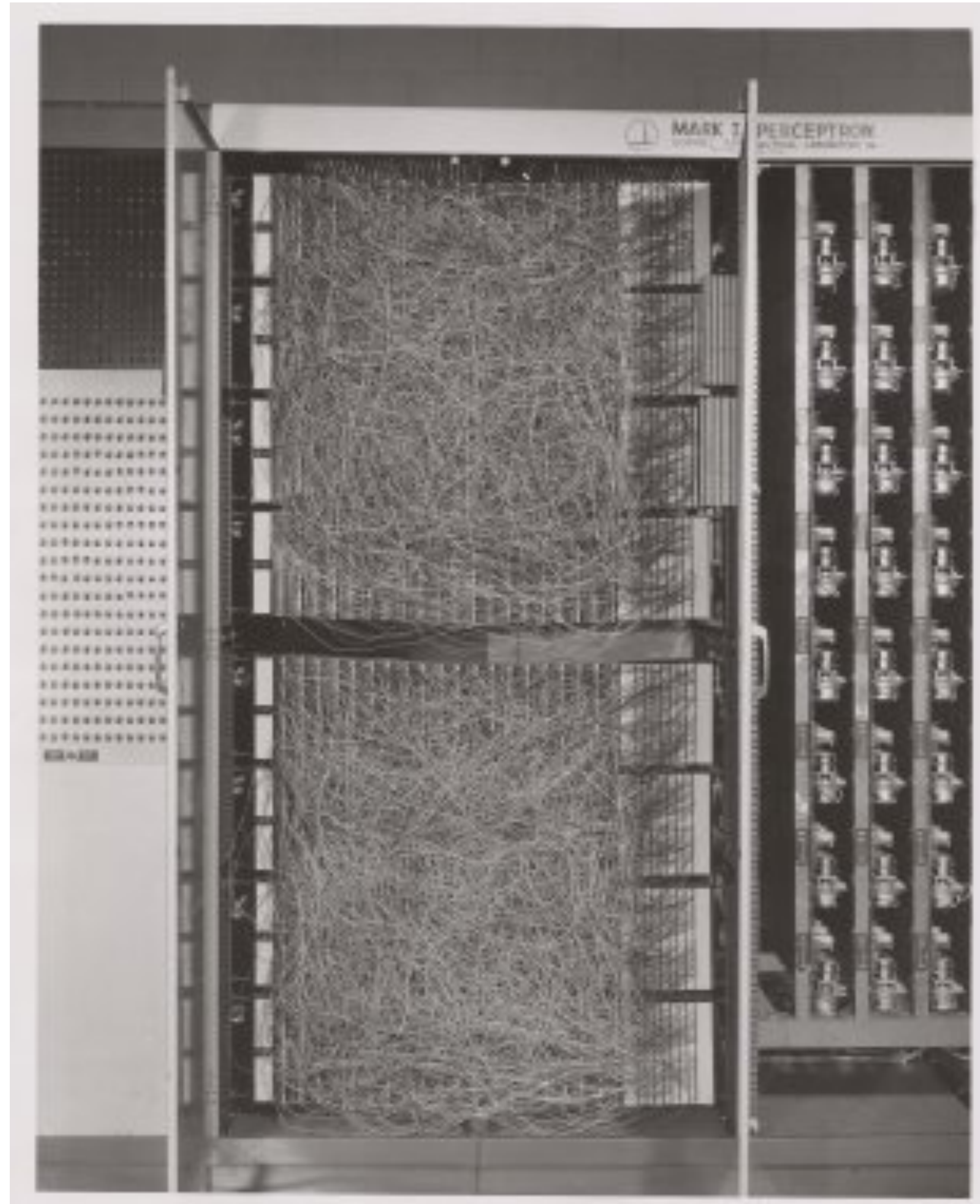


Perceptron Algorithm

16-385 Computer Vision (Kris Kitani)
Carnegie Mellon University

1950s Age of the Perceptron

1957 The Perceptron (Rosenblatt)

1969 Perceptrons (Minsky, Papert)

1980s Age of the Neural Network

1986 Back propagation (Hinton)

1990s Age of the Graphical Model

2000s Age of the Support Vector Machine

2010s Age of the Deep Network

deep learning = known algorithms + computing power + big data

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input-output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \qquad (1)$$

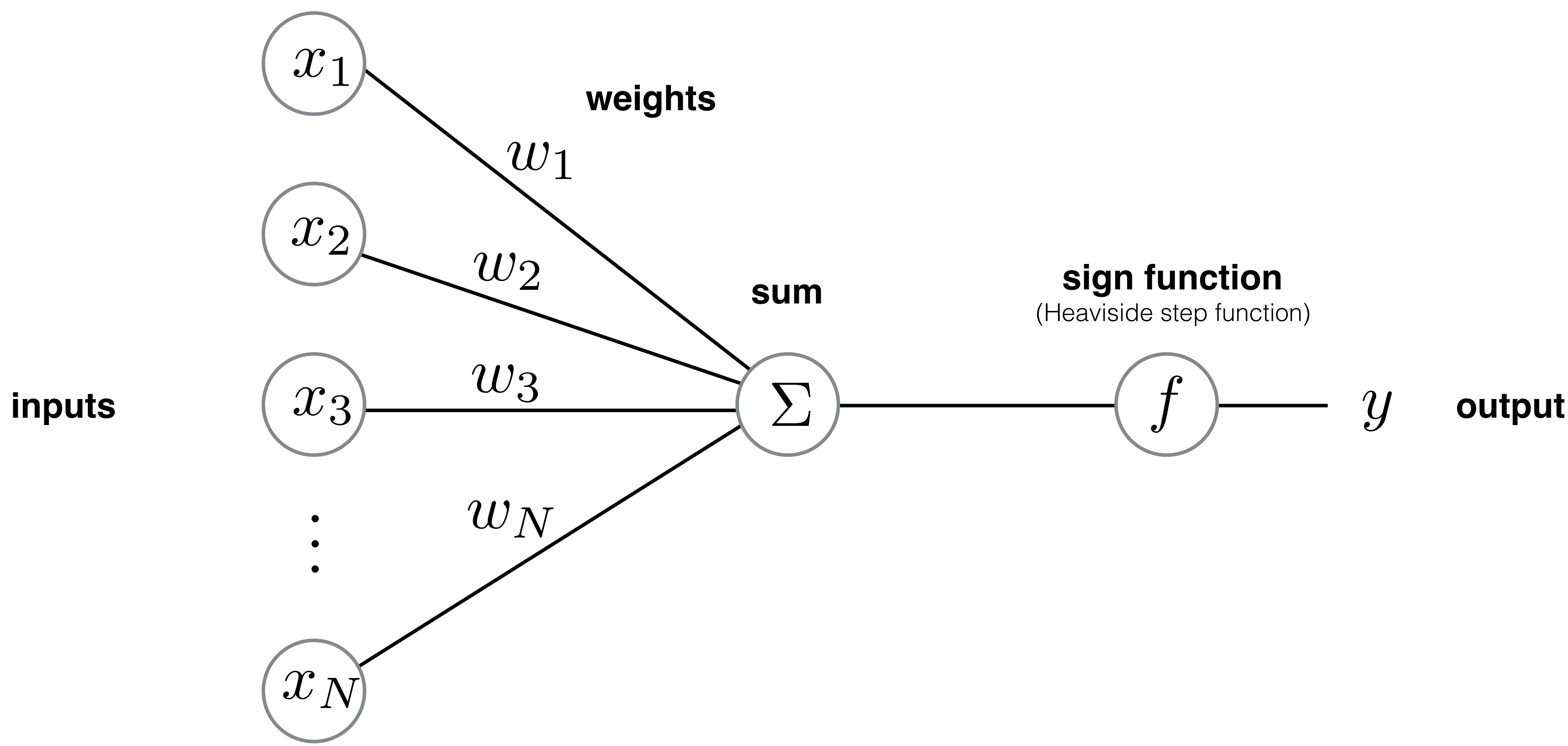
Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

$$y_j = \frac{1}{1+e^{-x_j}} \qquad (2)$$

† To whom correspondence should be addressed.

The Perceptron



function PERCEPTRON ALGORITHM

$\boldsymbol{w}^{(0)} \leftarrow \mathbf{0}$

for $t = 1, \ldots, T$ **do**

RECEIVE$(\boldsymbol{x}^{(t)})$ $\boldsymbol{x} \in \{0, 1\}^N$ N-d binary vector

$\hat{y}_A^{(t)} = \mathrm{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle \right)$ perceptron is just one line of code!

