

# Introduction to Machine Learning

## Linear Classification

Prof. Andreas Krause  
Learning and Adaptive Systems ([las.ethz.ch](http://las.ethz.ch))

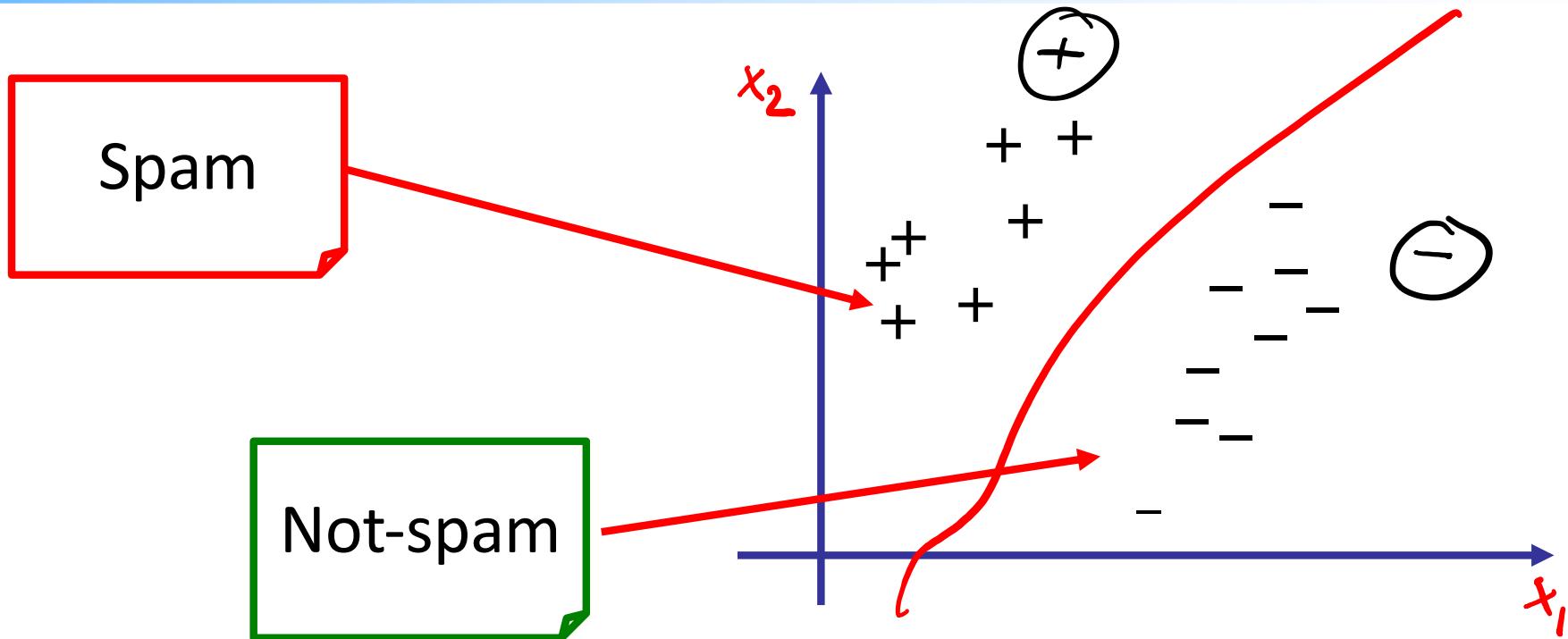
# Classification

---

- Instance of supervised learning where  $Y$  is discrete (categorical)
  - Want to assign data points  $X$ 
    - Documents
    - Queries
    - Images
    - User visits
    - ...
- a label  $Y$  (spam/not-spam; topic such as sports, politics, entertainment, click/no-click etc.)

