CS57300
PURDUE UNIVERSITY

FEBRUARY 21, 2019

DATA MINING

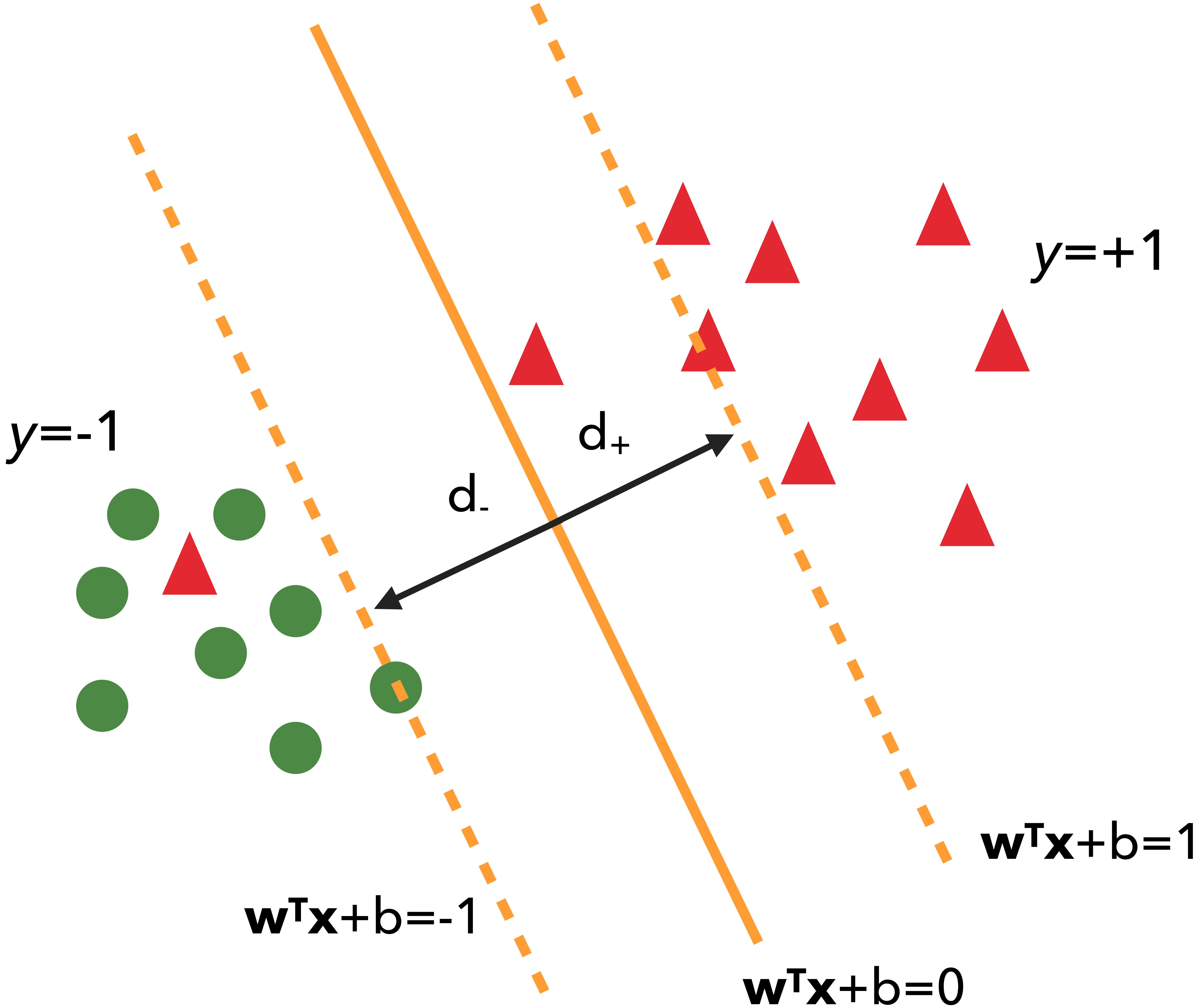
SVM: RECAP

WHAT ABOUT LINEARLY NON-SEPARABLE DATA?

- Introduce slack variables $\varepsilon_i \geq 0$ such that:

  $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \varepsilon_i, \forall i \in \{1,2,\ldots,N\}$

- ε_i measures the amount of error
  - When $0 < \varepsilon_i \leq 1$, data is between the margin, but classified correctly
  - When $\varepsilon_i > 1$, data is misclassified



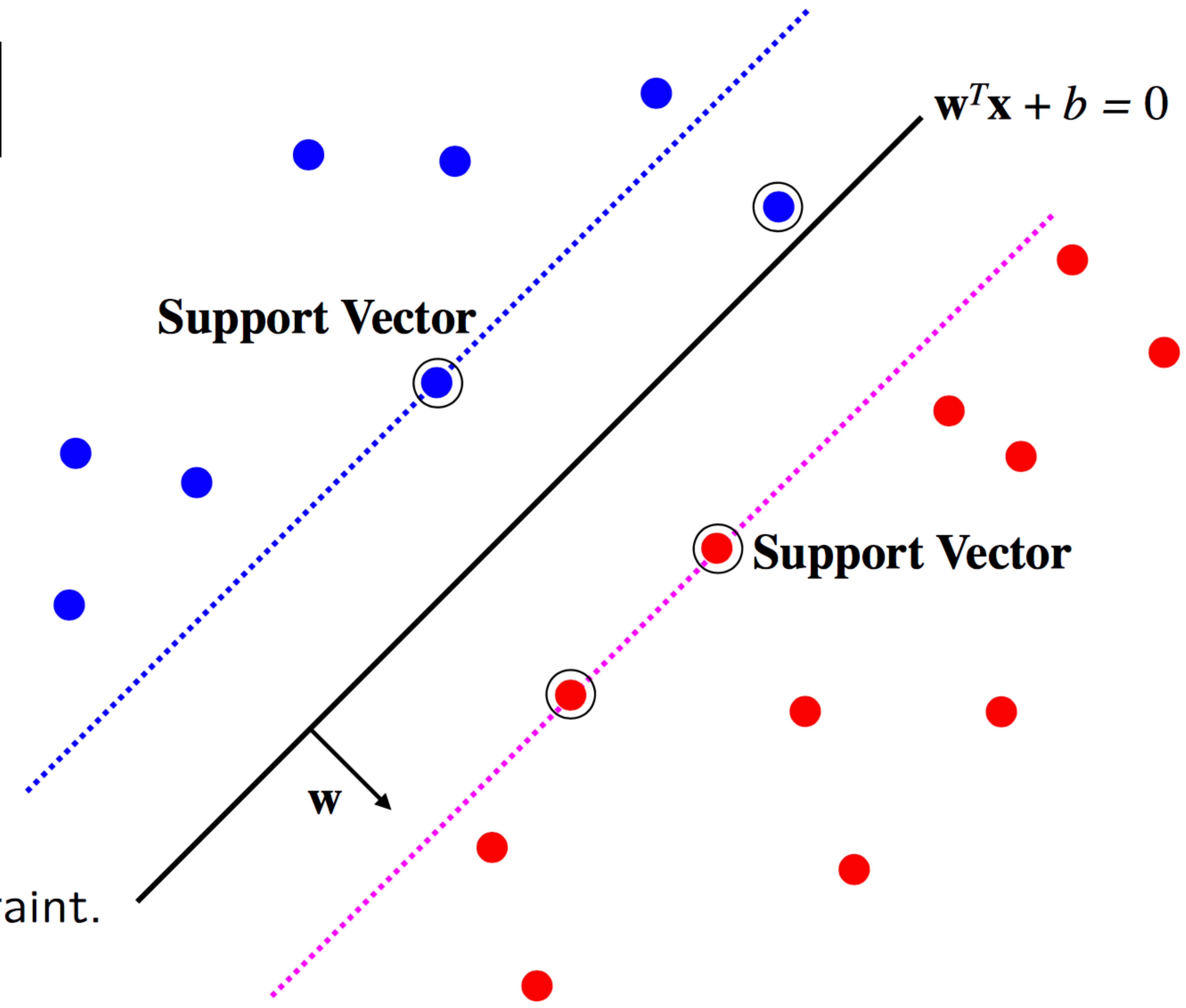
NEW OBJECTIVE

$$\min_{\mathbf{w}} ||\mathbf{w}||^2 + C \sum_i^N \left[\boxed{max\left(0, 1 - y_i f(x_i)\right)} \right]$$

