Pattern Recognition
and Applications Lab

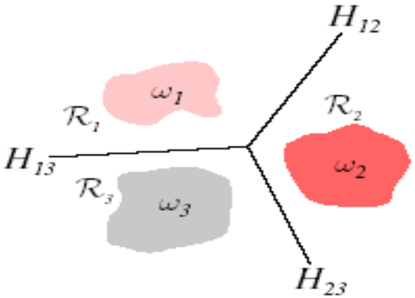
Elements of Linear Discriminant Functions

Battista Biggio

Department of Electrical and Electronic Engineering
University of Cagliari, Italy

Introduction

- We assume here that the form of the discriminant functions $f_k(\boldsymbol{x};\boldsymbol{\theta}),\ k=1,\ldots,K$ is given, and that we can use the training data to estimate their parameters $\boldsymbol{\theta}$
  - In Part 4, instead, we assumed that the underlying probability densities were known
- These methods are known as *nonparametric*
  - No assumption on the form of the underlying data probability distributions is made
- We focus here on functions that are *linear* in $\boldsymbol{x}$, i.e., $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{w}^T\boldsymbol{x} + b$, with $\boldsymbol{\theta} = (\boldsymbol{w}, b)$

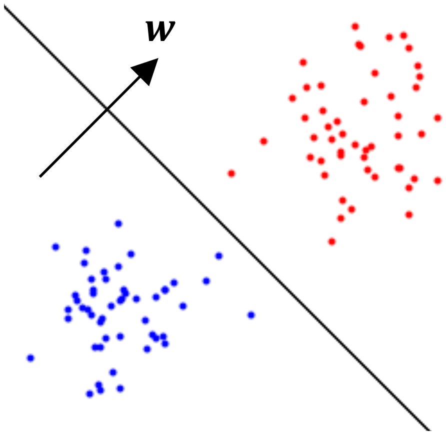


Example of linear discriminant functions on a 3-class classification problem

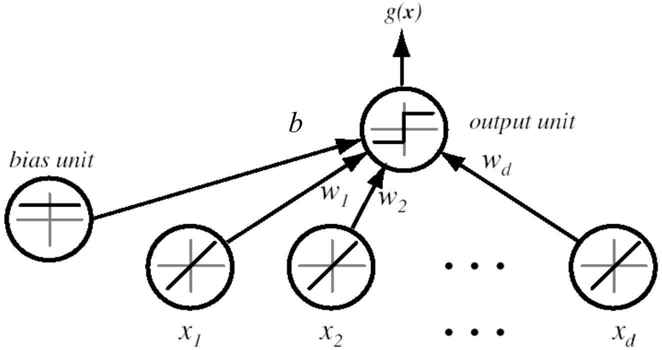
Linear Discriminant Functions

- **Linear function**: $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b = \sum_{j=1}^{d} w_j x_j + b$
  - $\boldsymbol{w}$ is the weight vector, and b the bias

- Two-class classification
  - Positive ($y = +1$) vs negative ($y = -1$) class
  - Decision rule: $y = \begin{cases} +1 & \text{if } f(\boldsymbol{x}) \geq 0 \\ -1, & \text{otherwise} \end{cases}$

