CS57300
PURDUE UNIVERSITY

OCTOBER 18, 2021

DATA MINING

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: $\mathbf{x} = (x_1, x_2); \phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
  - $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2$

$$
\begin{aligned}
&= (x_1y_1 + x_2y_2)^2 \\
&= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 \\
&= \phi(\mathbf{x})\phi(\mathbf{y})
\end{aligned}
$$

KERNEL SVM

- Different kernel functions:

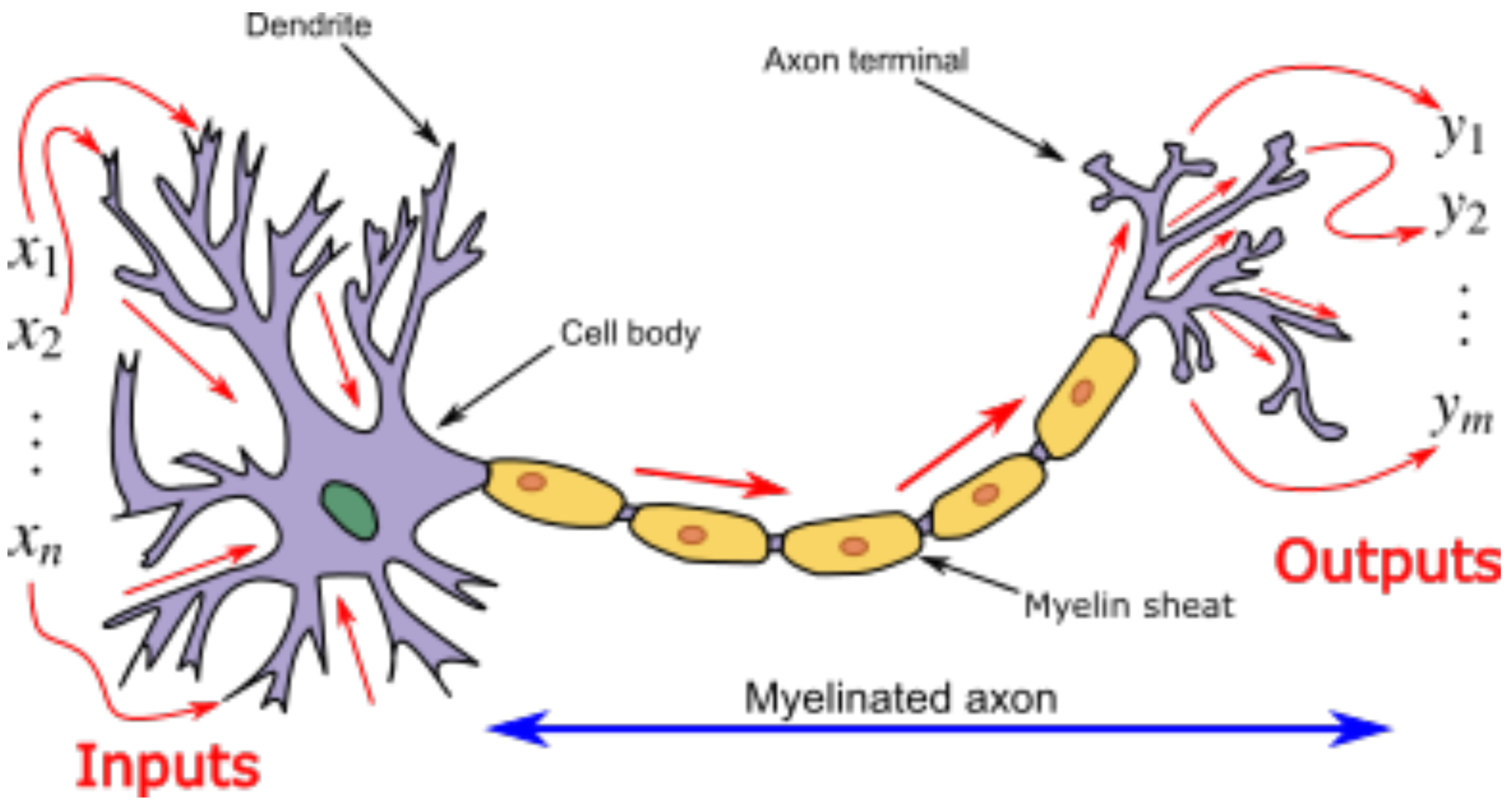
name	$k(\mathbf{x}, \mathbf{v})$
Linear	$\mathbf{x} \cdot \mathbf{v}$
Polynomial	$(r + \mathbf{x} \cdot \mathbf{v})^d$, for some $r \geq 0, d > 0$
Radial Basis Function	$\exp\left(-\gamma \|\|\mathbf{x} - \mathbf{v}\|\|^2\right)$, $\gamma > 0$
Gaussian	$\exp\left(-\frac{1}{2\sigma^2} \|\|\mathbf{x} -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

- Kernel SVM
  - Decide upon a kernel function k($\mathbf{x}$, $\mathbf{v}$)
  - 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 and a new data point $\mathbf{x}$, compute k($\mathbf{x}$, $\mathbf{x}_i$) for all the data points $\mathbf{x}_i$ in the training example to make predictions