sign of zero is +1

RECEIVE(y^t) $y \in \{1, -1\}$

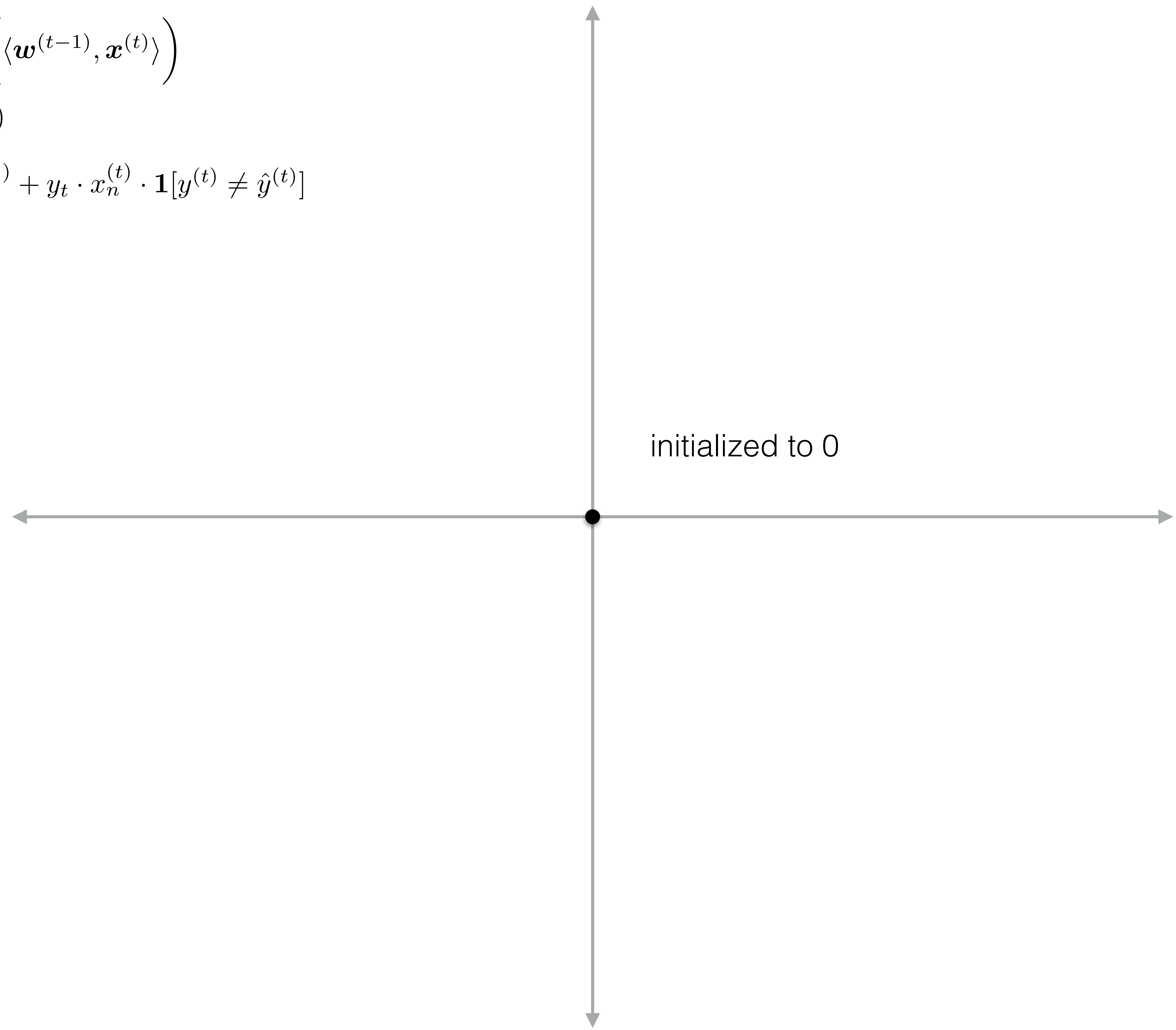
$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