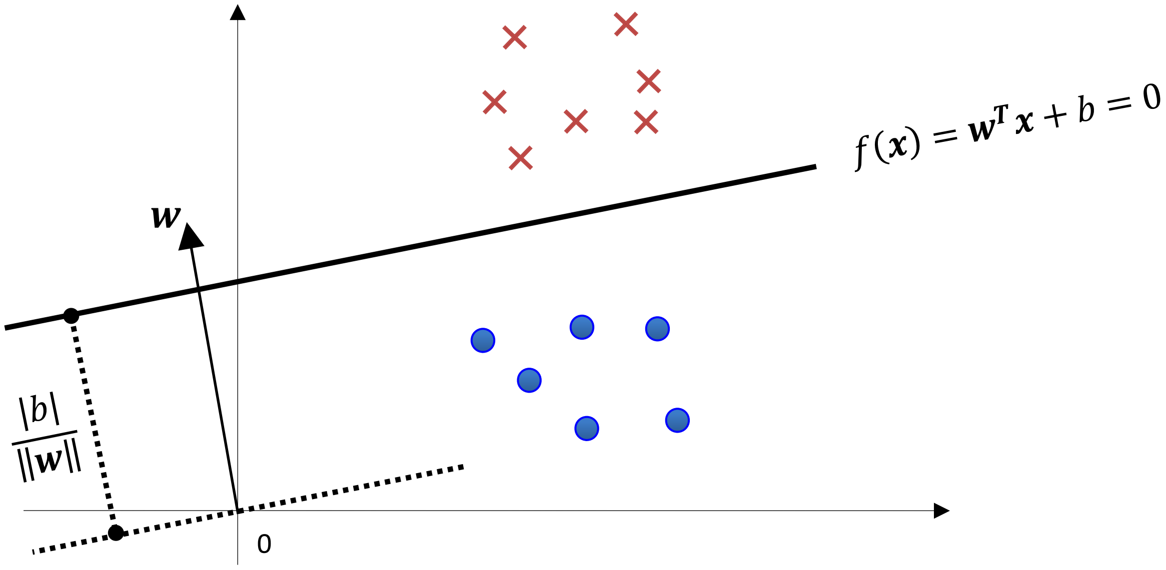


- **Graphical representation**
  - Each input feature value x_j is multiplied by the corresponding weight value w_j
  - Bias is multiplied by 1
  - The output unit sums all its inputs, computing $f(\boldsymbol{x})$ and thresholds it to estimate y (+1 or -1)



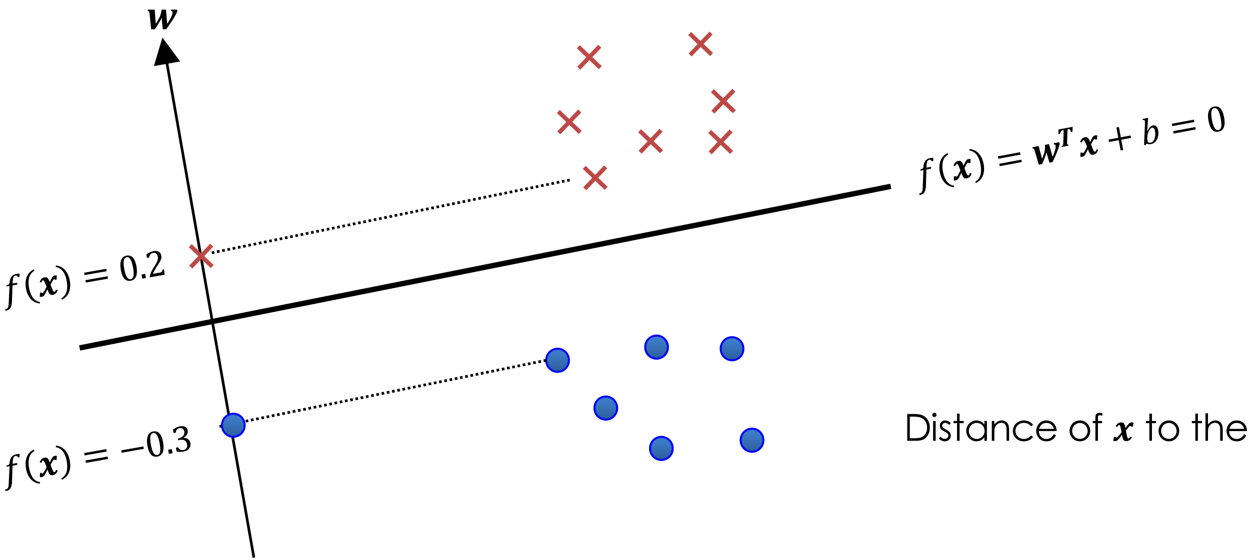
Linear Discriminant Functions

$\mathrm{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{n}$
$\boldsymbol{x} \in \mathbb{R}^d$
$y \in \{-1, +1\}$



