Project Proposal

- due Tuesday, 2/1 @ 11:59pm
- one page maximum stating:
  - student names
  - problem
  - data you will use
  - draft of proposed solution (can be **updated later**)
  - experiments you will run (can be **updated later**)
  - references (you can use an additional page for this)
- send me pdf by email (mvasconcelos@eng.ucsd.edu) with:
  - **Subject**: Group X Proposal, where X is the group number in this list
  - **cc to all group members**

This assignment is worth 5% of your class grade. **If you submit the proposal in time and make a serious attempt at addressing the bullet points above, you will get full score. I'm not, at this point, grading projects on their merits.** I will look at the proposal and give you some feedback. This will be mostly on issues that I think may become serious obstacles and you need to consider urgently. For example, if I find the problem you propose to be outside the scope of the class, that you may not be able to find data to train the methods you are proposing, etc. Note that if I say "OK", it just means that I see no such problems. It does not mean that you will receive an A just by doing what you proposed. I see these proposals more as a "direction to where the project is going." The projects themselves will be evaluated at the end of the quarter, according to the guidelines published.

1. **Hussain**, Tanvir; **Lewis**, Cameron; **Villamar**, Sandra
2. **Dong**, Meng; **Long**, Jianzhi; **Wen**, Bo; **Zhang**, Haochen
3. **Chen**, Yuzhao; **Li**, Zonghuan; **Song**, Yuze; **Yan**, Ge
4. **Li**, Jiayuan; **Xiao**, Nan; **Yu**, Nancy; **Zhou**, Pei
5. **Li**, Zheng; **Tao**, Jianyu; **Yang**, Fengqi
6. **Bian**, Xintong; **Jiang**, Yufan; **Wu**, Qiyao
7. **Chen**, Yongxing; **Yao**, Yanzhi; **Zhang**, Canwei
8. **Nukala**, Kishore; **Pulleti**, Sai; **Vaidyula**, Srikar
9. **Baluja**, Michael; **Cao**, Fangning; **Huff**, Mikael; **Shen**, Xuyang
10. **Arun**, Aditya; **Long**, Heyang; **Peng**, Haonan
11. **Cowin**, Samuel; **Hanna**, Aaron; **Liao**, Albert; **Mandadi**, Sumega
12. **Jia**, Yichen; **Jiang**, Zhiyun; **Li**, Zhuofan
13. **Dandu**, Murali; **Daru**, Srinivas; **Pamidi**, Sri
14. **He**, Bolin; **Huang**, Yen-Ting; **Wang**, Shi; **Wang**, Tzu-Kao
15. **Chen**, Luobin; **Feng**, Ruining; **Wu**, Ximei; **Xu**, Haoran
16. **Chen**, Rex; **Liang**, Youwei; **Zheng**, Xinran
17. **Aguilar**, Matthew; **Millhiser**, Jacob; **O'Boyle**, John; **Sharpless**, Will
18. **Wang**, Haoyu; **Wang**, Jiawei; **Zhang**, Yuwei
19. **Chen**, Yinbo; **Di**, Zonglin; **Mu**, Jiteng
20. **Chowdhury**, Debalina; **He**, Scott; **Ye**, Yiheng
21. **Lin**, Wei-Ru; **Ru**, Liyang; **Zhang**, Shaohua
22. **Bhavsar**, Shivad; **Blazej**, Christopher; **Bu**, Yinyan; **Liu**, Haozhe
23. **Chen**, Claire; **Hsieh**, Chia-Wei; **Lin**, Jui-Yu; **Tsai**, Ya-Chen
24. **Cheng**, Yu; **Yu**, Zhaowei; **Zaidi**, Ali
25. **Assadi**, Parsa; **Brugere**, Tristan; **Pathak**, Nikhil; **Zou**, Yuxin
26. **Candassamy**, Gokulakrishnan; **Dixit**, Rajeev; **Huang**, Joyce
27. **Kok**, Hong; **Wang**, Jacky; **Yan**, Yijia; **Yuan**, Zhouyuan
28. **Luan**, Zeting; **Yang**, Zheng
29. **Cuawenberghs**, Kalyani; **Mojtahed**, Hamed

ECE 271B: Take-Home Quizzes Guidelines

By submitting your quiz solution, you agree to comply with the following.

1. The quiz should be treated as a **take-home test** and be an **INDIVIDUAL** effort. **NO collaboration is allowed**. The submitted work must be yours and must be original.
2. The work that you turn-in to be your own, using the resources that are available to **all** students in the class.
3. You are not allowed to consult or use resources provided by tutors, previous students in the class, or any websites that provide solutions or help in solving assignments and exams.
4. You will not upload your solutions or any other course materials to any websites or in some other way distribute them outside the class.
5. 0 points will be assigned to any problem that seems to violate these rules and, if recurrent, the incident(s) will be reported to the Academic Integrity Office.

With respect with quiz logistics, you should do the following.

1. Quizzes should be submit in PDF format on Gradescope by **11:59 pm of the due date**. Late submissions will be accepted within 24 hours, but will incur a 20% penalty. After that, there will be no credit.
2. If there are issues that need clarification, feel free to ask on Piazza. However, make sure **not to give the solutions away**. General questions that are not specifically about the problems can, of course, be discussed openly. It follows that if you can frame your question about the problem more generally, you will get a lot more feedback. In general, this also applies to the TAs office hours. If you are stuck in a problem, feel free to go see the TAs. However, TAs will not solve the problem for you. Make sure to ask the question more generally.
3. **Start early** because some problems might need non-trivial amounts of computer time.
4. Unless instructed otherwise, you have to **write all the code** (no packages allowed). If in doubt, ask on Piazza.
5. **All code used to solve the computer problems must be submitted with your quiz**. While we will not be grading code, the TA might need to check it up. If the code is not submitted, 0 points will be assigned to the computer problem.
6. Any request of **quiz regrading** must be submitted on Gradescope **within one week** after the release of the respective graded quiz.
7. Be considerate of the TAs that will be grading your quiz by **submitting a readable PDF document**. Be aware that there is no obligation on the part of the TAs to put effort into deciphering quizzes beyond what is reasonably expected. Typical problems for handwritten documents are: 1) poor handwriting; 2) student writes on both sides of the page and ink bleeds from the background; 3) documents "scanned" by a taking a picture, where there are issues of camera focus or perspective effects that compromise reading; 4) a PDF that is compiled with pages or images upside-down, out of order, or with a skewed perspective. These are issues that severely affect the ability of the TAs to do their job and can be easily avoided with some minimal amount of planning. Now that you are made aware of them, it should be fairly trivial to avoid them. If the TAs are faced with these issues, they can choose not to grade the problem. I give them that discretion.

