Machine Learning

Supervised Machine Learning – Classification – Part 1

Classification

Regression vs Classification

Age	Income	Loan Amount
21	20000	0
37	55000	150000
29	35000	120000
23	17000	55000
34	70023	250000
25	30100	90000

Output is number (continuous)

Age	Income	Loan Status
21	20000	Rejected
37	55000	Approved
29	35000	Approved
23	17000	Rejected
34	70000	Approved
25	30000	Approved

Output is label (discrete/finite)

Regression vs Classification (cont’d)

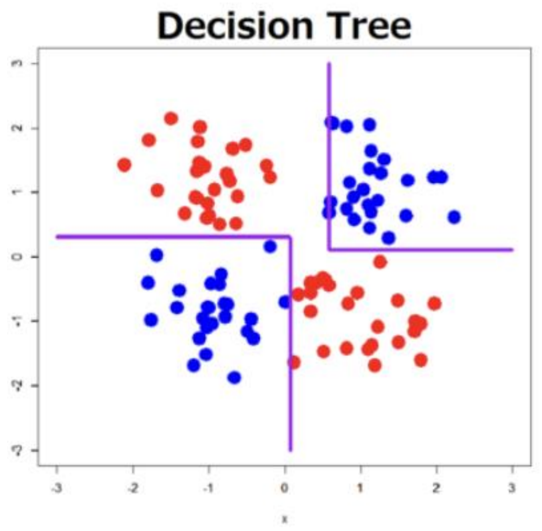
In **regression** we assign input $\boldsymbol{x}$ to one or more continuous variables y

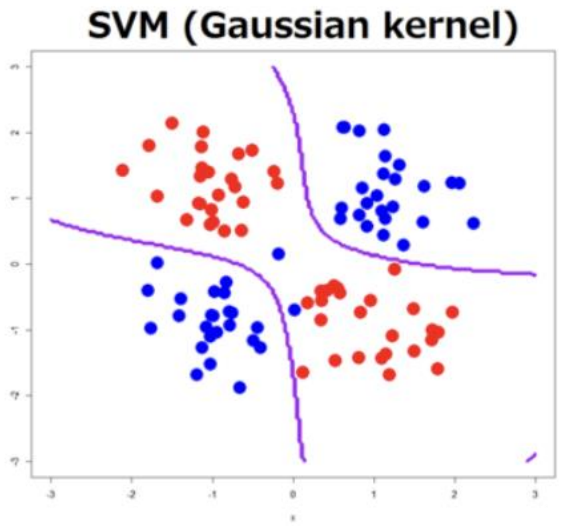
- Linear regression (even using basis functions) has simple analytical and computational properties

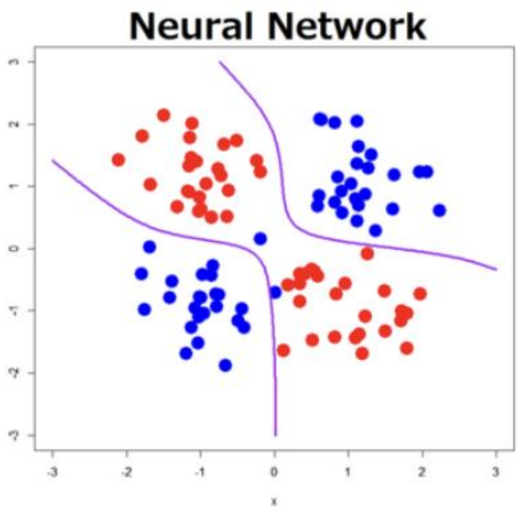
In **classification** we assign input $\boldsymbol{x}$ to one of the k discrete classes $C_k, k = 1,2,\dots,K$

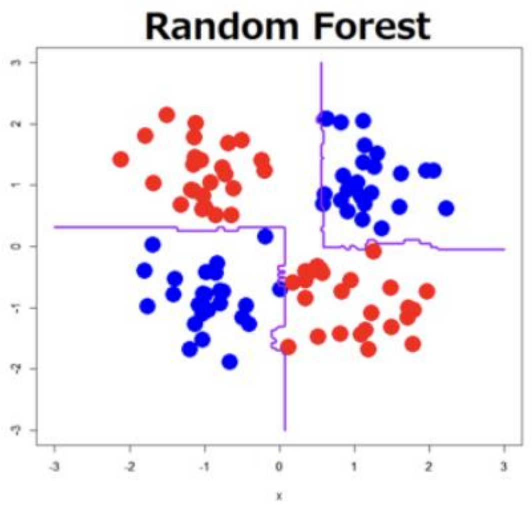
- Commonly we consider the classes disjoint
- Each input assigned a single class
- Input space is therefore divided in **decision regions**.
- Boundaries are called **decision boundaries**.

Example: Decision Boundaries





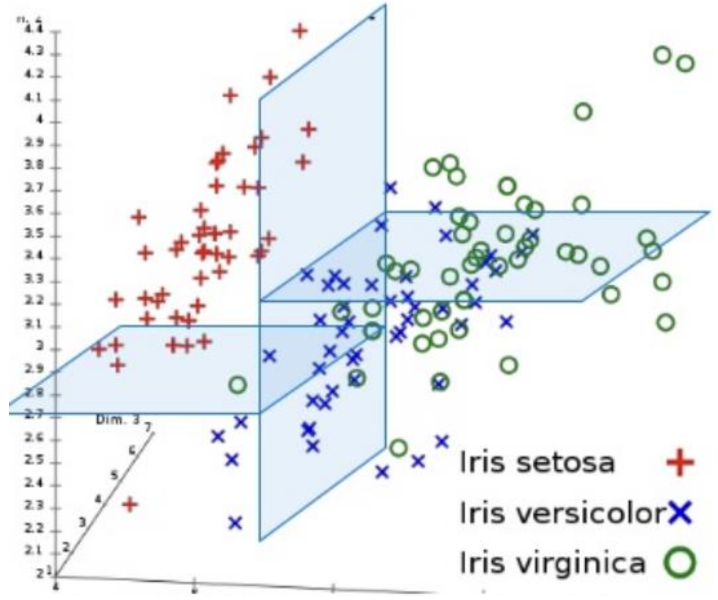




Example: Linear Boundaries

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.3	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
7.0	3.2	4.7	1.4	*I. versicolor*
6.4	3.2	4.5	1.5	*I. versicolor*
6.9	3.1	4.9	1.5	*I. versicolor*
5.5	2.3	4.0	1.3	*I. versicolor*
6.5	2.8	4.6	1.5	*I. versicolor*
5.7	2.8	4.5	1.3	*I. versicolor*
6.3	3.3	4.7	1.6	*I. versicolor*
6.3	3.3	6.0	2.5	*I. virginica*
5.8	2.7	5.1	1.9	*I. virginica*
7.1	3.0	5.9	2.1	*I. virginica*
6.3	2.9	5.6	1.8	*I. virginica*
6.5	3.0	5.8	2.2	*I. virginica*
7.6	3.0	6.6	2.1	*I. virginica*
4.9	2.5	4.5	1.7	*I. virginica*
7.3	2.9	6.3	1.8	*I. virginica*

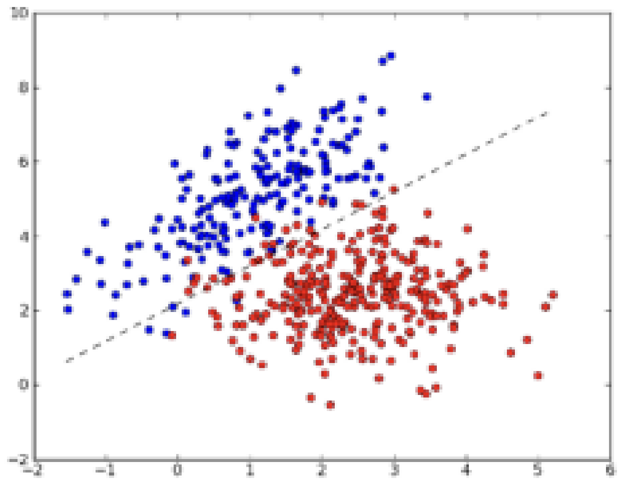
Example: Linear Boundaries (cont'd)