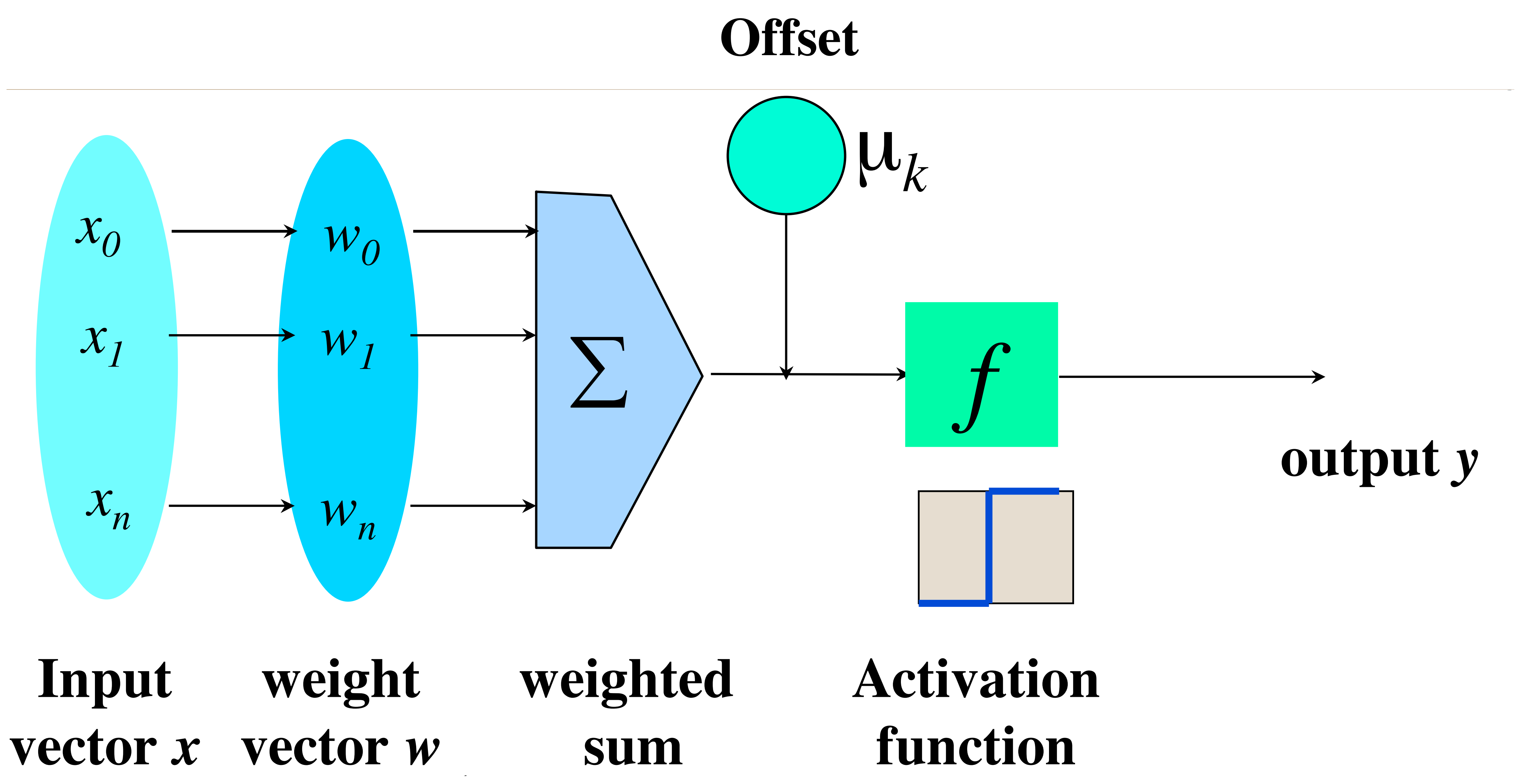
NEURAL NETWORK

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
μk
x0
x1
xn
w0
w1
wn
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