For now, focus on binary classification

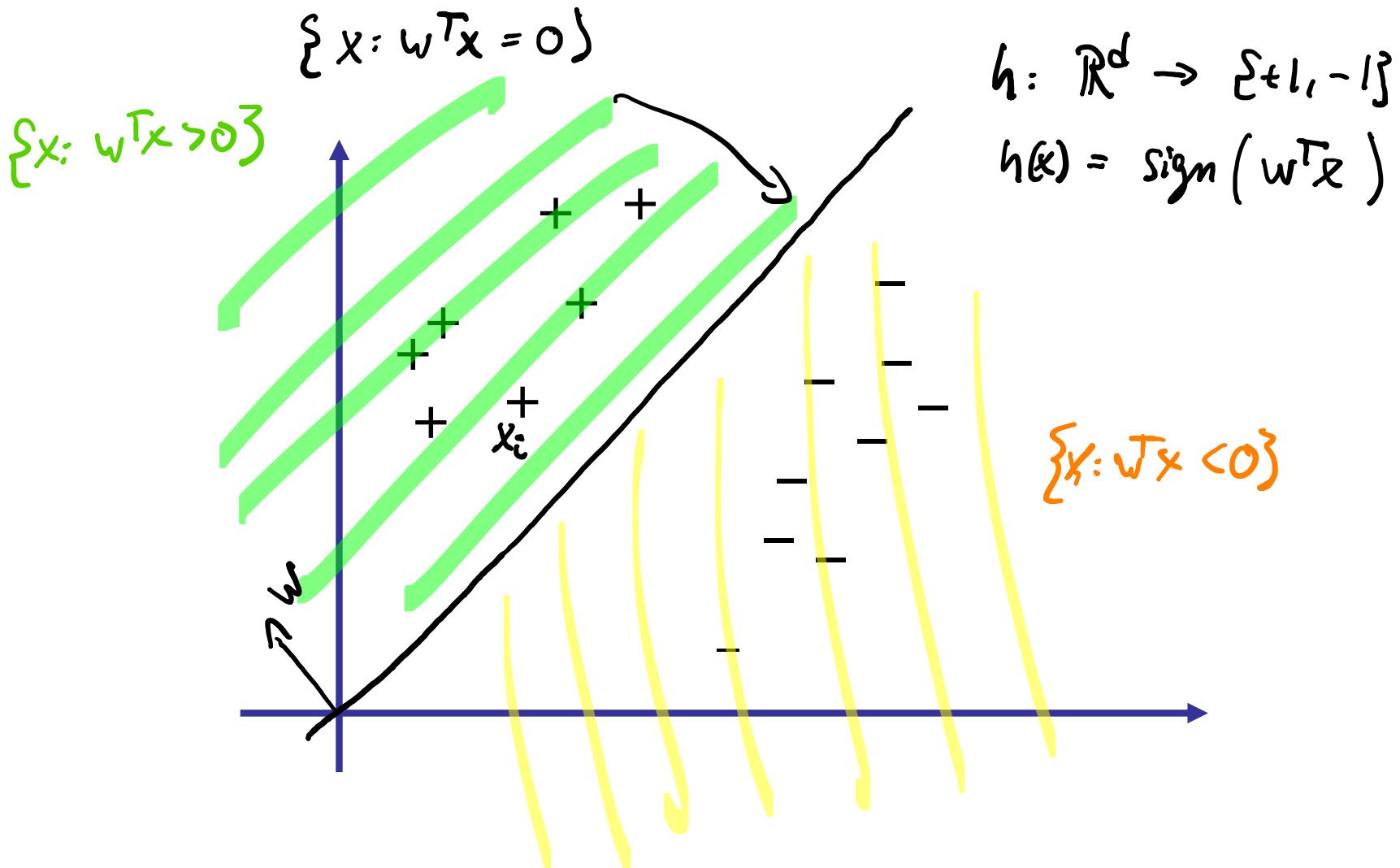
# Illustration of binary classification



- **Input:** Labeled data set (e.g., rep. bag-of-words) with positive (+) and negative (-) examples
- **Output:** Decision rule (hypothesis)

