Linear Models: (Sub)gradient Descent

CMSC 422

SOHEIL FEIZI

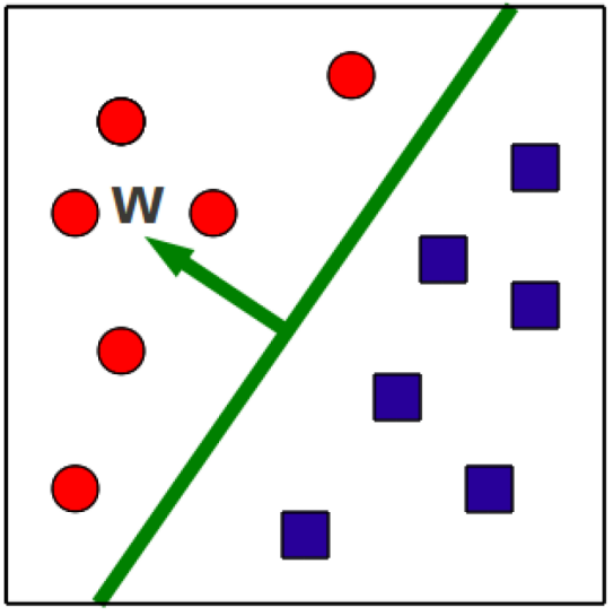
sfeizi@cs.umd.edu

Slides adapted from MARINE CARPUAT

Recap: Linear Models

- General framework for binary classification
- Cast learning as optimization problem
- Optimization objective combines 2 terms
  - Loss function
  - Regularizer
- Does not assume data is linearly separable
- Lets us separate model definition from training algorithm **(Gradient Descent)**