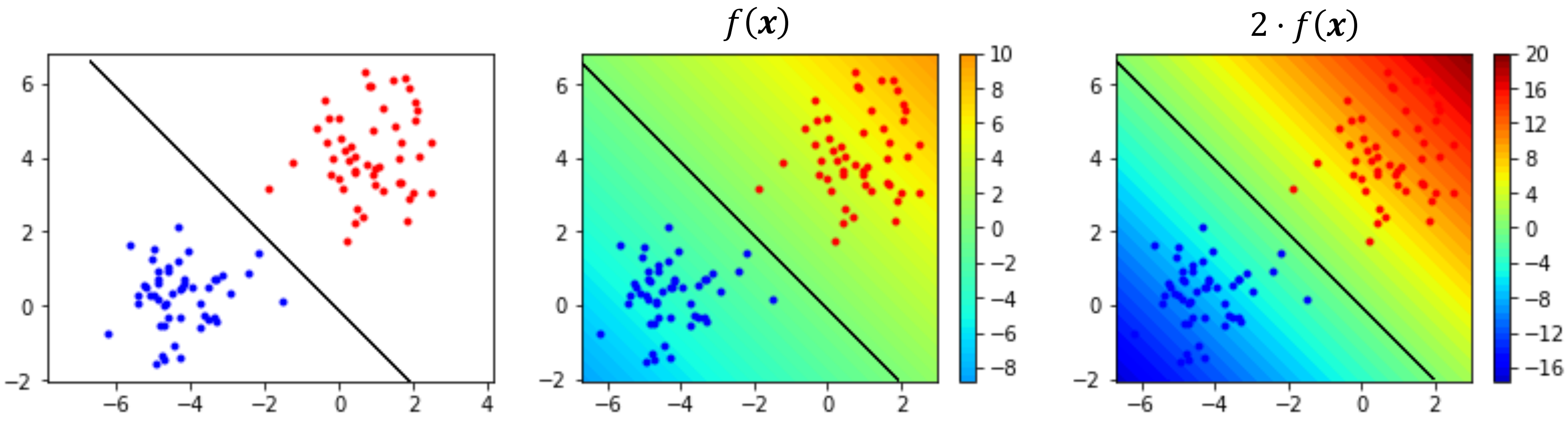
Linear Discriminant Functions

- The function $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$ projects $\boldsymbol{x}$ onto the hyperplane normal
  - Its value is proportional to the distance of $\boldsymbol{x}$ to the hyperplane



Distance of $\boldsymbol{x}$ to the hyperplane: $\dfrac{|f(\boldsymbol{x})|}{\|\boldsymbol{w}\|}$

Linear Discriminant Functions



- By linearity, it is easy to see that multiplying $f(\boldsymbol{x})$ by a constant factor amounts to multiplying its parameters by the same factor
  - For example: $2 \cdot f(\boldsymbol{x}) = 2\, \boldsymbol{w}^T \boldsymbol{x} + 2b = \widehat{\boldsymbol{w}}^T \boldsymbol{x} + \hat{b}$, with $\widehat{\boldsymbol{w}} = 2\boldsymbol{w}$ and $\hat{b} = 2b$

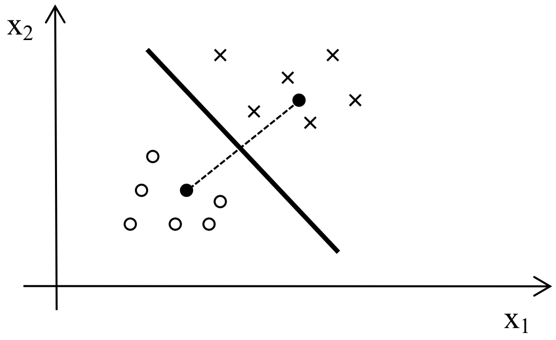
- While the boundary at $f(\boldsymbol{x}) = 0$ does not change, the slope of the function changes!

A Simple Example: The Nearest Mean Classifier (NMC)

- This classifier estimates the mean values $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ of the two classes from the training set and assigns unknown samples $\boldsymbol{x}^\star$ to the class with the smallest Euclidean distance:

$$d(\mathbf{x}^*, \boldsymbol{\mu}_2) \underset{\omega_2}{\overset{\omega_1}{\lesseqgtr}} d(\mathbf{x}^*, \boldsymbol{\mu}_1)$$

- It is easy to see that the decision boundary is the hyperplane perpendicular to the vector $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$ and passing through the mean point $(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2)/2$



- Accordingly, the NMC is a linear classifier
  - Try to find its $\boldsymbol{w}$ and b parameter values!

