CS446 Introduction to Machine Learning (Spring 2015)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURE 4: LINEAR CLASSIFIERS

Prof. Julia Hockenmaier
juliahmr@illinois.edu

Announcements

Homework 1 will be out after class.

http://courses.engr.illinois.edu/cs446/Homework/HW1.pdf

You have two weeks to complete the assignment.

Late policy:

- Up to two days late credit for the whole semester, but we don't give any partial late credit. If you're late for one assignment by up to 24 hours, that's 1 of your two late credit days.
- We don't accept assignments that are more than 48 hours late.

Is everybody on Compass???

https://compass2g.illinois.edu/

Let us know if you can't see our class.

Last lecture's key concepts

Decision trees for (binary) classification

- Non-linear classifiers

Learning decision trees (ID3 algorithm)

- Greedy heuristic (based on information gain)
- Originally developed for discrete features

Overfitting

- What is it? How do we deal with it?

Today's key concepts

Learning linear classifiers

Batch algorithms:

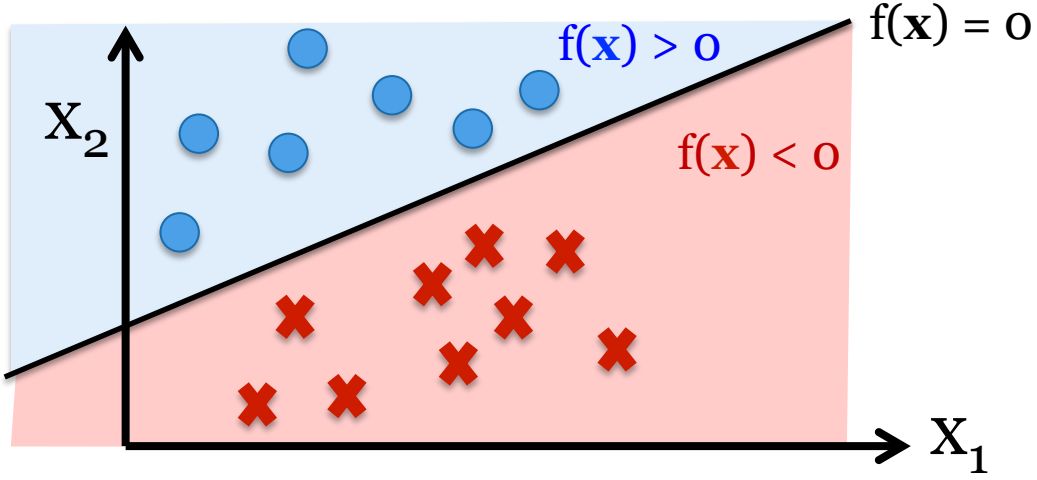
– Gradient descent for Least-mean squares

Online algorithms:

– Stochastic gradient descent

Linear classifiers

Linear classifiers: $f(\mathbf{x}) = w_0 + \mathbf{wx}$



Linear classifiers are defined over vector spaces

Every hypothesis f(**x**) is a hyperplane:

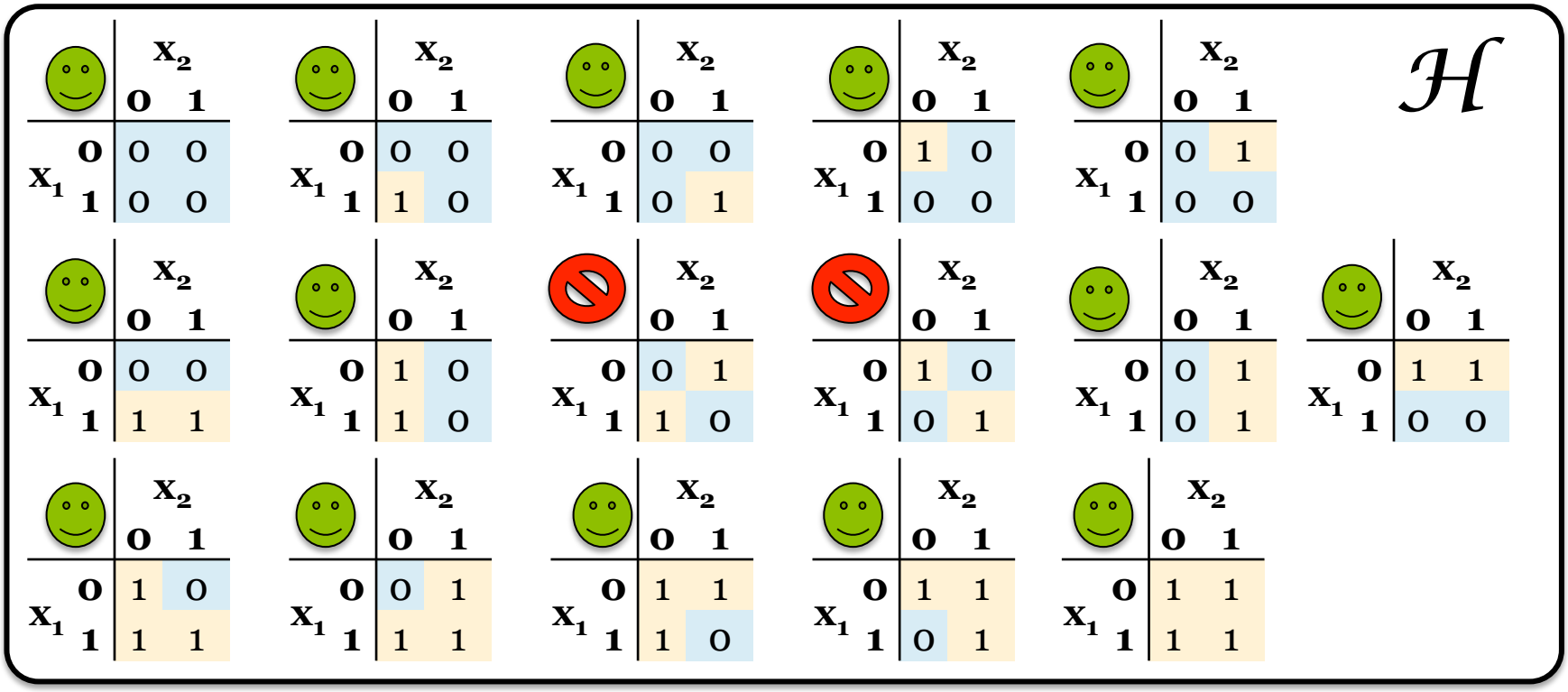
$$f(\mathbf{x}) = w_0 + \mathbf{wx}$$

f(**x**) is also called the decision boundary

- Assign $\hat{y} = 1$ to all **x** where $f(\mathbf{x}) > 0$
- Assign $\hat{y} = -1$ to all **x** where $f(\mathbf{x}) < 0$