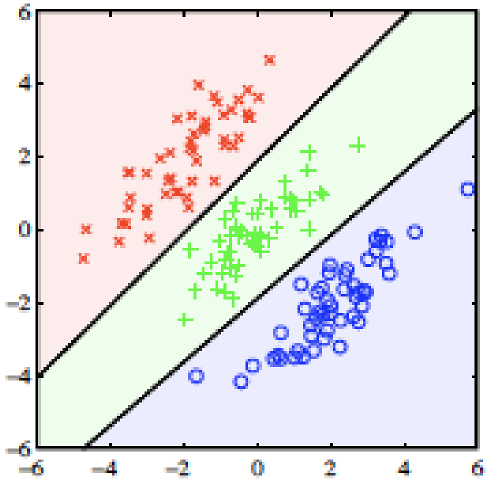
Linear Classification

- Decision surfaces are linear functions
- Each boundary separates two classes (pairwise)
- Each boundary defined as $(D - 1)$ dimensional planes within the D-dimensional input space
- Classes that can be separated by linear decision surfaces are said to be **linearly separable**

K=2



K=3



- Straight line is 1-D boundary in 2-D space
- A plane is 2-D surface in3-D space

Single-Dimensional Prediction

Previously we studied prediction:

$$y = \boldsymbol{w}^T \boldsymbol{b}(\boldsymbol{x}) \in \mathbb{R}$$

where

$$\boldsymbol{x} \in I \subseteq \mathbb{R}^d$$

$$\boldsymbol{b}(\boldsymbol{x}) = [1, \phi_1(\boldsymbol{x}), \dots, \phi_M(\boldsymbol{x})]^T \in \mathbb{R}^{M+1}$$

$$\boldsymbol{w} \in H \subseteq \mathbb{R}^{M+1}$$

Multi-Dimensional Prediction

We could extend, optionally, to multi-dimensional output:

$$\boldsymbol{y} = \boldsymbol{W}^T \boldsymbol{b}(\boldsymbol{x}) \in \mathbb{R}^D$$

where

$$\boldsymbol{x} \in I \subseteq \mathbb{R}^d$$

$$\boldsymbol{b}(\boldsymbol{x}) = [1, \phi_1(\boldsymbol{x}), \dots, \phi_M(\boldsymbol{x})]^T \in \mathbb{R}^{M+1}$$

$$\boldsymbol{W} \in H \subseteq \mathbb{R}^{D \times (M+1)}$$

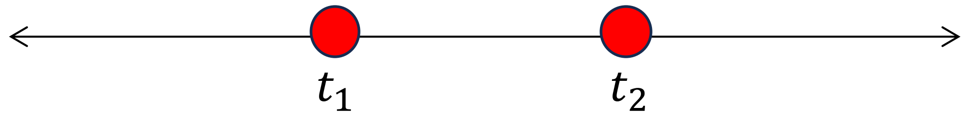
Scalar Output, Two Classes, and the Perceptron

From Categorical to Numerical

- In regression, prediction y is a real number, or vector of real numbers that we want to predict.
- In **classification**, prediction is class label.
- How to represent labels?
  - “Accept”, “Reject”?
  - “Iris Setosa”, “Iris Versicolor”,…
- It has to be numerical.
- There are multiple ways to numerically represent class labels.

Two Classes

Say $t_k \in \mathbb{R}$ is the target output for class $k \in \{1,2\}$



Two Classes

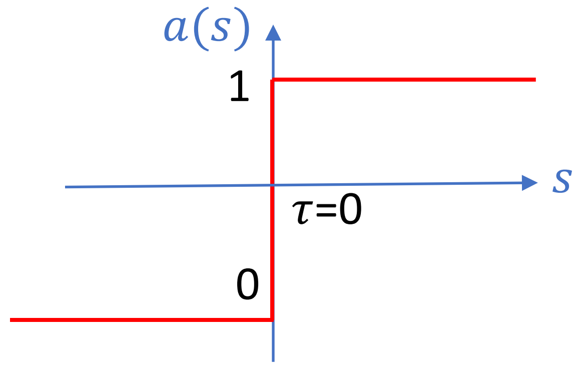
- Consider general regression:

$$\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x})$$

- Let target $t_1 = 1$ for Class 1 and $t_2 = 0$ for Class 2 (= not Class 1)
- Need a function $a(\cdot)$ to map regression to target $\{0,1\}$.

$$a(s) = \begin{cases} 1, & s \geq \tau \\ 0, & s < \tau \end{cases}$$

- We call $a(\cdot)$ **activation** function.



- We predict:

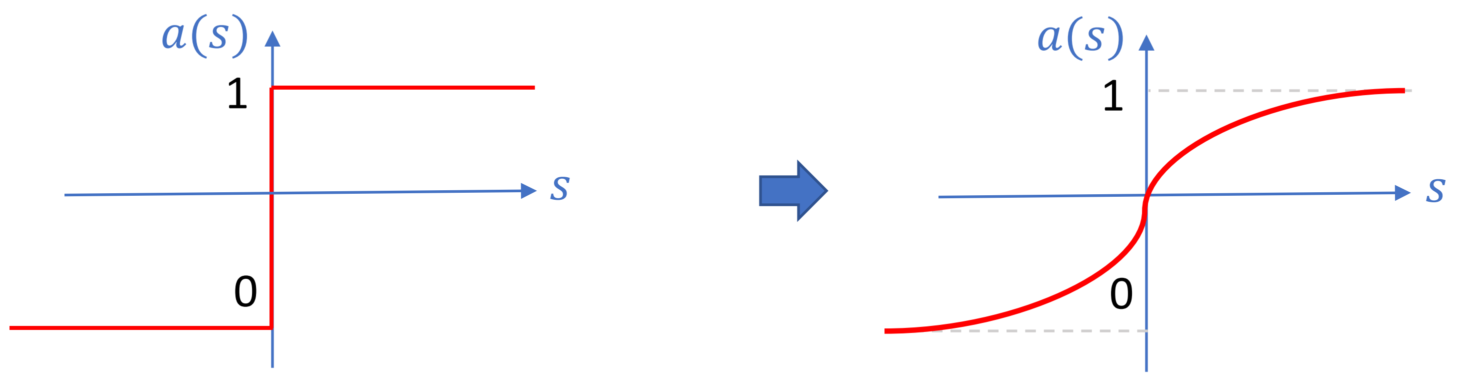
$$y = a(\boldsymbol{w}^T\boldsymbol{b}(\boldsymbol{x}))$$

- We can interpret y as **posterior decision probability for Class 1.**

How to chose τ? Absorbed by bias. Just set to $\tau = 0$.

Activation Function

From the posterior probability standpoint, we can also allow y to take values in [0,1] instead of {0,1}.

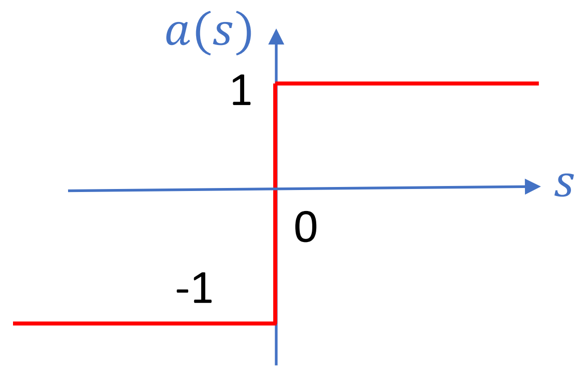


For both activation functions, we end up predicting Class 1 if $\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x}) > 0$ and Class 2 otherwise.

But with the right activation function, y captures the confidence of the prediction (the closer to 1, the more confident we are for class 1).

Perceptron Activation Function

For the perceptron algorithm (presented next), the activation function can be $a(s) = sign(s)$:



Again, we end up predicting Class 1 if $\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}) > 0$ and Class 2 otherwise.

Decision Boundary

Depending on $\boldsymbol{h}(\cdot)$ can be non-linear in $\boldsymbol{x}$ but it is linear in $\boldsymbol{w}$.

However, due to $a(\cdot)$, $y = a\left(\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x})\right)$ is be non-linear in $\boldsymbol{w}$

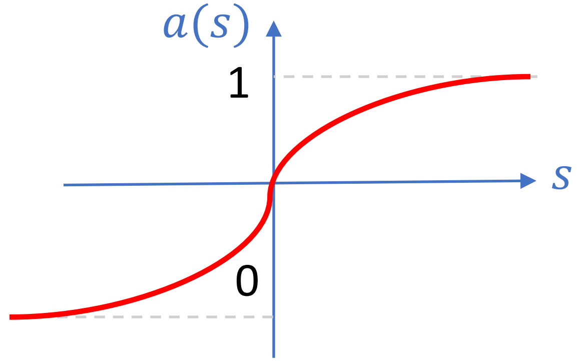
Recall, we predict Class 1 if $\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x}) > 0$ and Class 2 otherwise.