Hinge Loss

Points are in three categories:

1. $y_i f(x_i) > 1$
   Point is outside margin.
   No contribution to loss
2. $y_i f(x_i) = 1$
   Point is on margin.
   No contribution to loss.
   As in hard margin case.
3. $y_i f(x_i) < 1$
   Point violates margin constraint.
   Contributes to loss



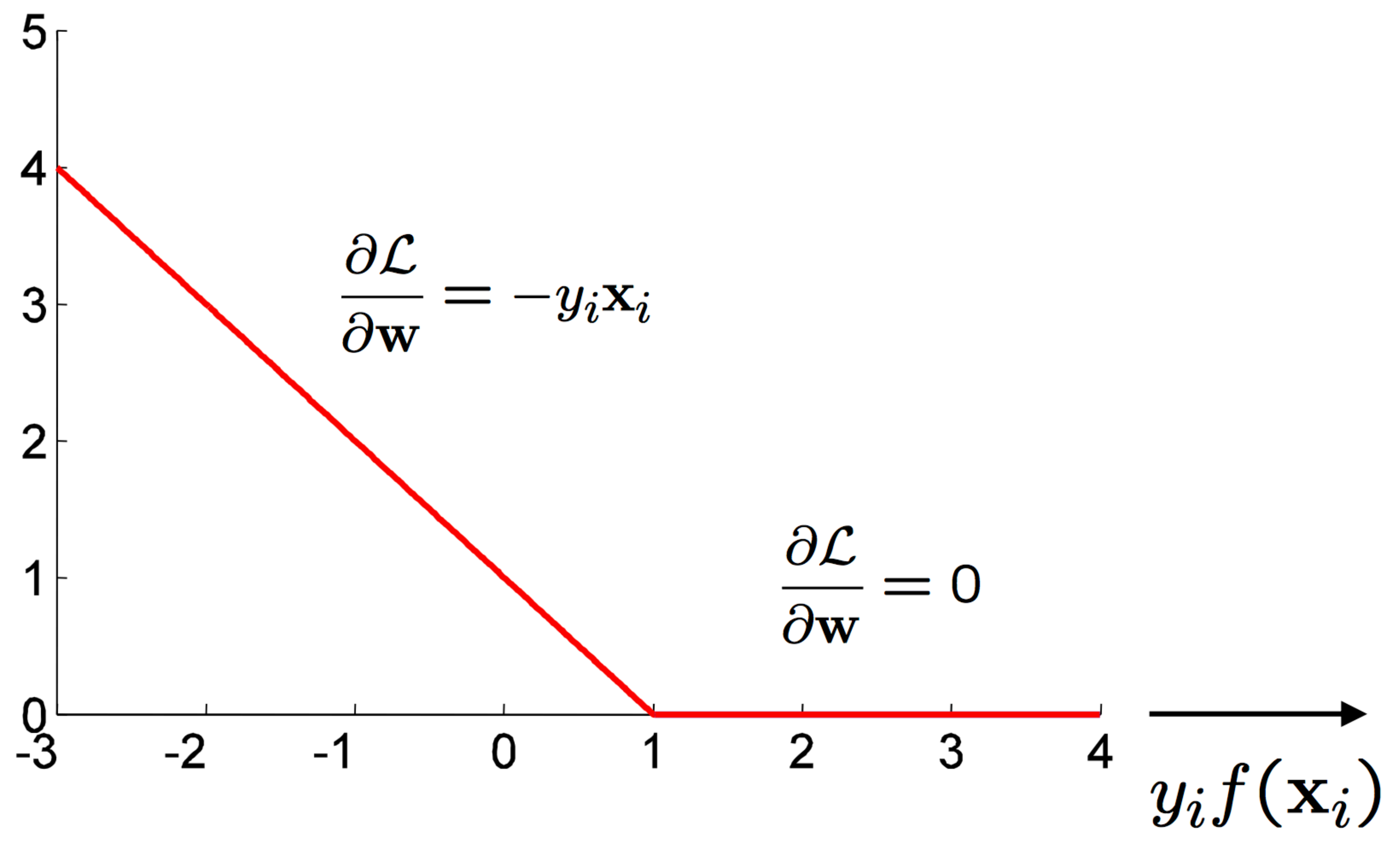
SVM OPTIMIZATION WITH SUB-GRADIENT

- Rewrite optimization problem:

$$\min_{\mathbf{w}} \frac{1}{N} \sum_{i}^{N} \left(\frac{\lambda}{2} ||\mathbf{w}||^2 + \left[max\left(0, 1 - y_i f(x_i)\right)\right]\right)$$

- Now $\lambda = \dfrac{2}{N \cdot C}$, becomes the regularization parameter

- Hinge loss is not differentiable however– so must use sub-gradient for optimization



SUB-GRADIENT DESCENT

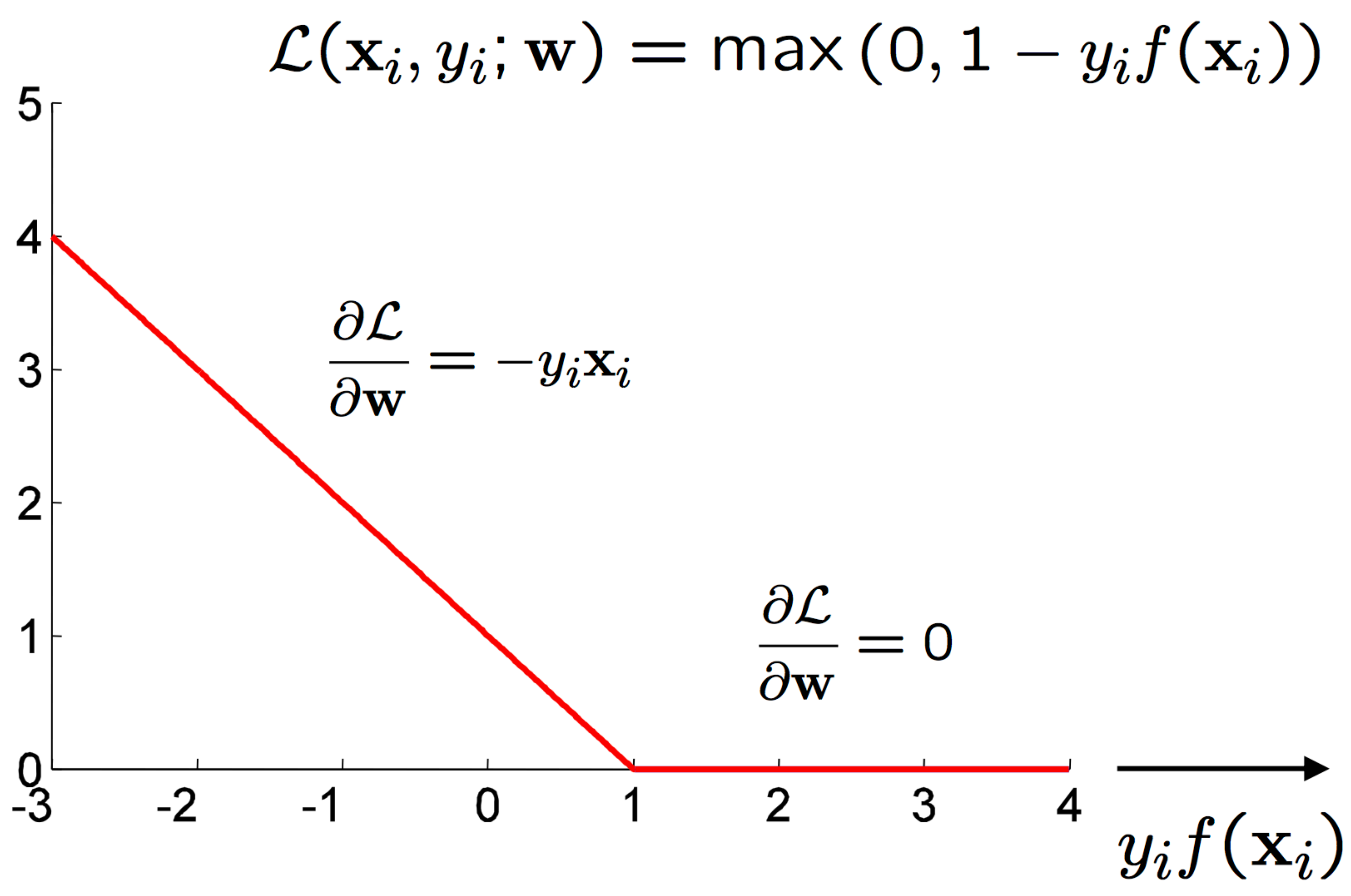
- Iterative update for SVM weights using sub-gradient descent:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{1}{N} \sum_i^N \left(\lambda \mathbf{w}_t + \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t) \right)$$