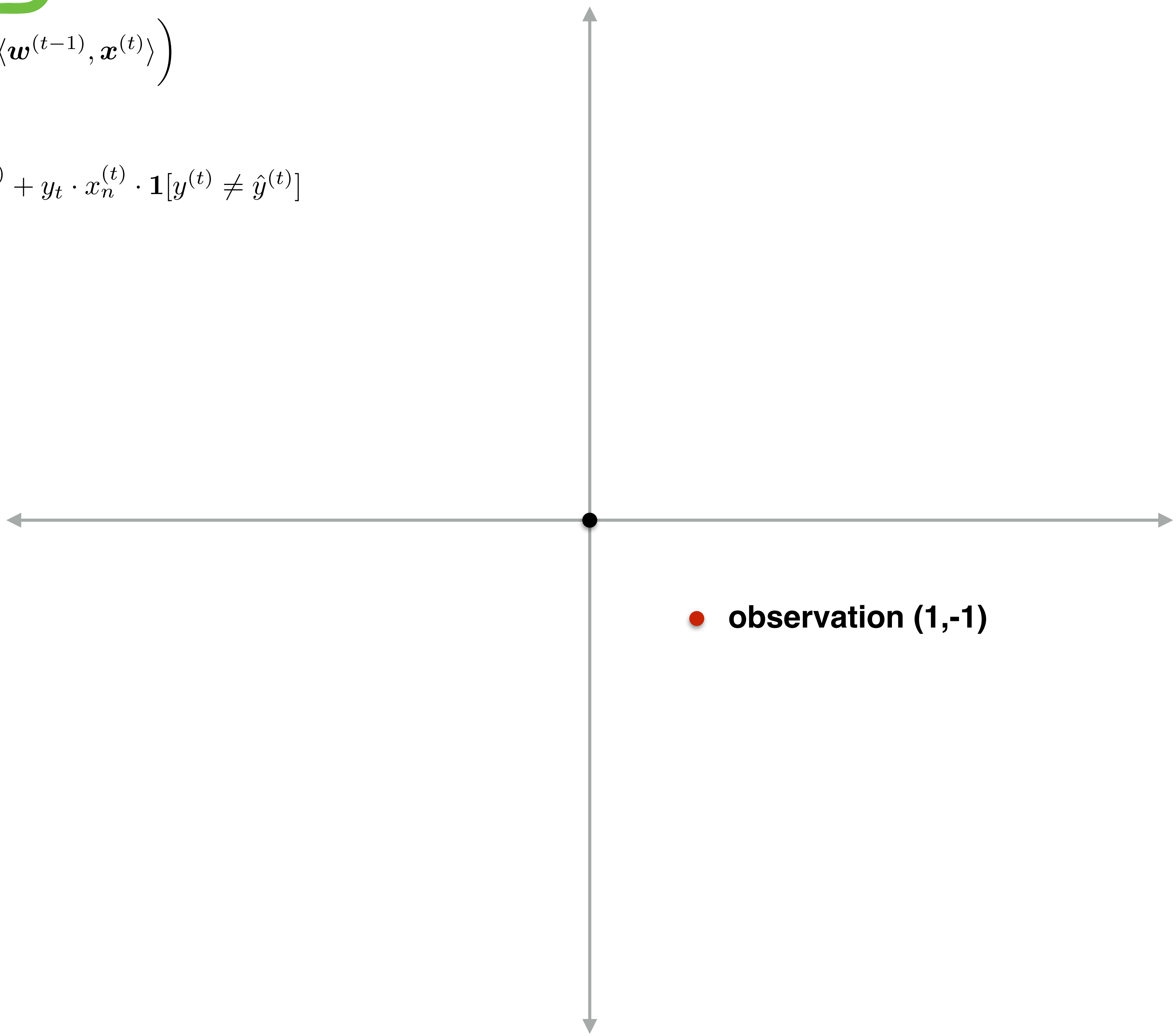


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

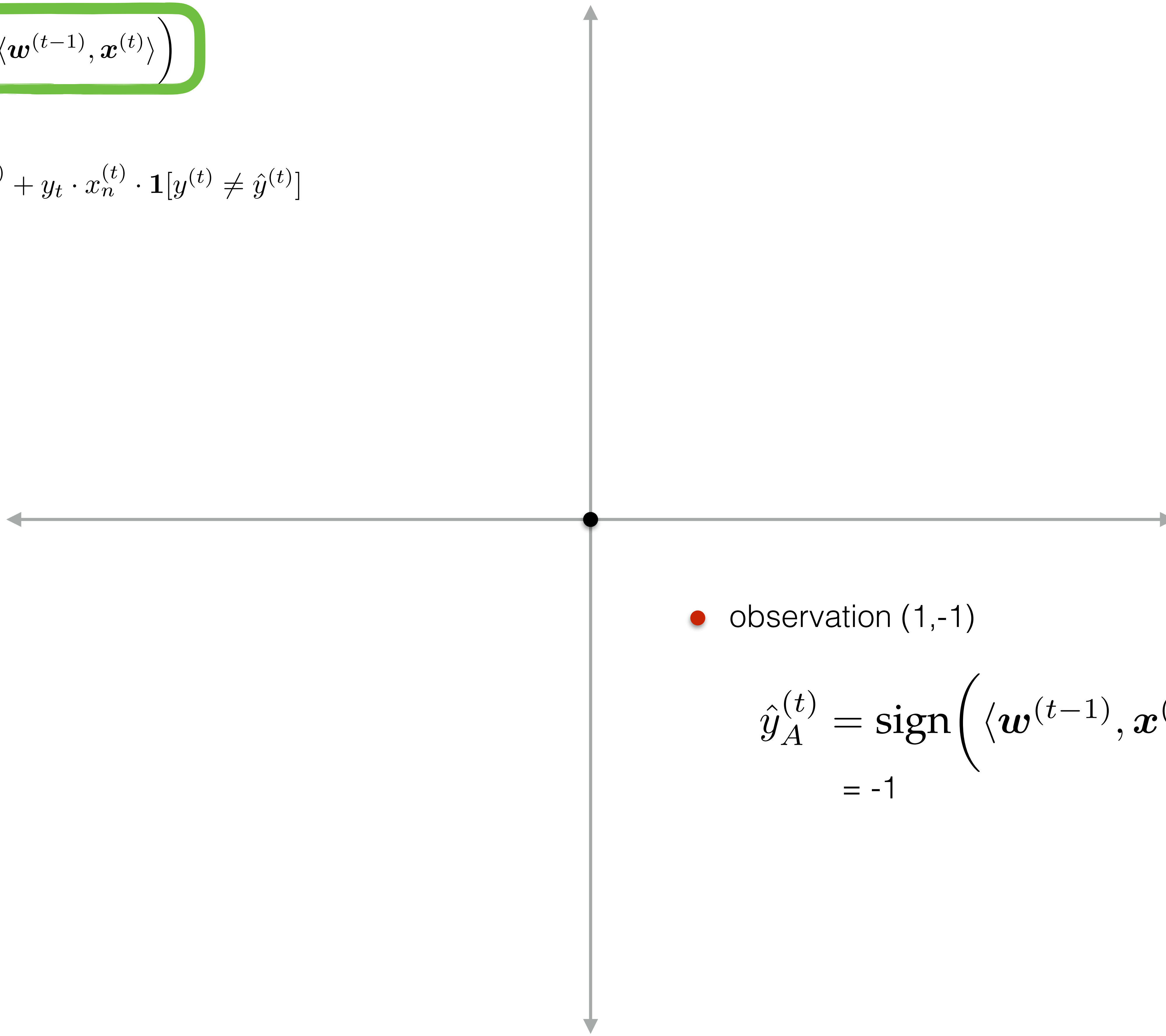


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$



$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

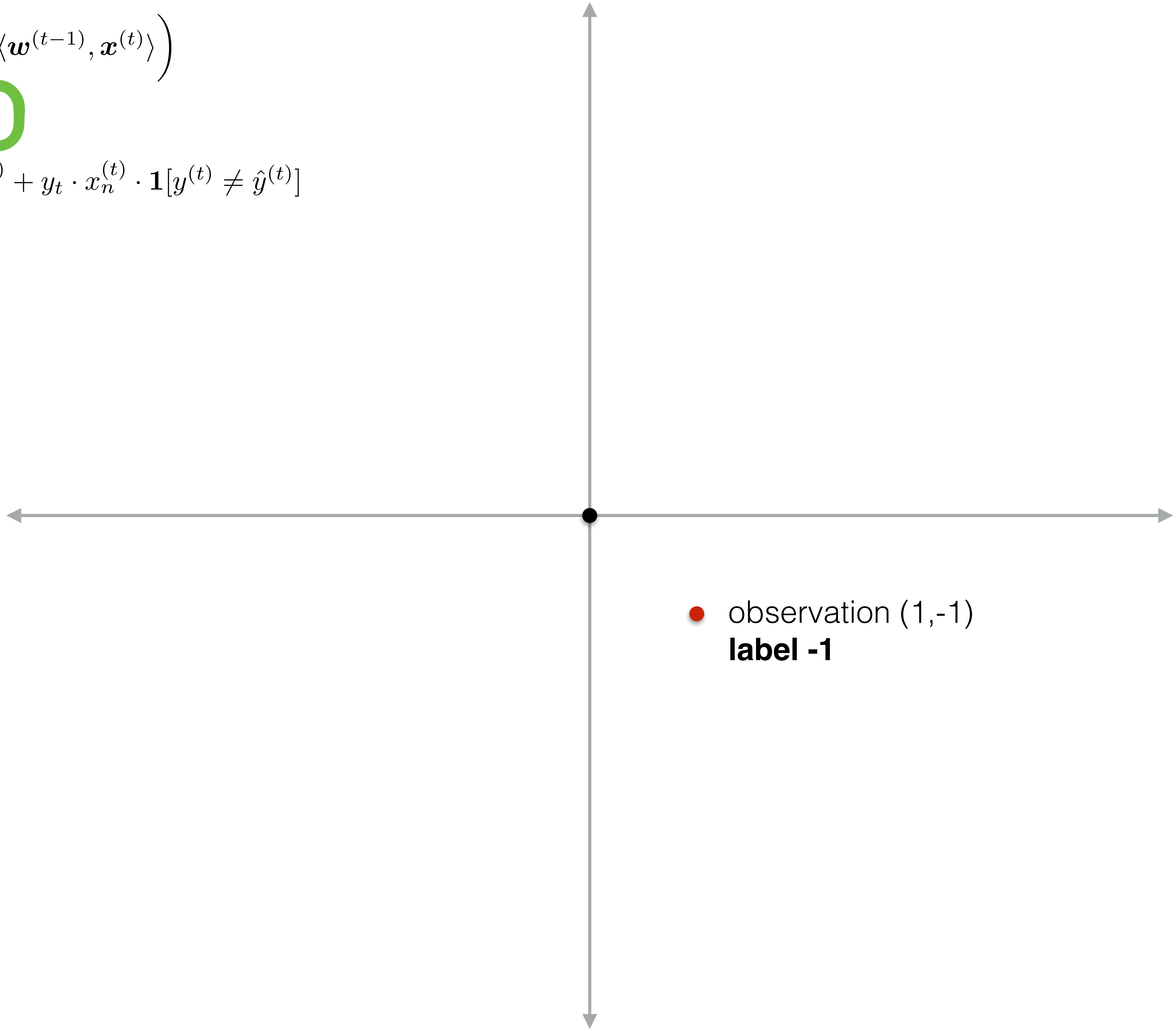
= -1

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$

$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$



RECEIVE($\boldsymbol{x}^{(t)}$)

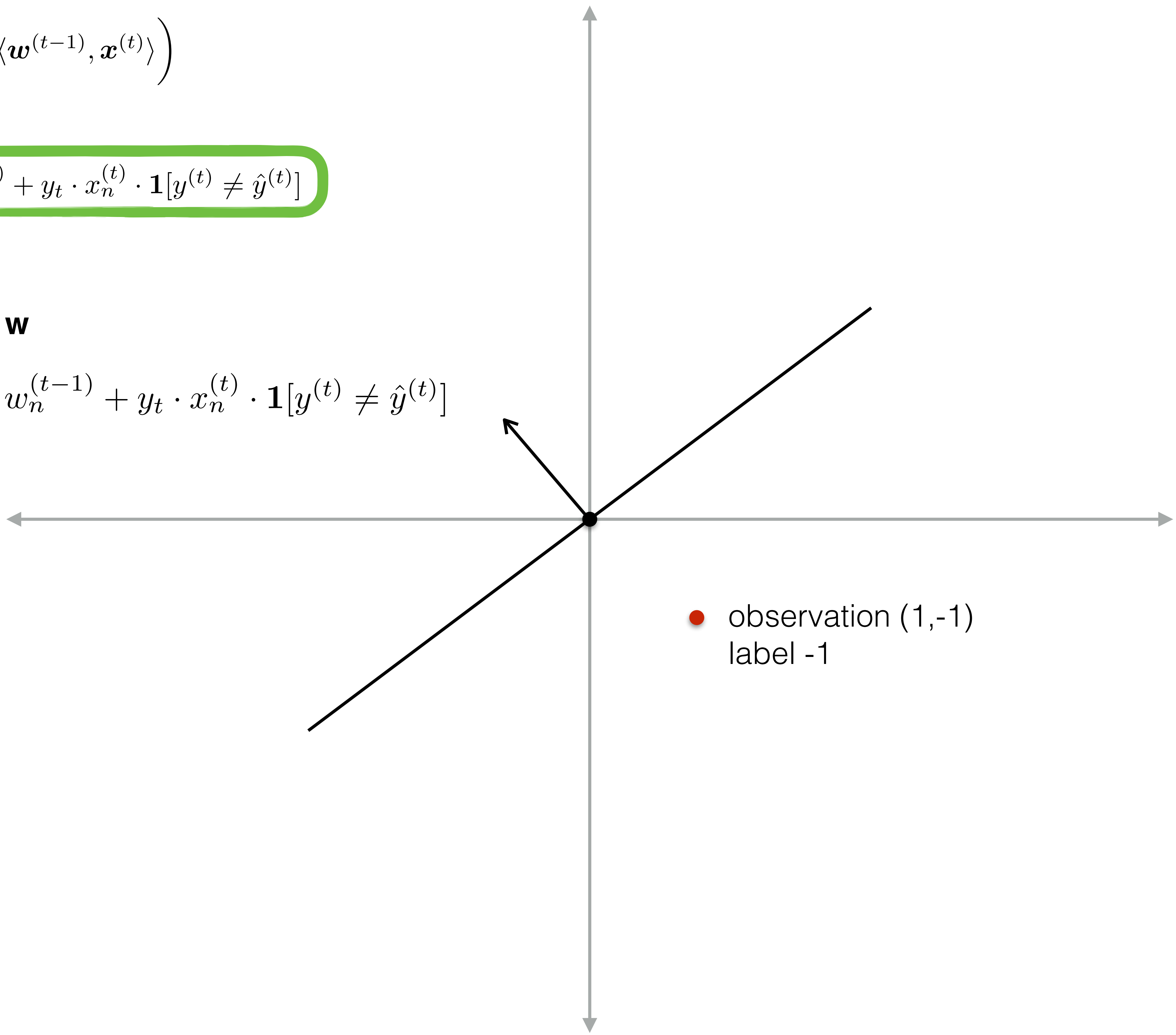
$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