Learning as an Optimization Problem

Learning as an Optimization Problem

- How do we estimate the classifier parameters $\boldsymbol{w}$ and b?

- Modern approaches formulate the learning problem as an **optimization problem**
  – This is generally true also for nonlinear classification functions $f(\boldsymbol{x}; \boldsymbol{\theta})$, including modern deep-learning approaches and neural networks

$$\boldsymbol{w}^\star, b^\star = \underset{\boldsymbol{w}, b}{\operatorname{argmin}} \underbrace{\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i))}_{\substack{\text{loss term} \\ L(D, \boldsymbol{\theta})}} + \lambda \underbrace{\Omega(\boldsymbol{w})}_{\substack{\text{regularization term} \\ \Omega(\boldsymbol{w})}}$$

λ: regularization hyperparameter

Learning as an Optimization Problem

- The loss function $\ell(y_i, f(\boldsymbol{x}_i))$ measures how much a prediction is wrong
  - e.g., the zero-one loss is 0 if points are correctly predicted, and 1 if they are not

- The regularization term $\Omega(\boldsymbol{\theta})$ imposes a penalty on the magnitude of the classifier parameters to avoid overfitting and promote smoother functions, i.e., functions that change more gradually as we move across the feature space

- The hyperparameter λ tunes the trade-off between the training loss and regularization
  - Larger values tend to promote more regularized functions but with a larger training error
  - Smaller values tend to reduce the training error but learn more complex functions

Learning as an Optimization Problem

- We start by considering a simplified setting in which we aim to find the best parameters $\boldsymbol{\theta} = (\boldsymbol{w}, b)$ that minimize the loss function $L(D, \boldsymbol{\theta})$, being $D = (\boldsymbol{x}_i, y_i)_{i=1}^{n}$ the training dataset:

$$\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} L(D, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$$

- The loss function quantifies the error that the classifier, parameterized by $\boldsymbol{\theta}$, is making on its predictions on the training data D
  - This is also known as the principle of **Empirical Risk Minimization (ERM)**

- How do we select the loss function $L(D, \boldsymbol{\theta})$ and solve the above problem?

Learning as an Optimization Problem

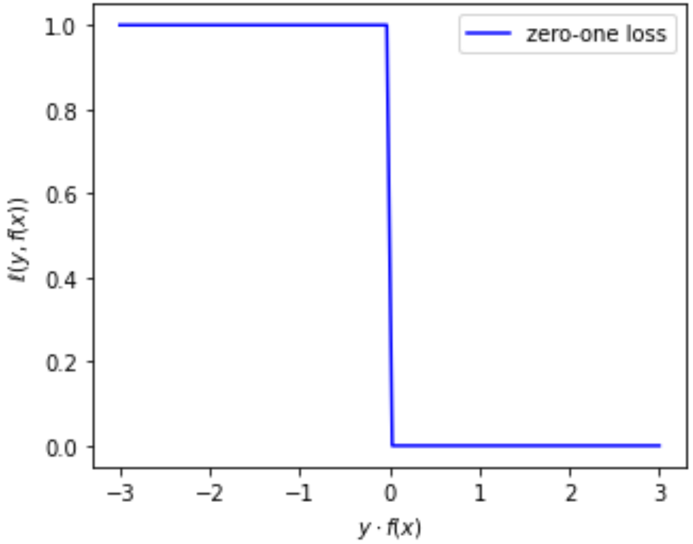
- In principle, we would like to minimize

$$L(D, \boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$$

- where $\ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$ is the zero-one loss
  - equal to 0 for correct predictions and 1 otherwise

$$\ell\big(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta})\big) = \begin{cases} 1, & \text{if } \; y \cdot f(\boldsymbol{x}) < 0 \\ 0, & \text{if } \; y \cdot f(\boldsymbol{x}) \geq 0 \end{cases}$$

- However, minimizing this *step function* is hard and computationally inefficient