- where η is the learning rate as per usual

- Each iteration cycles through the data:

$$\begin{aligned} \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \frac{\eta}{N}(\lambda \mathbf{w}_t - y_i \mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\ &\leftarrow \mathbf{w}_t - \frac{\eta}{N} \lambda \mathbf{w}_t && \text{otherwise} \end{aligned}$$



SVM EXAMPLE

Intercept	Age>40	Income=high	Student=yes	Credit=fair	BuysComp?
1	0	1	0	1	-1
1	0	1	0	0	-1
1	0	1	0	1	+1
1	1	0	0	1	+1
1	1	0	1	1	+1
1	1	0	1	0	-1
1	0	0	1	0	+1
1	0	0	0	1	-1
1	0	0	1	1	+1
1	1	0	1	1	+1
1	0	0	1	0	+1
1	0	0	0	0	+1
1	0	1	1	1	+1
1	1	0	0	0	-1

$$BC = +1 \quad \text{if} \quad \left[\mathbf{w}^T\mathbf{x}\right] > 0$$
$$BC = -1 \quad \text{otherwise}$$

$$\mathbf{x} = [Int, A, I, S, CR]$$
$$\mathbf{w} = [w_0, w_A, w_I, w_S, w_{CR}]$$

SVM parameters = w

- Score function: margin + hinge loss on errors
- Estimate **w** to maximize margin, while minimizing errors

SVM LEARNING

- Score function: soft margin (includes hinge loss on errors)

$$\min_{\mathbf{w}} \frac{1}{N}\sum_{i}^{N}\left(\frac{\lambda}{2}||\mathbf{w}||^2 + \left[max\left(0, 1 - y_i f(x_i)\right)\right]\right)$$

- Estimate **w** by minimizing objective function using gradient descent

Gradient descent:

Start at some **w**, e.g., **w**=[0,0,0,0,0]; make predictions given current **w**: $\forall i \;\; \widehat{y}_i = \mathbf{w}^T \mathbf{x_i}$

Calculate gradient for each parameter: $\nabla_j = \frac{1}{N}\sum_{i=1}^{N} -\nabla_{ji}$ (intercept)

or $\nabla_j = \frac{1}{N}\left[\sum_{i=1}^{N}(\lambda w_j - \nabla_{ji})\right]$ (non-intercept)

where $\nabla_{ji} = y_i x_{ij}$ if $y_i \widehat{y}_i < 1;\; 0$ otherwise

Move parameters in direction of gradient: $\forall j \;\; w_j^{new} = w_j - \eta \nabla_j$

Repeat until stopping criteria is met

SVM PREDICTION

Intercept	Age>40	Income=high	Student=yes	Credit=fair	BuysComp?
1	0	1	0	1	-1
1	0	1	0	0	-1
1	0	1	0	1	+1
1	1	0	0	1	+1
1	1	0	1	1	+1
1	1	0	1	0	-1
1	0	0	1	0	+1
1	0	0	0	1	-1
1	0	0	1	1	+1
1	1	0	1	1	+1
1	0	0	1	0	+1
1	0	0	0	0	+1
1	0	1	1	1	+1
1	1	0	0	0	-1
1	**0**	**1**	**0**	**0**	**?**

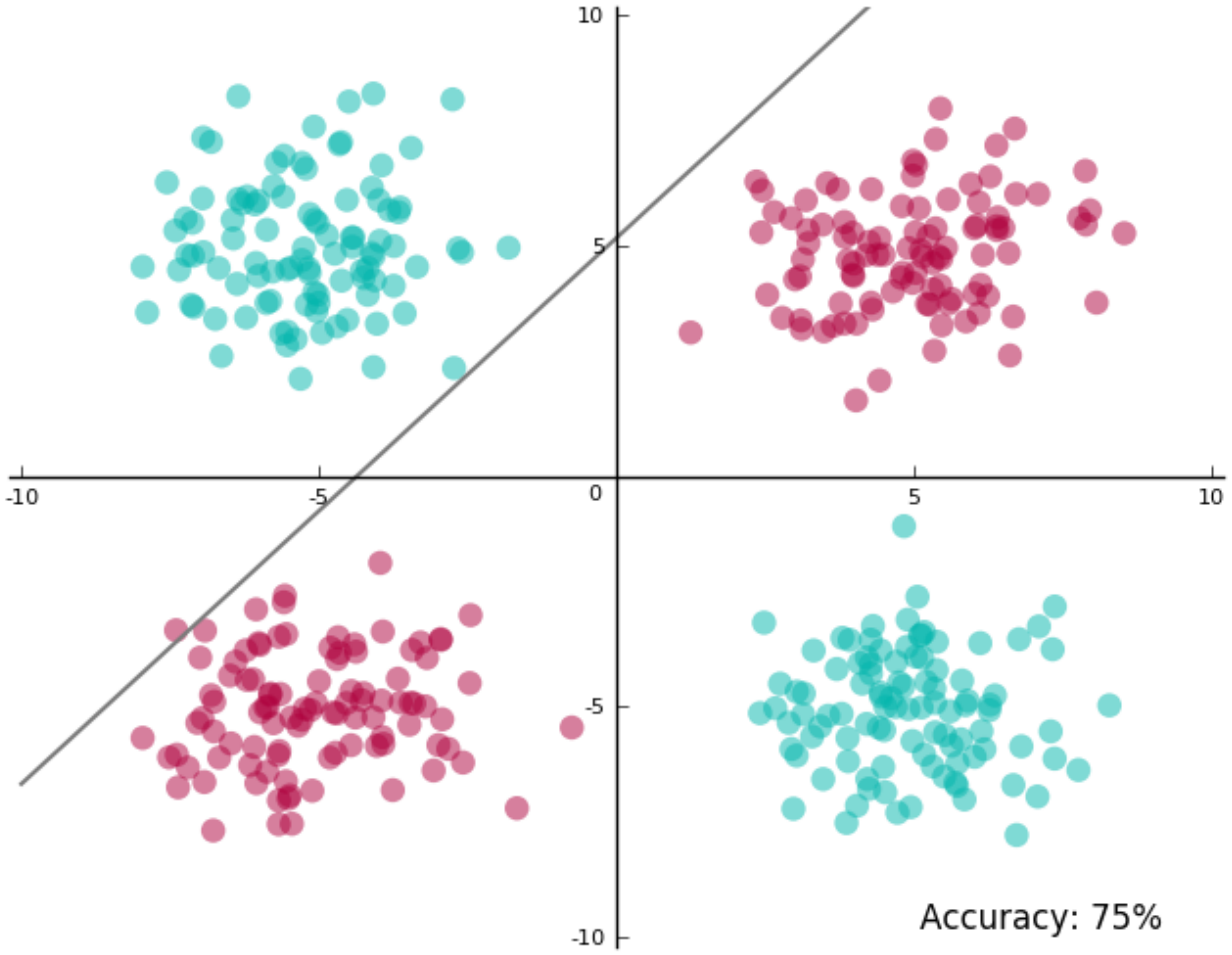
- What is the probability that new person will buy a computer?

