

# Section 2: Learning

Chuma Kabaghe, Chuanbo Pan

- Beyond Linear Functions
- Backpropagation
- Nearest Neighbors
- Scikit-learn Tutorial

# Beyond Linear Functions

# Example: beyond linear functions

Regression:  $x \in \mathbb{R}, y \in \mathbb{R}$

Linear functions:

$$\phi(x) = x$$

$$\mathcal{F}_1 = \{x \mapsto w_1x + w_2x^2 : w_1 \in \mathbb{R}, w_2 = 0\}$$

Quadratic functions:

$$\phi(x) = [x, x^2]$$

$$\mathcal{F}_2 = \{x \mapsto w_1x + w_2x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

[whiteboard]

# Example: even more flexible functions

Regression:  $x \in \mathbb{R}, y \in \mathbb{R}$

Piecewise constant functions:

$$\phi(x) = [\mathbf{1}[0 < x \leq 1], \mathbf{1}[1 < x \leq 2], \dots]$$

$$\mathcal{F}_3 = \{x \mapsto \sum_{j=1}^{10} w_j \mathbf{1}[j-1 < x \leq j] : \mathbf{w} \in \mathbb{R}^{10}\}$$

[whiteboard]

# Backpropagation

# Backpropagation

$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

# Backpropagation

$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

Weights

Predicted

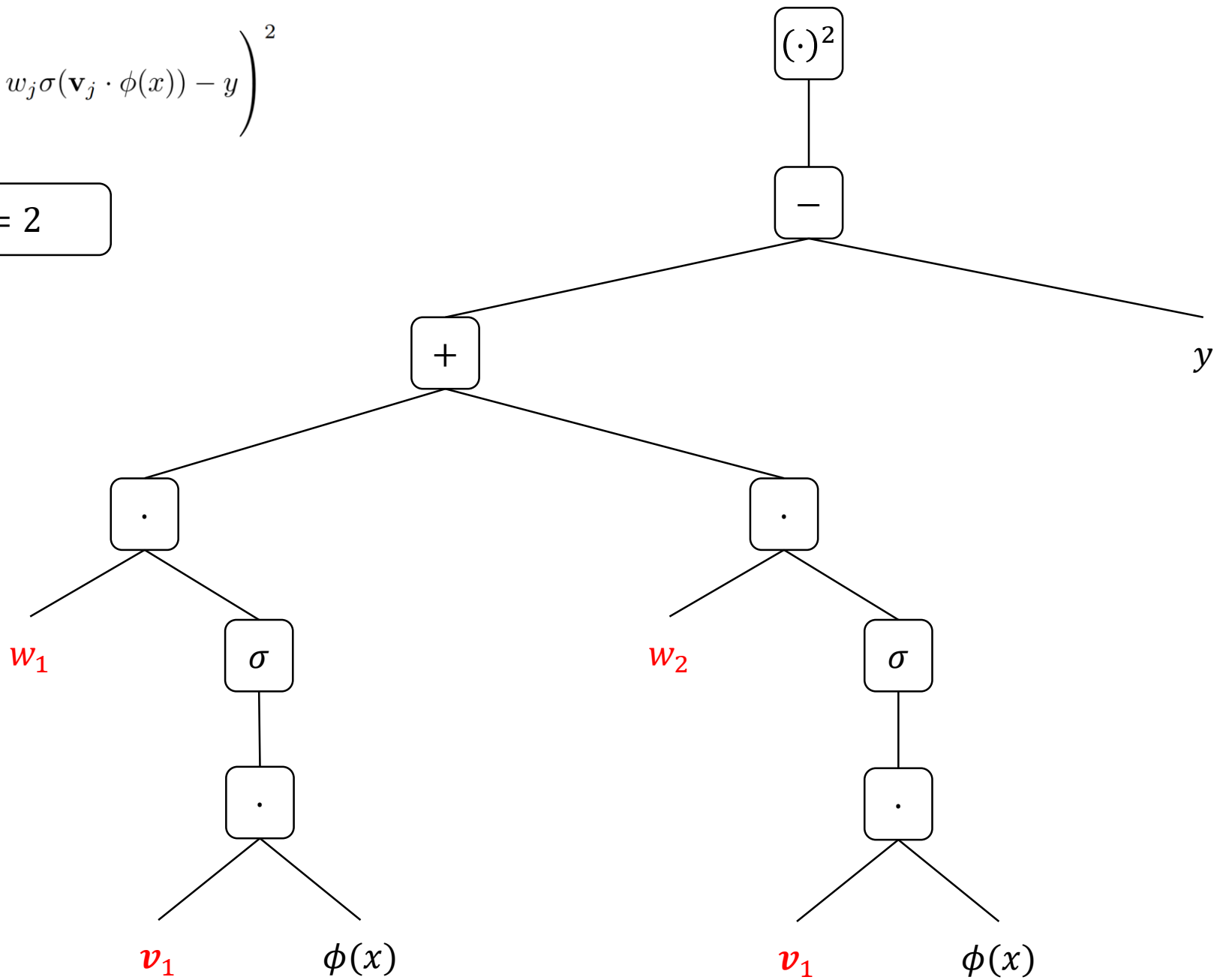
Actual

The diagram illustrates the loss function  $\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$ . A large green box encloses the summation term  $\sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x))$ , which is labeled "Predicted" below it. A smaller green box encloses the term  $y$ , which is labeled "Actual" below it. Two blue arrows originate from the word "Weights" above the summation box and point to the weights  $w_j$  and the vectors  $\mathbf{v}_j$  within the summation.