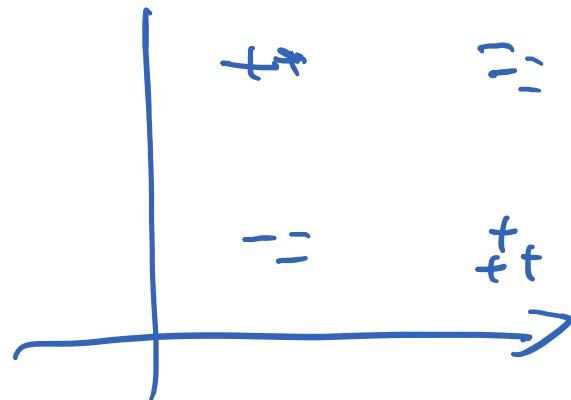
# Linear classifiers

- Data set  $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$



# Why linear classification?

- Linear classification seems restrictive:



- In high-dimensional settings / when using the right features, often works quite well!
- Prediction is typically very efficient

# Finding linear separators

- Want to write the search for a classifier as optimization problem
- What should we optimize?
- First Idea:** Seek  $w$  that minimizes #mistakes

$$\hat{w} = \underset{w \in \mathbb{R}^d}{\operatorname{arg\!min}} \sum_{i=1}^n [y_i \neq \operatorname{Sign}(w^T x_i)]$$

$\sum_{i=1}^n [y_i \neq \operatorname{Sign}(w^T x_i)]$

$\approx \begin{cases} 1 & \text{if } y_i \neq \operatorname{Sign}(w^T x_i) \\ 0 & \text{otherwise} \end{cases}$