$$\mathbf{x} = [1, 0, 1, 0, 0]$$
$$\mathbf{w} = [-.5, 1.2, 3, -2, 0.7]$$

$$\mathbf{x}^T\mathbf{w} = 2.5$$

$$BC = +1$$

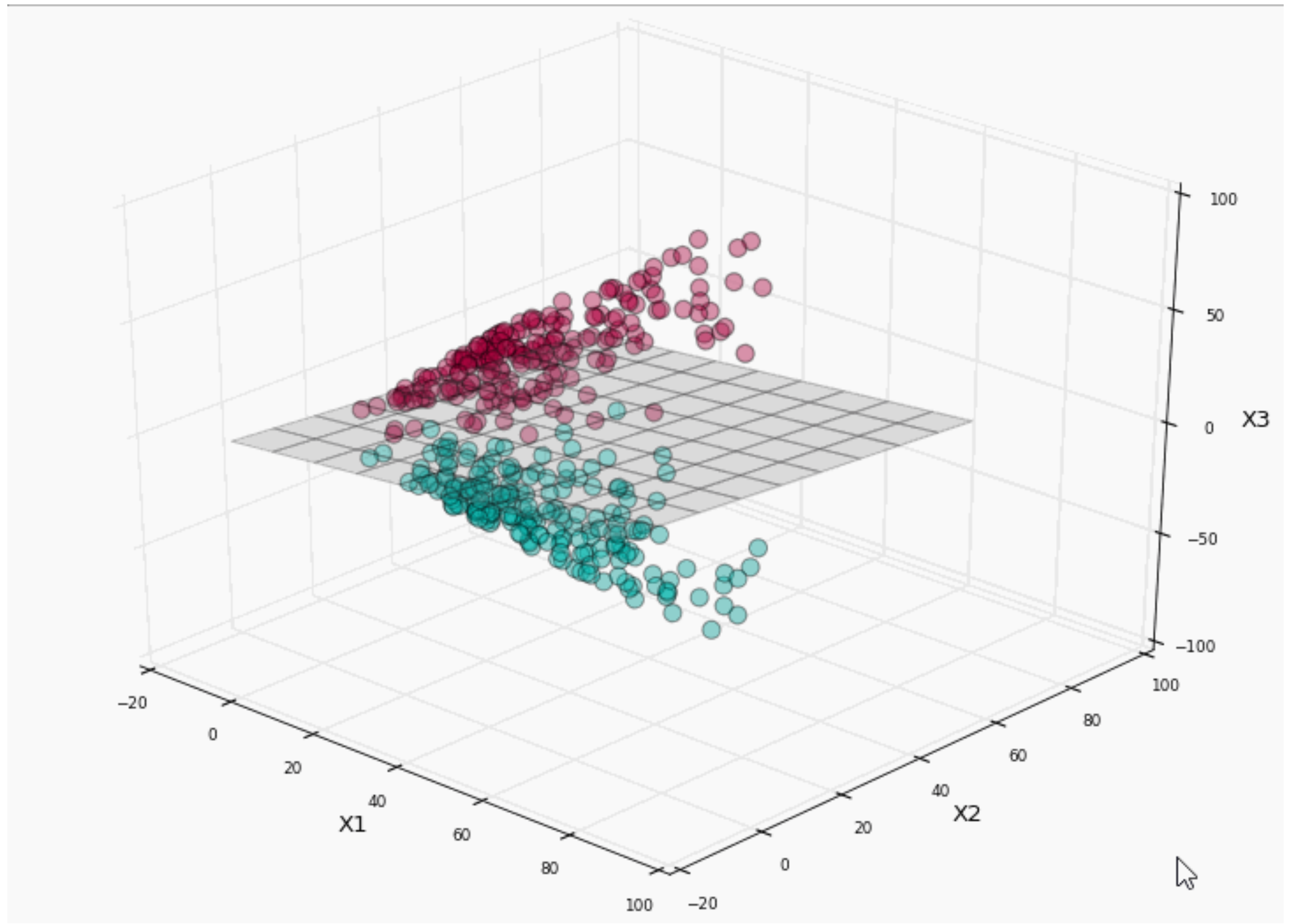
BEYOND LINEAR SVM

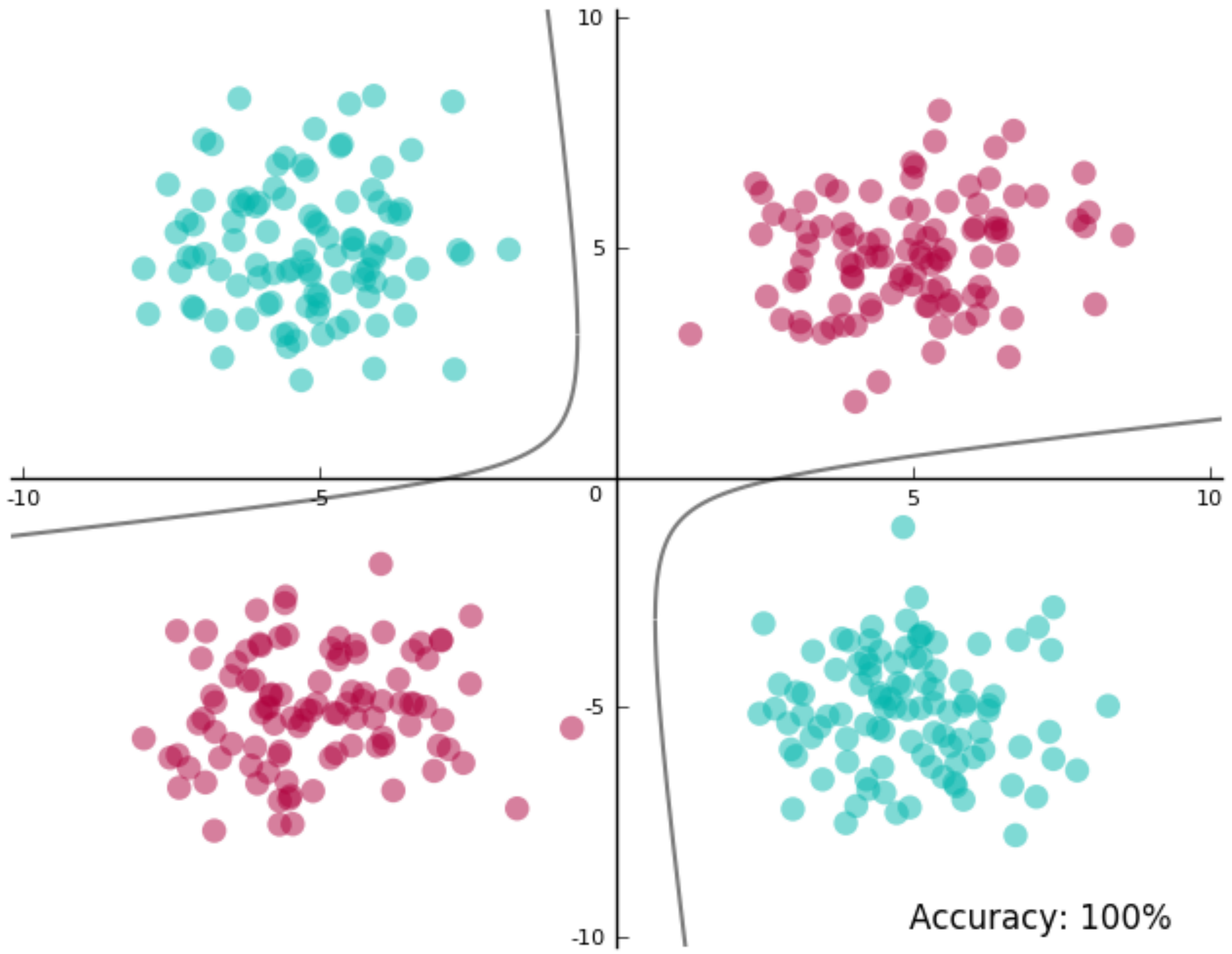


Hardly linearly-separable!

PROJECTING TO HIGHER DIMENSIONAL SPACE

- Data that is not linearly separable in lower-dimensional space is more likely to be linearly separable when projected onto higher dimensions
- $X_1 = x_1^2, X_2 = x_2^2, X_3 = \sqrt{2}x_1x_2$





EMPOWERING SVM

- Project data into a higher-dimensional space
- Find a hyperplane in the higher-dimensional space that can almost linearly separate the training examples
- Project the hyperplane back to the original lower-dimensional space to get the non-linear decision boundary!
- Which higher-dimensional space should I project the data into?

