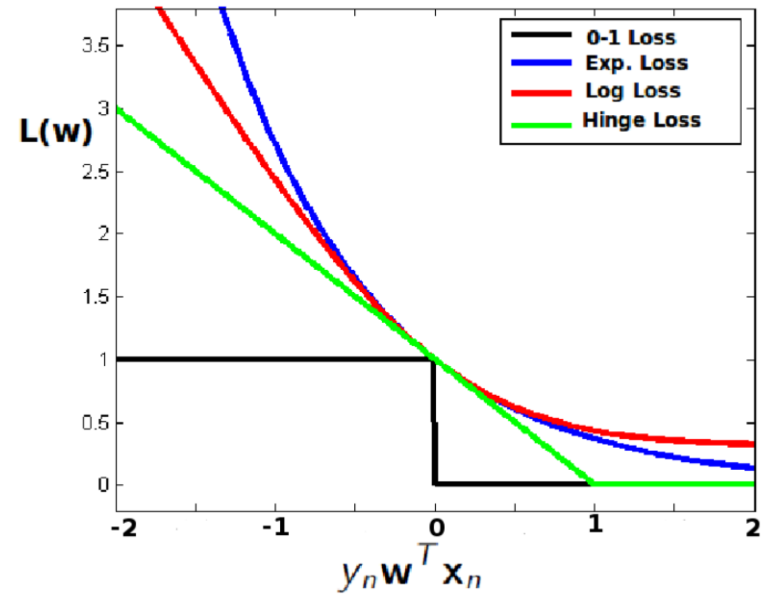
Approximating the 0-1 loss with surrogate loss functions

Examples (with b = 0)

Hinge loss $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$

Log loss $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$

Exponential loss $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$



Q: Which of these loss functions is not smooth?

Approximating the 0-1 loss with surrogate loss functions

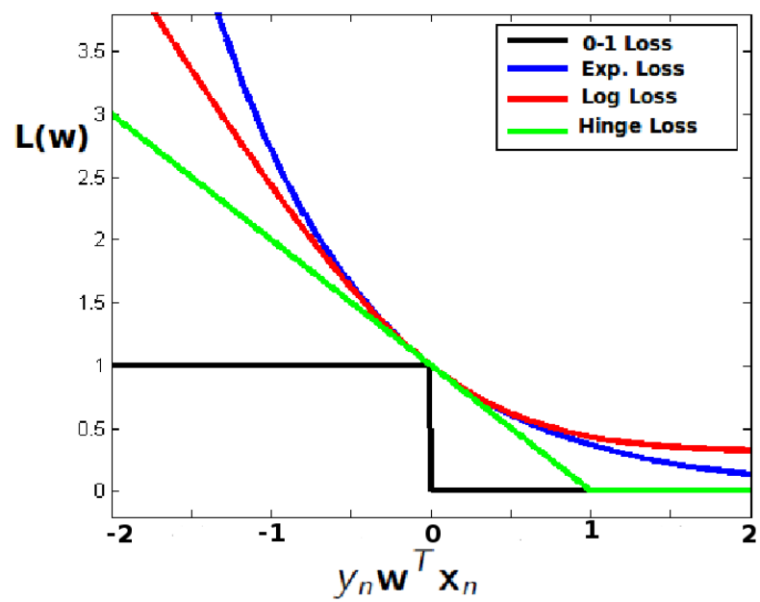
Examples (with b = 0)

Hinge loss $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$

Log loss $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$

Exponential loss $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

Q: Which of these loss functions is most sensitive to outliers?



Casting Linear Classification as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

$\mathbb{I}(.)$ Indicator function: 1 if (.) is true, 0 otherwise

The loss function above is called the 0-1 loss

The regularizer term

Goal: find simple solutions (inductive bias)

Ideally, we want most entries of w to be zero, so prediction depends only on a small number of features.