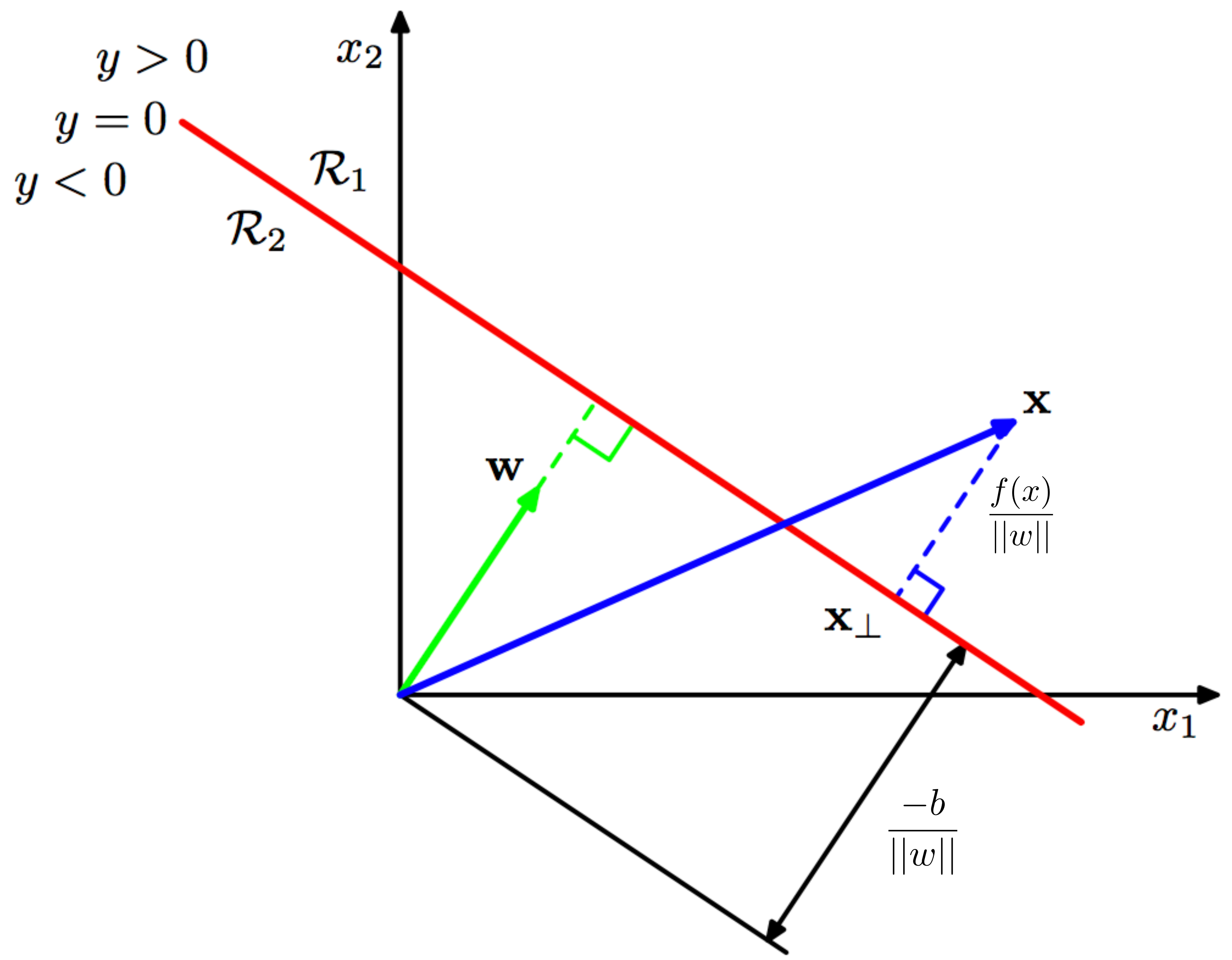


Figure: C. Bishop

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

