Project Presentations

9	3/1	Final project presentation
	3/3	Final project presentation
10	3/8	Final project presentation
	3/10	Final project presentation
Final's week	TBA	Project paper due

You are HIGHLY encouraged to **attend all days** of the presentations – this is a chance to see different approaches and areas of application of what was covered in class in a setting similar to a conference.

The PRESENTATION SLIDES of ALL GROUPS are due by

Monday, 2/28 @ 11:59 pm

The reason why **all groups** need to submit the slides is to make sure that they show the **status of project at the start of all presentations** (the **slides that you submit** are the ones that are going to **be presented and evaluated**). This is for fairness to all students, given the time differential between the first and last day of presentations.

The presentation should discuss the **problem that you are trying to solve**, the **data that you are using**, the **proposed solution**(s), and the **results that you have so far** (they can later be UPDATED IN THE PROJECT PAPER).

ECE 271B – Winter 2022

Boosting

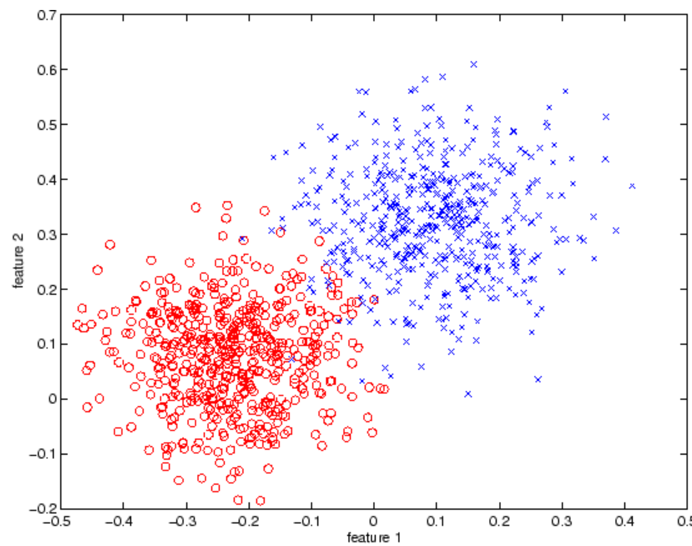
Disclaimer:
This class will be recorded
and made available to students asynchronously.

Manuela Vasconcelos

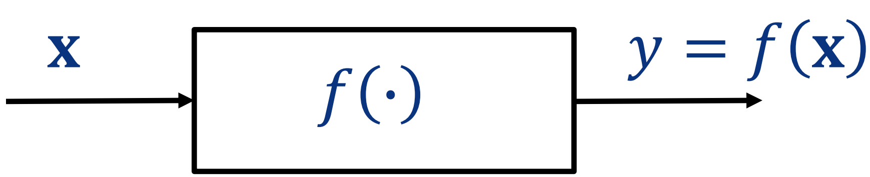
ECE Department, UCSD

Classification

- a **classification problem** has <u>two</u> types of variables
  - $\mathbf{x}$ – vector of observations (**features**) in the world
  - y – state (**class**) of the world
- e.g.
  - $\mathbf{x} \in \mathcal{X} \in \mathcal{R}^2 = (\text{fever}, \text{blood pressure})$
  - $\mathrm{y} \in \mathcal{Y} = \{\text{disease}, \text{no disease}\}$



- $\mathbf{x}$, y related by (unknown) function



- **goal**: design a **classifier** $h\colon \mathcal{X} \longrightarrow \mathcal{Y}$ such that $h(\mathbf{x}) = f(\mathbf{x}), \forall \mathbf{x}$

Perceptron Learning

- is simple stochastic gradient descent on the cost

```
set  k = 0, w_k = 0, b_k = 0
set  R = max_i ||x_i||
do {
   for i = 1:n {
      if y_i(w_k^T x_i + b_k) ≤ 0 then {
         • w_{k+1} = w_k + η y_i x_i
         • b_{k+1} = b_k + η y_i R^2
         • k = k + 1
      }
   }
} until y_i(w^T x_i + b_k) > 0, ∀i (no errors)
```

Where:

- set $k = 0, \mathbf{w}_k = 0, b_k = 0$
- set $R = \max_i \|\mathbf{x}_i\|$
- if $y_i(\mathbf{w}_k{}^T\mathbf{x}_i + b_k) \leq 0$ then
- $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
- $b_{k+1} = b_k + \eta\, y_i\, R^2$
- $k = k + 1$
- until $y_i(\mathbf{w}^T\mathbf{x}_i + b_k) > 0, \forall i$ (no errors)

Perceptron Learning

- the interesting part is that this is guaranteed to <u>converge</u> in **finite time**

- **Theorem:** Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ and

$$R = \max_i \|\mathbf{x}_i\|.$$

If there is $(\mathbf{w}^*, b^*)$ such that $\|\mathbf{w}^*\| = 1$ and

$$y_i\left({\mathbf{w}^*}^T \mathbf{x}_i + b^*\right) > \gamma, \forall i,$$

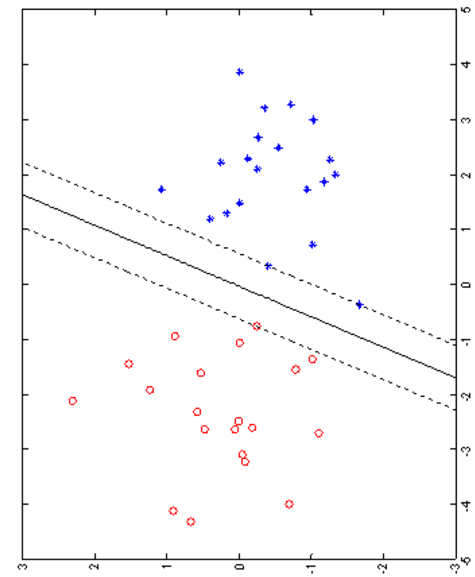
then the Perceptron will find an error free hyper−plane in at most

$$\left(\frac{2R}{\gamma}\right)^2 \text{ iterations}$$

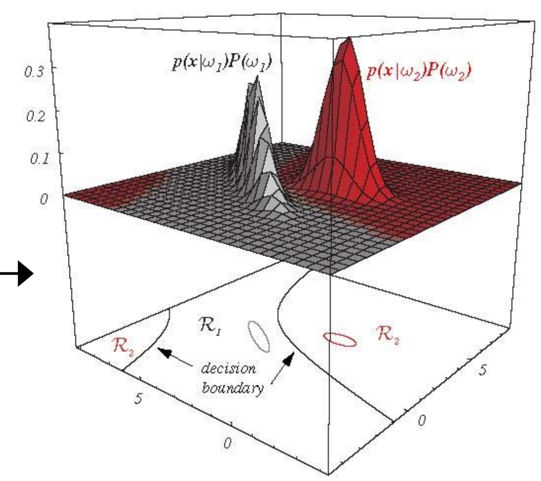
- the **main problem** is that it <u>only</u> implements a linear discriminant

Linear Discriminant

- Q: when is this a good decision function?
- clearly works if data is linearly separable
  - there is a plane which has
    - all -1's on one side
    - all 1's on the other



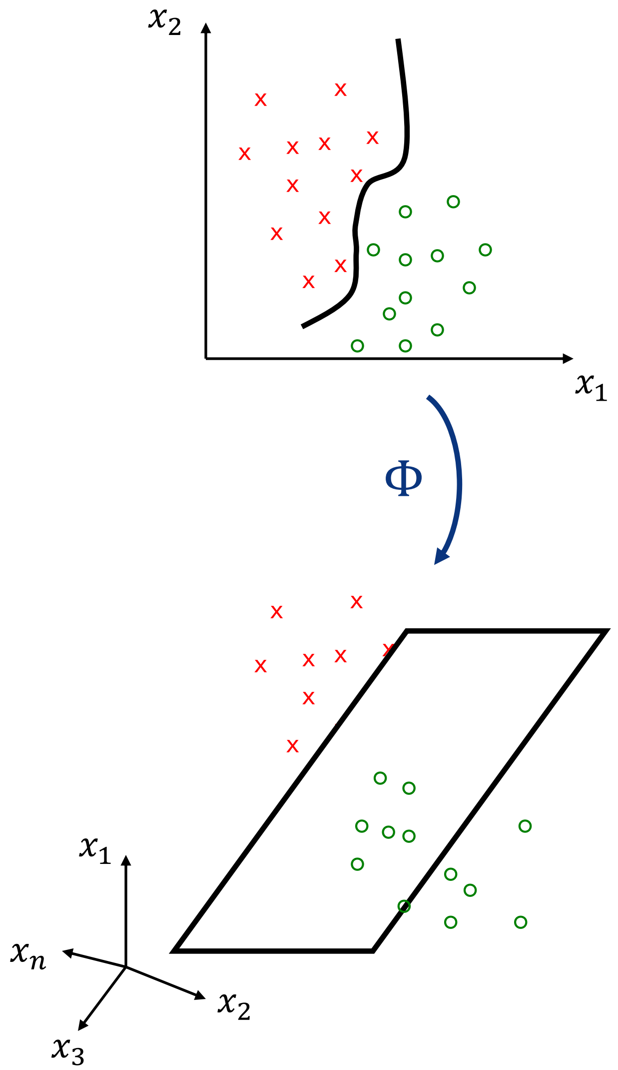
- 271A: it was also showed that it is optimal for
  - two–Gaussian classes
  - equal class probability and covariance
- but, clearly, will **not** work even for only slightly more general Gaussian cases



- Q: what are possible **solutions** to this problem?