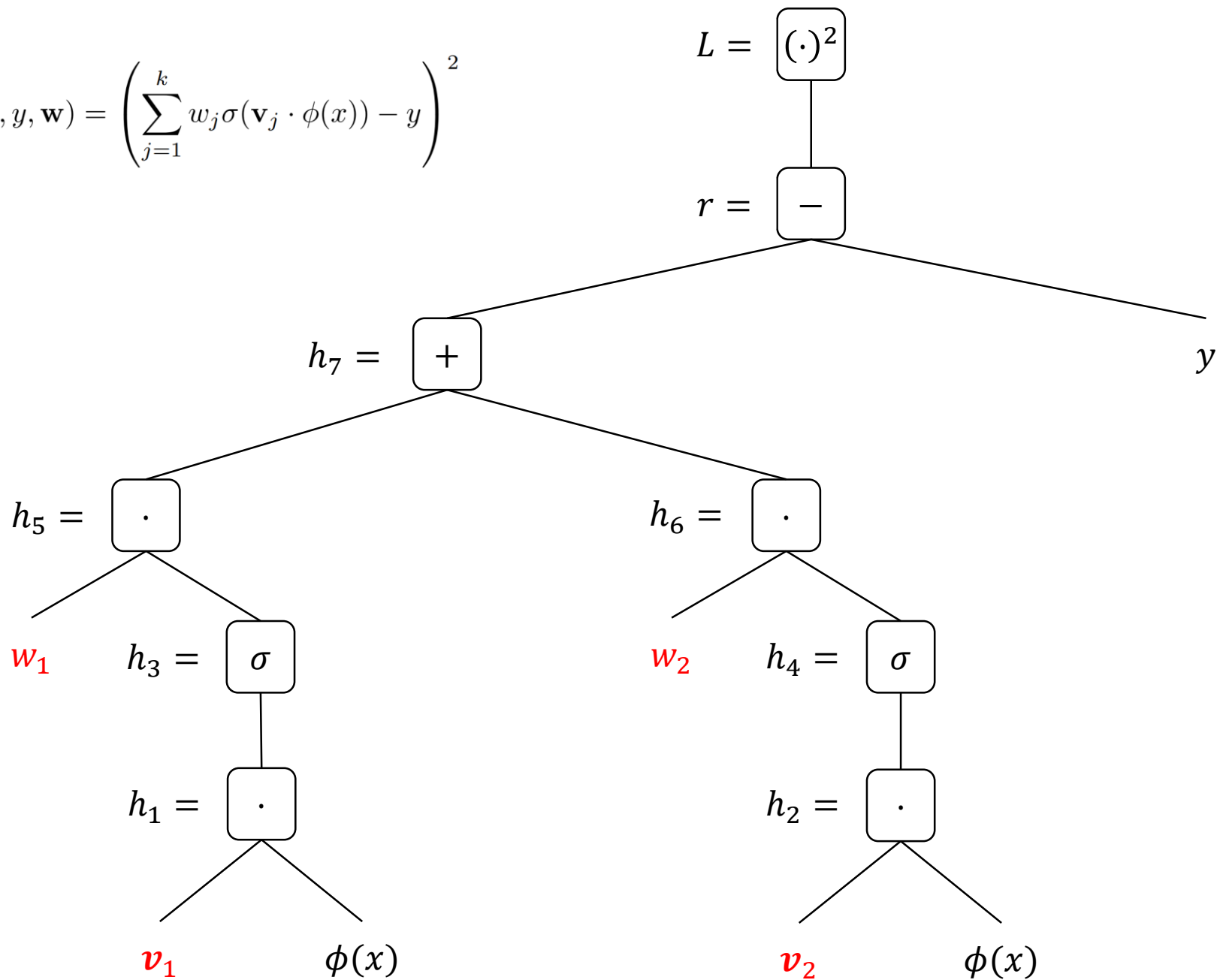


$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

Consider  $k = 2$



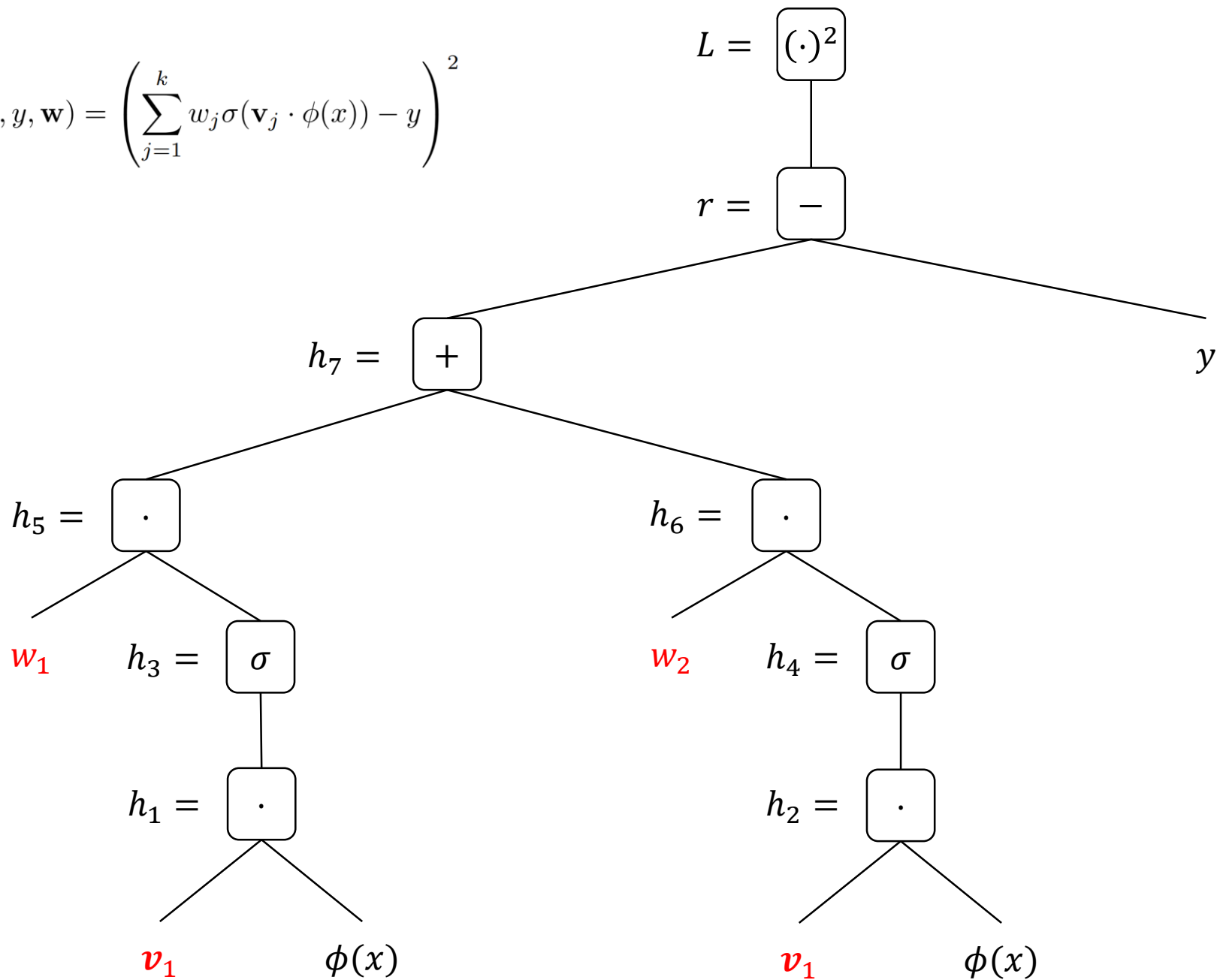
$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$