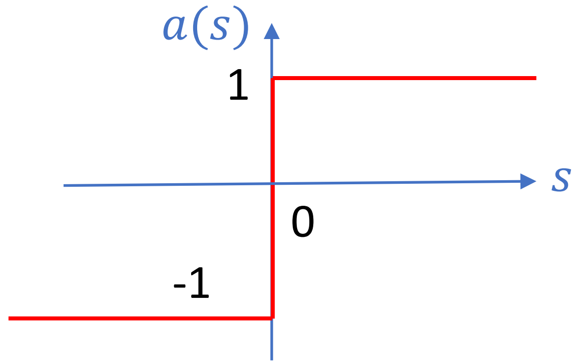
Thus, the decision boundary/surface between class 1 and 2 is

$$D = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x}) = 0\}$$

Therefore, if $\boldsymbol{h}(\boldsymbol{x})$ is linear in $\boldsymbol{x}$, then the decision surfaces are linear in $\boldsymbol{x}$ even if $a(\cdot)$ is non-linear.

In fact, this is a linear hyper-plane…





Decision Boundary

- This looks like a linear hyper-plane…
- To keep discussion simple, let $\boldsymbol{h}(\boldsymbol{x}) = [1, \boldsymbol{x}^T]^T$, so that $M = d$.
- Define $\widetilde{\boldsymbol{w}} = [w_2, \dots, w_{M+1}]^T$. $\widetilde{\boldsymbol{w}}$ is typically known as **weight vector** and w_1 referred to as **bias.**
- Then, $a\left(\boldsymbol{w}^T \boldsymbol{b}(\boldsymbol{x})\right) = a(\widetilde{\boldsymbol{w}}^T \boldsymbol{x} + w_1)$ and the decision boundary is

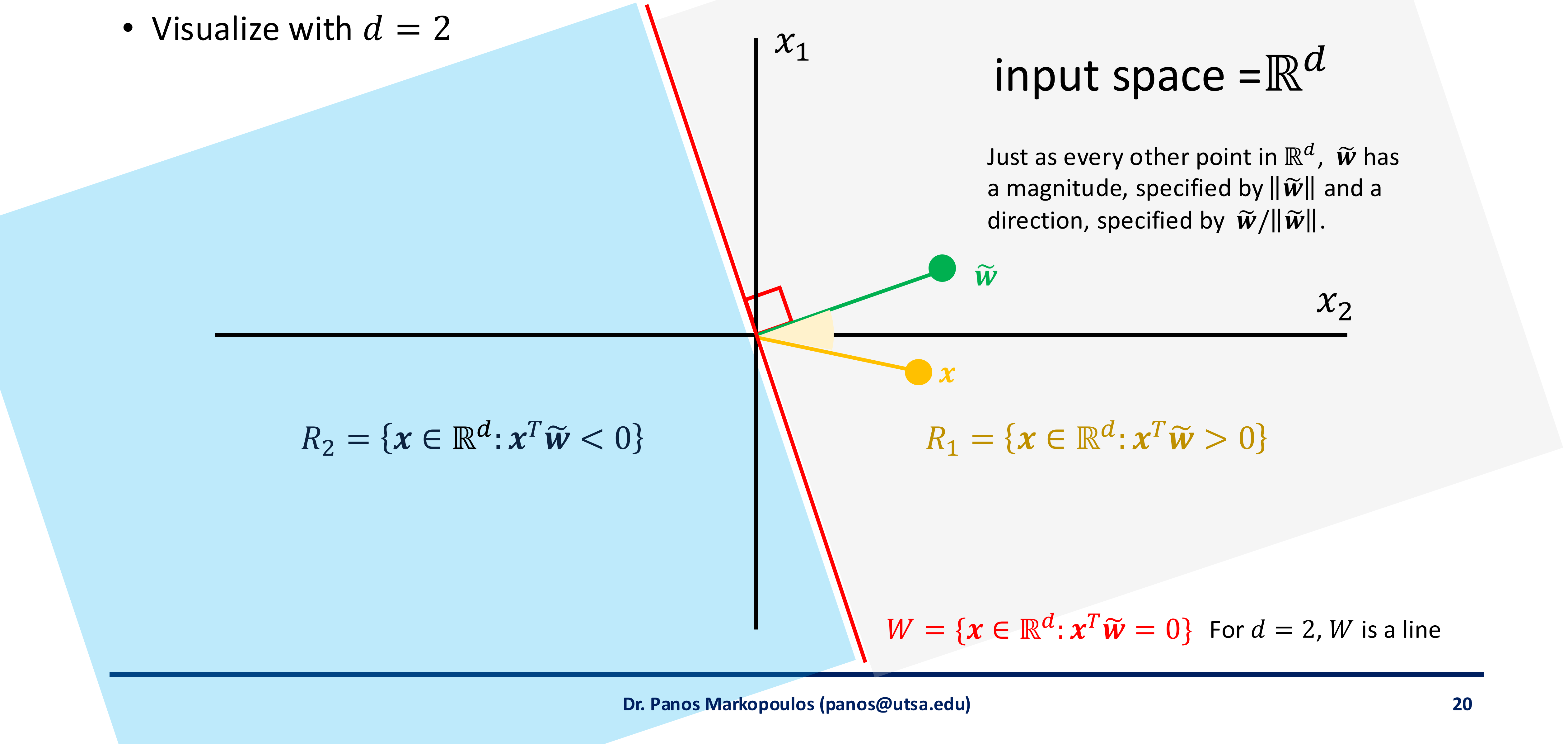
$$D = \{\boldsymbol{x} \in \mathbb{R}^d : \widetilde{\boldsymbol{w}}^T \boldsymbol{x} = -w_1\}$$

Decision Regions

- Define $W = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T \widetilde{\boldsymbol{w}} = 0\}$, which is an $(d-1)$-dimensional plane. That is, it is a linear subspace of dimension $d-1$ (i.e., hyperplane in $\mathbb{R}^d$).
- W partitions the $\mathbb{R}^d$ (the input space) in 2 halves: R_1 and R_2

Decision Regions

- Visualize with $d = 2$



input space $=\mathbb{R}^d$

Just as every other point in $\mathbb{R}^d$, $\tilde{\boldsymbol{w}}$ has a magnitude, specified by $\|\tilde{\boldsymbol{w}}\|$ and a direction, specified by $\tilde{\boldsymbol{w}}/\|\tilde{\boldsymbol{w}}\|$.

$R_2 = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} < 0\}$

$R_1 = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} > 0\}$

$W = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} = 0\}$ For $d = 2$, W is a line

Decision Regions

- The weights in $\tilde{\boldsymbol{w}}$ create 3 areas:

  hyper-plane W, half-space R_1, and half-space R_2

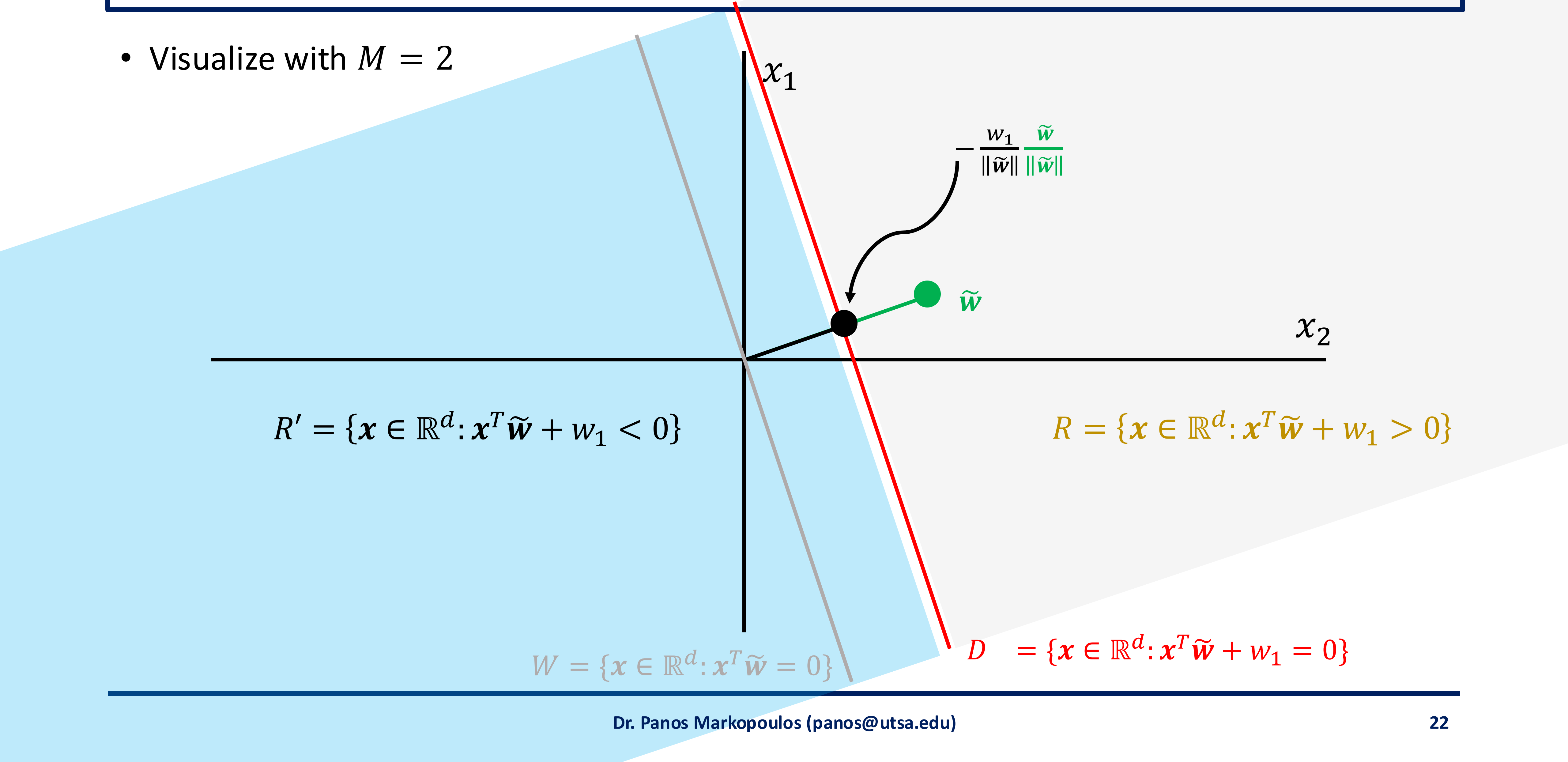
- Translate W by $\frac{-w_1}{||\tilde{\boldsymbol{w}}||}$ along $\tilde{\boldsymbol{w}}$.
  - You obtain the affine hyperplane $D = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T \tilde{\boldsymbol{w}} = -w_1\}$. Our decision boundary.
  - It is affine hyper-plane wrt $\boldsymbol{x}$ and linear hyperplane with respect to $\boldsymbol{h}(\boldsymbol{x}) = [1, \boldsymbol{x}^T]^{\mathrm{T}}$.

- D partitions the input space in 2 half-spaces:

  $R = \{\boldsymbol{z} \in \mathbb{R}^d : \boldsymbol{z}^T \tilde{\boldsymbol{w}} > -w_1\}$ and $R' = \{\boldsymbol{z} \in \mathbb{R}^d : \boldsymbol{z}^T \tilde{\boldsymbol{w}} < -w_1\}$

Decision Regions

- Visualize with $M = 2$



x_1
x_2
$-\frac{w_1}{\|\tilde{\boldsymbol{w}}\|}\frac{\tilde{\boldsymbol{w}}}{\|\tilde{\boldsymbol{w}}\|}$
$\tilde{\boldsymbol{w}}$

$R' = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} + w_1 < 0\}$

$R = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} + w_1 > 0\}$

$W = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} = 0\}$

$D \quad = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{x}^T\tilde{\boldsymbol{w}} + w_1 = 0\}$

Classification

- Back to classification

$$y = a\left(\boldsymbol{w}^T \boldsymbol{b}(\boldsymbol{x})\right)$$

- Parameter vector $\boldsymbol{w} = [w_1, \tilde{\boldsymbol{w}}^T]^T$ creates 3 areas:
  - $D = \{\boldsymbol{x} \in \mathbb{R}^d : y = 0\}$: decision boundary
  - $R = \{\boldsymbol{x} \in \mathbb{R}^d : y > 0\}$: decision region for class 1, C_1
  - $R' = \{\boldsymbol{x} \in \mathbb{R}^d : y < 0\}$: decision region for class 2, C_2

- Goal: optimize $\boldsymbol{w}$ so classification is accurate, on unseen data.

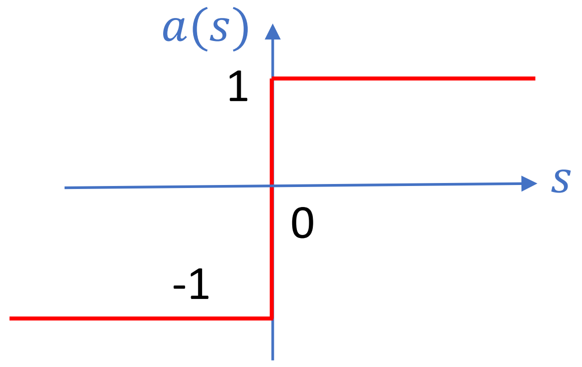
The Perceptron Algorithm

- The output is

$$y(\boldsymbol{x}) = a(\boldsymbol{w}^T \boldsymbol{b}(\boldsymbol{x}))$$

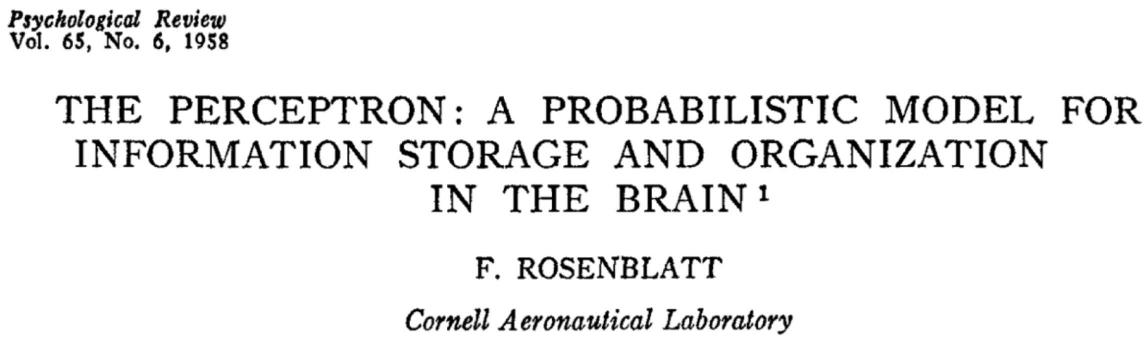
- Activation function $a(.)$ can be the sign function $a(s) = 1$ if $s \geq 0$ and -1 otherwise.

$$a(s) = \mathrm{sign}(s) = \begin{cases} 1, s \geq 0 \\ -1, s < 0 \end{cases}$$



The Perceptron Algorithm

- A historic linear discriminant model



Psychological Review
Vol. 65, No. 6, 1958

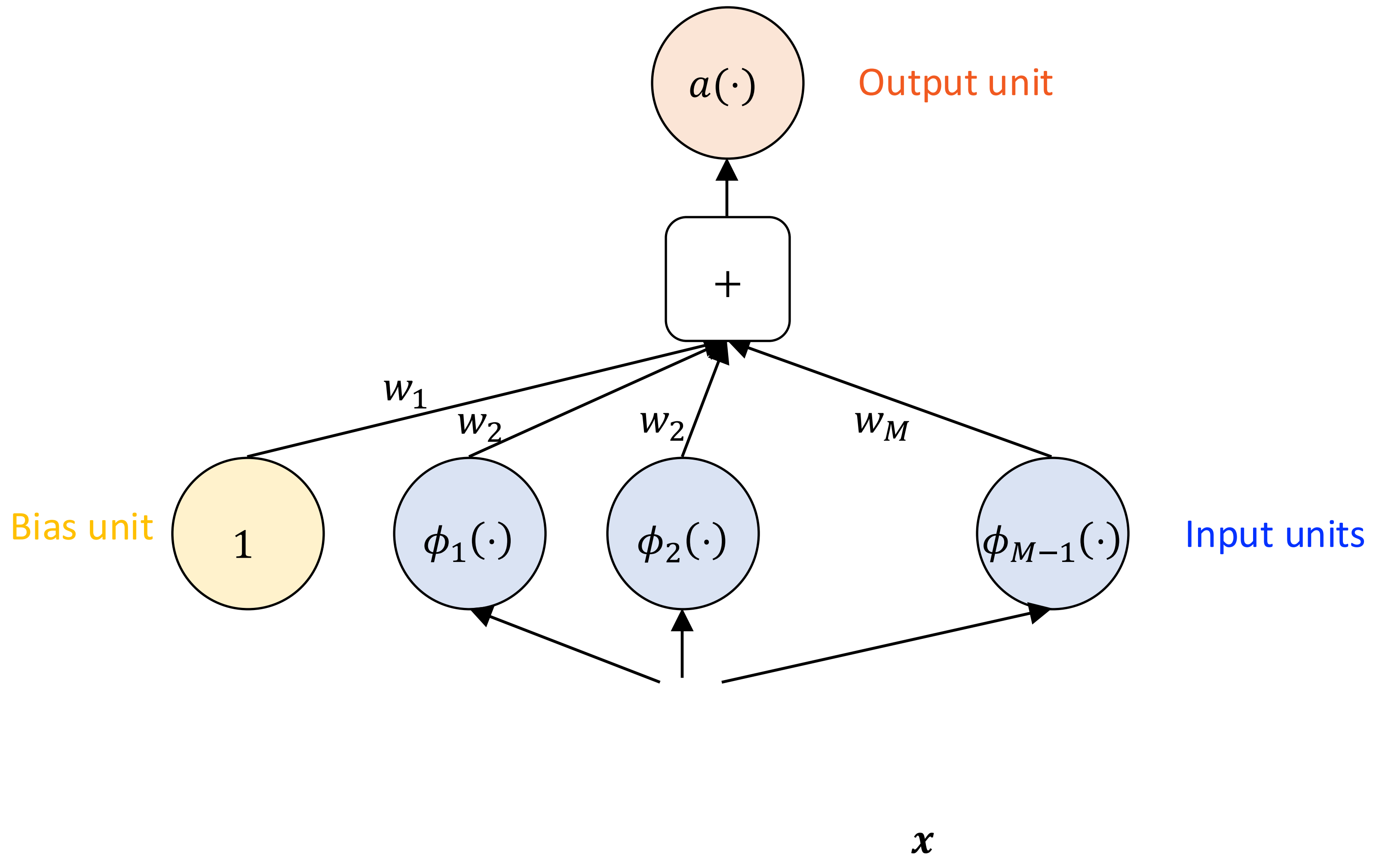
THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