$$\hat{y} = \text{sgn}(f(\mathbf{x}))$$

Hypothesis space for linear classifiers



Canonical representation

With $\mathbf{w} = (w_1, \ldots, w_N)^T$ and $\mathbf{x} = (x_1, \ldots, x_N)^T$:

$$f(x) = w_0 + \mathbf{wx}$$
$$= w_0 + \sum_{i=1\ldots N} w_i x_i$$

w_0 is called the bias term.

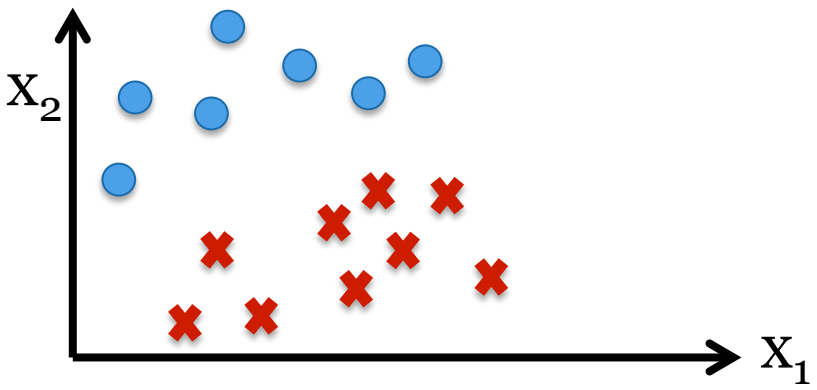
The canonical representation redefines $\mathbf{w}$, $\mathbf{x}$

as $\mathbf{w} = (w_0, w_1, \ldots, w_N)^T$

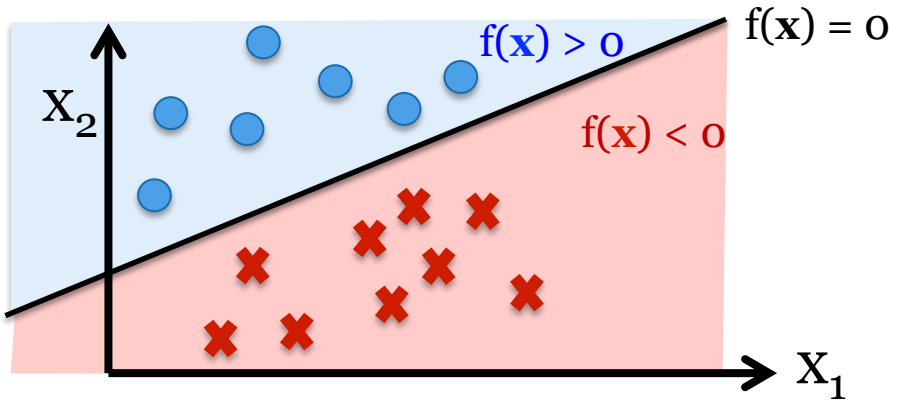
and $\mathbf{x} = (1, x_1, \ldots, x_N)^T$

=> $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$

Learning a linear classifier

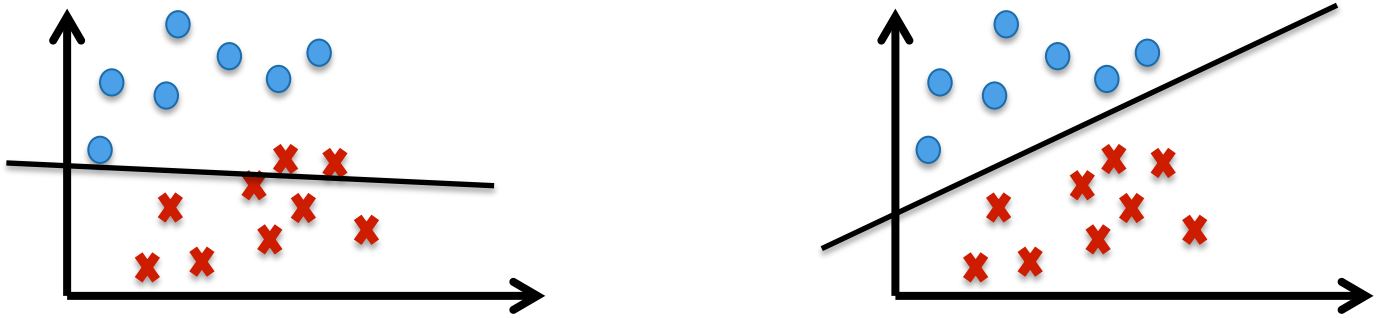


Input: Labeled training data
$\mathcal{D} = \{(\mathbf{x}^1, y^1),\ldots,(\mathbf{x}^D, y^D)\}$
plotted in the sample space $\mathcal{X} = \mathbf{R}^2$
with ●: $y^i = +1$, ✖: $y^i = -1$



Output: A decision boundary $f(\mathbf{x}) = 0$ that separates the training data
$y^i \cdot f(\mathbf{x}^i) > 0$

Which model should we pick?