Formally, we want to minimize:

$$R^{cnt}(\mathbf{w}, b) = \sum_{d=1}^{D} \mathbb{I}(w_d \neq 0)$$

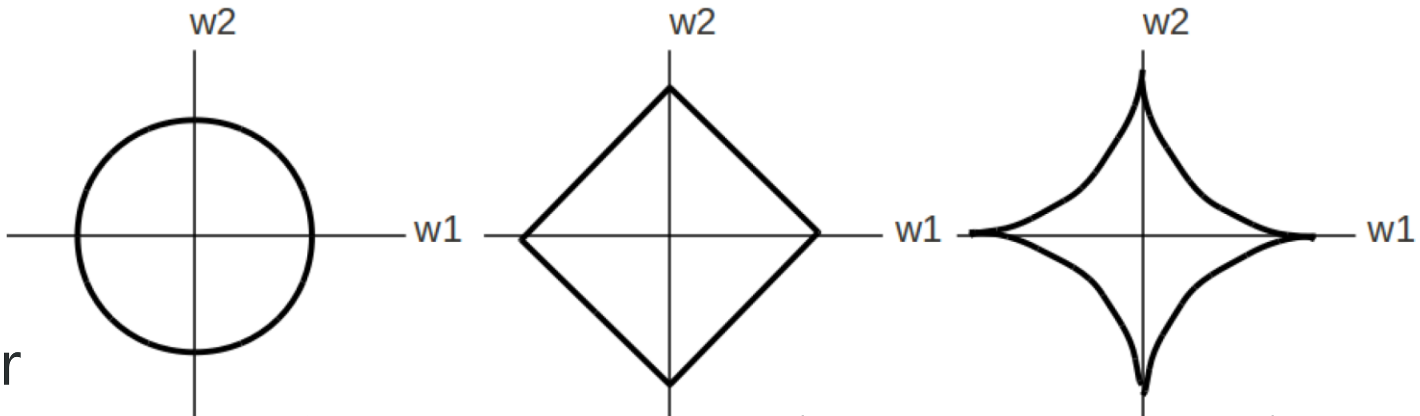
That's NP-hard, so we use approximations instead.

- E.g., we encourage w_d's to be small

Norm-based Regularizers

l_p norms can be used as regularizers

$$||\mathbf{w}||_2^2 = \sum_{d=1}^{D} w_d^2$$
$$||\mathbf{w}||_1 = \sum_{d=1}^{D} |w_d|$$
$$||\mathbf{w}||_p = (\sum_{d=1}^{D} w_d^p)^{1/p}$$



Contour plots for p = 2 p = 1 p < 1

Norm-based Regularizers

l_p norms can be used as regularizers

Smaller p favors sparse vectors w

  i.e. most entries of w are close or equal to 0

  l_2 norm: convex, smooth, easy to optimize

l_1 norm: encourages sparse w, convex, but not smooth at axis points

$p < 1$: norm becomes non convex and hard to optimize

Casting Linear Classification as an Optimization Problem

Objective function

Loss function measures how well classifier fits training data

Regularizer prefers solutions that generalize well

$$\min_{\mathbf{w},b} L(\mathbf{w},b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w},b)$$

$\mathbb{I}(.)$ Indicator function: 1 if (.) is true, 0 otherwise

The loss function above is called the 0-1 loss

What is the perceptron optimizing?

Algorithm 5 PerceptronTrain(**D**, *MaxIter*)

$w_d \leftarrow 0$, for all $d = 1 \ldots D$ — // initialize weights
$b \leftarrow 0$ — // initialize bias
for $iter = 1 \ldots MaxIter$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ — // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + y x_d$, for all $d = 1 \ldots D$ — // update weights
      $b \leftarrow b + y$ — // update bias
    end if
  end for
end for
return $w_0, w_1, \ldots, w_D, b$

Loss function is a variant of the hinge loss

$$\max\{0, -y_n(\mathbf{w}^T \mathbf{x}_n + b)\}$$

Recap: Linear Models

General framework for binary classification

Cast learning as optimization problem

Optimization objective combines 2 terms

- loss function: measures how well classifier fits training data
- Regularizer: measures how simple classifier is

- Does not assume data is linearly separable

Lets us separate model definition from training algorithm

Calculus refresher:
Gradients

Gradient descent

A general solution for our optimization problem

$$\min_{\mathbf{w},b} L(\mathbf{w}, b) = \min_{\mathbf{w},b} \sum_{n=1}^{N} \mathbb{I}(y_n(\mathbf{w}^T\mathbf{x}_n + b) < 0) + \lambda R(\mathbf{w}, b)$$

