Machine Learning

Multiclass classification and Learning as Optimization

Dan Goldwasser

dgoldwas@purdue.edu

Multiclass classification

So far we have focused on Binary prediction problem.
While not an issue for KNN and DT, it did simplify things
when dealing with linear models, we could just view the classifier as a hyperplane dissecting the space into two halfspaces – class 1 and class 2. .. But what happens if you have class 3 and class 4?

We will introduce several approaches for task decomposition
and discuss their advantages

Multi-Categorical Output Tasks

- So far, our discussion was limited to ***binary predictions***
  - Well, ***almost*** (?)
- What happens if our decision is not over binary labels?
  - *Many interesting classification problems are not!*
  - **Credit card**: Approved, Deny, Further investigation needed
  - **Document classification**: sports, finance, politics
  - **OCR**: *0,1,2,3..9,A,..,Z*

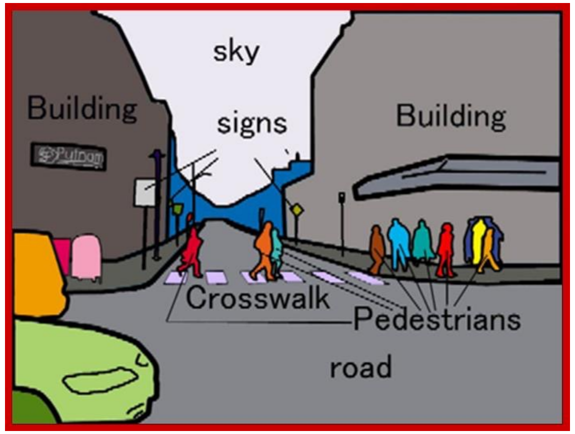
How can we approach these problems?

Hint: What is the computer science solution to: *"I can solve problem A, but now I have problem B, so…"*

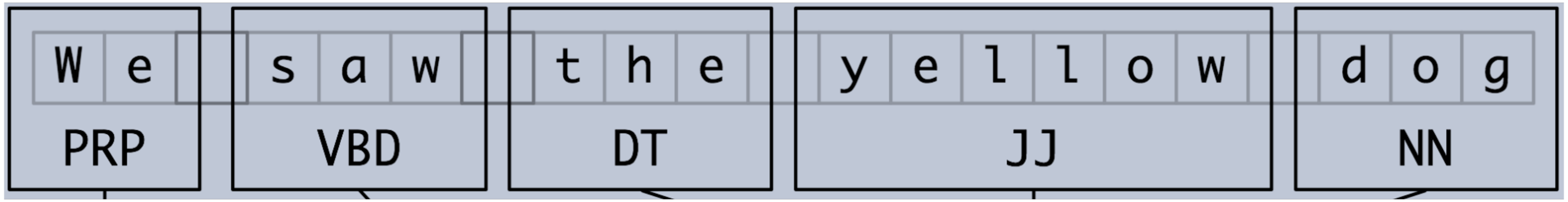
- *We will look into different reductions to binary classification problems!*

Beyond Binary Classification: Applications

Image taken from Lior Wolf VOR page



Politics?
Fashions?
Business?



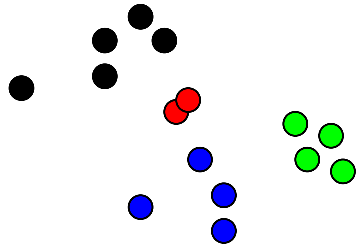
One-Vs-All

Assumption: *Each class can be separated from the rest using a binary classifier*

- **Learning**: Decomposed to learning k independent binary classifiers, one corresponding to each class
  - An example (x,y) is considered positive for class y and negative to all others.
  - Assume m examples, k class labels (assume m/k in each)
  - Classifier f_i: m/k (positive) and (k-1)m/k (negative)
- **Decision**: Winner Takes All:
  - $f(x) = \text{argmax}_i \ f_i(x) = \text{argmax}_i \ (v_i x)$

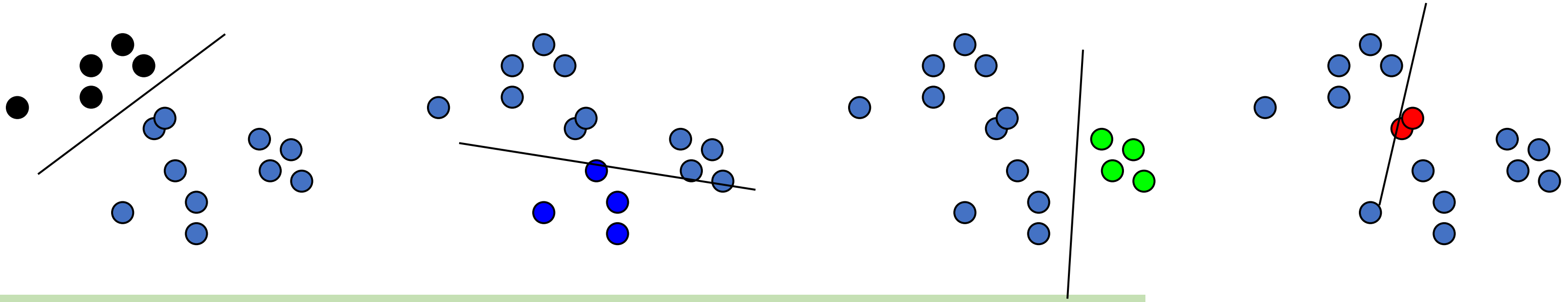
Q: Why do we need the assumption above?