**Forward Pass**

Compute each node and  
store intermediate values

$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$



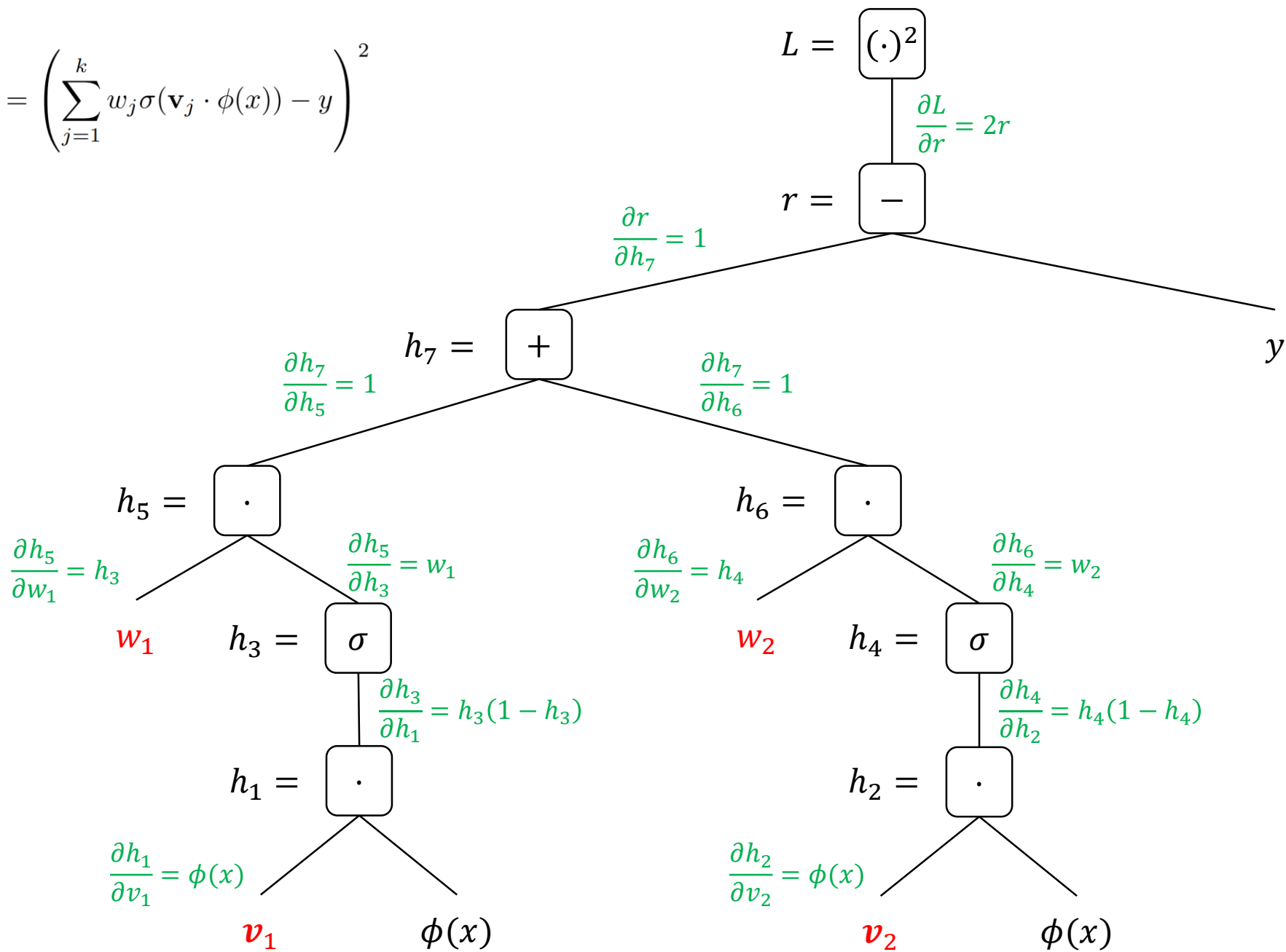
## Backward Pass

Compute derivatives with respect to intermediate variables:

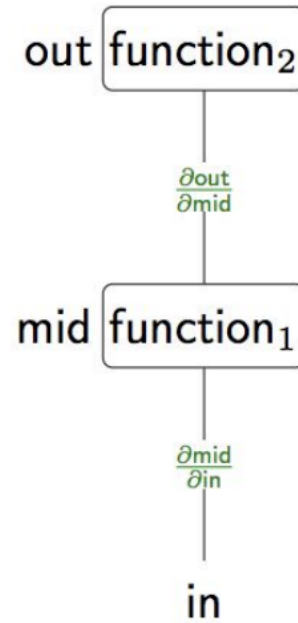
$$\frac{\partial L}{\partial r}, \frac{\partial r}{\partial h_7}, \frac{\partial h_6}{\partial h_4}, \frac{\partial h_5}{\partial h_3}, \text{etc}$$

[Whiteboard]

