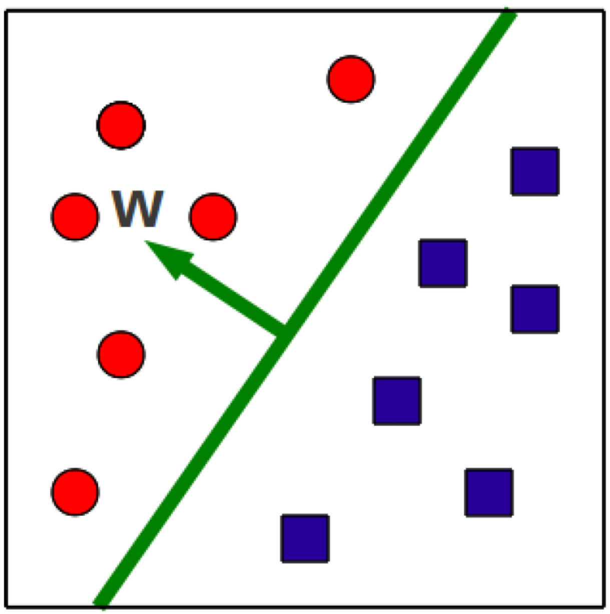
# Linear Models & Gradient Descent

CMSC 422

MARINE CARPUAT
marine@cs.umd.edu

Figures credit: Piyush Rai

# 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

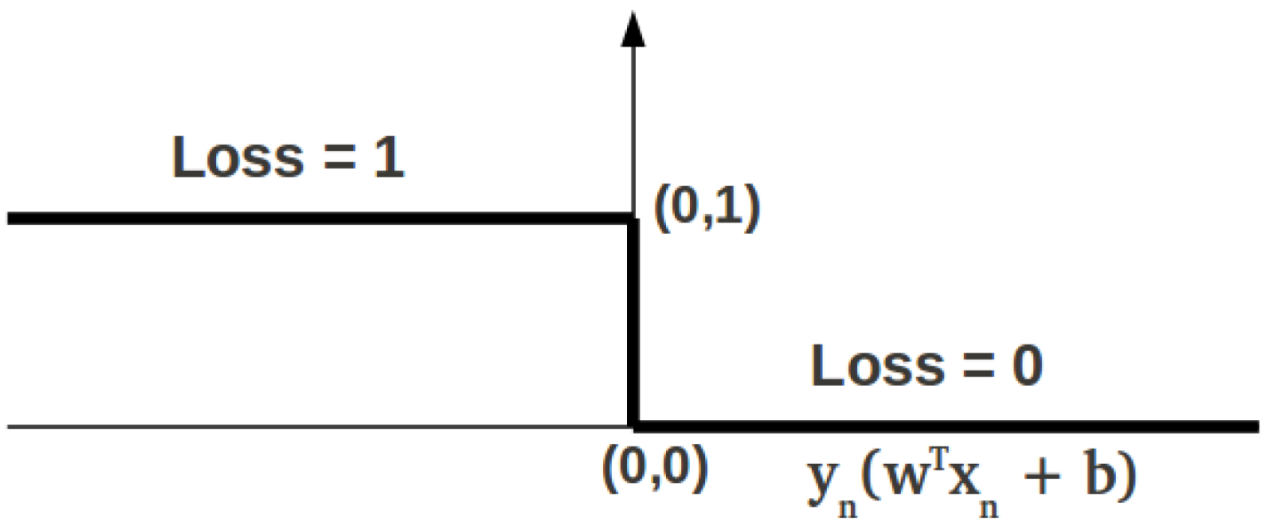
**Objective function**

**Loss function** measures how well classifier fits training data

**Regularizer** prefers solutions that generalize well

$$\min_{\mathbf{w}, b} L(\mathbf{w}, b)$$

# The 0-1 Loss

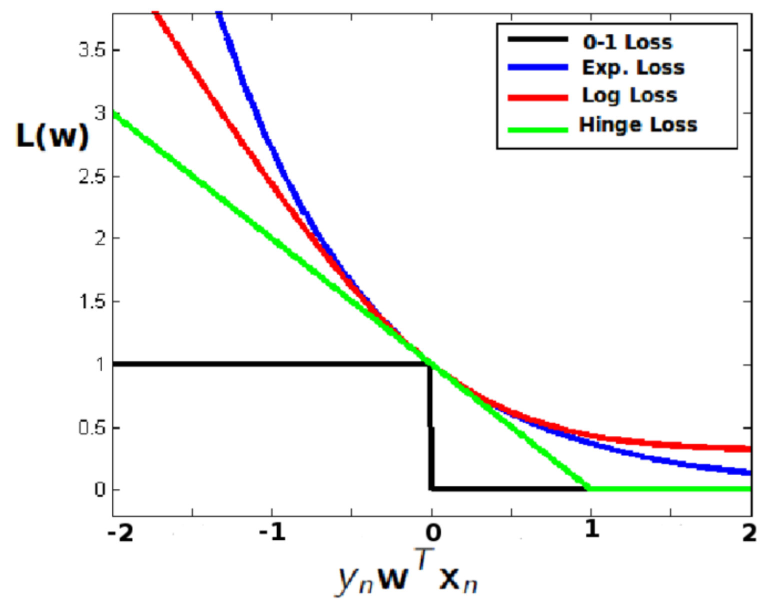



- Small changes in w,b can lead to big changes in the loss value
- 0-1 loss is non-smooth, non-convex