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

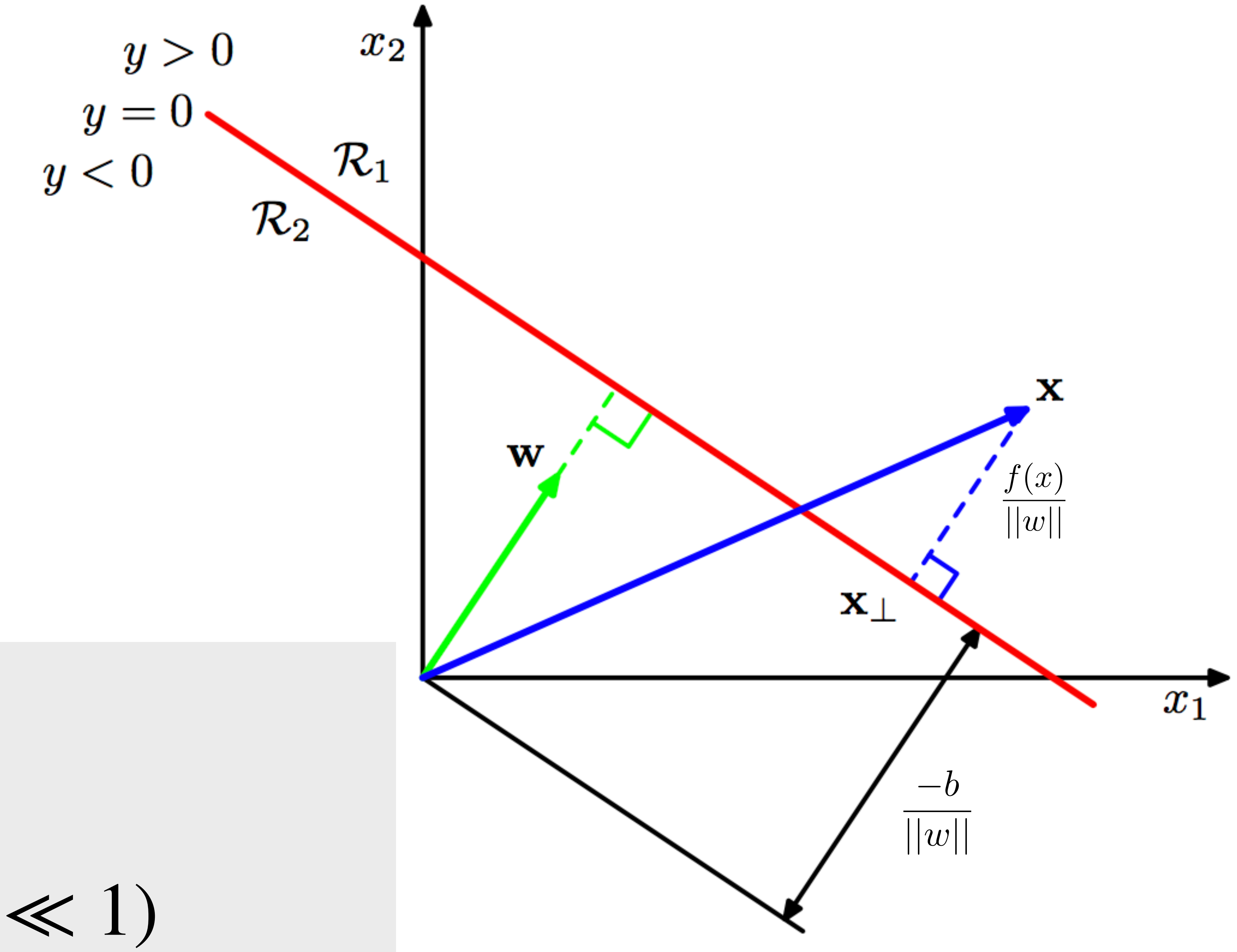
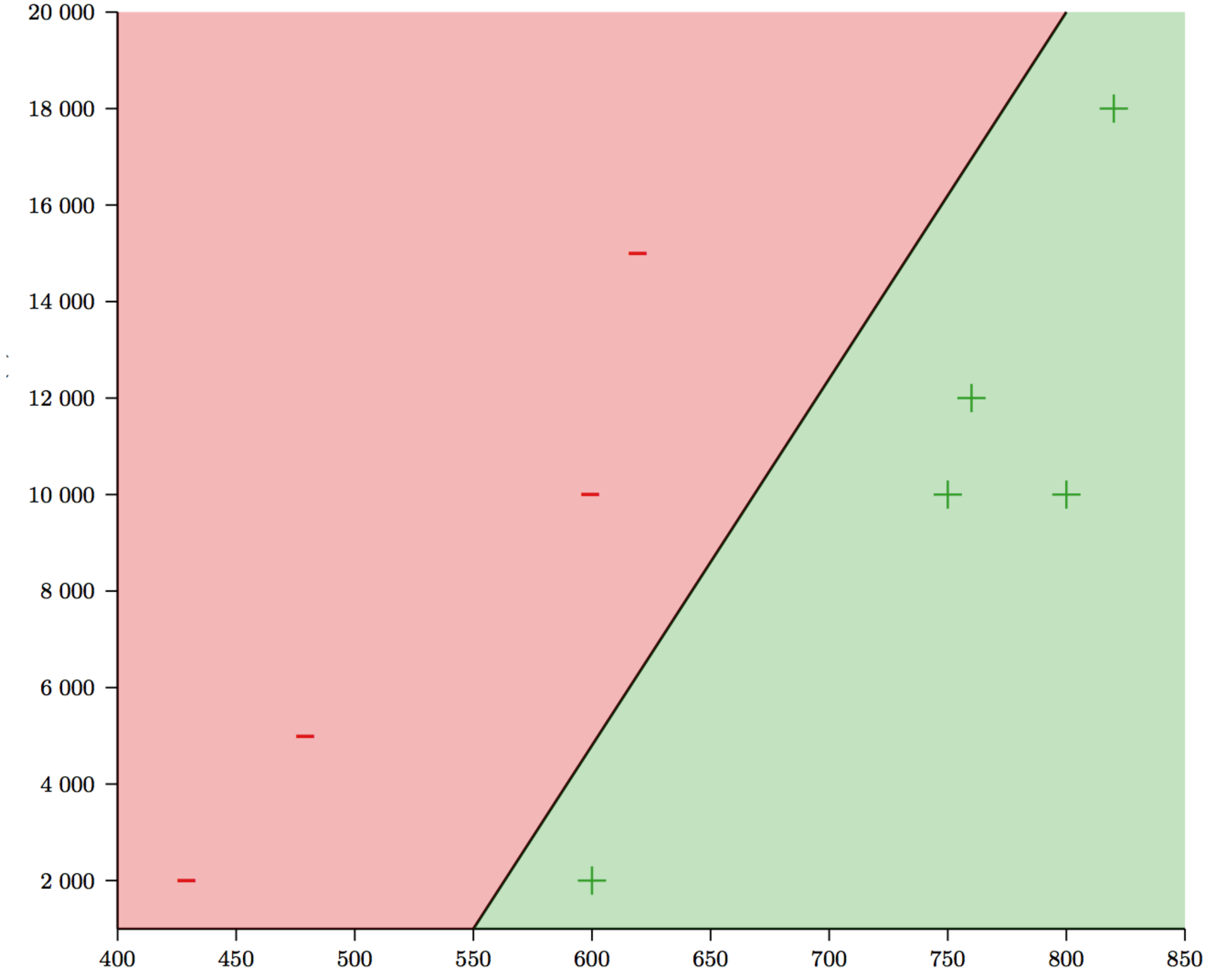