$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

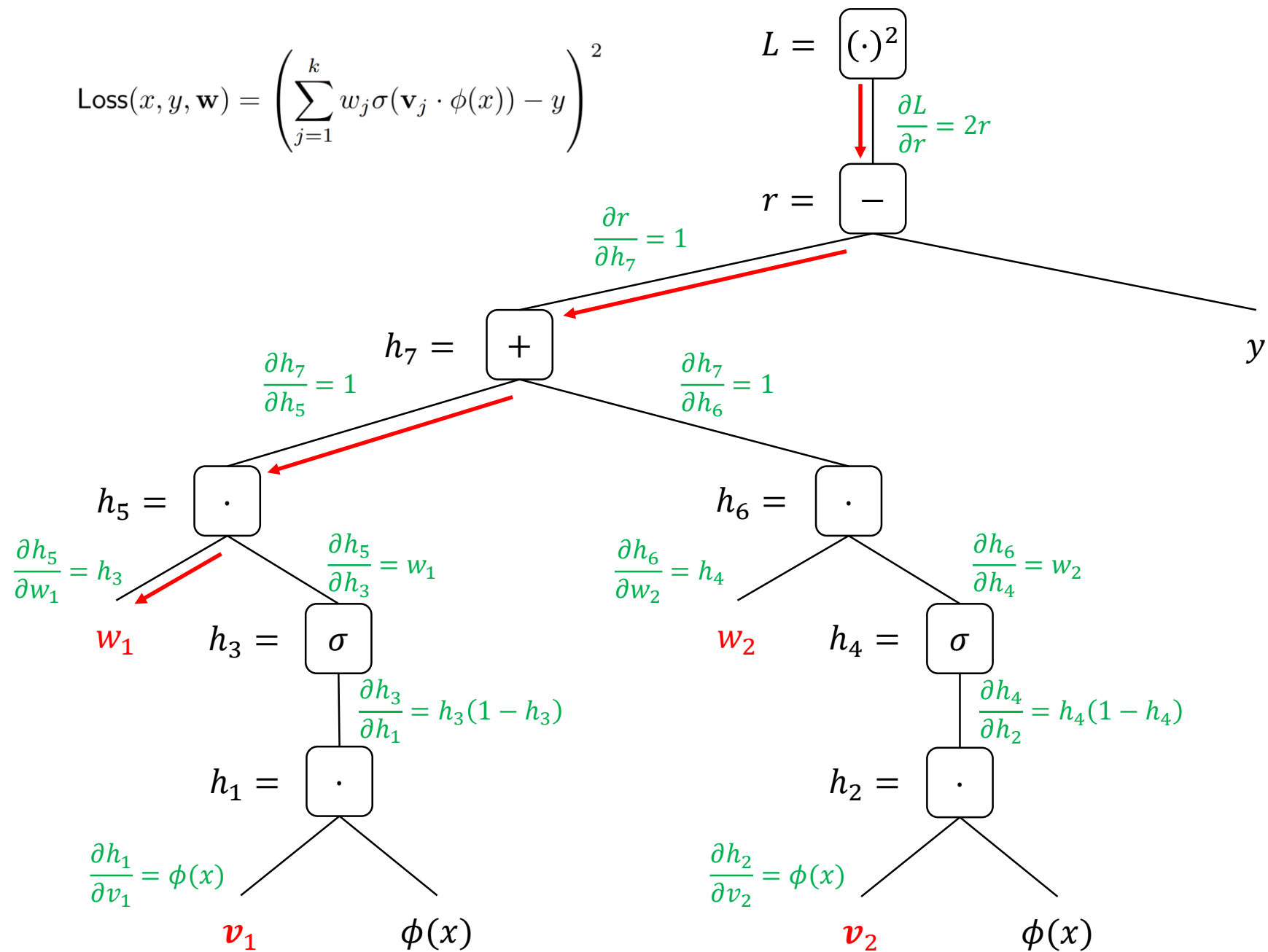


# Backpropagation



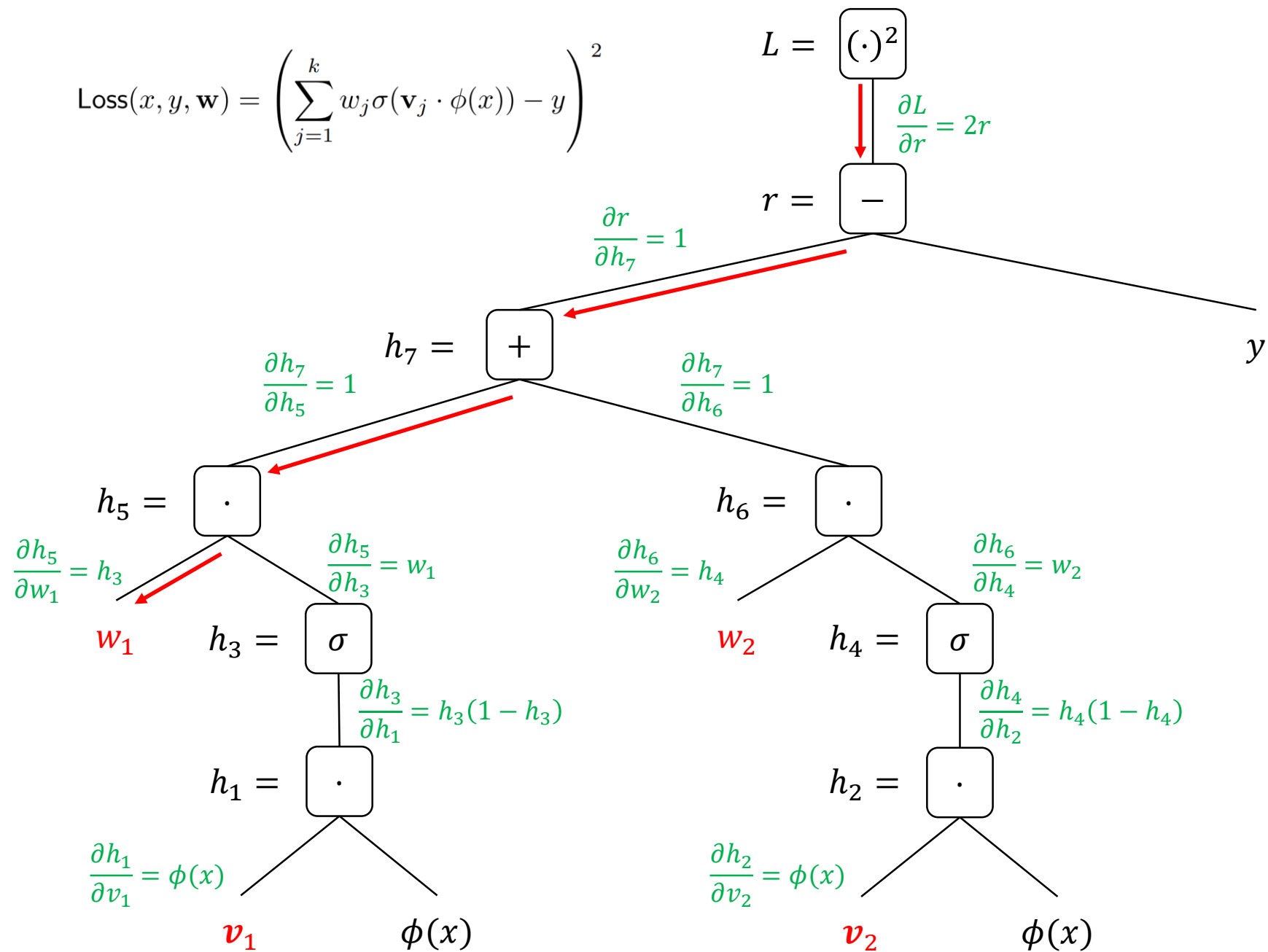
Chain rule:  $\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$

$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$



[Whiteboard]