$l_{0/1}(w; x_i, y_i)$

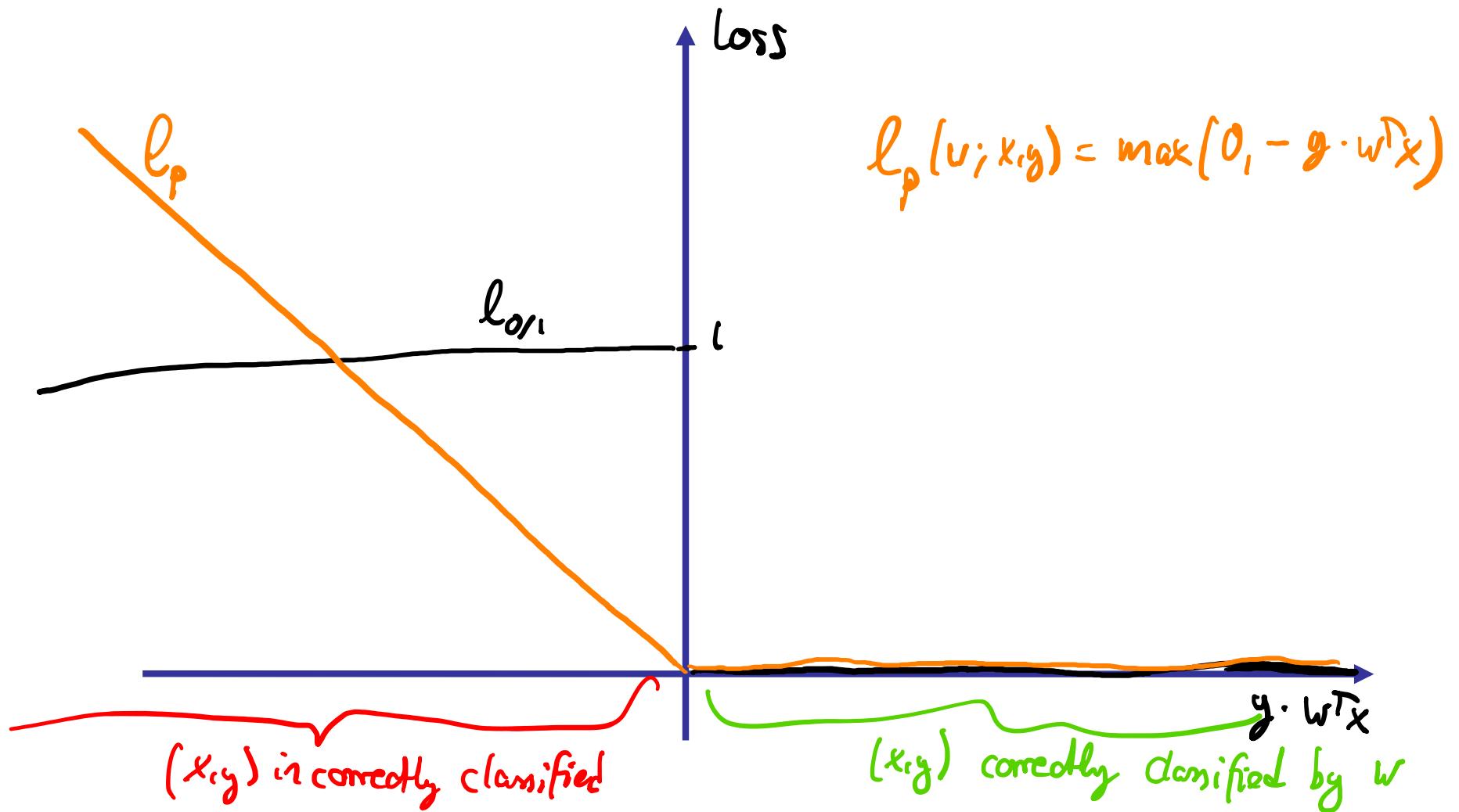
# Optimization problem

- **Goal:**

$$\begin{aligned}\hat{\mathbf{w}} &= \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n [y_i \neq \text{sign}(\mathbf{w}^T \mathbf{x}_i)] \\ &= \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n \ell_{0/1}(\mathbf{w}; \mathbf{x}_i, y_i)\end{aligned}$$

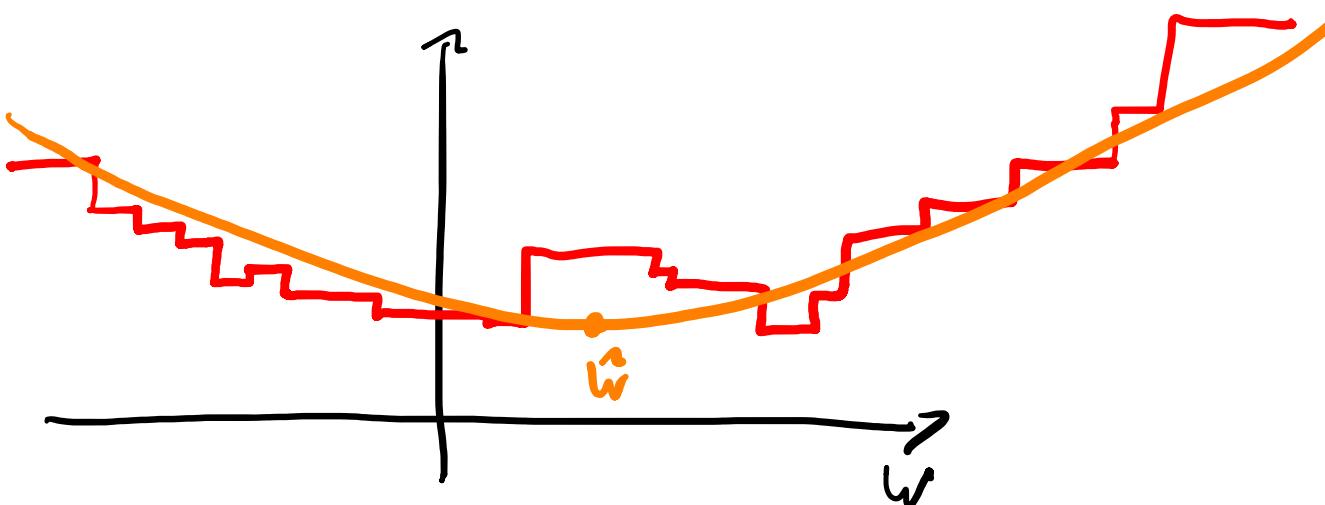
- **Challenge:** in contrast to squared loss, 0/1 loss is not convex (not even differentiable!)