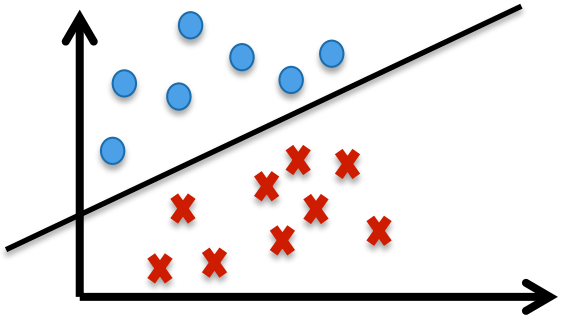


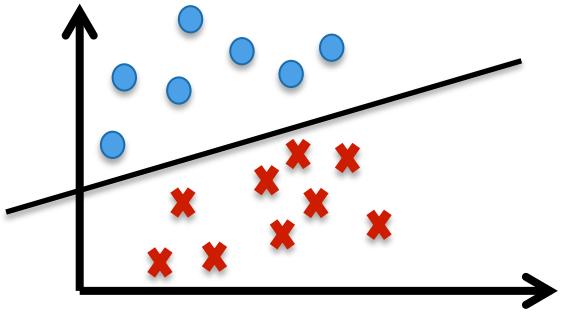
We need a metric (aka an objective function)

We would like to minimize the probability of misclassifying *unseen* examples, but we can't measure that probability.

Instead: minimize the number of misclassified training examples

Which model should we pick?



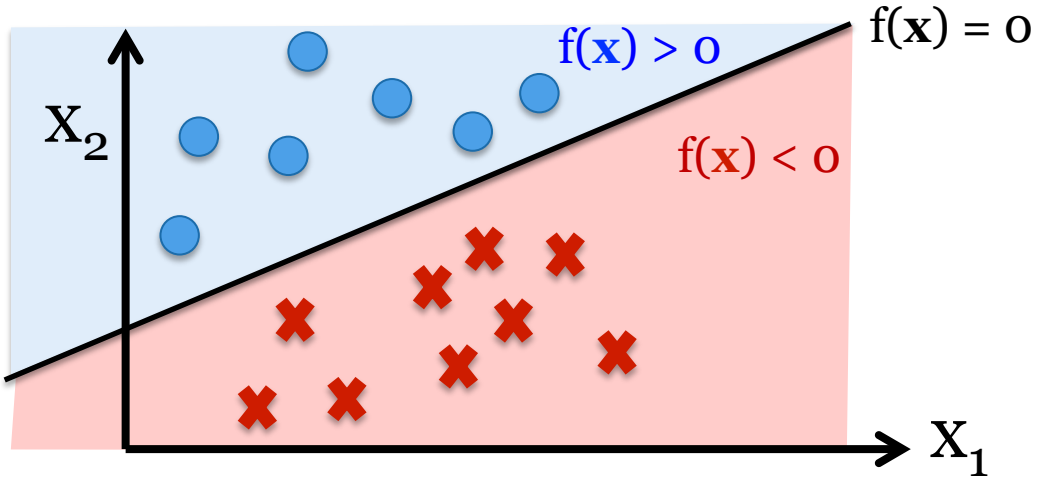


We need a more specific metric:
There may be many models that are consistent with the training data.

Loss functions provide such metrics.

Loss functions for classification

$y \cdot f(\mathbf{x}) > 0$: Correct classification



An example ($\mathbf{x}$, y) is correctly classified by $f(\mathbf{x})$ if and only if $y \cdot f(\mathbf{x}) > 0$:

Case 1 ($y = +1 = \hat{y}$): $f(\mathbf{x}) > 0 \Rightarrow y \cdot f(\mathbf{x}) > 0$

Case 2 ($y = -1 = \hat{y}$): $f(\mathbf{x}) < 0 \Rightarrow y \cdot f(\mathbf{x}) > 0$

Case 3 ($y = +1 \neq \hat{y} = -1$): $f(\mathbf{x}) > 0 \Rightarrow y \cdot f(\mathbf{x}) < 0$

Case 4 ($y = -1 \neq \hat{y} = +1$): $f(\mathbf{x}) < 0 \Rightarrow y \cdot f(\mathbf{x}) < 0$

Loss functions for classification

Loss = What penalty do we incur if we misclassify $\mathbf{x}$?

$L(y, f(\mathbf{x}))$ is the loss (aka cost) of classifier f on example $\mathbf{x}$ when the true label of $\mathbf{x}$ is y.

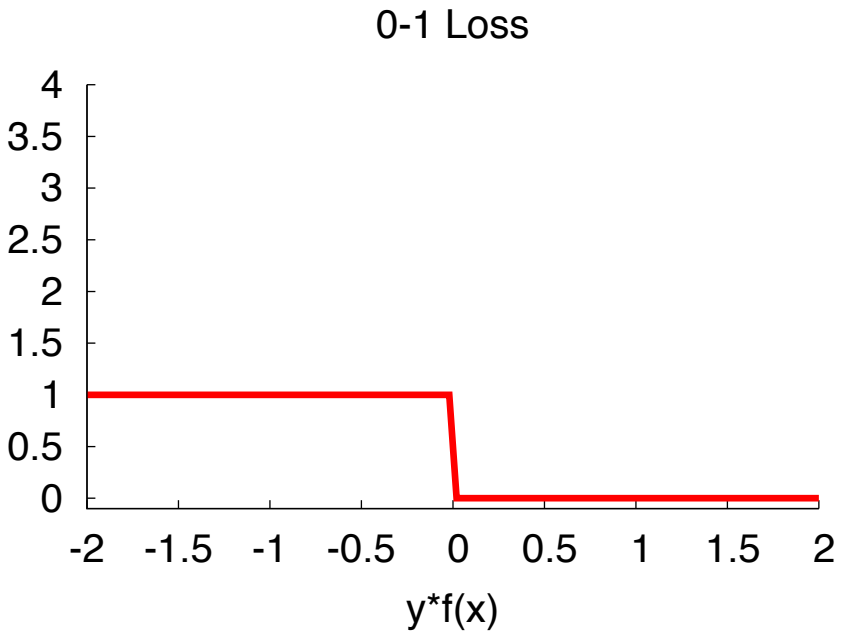
We assign label $\hat{y} = \text{sgn}(f(\mathbf{x}))$ to $\mathbf{x}$