update w

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$



$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

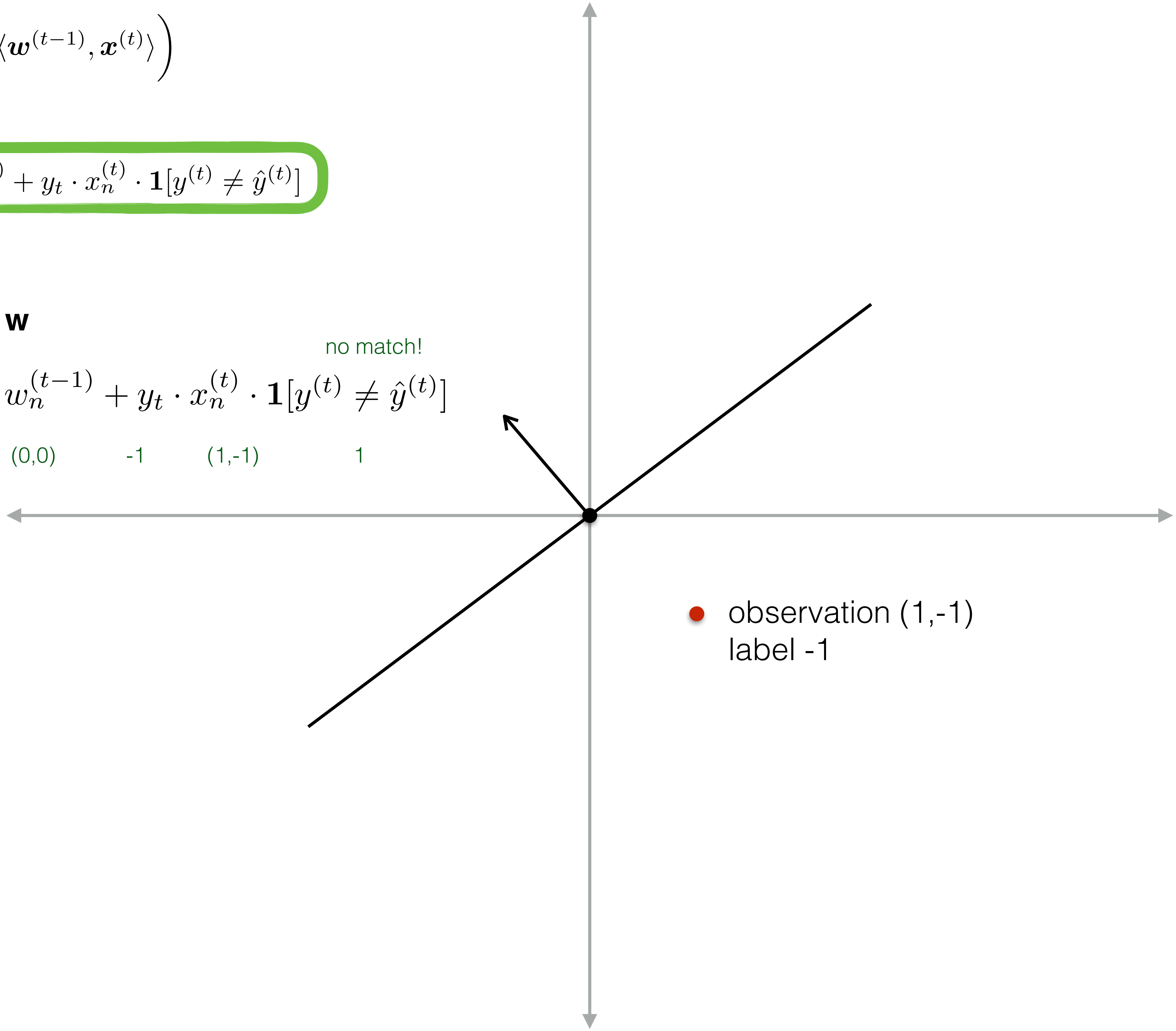
$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

update w

no match!

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1) (0,0) -1 (1,-1) 1



$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

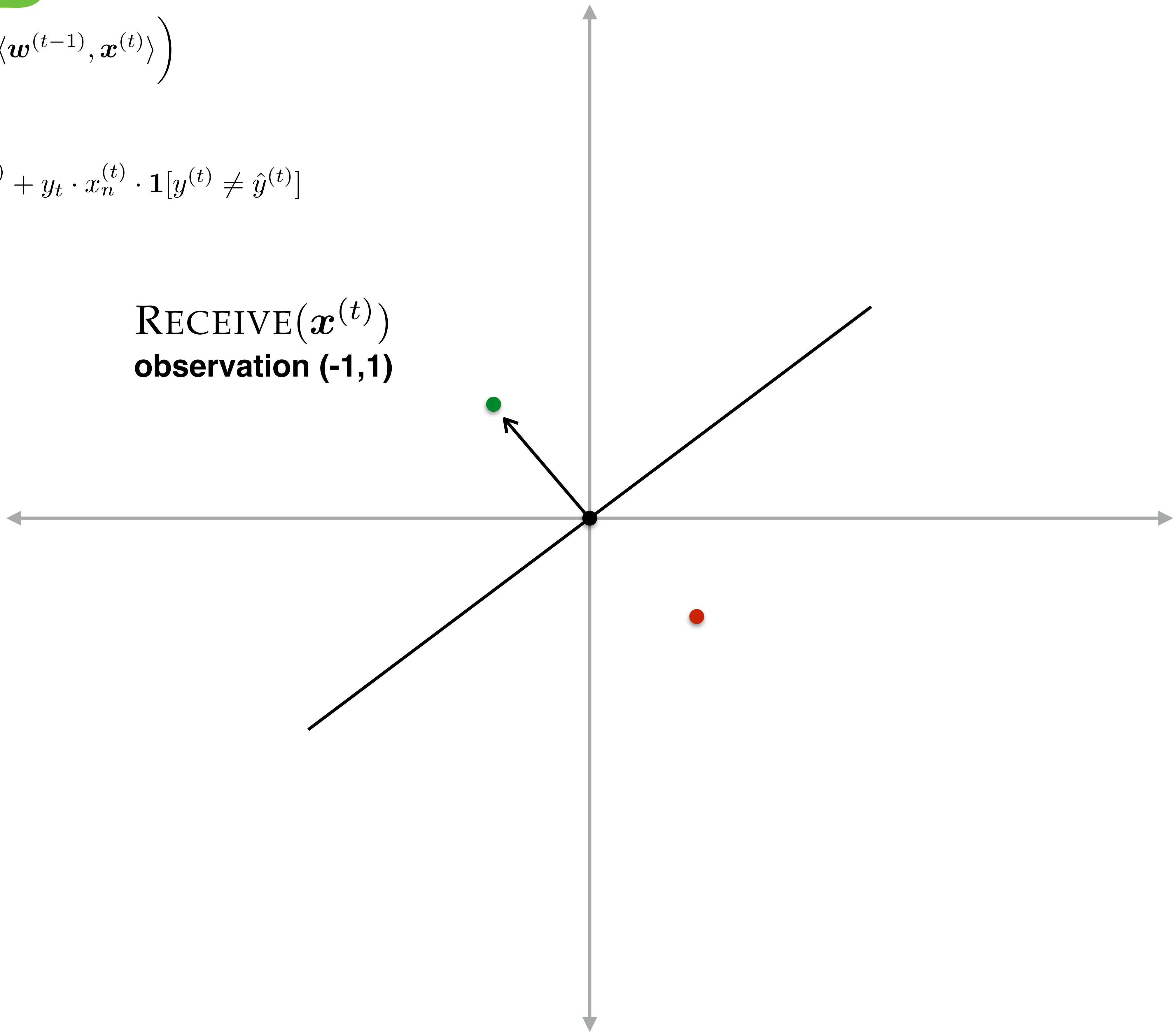
$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

(-1,1)

$\textsc{Receive}(\boldsymbol{x}^{(t)})$

observation (-1,1)



RECEIVE($\boldsymbol{x}^{(t)}$)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

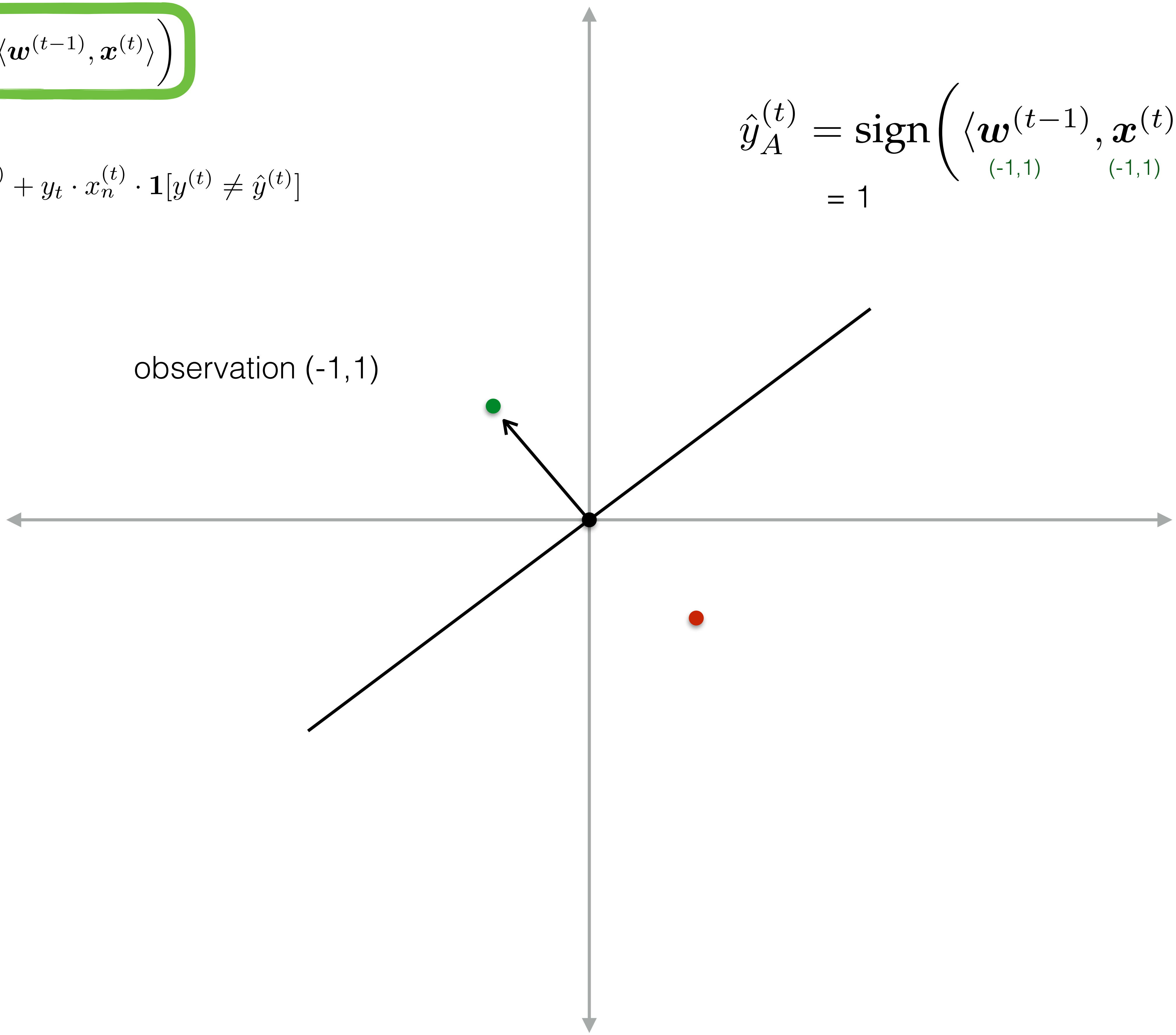
$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