# A surrogate loss function



# Key concept: Surrogate losses

- Replace intractable cost function that we care about (e.g., 0/1-loss) by tractable loss function (e.g., Perceptron loss) for sake of **optimization / model fitting**
- When evaluating a model (e.g., via cross-validation), use **original cost / performance function**



# Surrogate optimization problem

- Instead of

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n \ell_{0/1}(\mathbf{w}; \mathbf{x}_i, y_i)$$

- Solve

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n \ell_P(\mathbf{w}; \mathbf{x}_i, y_i)$$

where  $\ell_P(\mathbf{w}; y_i, \mathbf{x}_i) = \max(0, -y_i \mathbf{w}^T \mathbf{x}_i)$

# Gradient descent

- Compute gradient of the Perceptron objective function

$$R(w) = \frac{1}{n} \sum_{i=1}^n \max (0, -w^T x_i \cdot g_i)$$

$$\nabla_w \hat{R}(w) = \frac{1}{n} \sum_{i=1}^n \underbrace{\nabla_w \ell_p(w; x_i, y_i)}_{(*)}$$

$$= -\frac{1}{n} \sum_{i: y_i w^T x_i \leq 0} y_i x_i$$

$$\Rightarrow w_{t+1} \leftarrow w_t + \eta_t \sum_{i: (x_i, y_i) \text{ incorrect}} y_i x_i$$

# *Stochastic gradient decent*

---

- Computing the gradient requires summing over all data
- For large data sets, this is inefficient
- Moreover, our initial estimates are likely very wrong, and we can get a good (unbiased) gradient estimate by evaluating the gradient on few points
- Extreme case: Evaluate on **only one** randomly chosen point!

→ Stochastic gradient descent

# Stochastic Gradient Descent

- Start at an arbitrary  $\mathbf{w}_0 \in \mathbb{R}^d$
- For  $t=1,2,\dots$  do
  - Pick data point  $(\mathbf{x}', y') \in D$  from training set uniformly at random (with replacement), and set

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla \ell(\mathbf{w}_t; \mathbf{x}', y')$$

- Hereby,  $\eta_t$  is called learning rate
- Guaranteed to converge under mild conditions, if

$$\sum_t \eta_t = \infty \quad \text{and} \quad \sum_t \eta_t^2 < \infty$$

# The Perceptron algorithm

---

- Is just stochastic gradient descent (SGD) on the Perceptron loss function  $\ell_P$  with learning rate 1
- **Theorem:** If the data is linearly separable, the Perceptron will obtain a linear separator

# Perceptron Demo

---

# Mini-batch SGD

---

- Using single points might have large variance in the gradient estimate, and hence lead to slow convergence.
- Can reduce variance by averaging over the gradients w.r.t. multiple randomly selected points (**mini-batches**)