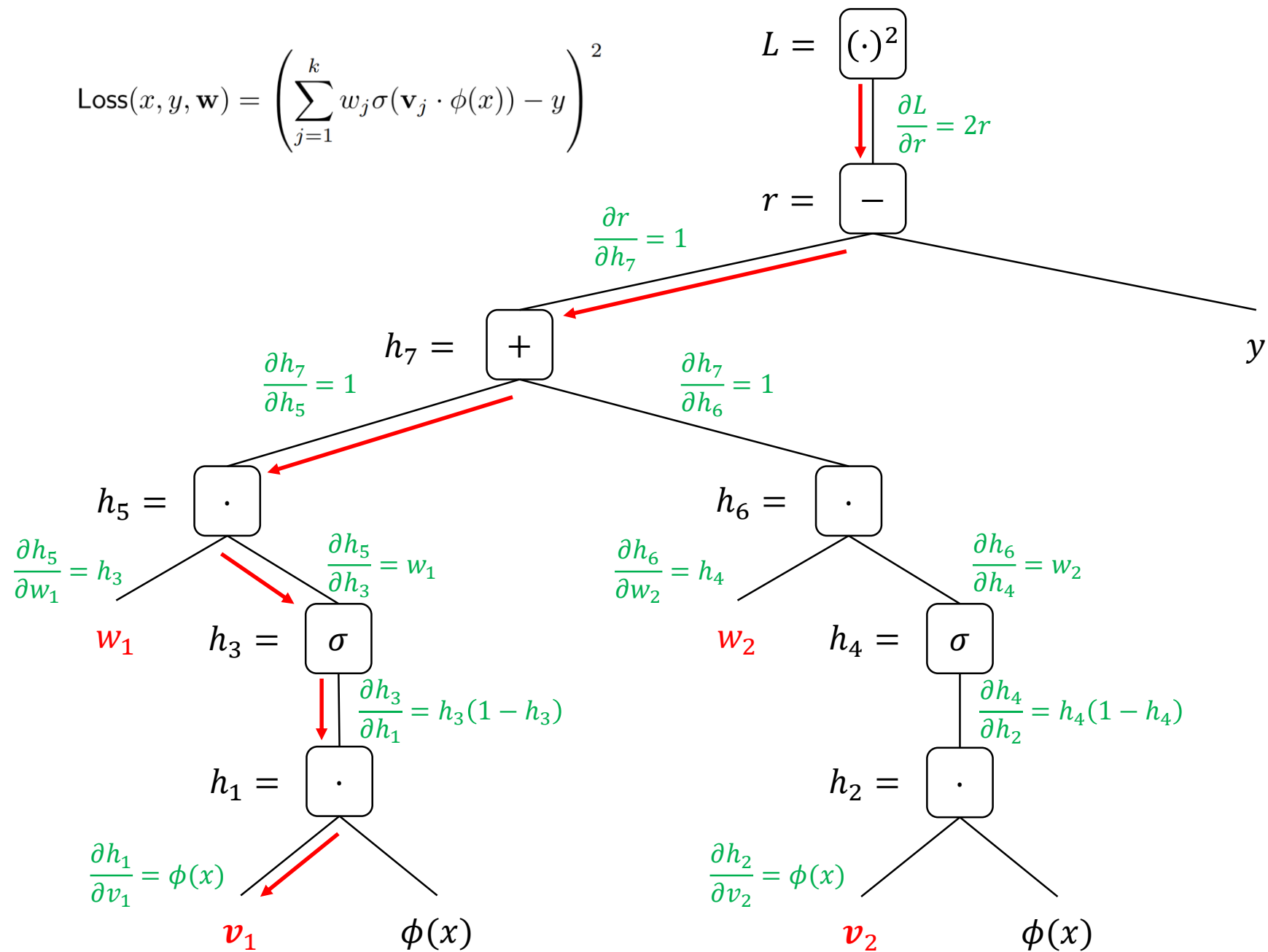
$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial r} \cdot \frac{\partial r}{\partial h_7} \cdot \frac{\partial h_7}{\partial h_5} \cdot \frac{\partial h_5}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = (2r)(1)(1)(h_3)$$

$$\text{Loss}(x, y, \mathbf{w}) = \left( \sum_{j=1}^k w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial r} \cdot \frac{\partial r}{\partial h_7} \cdot \frac{\partial h_7}{\partial h_5} \cdot \frac{\partial h_5}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = (2r)(1)(1)(h_3)$$

$$\frac{\partial L}{\partial v_1} = \frac{\partial L}{\partial r} \cdot \frac{\partial r}{\partial h_7} \cdot \frac{\partial h_7}{\partial h_5} \cdot \frac{\partial h_5}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_1} \cdot \frac{\partial h_1}{\partial v_1}$$

$$\frac{\partial L}{\partial w_1} = (2r)(1)(1)(w_1)(h_3(1 - h_3))(\phi(x))$$



# Backpropagation

Backprop: <http://cs231n.github.io/optimization-2/>

Vector, Matrix, and Tensor Derivatives: <http://cs231n.Stanford.edu/vecDerivs.pdf>



Nearest Neighbors

# Nearest neighbors



## Algorithm: nearest neighbors

Training: just store  $\mathcal{D}_{\text{train}}$

Predictor  $f(x')$ :

- Find  $(x, y) \in \mathcal{D}_{\text{train}}$  where  $\|\phi(x) - \phi(x')\|$  is smallest
- Return  $y$

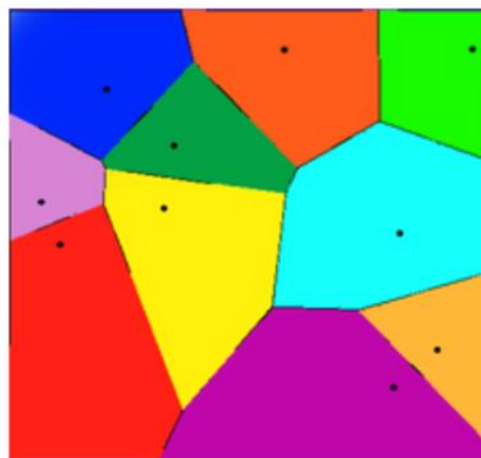


## Key idea: similarity

Similar examples tend to have similar outputs.