Solving Multi-Class with binary learning



- **Multi-Class classifier**
  - **Function** $f : x \rightarrow \{1,2,3,\ldots,k\}$

- **Decompose into binary problems**

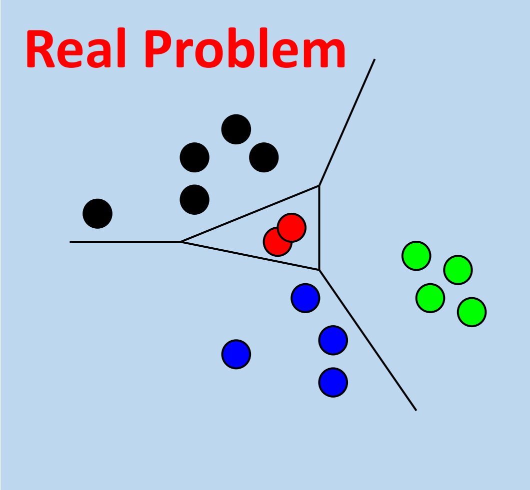


- **Not always possible to learn**
  - *No theoretical justification*
    ..unless the problem is "easy"

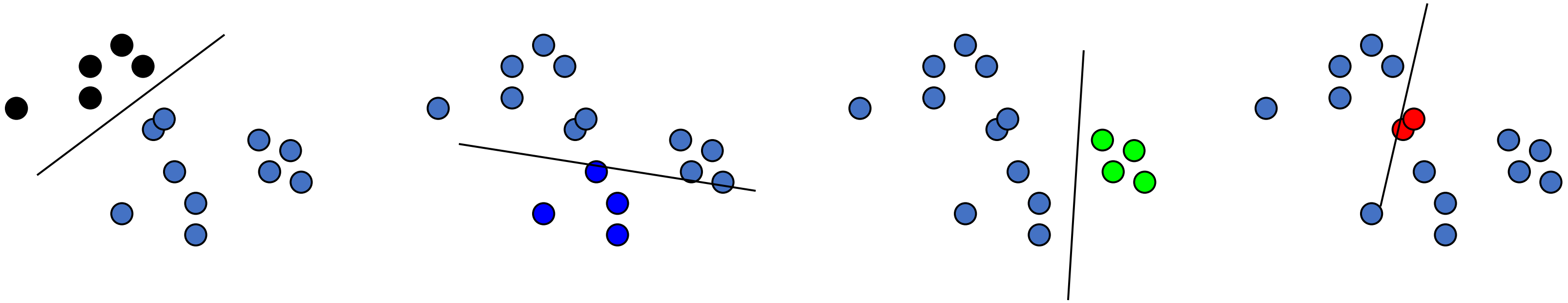
Learning via One-Versus-All

- Find $v_r, v_b, v_g, v_y \in \mathbf{R}^n$ such that
  - $v_r x > 0$ iff y = red ⊗
  - $v_b x > 0$ iff y = blue √
  - $v_g x > 0$ iff y = green √
  - $v_y x > 0$ iff y = yellow √

$\mathbf{H} = \mathbf{R}^{kn}$



- *Classification*: $f(x) = \text{argmax}_i (v_i x)$



All-Vs-All

Assumption: There is a separation between **every pair of classes** using a binary classifier in the hypothesis space.

- **Learning**: Decomposed to learning [k choose 2] ~ k^2 independent binary classifiers, separating between every two classes
  - An example (x,y) is positive for class y and negative to all others.
  - Assume m examples, k class labels. For simplicity, say, m/k in each.
  - Classifier f_{ij}: m/k (positive) and m/k (negative)

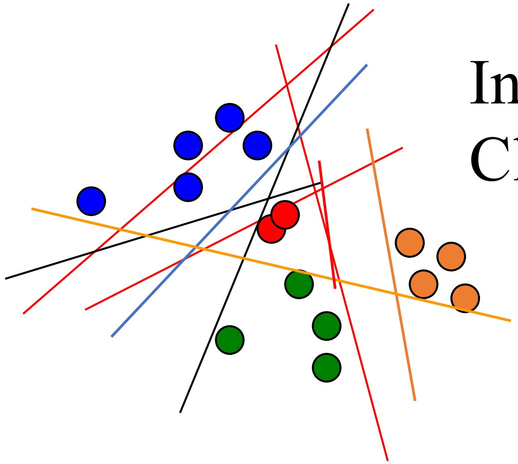
- **Decision**: Winner Decision procedure is more involved since output of binary classifier may not cohere (transitivity no ensured) :
  - **Majority**: classify example x to label i if i wins on it more often that j (j=1,…k)
  - ***Tournament***: start with n/2 pairs; continue with winners

Learning via All-Verses-All (AVA)

- Find $v_{rb}, v_{rg}, v_{ry}, v_{bg}, v_{by}, v_{gy} \in \mathbf{R}^d$ such that
  - $v_{rb}\, x > 0$ if y = red
    < 0 if y = blue
  - $v_{rg}\, x > 0$ if y = red
    < 0 if y = green
  - *... (for all pairs)*

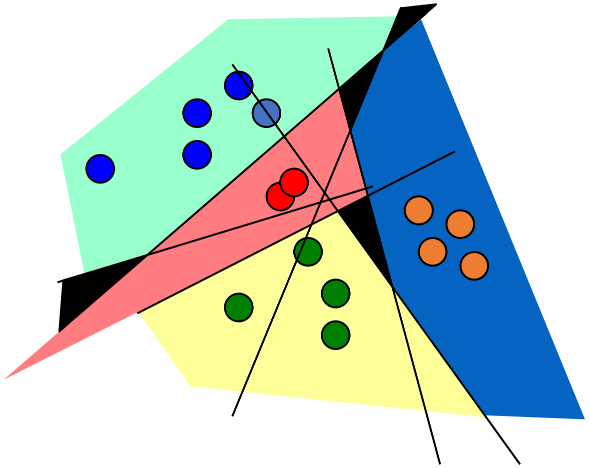
$\mathbf{H} = \mathbf{R}^{kkn}$

How to classify?



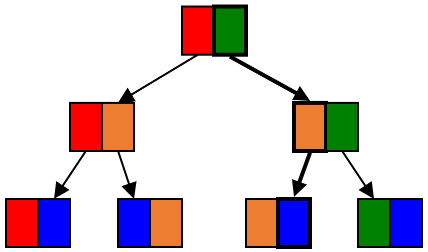
Individual Classifiers

Decision Regions

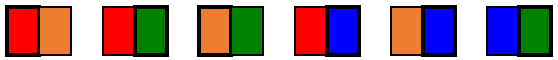


Classifying with AvA

Tree



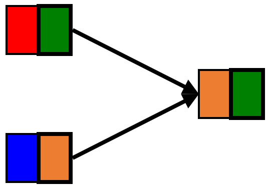
Majority Vote



1 red, 2 yellow, 2 green

➔ ?