Quiz #1 due **today** @ 11:59pm

Quiz #2 posted on Canvas

Due date: **Tuesday, 2/8**

ECE 271B – Winter 2022

The Perceptron

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

Manuela Vasconcelos

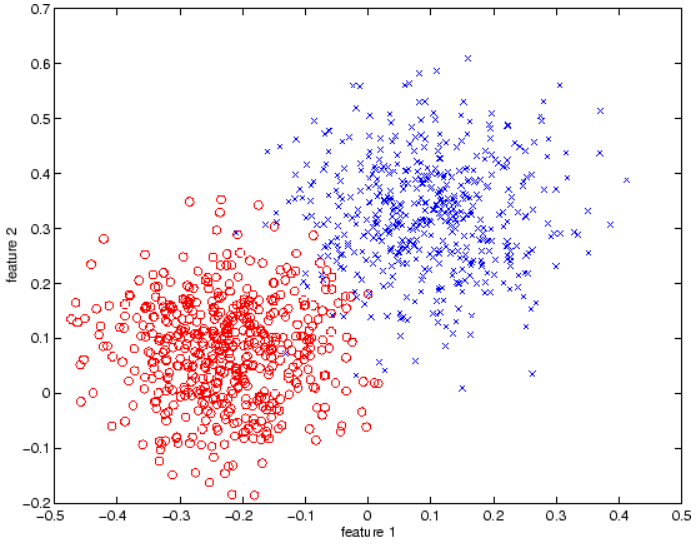
ECE Department, UCSD

Plan for Today

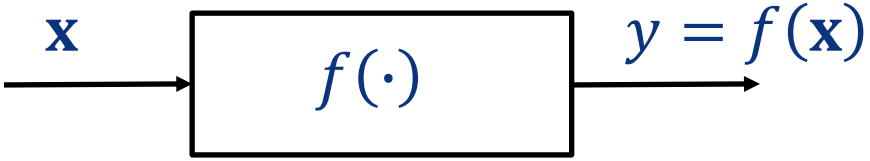
- so far, we have discussed linear discriminants
- today, we start studying how they can be used for classification
- introduce earliest algorithm for discriminant classification, known as the Perceptron
  - cost function
  - relations to gradient descent
  - theoretical convergence analysis
  - insight on the importance of the margin for classification
- this will lead to Multi–Layer Perceptron (MLP) and neural networks

Classification

- a **classification problem** has two 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: $\mathcal{X} \rightarrow \mathcal{Y}$ such that $h(\mathbf{x}) = f(\mathbf{x}), \forall \mathbf{x}$

Linear Discriminant

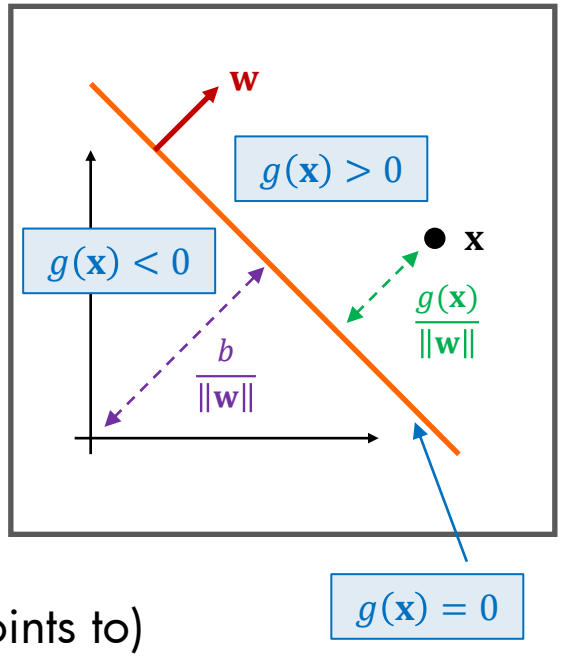
- the classifier implements the linear decision rule

$$h^*(\mathbf{x}) = \begin{cases} 1, & \text{if } g(\mathbf{x}) > 0 \\ 0, & \text{if } g(\mathbf{x}) < 0 \end{cases}$$

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

and has the **properties**

- it divides $\mathcal{X}$ into two "half–planes"
- boundary is the plane with
  - normal $\mathbf{w}$
  - distance to the origin $b/\|\mathbf{w}\|$
- $g(\mathbf{x})/\|\mathbf{w}\|$ is the distance from point $\mathbf{x}$ to the boundary
  - $g(\mathbf{x}) = 0$ for points on the plane
  - $g(\mathbf{x}) > 0$ on the "positive side" (side $\mathbf{w}$ points to)
  - $g(\mathbf{x}) < 0$ on the "negative side"



Linear Discriminant

- using $y \in \{-1,1\}$ instead of $y \in \{0,1\}$, the **decision rule** becomes

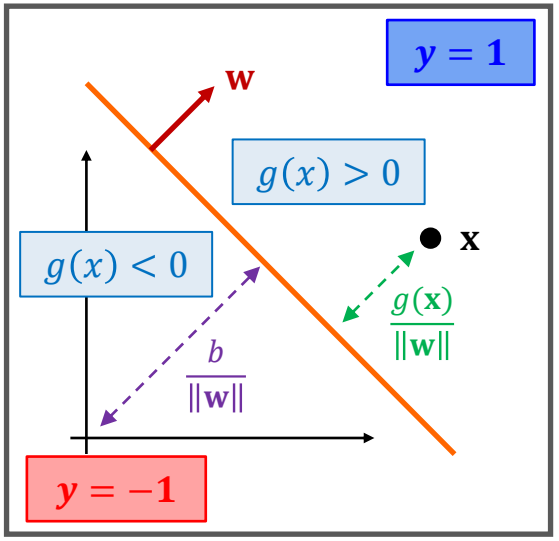
$$h^*(\mathbf{x}) = \begin{cases} 1, & \text{if } g(\mathbf{x}) > 0 \\ -1, & \text{if } g(\mathbf{x}) < 0 \end{cases} = \text{sgn}[g(\mathbf{x})]$$

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

- given a **linearly separable** training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$

- **no errors** if and only if, $\forall i$
  - $y_i = 1$ and $g(\mathbf{x}_i) > 0$ or $y_i = -1$ and $g(\mathbf{x}_i) < 0$
  - i.e. $y_i\, g(\mathbf{x}_i) > 0$

- this allows a **very concise expression** for the situation of "**no training error**" or "**zero empirical risk**"



Learning as Optimization

- necessary and sufficient condition for zero empirical risk

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

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

- this is interesting because it allows the formulation of the learning problem as one of function optimization
  - starting from a random guess for parameters $\mathbf{w}$ and b
  - maximize the reward function

$$\sum_{i=1}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

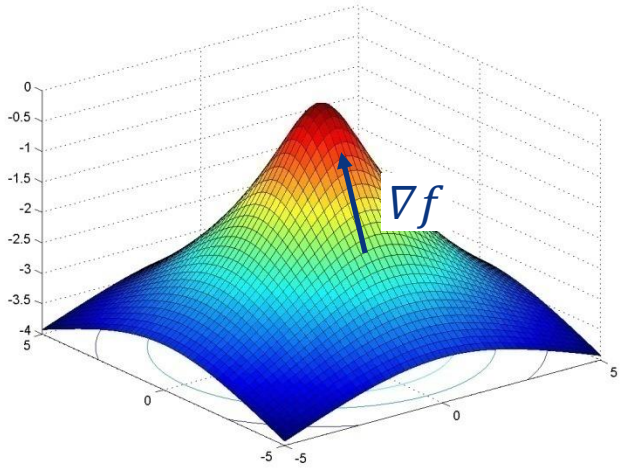
or, equivalently, minimize the cost function

$$J(\mathbf{w}, b) = -\sum_{i=1}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