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

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

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

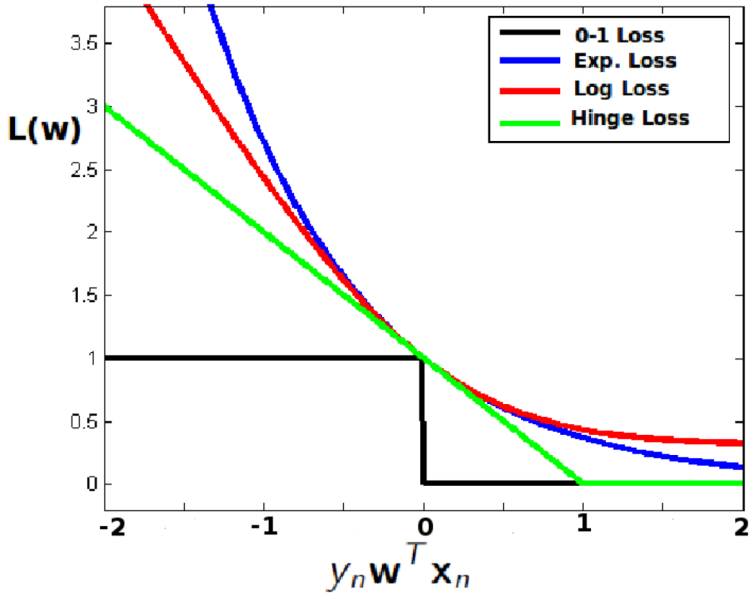


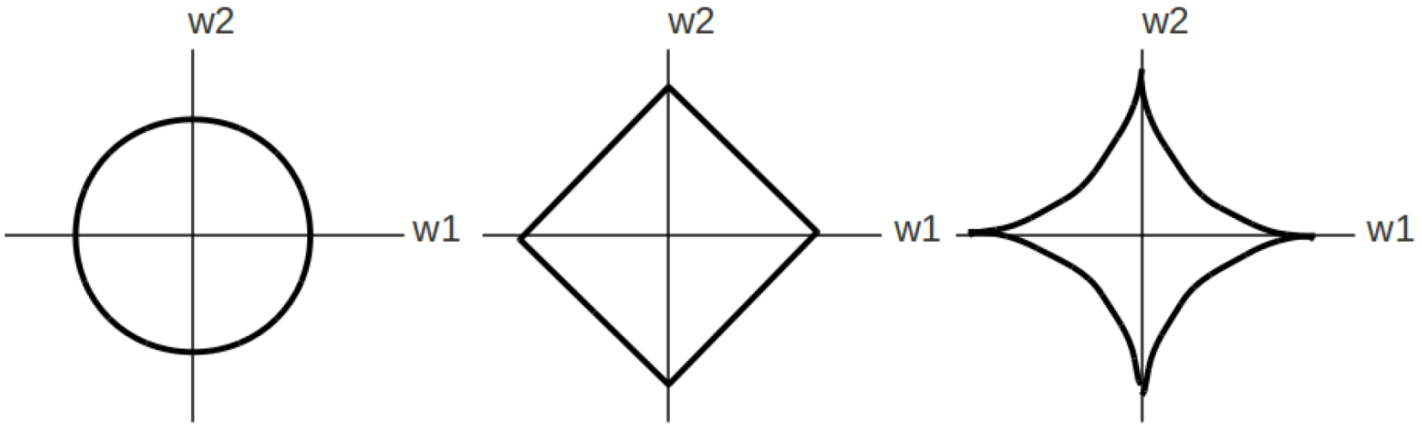
Figure credit: Piyush Rai

Norm-based Regularizers

- l_p norms can be used as regularizers

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

Contour plots for



p = 2 p = 1 p < 1

Figure credit: Piyush Rai

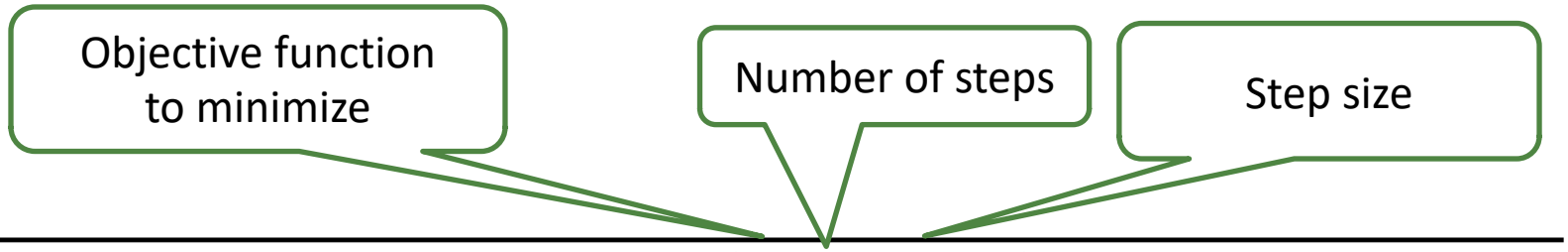
Gradient descent

- 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



Algorithm 22 GRADIENTDESCENT($\mathcal{F}$, K, η_1, ...)

1: $z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing
2: **for** $k = 1 \ldots K$ **do**
3: $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location
4: $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient
5: **end for**
6: **return** $z^{(K)}$