Tournament



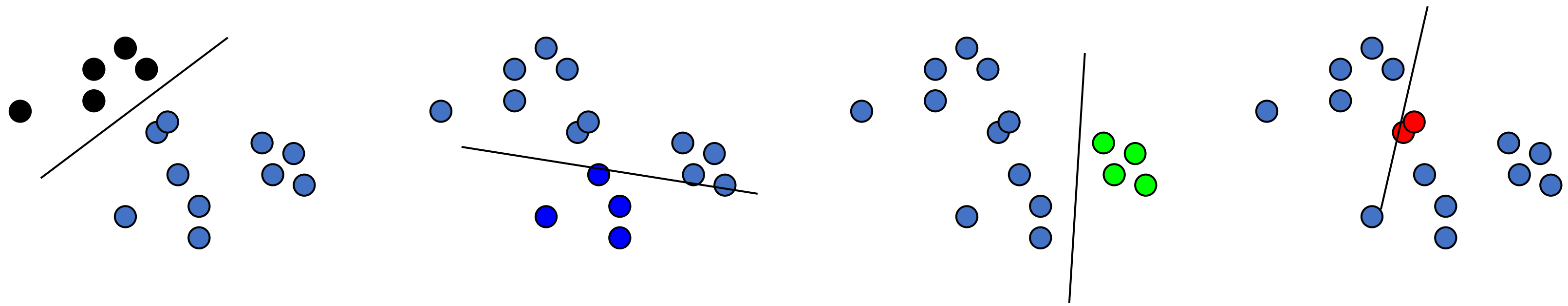
All are post-learning, may not be consistent

Problems with Decompositions

- **Learning optimizes over *local* metrics**
  - Poor *global* performance
  - What is the metric?
    - We don't care about *local* classifiers performance
  - ***Poor decomposition*** $\Rightarrow$ ***poor performance***
  - Especially true for Error Correcting Output Codes
- **Difficulty:** *how to ensure that the resulting problems are separable.*
- Can we ensure separability and learn the real objective?
- Real objective: final performance, rather than local metric

Intuition

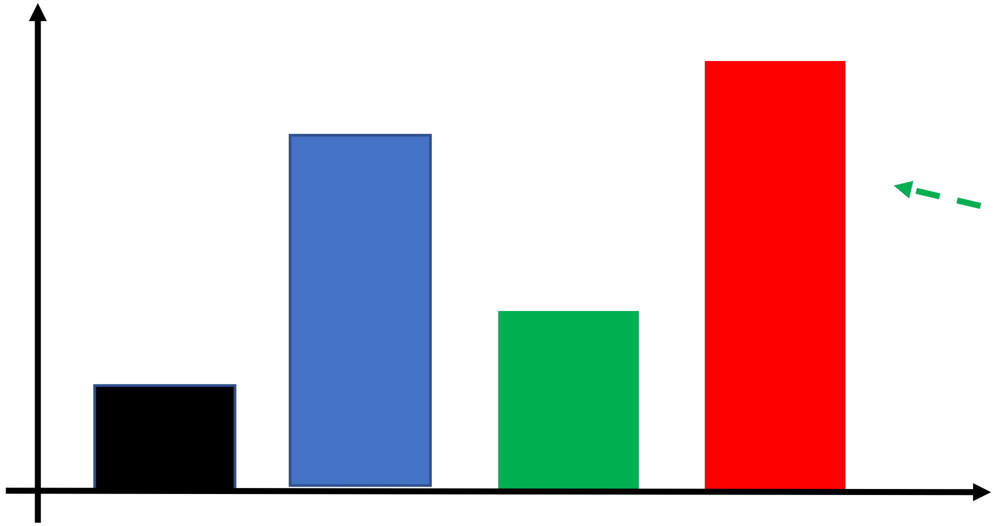
- One vs. All



- *We can think about the problem in a different way*

Score =
$F_L(x) = W_L x$

Each W_L is defined over different features



Predicted label

Multiclass classification

- The examples given to learner are pairs (x,y), $y \in \{1,\ldots k\}$
- The "black box learner" seems like *a function of only* x but, actually, *we made use of the labels* y

- ***How is y being used?***
  - y "decides" which of the k classifiers should take the example as a positive example (making it a negative to all the others)

Multiclass classification

- *How do we make decisions:*
  - Let: $f_y(x) = w_y^T x$
  - Then, we predict using: $y^* = \text{argmax}_{y=1,\ldots k}\ f_y(x)$

- ***Equivalently, we can say that we predict as follows:***
  - Predict y iff: $\forall y' \in \{1,\ldots k\},\ y' \mathrel{!}= y,\ (w_y^T - w_{y'}^T)x > 0$

- *How do we learn the k weight vectors w_y (y = 1,…k) ?*

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into $w = (w_1, w_2, \dots w_k) \in R^{nk}$
- **Key Construction**: (**Kesler Construction**)
- Represent each example (x,y), as an nk-dimensional vector, x_y with x embedded in the y-th part of it (y=1,2,…k) and the other coordinates are 0

E.g., $\mathbf{x}_y = (\mathbf{0},x,\mathbf{0},\mathbf{0}) \in \mathbf{R}^{kn}$ (*here k=4, y=2*)

Example: *predict sentiment of a product review*

Features *– words (unigram)*
Labels: *Good, Bad, Neutral.*

The feature corresponding to "sick" when the label is "good"

$$\underbrace{(\text{[”sick”,good]}, \dots,}_{\text{Good label features}} \underbrace{\text{[“sick”,neutral]}, \dots,}_{\text{Neutral label features}} \underbrace{\text{[“sick”,bad],..)}}_{\text{bad label features}}$$

Linear Separability for Multiclass

- We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into $w = (w_1, w_2, \ldots w_k) \in R^{nk}$
- **Key Construction**: (**Kesler Construction**)
- Represent each example (x,y), as an nk-dimensional vector, x_y with x embedded in the y-th part of it (y=1,2,…k) and the other coordinates are 0

E.g., $\mathbf{x}_y = (\mathbf{0}, x, \mathbf{0}, \mathbf{0}) \in \mathbf{R}^{kn}$ (*here k=4, y=2*)