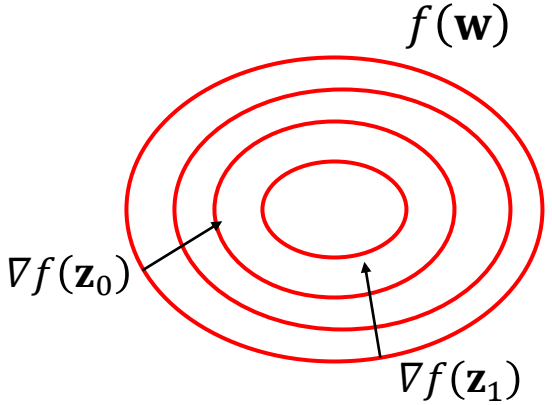
The Gradient

- we have seen that the **gradient** of a function $f(\mathbf{w})$ at $\mathbf{z}$ is

$$\nabla f(\mathbf{z}) = \left(\frac{\partial f}{\partial w_0}(\mathbf{z}), \cdots, \frac{\partial f}{\partial w_{n-1}}(\mathbf{z}) \right)^T$$



- **Theorem:** The gradient points in the direction of maximum growth.



- gradient is
  - the **direction of greatest increase** of $f(\mathbf{w})$ at $\mathbf{z}$
  - **normal to the iso–contours** of $f(\cdot)$

Critical Point Conditions

- we also have seen that

Theorem: Let $f(\mathbf{w})$ be continuously differentiable. $\mathbf{w}^*$ is a **local minimum** of $f(\mathbf{w})$ if and only if

- f has zero gradient at $\mathbf{w}^*$

$$\nabla f(\mathbf{w}^*) = 0$$

- and the Hessian of f at $\mathbf{w}^*$ is positive−semidefinite

$$\mathbf{d}^T \nabla^2 f(\mathbf{w}^*)\mathbf{d} \geq 0, \forall \mathbf{d} \in \mathbb{R}^n$$

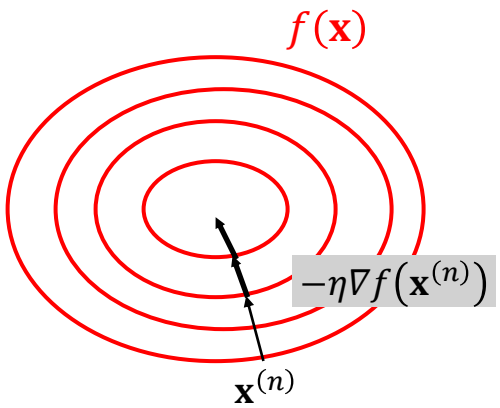
where

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_0^2}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_0 \partial x_{n-1}}(\mathbf{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_{n-1} \partial x_0}(\mathbf{x}) & \cdots & \frac{\partial^2 f}{\partial x_{n-1}^2}(\mathbf{x}) \end{bmatrix}$$

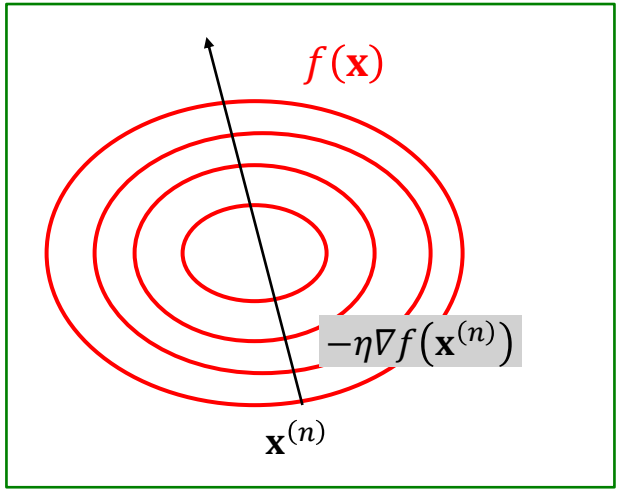
Gradient Descent

- this suggests a **simple** minimization technique
  - pick initial estimate $\mathbf{x}^{(0)}$
  - follow the negative gradient

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \eta \nabla f\left(\mathbf{x}^{(n)}\right)$$



- this is gradient descent
- η is the learning rate and needs to be carefully chosen
  - if η too large, descent may diverge



- many extensions are possible
- main point:
  once framed as optimization, we can (in general) solve it by gradient descent

The Perceptron

Frank Rosenblatt's Perceptron was initially simulated at Cornell Aeronautical Laboratory in 1957

Rosenblatt, Frank; The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, v65, No. 6, pp. 386–408, 1958.

- this was the main insight of **Rosenblatt**, which lead to the **Perceptron**
- the basic idea is to do **gradient descent** on our **cost**

$$J(\mathbf{w}, b) = -\sum_{i=1}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

Recall:
$\mathbf{x}_i$ correctly classified iff

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

- we know that
  - if the training set is **linearly separable**, there is **at least** a pair $(\mathbf{w}, b)$ (i.e. plane) such that $J(\mathbf{w}, b) < 0$
  - **any** minimum that is equal to or better than this will in principle be a **good** classifier
- **Q: can we find one such minimum?**

$$J(\mathbf{w}, b) = -\sum_{i=1}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

Perceptron Learning

- the gradient is straightforward to compute

$$\frac{\partial J}{\partial \mathbf{w}} = -\sum_i y_i \mathbf{x}_i \qquad \frac{\partial J}{\partial b} = -\sum_i y_i$$

  and gradient descent is **trivial**

- there is, however, one **problem**:
  - $J(\mathbf{w}, b)$ is not bounded below
    - if $J(\mathbf{w}, b) = -1$, $J(\lambda\mathbf{w}, \lambda b) = \lambda J(\mathbf{w}, b) = -\lambda$
    - can make $J \to -\infty$ by multiplying $\mathbf{w}$ and b by $\lambda > 0$
    - the minimum is always at $-\infty$ which is quite bad, numerically

- this is really just the **normalization problem** that we already talked about

Rosenblatt's Idea

$$J(\mathbf{w}, b) = -\sum_{i=1}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