# Expressivity of nearest neighbors

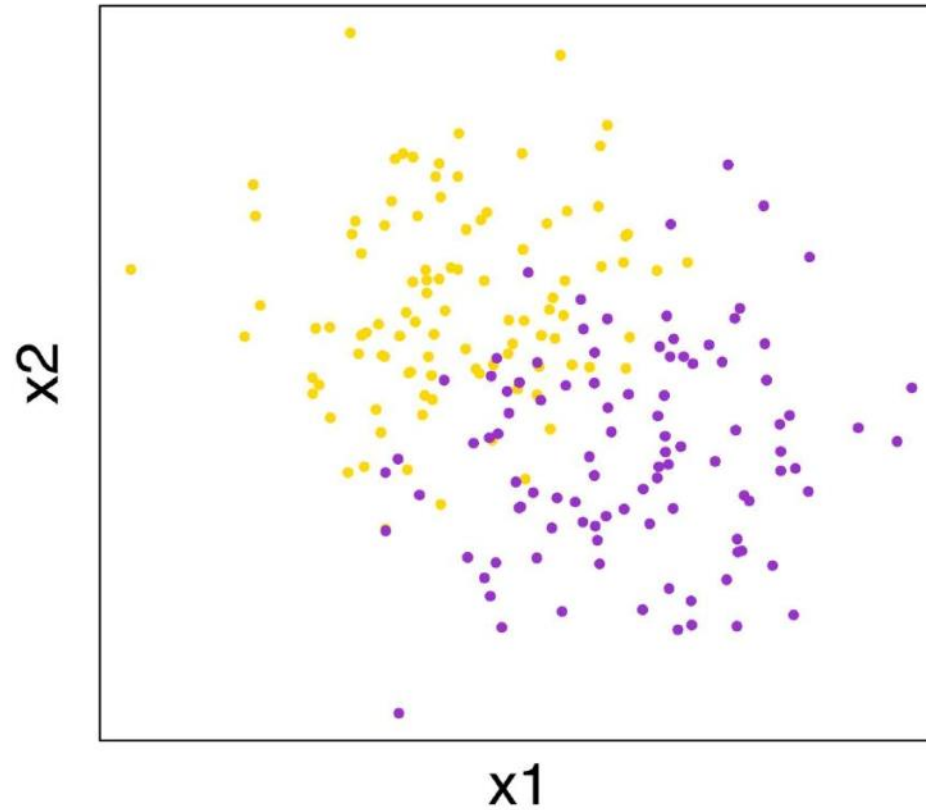
Decision boundary: based on Voronoi diagram



- Much more expressive than quadratic features
- **Non-parametric**: the hypothesis class adapts to number of examples
- Simple and powerful, but kind of brute force

# $k$ -nearest neighbors

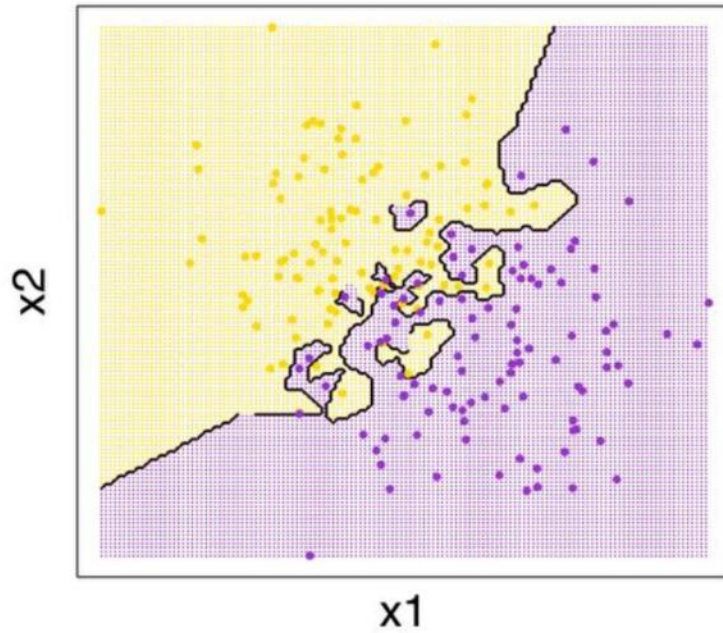
Binary kNN Classification Training Set



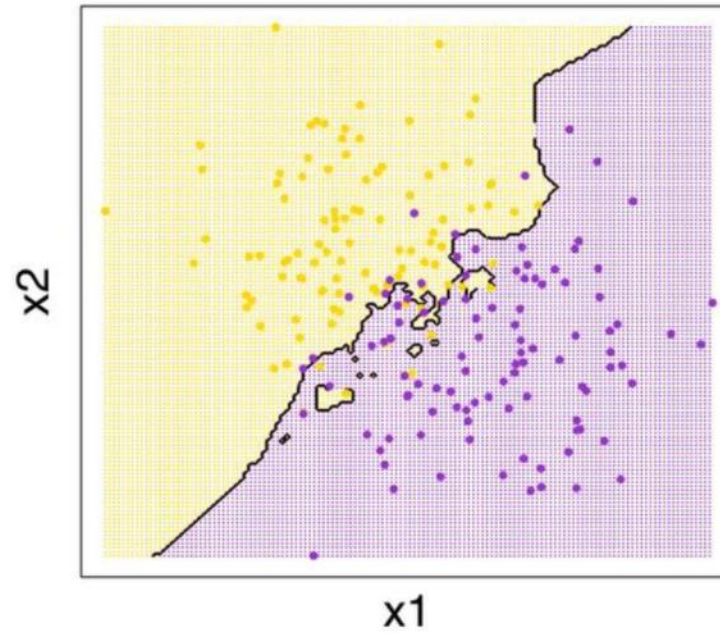
# $k$ -nearest neighbors

Effect of  $k$ :