THE KERNEL TRICK

- You only need to know the dot products between data points to learn SVM and make prediction with SVM (related to primal-dual of optimization problems)
    - Given a training dataset, you only need to know $\mathbf{x}_i^T\mathbf{x}_j$ for any two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ in the training example to learn the linear SVM
    - After a linear SVM is learned, given a test data point $\mathbf{x}$, you only need to know $\mathbf{x}^T\mathbf{x}_i$ for all the data points $\mathbf{x}_i$ in the training example to make predictions
- Given a projection function $\mathbf{x} \rightarrow \phi(\mathbf{x})$
    - The linear SVM in the higher-dimensional space can be learned and used as long as we know $\phi(\mathbf{x})^T\phi(\mathbf{y})$

THE KERNEL TRICK

- Use **kernel function** to compute dot products in higher-dimensional space in the original lower-dimensional space: $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$
- Example: $\mathbf{x} = (x_1, x_2); \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
  - $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$

$$= (x_1y_1 + x_2y_2)^2$$

$$= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2$$

$$= \phi(\mathbf{x})\phi(\mathbf{y})$$

KERNEL SVM

- Different kernel functions:

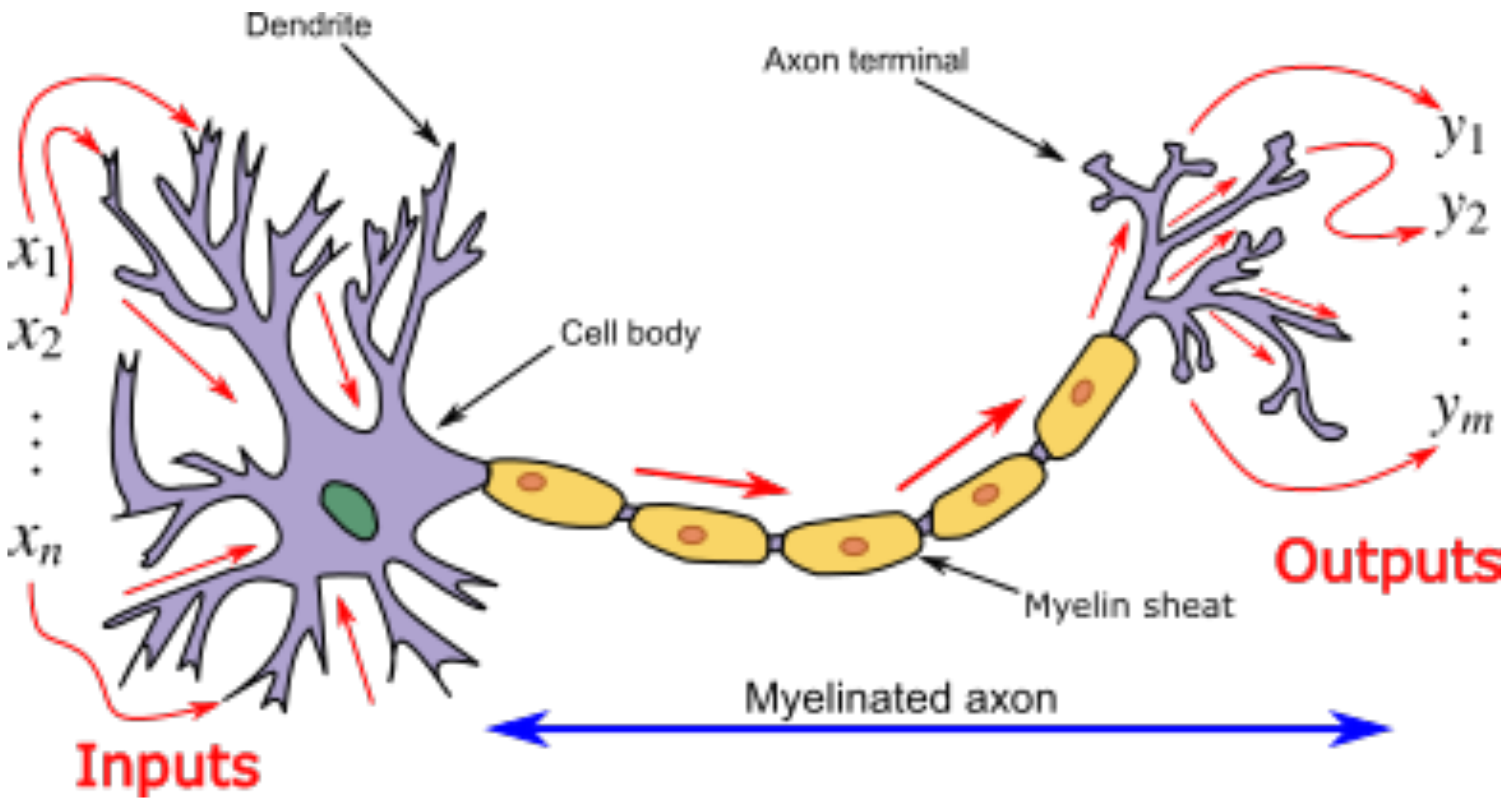
name	$k(\mathbf{x}, \mathbf{v})$
Linear	$\mathbf{x} \cdot \mathbf{v}$
Polynomial	$(r + \mathbf{x} \cdot \mathbf{v})^d$, for some $r \geq 0, d > 0$
Radial Basis Function	$\exp\left(-\gamma \|\|\mathbf{x} - \mathbf{v}\|\|^2\right)$, $\gamma > 0$
Gaussian	$\exp\left(-\frac{1}{2\sigma^2} \|\|\mathbf{x} - \mathbf{v}\|\|^2\right)$

- Kernel SVM
  - Decide upon a kernel function k(**x**, **v**)
  - Use this kernel function to compute $k(\mathbf{x}_i, \mathbf{x}_j)$ for all $\mathbf{x}_i$ and $\mathbf{x}_j$ in the training example and learn the linear SVM in the high-dimensional space
  - Given the learned linear SVM (in the high-dimensional space) and a new data point **x**, compute $k(\mathbf{x}, \mathbf{x}_i)$ for all the data points $\mathbf{x}_i$ in the training example to make predictions

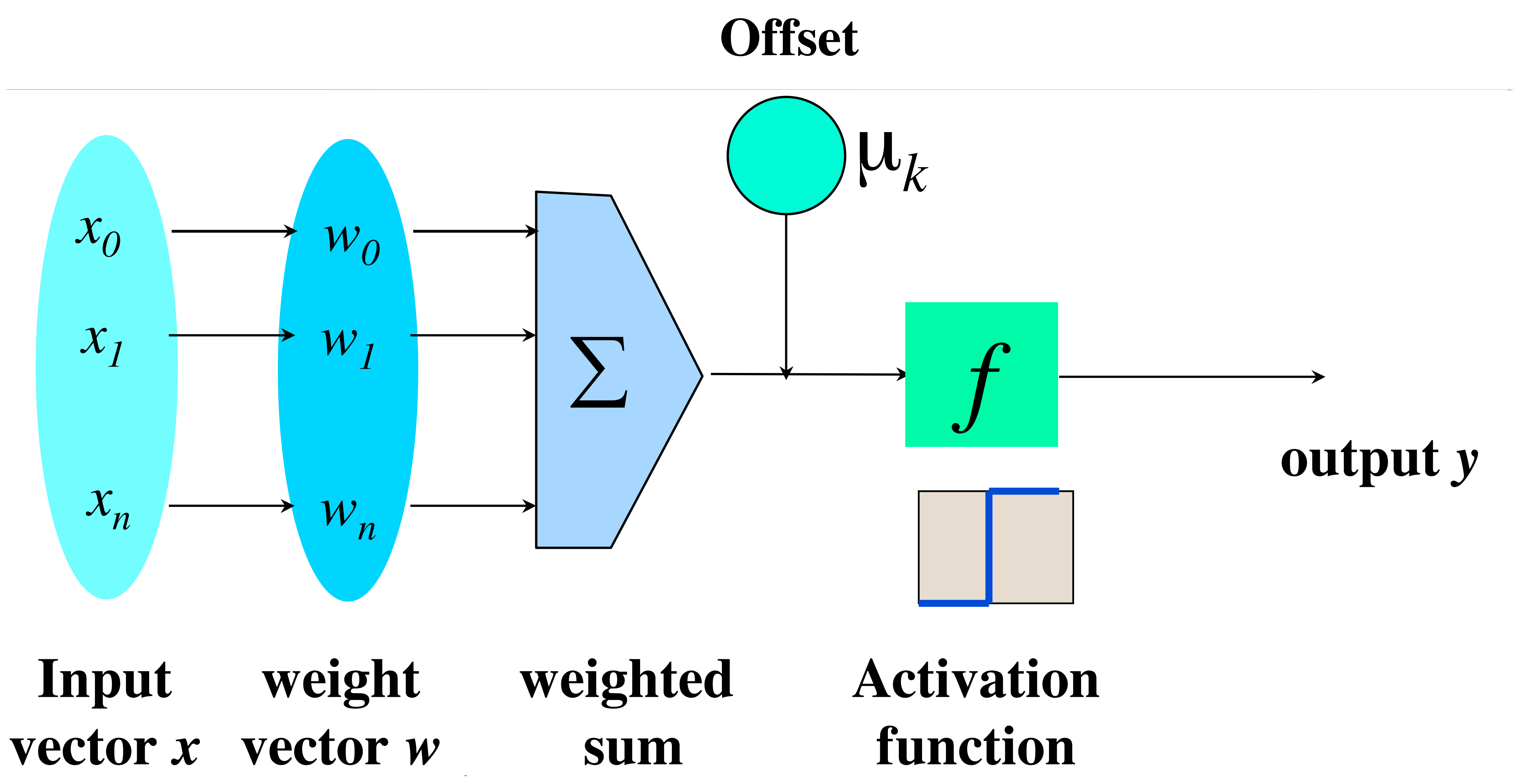
NEURAL NETWORK

NEURAL NETWORKS

- Analogous to biological systems
- Build artificial neurons to transfer inputs to outputs
- Outputs of a neuron can be “transmitted” and serve as inputs for other neurons