Alternatives

- 1) more complex classifier
  - let's try to avoid this
- 2) transform the space
  - introduce a mapping
    $$\Phi \colon \mathcal{X} \to \mathcal{Z}$$
    such that $\dim(\mathcal{Z}) > \dim(\mathcal{X})$
  - learning a **linear** boundary in $\mathcal{Z}$ is equivalent to learning a **non−linear** boundary in $\mathcal{X}$
  - **how do we do this?**
    - we already mentioned **two** possibilities

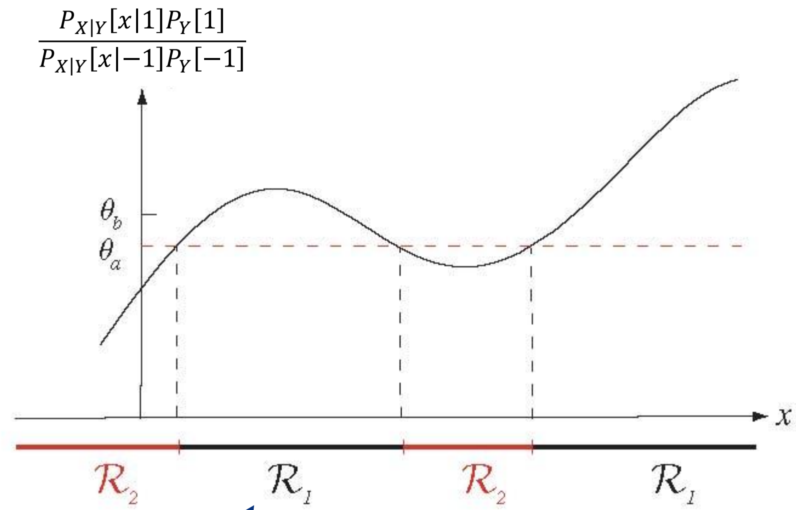


Solution One

- because the BDR is
  - pick $h(\mathbf{x}) = 1$ if

$$\frac{P_{\mathbf{X}|Y}[\mathbf{x}|1]P_Y[1]}{P_{\mathbf{X}|Y}[\mathbf{x}|-1]P_Y[-1]} > 1$$

  - and $h(\mathbf{x}) = -1$, otherwise



- the **mapping**

$$\Phi_{BDR}: \mathbb{R}^d \to \mathbb{R}^{d+1} \text{ with } \Phi_{BDR}(\mathbf{x}) = \left(\mathbf{x}, \frac{P_{\mathbf{X}|Y}[\mathbf{x}|1]P_Y[1]}{P_{\mathbf{X}|Y}[\mathbf{x}|-1]P_Y[-1]}\right)$$

always works, since the hyperplane

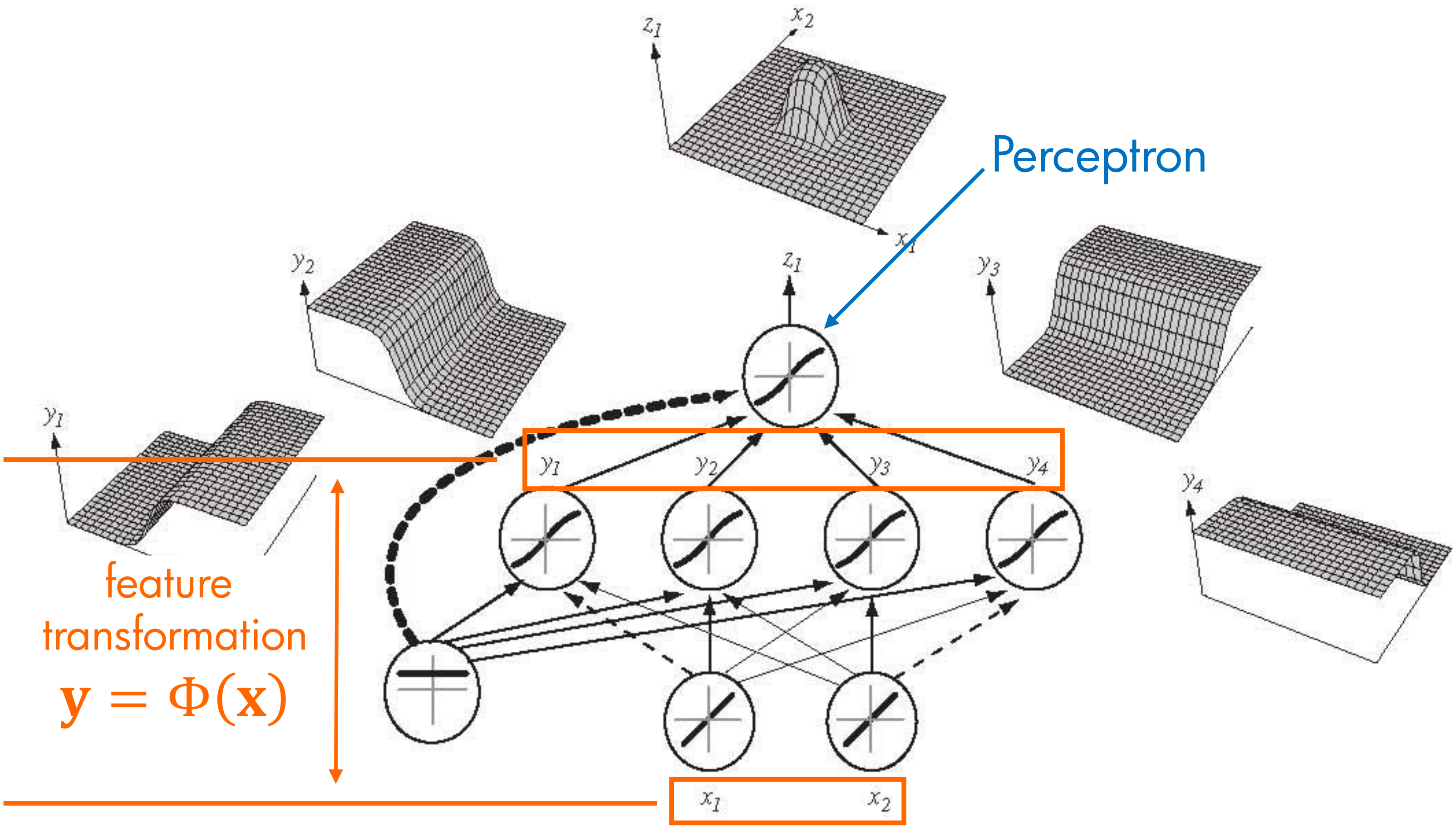
$$\mathbf{w}^T\Phi_{BDR}(\mathbf{x}) + b \quad \text{with} \quad \mathbf{w} = (0, 0, \cdots, 1)^T \text{ and } b = -1$$

optimally separates the classes

Solution Two

- add Perceptron **layers**:

MLP : non−linear feature transformation + linear discriminant



MLP Feature Transformation

- learned by the backpropagation algorithm
- stochastic gradient descent on

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} J(\mathbf{W})$$

$$J(\mathbf{W}) = \sum_i L\big(\mathbf{t}_i, \mathbf{z}(\mathbf{x}_i; \mathbf{W})\big)$$

$$L(\mathbf{t}, \mathbf{z}) = \frac{1}{2} \sum_k [t_k - z_k]^2$$

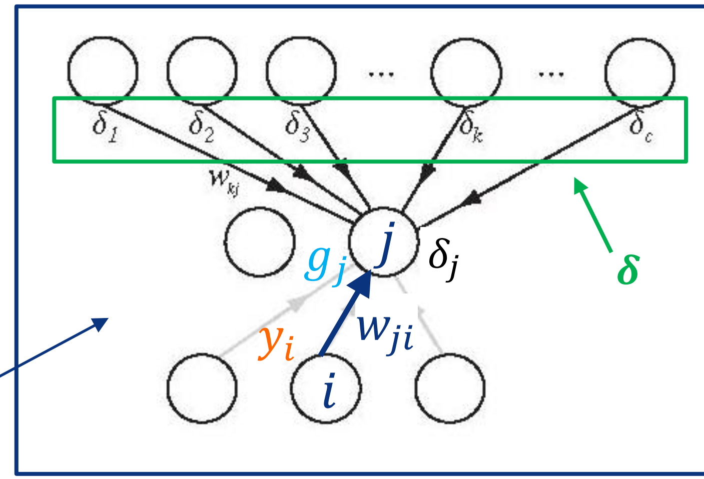
- for any pair (i, j)

$$\frac{\partial L}{\partial w_{ji}} = -\delta_j \, y_i$$

with

$$\delta_j = (t_j - z_j) s'[u_j] \qquad \text{if } j \text{ is output}$$

$$\delta_j = \left[\textstyle\sum_k \delta_k w_{kj}\right] s'[g_j] \qquad \text{if } j \text{ is hidden}$$



the weight updates are

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} - \eta \frac{\partial L}{\partial w_{ji}}$$

Problems

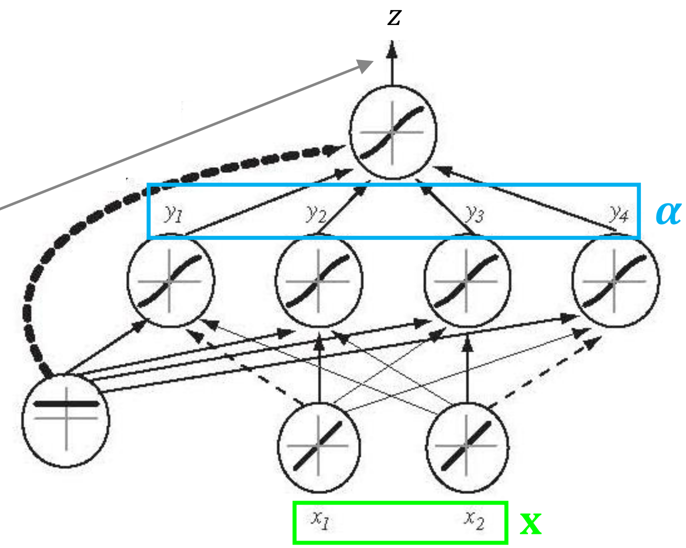
- while **theoretically** **feasible**, these solutions have **various** **problems**

- Φ_{BDR}
  - requires the knowledge of the densities $P_{\mathbf{X}|Y}(\mathbf{x}|i)$
  - density estimation is **quite hard**, specially when d is large

- Φ_{MLP}
  - how **many** hidden units, layers, what configuration?
  - takes long time to search
  - can only be learned when large datasets are available

- this motivated people to think about other solutions
  - let's start by looking at what the NN MLP is doing

MLP and Voting



- consider the **output** of an MLP

$$z = \sigma(\mathbf{w}^T \boldsymbol{\alpha} + b)$$

  where

  - $\sigma(\cdot)$ is the sigmoid
  - $\boldsymbol{\alpha} = (y_1, \ldots, y_k)$ is the vector of outputs of the penultimate layer
  - b is a bias term (optional)

- if you think of the **sigmoid** as an approximation to the sgn function, this can be written in our well–known form

$$z = h(\mathbf{x}) = \mathrm{sgn}[g(\mathbf{x})] \qquad g(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\alpha}(\mathbf{x}) + b$$

- this is like the **Perceptron**, but we no longer work with $\mathbf{x}$
- instead, we use $\boldsymbol{\alpha}(\mathbf{x})$, which is a **non–linear** function of $\mathbf{x}$ (previous NN layers)

MLP and Voting

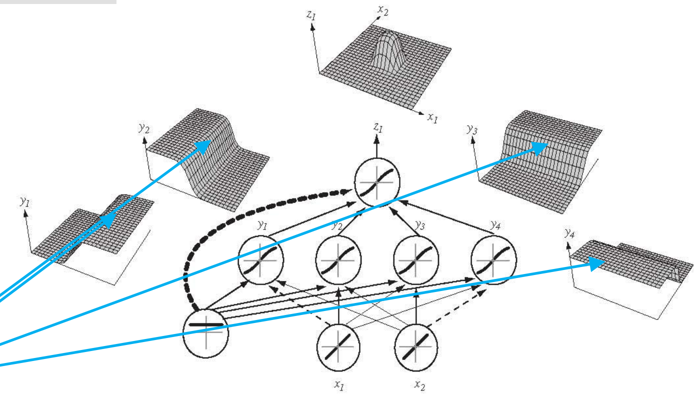
- from

$$z = h(\mathbf{x}) = \mathrm{sgn}[g(\mathbf{x})] \qquad g(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\alpha}(\mathbf{x}) + b$$

- the dot–product can be written as

$$\boxed{\mathbf{w}^T \boldsymbol{\alpha}(\mathbf{x}) = \sum_i w_i \alpha_i(\mathbf{x})}$$

- note that each $\alpha_i(\mathbf{x})$ is itself a classifier



- we can think of the dot–product as a voting mechanism
  - we have a bunch of **weak** classifiers – the functions $\alpha_i(\mathbf{x})$
  - each **votes** for a class label $(1, -1)$
  - the dot–product **sums** the votes (with weights w_i)
  - z is the **majority** rule (1 if there are more positives, -1 otherwise)

Ensemble Learners

- this motivated people to consider **classifiers** of the form

$$\boxed{h(\mathbf{x}) = \mathrm{sgn}[g(\mathbf{x})]} \qquad \boxed{g(\mathbf{x}) = \sum_i w_i \alpha_i(\mathbf{x})}$$

- these are sometimes called **ensemble learners**
- the functions $\alpha_i(\mathbf{x})$ are called **weak learners** (we will later see why)
- note that this is **like the Perceptron**, but the **decision boundary** is no longer linear in $\mathbf{x}$
  - the functions $\alpha_i(\mathbf{x})$ can be very **non–linear**
- the question is:

  how do we learn **the "right" functions** and the **weights** w_i?

Loss Functions and Risk

- as before, we consider a **loss/cost** $L[y, g(\mathbf{x})]$ of **making a prediction** $g(\mathbf{x})$ when the **true value** is y

- given the training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, the goal is to minimize the **empirical risk**

$$\boxed{R_{emp} = \frac{1}{n}\sum_{i=1}^{n} L[y_i, g(\mathbf{x}_i)]} \quad (*)$$

- let's consider the Perceptron again, where

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \qquad h(\mathbf{x}) = \mathrm{sgn}[g(\mathbf{x})]$$

and $E = \{\mathbf{x}_i | y_i(\mathbf{w}^T\mathbf{x}_i + b) \le 0\}$ is the set of errors of $g(\mathbf{x})$

- the cost is

$$\boxed{J_P(\mathbf{w}, b) = -\sum_{i|\mathbf{x}_i \in E} y_i(\mathbf{w}^T\mathbf{x}_i + b) = -\sum_{i|y_i g(\mathbf{x}_i) \le 0} y_i g(\mathbf{x}_i) = \sum_i \phi[y_i g(\mathbf{x}_i)]}$$

- hence, $J_p(\mathbf{w}, b)$ can be written as $(*)$ for

$$\boxed{L[y, g(\mathbf{x})] = \phi\big(y g(\mathbf{x})\big)}$$

$$\boxed{\phi(v) = \max(-v, 0)}$$

Loss Functions and Risk

- in **summary**, the <u>Perceptron</u> learns a classifier

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \qquad h(\mathbf{x}) = \operatorname{sgn}[g(\mathbf{x})]$$