$\mathbf{x}_i$ correctly classified iff

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

- restrict attention to the points incorrectly classified (use a cost that only accounts for the errors)
- at each iteration, define set of errors

$$E = \{\mathbf{x}_i \mid y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0\}$$

and make the cost

$$J_P(\mathbf{w}, b) = -\sum_{i|\mathbf{x}_i \in E} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

- note that
  - J_p **cannot** be negative since, in E, all $y_i(\mathbf{w}^T\mathbf{x}_i + b)$ are non−positive
  - if we get to **zero**, we know we have the best possible solution ($E = \emptyset$ or only contains $\mathbf{x}_i$ on the boundary)

$$J_P(\mathbf{w}, b) = -\sum_{i|\mathbf{x}_i \in E}^{n} y_i(\mathbf{w}^T\mathbf{x}_i + b)$$

Perceptron Learning

- learning is **trivial**, just do **gradient descent** on $J_P(\mathbf{w}, b)$

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta \sum_{i|\mathbf{x}_i \in E} y_i \mathbf{x}_i$$

$$b^{(k+1)} = b^{(k)} + \eta \sum_{i|\mathbf{x}_i \in E} y_i$$

Recall:

gradient descent

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - \eta \nabla f(\mathbf{x}^{(n)})$$

$$\frac{\partial J}{\partial \mathbf{w}} = -\sum_i y_i \mathbf{x}_i$$

$$\frac{\partial J}{\partial b} = -\sum_i y_i$$

- this turns out **not** to be very effective if the $\mathcal{D}$ is large
  - loop over the **entire** set E to take a **small** step at the end
- one alternative that frequently is better is **stochastic gradient descent**
  - take the step **immediately** after each example
  - **no** guarantee this is a descent step but, **on average**, you follow the **same** direction after processing entire $\mathcal{D}$
  - very popular in learning, where $\mathcal{D}$ is **usually large**

$E = \{\mathbf{x}_i \mid y_i(\mathbf{w}^T\mathbf{x}_i + b) \le 0\}$

for $\mathbf{x}_i \in E$,

$$\mathbf{w}^{(n+1)} = \mathbf{w}^{(n)} + \eta y_i \mathbf{x}_i$$
$$b^{(n+1)} = b^{(n)} + \eta y_i$$

Perceptron Learning

- the algorithm is as follows:

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

    if $y_i(\mathbf{w}_k{}^T\mathbf{x}_i + b_k) \le 0$ then { ($\mathbf{x}_i \in E$)

      - $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
      - $b_{k+1} = b_k + \eta\, y_i\, R^2$
      - $k = k + 1$

    }

  }

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

note:
we will talk about R shortly

$E = \{\mathbf{x}_i \mid y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0\}$

Perceptron Learning

- does this make sense? consider the example below

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

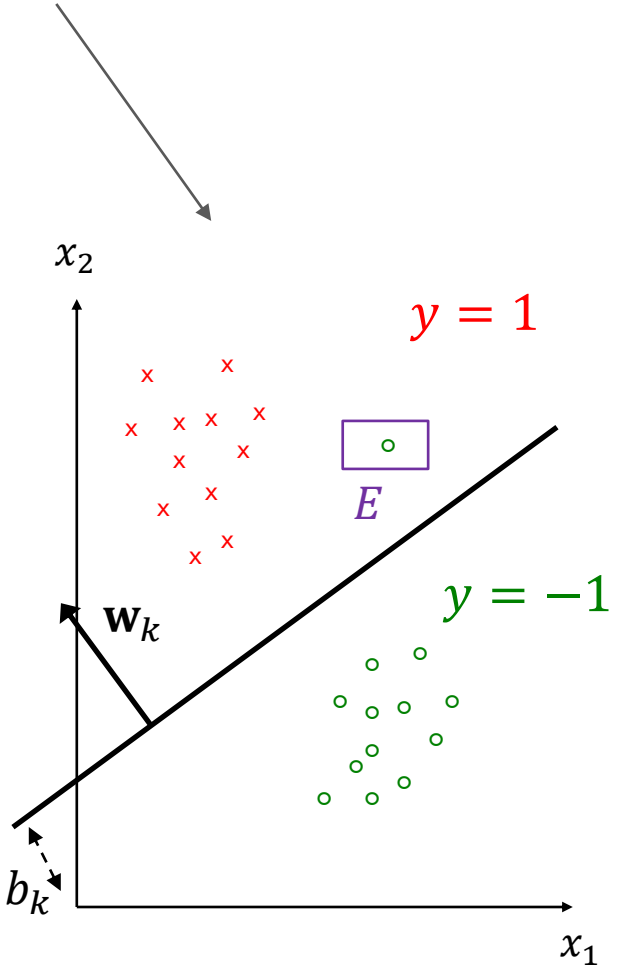
    if $y_i(\mathbf{w}_k{}^T\mathbf{x}_i + b_k) \leq 0$ then { ($\mathbf{x}_i \in E$)

      - $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
      - $b_{k+1} = b_k + \eta\, y_i\, R^2$
      - $k = k + 1$

    }

  }

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



Perceptron Learning

- does this make sense? consider the example below

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

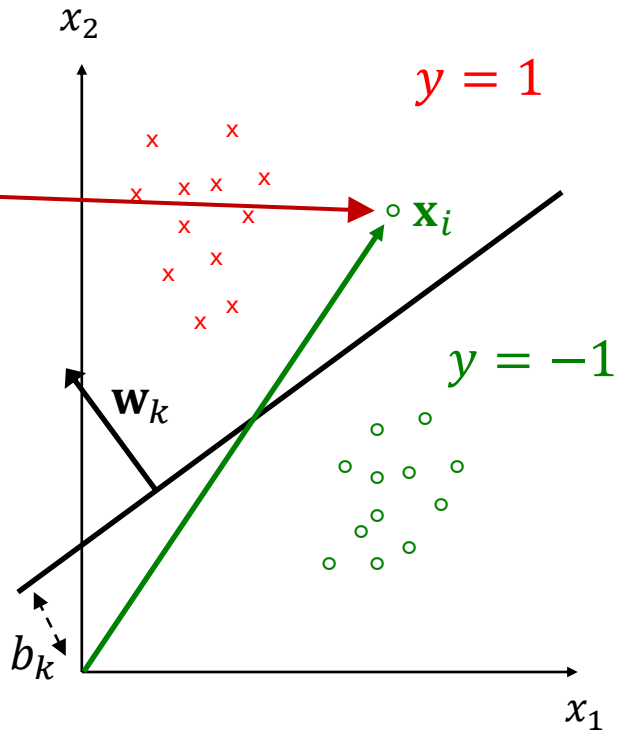
    if $y_i(\mathbf{w}_k{}^T \mathbf{x}_i + b_k) \leq 0$ then {

      - $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
      - $b_{k+1} = b_k + \eta\, y_i\, R^2$
      - $k = k + 1$