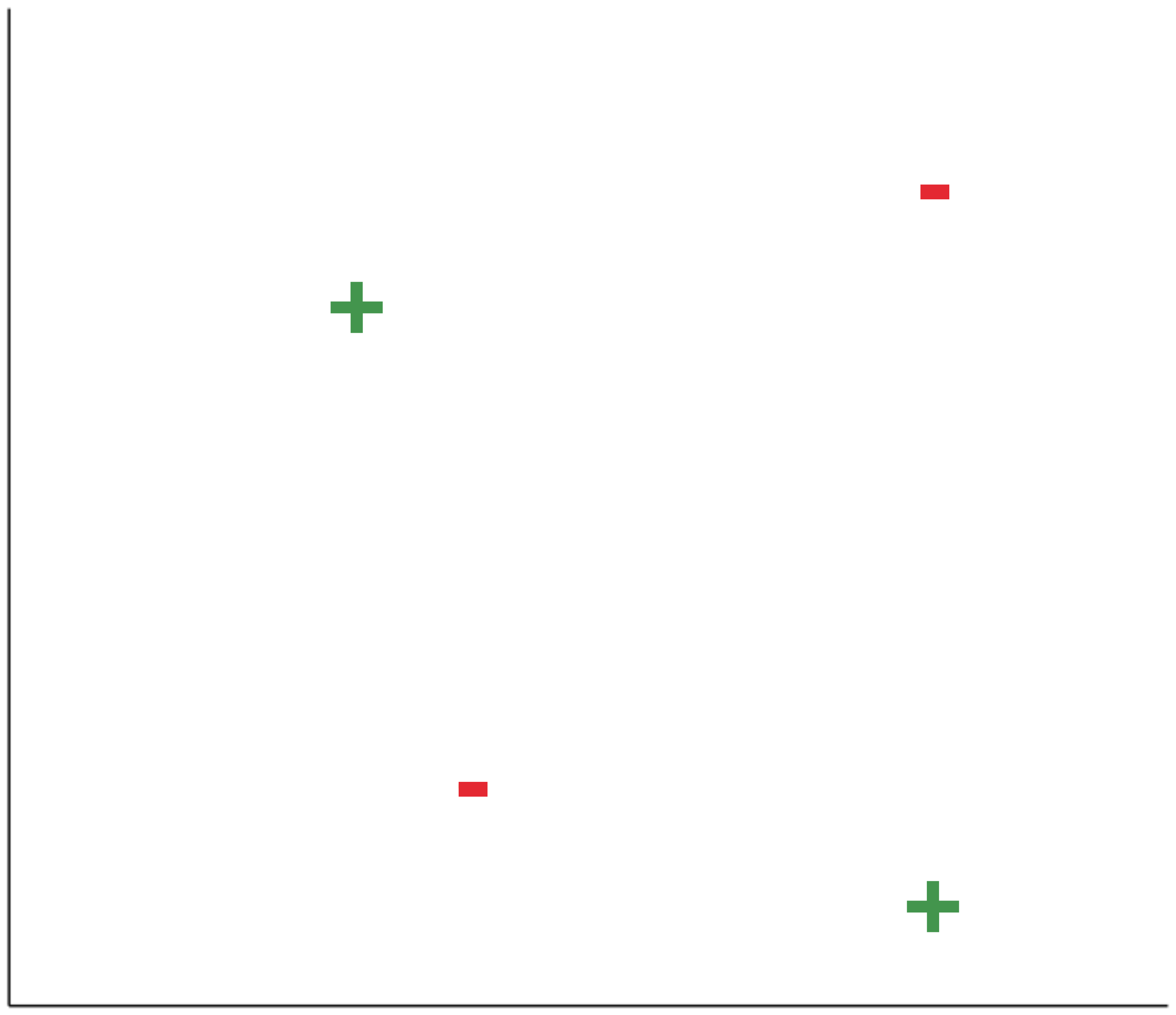
Figure: C. Bishop

procedure LEARNPERCEPTRON(data,numIters,learnRate)
  $\mathbf{w} \leftarrow 0$ (for $p = 1 ... numAttrs$)
  $b \leftarrow 0$
  $\eta \leftarrow learnRate$
  for $iter \leq numIters$ **do**
    for $i = (\mathbf{x}_i, y_i) \in data$ **do**
      $\hat{y}_i = sign(\mathbf{w} \cdot \mathbf{x}_i + b)$
      if $y_i \hat{y}_i \leq 0$ **then**
        $e = \eta y_i$
        $\mathbf{w} \leftarrow \mathbf{w}^{old} + e\, \mathbf{x}_i$
        $b \leftarrow b + e$
      end if
    end for
  end for
  return $\mathbf{w}, b$
end procedure