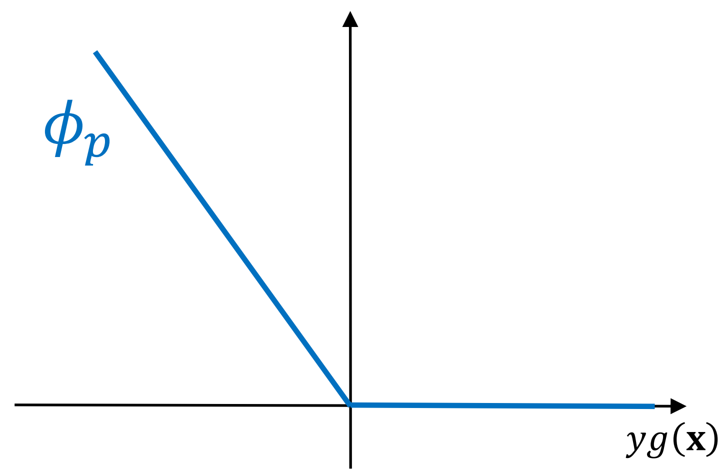
- by minimizing the empirical risk

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} L[y_i, g(\mathbf{x}_i)]$$

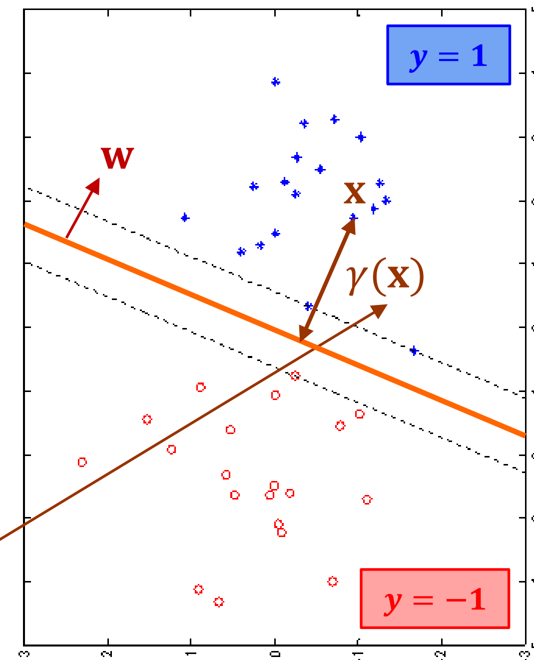
defined by the loss

$$L[y, g(\mathbf{x})] = \phi\big(yg(\mathbf{x})\big)$$

$$\phi(v) = \max(-v, 0)$$



Loss Functions and Risk



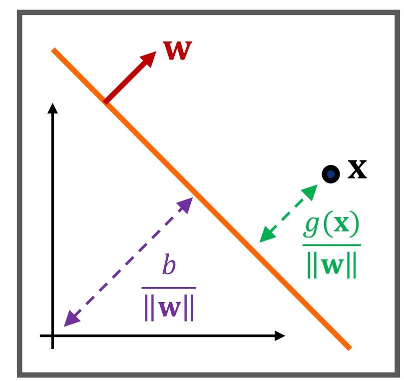
- if we take a closer look at the loss

$$\boxed{L[y, g(\mathbf{x})] = \phi\big(yg(\mathbf{x})\big)}$$

- we note that

$$yg(\mathbf{x}) = y(\mathbf{w}^T\mathbf{x} + b) = \gamma(\mathbf{x})$$

which is what we called the **margin** of **example $\mathbf{x}$**



- it is the signed distance from the plane to $\mathbf{x}$ (up to the constant $||\mathbf{w}||$)

- more generally
  - $yg(\mathbf{x})$ is the **margin** for a generic decision function $g(\mathbf{x})$
  - a loss $\phi\big(yg(\mathbf{x})\big) = \phi\big(\gamma(\mathbf{x})\big)$ is called a $\boxed{\text{margin loss}}$

Loss Functions and Risk

- Recall: given a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, the necessary and sufficient condition to have **zero empirical risk** is

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0, \forall i$$

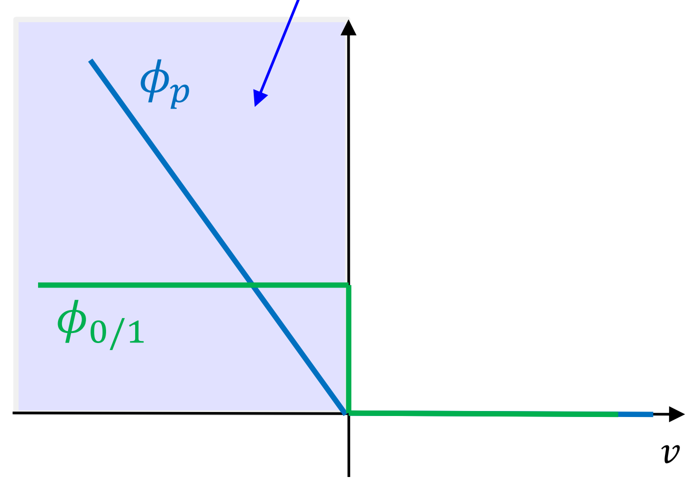
- hence, for **margin losses** $L[y, g(\mathbf{x})] = \phi\big(yg(\mathbf{x})\big)$, the error penalty is determined by the behavior of the loss for **negative arguments**

- note that we have seen some of these losses **already**
  - the 0/1 loss, for which the BDR is optimal

$$\phi_{0/1}(v) = \begin{cases} 0, & v \geq 0 \\ 1, & v < 0 \end{cases}$$

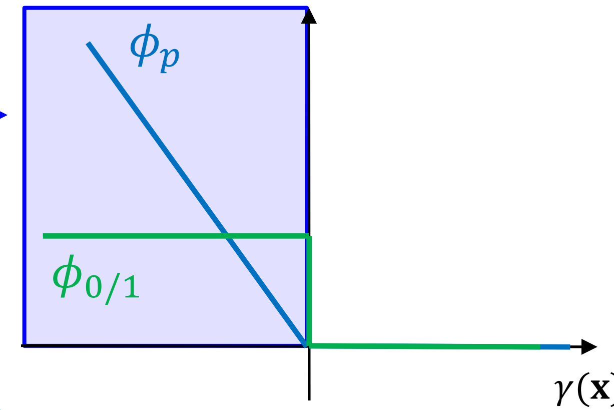
  - the Perceptron loss

$$\phi_P(v) = \begin{cases} 0, & v \geq 0 \\ -v, & v < 0 \end{cases}$$

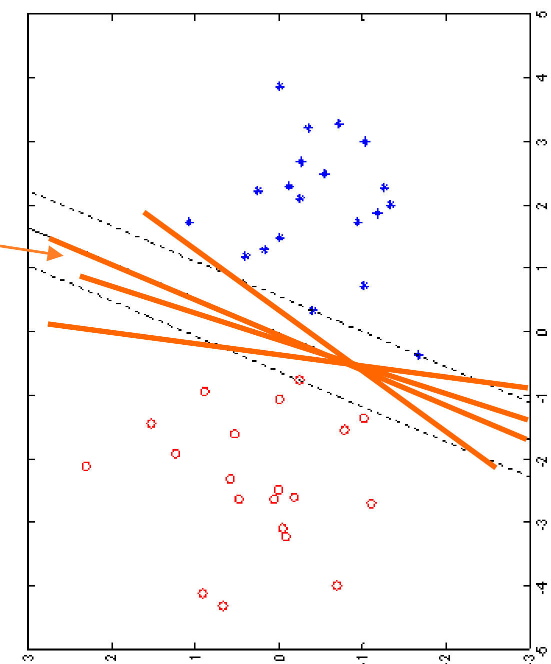


Loss Functions and Risk

- these losses **penalize** "negative margins" or "errors"



- Q: is this good enough?
- think about a problem that is linearly separable
  - if there is one plane that separates the data, there are many planes that separate the data
  - these losses do not favor any of them because they **all** have zero training errors



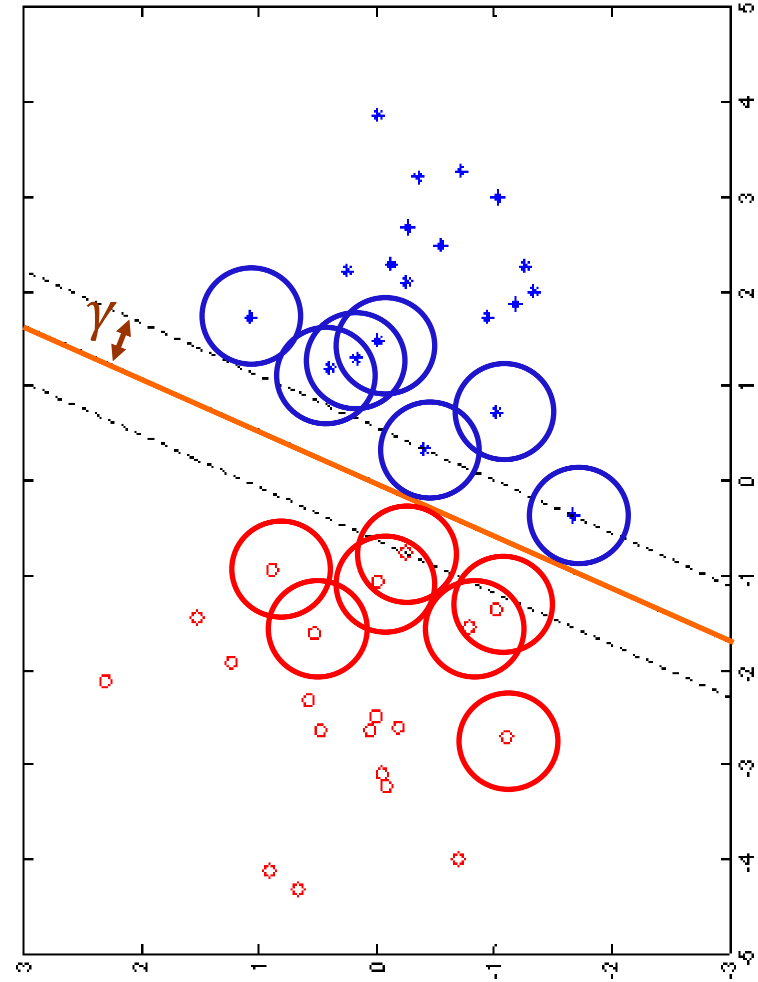
- however, we saw that, for the Perceptron, the margin of the plane determines the complexity of learning
  - convergence in less than $(k/\gamma)^2$ iterations