    }

  }

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



Perceptron Learning

- does this make sense? consider the example below

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

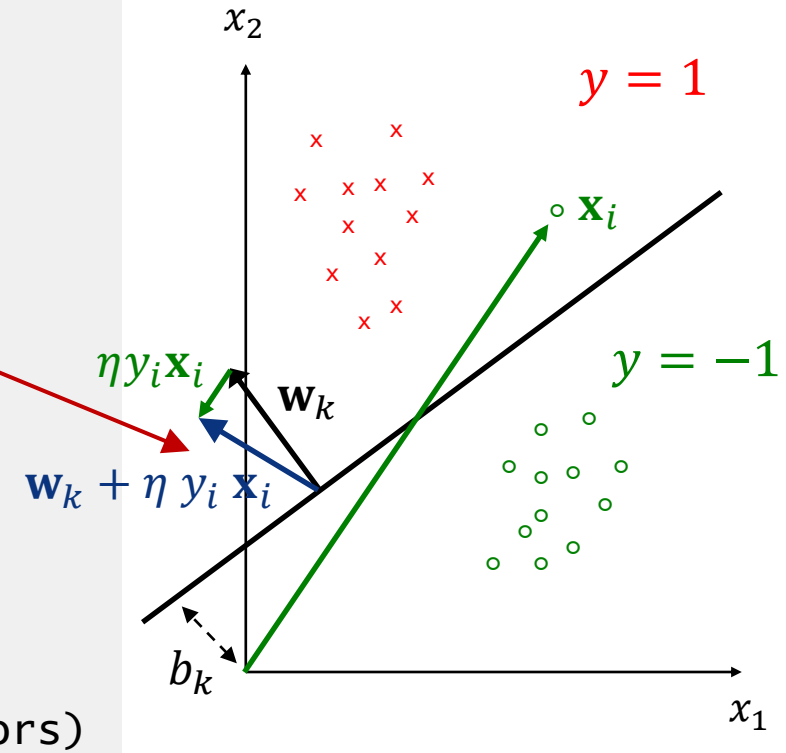
    if $y_i(\mathbf{w}_k{}^T \mathbf{x}_i + b_k) \leq 0$ then {

- $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
- $b_{k+1} = b_k + \eta\, y_i\, R^2$
- $k = k + 1$

    }

  }

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



Perceptron Learning

- does this make sense? consider the example below

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

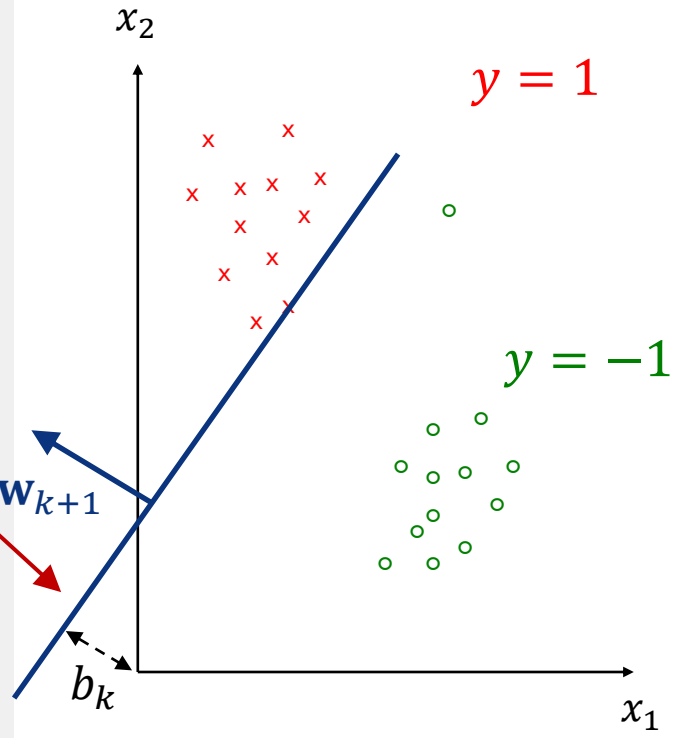
    if $y_i(\mathbf{w}_k{}^T \mathbf{x}_i + b_k) \leq 0$ then {

- $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
- $b_{k+1} = b_k + \eta\, y_i\, R^2$
- $k = k + 1$

    }

  }

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



Perceptron Learning

- does this make sense? consider the example below

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

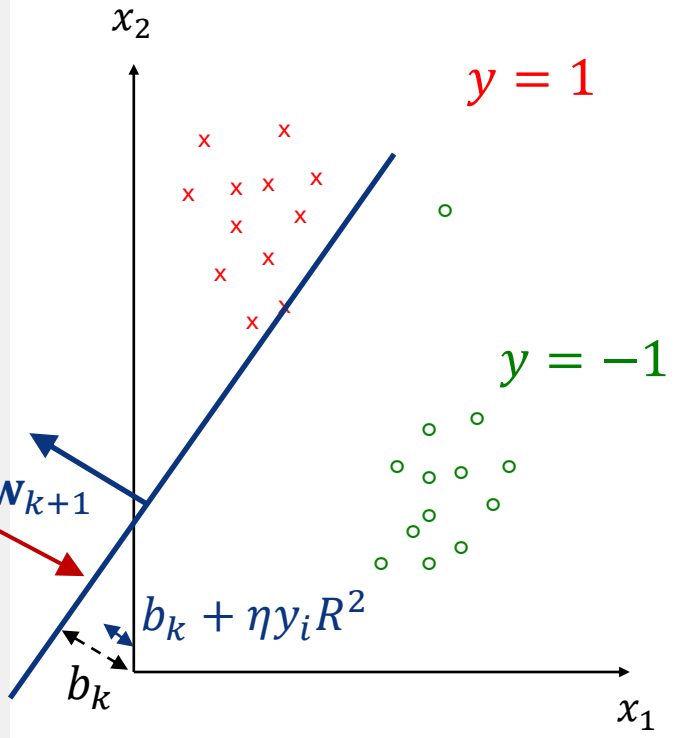
    if $y_i(\mathbf{w}_k{}^T \mathbf{x}_i + b_k) \leq 0$ then {

- $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
- $b_{k+1} = b_k + \eta\, y_i\, R^2$
- $k = k + 1$