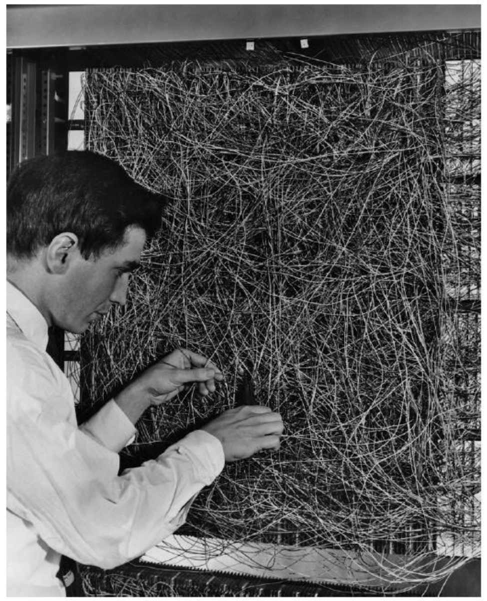
F. ROSENBLATT

Cornell Aeronautical Laboratory

Frank Rosenblatt
“the father of deep learning”

The Perceptron Algorithm





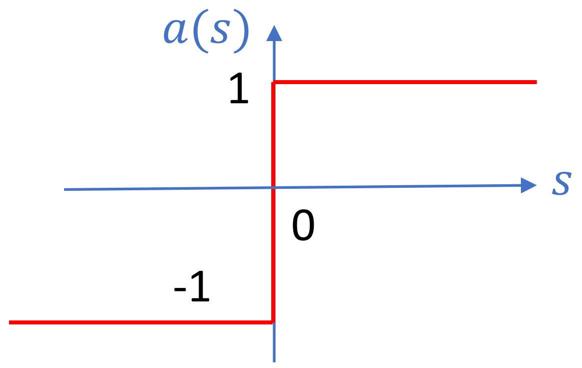
Perceptron Target Coding

Use target “coding” scheme

Earlier we focused on $t \in \{0,1\}$, which is appropriate for probabilistic models

For perceptrons it is more convenient to use target values {+1, -1}

For that purpose, the activation function can be $a(s) = sign(s)$:



Perceptron Error Criterion

Perception criterion is motivated by error function minimization

How do we define “error” in classification?

A natural choice is total number of misclassifications

Let $t \in \{-1, +1\}$ be the training output and $\hat{t} \in \{-1, +1\}$ be the predicted one.

If t and $\hat{t}$ differ, then $|t - \hat{t}| = 2$; otherwise, it is 0.

Therefore, our normalized error can be

$$\left|t - a\left(\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x})\right)\right|/2$$

Perceptron Error Criterion

$$\frac{\left|t - a\left(\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x})\right)\right|}{2}$$

This is a piecewise linear function on $\boldsymbol{w}$, with discontinuities

Cannot use methods that change $\boldsymbol{w}$ based on gradient of error

Alternative error criterion is the **perceptron criterion**

Perceptron Error Criterion

Same as before, we will try to find $\boldsymbol{w}$ that satisfies

$$\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x}) = \begin{cases} \geq 0, \text{for } \boldsymbol{x} \text{ in } C_1 \\ < 0, \text{for } \boldsymbol{x} \text{ in } C_2 \end{cases}$$

If t is the corresponding target for which it holds $t = sign\left(\boldsymbol{w}^T\boldsymbol{b}(\boldsymbol{x})\right)$. That is,

$$\boldsymbol{w}^T\boldsymbol{b}(\boldsymbol{x}) \cdot t = |\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x})| > 0$$

The perceptron would like to make all classifications correct.

That is, during training, it wants to make $\boldsymbol{w}^T\boldsymbol{h}(\boldsymbol{x}_n)\, t_n > 0$ for every training sample $\boldsymbol{x}_n$

Perceptron Error Criterion

For any $\boldsymbol{w}$, define:

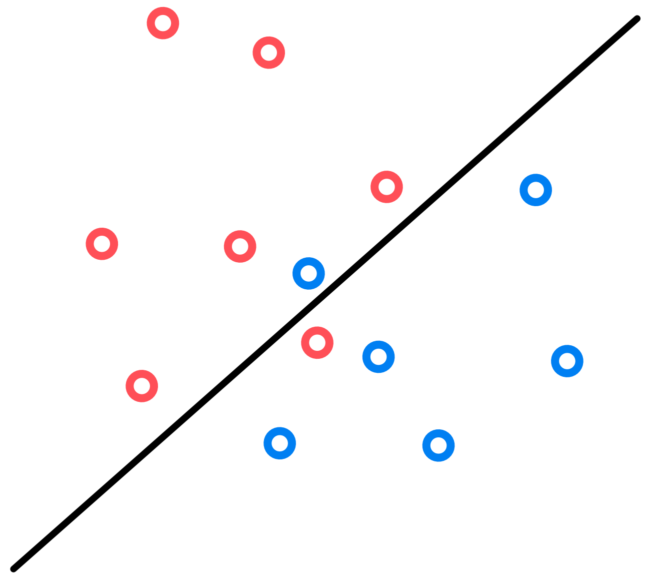
$$I_e(\boldsymbol{w}) = \{n \in [N]: \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n < 0\}$$

Accordingly, define:

$$I_c(\boldsymbol{w}) = \{n \in [N]: \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n \geq 0\}$$

It holds that

$$I_e(\boldsymbol{w}) \cup I_c(\boldsymbol{w}) = [N]$$



One idea is to design $\boldsymbol{w}$ that minimizes $|I_e(\boldsymbol{w})|$

Another idea is to find $\boldsymbol{w}$ that minimizes $\left|\sum_{n \in I_e(\boldsymbol{w})} \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n\right|^2$

Perceptron Error Criterion

Perceptron minimizes the error on the misclassifications as:

$$L_p(\boldsymbol{w}) = -\sum_{n \in I_e(\boldsymbol{w})} \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n$$

An alternative way to write this is

$$L_p(\boldsymbol{w}) = \sum_{n \in [N]} \max(0, -\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n)$$

This is known as the **perceptron cost**

Perceptron Error Criterion

Reformulating, define $\boldsymbol{H} = [\boldsymbol{h}(\boldsymbol{x}_1), \dots, \boldsymbol{h}(\boldsymbol{x}_N)]$, $\boldsymbol{V} = \mathrm{diag}([t_1, \dots, t_N])$ and $\boldsymbol{c}(\boldsymbol{w})$ such that

$$[\boldsymbol{c}(\boldsymbol{w})]_n = \begin{cases} -1, \text{if } \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n < 0 \\ 0, \text{otherwise} \end{cases}$$

Then, the perceptron optimization becomes

$$\min_{\boldsymbol{w}} \boldsymbol{w}^T \boldsymbol{H} \boldsymbol{V} \boldsymbol{c}(\boldsymbol{w})$$