Idea: take iterative steps to update parameters in the direction of the gradient

Gradient descent algorithm

Objective function to minimize

Number of steps

Step size

Algorithm 22 $\textsc{GradientDescent}(\mathcal{F}, K, \eta_1, \ldots)$

$z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing

for $k = 1 \ldots K$ **do**

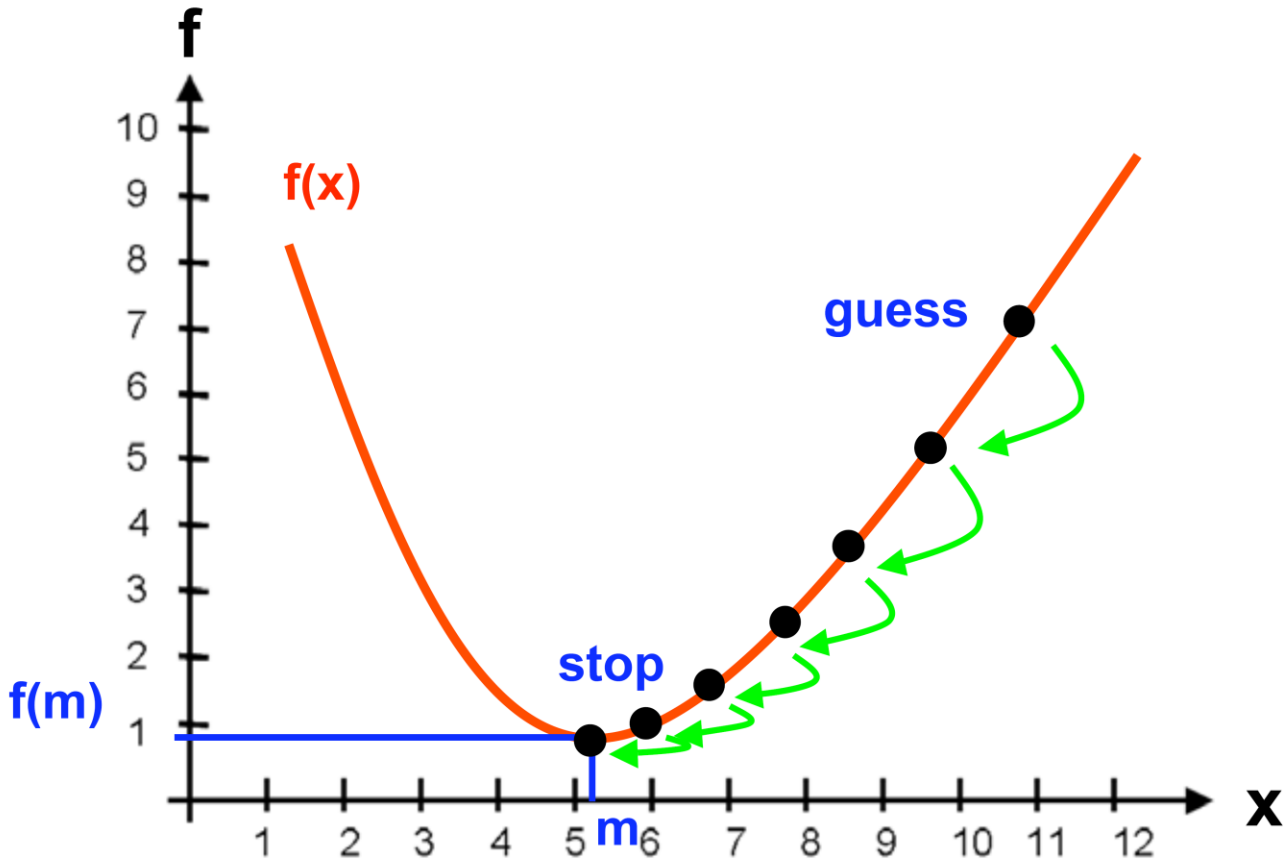
  $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location

  $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient

end for

return $z^{(K)}$

Illustrating gradient descent in 1-dimensional case



Gradient Descent

2 questions

- When to stop?
  - When the gradient gets close to zero
  - When the objective stops changing much
  - When the parameters stop changing much
  - Early
  - When performance on held-out dev set plateaus
- How to choose the step size?
  - Start with large steps, then take smaller steps

Now let's calculate gradients for multivariate objectives

Consider the following learning objective

$$\mathcal{L}(\boldsymbol{w}, b) = \sum_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \frac{\lambda}{2} ||\boldsymbol{w}||^2$$

What do we need to do to run gradient descent?