    }

  }

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



Perceptron Learning

- does this make sense? consider the example below

set $k = 0, \mathbf{w}_k = 0, b_k = 0$

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

do {

  for $i = 1{:}n$ {

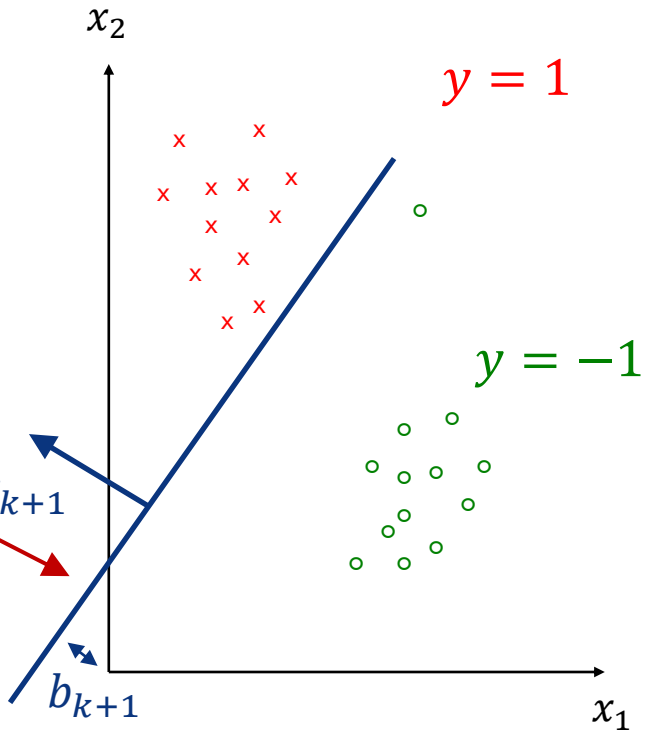
    if $y_i(\mathbf{w}_k{}^T \mathbf{x}_i + b_k) \leq 0$ then {

      - $\mathbf{w}_{k+1} = \mathbf{w}_k + \eta\, y_i\, \mathbf{x}_i$
      - $b_{k+1} = b_k + \eta\, y_i\, R^2$
      - $k = k + 1$

    }

  }

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



Perceptron Learning

- OK, **makes intuitive sense**
- how do we know it will **not** get stuck on a **local minimum?**
- this was Rosenblatt's seminal contribution

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

$$R = \max_i \|\mathbf{x}_i\|. \qquad (*)$$

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

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

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

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

converges in finite time!

Proof

$$R = \max_i \|\mathbf{x}_i\| \quad (*)$$

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

- denote **iteration** by t, assume example processed at iteration $t-1$ is $(\mathbf{x}_i, y_i)$
- for simplicity, use homogeneous coordinates
  - defining

$$\mathbf{z}_i = \begin{bmatrix} \mathbf{x}_i \\ R \end{bmatrix} \qquad \boldsymbol{a}_t = \begin{bmatrix} \mathbf{w}_t \\ b_t/R \end{bmatrix}$$

  - allows the compact notation

$$y_i(\mathbf{w}_{t-1}^T \mathbf{x}_i + b_{t-1}) = y_i \boldsymbol{a}_{t-1}^T \boldsymbol{z}_i$$

- since only **misclassified** points are processed, we have

$E = \{\mathbf{x}_i \mid y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0\}$

$$y_i \boldsymbol{a}_{t-1}^T \boldsymbol{z}_i \leq 0 \quad (***)$$

Proof

$$R = \max_i \|\mathbf{x}_i\| \quad (*)$$

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

$$y_i \boldsymbol{a}_{t-1}^T \mathbf{z}_i \leq 0 \quad (***)$$

- how does $\boldsymbol{a}_t$ evolve?

$$\boldsymbol{a}_t = \begin{bmatrix} \mathbf{w}_t \\ b_t/R \end{bmatrix} = \begin{bmatrix} \mathbf{w}_{t-1} \\ b_{t-1}/R \end{bmatrix} + \eta \begin{bmatrix} y_i \mathbf{x}_i \\ y_i R \end{bmatrix} = \boldsymbol{a}_{t-1} + \eta\, y_i \mathbf{z}_i$$

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta y_i \mathbf{x}_i$$

$$b_t = b_{t-1} + \eta y_i R^2$$

$$\mathbf{z}_i = \begin{bmatrix} \mathbf{x}_i \\ R \end{bmatrix} \quad \boldsymbol{a}_t = \begin{bmatrix} \mathbf{w}_t \\ b_t/R \end{bmatrix}$$

- denoting the **optimal solution** by $\boldsymbol{a}^* = (\mathbf{w}^*, b^*/R)^T$

$$\boldsymbol{a}_t^T \boldsymbol{a}^* = \boldsymbol{a}_{t-1}^T \boldsymbol{a}^* + \eta\, y_i\, \boldsymbol{z}_i^T \boldsymbol{a}^* = \boldsymbol{a}_{t-1}^T \boldsymbol{a}^* + \eta \underbrace{y_i\left(\mathbf{w}^{*T}\mathbf{x}_i + b^*\right)}_{>\gamma}$$

and, from assumption $(**)$,

$$\boldsymbol{a}_t^T \boldsymbol{a}^* > \boldsymbol{a}_{t-1}^T \boldsymbol{a}^* + \eta\, \gamma$$

- solving the recursion

$$\boldsymbol{a}_t^T \boldsymbol{a}^* > \boldsymbol{a}_{t-1}^T \boldsymbol{a}^* + \eta\, \gamma > \boldsymbol{a}_{t-2}^T \boldsymbol{a}^* + 2\, \eta\, \gamma > \cdots > t\, \eta\, \gamma$$

- if the magnitude of $\boldsymbol{a}_t$ is bounded, this implies that $\boldsymbol{a}_t$ is becoming more similar to $\boldsymbol{a}^*$ and thus converging

Proof

$$R = \max_i \|\mathbf{x}_i\| \quad (*)$$

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

$$y_i \boldsymbol{a}_{t-1}^T \mathbf{z}_i \leq 0 \quad (***)$$

- hence, we need to bound the magnitude of a_t
- what is this magnitude?
- since

$$\boldsymbol{a}_t = \boldsymbol{a}_{t-1} + \eta\, y_i\, \boldsymbol{z}_i$$

($y_i = \pm 1$)

- we have

$$\|\boldsymbol{a}_t\|^2 = \boldsymbol{a}_t^T \boldsymbol{a}_t = \|\boldsymbol{a}_{t-1}\|^2 + 2\,\eta\, \underbrace{y_i \boldsymbol{a}_{t-1}^T \mathbf{z}_i}_{\leq 0} + \eta^2 \|\mathbf{z}_i\|^2$$

(≤ 0 from $(***)$)

and,

$$\|\boldsymbol{a}_t\|^2 \leq \|\boldsymbol{a}_{t-1}\|^2 + \eta^2 \|\boldsymbol{z}_i\|^2 \quad \text{from } (***)$$

$$= \|\boldsymbol{a}_{t-1}\|^2 + \eta^2(\|\mathbf{x}_i\|^2 + R^2) \quad \text{from definition of } \mathbf{z}$$

$$\mathbf{z}_i = \begin{bmatrix} \mathbf{x}_i \\ R \end{bmatrix}$$

$$< \|\boldsymbol{a}_{t-1}\|^2 + 2\,\eta^2 R^2 \quad \text{from } (*)$$

- solving the recursion

$$\|\boldsymbol{a}_t\|^2 < 2\,t\,\eta^2 R^2$$

Proof

$$\boldsymbol{a}_t^T \boldsymbol{a}^* > t\,\eta\,\gamma$$
$$\|\boldsymbol{a}_t\|^2 < 2\,t\,\eta^2 R^2$$