LIMITATION OF PERCEPTRON

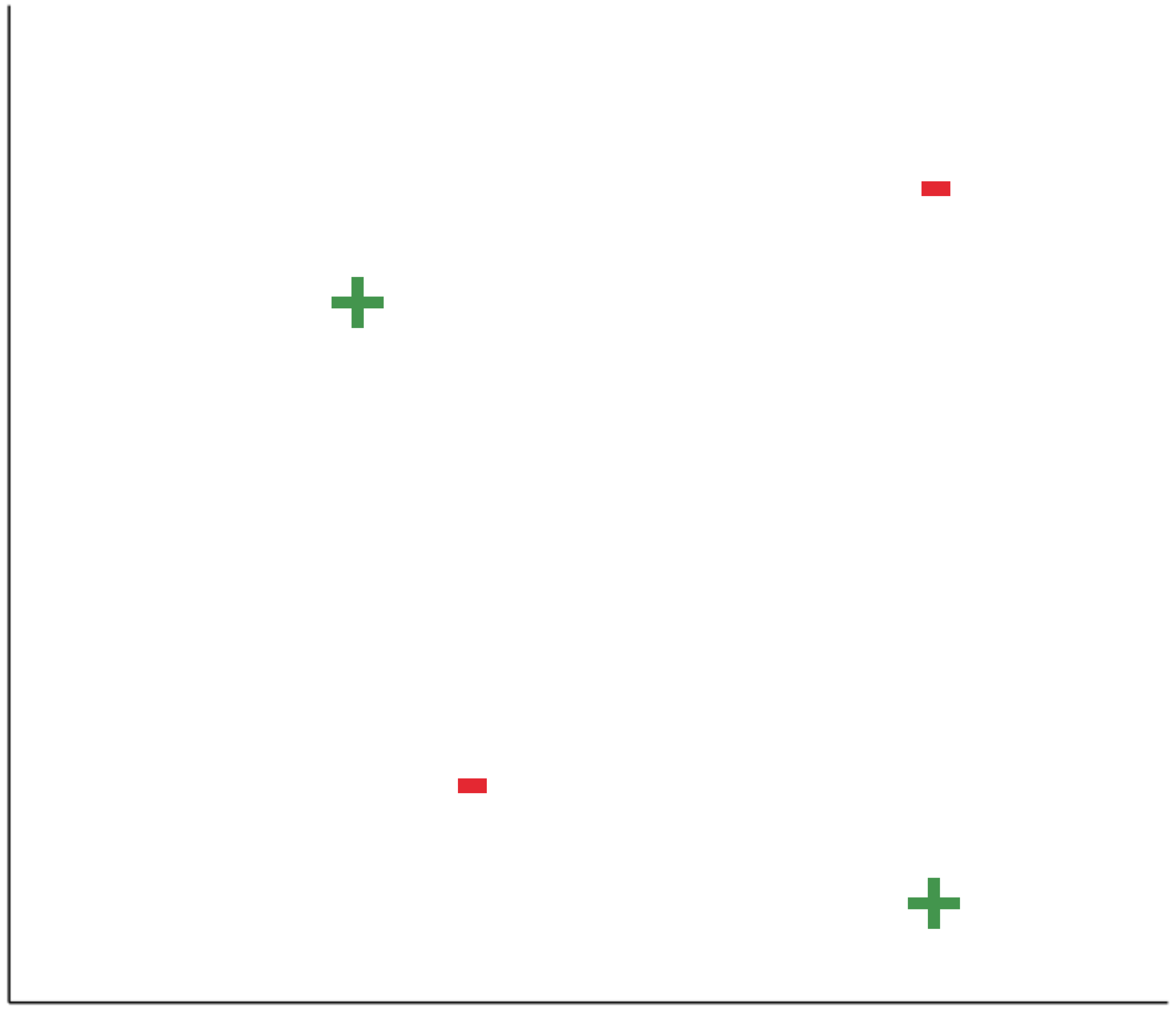


How about this?