- Now we can understand the decision
  - Predict y iff $\forall\, y' \in \{1,\ldots k\}, y!=y$ $(w_y^T - w_{y'}^T)\ x > 0$

  … In the nk-dimensional space:

  - Predict y iff $\forall\, y' \in \{1,\ldots k\}, y'!=y$ $w^T (x_y - x_{y'}) \equiv w^T x_{yy'} > 0$

Conclusion: The set $(x_{yy'}, +) \equiv (x_y - x_{y'}, +)$ is ***linearly separable*** from the set $(x_{yy'}, -)$ using the linear separator $w \in R^{kn}$,

Learning via Kesler Construction

- A **perceptron update** rule applied in the nk-dimensional space due to a mistake in $w^T x_{ij} > 0$
- Implies the following update:

> - Given example (x,i) (example $x \in R^n$, labeled i)
>   - $\forall j = 1,\ldots k,\ \ i \neq j$
>   - If $(w_i^T - w_j^T)\ x < 0$ (mistaken prediction)
>   - $w_i \leftarrow w_i + x$ (*promotion*)
>   - $w_j \leftarrow w_j - x$ (*demotion*)

For any given example, you can potentially make K-1 promotion steps for w_i and K-1 demotion steps, for the *different* w_j

Conservative update

- The general scheme suggests that on every mistake:
  - Promote w_i and demote k-1 weight vectors w_j
- A **conservative** update:
  - In case of a mistake: only the weights corresponding to the target node i and that closest node j are updated.
  - Let: $j^* = \text{argmin}_{j=1,\ldots k} (w_i^T - w_j^T)\, x$
  - If $(w_i^T - w_{j^*}^T)\; x < 0$ (mistaken prediction)
  - $w_i \leftarrow w_i + x$ (promotion) and $w_{j^*} \leftarrow w_{j^*} - x$ (demotion)
  - **Other weight vectors are not updated.**

Margin in the Multiclass case

- How would you change the notion of a margin for multiclass classification case?

Goal for Today's class

Learning as Optimization

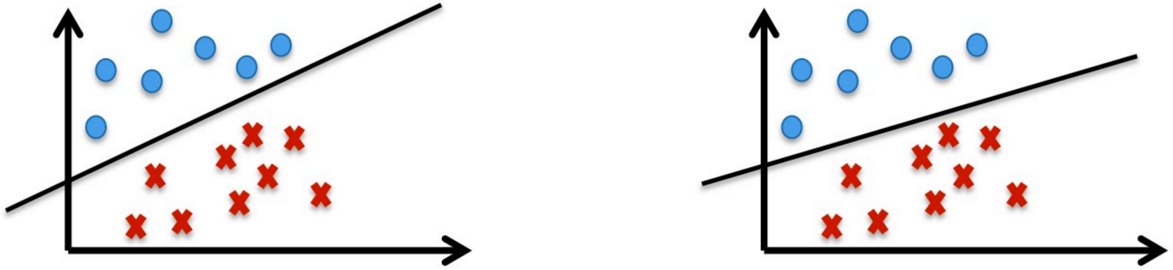
Up until now we thought about learning algorithms as separate things each having different implementation, convergence properties, etc.
Today, we'll start talking about a shared framework for learning that unifies this discussion. The idea is to think about learning as an optimization process, with a clearly defined objective.
Learning algorithms would now just be different ways of optimizing the same function.

Quick Recap

- **Linear classifiers**
  - So far we looked perceptron (winnow)
  - Combines model (linear representation) with algorithm (update rule)
  - Let's try to abstract – we want to find a linear function performing best on the data
    - What are good properties of this classifier?
    - Want to explicitly control for error + "simplicity"
  - **How can we discuss these terms separately from a specific algorithm?**

Learning using the Perceptron Algorithm

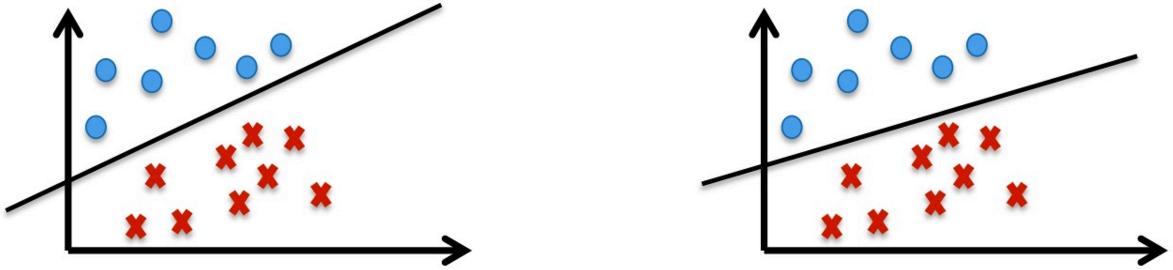
- Perceptron guarantee: *find a linear separator (if the data is separable)*



- There could be many models consistent with the training data
  - ***How did the perceptron algorithm deal with this problem?***
- Trading some training error for better generalization

Learning as Optimization

- Instead we can think about learning as optimizing an objective function (e.g., *minimize mistakes*)



  - We can incorporate other considerations by modifying the objective function (*regularization*)
  - Sometime referred to as *structural risk minimization*

Reminder: Loss functions

- To formalize performance let's define a *loss function:*

$$loss(y, \hat{y})$$

  - Where $\hat{y}$ is the gold label

- *The loss function measures the error on a single instance*
  - Specific definition depends on the learning task