# Demo: SGD for regression

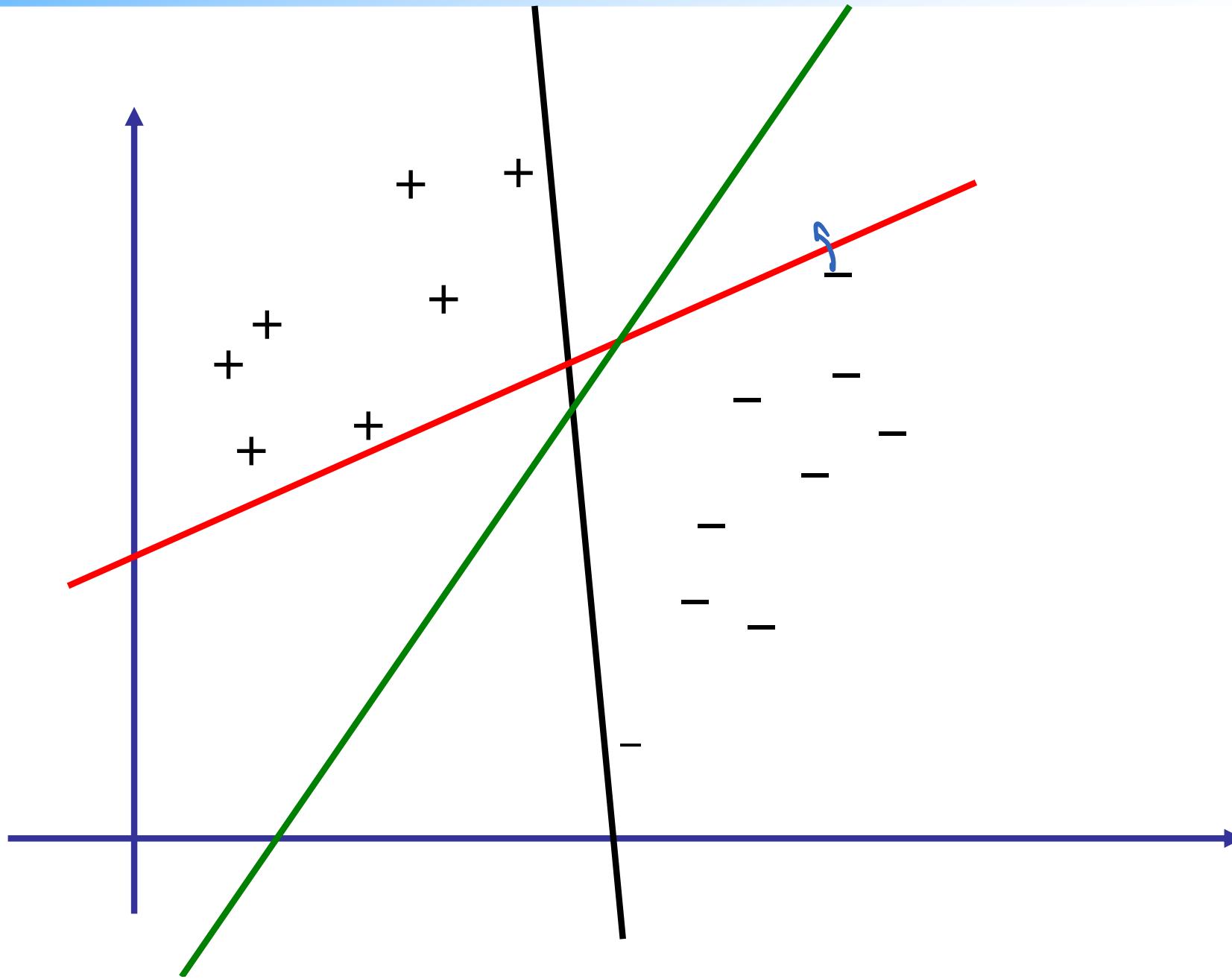
---

# Adaptive learning rates

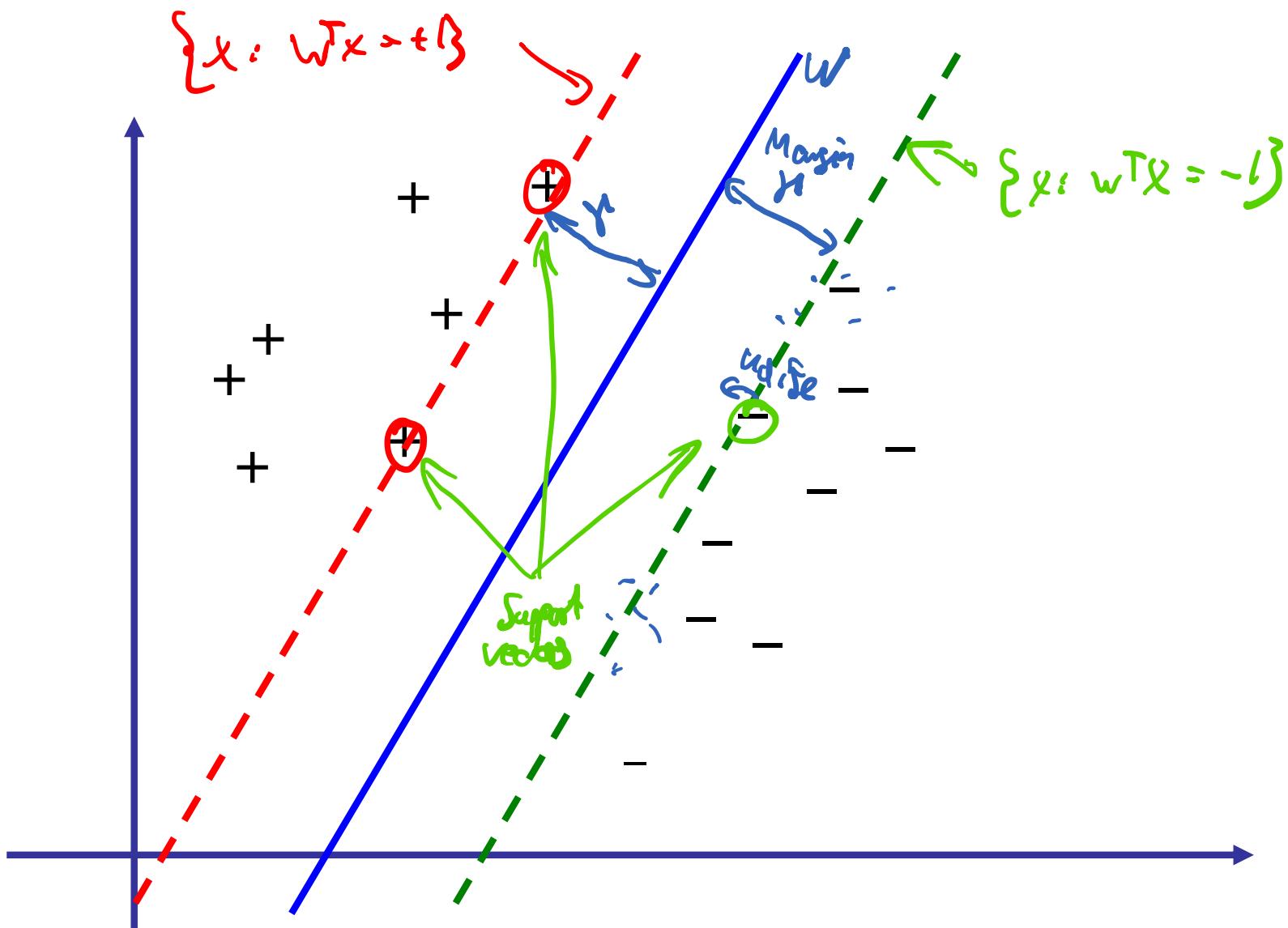
---

- Similar as in gradient descent, the learning rate is very important
- There exist various approaches for adaptively tuning the learning rate. Often times, these even use a different learning rate per feature
- Examples: AdaGrad, RMSProp, Adam, ...