NEURON


Offset
μ_k
x_0
x_1
x_n
w_0
w_1
w_n
Σ
f
output y
Input vector x
weight vector w
weighted sum
Activation function

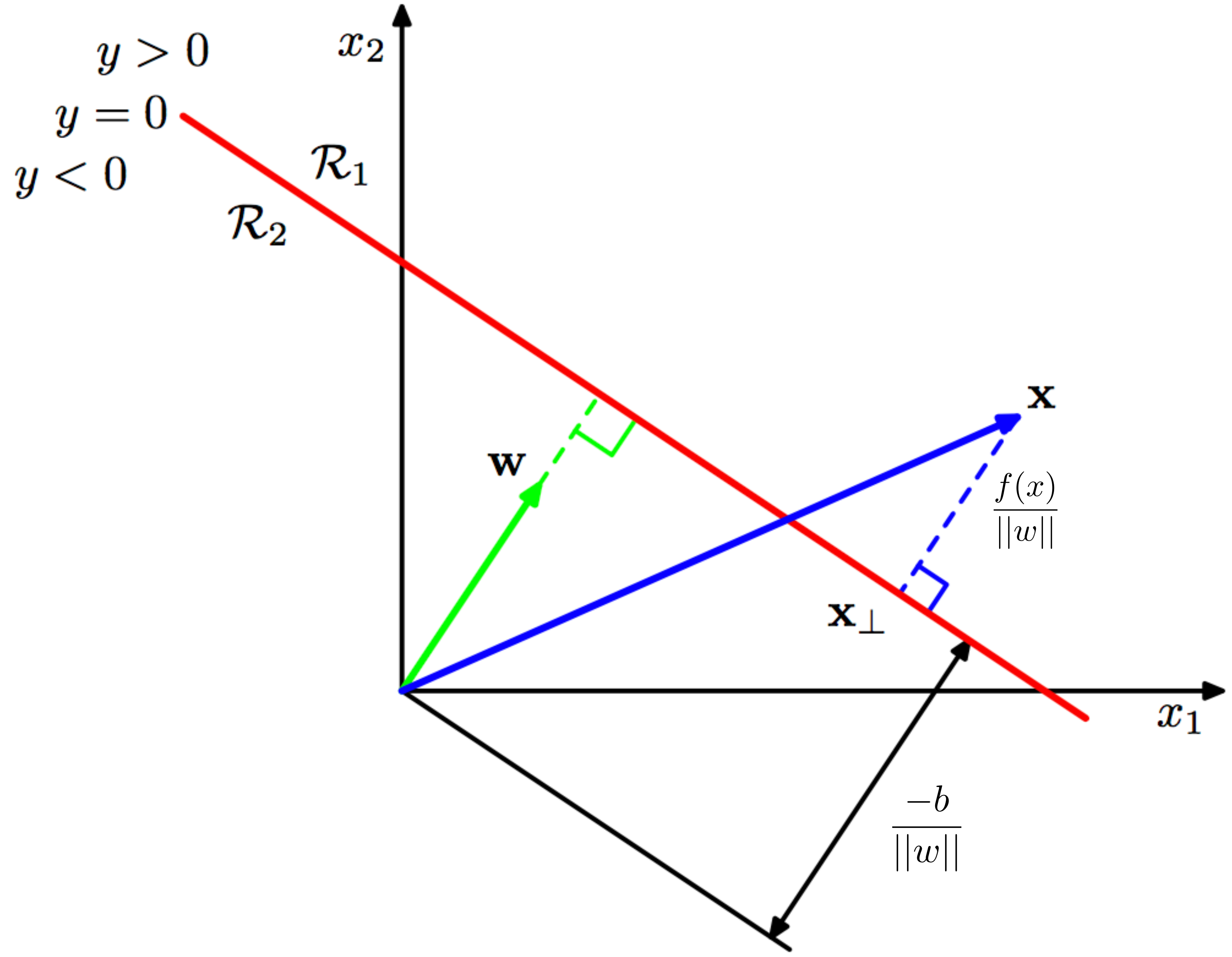
SIMPLEST NEURON NETWORK: PERCEPTRON

First introduced in late 1950s by Minsky and Papert

Model: $$f(x) = \sum_{i=1}^{m} w_i x_i + b$$ *Offset*

$$y = sign[f(x)]$$

Activation function



Model space: All possible weights ***w*** and *b*

Figure: C. Bishop

PERCEPTRON COMPONENTS

- **Model space**
  - Set of weights ***w*** and b (hyperplane boundary)
- **Score function**
  - Minimize misclassification rate
- **Search algorithm**
  - Iterative refinement of ***w*** and *b*

PERCEPTRON LEARNING

Model: $$f(x) = \sum_{i=1}^{m} w_i x_i + b$$

$$y = sign[f(x)]$$

Learning: $$if \;\; y(j)(\sum_{i=1}^{m} w_i x_i(j) + b) \leq 0$$

$$then \;\; w \leftarrow w + \eta y(j) x(j) \;\; (0 < \eta \ll 1)$$

Iterate over training examples for fixed number of iterations or until error is below a pre-specified threshold