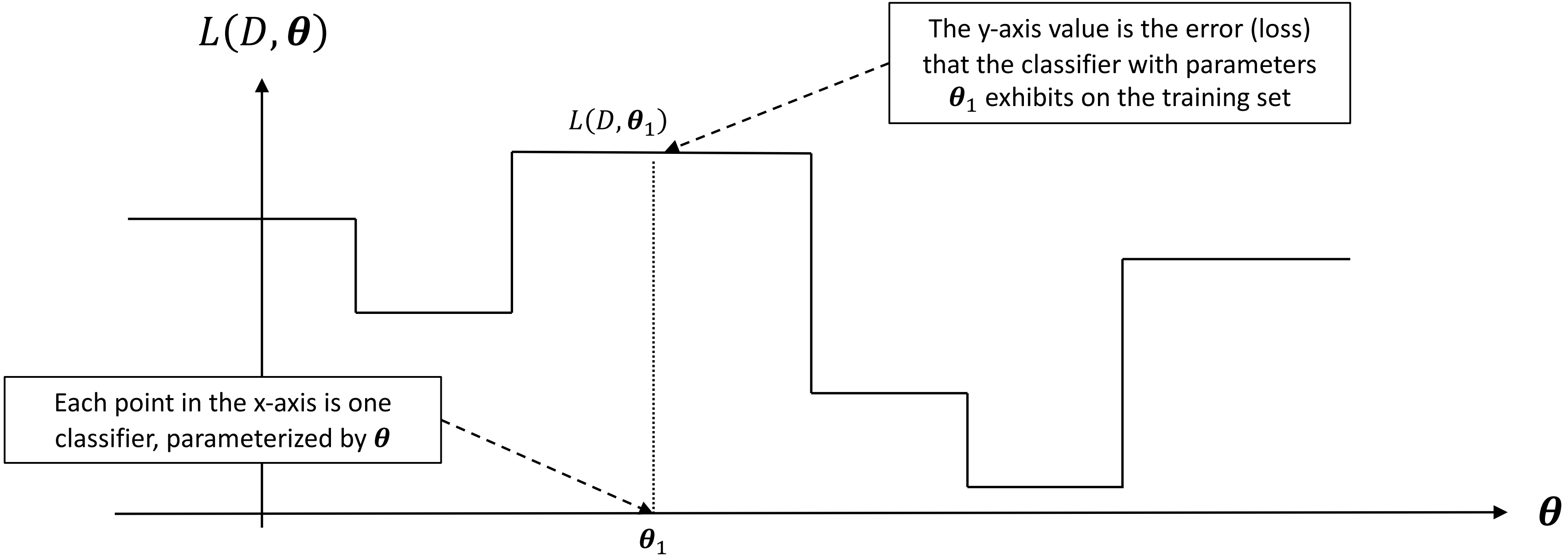


Note that $yf(\boldsymbol{x})$ is positive for correct predictions and negative for wrong ones.

For correct (wrong) predictions, y and $f(\boldsymbol{x})$ agree (disagree) in sign.