Maximizing the Margin

Recall:
the margin of the plane is
the margin of the closest point

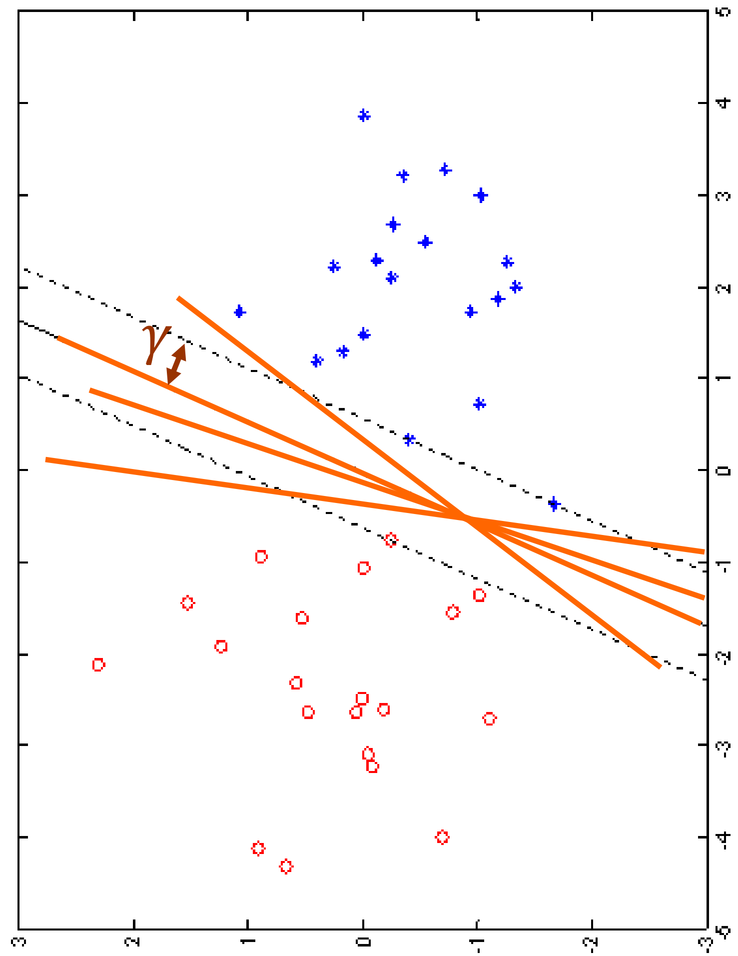
- **what if we favor the plane of <u>maximum</u> margin?**

- <u>Intuitively</u>, this is a <u>good idea</u>
  - think of **each point** in the training set as a <u>sample</u> **from a probability density** centered on it
  - if we draw <u>another</u> sample, we will <u>not</u> get the same points
  - each point is really a pdf with a certain **variance**
  - if we leave a margin of γ on the training set, we are safe against this **uncertainty** (as long as the support of the pdf is smaller than γ)
  - the <u>larger</u> the γ, the **more <u>robust</u> the classifier!**



Maximizing the Margin

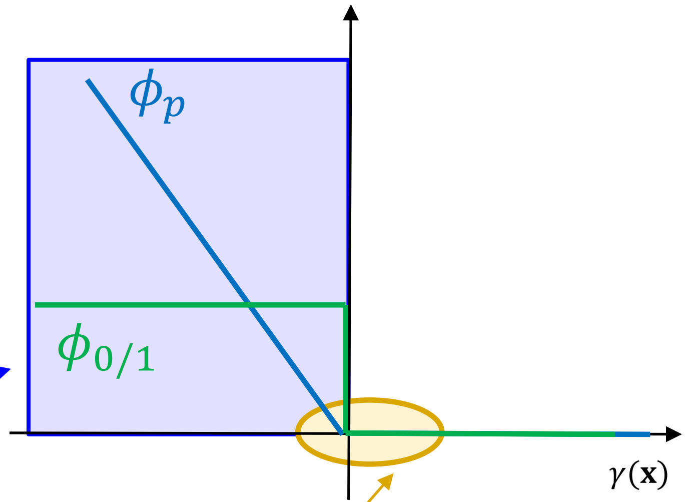
- or:
  - think of the plane as an **uncertain estimate** because it is learned from a sample drawn at random
  - since the sample changes from draw to draw, the plane parameters are random variables of **non−zero variance**
  - instead of a single plane we have a **probability distribution** over planes
  - the larger the margin γ, the larger the number of planes that will **not** originate errors
  - the larger the γ, the larger the variance allowed on the plane parameter estimates!



Loss Functions and Risk

- **in summary**, we want a margin loss

$$L[y, g(\mathbf{x})] = \phi\big(yg(\mathbf{x})\big) = \phi(\gamma(\mathbf{x}))$$



- that
  - besides penalizing errors
  - encourages large margins if there is no error
- this can be achieved by introducing a penalty for small positive margins
- this is trivial to do if the loss $\phi(\cdot)$ is monotonically decreasing
- it suffices that

$$\lim_{v\to\infty} \phi(v) = 0 \quad \text{and} \quad \phi(v) > 0$$

- in this case, the loss is called margin enforcing

Exponential Loss

- boosting (AdaBoost) relies on one such margin enforcing loss
- the exponential loss

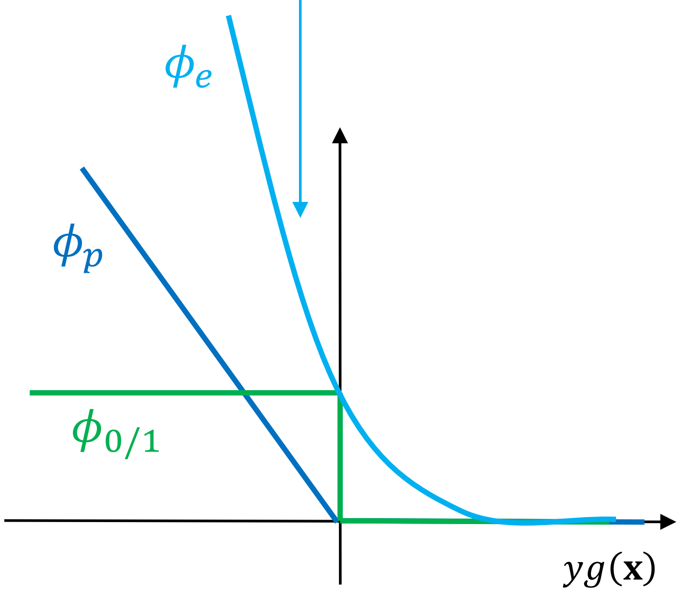
$$L[y, g(\mathbf{x})] = \phi_e\big(yg(\mathbf{x})\big) = \exp\big(-yg(\mathbf{x})\big)$$