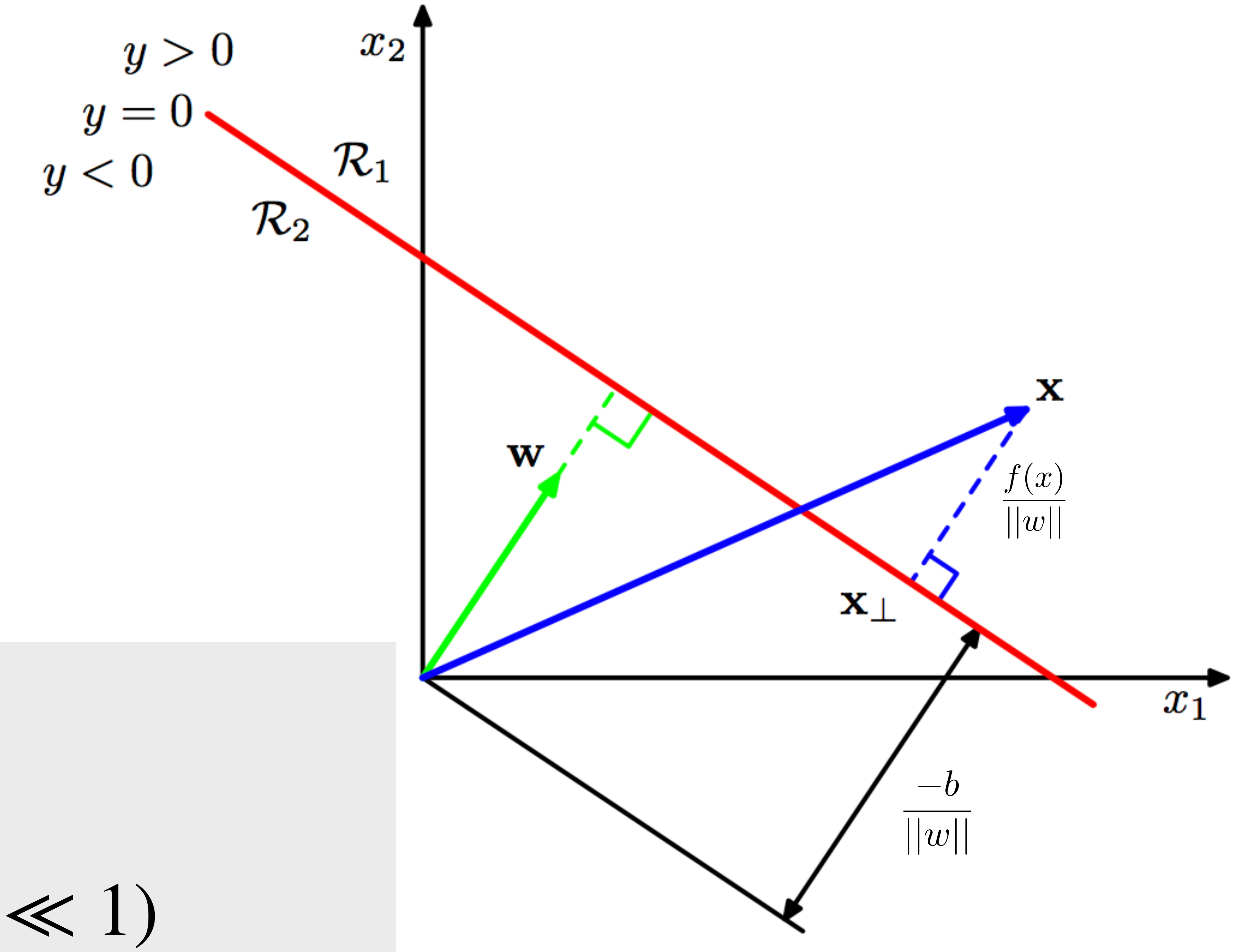
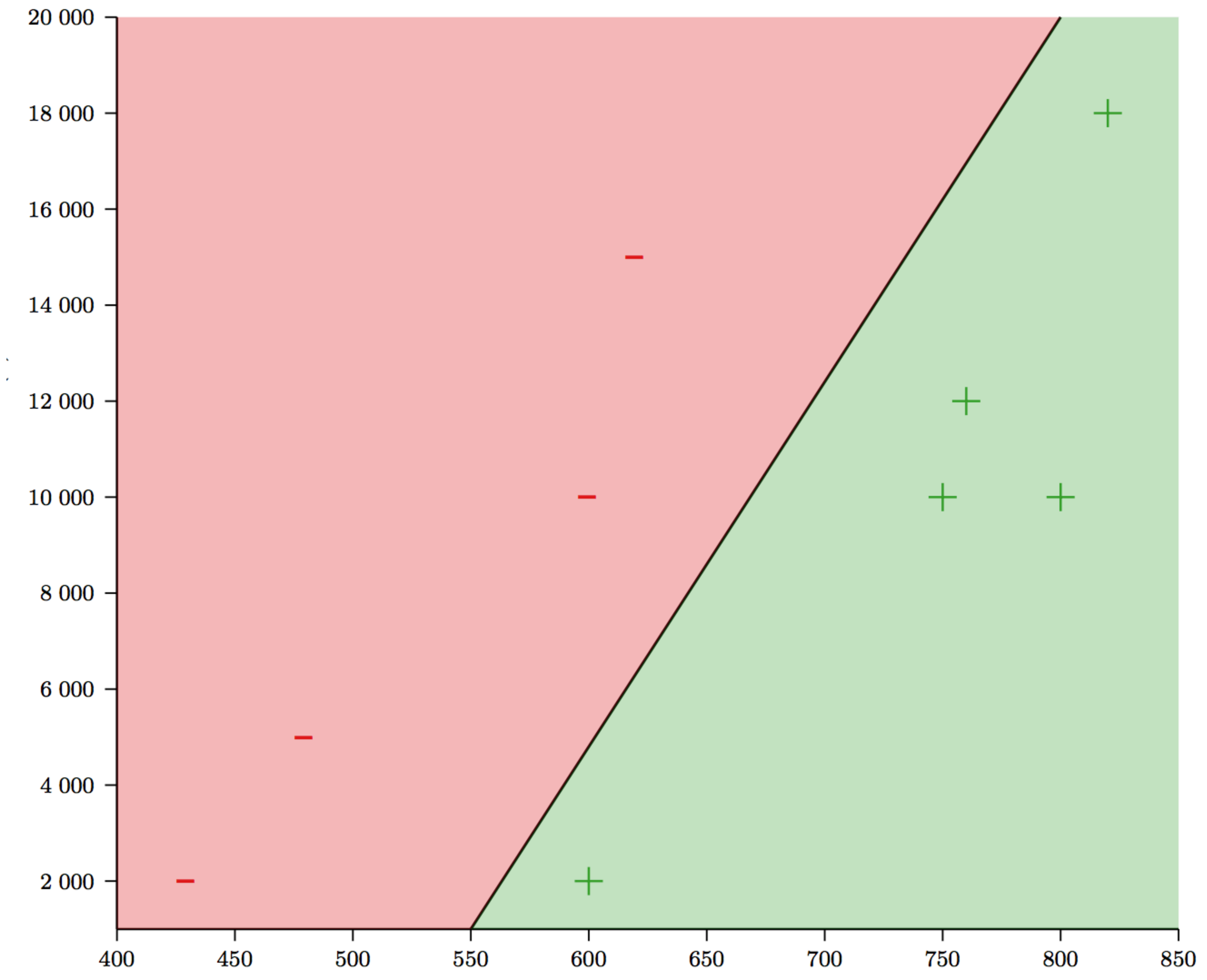


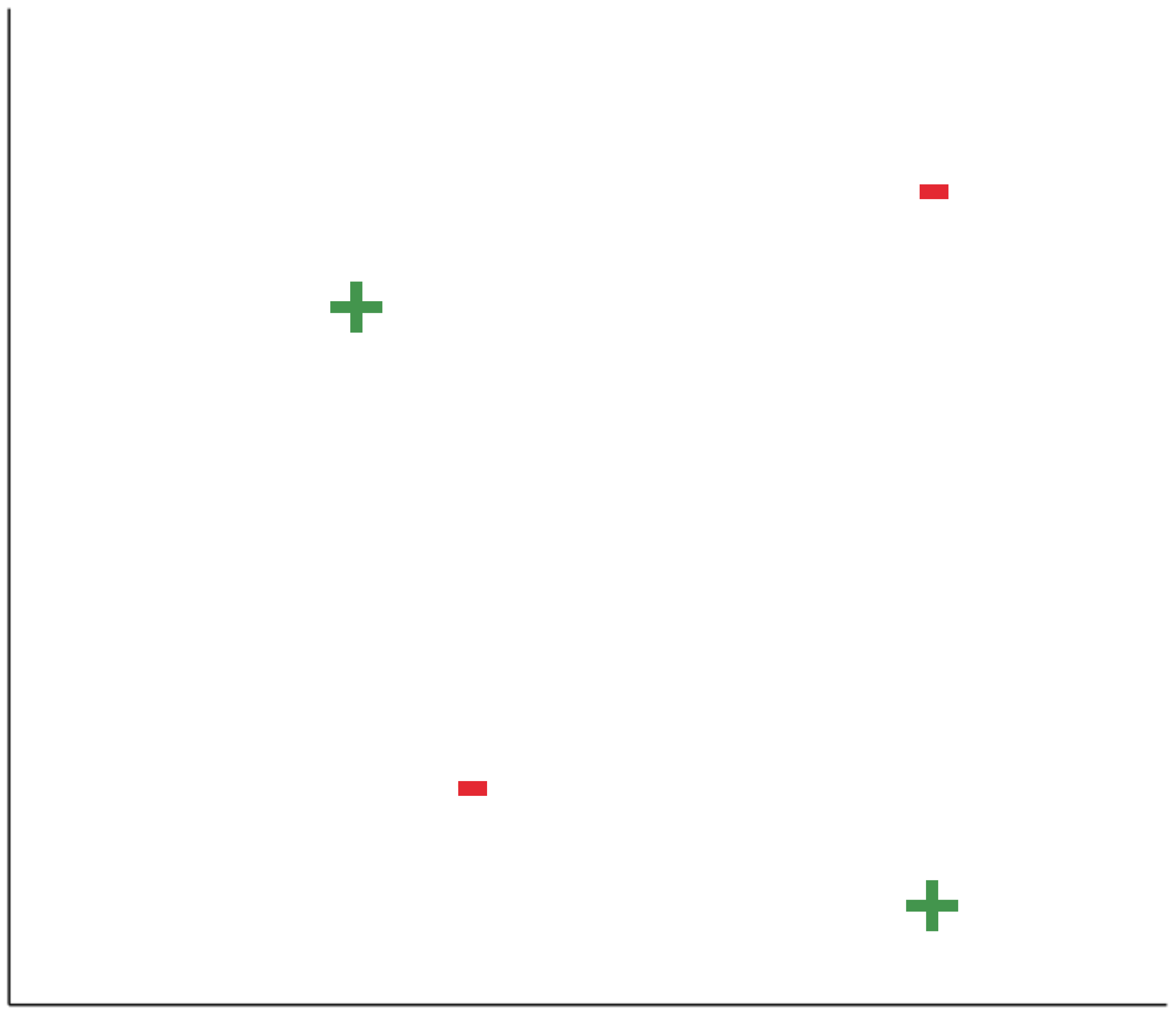
Figure: C. Bishop

```
procedure LearnPerceptron(data,numIters,learnRate)
    w ← 0 (for p = 1...numAttrs)
    b ← 0
    η ← learnRate
    for iter ≤ numIters do
        for i = (x_i, y_i) ∈ data do
            ŷ_i = sign(w · x_i + b)
            if y_i ŷ_i ≤ 0 then
                e = η y_i
                w ← w^old + e x_i
                b ← b + e
            end if
        end for
    end for
    return w, b
end procedure
```