Plots of $L(y, f(\mathbf{x}))$: x-axis is typically $y \cdot f(\mathbf{x})$

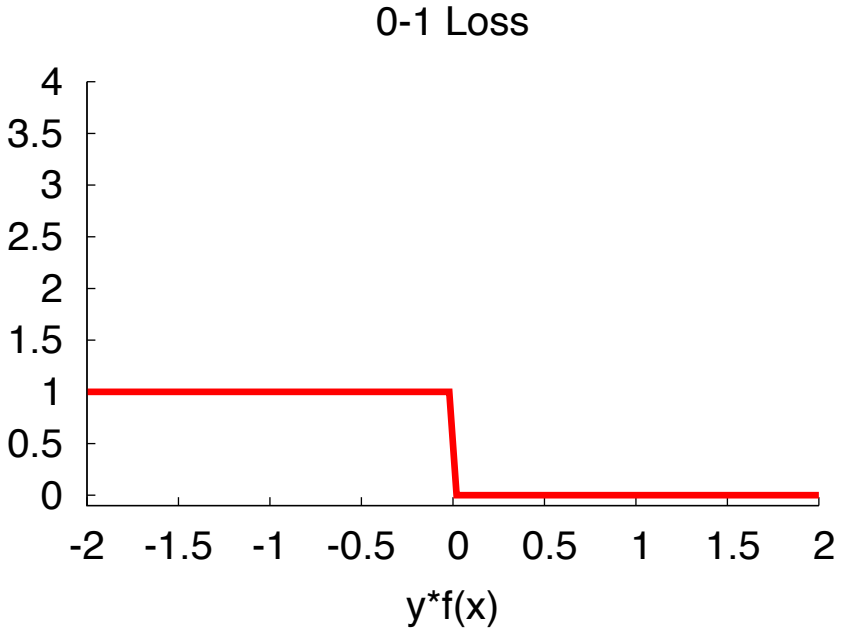
Today: 0-1 loss and square loss

(more loss functions later)

0-1 Loss



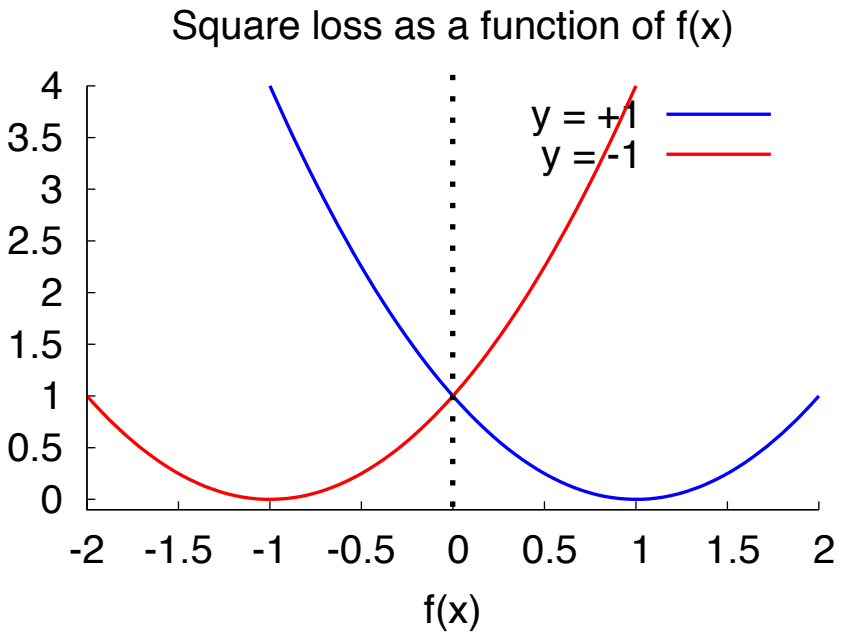
0-1 Loss

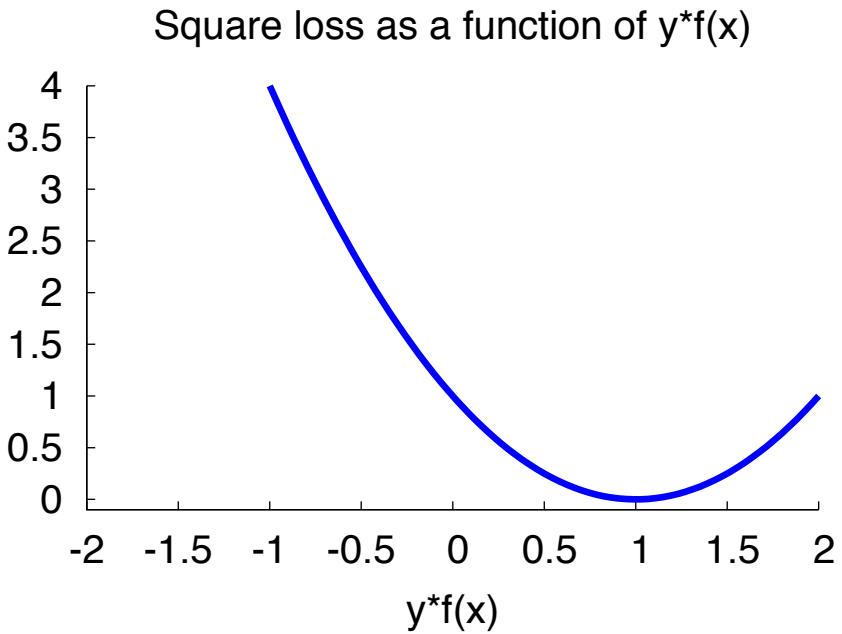


$$L(y, f(\mathbf{x})) = 0 \quad \text{iff} \quad y = \hat{y}$$
$$= 1 \quad \text{iff} \quad y \neq \hat{y}$$

$$L(\, y \cdot f(\mathbf{x})\,) = 0 \quad \text{iff } y \cdot f(\mathbf{x}) > 0 \text{ (correctly classified)}$$
$$= 1 \quad \text{iff } y \cdot f(\mathbf{x}) < 0 \text{ (misclassified)}$$