(-1,1) (-1,1)

= 1



$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

$\textsc{Receive}(y^t)$

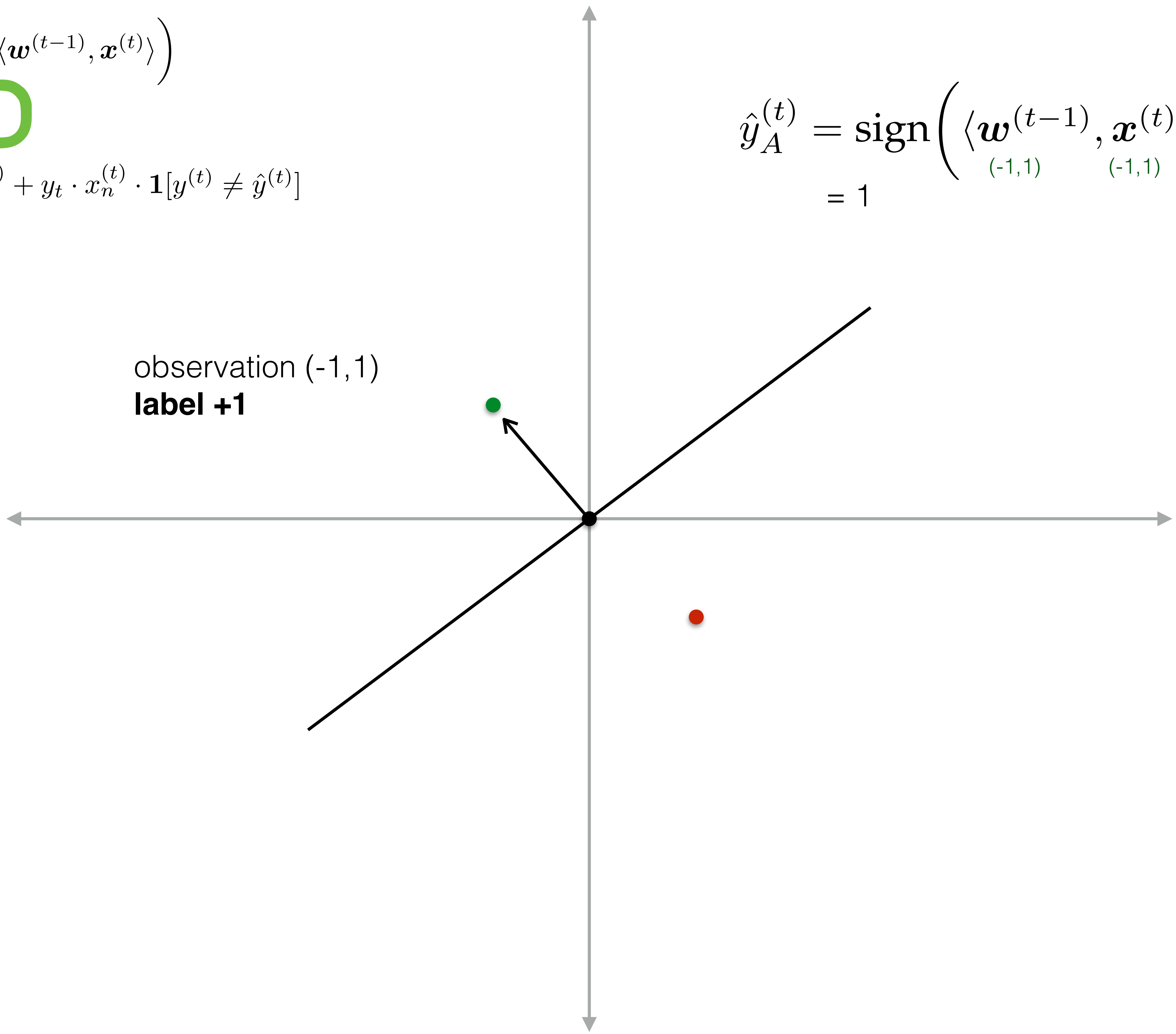
$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

(-1,1) (-1,1)

= 1



RECEIVE$(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \text{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$$

RECEIVE(y^t)

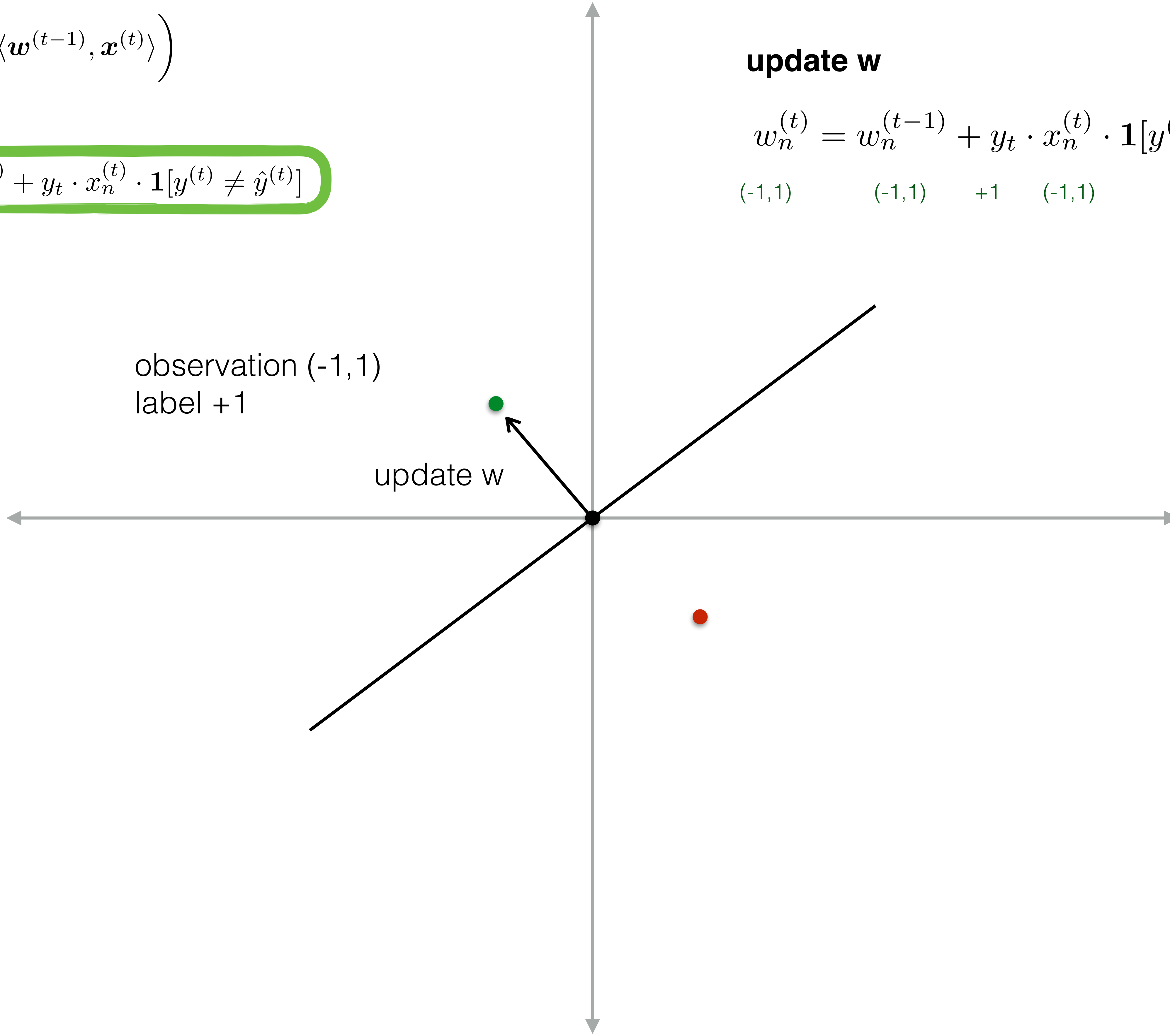
$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

update w

match!