(1) Derivative with respect to b

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial}{\partial b} \sum_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + \frac{\partial}{\partial b} \frac{\lambda}{2} ||\boldsymbol{w}||^2 \tag{6.12}$$

$$= \sum_n \frac{\partial}{\partial b} \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] + 0 \tag{6.13}$$

$$= \sum_n \left(\frac{\partial}{\partial b} - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right) \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] \tag{6.14}$$

$$= -\sum_n y_n \exp\left[-y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\right] \tag{6.15}$$

(2) Gradient with respect to w

$$\nabla_{w}\mathcal{L} = \nabla_{w}\sum_{n}\exp\left[-y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\right] + \nabla_{w}\frac{\lambda}{2}\,||\boldsymbol{w}||^2 \tag{6.16}$$

$$= \sum_{n}\left(\nabla_{w} - y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\right)\exp\left[-y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\right] + \lambda\boldsymbol{w} \tag{6.17}$$

$$= -\sum_{n} y_n\boldsymbol{x}_n\exp\left[-y_n(\boldsymbol{w}\cdot\boldsymbol{x}_n+b)\right] + \lambda\boldsymbol{w} \tag{6.18}$$

Subgradients

Problem: some objective functions are not differentiable everywhere

Hinge loss, l1 norm

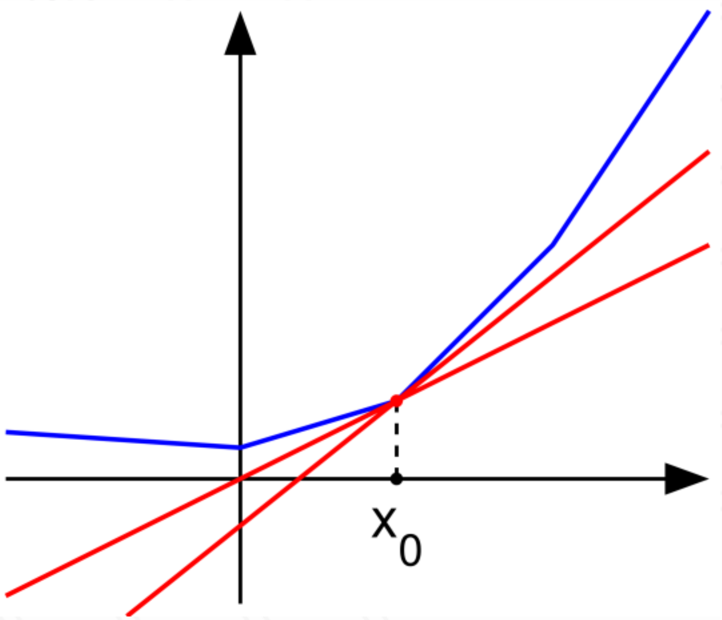
Solution: subgradient optimization

Let's ignore the problem, and just try to apply gradient descent anyway!!

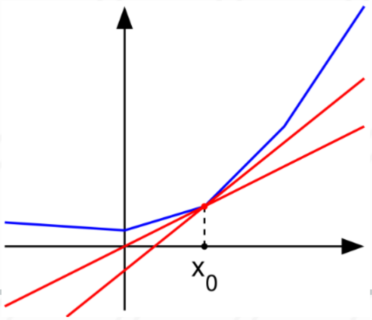
we will just differentiate by parts…

Subgradient Review

- Subgradient generalized the derivative to functions which are not differentiable.
- For any x_0 in the domain of the function one can draw a line which goes through the point (x_0, $f(x_0)$) and which is everywhere either **touching** or **below** the graph of f.
- Set-valued



Subgradient Review



Rigorously, a *subgradient* of a convex function $f:I\rightarrow\mathbf{R}$ at a point x_0 in the open interval I is a real number c such that

$$f(x) - f(x_0) \geq c(x - x_0)$$

for all x in I. One may show that the set of subgradients at x0 for a convex function is a nonempty closed interval [a, b], where a and b are the one-sided limits.

$$a = \lim_{x\to x_0^-} \frac{f(x) - f(x_0)}{x - x_0}$$

$$b = \lim_{x\to x_0^+} \frac{f(x) - f(x_0)}{x - x_0}$$

which are guaranteed to exist and satisfy a ≤ b.

- The set [a, b] of all subgradients is called the subgradients of the function f at x_0. Since f is convex, if its subdifferential at x_0 contains exactly one subgradient, then f is differentiable at x_0.