and minimizes the empirical risk

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} L[y_i, g(\mathbf{x}_i)]$$

of an ensemble learner

$$g(\mathbf{x}) = \sum_{i=1}^{n} w_i \alpha_i(\mathbf{x})$$



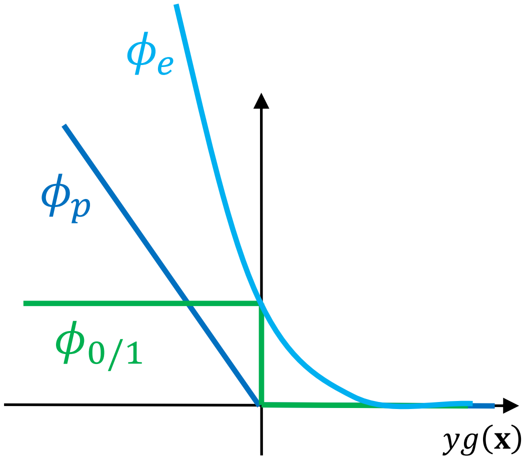
Boosting

- in summary, the <u>goal</u> is to find the ensemble learner

$$g(\mathbf{x}) = \sum_{i=1}^{n} w_i \alpha_i(\mathbf{x})$$

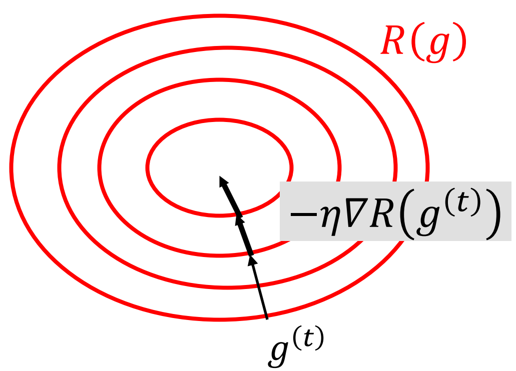
that minimizes the risk

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]$$



- to see <u>how</u> to do this, let's go **back to the Perceptron again**
- we minimize the risk by **gradient descent**

> - pick initial estimate $g^{(0)}$
> - follow the negative gradient
>
> $$g^{(t+1)} = g^{(t)} - \eta \nabla R_{emp}\left[g^{(t)}\right]$$

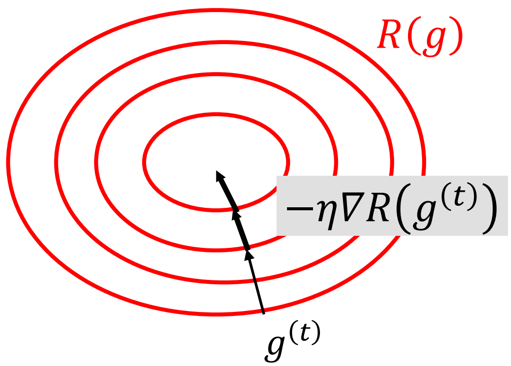


Boosting

- for the **Perceptron**, the problem was **easier** because

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

and the gradient was <u>simply</u> the gradient with respect to the parameters **w** and b



- for **boosting**,

$$\boxed{g(\mathbf{x}) = \sum_{i=1}^{n} w_i \alpha_i(\mathbf{x})}$$

is a <u>combination of functions</u>

- note, however, that <u>**if we can compute the gradient**</u>, then (<u>**assuming**</u> that **we can pick a different step–size η per iteration**)

$$\boxed{g^{(t+1)} = g^{(t)} - \eta^{(t)} \nabla R_{emp}\left[g^{(t)}\right]}$$

results in

Boosting

a g **learned** after $t+1$ iterations that is given by

assuming that we can
- compute the gradient
- pick a different step−size η per iteration

$$\boxed{g^{(t+1)}} = g^{(t)} - \eta^{(t)} \nabla R_{emp}\left[g^{(t)}\right]$$

$$= g^{(t-1)} - \eta^{(t-1)} \nabla R_{emp}\left[g^{(t-1)}\right] - \eta^{(t)} \nabla R_{emp}\left[g^{(t)}\right]$$

$$= \cdots$$

$$= \boxed{-\sum_{i=1}^{n} \eta^{(i)} \nabla R_{emp}\left[g^{(i)}\right]}$$

(where we have assumed that $g^{(0)} = 0$)

- note that this is our **ensemble learner**

$$g(\mathbf{x}) = \sum_{i=1}^{n} w_i \alpha_i(\mathbf{x})$$

if we make the **equivalences**

$$\boxed{\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right]} \qquad \boxed{w_t = \eta^{(t)}}$$

Boosting

$$g(\mathbf{x}) = \sum_{i=1}^{n} w_i \alpha_i(\mathbf{x})$$

- in summary, we can **learn $g(\mathbf{x})$** with the following procedure
  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}[g^{(t)}]$ is decreasing
    - compute the gradient

      $$\alpha_t = -\nabla R_{emp}[g^{(t)}]$$

    - compute the step−size

      $$w_t = \eta^{(t)}$$

    - update the learned function

      $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

NOTE:
at this point, we are **assuming** that we
- can **compute the gradient**
- pick a **different step−size** η **per iteration**

we still need to see how to do these

- this is the **essence of boosting**

Recall: Functions of Functions

- let's start by looking at how to determine $\boxed{\nabla R_{emp}[g^{(t)}]}$
- this is another example of a functional derivative

- consider the vector $w = (w_1, \dots, w_n)$ and **increase** n **until infinity** → this is an infinite−dimensional vector
- note that we can write the vector elements as $w(x), x \in \{1, \dots, \infty\}$
- the next step is to **make** x **continuous**, i.e. consider the infinite−dimensional vector of elements $w(x), x \in \mathbb{R}$
- we call this a function, but there is no fundamental difference
  - in fact, if you define function addition and scalar−function multiplication in the standard way, the space of functions is a vector space
- we can then define the inner−product between functions as

$$\langle w, u \rangle = \int w(x)u(x)\,dx$$

- which generalizes the standard dot−product

$$\langle w, u \rangle = \sum_x w_x u_x$$

Recall: Functional Derivative

- consider now a function of a function $F[w(x)]$
- the **directional derivative** of $F[w(x)]$ at point $w(x)$, along (function) direction u is

$$D_u F[w(x)] = \lim_{h\to 0} \frac{F[w(x) + hu(x)] - F[w(x)]}{h}$$

  - this **measures** how much the **function** F **grows** if we give an infinitesimal step along the direction defined by function u
  - note that we should think of the function $u(x)$ as a **vector**, not as "the value of u at point x"

- there is an **alternative** definition,

$$\boxed{D_u F[w(x)]} = \left[\lim_{h\to 0} \frac{F[w(x) + (\varepsilon + h)u(x)] - F[w(x) + \varepsilon u(x)]}{h}\right]_{\varepsilon=0}$$

$$= \boxed{\left[\frac{d}{d\varepsilon} F[w(x) + \varepsilon u(x)]\right]_{\varepsilon=0}}$$

and this definition only requires computing a **scalar derivative**

Boosting

$$D_u F[w(x)] = \left[\frac{d}{d\varepsilon} F[w(x) + \varepsilon u(x)]\right]_{\varepsilon=0}$$

- in the case of boosting, instead of $\boxed{F[w(\mathbf{x})]}$, we have $R_{emp}[g^{(t)}(\mathbf{x})]$
- the **directional derivative** of $R_{emp}[g^{(t)}(\mathbf{x})]$ at point $g^{(t)}(\mathbf{x})$, along the direction $u(\mathbf{x})$, is

$$D_u R_{emp}[g^{(t)}(\mathbf{x})] = \left[\frac{d}{d\varepsilon} R_{emp}[\, g^{(t)}(\mathbf{x}) + \varepsilon\, u(\mathbf{x})]\right]_{\varepsilon=0}$$

- using the risk of boosting

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]$$

- we obtain

$$\begin{aligned}
D_u R_{emp}[g^{(t)}(\mathbf{x})] &= \frac{1}{n}\sum_i \left[\frac{d}{d\varepsilon}\exp\{-y_i[\, g^{(t)}(\mathbf{x}_i) + \varepsilon u(\mathbf{x}_i)]\}\right]_{\varepsilon=0} \\
&= \frac{1}{n}\sum_i \left[\frac{d}{d\varepsilon}\{-y_i[\, g^{(t)}(\mathbf{x}_i) + \varepsilon u(\mathbf{x}_i)]\} \exp\{-y_i[\, g^{(t)}(\mathbf{x}_i) + \varepsilon u(\mathbf{x}_i)]\}\right]_{\varepsilon=0} \\
&= \frac{1}{n}\sum_i [-y_i u(\mathbf{x}_i)] \exp[-y_i\, g^{(t)}(\mathbf{x}_i)]
\end{aligned}$$

while $R_{emp}(g^{(t)})$ is decreasing

- compute the negative gradient

$$\alpha_t = -\nabla R_{emp}[g^{(t)}]$$

- compute the step size

$$w_t = \eta^{(t)}$$

- update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

Boosting

- hence, the derivative of the risk along the direction u is

$$D_u R_{emp}[g^{(t)}(\mathbf{x})] = -\frac{1}{n}\sum_i y_i u(\mathbf{x}_i) \exp[-y_i\, g^{(t)}(\mathbf{x}_i)]$$

and the gradient is simply the direction of **largest derivative**

$$\boxed{\begin{aligned}\nabla R_{emp}[g^{(t)}(\mathbf{x})] &= \arg\max_u \left\{-\frac{1}{n}\sum_i y_i u(\mathbf{x}_i) \exp[-y_i\, g^{(t)}(\mathbf{x}_i)]\right\} \\ &= \arg\min_u \sum_i y_i u(\mathbf{x}_i) \exp[-y_i\, g^{(t)}(\mathbf{x}_i)]\end{aligned}}$$