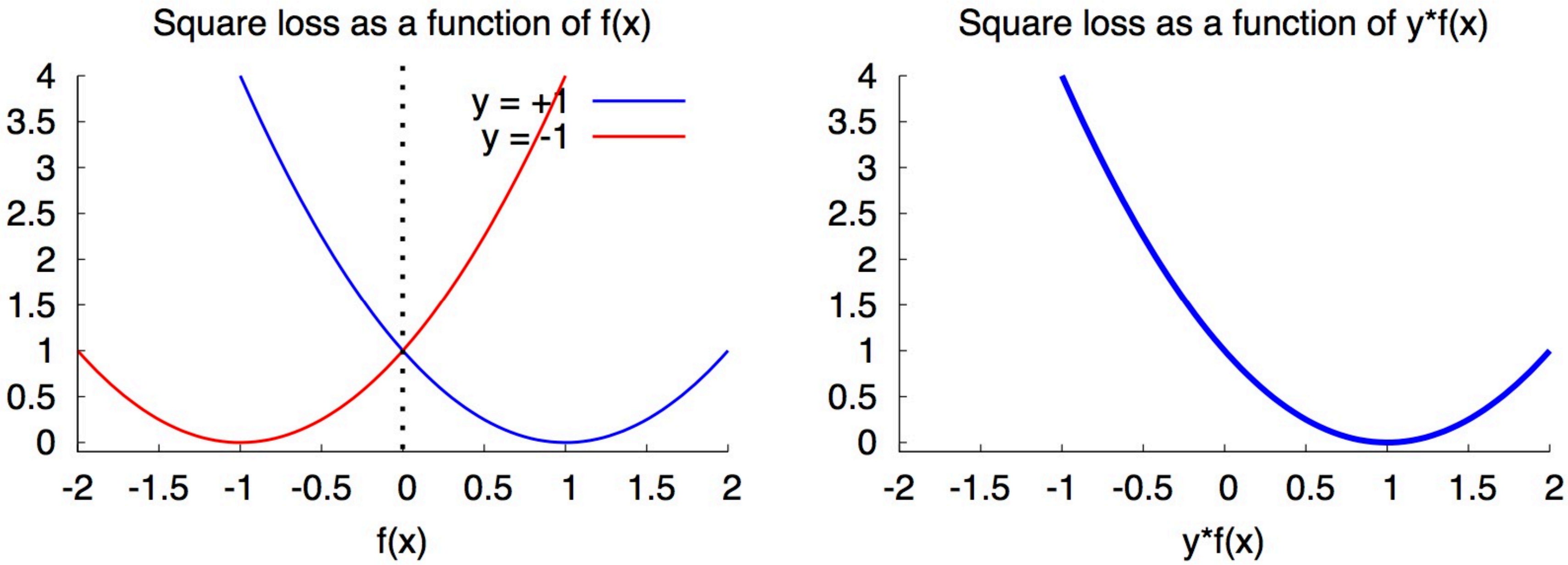
Regression

$$loss(y, \hat{y}) = (y - \hat{y})^2$$

Binary classification

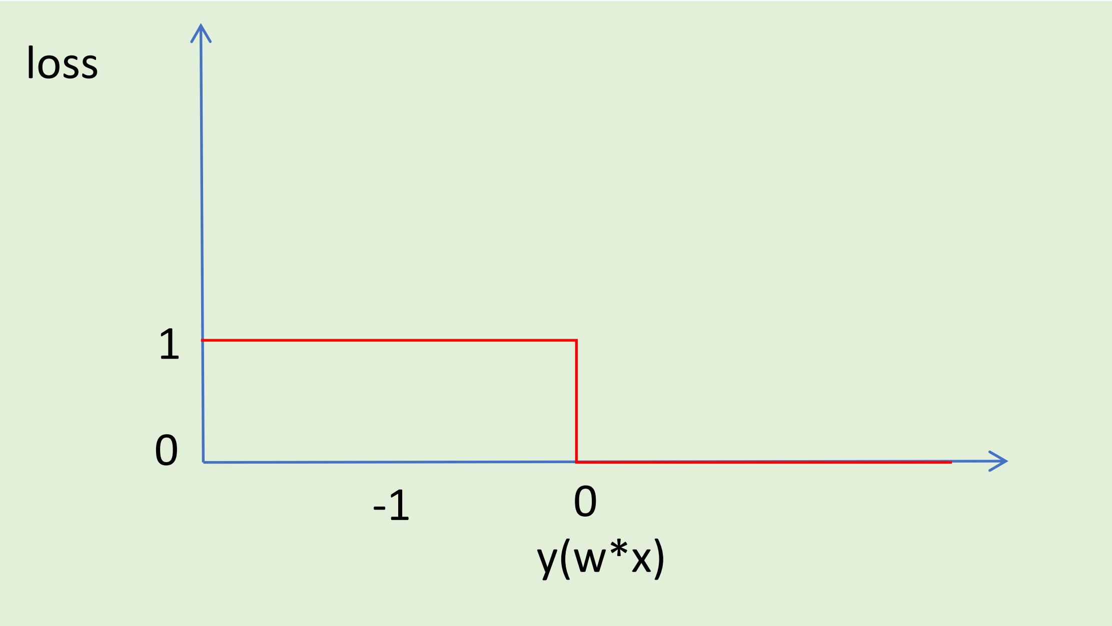
$$loss(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & otherwise \end{cases}$$

Reminder: Square Loss



$$Loss(y, f(x)) = (y - f(x))^2$$

Reminder: 0-1 Loss



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

$$loss(y \cdot f(x)) = 0 \quad \text{iff} \quad y \cdot f(x) > 0 \quad (\text{correct})$$
$$loss(y \cdot f(x)) = 1 \quad \text{iff} \quad y \cdot f(x) < 0 \quad (\text{Misclassified})$$

The **Risk** of a Classifier: R(f)

- The risk (aka generalization error) of a classifier is its ***expected loss*** (the loss averaged over all possible datasets)

$$R(f) = \int L(y, f(x))P(x, y)dx, y$$

$$\epsilon \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(y, f(x))] = \sum_{(x,y)} \mathcal{D}(x, y)\ell(y, f(x))$$

- **Ideal learning objective**: find an f that minimizes the risk

The **Empirical** Risk of f(x)