Approximating the 0-1 loss with surrogate loss functions

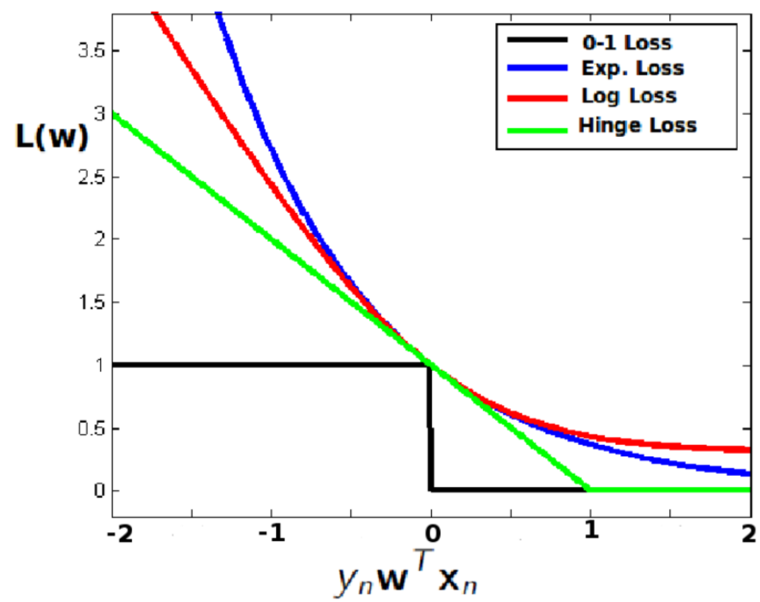
Examples (with b = 0)

Hinge loss $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$

Log loss $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$

Exponential loss $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

What if $b \neq 0$?



Example: subgradient of hinge loss

For a given example n

$$\partial_w \max\{0, 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b)\} \tag{6.22}$$

$$= \partial_w \begin{cases} 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ \text{-}\, y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \tag{6.23}$$

$$= \begin{cases} \partial_w 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ \text{-}\partial_w y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \tag{6.24}$$

$$= \begin{cases} \boldsymbol{0} & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) > 1 \\ \text{-}y_n \boldsymbol{x}_n & \text{otherwise} \end{cases} \tag{6.25}$$

Subgradient Descent for Hinge Loss

Algorithm 23 HINGEREGULARIZEDGD($\mathbf{D}$, λ, *MaxIter*)

$\boldsymbol{w} \leftarrow \langle 0, 0, \ldots 0 \rangle \quad , \quad b \leftarrow 0$ // initialize weights and bias
for $iter = 1 \ldots MaxIter$ **do**
  $\boldsymbol{g} \leftarrow \langle 0, 0, \ldots 0 \rangle \quad , \quad g \leftarrow 0$ // initialize gradient of weights and bias
  for all $(\boldsymbol{x}, y) \in \mathbf{D}$ **do**
    if $y(\boldsymbol{w} \cdot \boldsymbol{x} + b) \leq 1$ **then**
      $\boldsymbol{g} \leftarrow \boldsymbol{g} + y\, \boldsymbol{x}$ // update weight gradient
      $g \leftarrow g + y$ // update bias derivative
    end if
  end for
  $\boldsymbol{g} \leftarrow \boldsymbol{g} - \lambda \boldsymbol{w}$ // add in regularization term
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{g}$ // update weights
  $b \leftarrow b + \eta g$ // update bias
end for
return $\boldsymbol{w}, b$

What is the perceptron optimizing?

Algorithm 5 $\textsc{PerceptronTrain}(\mathbf{D}, \mathit{MaxIter})$

$w_d \leftarrow 0$, for all $d = 1 \dots D$ // initialize weights
$b \leftarrow 0$ // initialize bias
for $\mathit{iter} = 1 \dots \mathit{MaxIter}$ **do**
  for all $(x,y) \in \mathbf{D}$ **do**
    $a \leftarrow \sum_{d=1}^{D} w_d\, x_d + b$ // compute activation for this example
    if $ya \leq 0$ **then**
      $w_d \leftarrow w_d + y x_d$, for all $d = 1 \dots D$ // update weights
      $b \leftarrow b + y$ // update bias
    end if
  end for
end for
return $w_0, w_1, \dots, w_D, b$

Loss function is a variant of the hinge loss

$$\max\{0, -y_n(\mathbf{w}^T \mathbf{x}_n + b)\}$$

Recap: Linear Models

Lets us separate model definition from training algorithm **(Gradient Descent)**

Summary

Gradient descent

- A generic algorithm to minimize objective functions
- Works well as long as functions are well behaved (ie convex)
- Subgradient descent can be used at points where derivative is not defined
- Choice of step size is important

Optional: can we do better?

- For some objectives, we can find closed form solutions (see CIML 6.6)

UNIVERSITY OF MARYLAND

Furong Huang
4124 IRB, College Park, MD 20740
301.405.8010 / furongh@umd.edu