LIMITATION OF PERCEPTRON

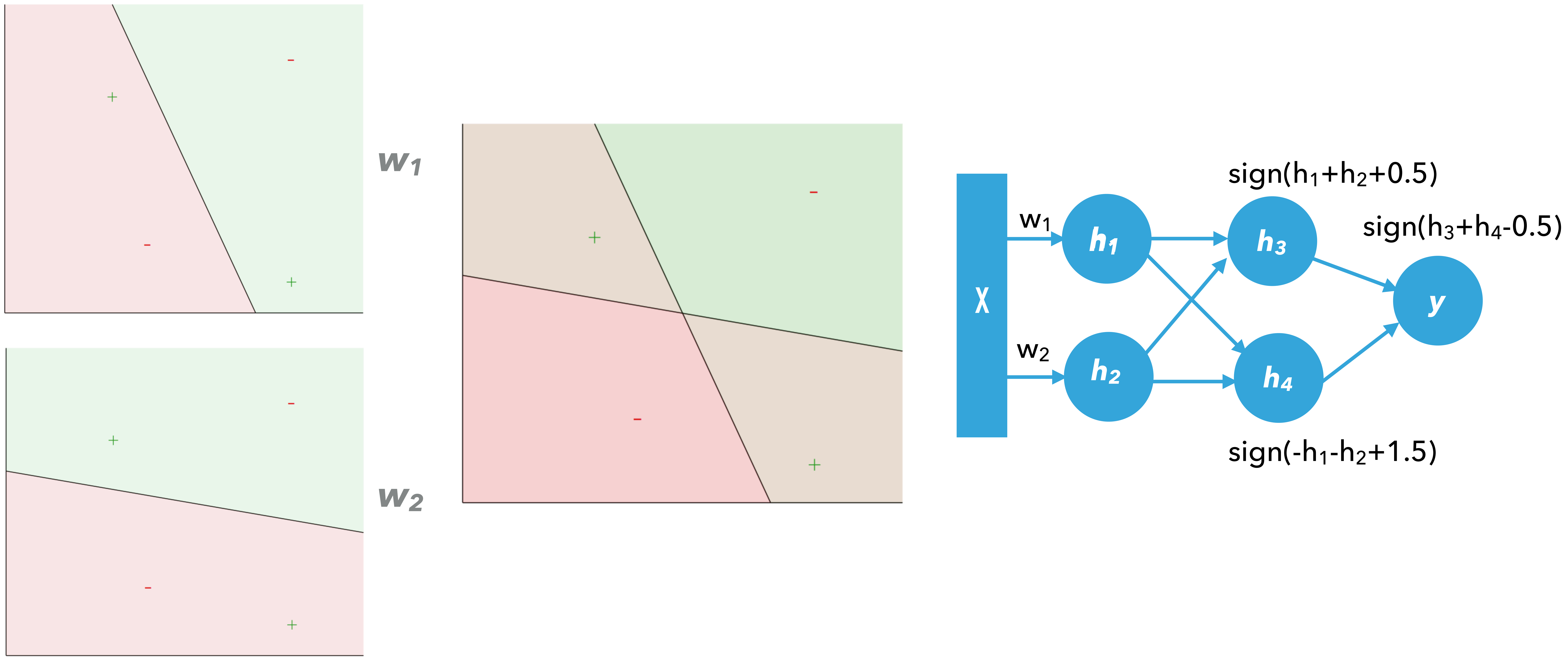


How about this?



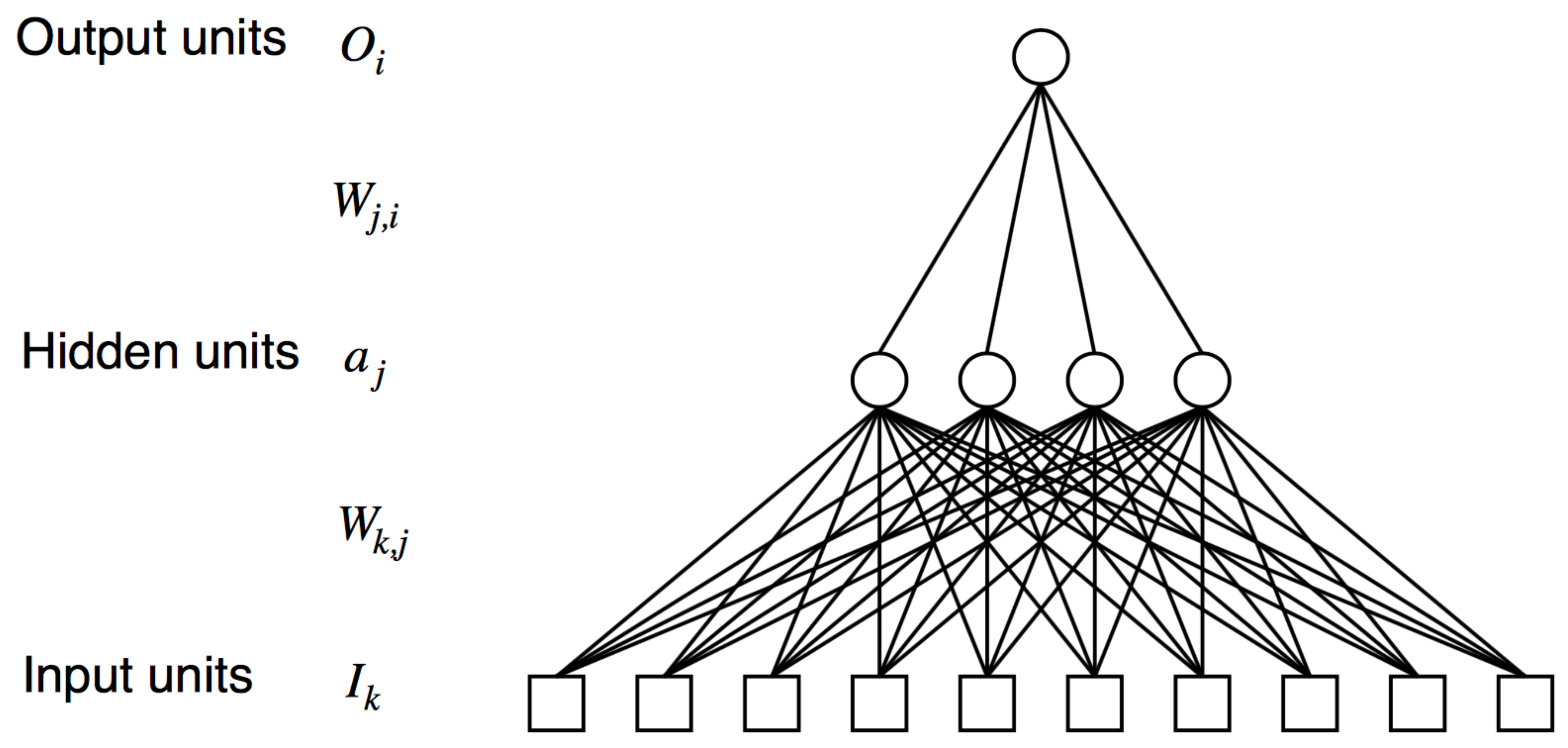
Perceptron is suitable for classifying a set of linearly separable data

FROM PERCEPTRON TO MULTI-LAYER NEURAL NETWORKS



MULTI-LAYER NEURAL NETWORK

- Increase expressive power by combining multiple perceptrons into ensemble
- Two-layer neural network: each perceptron output is a hidden unit, which are then aggregated into a final output



Output $$O_i = g(\sum_j W_{j,i} a_j)$$

Hidden units $$a_j = g(\sum_k W_{k,j} I_k)$$

Figure: M. Velosa