Perceptron Error Criterion

Then, the perceptron optimization becomes

$$\min_{\boldsymbol{w}} \boldsymbol{w}^T \boldsymbol{H}\boldsymbol{V}\boldsymbol{c}(\boldsymbol{w})$$

Let's work on this some more. Define $\boldsymbol{A} = \boldsymbol{H}\boldsymbol{V}$**.** The objective above becomes $\boldsymbol{w}^T \boldsymbol{A}\boldsymbol{c}(\boldsymbol{w})$

We observe that $\boldsymbol{c}(\boldsymbol{w}) = (\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) - \mathbf{1}_N)/2$

Thus, the perceptron objective (x2) becomes:

$$2\,\boldsymbol{w}^T \boldsymbol{A}\boldsymbol{c}(\boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{A}(\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) - \mathbf{1}_N) = \boldsymbol{w}^T \boldsymbol{A}\,\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) - \boldsymbol{w}^T \boldsymbol{A}\mathbf{1}_N$$

Perceptron Error Criterion

Thus, the perceptron objective (x2) becomes:

$$2\,\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{c}(\boldsymbol{w}) = \boldsymbol{w}^T\boldsymbol{A}(\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) - \mathbf{1}_N) = \boldsymbol{w}^T\boldsymbol{A}\,\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) - \boldsymbol{w}^T\boldsymbol{A}\mathbf{1}_N$$

Notice that for any vector $\boldsymbol{z}$, $\|\boldsymbol{z}\|_1 = \boldsymbol{z}^T\mathrm{sign}(\boldsymbol{z})$

Therefore, the perceptron optimization can be rewritten as

$$\min_{\boldsymbol{w}} \|\boldsymbol{A}^T\boldsymbol{w}\|_1 - \mathbf{1}_N^T\boldsymbol{A}^T\boldsymbol{w}$$

where $\boldsymbol{A} = \boldsymbol{X}\boldsymbol{V}$, $\boldsymbol{X} = [\boldsymbol{b}(\boldsymbol{x}_1), \dots, \boldsymbol{b}(\boldsymbol{x}_N)]$, and $\boldsymbol{V} = \mathrm{diag}([t_1, \dots, t_N])$

Optimizing Perceptron Parameters

$$\min_{\boldsymbol{w}} \|\boldsymbol{A}^T\boldsymbol{w}\|_1 - \boldsymbol{1}_N^T\boldsymbol{A}^T\boldsymbol{w}$$

The objective is lower bounded by 0, since $\|\boldsymbol{A}^T\boldsymbol{w}\|_1 > \boldsymbol{1}_N^T\boldsymbol{A}^T\boldsymbol{w}$.

Equality to 0 is attained for $\boldsymbol{w} = \boldsymbol{0}$, which has no training value.

Also, equality to 0 is attained for any $\boldsymbol{w}$ such that

$$\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) = \boldsymbol{1}_N$$

This means, this error becomes 0 if there are no misclassifications.

Depending on dataset, there might exist no $\boldsymbol{w}$ such that $\mathrm{sign}(\boldsymbol{A}^T\boldsymbol{w}) = \boldsymbol{1}_N$.

For finite noisy data, perfect classification might mean overfitting.

Optimizing Perceptron Parameters

$$\min_{\boldsymbol{w}} \|\boldsymbol{A}^T\boldsymbol{w}\|_1 - \mathbf{1}_N^T\boldsymbol{A}^T\boldsymbol{w}$$

- So, we can solve this in closed form as $\boldsymbol{w} = \mathbf{0}$ which would be of no use.
- Here is how gradient descent would look like.
- Gradient: $\frac{\partial}{\partial x}|x| = \text{sign}(x)$, defined only for $x \neq 0$.

$$\boldsymbol{g}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}}\|\boldsymbol{A}^T\boldsymbol{w}\|_1 \;- \mathbf{1}_N^T\boldsymbol{A}^T\boldsymbol{w}$$

$$= \nabla_{\boldsymbol{w}} \sum_{n=1:N} |\boldsymbol{w}^T\boldsymbol{a}_n| - \boldsymbol{w}^T\boldsymbol{A}\mathbf{1}_N = \sum_{n=1:N} \boldsymbol{a}_n(\text{sign}(\boldsymbol{w}^T\boldsymbol{a}_n) - 1) = \boldsymbol{A}\,(\text{sign}(\boldsymbol{A}^T\boldsymbol{w}) - \mathbf{1}_N)$$

Optimizing Perceptron Parameters

- The gradient descent algorithm, initialized at any $\boldsymbol{w}(0) \neq 0$ would iterate as

$$\boldsymbol{w}(i) = \boldsymbol{w}(i-1) - \mu\, \boldsymbol{g}_i$$

- Looking into the gradient

$$\boldsymbol{g}_i = \boldsymbol{g}\big(\boldsymbol{w}(i-1)\big) = \boldsymbol{A}\left(\operatorname{sign}\big(\boldsymbol{A}^T\boldsymbol{w}(i-1)\big) - \mathbf{1}_N\right) = \sum_{n=1:N} \boldsymbol{a}_n \left(\operatorname{sign}(\boldsymbol{a}_n^T\boldsymbol{w}(i-1)) - 1\right)$$

- If $\boldsymbol{w}(i-1)$ classifies $\boldsymbol{x}_n$ correctly, then $\boldsymbol{a}_n^T\boldsymbol{w}(i-1) > 0$ and $(\operatorname{sign}(\boldsymbol{a}_n^T\boldsymbol{w}(i-1)) - 1) = 0$. Thus, $\boldsymbol{x}_n$ does not participate in the gradient.

- If $\boldsymbol{w}(i-1)$ classifies $\boldsymbol{x}_n$ incorrectly, then $\boldsymbol{a}_n^T\boldsymbol{w}(i-1) < 0$ and $(\operatorname{sign}(\boldsymbol{a}_n^T\boldsymbol{w}(i-1)) - 1) = -2$.

- Thus,

$$\boldsymbol{g}\big(\boldsymbol{w}(i-1)\big) = -2 \sum_{n \in I_e(\boldsymbol{w}(i-1))} \boldsymbol{a}_n$$

Optimizing Perceptron Parameters

- Because the problem is convex, GD iterations will converge to the exact solution.
  - Either perfectly classify all training data (might not be feasible if not perfectly linearly separable), or $\boldsymbol{w} = \mathbf{0}$
- Instead, run stochastic gradient descent (SGD)
- SGD is like GD, but the gradient is computed on a subset of the available training data (we call that **mini-batch**). In the case of perceptron, one data point at the time.
- SGD might not converge to exact solution
  - Might not converge to $\mathbf{0}$ and limit overfitting.

Perceptron Algorithm (SGD)

SGD for perceptron:

```
1. Initialize $\boldsymbol{w}(0)$, so that $\|\boldsymbol{w}(0)\| = 1$

2. For $i = 1,2,\ldots,$

       Update: $\boldsymbol{w}(i) = \boldsymbol{w}(i-1) - \mu\, \boldsymbol{a}_i(\mathrm{sign}(\boldsymbol{a}_i^T \boldsymbol{w}(i-1)) - 1)$
```

Or:

```
1. Initialize $\boldsymbol{w}(0)$, so that $\|\boldsymbol{w}(0)\| = 1$

2. For $i = 1,2,\ldots,$

       If $\boldsymbol{w}(i-1)$ misclassifies $\boldsymbol{x}_i$ (i.e., if $\boldsymbol{a}_i^T \boldsymbol{w}(i-1)<0$),

           Update: $\boldsymbol{w}(i) = \boldsymbol{w}(i-1) + 2\mu\, \boldsymbol{a}_i$
```

Perceptron Algorithm (SGD)

Looking into the update…

If $\boldsymbol{w}(i-1)$ misclassifies $\boldsymbol{x}_i$ (i.e., if $\boldsymbol{a}_i^T\boldsymbol{w}(i-1)$<0), $\boldsymbol{w}(i) = \boldsymbol{w}(i-1) + r\,\boldsymbol{a}_i$, $r = 2\mu$

This means that $\boldsymbol{a}_i^T\boldsymbol{w}(i) = \boldsymbol{a}_i^T\boldsymbol{w}(i-1) + 2\mu\|\boldsymbol{a}_i\|^2 > \boldsymbol{a}_i^T\boldsymbol{w}(i-1)$.