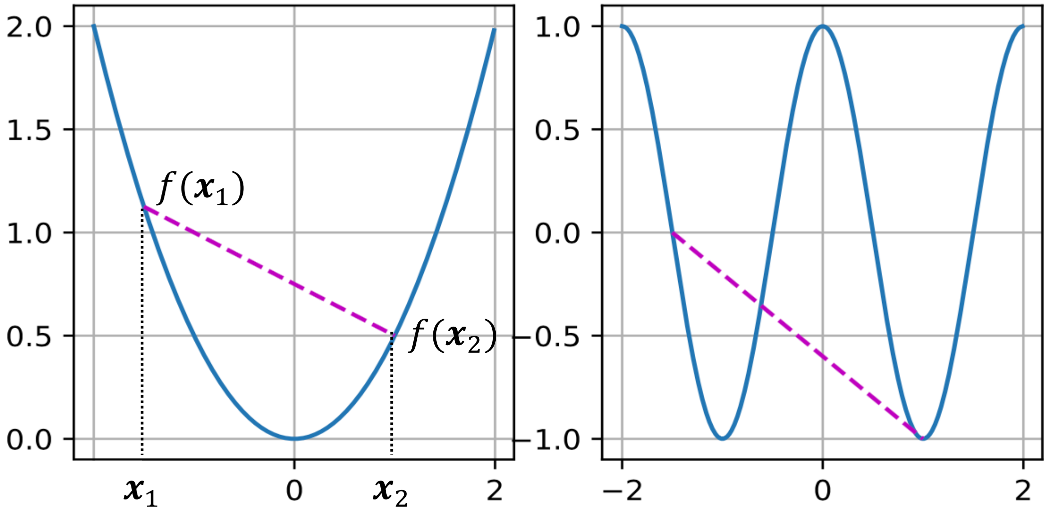
The Zero-One Loss Landscape

- Non convex, hard to optimize (flat regions, bad local minima)



Why Is Convexity Important for Optimization?

- **Convexity**: $f(\lambda \boldsymbol{x}_1 + (1-\lambda)\boldsymbol{x}_2) \leq \lambda f(\boldsymbol{x}_1) + (1-\lambda) f(\boldsymbol{x}_2)$



- **Desirable properties for optimization:** no local minima, convergence guarantees, etc.

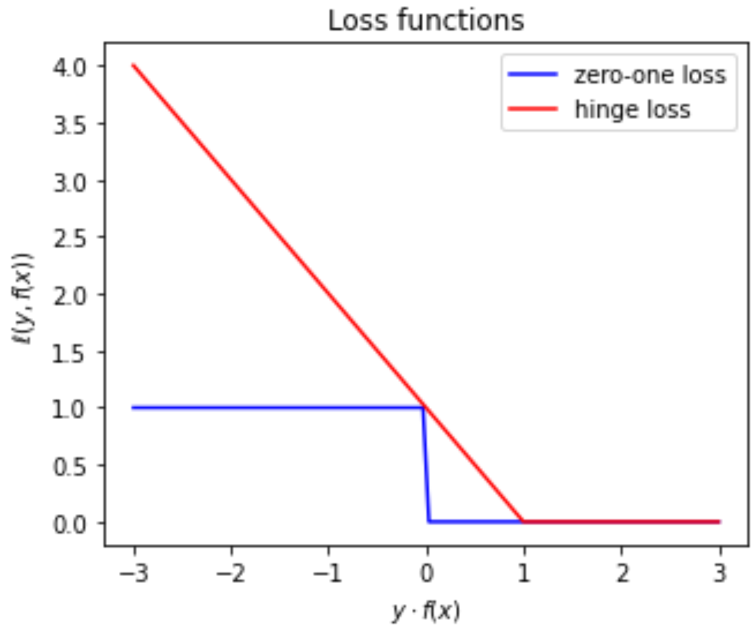
https://d2l.ai/chapter_optimization/convexity.html

Loss Functions

- Now, recall that we aim to minimize:

$$L(D, \boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$$

- being $\ell(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$ the zero-one loss
- However, we know that minimizing this non-convex function is particularly difficult
- For this reason, convex (surrogate) loss functions are typically preferred
  - The tighter convex upper bound on the zero-one loss is called the **hinge loss**



Hinge Loss

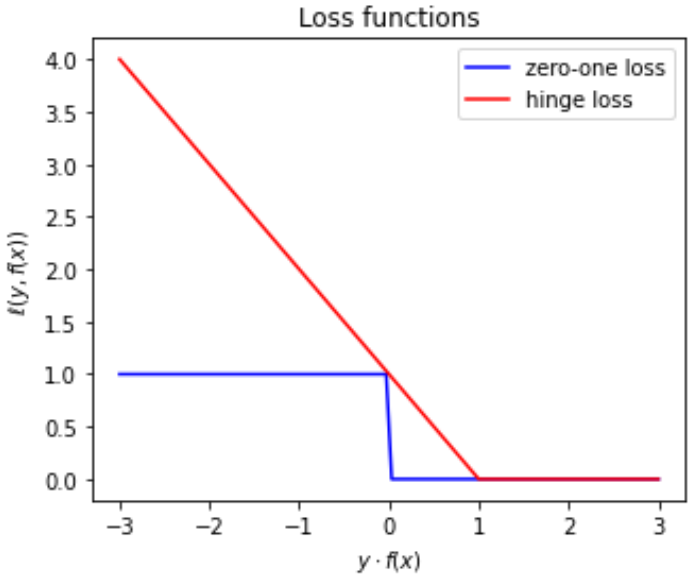
- The hinge loss is computed as:

$$\ell\big(y, f(\boldsymbol{x}; \theta)\big) = \max(0, 1 - yf)$$

- It is the closest convex upper bound on the 0-1 loss

- Why are we interested in an upper bound?
  – since minimizing it, we also minimize the 0-1 loss