Illustrating gradient descent in 1-dimensional case

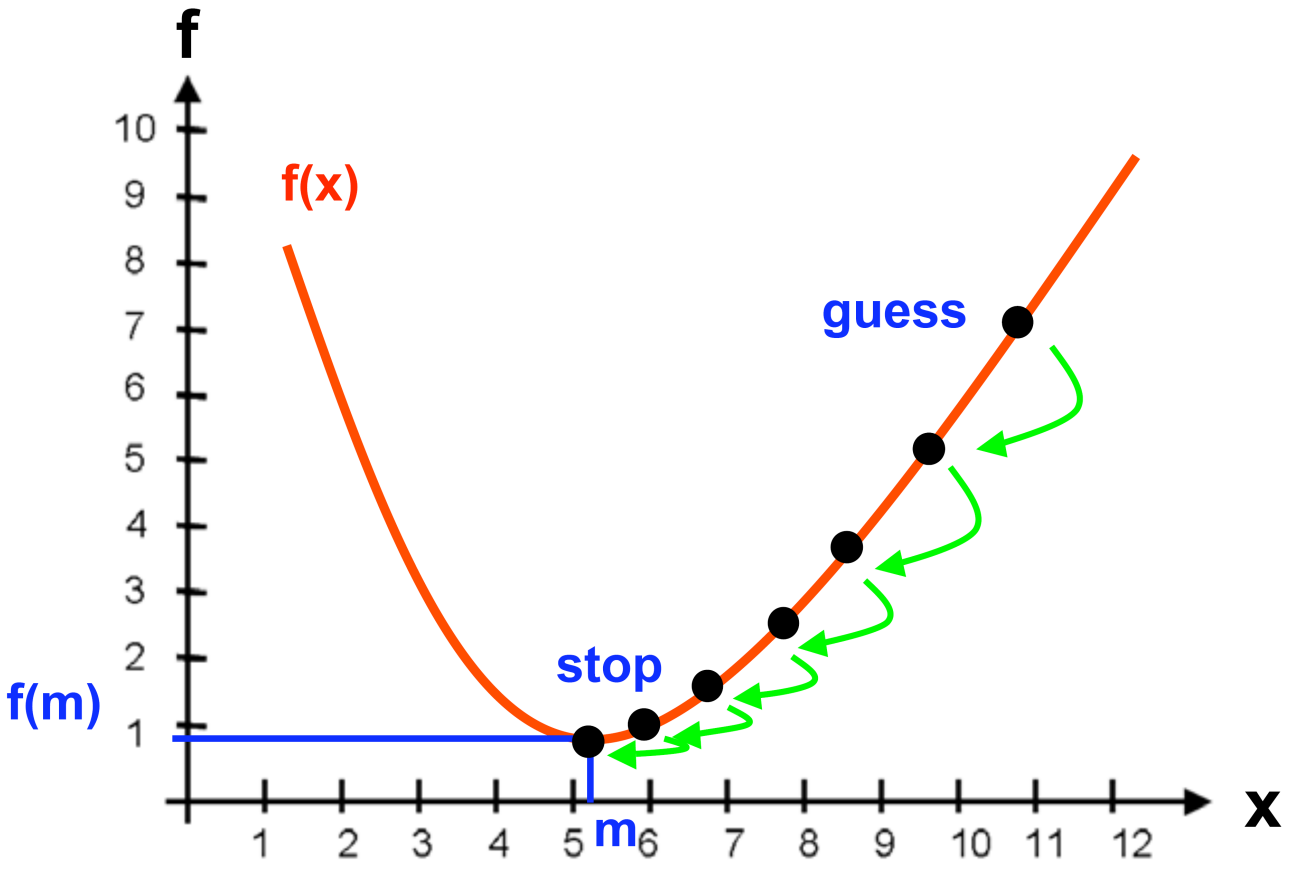


Figure credit: Piyush Rai

Impact of step size

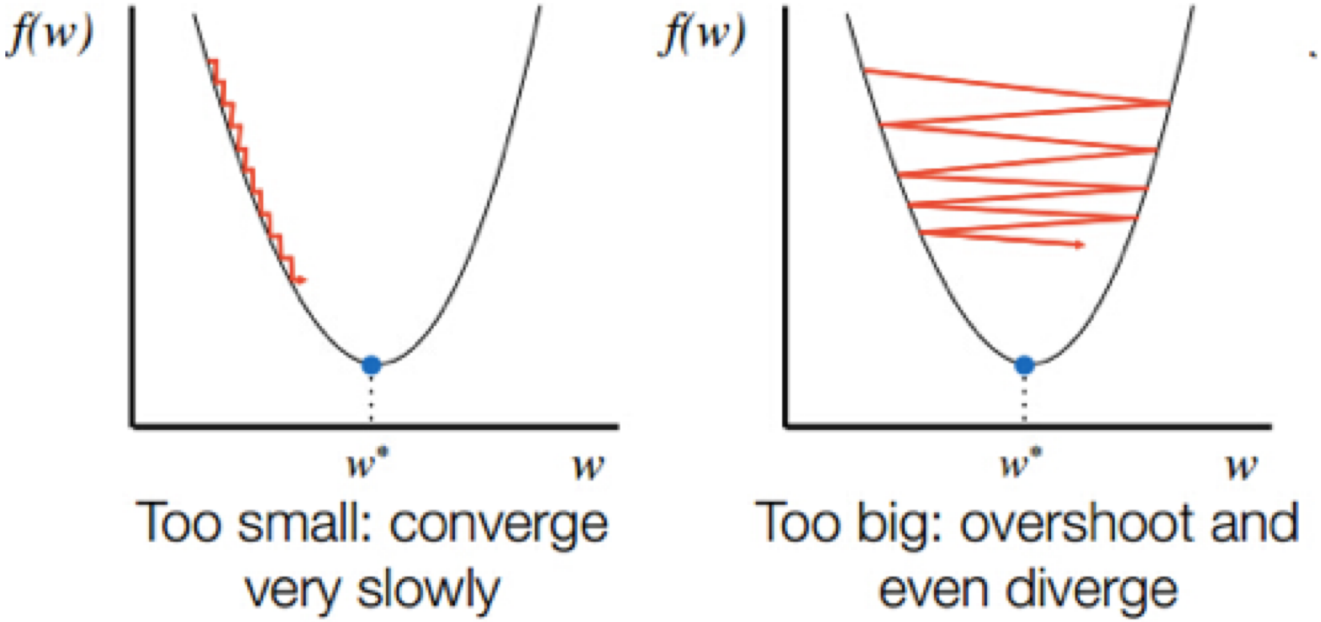


Image source: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

Illustrating gradient descent in 2-dimensional case

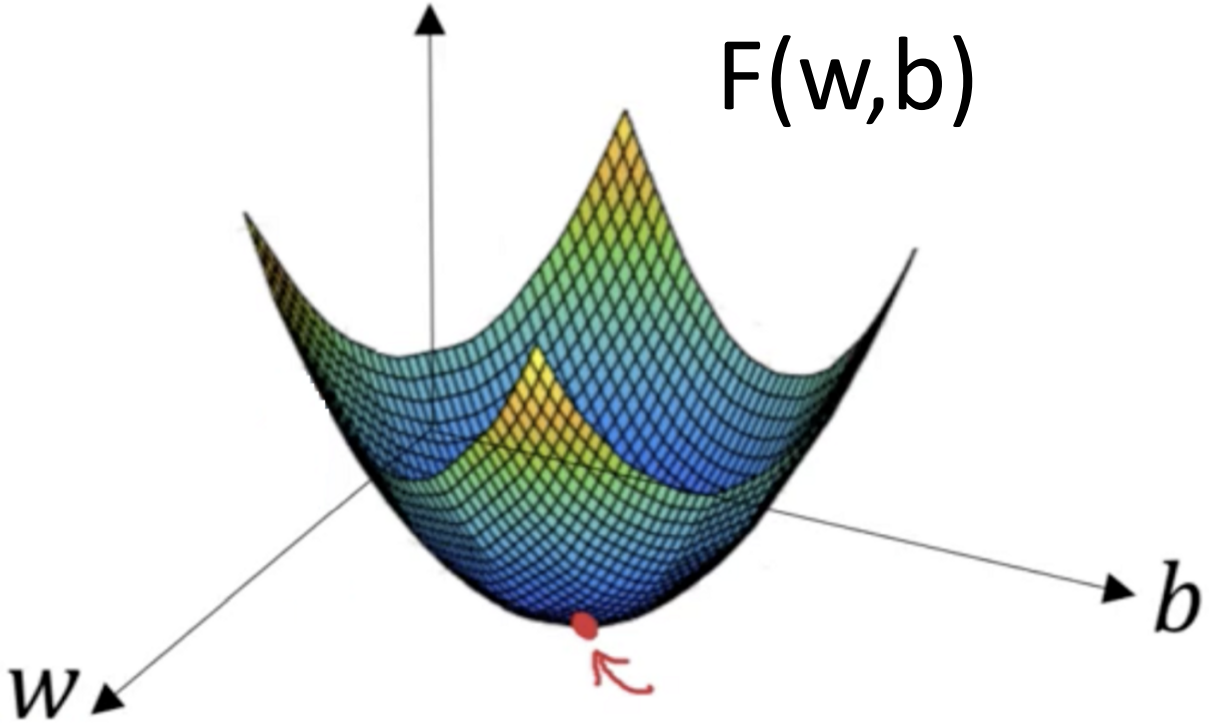


Image source: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

Illustrating gradient descent in 2-dimensional case

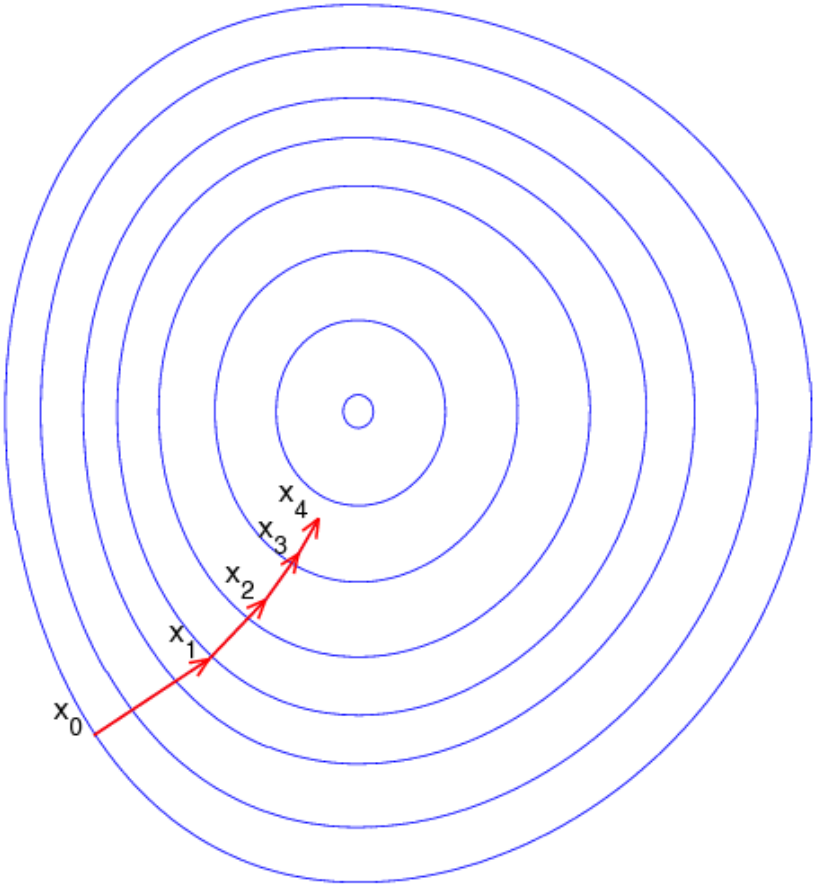
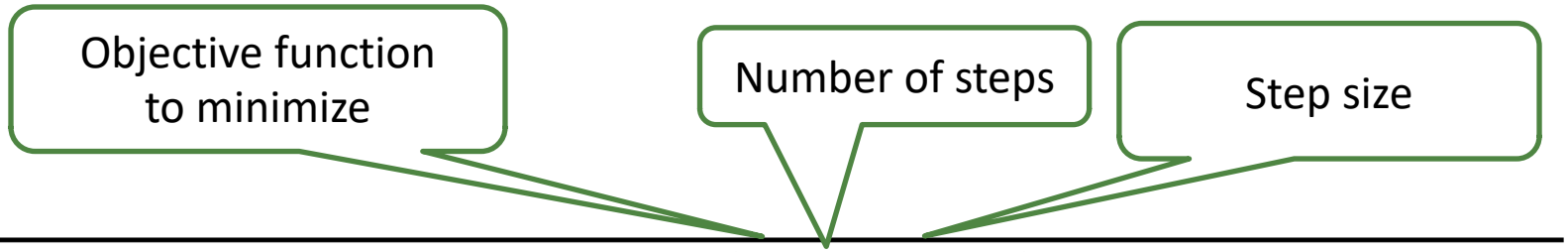


Image source: https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

Gradient descent algorithm



Algorithm 22 GRADIENTDESCENT($\mathcal{F}$, K, η_1, ...)

1: $z^{(0)} \leftarrow \langle 0, 0, \ldots, 0 \rangle$ // initialize variable we are optimizing
2: **for** $k = 1 \ldots K$ **do**
3: $g^{(k)} \leftarrow \nabla_z \mathcal{F}|_{z^{(k-1)}}$ // compute gradient at current location
4: $z^{(k)} \leftarrow z^{(k-1)} - \eta^{(k)} g^{(k)}$ // take a step down the gradient
5: **end for**
6: **return** $z^{(K)}$

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