That is, the perceptron updated to $\boldsymbol{w}(i)$ in a way that "improves" classification on point $\boldsymbol{x}_i$

If misclassified $\boldsymbol{x}_i$ was from class 1 ($t_i = +1$), then $\boldsymbol{a}_i = \boldsymbol{h}(\boldsymbol{x}_i)$ and

$$\boldsymbol{w}(i) = \boldsymbol{w}(i-1) + r\,\boldsymbol{h}(\boldsymbol{x}_i)$$

If misclassified $\boldsymbol{x}_i$ was from class 2 ($t_i = -1$), then $\boldsymbol{a}_i = -\boldsymbol{h}(\boldsymbol{x}_i)$ and

$$\boldsymbol{w}(i) = \boldsymbol{w}(i-1) - r\,\boldsymbol{h}(\boldsymbol{x}_i)$$

SGD Epochs

When you process all training data, start again from the beginning. This is a new “epoch”.

You can randomly re-order the training data.

The new epoch is initialized to the parameter vector returned by the previous epoch.

You can terminate upon argument/parameter convergence or other termination condition.

Properties of the Perceptron

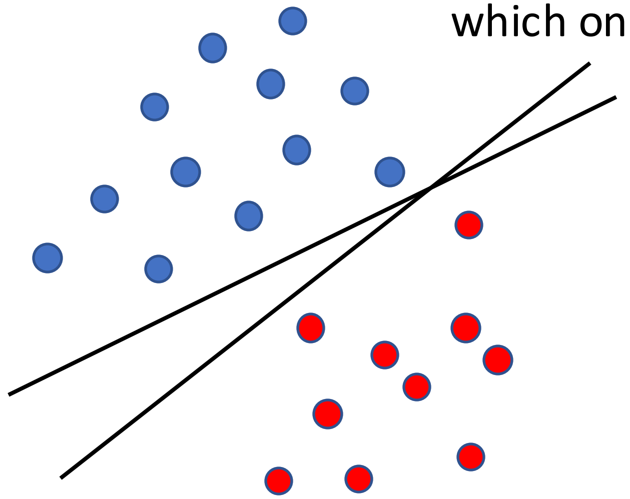
Convergence theorem: If the training data are **linearly separable**, the perceptron algorithm with converge to perfect classification.

Cycling theorem: If the training data are **not linearly separable**, the perceptron algorithm cannot attain perfect classification; moreover, it will start repeating the same weights in an infinite loop.

Variants of the Perceptron

Standard Algorithm

```
1. Initialize $w(0)$, so that $\|w(0)\| = 1$

2. For $i = 1,2,\ldots,$

        If $t_i \boldsymbol{h}(\boldsymbol{x}_i)^T \boldsymbol{w}(i-1) < 0$,

            Update: $\boldsymbol{w}(i) = \boldsymbol{w}(i-1) + r\, t_i \boldsymbol{h}(\boldsymbol{x}_i)$
```



which one is better?

Margin Perceptron

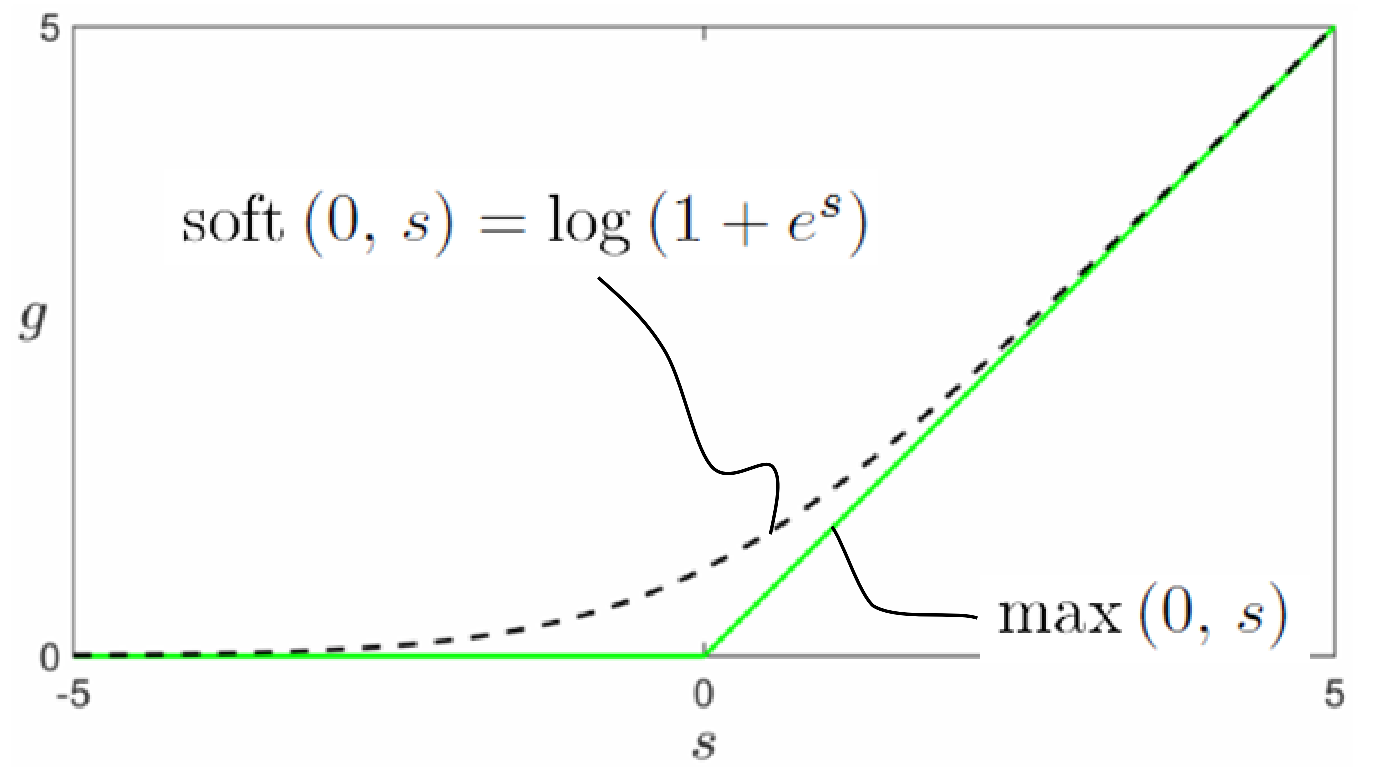
Do not update only when classification is incorrect.

Update also when the classification is correct, but a close call.

Margin perceptron criterion: $L_{mp}(\boldsymbol{w}) = \sum_{n\in[N]} \max(0, \eta - \boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n)$

```
Margin Perceptron Algorithm

1. Initialize $\boldsymbol{w}(0)$, so that $\|\boldsymbol{w}(0)\| = 1$

2. For $i = 1,2,\dots,$

        If $t_i \boldsymbol{h}(\boldsymbol{x}_i)^T \boldsymbol{w}(i-1) < \eta$,

                Update: $\boldsymbol{w}(i) = \boldsymbol{w}(i-1) + r\, t_i \boldsymbol{h}(\boldsymbol{x}_\mathrm{i})$
```

Softmax Perceptron



Change criterion to

$$L_{smp}(\boldsymbol{w}) = \sum_{n\in[N]} \mathrm{softmax}(0, -\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n) = \sum_{n\in[N]} \log(1 + \exp(-\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n))$$

SoftMax Perceptron

- We find that

$$\nabla L_{smp}(\boldsymbol{w}) = g(\boldsymbol{w}) = -\sum_{n\in[N]} \sigma(-y_n \boldsymbol{h}(\boldsymbol{x}_n)^T \boldsymbol{w}) y_n \boldsymbol{h}(\boldsymbol{x}_n)$$

- For softmax perceptron start with $\boldsymbol{w}(0)$ and perform GD.
- We can apply softmax instead of max to margin perceptron as well. We call that soft-margin perceptron.

Measuring Classification Accuracy

To measure performance attained by $\boldsymbol{w}$ in any dataset $(\boldsymbol{t}, \boldsymbol{X})$:

$$L_p(\boldsymbol{w}; \boldsymbol{t}, \boldsymbol{X}) = \sum_{n \in [N]} \max(0, -\boldsymbol{w}^T \boldsymbol{h}(\boldsymbol{x}_n)\, t_n)$$

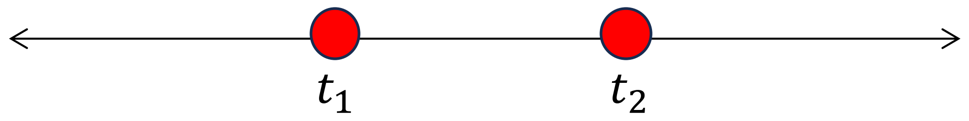
Another metric is accuracy:

$$1 - \frac{L_p(\boldsymbol{w}; \boldsymbol{t}, \boldsymbol{X})}{N}$$

Multiple Classes

Two Classes

Say $t_k \in \mathbb{R}$ is the target output for class $k \in \{1,2\}$



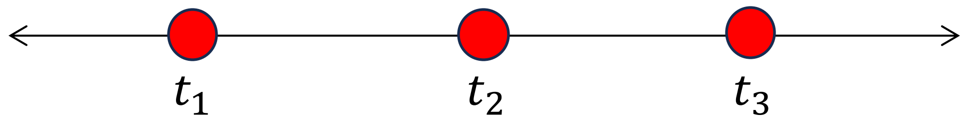
Label Bias and Multidimensional Output (cont'd)

For $K > 2$ classes, this will create a bias.