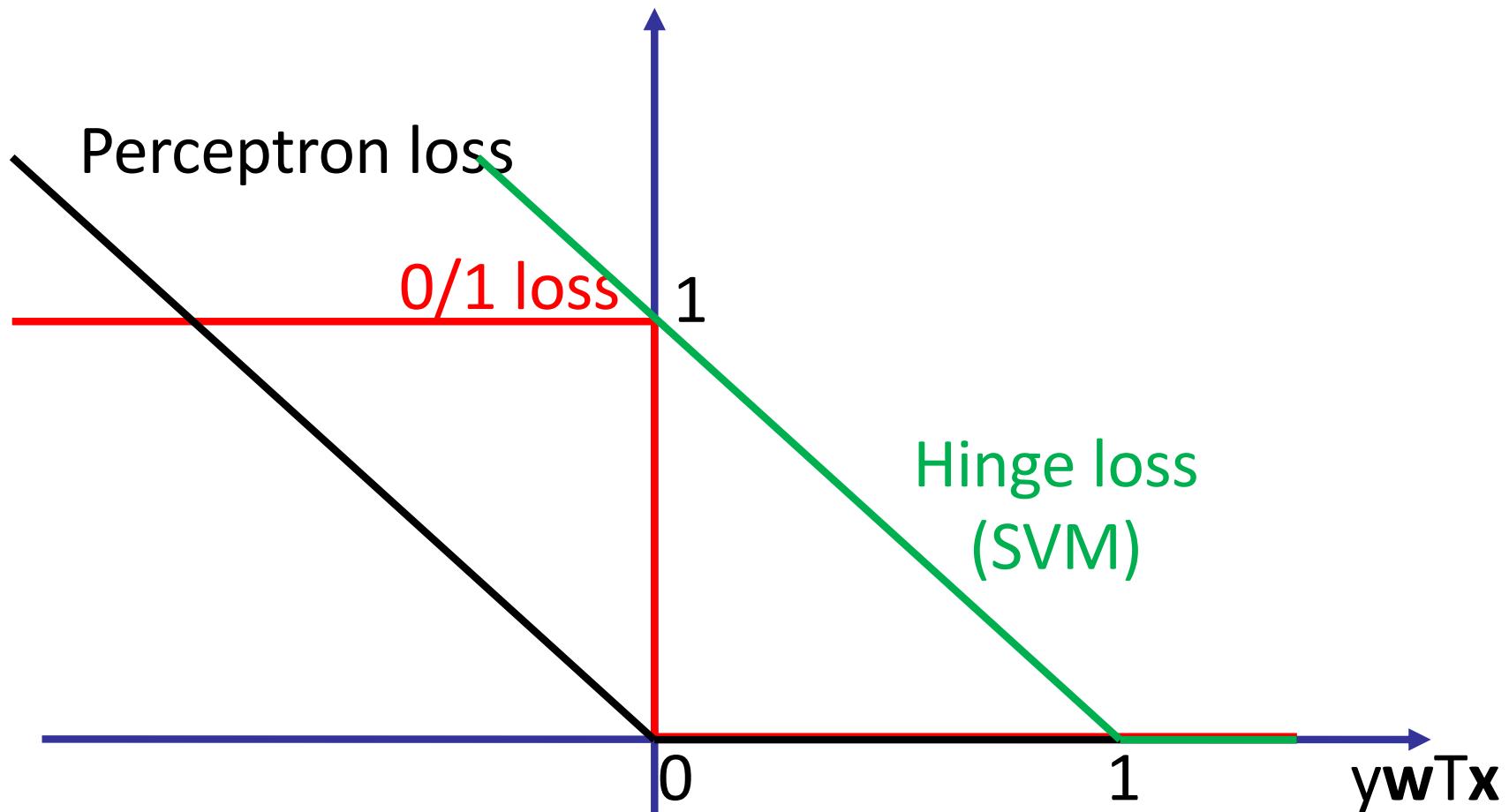
Which of these separators will the Perceptron “prefer”?



# Support Vector Machines (SVMs): “max. margin” linear classification



# Hinge vs. Perceptron loss



Hinge loss upper bounds #mistakes; encourages „margin“

$$\ell_H(\mathbf{w}; \mathbf{x}, y) = \max\{0, 1 - y\mathbf{w}^T \mathbf{x}\}$$

# SVM vs. Perceptron

---

- Perceptron:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n \max\{0, -y_i \mathbf{w}^T \mathbf{x}_i\}$$

- Support vector machine (SVM):

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n \max\{0, 1 - y_i \mathbf{w}^T \mathbf{x}_i\} + \lambda \|\mathbf{w}\|_2^2$$

# Support vector machines

---

- Widely used, very effective linear classifier
- Almost like Perceptron. Only differences:
  - Optimize slightly different, shifted loss (hinge loss)
  - Regularize weights (like ridge regression)
- Can optimize via stochastic gradient descent
- More details in Advanced Machine Learning lecture

# SGD for SVM

$$\nabla_w \frac{1}{n} \sum_{i=1}^n \left( \max(1 - w^T x_i \cdot y_i, 0) + \lambda \|w\|_2^2 \right)$$

$\ell_H(w; x_i, y_i)$

$$\nabla \ell_H(w; x_i, y_i) = \begin{cases} 0 & \text{if } y_i w^T x_i \geq 1 \\ -y_i x_i & \text{otherwise.} \end{cases}$$

$$\nabla_w \|w\|_2^2 = \nabla_w w^T w = 2w$$

$$w_{t+1} \leftarrow w_t (1 - 2\gamma_t \lambda) + y_i x_i \gamma_t [y_i w_t^T x_i < 1]$$

$\uparrow$   
 $i$  is the index of random point

# Demo: Monitoring SGD

---