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

(-1,1) (-1,1) +1 (-1,1) 0

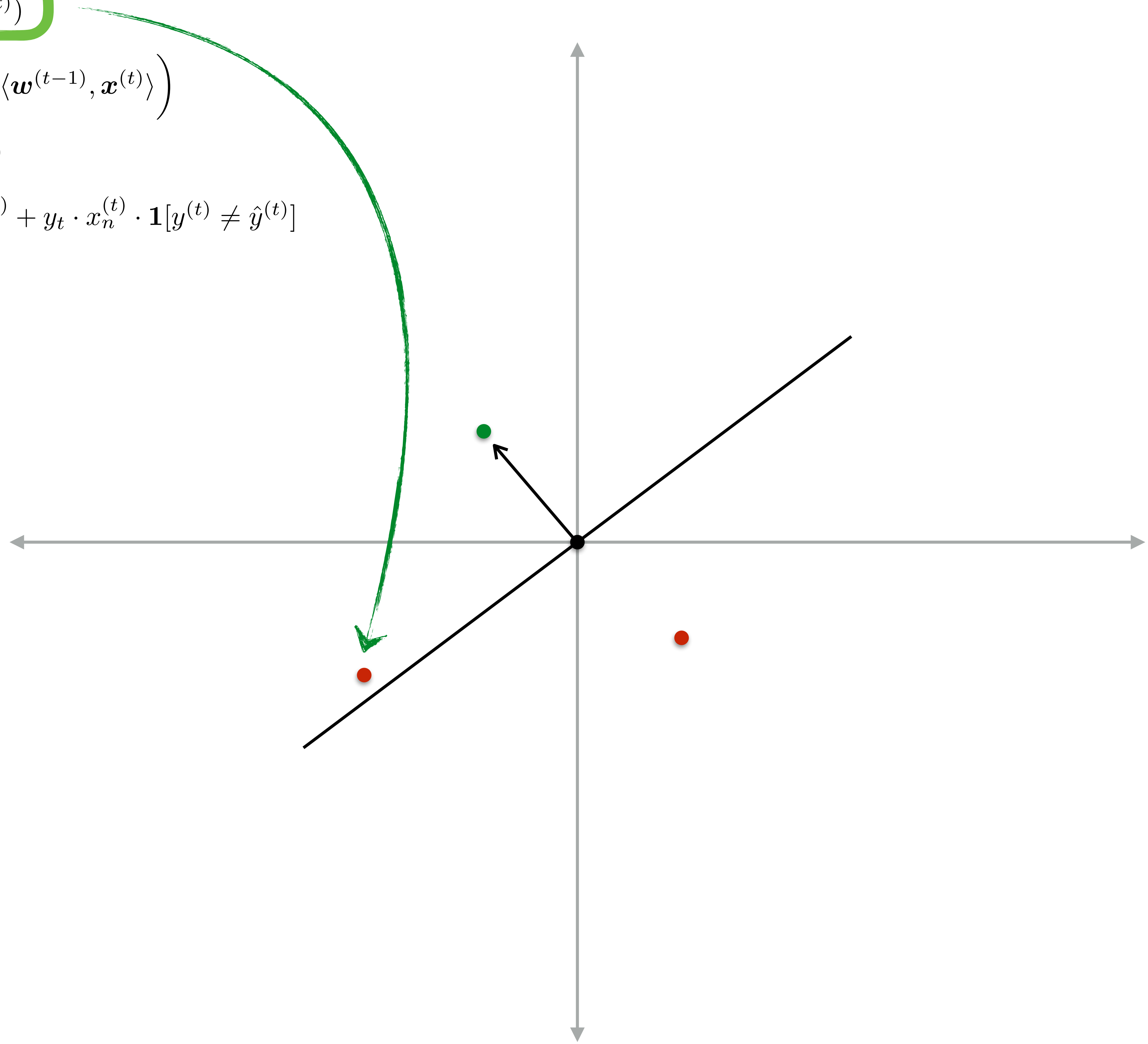


RECEIVE($\boldsymbol{x}^{(t)}$)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

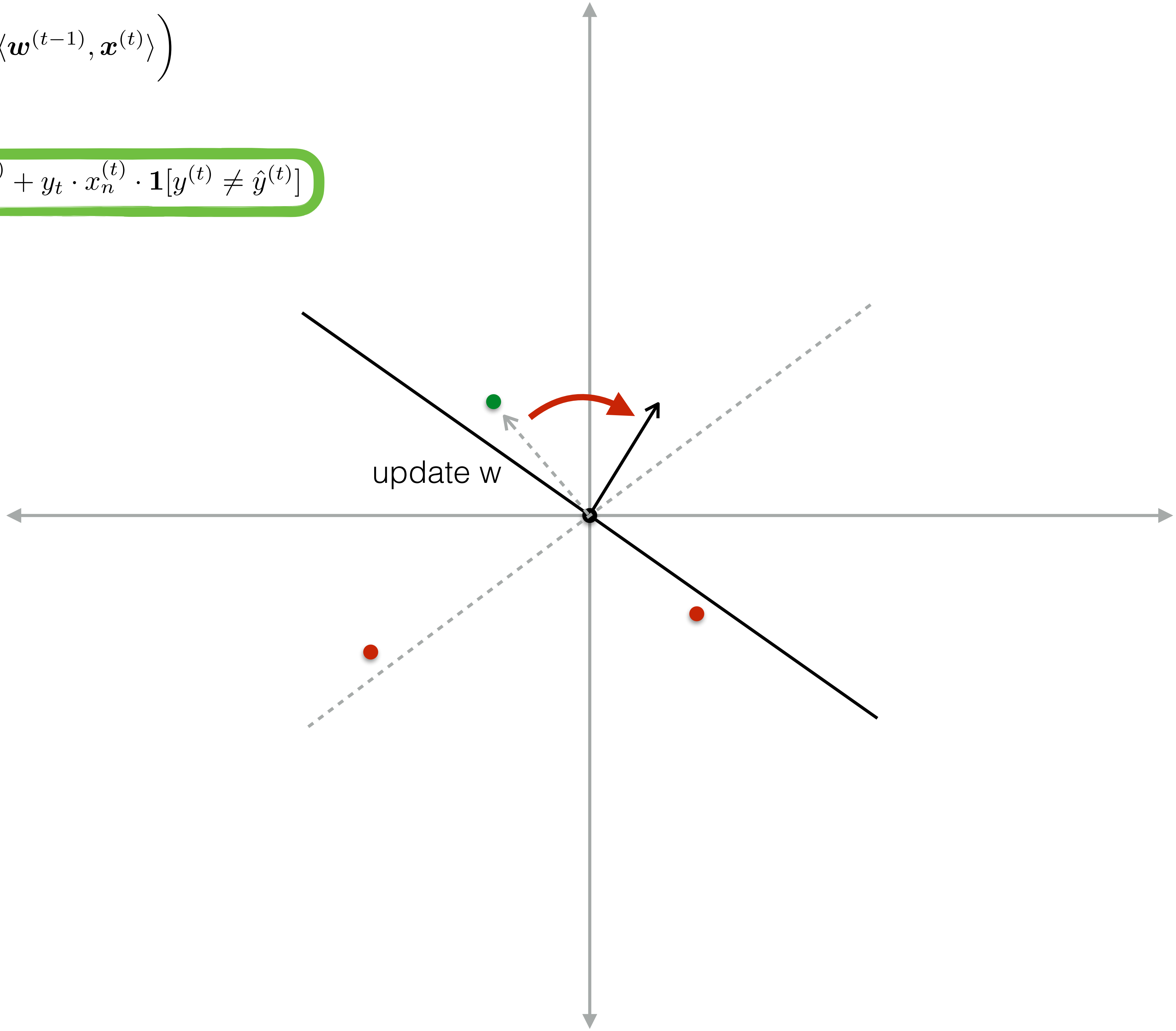


RECEIVE$(\boldsymbol{x}^{(t)})$

$$\hat{y}_A^{(t)} = \operatorname{sign}\left(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\right)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