- note that the term

$$\boxed{\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)]}$$

does not depend on the direction u, just on the classifier **already available** at iteration t

- $\boxed{\varpi_i}$ can be seen as the **weight of example $\mathbf{x}_i$**

Boosting

$$\nabla R_{emp}[g^{(t)}(\mathbf{x})] = \arg\min_u \sum_i y_i u(\mathbf{x}_i) \exp[-y_i\, g^{(t)}(\mathbf{x}_i)]$$

- we can write this as

$$\boxed{\nabla R_{emp}[g^{(t)}(\mathbf{x})] = \arg\min_u \sum_i y_i u(\mathbf{x}_i) \varpi_i}$$

with

$$\boxed{\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i}$$

- in general, we do **not** optimize **over all set of possible functions**

- instead, we define a family U of functions and optimize over the elements of U

$$\nabla R_{emp}[g^{(t)}(\mathbf{x})] = \arg\min_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

- U can be **many** things (more on this later)
  - e.g., pick entry k of $\mathbf{x}$ and threshold it, i.e.

$$u(\mathbf{x}_i) = 1, \text{ if } x_{ik} > T \quad \text{and} \quad u(\mathbf{x}_i) = -1, \text{otherwise}$$

Boosting

- this leads to the **algorithm**

  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}\left[g^{(t)}\right]$ is decreasing
    - compute the weights

      $$\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

    - compute the negative gradient

      $$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u\,\in\,U}{\arg\max} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$

    - compute the step−size

      $$w_t = \eta^{(t)}$$

    - update the learned function

      $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

The Step–Size

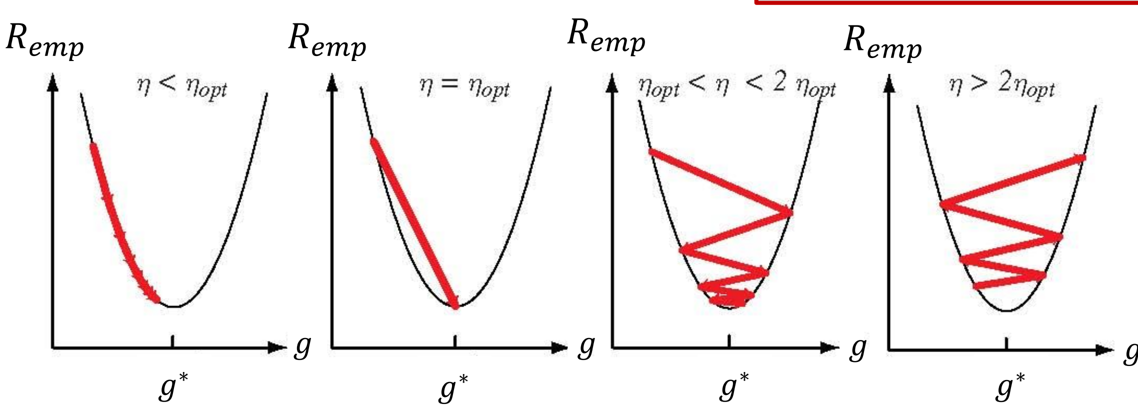
- how do we pick the **step–size** $w_t = \eta^{(t)}$?
  - if too small or too big, we will need various iterations
  - could even diverge

- **line search:**
  - pick $\eta^{(0)}$
  - compute $R_{emp}\left[\eta^{(k)}\right]$

$$R_{emp}(\eta) = \frac{1}{n}\sum_{i=1}^{n} \exp\left[-y_i\left(g^{(t)}(\mathbf{x}_i) + \eta\alpha_t(\mathbf{x}_i)\right)\right]$$

  - repeat for $\eta^{(k+1)} = \beta\,\eta^{(k)}$ (with $\beta < 1$) until you get a minimum of $R_{emp}(\eta)$

$$\boxed{w_t = \arg\min_{\eta} R_{emp}\left[g^{(t)} + \eta\alpha_t\right]}$$



Boosting

- this leads to the final form of the **algorithm**

- initialize $t = 0,\ g^{(t)} = 0$
- while $R_{emp}[g^{(t)}]$ is decreasing
  - compute the weights

$$\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

  - compute the negative gradient

$$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u \in U}{\arg\max} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

  - compute the step−size

$$w_t = \eta^{(t)} = \underset{w}{\arg\min}\ R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

  - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

Boosting: Weight

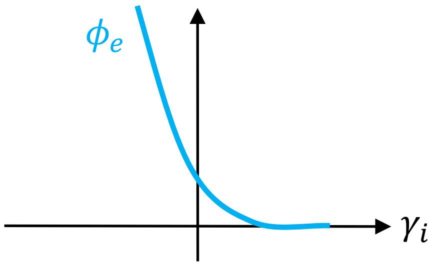
while $R_{emp}[g^{(t)}]$ is decreasing

- compute the weights
  $\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$
- compute the negative gradient
  $\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i)\varpi_i$
- compute the step size
  $w_t = \arg\min_w R_{emp}\left[g^{(t)} + w\alpha_t\right]$
- update the learned function
  $g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$

- we can get some **intuition** by recalling that
  - the risk is

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} \phi_e[y_i g(\mathbf{x}_i)] = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]$$

where $y_i g(\mathbf{x}_i) = \gamma_i$ is the margin of example $\mathbf{x}_i$



- hence, the **boosting weight** ϖ_i of $\mathbf{x}_i$

$$\boxed{\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)] = \phi_e(\gamma_i), \forall i}$$

depends on the margin γ_i of example $\mathbf{x}_i$ under the **current** function $g^{(t)}(\mathbf{x}_i)$

- it is large for the examples of large negative margin $\gamma_i \ll 0$
  (these are examples $\mathbf{x}_i$ with **large error** under the **current** classifier)
- it is approximately zero for the examples of **positive** margin $\gamma_i > 0$
  (these are examples $\mathbf{x}_i$ **correctly** classified under the **current** classifier)

- **in summary**, the weighting mechanism makes **boosting** focus on the **hard examples**

Boosting

emphasizes "hard" examples

- initialize $t = 0,\ g^{(t)} = 0$
- while $R_{emp}\left[g^{(t)}\right]$ is decreasing
  - compute the weights

$$\boxed{\varpi_i = \exp\left[-y_i\ g^{(t)}(\mathbf{x}_i)\right], \forall i}$$

  - compute the negative gradient

$$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u \in U}{\arg\max} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$

  - compute the step−size

$$w_t = \underset{w}{\arg\min}\ R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

  - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

note that this is a **generalization** of the **Perceptron**, which only considers errors, but weights all errors equally

Perceptron Learning

set $k = 0, w_k = 0, b_k = 0$
set $R = \max_i \|\mathbf{x}_i\|$
do {
  for $i = 1{:}n$ {
    if $\boxed{y_i(\mathbf{w}_k{}^T\mathbf{x}_i + b_k) \leq 0}$ then {
      - $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta y_i \mathbf{x}_i$
      - $b_{k+1} = b_k + \eta y_i R^2$
      - $k = k + 1$

    }
  }
} until $y_i(\mathbf{w}^T\mathbf{x}_i + b_k) > 0, \forall i$ (no errors)

Boosting

- what about the gradient step?

  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}\left[g^{(t)}\right]$ is decreasing
    - compute the weights

$$\boxed{\varpi_i \;=\; \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right],\ \forall i}$$

emphasizes "hard" examples

    - compute the negative gradient

$$\boxed{\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \underset{u\,\in\,U}{\arg\max} \sum_i y_i u(\mathbf{x}_i)\varpi_i}$$

    - compute the step–size

$$w_t = \underset{w}{\arg\min}\ R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

    - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

Boosting : Gradient Step

while $R_{emp}[g^{(t)}]$ is decreasing
- compute the weights

  $$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$
- compute the negative gradient

  $$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$
- compute the step size

  $$w_t = \arg\min_w R_{emp}\left[g^{(t)} + w\alpha_t\right]$$
- update the learned function

  $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

- the gradient step
  - consists of selecting the "weak learner" u in U such that

    $$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$
  - note that

    $$y_i\, u(\mathbf{x}_i) = \gamma_i'$$

    is the example margin of $\mathbf{x}_i$ for classification by the weak learner $u(\mathbf{x})$

    and

    $$\sum_i y_i\, u(\mathbf{x}_i)\, \varpi_i = \sum_i \gamma_i' \varpi_i$$

    (up to a scaling constant which makes no difference in the maximization)

    is a weighted average of the margin over all examples $\mathbf{x}_i$, where example $\mathbf{x}_i$ is weighted (ϖ_i) by how hard it is to classify
- **in summary**, boosting picks the weak learner of largest margin on the reweighted training set

Boosting

- what about the gradient step?