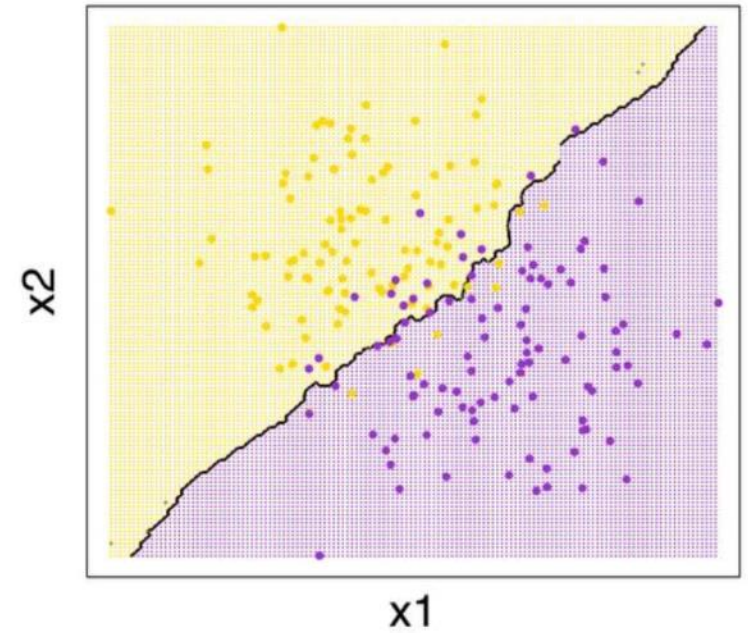
Binary kNN Classification ( $k=1$ )



Binary kNN Classification ( $k=5$ )

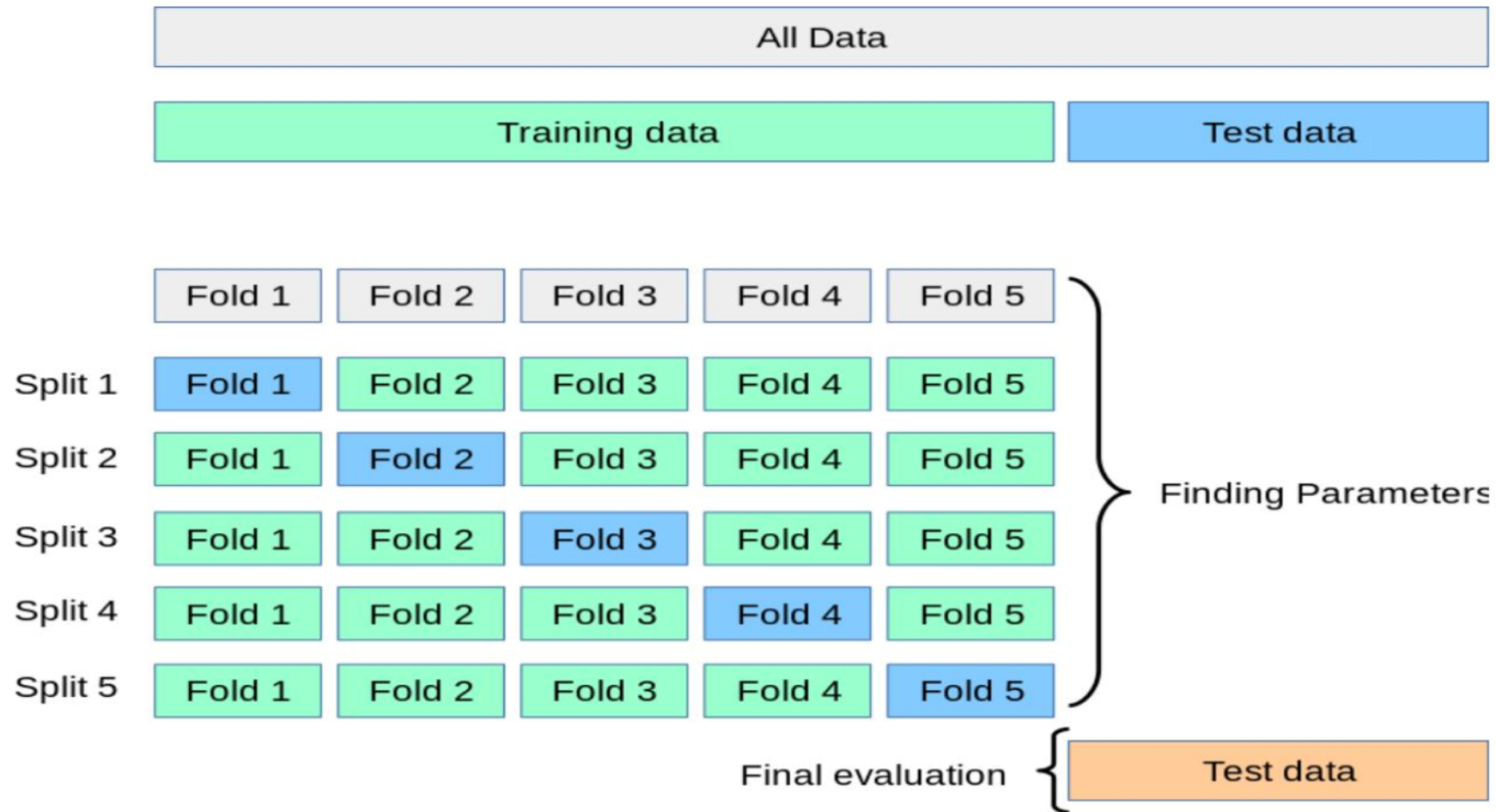


Binary kNN Classification ( $k=25$ )





# $k$ -fold cross-validation



# Scikit-learn Tutorial



[Switch to ipython notebook]