There exists no selection of class targets $\{t_k\}_{k=1}^{K} \subset \mathbb{R}$ and $c \in \mathbb{R}_+$, so that

$$|t_k - t_l| = c \ \forall k \neq l$$

This will bias the model towards confusing classes with more proximal target values.



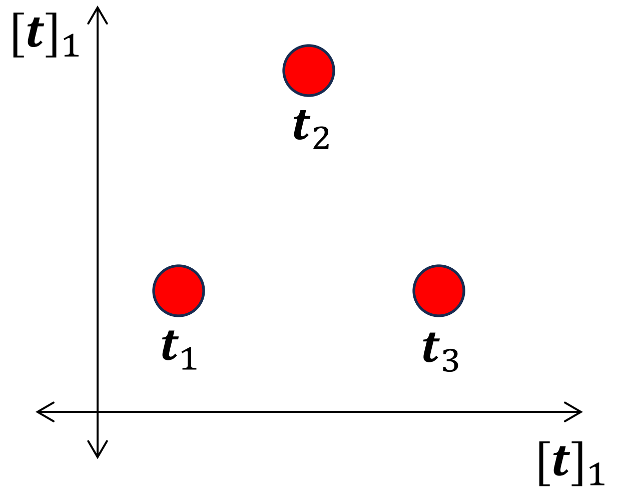
What if I increase the output dimensionality?

Label Bias and Multidimensional Output (cont'd)

What if $D = 2$?

Up to $K \leq 3$ will work.

But for $K > 3$, there is no selection of class targets $\{\boldsymbol{t}_k\}_{k=1}^{K} \subset \mathbb{R}$ and $c \in \mathbb{R}_+$:



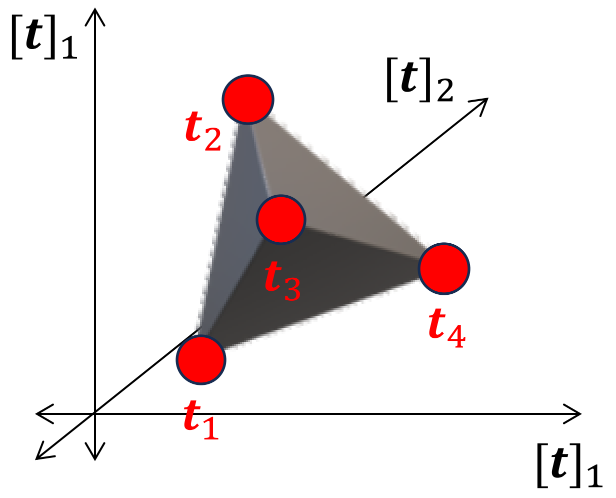
$$\|\boldsymbol{t}_k - \boldsymbol{t}_l\| = c \;\; \forall k \neq l$$

Again, label bias.

Label Bias and Multidimensional Output (cont'd)

What if $D = 3$?

Up to $K \leq 4$ will work.



In general:

For any D we can support up to $K + 1$ classes.

Equivalently, for K classes we need at least $D = K - 1$.

For $D = K - 1$, target labels at vertices of D-dimensional regular simplex.

In fact, any $D \geq K - 1$ should work.

One-Hot Label Encoding

We set $D = K$ and define $\{\boldsymbol{t}_k\}_{k=1}^{K}$ to be any selection of orthonormal vectors in $\mathbb{R}^D$.

That is, $\boldsymbol{T} = [\boldsymbol{t}_1, \dots, \boldsymbol{t}_D]$ such that $\boldsymbol{T}^T\boldsymbol{T} = \boldsymbol{I}_D$.

Most commonly, we set, $\boldsymbol{T} = \boldsymbol{I}_D$. That is,

$$[\boldsymbol{t}_k]_l = \delta_{k,l} \; \forall k, l$$

This is known as 1-of-K or one-hot encoding scheme.

One-Hot Label Encoding – Probabilistic Interpretation

- We can design classifier with prediction $\boldsymbol{t} = \{0,1\}^K$.
- For example, if $K = 5$, a class 2 is represented by one-hot target $\boldsymbol{t} = [0,1,0,0,0]^T$.
- The value of t_k can interpreted as **posterior probability** of class C_k, given training data and input.

Posterior Class Probability

Design classifier with output $\boldsymbol{y} \in [0,1]^5$ and $\|\boldsymbol{y}\|_1 = 1$, and interpret it as the vector of posterior class probabilities given training data and input:

$y_k = P(C_k|\boldsymbol{x}, S)$ or simpler notation: $y_k = P(C_k|\boldsymbol{x})$

Accordingly, the maximum-aposteriori-probability (MAP) classification decision is

$$\text{Class } k \text{ if } l = \text{argmax}_{k \in [K]} y_k$$

This is called the **discriminative approach**, when the model outputs $P(C_k|\boldsymbol{x})$ directly.

Alternative is the **generative approach:** model likelihood function $f(x|C_k)$ and then use prior class probabilities $P(C_k)$ and the Bayes Rule to obtain $P(C_k|\boldsymbol{x})$ –then proceed with classification as above.

Posterior Class Probability

- K linear functions; each function “champions” a class.
- Function for class k

$$y_k = \boldsymbol{w}_k \boldsymbol{h}(\boldsymbol{x})$$

- y_k plays the role of (scaled) $P(C_k|\boldsymbol{x})$
- Prediction vector: $\boldsymbol{y} = \boldsymbol{W}^T \boldsymbol{h}(\boldsymbol{x})$, where $\boldsymbol{W} = [\boldsymbol{w}_1, \dots, \boldsymbol{w}_K]$.
- Assign a point $\boldsymbol{x}$ to class C_k if $y_k > y_i$ for every $i \neq k$
- Decision boundary between C_k and C_i is $\{\boldsymbol{x} \in \mathbb{R}^d;\ y_k = y_i\}$

MAP Classification

- Pass through activation function (element-wise) to form $a(\boldsymbol{y})$.
- Decide again by the index of the maximum. Should be same decision for monotonically increasing activation.
- Maximum a-posteriori probability classification: decide Class k, where

$$k = \operatorname{argmax}_{l \in [K]} p_l$$

and

$$\boldsymbol{p} = \frac{a(\boldsymbol{y})}{\|a(\boldsymbol{y})\|_1}$$

Optimizing Parameters

- But how do we design/optimize $\boldsymbol{W}$?
- Up until class detection the formulation looked like regression. Let's do LS?
- Remember, we classified based on the index of the highest value of the output

- Now we need to give a numerical value to the training outputs (training targets)
- This numerical value will critically determine the design of $\boldsymbol{W}$.
- Let the training targets have entries in {0,1}, as we showed early on
- That is, $\boldsymbol{t}_n = \boldsymbol{e}_{k,K}$ (the k-th column of $\boldsymbol{I}_K$) iff x_n came from Class k.

Optimizing Parameters - LS

Design $\boldsymbol{W}$ by minimizing least squares:

$$\sum_{n=1}^{N} \|\boldsymbol{t}_n - \boldsymbol{W}^T \boldsymbol{b}(\boldsymbol{x}_n)\|^2 = \left\|\boldsymbol{T} - \boldsymbol{W}^T \boldsymbol{H}\right\|_F^2$$

Where $\boldsymbol{H} = [\boldsymbol{h}(\boldsymbol{x}_1), \dots, \boldsymbol{h}(\boldsymbol{x}_N)]$ and $\|\boldsymbol{A}\|_F^2 = \sum_{n,k} \left|A_{n,k}\right|^2$.

How do we solve this?

Same as before: $\nabla_{\boldsymbol{W}} \left\|\boldsymbol{T} - \boldsymbol{W}^T \boldsymbol{Q}\right\|_F^2 = \boldsymbol{0} \Rightarrow (\boldsymbol{Q}\boldsymbol{Q}^T)\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{T}^T \Rightarrow \boldsymbol{W} = (\boldsymbol{Q}\boldsymbol{Q}^T)^{-1}\boldsymbol{Q}\boldsymbol{T}^T$

This does not work well, as it is biased to specific values of $\boldsymbol{T}$.Next, we will study methods that uses the train labels without assigning arbitrary numerical targets