- combining the two recursions

$$t\,\eta\,\gamma < \boldsymbol{a}_t^T\,\boldsymbol{a}^* \le \|\boldsymbol{a}_t\|\|\boldsymbol{a}^*\| < \|\boldsymbol{a}^*\|\sqrt{2t}\,\eta\,R$$

- we get

$$t < \frac{2R^2}{\gamma^2}\|\boldsymbol{a}^*\|^2 = \frac{2R^2}{\gamma^2}\left(\|\mathbf{w}^*\|^2 + \frac{b^{*2}}{R^2}\right)$$

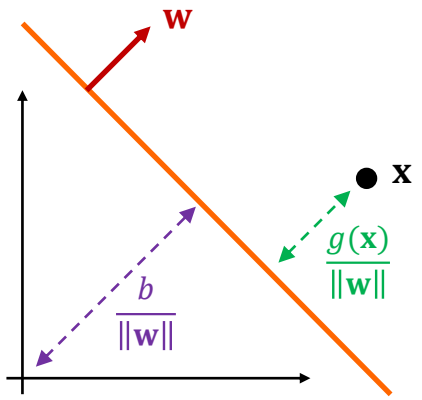
from definition of $\boldsymbol{a}^*$ $\quad \boldsymbol{a}^* = \begin{bmatrix}\mathbf{w}^* \\ b^*/R\end{bmatrix}$

$$= \frac{2R^2}{\gamma^2}\left(1 + \frac{b^{*2}}{R^2}\right)$$

from assumption $\|\mathbf{w}^*\| = 1$

- since $b^*/\|\mathbf{w}^*\| = b^*$ is the distance to the origin, we have $b^* < R = \max_i \|\mathbf{x}_i\|$ and

$$t < \left(\frac{2R}{\gamma}\right)^2 \blacksquare$$



Note

- this is **not** the "standard proof" (e.g. see Duda, Hart, Stork)
- standard proof:
  - regular algorithm (**no** R in update equations)
  - tighter bound $t < (R/\gamma)^2$ (instead of $t < (2R/\gamma)^2$)
- this appears better, but **requires** choosing $\eta = R^2/\gamma$
- which **requires** knowledge of γ, that we do not have until we find $\boldsymbol{a}^*$
- i.e. the proof is non–constructive → cannot design algorithm that way
- the algorithm above just works!
- hence, I like this proof better despite looser bound

Perceptron Learning

- **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

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

- this result was the **start** of learning theory
- for the **first time**, there was a **proof** that a learning machine could actually learn something!

The Margin

- note that $y_i = \pm 1$

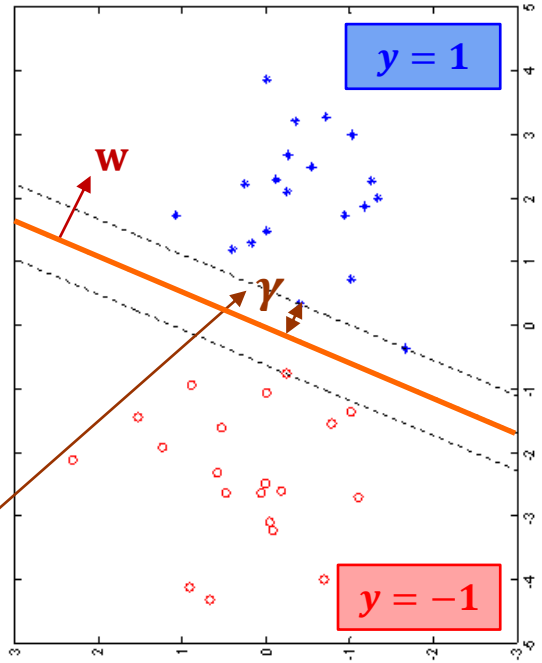
$$\boxed{y_i}\left(\mathbf{w}^{*T}\mathbf{x}_i + b^*\right) \geq \gamma, \forall i$$

will hold when

from assumption $\|\mathbf{w}\| = 1$

$$\gamma = \min_i \frac{|\mathbf{w}^T\mathbf{x}_i + b|}{\|\mathbf{w}\|} = \min_i |\mathbf{w}^T\mathbf{x}_i + b|$$

which is how we defined the **margin**



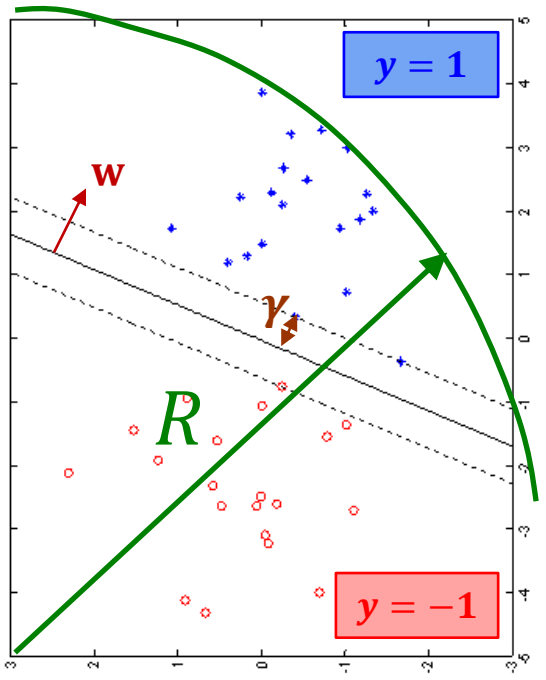
the **smaller** the **margin**, the **longer** the algorithm will take to converge

- this says that the bound on time of convergence is inversely proportional to the margin → $\left(\frac{2R}{\gamma}\right)^2$

- even in this **early** result, the **margin** appears as a measure of the difficulty of the learning problem

The Role of R

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



- scaling the space should not make a difference as to whether the problem is solvable
- R accounts for this
- if the $\mathbf{x}_i$ are re–scaled, both R and γ are re–scaled and the bound

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

remains the same

- once again, just a question of normalization
  - illustrates the fact that the normalization $\|\mathbf{w}\| = 1$ may not be sufficient
  - in this case, we have to introduce R

Some History

- Rosenblatt's result generated a lot of excitement about learning in the 50s
- later, **Minsky** and **Papert** identified **serious** problems with the Perceptron
  - there are **very simple** logic problems that it **cannot** solve
  - more on this on the quiz (Quiz#2 Prob.1)

M. Minsky and S. Papert, Perceptrons: An Introduction to Computational Geometry. *MIT Press, Cambridge, MA, USA,* 1969.

- this killed off the enthusiasm until an old result by Kolmogorov saved the day
- **Theorem:** Any continuous function $g(\mathbf{x})$ defined on $[0,1]^n, n \geq 2$, can be represented in the form

$$g(\mathbf{x}) = \sum_{j=1}^{2n+1} \Gamma_j \left(\sum_{k=1}^{n} \Psi_{jk}(x_k) \right)$$

for properly chosen functions Γ_j and Ψ_{jk}.