- The empirical risk of a classifier on a dataset D is its average loss on the items in d

$$R_D(f) = \frac{1}{D} \sum_{D} loss(y_i, f(x_i))$$

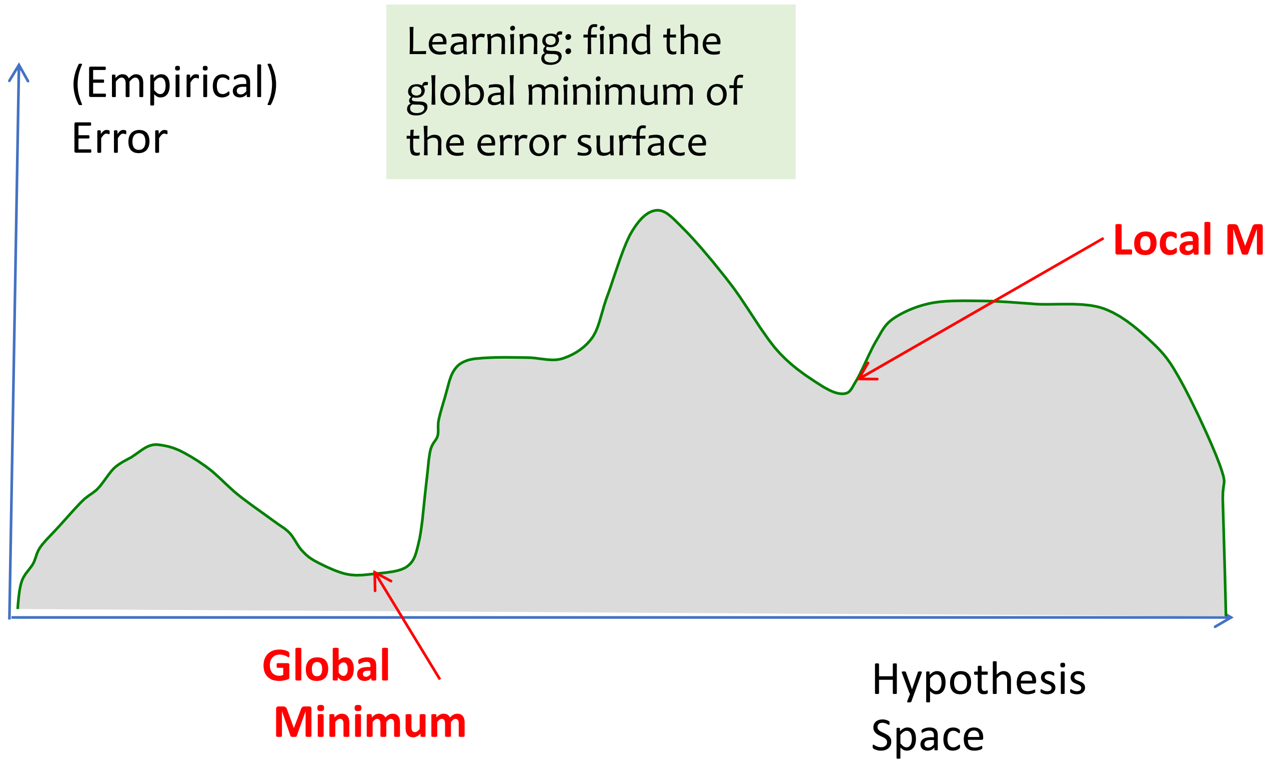
- **Realistic learning objective**: find an *f* that minimizes the empirical risk

Empirical Risk Minimization

- Learning: Given a training dataset D, return the classifier f(x) that minimizes the empirical risk
- *Given this definition we can view learning as a* ***minimization problem***
  - The objective function to minimize (empirical risk) is defined with respect to a specific loss function
  - Our minimization procedure (aka **learning**) will be influenced by the choice of loss function
    - **Some are easier to minimize than other!**

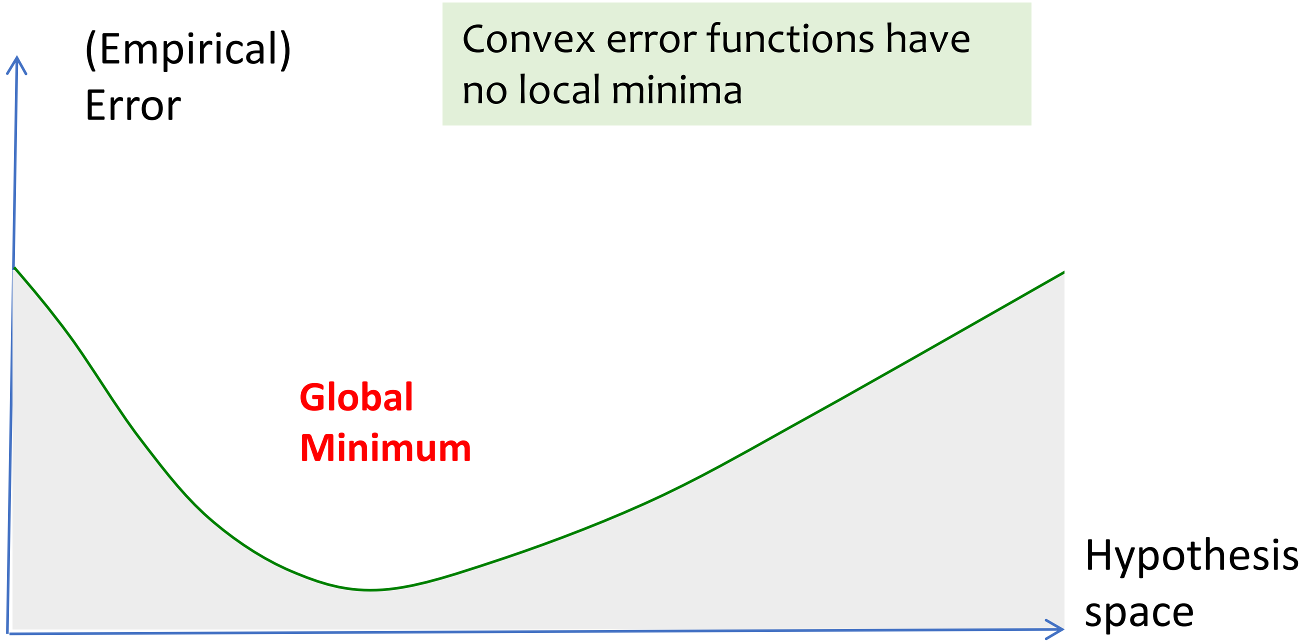
Error Surface

- **Linear classifiers**: hypothesis space parameterized by w
- *Error/Loss/Risk* are all functions of w