# Approximating the 0-1 loss with surrogate loss functions

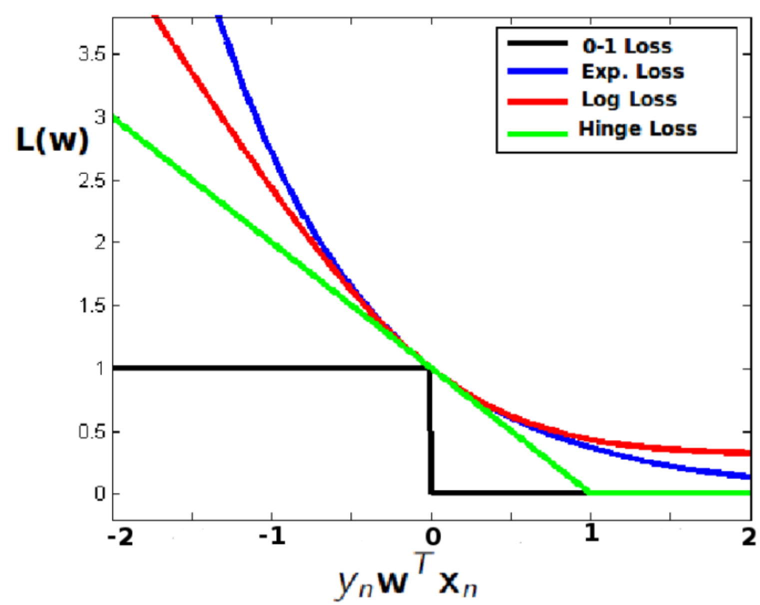
- Examples (with b = 0)
  - Hinge loss $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$

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



# Approximating the 0-1 loss with surrogate loss functions

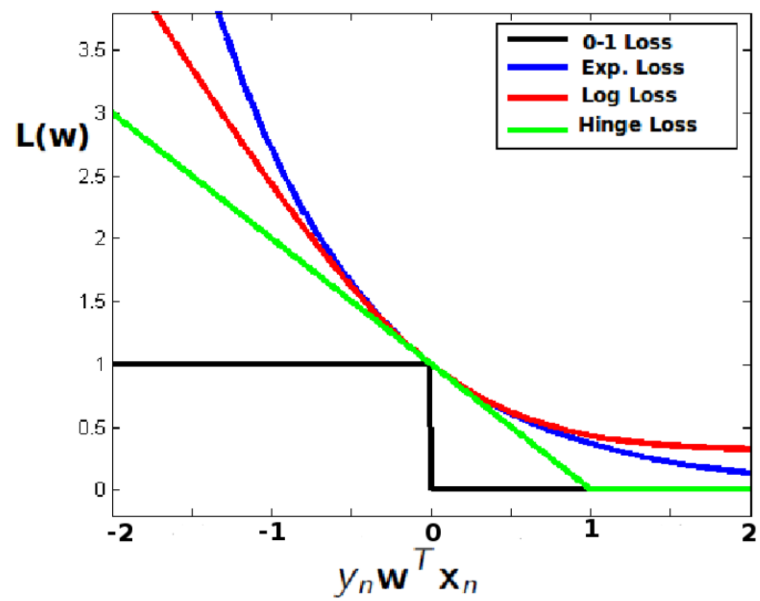
- Examples (with b = 0)
  - Hinge loss $[1 - y_n \mathbf{w}^T \mathbf{x}_n]_+ = \max\{0, 1 - y_n \mathbf{w}^T \mathbf{x}_n\}$
  - Log loss $\log[1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)]$
  - Exponential loss $\exp(-y_n \mathbf{w}^T \mathbf{x}_n)$




- **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)

- Example of simple solution
  - if most of w elements are zero, prediction depends only on a small number of features.
  - Formally, we want to minimize:

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

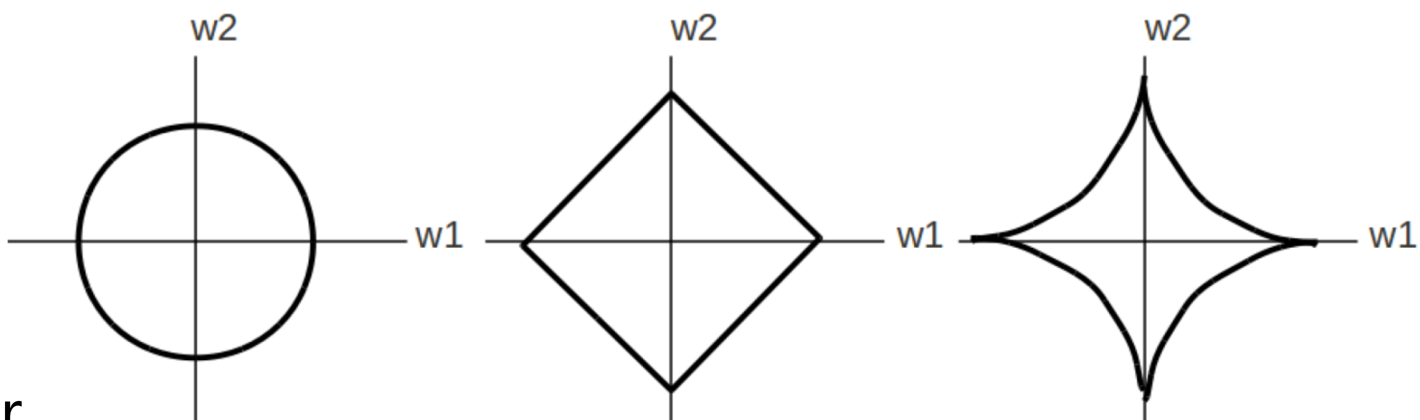
  - That's NP-hard, so we use approximations instead.

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

# 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 **(Gradient Descent)**

# 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** $\textsc{GradientDescent}(\mathcal{F}, K, \eta_1, \dots)$

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

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

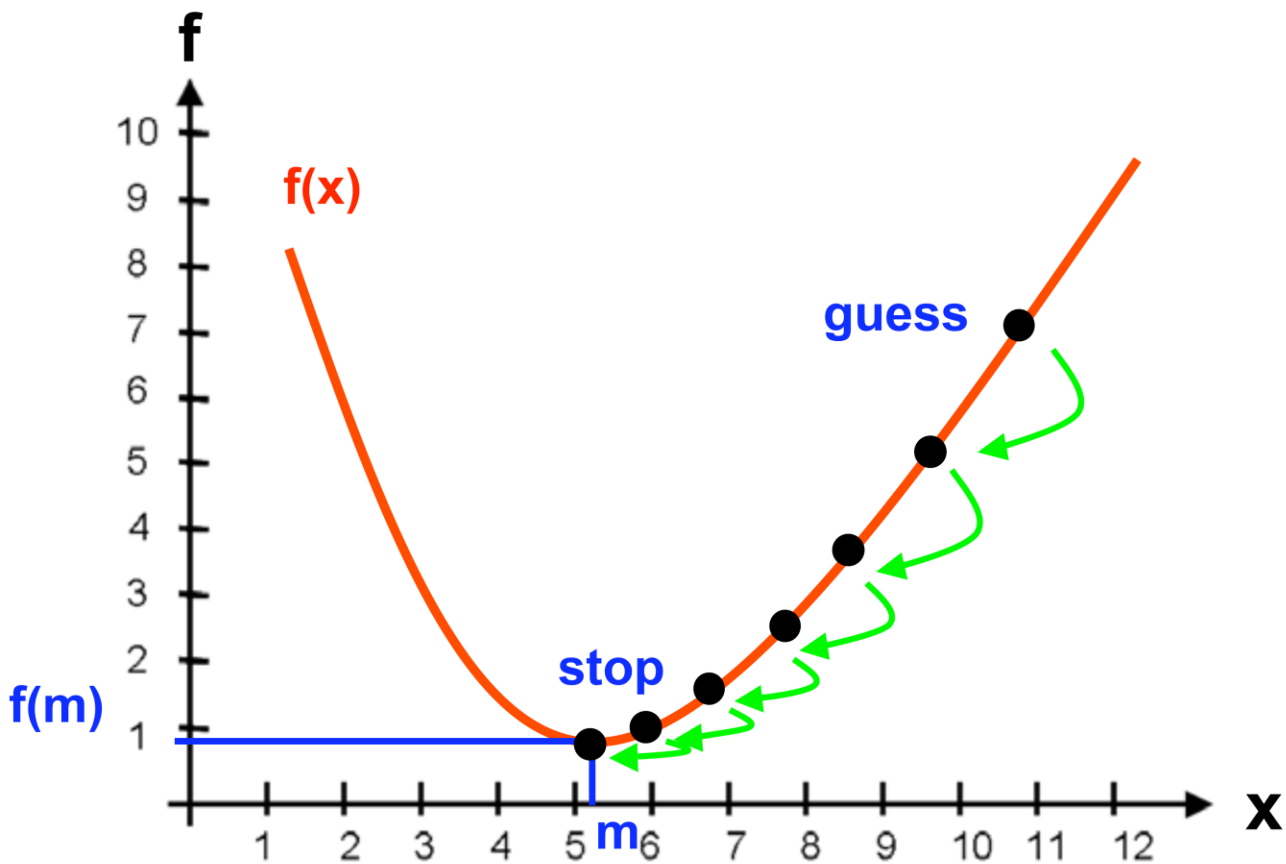
  $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location

  $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // 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?
  - How to choose the step size?

# 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

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

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

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

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

# (2) Gradient with respect to w

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

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

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

# 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 7.6)