Some History

$$g(\mathbf{x}) = \sum_{j=1}^{2n+1} \Gamma_j \left(\sum_{k=1}^{n} \Psi_{jk}(x_k) \right)$$

- noting that the Perceptron can be written as

$$h(\mathbf{x}) = \text{sgn}[\mathbf{w}^T\mathbf{x} + b] = \text{sgn}\left[\sum_{k=1}^{d} w_k x_k + w_0\right]$$

- this looks like having **two Perceptron layers**
  - layer 1: J hyper–planes $\mathbf{w}_j$

$$h_j(\mathbf{x}) = \text{sgn}\left[\sum_{k=1}^{d} w_{jk} x_k + w_{j0}\right], j = 1, \cdots, J$$

  - layer 2: hyper–plane $\mathbf{v}$

$$u(\mathbf{x}) = \text{sgn}\left[\sum_{j=1}^{J} v_j h_j(\mathbf{x}) + v_{j0}\right]$$

$$= \text{sgn}\left[\sum_{j=1}^{J} v_j \, \text{sgn}\left[\sum_{k=1}^{d} w_{jk} x_k + w_{j0}\right] + v_{j0}\right]$$

Some History

- layer 2: hyper–plane $\mathbf{v}$ can be written as

$$u(\mathbf{x}) = \text{sgn}[g(\mathbf{x})] \quad \text{with} \quad g(\mathbf{x}) = \sum_{j=1}^{J} v_j \, \text{sgn}\left[\sum_{k=1}^{d} w_{jk} x_k + w_{j0}\right] + v_{j0}$$

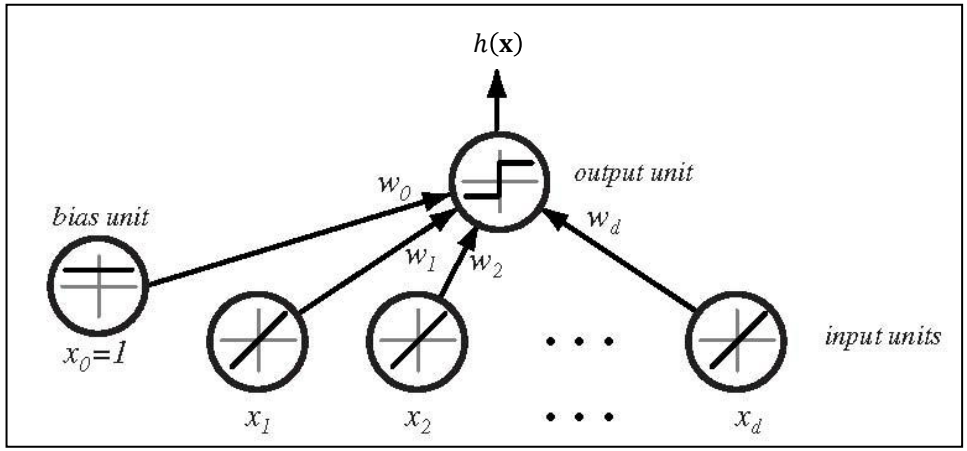
and resembles

$$g(\mathbf{x}) = \sum_{j=1}^{2n+1} \Gamma_j \left(\sum_{k=1}^{n} \Psi_{jk}(x_k) \right)$$

- it suggested the idea that
  - while **one** Perceptron is **not good enough**
  - maybe a **Multi–Layered Perceptron (MLP)** will work
- a lot of work on MLPs ensued under the name of **neural networks**
- eventually, it was shown that most functions can be approximated by MLPs

Graphical Representation

- the Perceptron is usually represented as



- input units: coordinates of $\mathbf{x}$
- weights: coordinates of $\mathbf{w}$
- homogeneous coordinates: $\mathbf{x} = (\mathbf{x}, 1)^T$

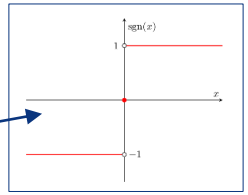
homogeneous coordinates in 2D:

$$\begin{aligned}\mathbf{w}^T\mathbf{x} + b &= w_1x_1 + w_2x_2 + b \\ &= (w_1, w_2, b)^T(x_1, x_2, 1) \\ &= \mathbf{w}^T\mathbf{x}\end{aligned}$$

$$h(\mathbf{x}) = \text{sgn}\left(\sum_i w_i x_i + w_0\right) = \text{sgn}(\mathbf{w}^T\mathbf{x})$$

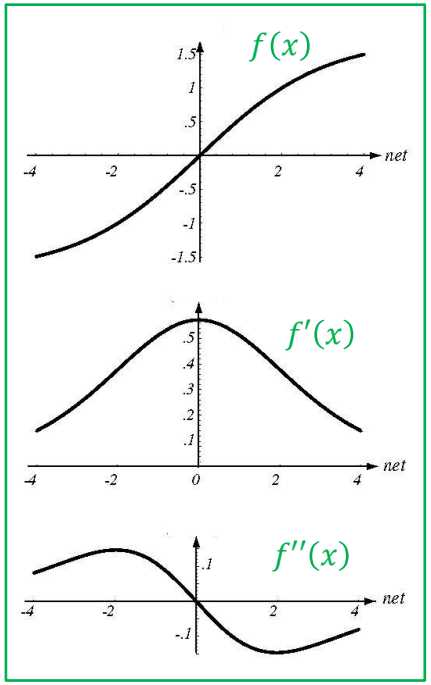
bias term
(what we have called b)

Non–Linearities



- the $\mathrm{sgn}(\cdot)$ function is **problematic** in two ways:
  - no derivative at 0, non–smooth
- it **can be approximated** in various ways
  - e.g. by the **hyperbolic tangent**

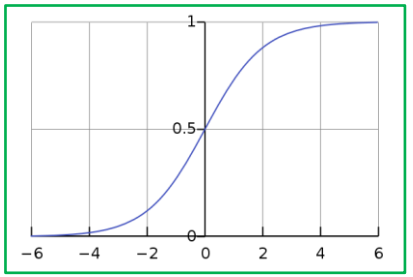
$$f(x) = \tanh(\sigma x) = \frac{e^{\sigma x} - e^{-\sigma x}}{e^{\sigma x} + e^{-\sigma x}}$$



σ controls the approximation error, but has derivative everywhere, smooth

- another **popular choice** is the **sigmoid**

$$f(x) = \frac{1}{1 + e^{-\sigma x}}$$



- The main difference between the two is the **output range**
  - $[-1,1]$ for the tanh
  - $[0,1]$ for the sigmoid

we will just refer to these as non–linearities, the exact form should be clear from context

- **early neural networks** were implemented with these functions

Neural Network

- the MLP as function approximation

even with just 2 layers,
it is possible
to approximate
complicated functions!