  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}\left(g^{(t)}\right)$ is decreasing
    - compute the weights

$$\varpi_i = \exp\left[-y_i\, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

    - compute the negative gradient

$$\alpha_t = -\nabla R_{emp}\left[g^{(t)}\right] = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i)\varpi_i$$

    - compute the step−size

$$w_t = \arg\min_w R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

    - update the learned function

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

emphasizes “hard” examples

picks weak learner of largest weighted margin

$$R_{emp} = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i g(\mathbf{x}_i)]$$

$$\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i$$

Boosting: Step–Size

- what about the **step–size**?

$$w_t = \arg\min_{w} R_{emp}\left[g^{(t)} + w\alpha_t\right]$$

- since

$$\begin{aligned}
R_{emp}\left[g^{(t+1)}\right] = R_{emp}\left[g^{(t)} + w\alpha_t\right] &= \frac{1}{n}\sum_{i=1}^{n} \exp\left[-y_i\left(g^{(t)}(\mathbf{x}_i) + w\alpha_t(\mathbf{x}_i)\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n} \exp\left[-y_i g^{(t)}(\mathbf{x}_i)\right] \exp[-y_i w\alpha_t(\mathbf{x}_i)] \\
&= \frac{1}{n}\sum_{i=1}^{n} \varpi_i \exp[-y_i w\alpha_t(\mathbf{x}_i)]
\end{aligned}$$

and

$$\frac{d}{dw} R_{emp}\left[g^{(t+1)}\right] = \frac{1}{n}\sum_{i=1}^{n} \varpi_i \frac{d}{dw}\exp[-y_i w\alpha_t(\mathbf{x}_i)] = -\frac{1}{n}\sum_{i=1}^{n} \varpi_i\, y_i\alpha_t(\mathbf{x}_i)\exp[-y_i w\alpha_t(\mathbf{x}_i)]$$

- the optimal **step–size** must satisfy the **condition**

$$\boxed{\sum_{i=1}^{n} \varpi_i\, y_i\alpha_t(\mathbf{x}_i)\exp[-y_i w\alpha_t(\mathbf{x}_i)] = 0}$$

Boosting: Step–Size

optimal step–size must satisfy

$$\sum_{i=1}^{n} \varpi_i \, y_i \alpha_t(\mathbf{x}_i) \exp[-y_i w \alpha_t(\mathbf{x}_i)] = 0$$

while $R_{emp}[g^{(t)}]$ is decreasing

- compute the weights
  $$\varpi_i = \exp[-y_i \, g^{(t)}(\mathbf{x}_i)], \forall i$$
- compute the negative gradient
  $$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$
- compute the step size
  $$w_t = \arg\min_w R_{emp}\left[g^{(t)} + w\alpha_t\right]$$
- update the learned function
  $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

$$\boxed{0} = \sum_{i=1}^{n} \varpi_i \, y_i \alpha_t(\mathbf{x}_i) \exp[-y_i w \alpha_t(\mathbf{x}_i)]$$

$$\varpi_i = \exp\left[-y_i \, g^{(t)}(\mathbf{x}_i)\right], \forall i$$

$$= \sum_{i=1}^{n} y_i \alpha_t(\mathbf{x}_i) \exp\left[-y_i \left(g^{(t)}(\mathbf{x}_i) + w_t \alpha_t(\mathbf{x}_i)\right)\right]$$

$$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t \alpha_t(\mathbf{x})$$

$$= \sum_{i=1}^{n} y_i \alpha_t(\mathbf{x}_i) \exp[-y_i \, g^{(t+1)}(\mathbf{x}_i)]$$

$$= \sum_{i=1}^{n} y_i \alpha_t(\mathbf{x}_i) \, \varpi_i^{(t+1)}$$

$\gamma_i{}'$ – example margin of $\mathbf{x}_i$ for the selected (iteration t) weak learner

- this guarantees that the set of weights for the <u>next</u> iteration is "**balanced**"
- under the new weights (iteration $t+1$), the weak learner selected in the <u>current</u> iteration (t) has average margin equal to 0! (is "useless")
- the new weights are such that the weak learner <u>just</u> chosen (iteration t) has **no "confidence"** on the classification of the reweighted dataset ($t+1$)!
- "we **squeezed** <u>**all**</u> the juice out of weak learner selected at t"

AdaBoost

Freund, Yoav; Schapire, Robert; A Decision–Theoretic Generalization of On–Line Learning and an Application to Boosting. *Journal of Computer and System Sciences* 55, 119–139,1997.

- so far, we have considered ensemble classifiers

$$h(\mathbf{x}) = \operatorname{sgn}[g(\mathbf{x})] \qquad g(\mathbf{x}) = \sum_t w_t \alpha_t(\mathbf{x})$$

whose weak learners **can be <u>any</u> functions**

- what if we restrict the weak learners $\alpha_t(\mathbf{x})$ to be **<u>classifiers themselves</u>**?

$$\alpha_t = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i) \varpi_i$$

$$\boxed{\alpha_t(\mathbf{x}) \in \{-1,1\}, \forall\, \mathbf{x}, t}$$

- in this case, the **ensemble rule**

$$\boxed{g(\mathbf{x}) = \sum_t w_t \alpha_t(\mathbf{x}) = \sum_{t|\alpha_t(\mathbf{x})=1} w_t - \sum_{t|\alpha_t(\mathbf{x})=-1} w_t}$$

is a **<u>true</u> voting procedure**

- $\alpha_t(\mathbf{x})$ **votes** for classes +1 or −1 with strength w_t
- the rule "**tallies**" the difference between the **strength** of positive and negative votes

AdaBoost

- and the optimal step−size condition is

$$\alpha_t(\mathbf{x}) \in \{-1,1\}, \forall\, \mathbf{x}, t$$

$$0 = \sum_i \varpi_i\, y_i \alpha_t(\mathbf{x}_i) \exp[-y_i w_t \alpha_t(\mathbf{x}_i)] = \sum_{i|y_i=\alpha_t(\mathbf{x}_i)} \varpi_i e^{-w_t} - \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i e^{w_t}$$

and this holds if

$$e^{-w_t} \sum_{i|y_i=\alpha_t(\mathbf{x}_i)} \varpi_i = e^{w_t} \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i \;\Leftrightarrow\; e^{-w_t}\left(\sum_i \varpi_i - \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i\right) = e^{w_t} \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i$$

$$\Leftrightarrow\; e^{2w_t} = \frac{\sum_i \varpi_i - \sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i} = \frac{1-\varepsilon}{\varepsilon} \quad \text{with} \quad \varepsilon = \frac{\sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

- hence, we have a **closed−form** for the step−size

$$\boxed{w_t = \frac{1}{2}\log\frac{1-\varepsilon}{\varepsilon}} \qquad \varepsilon = \frac{\sum_{i|y_i\neq\alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

AdaBoost

- this is the AdaBoost algorithm
  - initialize $t = 0,\ g^{(t)} = 0$
  - while $R_{emp}[g^{(t)}]$ is decreasing
    - compute the weights

      $$\boxed{\varpi_i = \exp[-y_i\, g^{(t)}(\mathbf{x}_i)], \forall i}$$

    - compute the negative gradient

      $$\boxed{\alpha_t = -\nabla R_{emp}[g^{(t)}] = \arg\max_{u \in U} \sum_i y_i u(\mathbf{x}_i)\varpi_i}$$

    - compute the step−size

      $$\boxed{w_t = \frac{1}{2}\log\frac{1-\varepsilon}{\varepsilon}} \qquad \varepsilon = \frac{\sum_{i|y_i \neq \alpha_t(\mathbf{x}_i)} \varpi_i}{\sum_i \varpi_i}$$

    - update the learned function

      $$g^{(t+1)}(\mathbf{x}) = g^{(t)}(\mathbf{x}) + w_t\alpha_t(\mathbf{x})$$

emphasizes "hard" examples

picks weak learner of largest weighted margin

there is **no** simpler ML algorithm that works!