Square Loss: $(y - f(\mathbf{x}))^2$



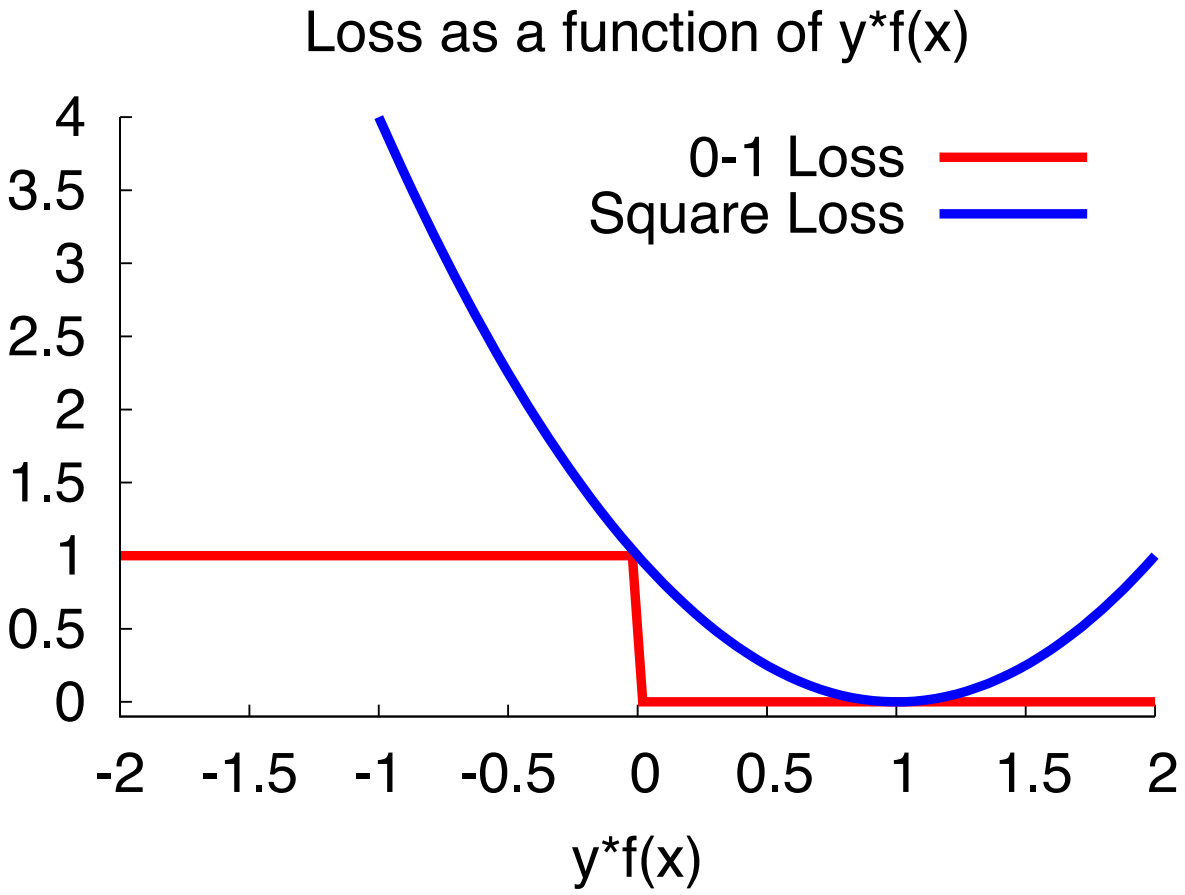


$$L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$$

Note: $L(-1, f(\mathbf{x})) = (-1 - f(\mathbf{x}))^2 = (1 + f(\mathbf{x}))^2 = L(1, -f(\mathbf{x}))$

(the loss when y=-1 [red] is the mirror of the loss when y=+1 [blue])

The square loss is a **convex upper bound** on 0-1 Loss



Loss surface

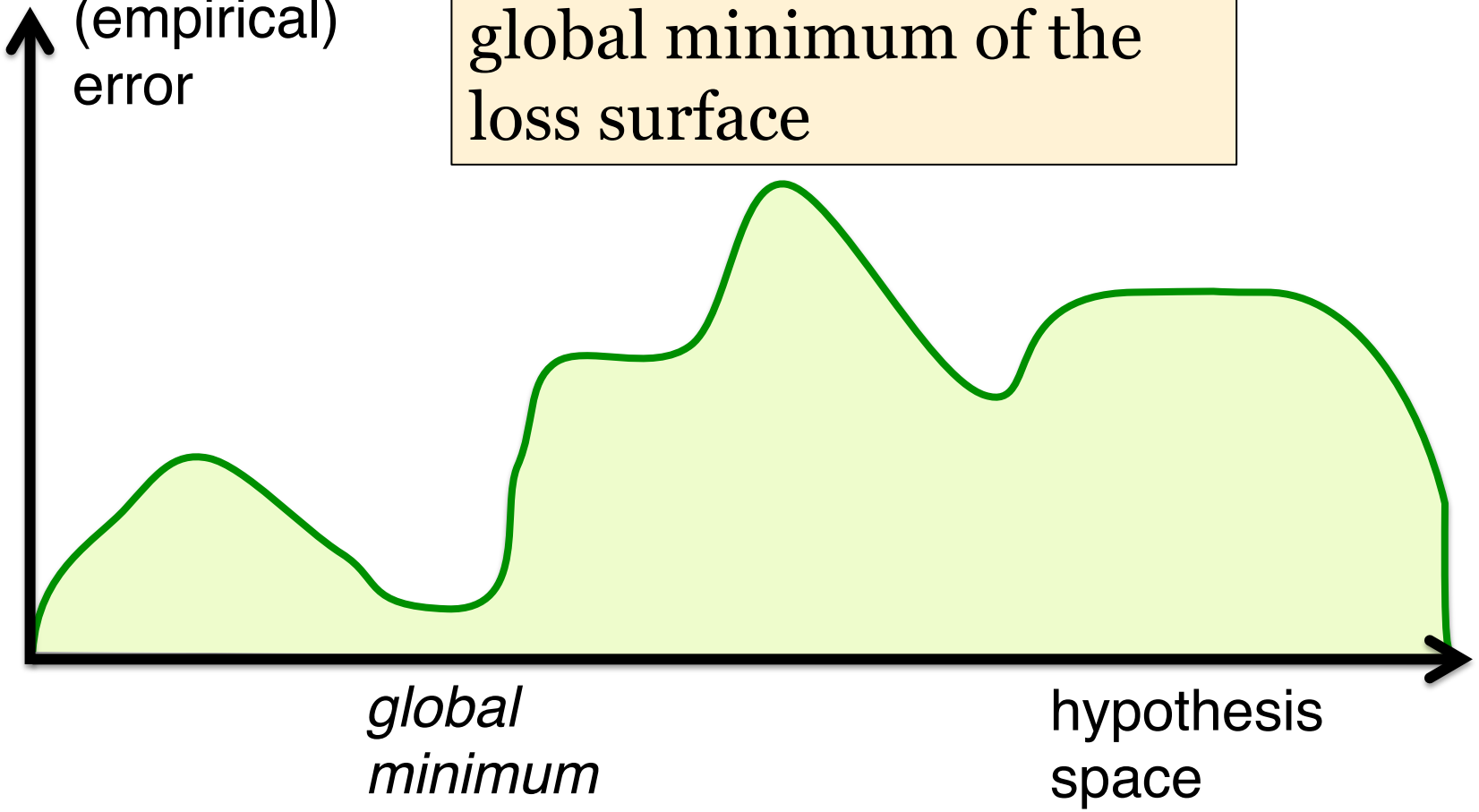
Linear classification:
Hypothesis space is parameterized by **w**