Convex Error Surfaces

- **Convex functions** have a single minimum point
  - Local minimum = global minimum
  - *Easier to optimize*

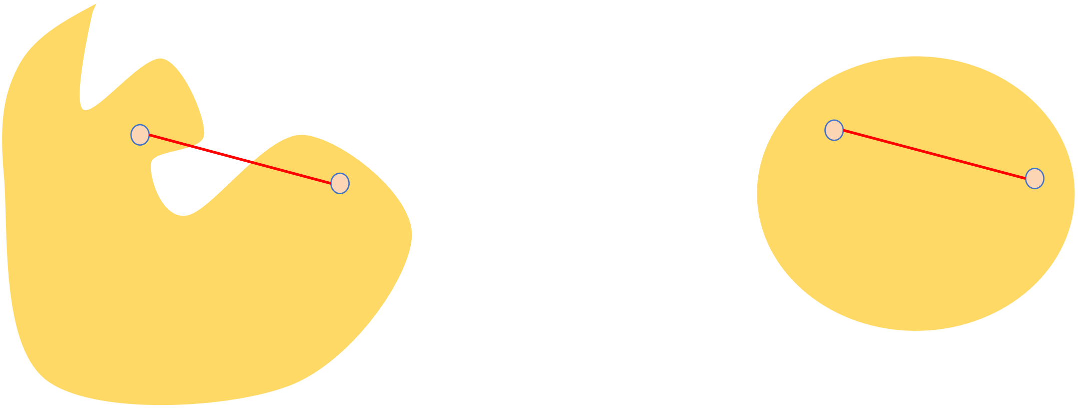


Convex Sets

- A **set** S is **convex** if

$$\forall x, y \in S, \alpha x + (1-\alpha) y \in S \qquad (0 \le \alpha \le 1)$$

(the line segment joining x and y is contained in S)



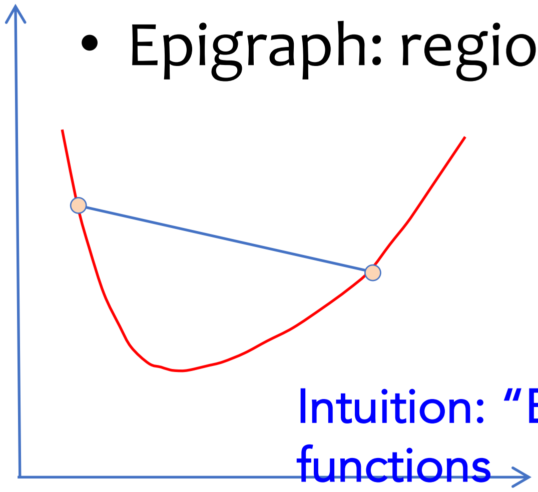
Convex Function

- A function f defined on a convex set is convex if

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \qquad (0 < \alpha < 1)$$

f is convex if the chord joining any two points is always above the graph.

- A function f is convex if its epigraph is a convex set
  - Epigraph: region above the graph of the function f



Intuition: "Bowl shaped" functions

If f is convex ➔ - f is concave

Checking Convexity

- According to definition: chord lies above function

$$f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha) f(y) \qquad (0 < \alpha < 1)$$

- If f is differentiable: f lies above all tangent lines

$$f(y) \ge f(x) + f'(x)(y-x) \qquad f(y) \ge f(x) + \nabla f(x)(y-x)$$

- For twice differentiable functions: 2nd derivative is non-negative

$$f''(x) \ge 0 \qquad \nabla^2 f(x) \succeq 0$$

- If the above functions are strict inequalities, f is strictly convex

Example: Checking Convexity

$$f(x) = x\log x$$

$$f'(x) = \log x + 1$$

$$f''(x) = \frac{1}{x}$$

- f''(x) > 0 for all x>0 ➔ f(x) is (strictly) convex!
  - *Note that the domain of log(x) is x>0*

A Few More Examples

Exponential: e^{ax} is convex, for any a

Powers: e^a is convex, if $a \geq 1, \ a \leq 0$ else: concave

Powers of abs: $||x||^p, \quad p \geq 1$

Logarithm: $\log x$ is concave

Log sum exp: $f(x) = log(e^{x_1} + ... + e^{x_n})$

Operations that preserve convexity

1. Positive Scaling and addition: f, g convex $\rightarrow w_1 f + w_2 g$ convex
2. Affine function composition: f convex $\rightarrow f(Ax+b)$ convex
3. Pointwise maximum: f, g convex $\rightarrow h(x) = max(f(x), g(x))$ convex
4. f convex, g convex non decreasing $\rightarrow h(x) = g(f(x))$ convex
5. f concave , g convex non increasing $\rightarrow h(x) = g(f(x))$ convex

You can also show convexity using these operations

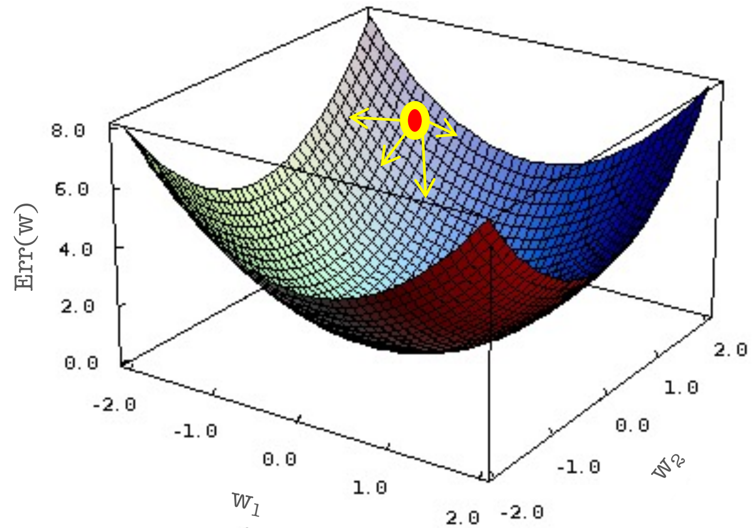
$h(x) = max(\|x\|, x^2)$	Property 3
$exp(f(x)), f(x)$ is convex	Property 4
$\frac{1}{\log x}, \quad x > 1$	Property 5