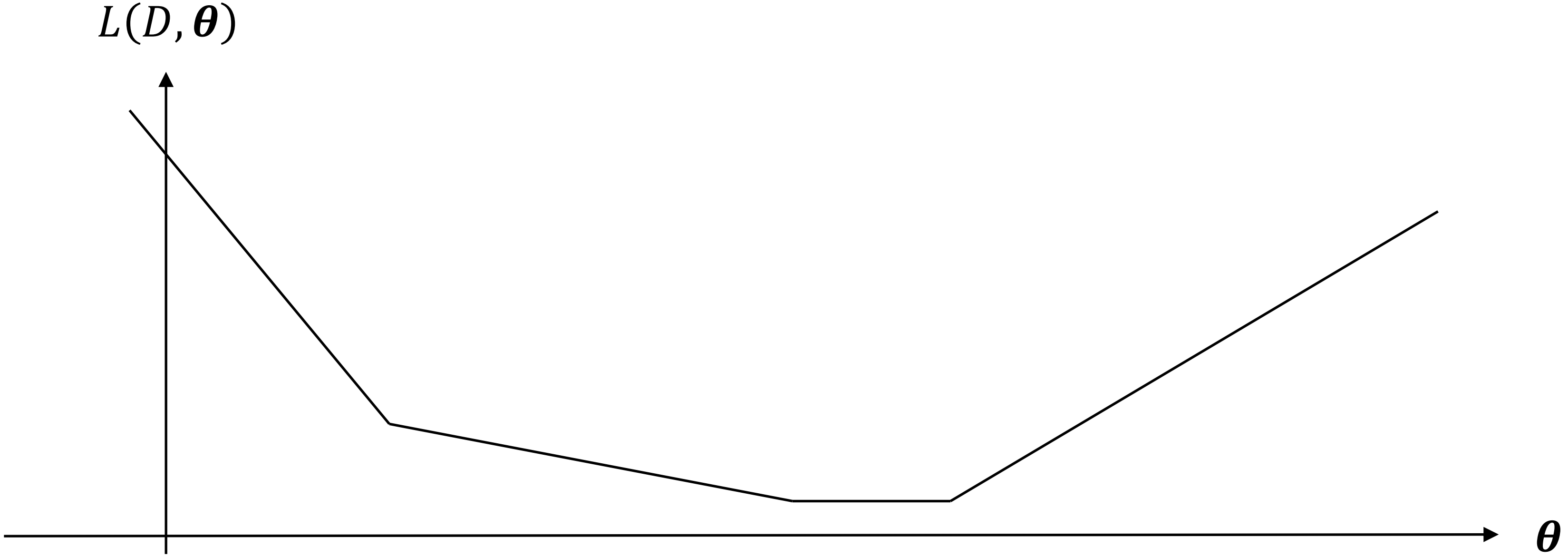
- Convexity helps finding solutions efficiently while also providing guarantees on the uniqueness of the solution, algorithmic convergence, etc.
  – This is why convex surrogate functions are typically preferred in optimization



Note that $yf(\boldsymbol{x})$ is positive for correct predictions and negative for wrong ones

The Hinge Loss Landscape

- Piecewise linear and convex, easier to optimize



A Closer Look at the Loss Minimization Problem

- Let's assume we fix $b = 0$ and aim to minimize the training loss only w.r.t. w_1, w_2
- Each pair w_1, w_2 thus represents a different linear classifier (passing through the origin)
- For each of these classifiers, we report the corresponding training loss in a colored plot
  - this will show us the optimization *landscape*, i.e., the surface of the function we aim to minimize