Plain English: Each **w** yields a different classifier

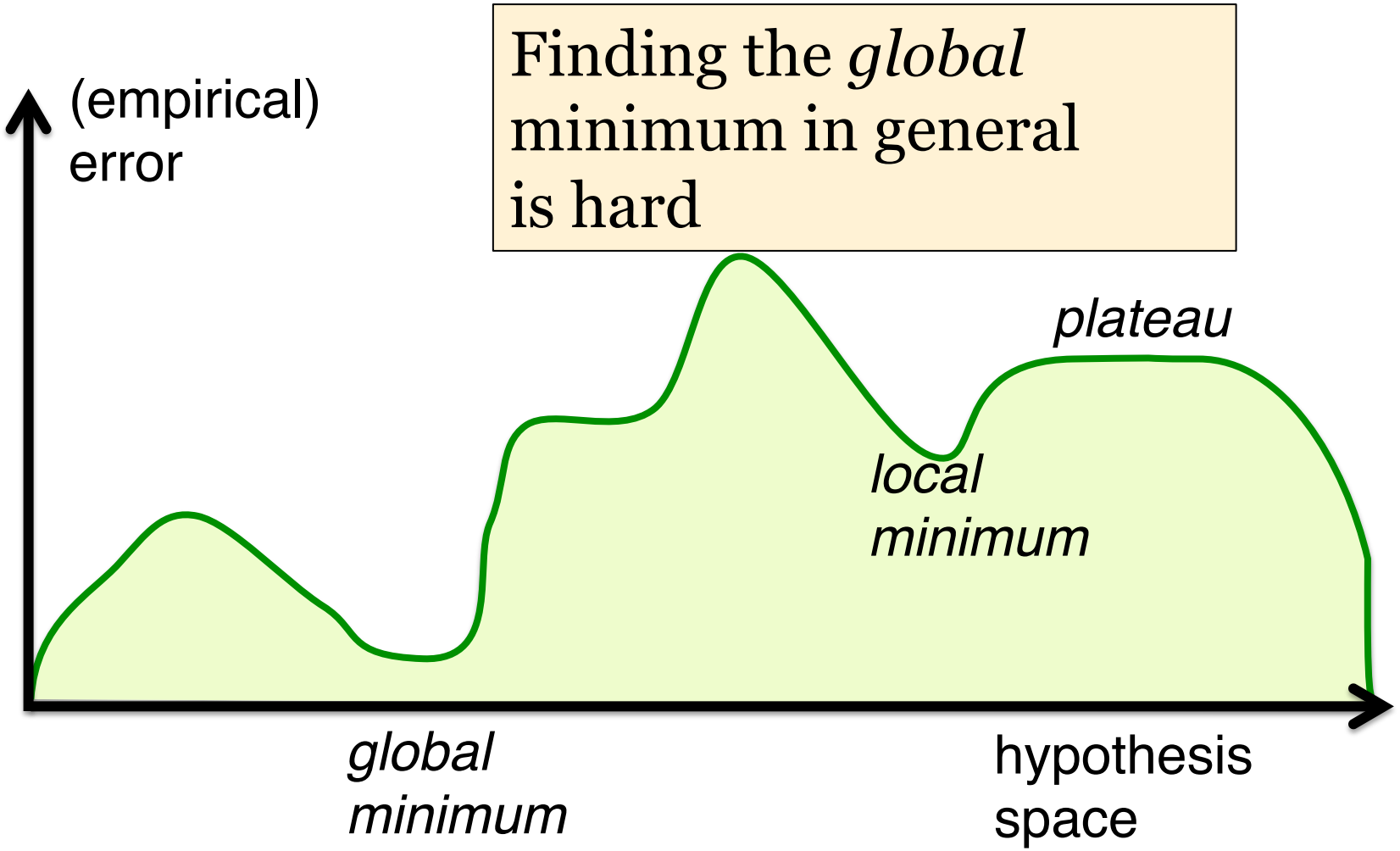
Error/Loss/Risk are all functions of **w**