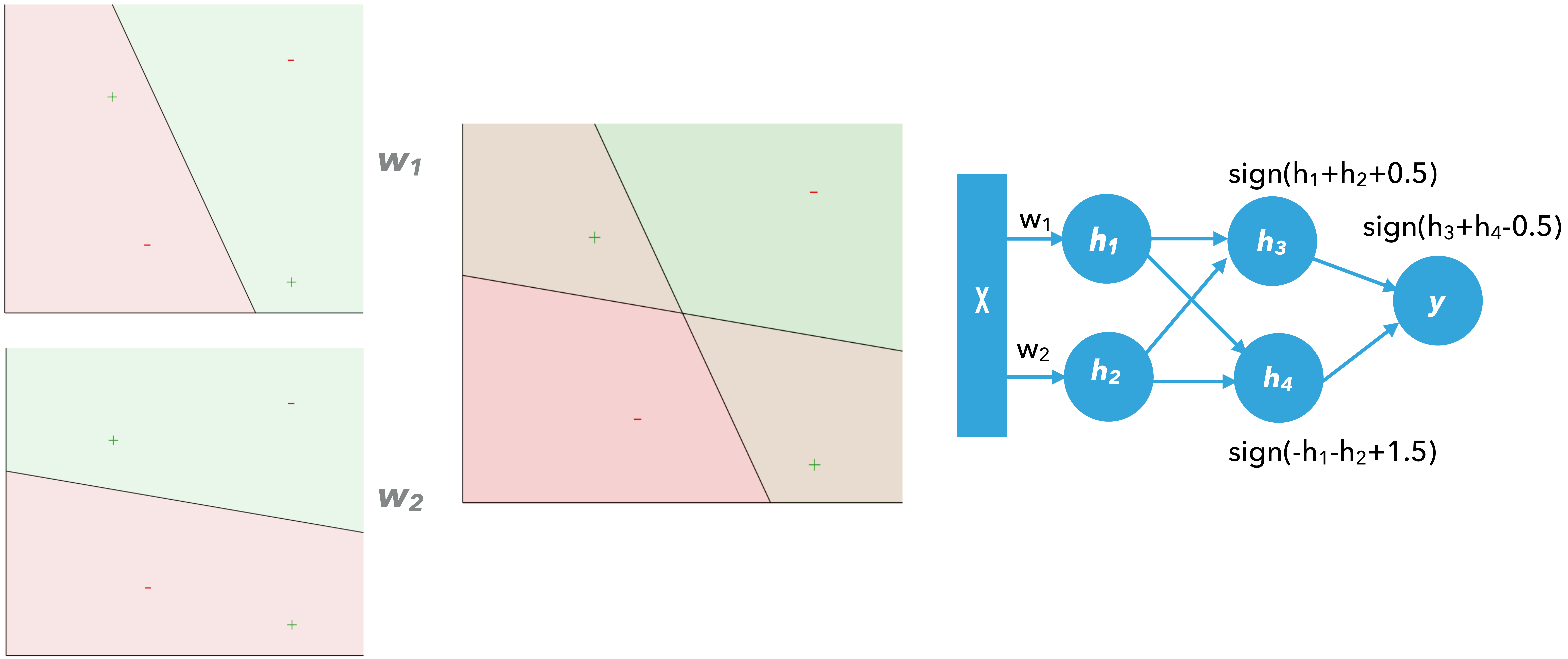


Perceptron is suitable for classifying a set of linearly separable data

FROM PERCEPTRON TO MULTI-LAYER NEURAL NETWORKS

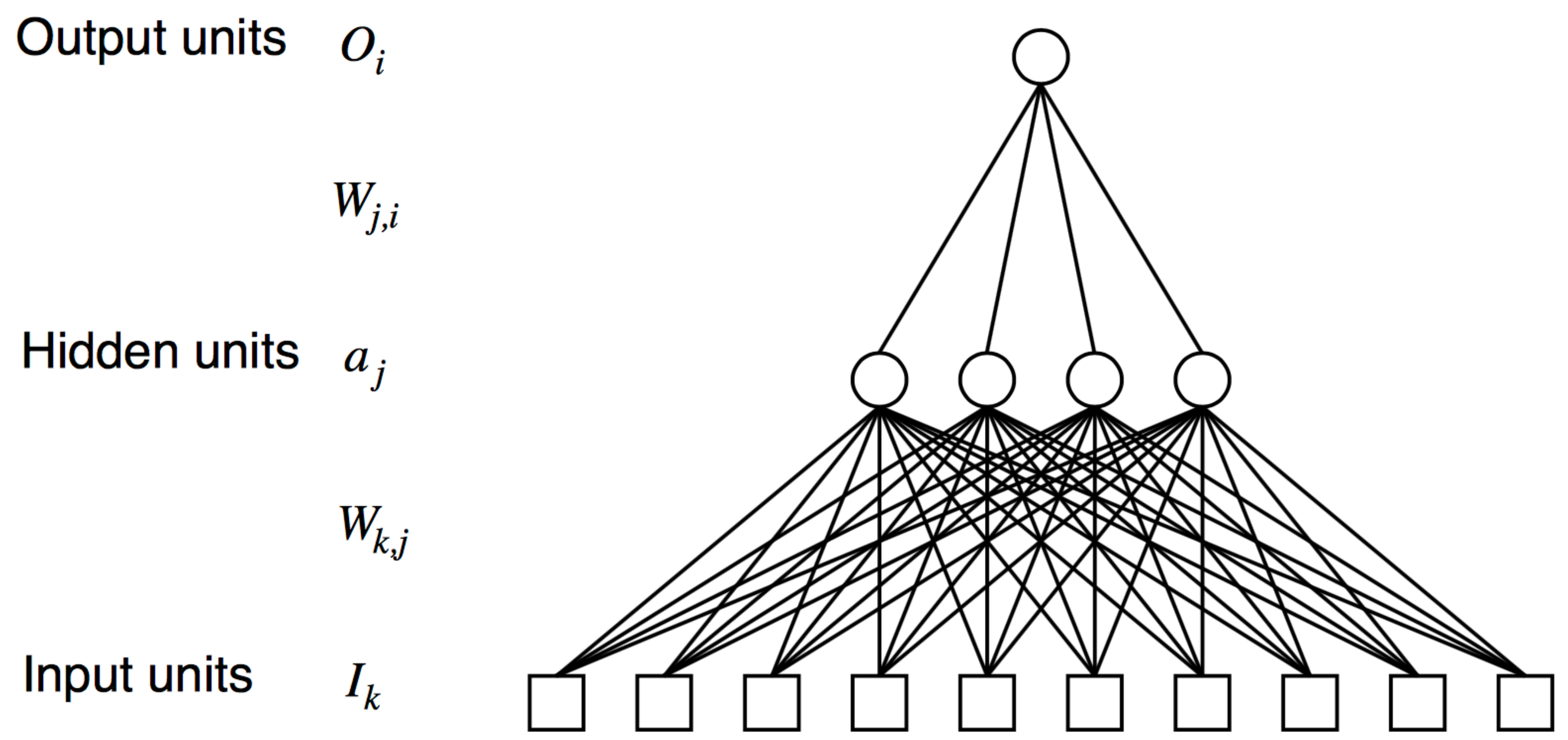


FROM PERCEPTRON TO MULTI-LAYER NEURAL NETWORKS



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



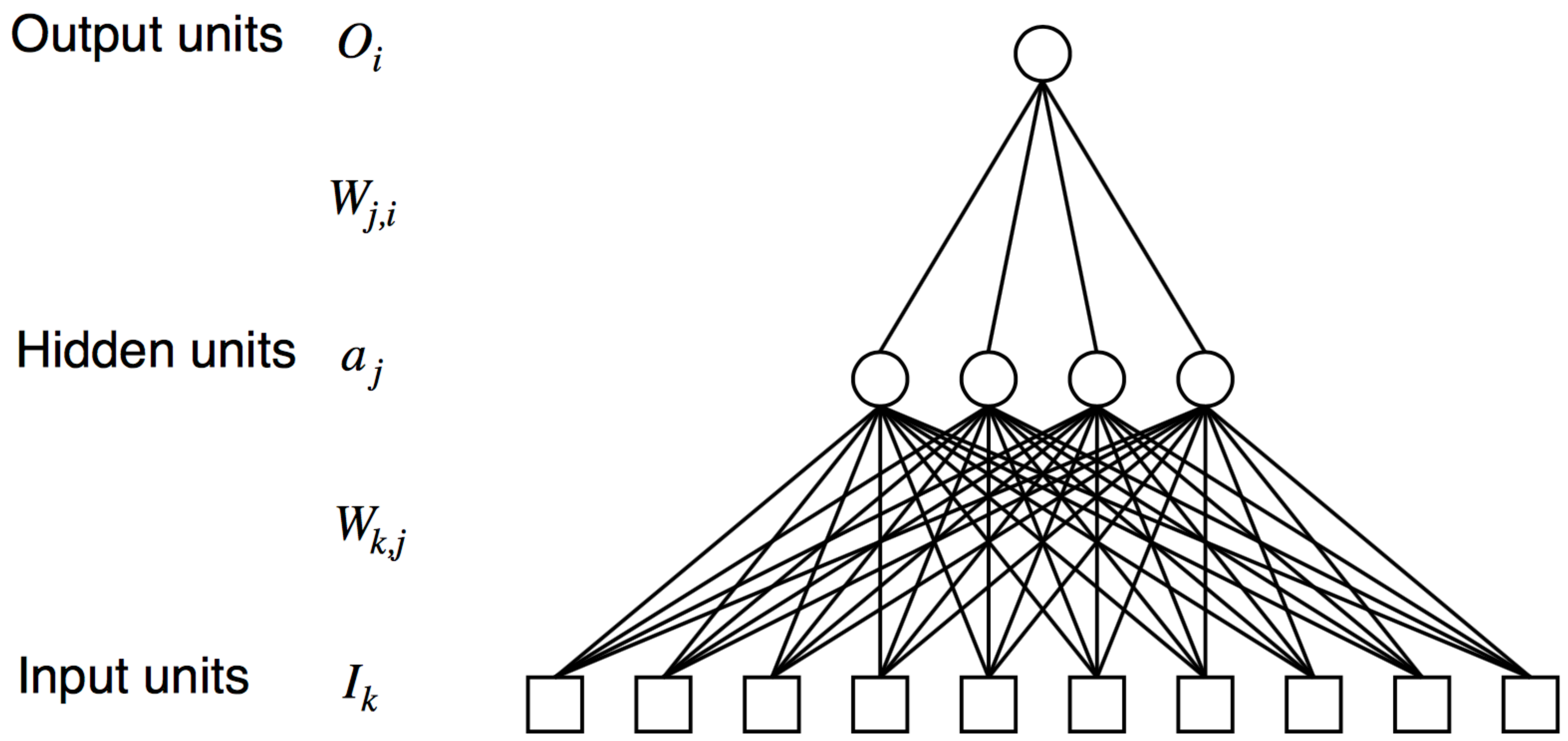
Output $$O_i = g(\sum_j W_{j,i} a_j)$$

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

Figure: M. Velosa

LEARNING MULTI-LAYER NEURAL NETWORKS

- Does the algorithm used for learning perceptron still work?



- Randomly set an initial set of weight
- Compute the outputs for each hidden unit and output unit
- Compare outputs from output unit and true labels, and update $W_{j,i}$
- Wait…what about weights associated with hidden units, $W_{k,j}$?

DIFFERENTIABLE SCORING FUNCTIONS AND ACTIVATION FUNCTIONS

- The scoring function *S* will take as inputs **x** (attributes), y (true label), $W_{k,j}$ (weights associated with hidden units), $W_{j,i}$ (weights associated with output units)