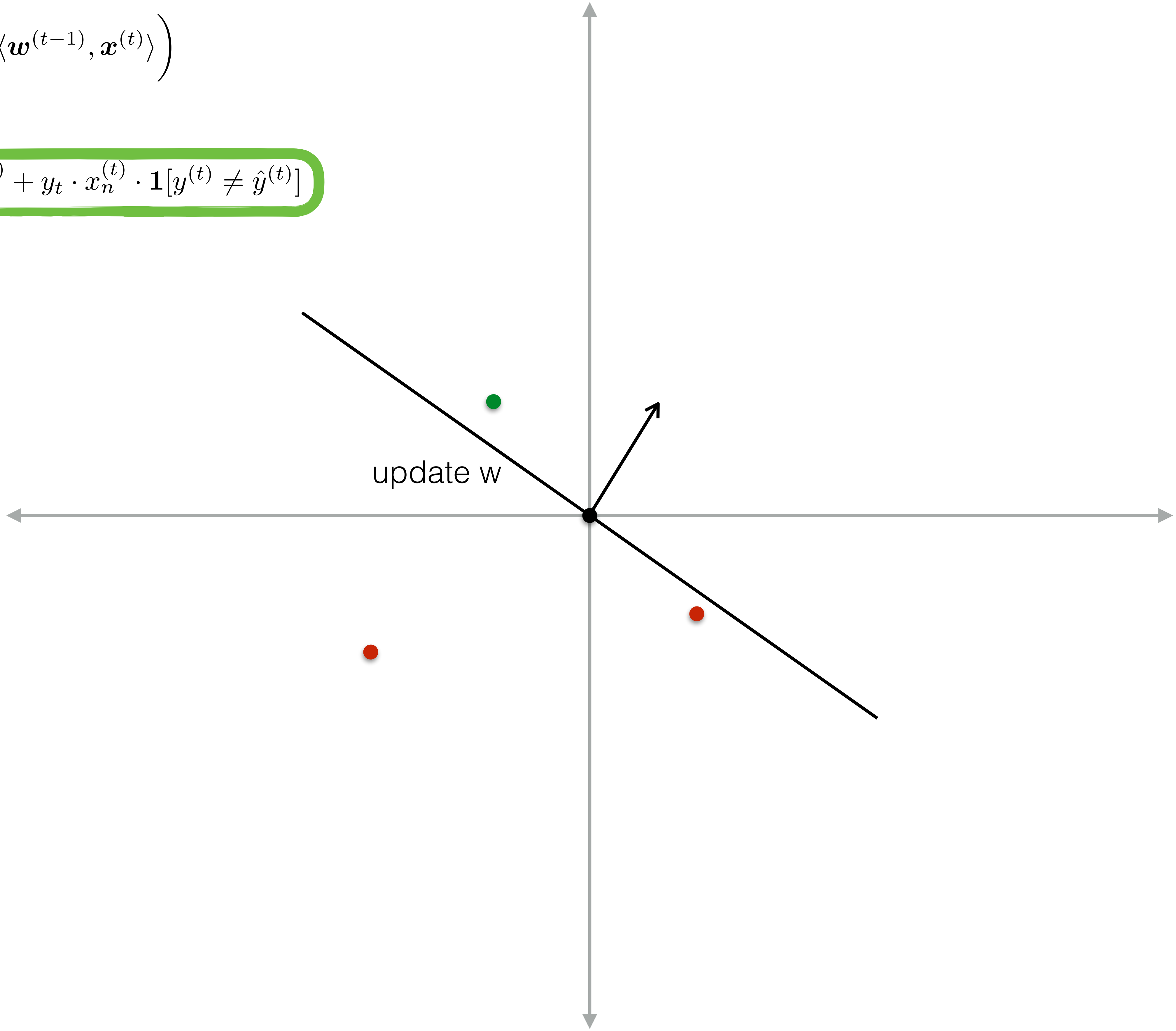


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

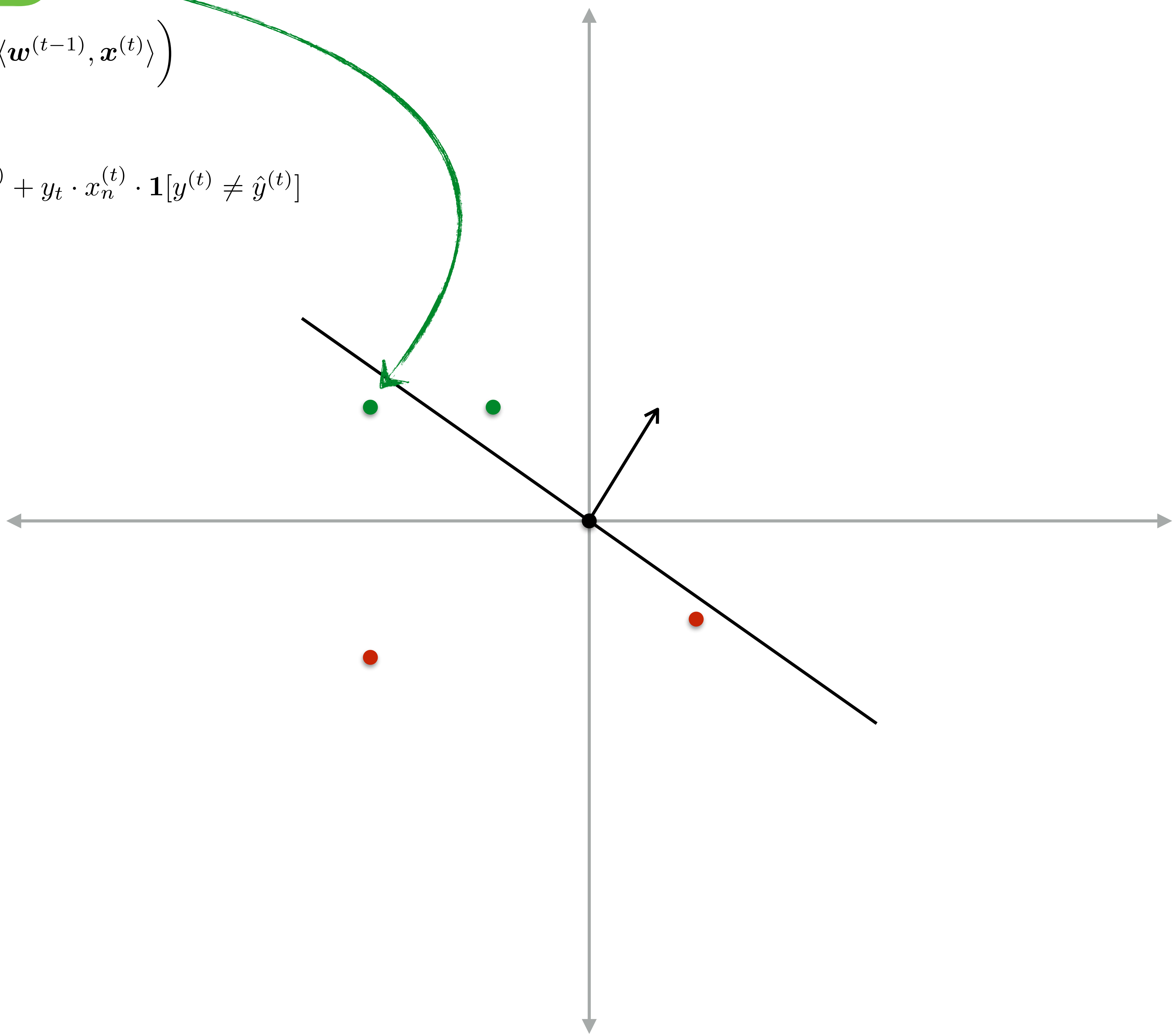


RECEIVE($\boldsymbol{x}^{(t)}$)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

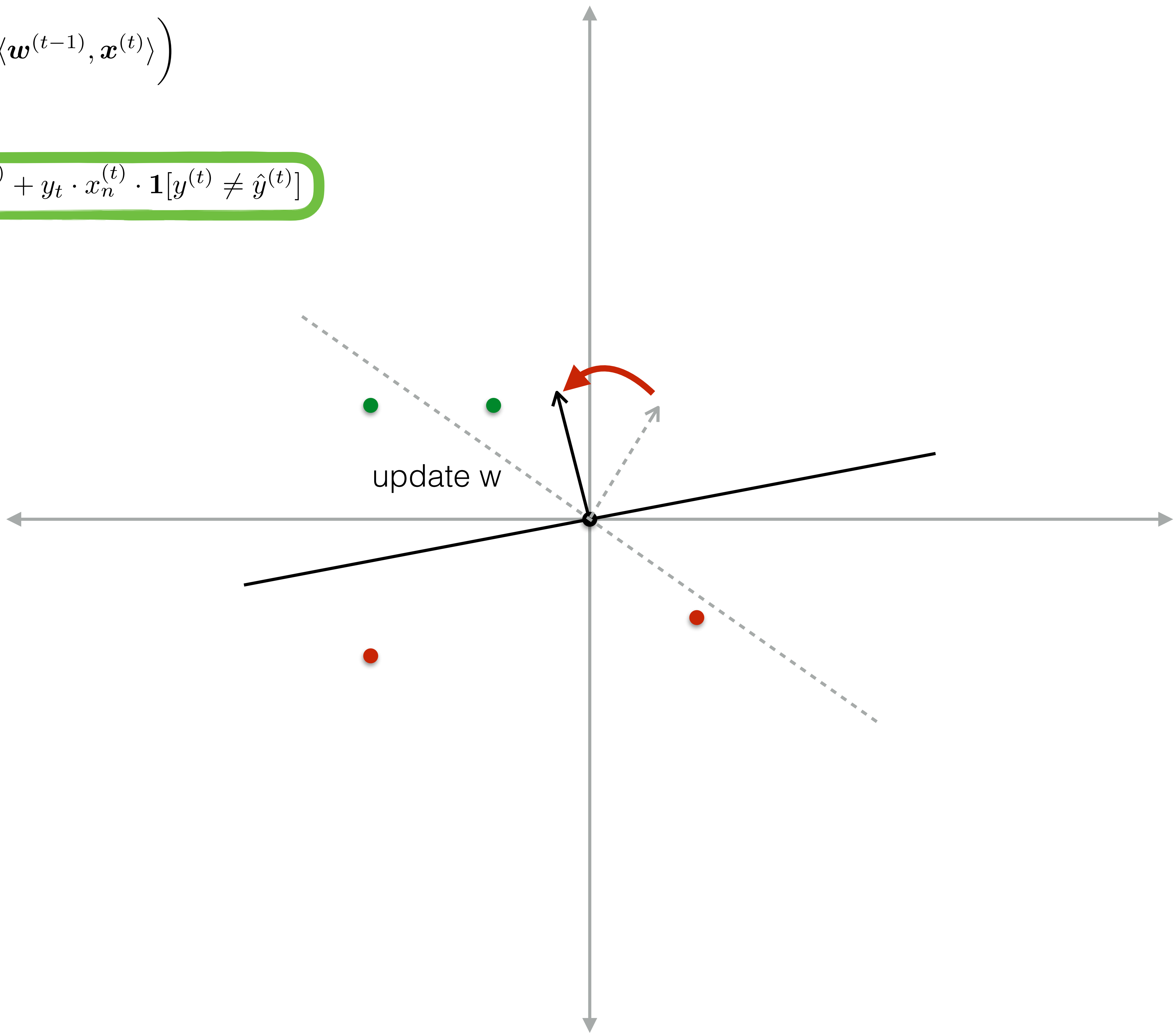


RECEIVE($\boldsymbol{x}^{(t)}$)

$$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

RECEIVE(y^t)