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

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

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

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…

Example: subgradient of hinge loss

For a given example n

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

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

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

Subgradient Descent for Hinge Loss

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

1: $\boldsymbol{w} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $b \leftarrow 0$ // initialize weights and bias
2: **for** $iter = 1 \ldots MaxIter$ **do**
3: $\boldsymbol{g} \leftarrow \langle 0, 0, \ldots 0 \rangle$, $g \leftarrow 0$ // initialize gradient of weights and bias
4: **for all** $(\boldsymbol{x}, y) \in \mathbf{D}$ **do**
5: **if** $y(\boldsymbol{w} \cdot \boldsymbol{x} + b) \leq 1$ **then**
6: $\boldsymbol{g} \leftarrow \boldsymbol{g} + y\,\boldsymbol{x}$ // update weight gradient
7: $g \leftarrow g + y$ // update bias derivative
8: **end if**
9: **end for**
10: $\boldsymbol{g} \leftarrow \boldsymbol{g} - \lambda \boldsymbol{w}$ // add in regularization term
11: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{g}$ // update weights
12: $b \leftarrow b + \eta g$ // update bias
13: **end for**
14: **return** $\boldsymbol{w}$, b

What is the perceptron optimizing?

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

```
1:  w_d ← 0, for all d = 1 ... D                          // initialize weights
2:  b ← 0                                                  // initialize bias
3:  for iter = 1 ... MaxIter do
4:      for all (x,y) ∈ D do
5:          a ← Σ_{d=1}^{D} w_d x_d + b                    // compute activation for this example
6:          if ya ≤ 0 then
7:              w_d ← w_d + y x_d, for all d = 1 ... D     // update weights
8:              b ← b + y                                  // update bias
9:          end if
10:     end for
11: end for
12: return w_0, w_1, ..., w_D, b
```

- Loss function is a variant of the hinge loss

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

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
- Can be used to find parameters of linear models
- Optional: alternatives to gradient descent