Error Surface for Squared Loss

We add the ½ for convenience

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

$$y = \mathrm{w}^T x$$



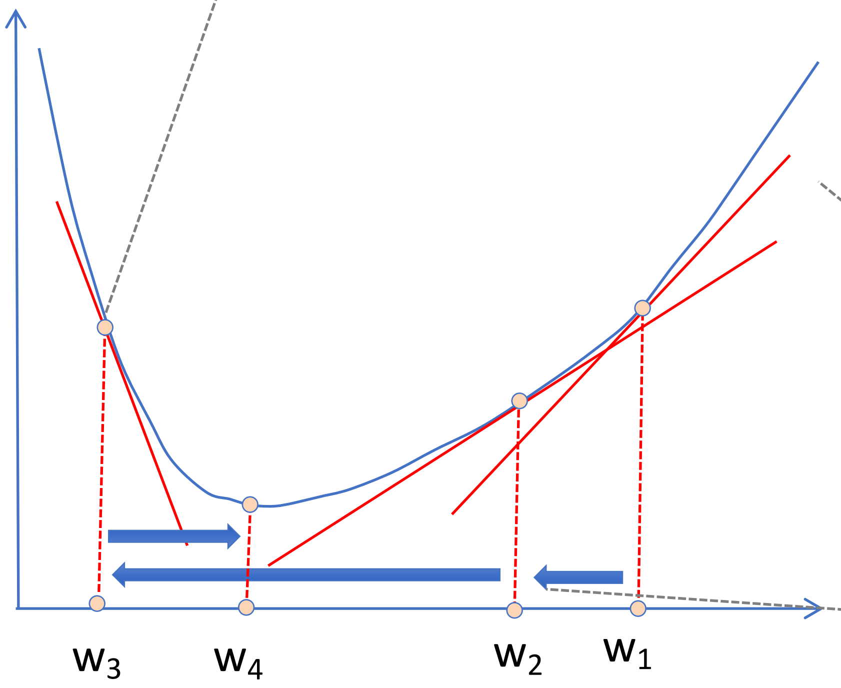
Since $\hat{y}$ is a constant (for a given dataset), the Error function is a *quadratic function* of W (paraboloid)

➔ Squared Loss function is convex!

How can we find the global minimum?

Gradient Descent Intuition

(3) We also need to determine the step size (aka learning rate).
What happens if we overshoot?



(1) The derivative of the function at w_1 is the slope of the tangent line
→ Positive slope (increasing)
Which direction should we move to decrease the value of Err(w) ?

(2) The gradient determines the direction of steepest increase of Err(w) *(go in the opposite direction)*

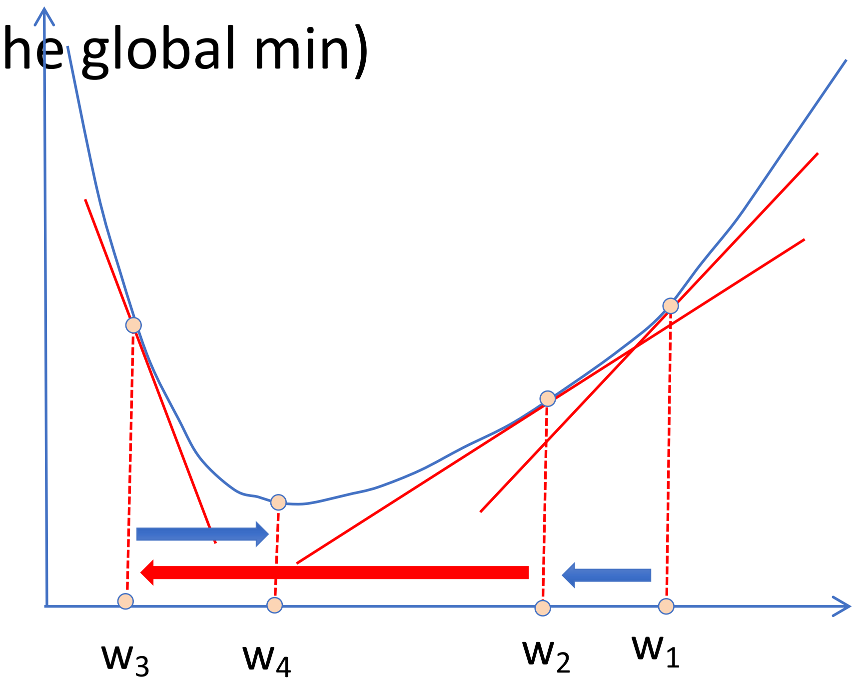
What is the gradient of Error(w) at this point?

Note about GD step size

- Setting the step size to a very small value
  - Slow convergence rate
- Setting the step size to a large value
  - May oscillate (consistently overshoot the global min)
- Tune experimentally
  - More sophisticated algorithm

  set the value automatically

  (conjugate gradient)



The Gradient of Error(w)

The gradient is a generalization of the derivative

$$\nabla Err(\mathrm{w}) = \left(\frac{\partial Err(w)}{\partial w_0}, \frac{\partial Err(w)}{\partial w_1}, \ldots, \frac{\partial Err(w)}{\partial w_n} \right)$$

The gradient is a vector of partial derivatives.

It Indicates the *direction of steepest increase* in Err(w), for each one of w's coordinates

Gradient Descent Updates

- Compute the gradient of the training error at each iteration
  - Batch mode: compute the gradient over all training examples

$$\nabla Err(\mathrm{w}) = \left(\frac{\partial Err(w)}{\partial w_0}, \frac{\partial Err(w)}{\partial w_1}, \ldots, \frac{\partial Err(w)}{\partial w_n} \right)$$

- Update w:

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

Learning rate (>0)

Computing ∇Err(w^i) for Squared Loss

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

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

$$= \frac{1}{2}\frac{\partial}{\partial \mathrm{w}_i}\sum_{d \in D}(\mathrm{y}_d - \mathrm{f}(\mathbf{x}_d))^2$$

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

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

Batch Update Rule for Each w_i

- **Implementing gradient descent**: *as you go through the training data, accumulate the change in each w_i of W*

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