The loss surface

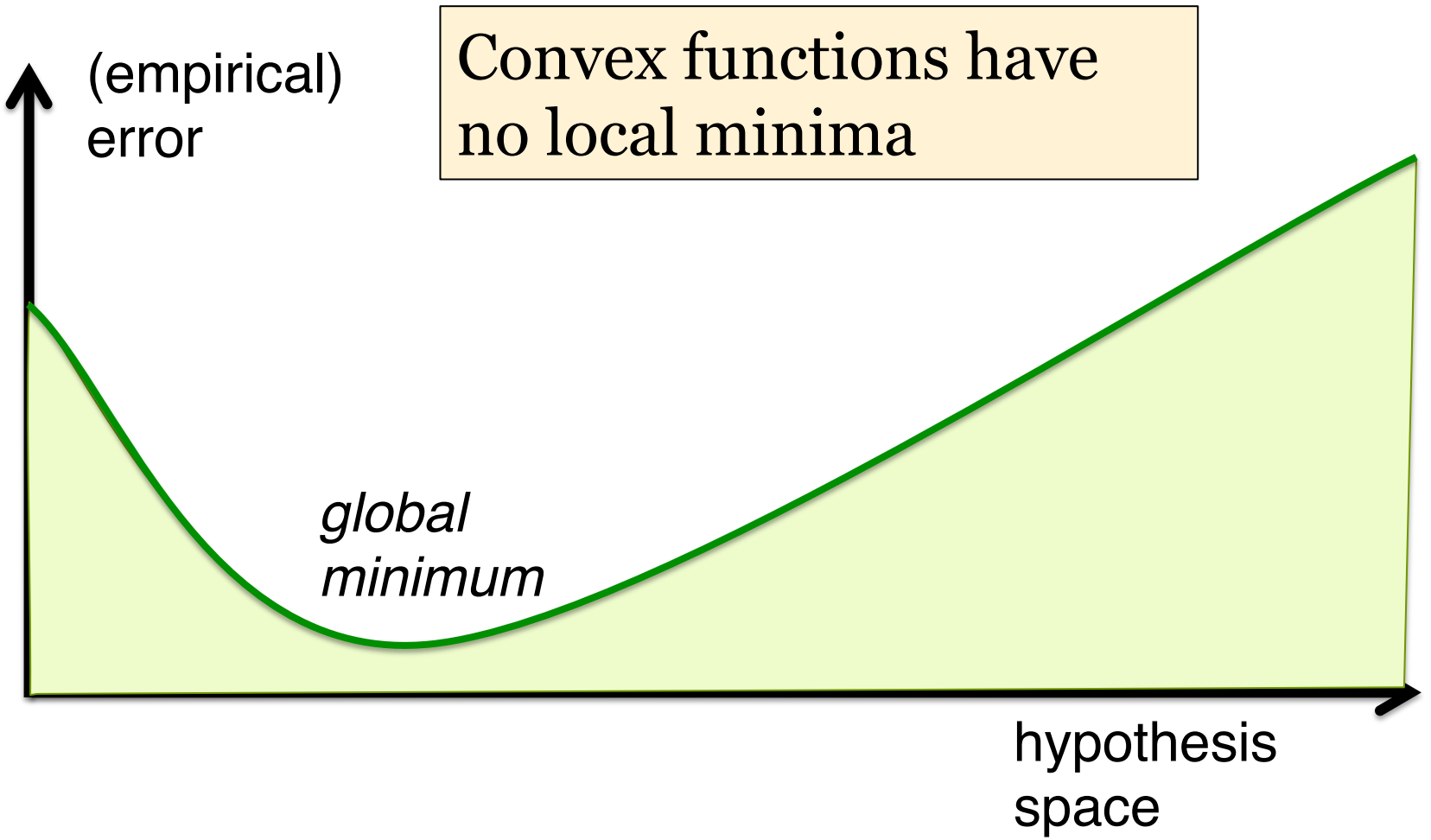
Learning = finding the global minimum of the loss surface



The loss surface



Convex loss surfaces



The **risk** of a classifier R(f)

The risk (aka generalization error) of a classifier $f(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x}$ is its **expected loss:** (= loss, averaged over all possible data sets):

$$R(f) = \int L(y, f(\mathbf{x}))\, P(\mathbf{x}, y)\, d\mathbf{x},y$$

Ideal learning objective:
Find an f that minimizes risk

Aside: The i.i.d. assumption

We always assume that training and test items are independently and identically distributed (i.i.d.):

– There is a distribution $P(\mathbf{X}, Y)$ from which the data $\mathcal{D} = \{(\mathbf{x}, y)\}$ is generated.

  Sometimes it's useful to rewrite $P(\mathbf{X}, Y)$ as $P(\mathbf{X})P(Y|\mathbf{X})$
  Usually $P(\mathbf{X}, Y)$ is unknown to us (we just know it exists)

– Training and test data are samples drawn from the *same* $P(\mathbf{X}, Y)$: they are identically distributed

– Each $(\mathbf{x}, y)$ is drawn independently from $P(\mathbf{X}, Y)$

The **empirical risk** of f(**x**)

The empirical risk of a classifier f(**x**) = **w**·**x**
on data set $\mathcal{D}$ = {(**x**^{1}, y^1),…,(**x**^{D}, y^D)}
is its average loss on the items in $\mathcal{D}$

$$R_{\mathcal{D}}(\mathrm{f}) = \frac{1}{D}\sum_{i=1}^{D} L(y^i, \mathrm{f}(\mathbf{x}^i))$$

Realistic learning objective:
Find an f that minimizes empirical risk

(Note that the learner can ignore the constant 1/D)

Empirical risk minimization

Learning:
Given training data $\mathcal{D} = \{(\mathbf{x}^1, y^1),\ldots,(\mathbf{x}^D, y^D)\}$, return the classifier $f(\mathbf{x})$ that minimizes the empirical risk $R_{\mathcal{D}}(f)$

Batch learning: Gradient Descent for Least Mean Squares (LMS)

Gradient Descent

Iterative batch learning algorithm:

– Learner updates the hypothesis based on the entire training data

– Learner has to go multiple times over the training data

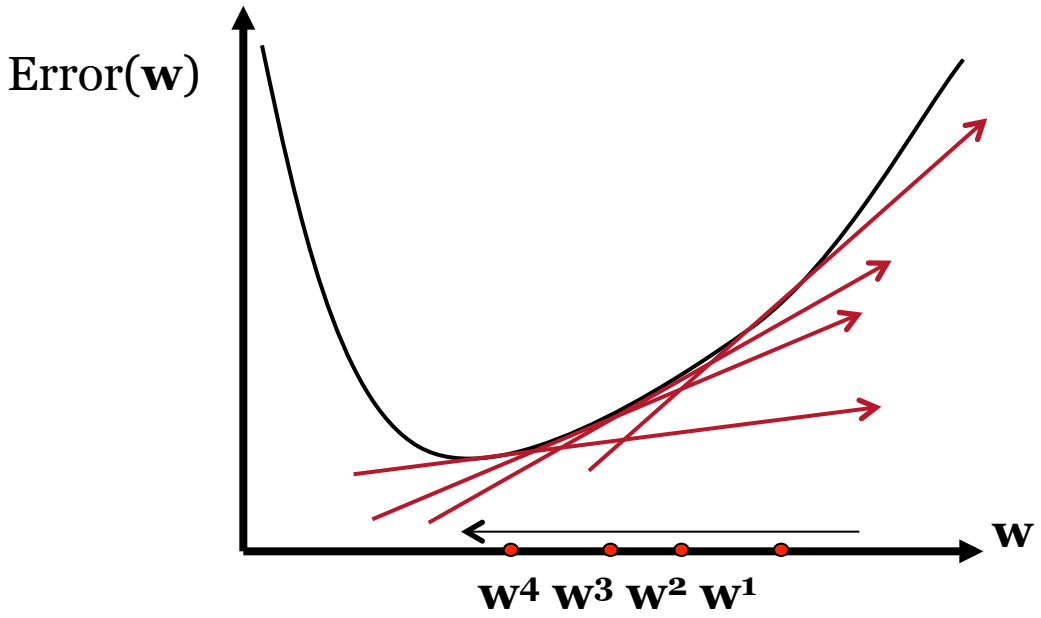
Goal: Minimize training error/loss

– At each step: move **w** in the direction of *steepest descent* along the error/loss surface

Gradient Descent

Error(**w**): Error of **w** on training data

$\mathbf{w}^i$: Weight at iteration i



Least Mean Square Error

$$\mathrm{Err}(\mathbf{w}) = \frac{1}{2}\sum_{d \in D}(y_d - \hat{y}_d)^2$$

LMS Error:
Sum of square loss over all training items (multiplied by 0.5 for convenience)

D is fixed, so no need to divide by its size

Goal of learning: Find $\mathbf{w}^* = \mathrm{argmin}(\mathrm{Err}(\mathbf{w}))$

Iterative batch learning

Initialization:

Initialize $\mathbf{w}^0$ (the initial weight vector)

For each iteration:

for i = 0…T:

Determine by how much to change w based on the entire data set $\mathcal{D}$

$\Delta\mathbf{w}$ = computeDelta($\mathcal{D}$, $\mathbf{w}^i$)

Update w:

$\mathbf{w}^{i+1}$ = update($\mathbf{w}^i$, $\Delta\mathbf{w}$)

Gradient Descent: Update

1. Compute $\nabla \mathrm{Err}(\mathbf{w}^i)$, the gradient of the training error at $\mathbf{w}^i$

   This requires going over the entire training data

$$\nabla \mathrm{Err}(\mathbf{w}) = \left(\frac{\partial \mathrm{Err}(\mathbf{w})}{\partial w_0}, \frac{\partial \mathrm{Err}(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial \mathrm{Err}(\mathbf{w})}{\partial w_N} \right)^T$$

2. Update $\mathbf{w}$:

   $\mathbf{w}^{i+1} = \mathbf{w}^i - \alpha \nabla \mathrm{Err}(\mathbf{w}^i)$

   $\alpha > 0$ is the learning rate

What's a gradient?

$$\nabla \mathrm{Err}(\mathbf{w}) = \left(\frac{\partial \mathrm{Err}(\mathbf{w})}{\partial w_0}, \frac{\partial \mathrm{Err}(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial \mathrm{Err}(\mathbf{w})}{\partial w_N} \right)^T$$

The gradient is a vector of partial derivatives

It indicates the direction of steepest *increase* in Err($\mathbf{w}$)

Hence the *minus* in the upgrade rule: $\mathbf{w}^i - \alpha \nabla \mathrm{Err}(\mathbf{w}^i)$

Computing $\nabla Err(\mathbf{w}^i)$

$$\frac{\partial Err(\mathbf{w})}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(y_d - f(\mathbf{x}_d))^2$$

$$Err(\mathbf{w}^{(j)}) = \frac{1}{2}\sum_{d\in D}(y_d - f(\mathbf{x})_d)^2$$

$$= \frac{1}{2}\frac{\partial}{\partial w_i}\sum_{d\in D}(y_d - f(\mathbf{x}_d))^2$$

$$= \frac{1}{2}\sum_{d\in D} 2(y_d - f(\mathbf{x}_d))\frac{\partial}{\partial w_i}(y_d - \mathbf{w}\cdot\mathbf{x}_d)$$

$$= -\sum_{d\in D}(y_d - f(\mathbf{x}_d))x_{di}$$

Gradient descent (batch)

Initialize $\mathbf{w}^0$ randomly

for i = 0…T:

$\Delta\mathbf{w}$ = (0, …., 0)

for every training item d = 1…D:

$f(\mathbf{x_d}) = \mathbf{w}^i \cdot \mathbf{x_d}$

for every component of $\mathbf{w}$ j = 0…N:

$\Delta w_j \mathrel{+}= \alpha(y_d - f(\mathbf{x_d})) \cdot x_{dj}$

$\mathbf{w}^{i+1} = \mathbf{w}^i + \Delta\mathbf{w}$

return $\mathbf{w}^{i+1}$ when it has converged

The batch update rule for each component of **w**

$$\Delta w_i = \alpha \sum_{d=1}^{D} (y_d - \mathbf{w}^i \cdot \mathbf{x}_d) x_{di}$$

Implementing gradient descent:
As you go through the training data, you can just accumulate the change in each component w_i of **w**

Learning rate and convergence

The learning rate is also called the *step size*.

More sophisticated algorithms (Conjugate Gradient) choose the step size automatically and converge faster.

- When the learning rate is too small, convergence is very slow
- When the learning rate is too large, we may oscillate (overshoot the global minimum)
- You have to experiment to find the right learning rate for your task

Online learning with Stochastic Gradient Descent

Stochastic Gradient Descent

Online learning algorithm:

- Learner updates the hypothesis with each training example
- No assumption that we will see the same training examples again
- Like batch gradient descent, except we update after seeing each example

Why online learning?

Too much training data:

– Can't afford to iterate over everything

Streaming scenario:

– New data will keep coming

– You can't assume you have seen everything

– Useful also for adaptation (e.g. user-specific spam detectors)

Stochastic Gradient descent (online)

Initialize $\mathbf{w^0}$ randomly

for m = 0…M:

$$f(\mathbf{x_m}) = \mathbf{w^i} \cdot \mathbf{x_m}$$

$$\Delta w_j = \alpha(y_d - f(\mathbf{x_m})) \cdot x_{mj}$$

$$\mathbf{w^{i+1}} = \mathbf{w^i} + \Delta \mathbf{w}$$

return $\mathbf{w^{i+1}}$ when it has converged

Today's key concepts

Linear classifiers

Loss functions

Gradient descent

Stochastic gradient descent