$$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$$

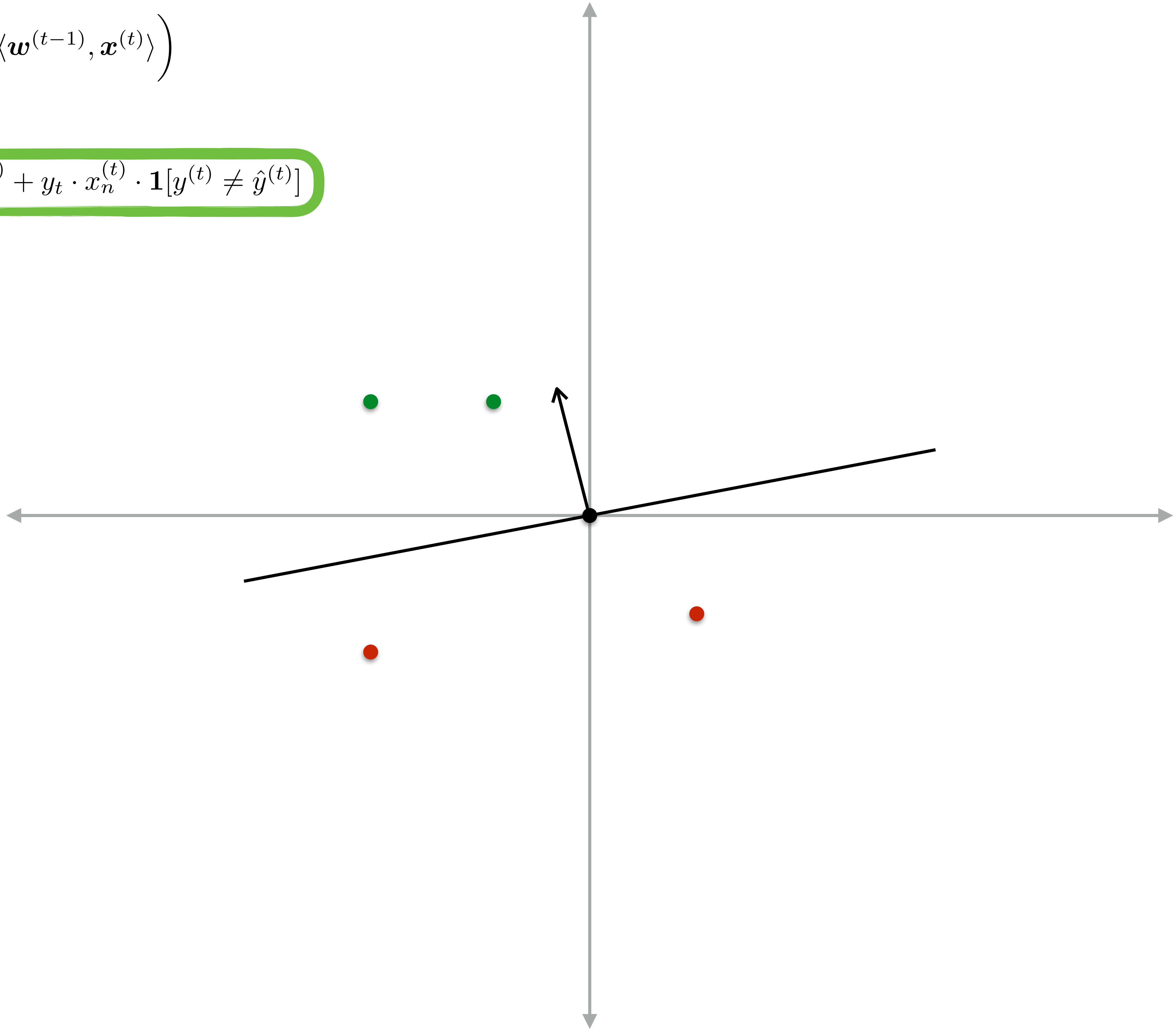


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

$\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$

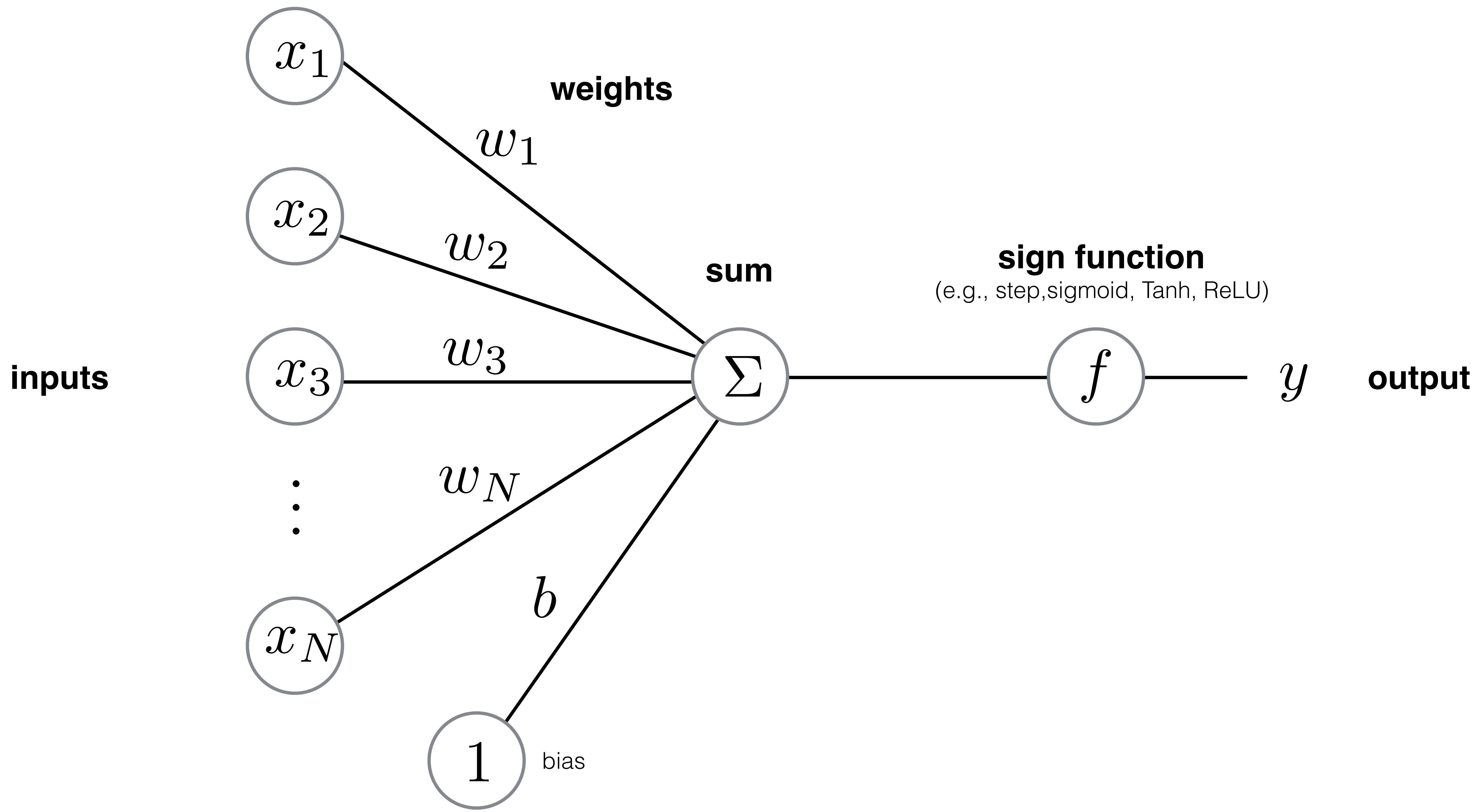
$\textsc{Receive}(y^t)$

$w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

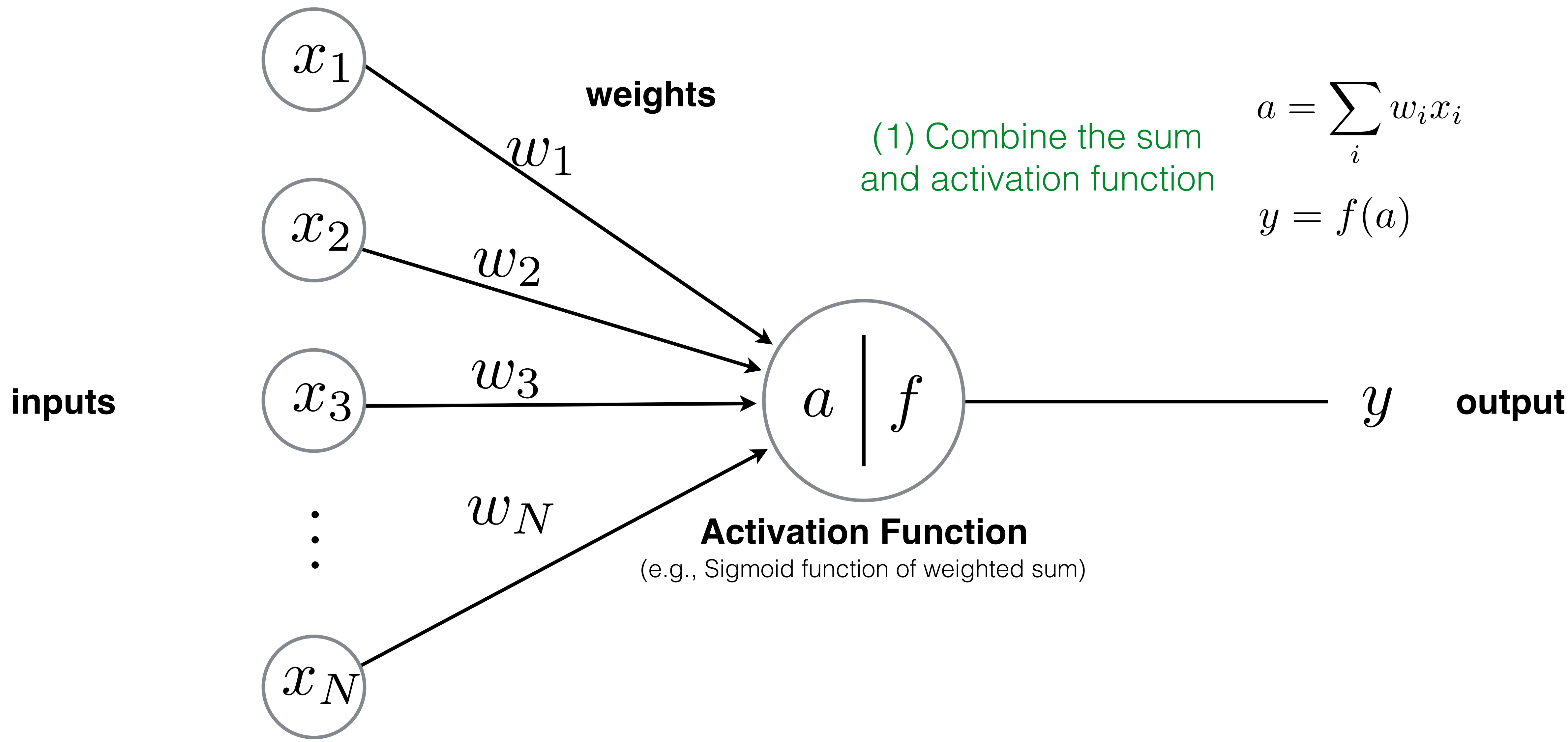


repeat …

The Perceptron



Another way to draw it…



(1) Combine the sum and activation function

$$a = \sum_i w_i x_i$$

$$y = f(a)$$

(2) suppress the bias term (less clutter)

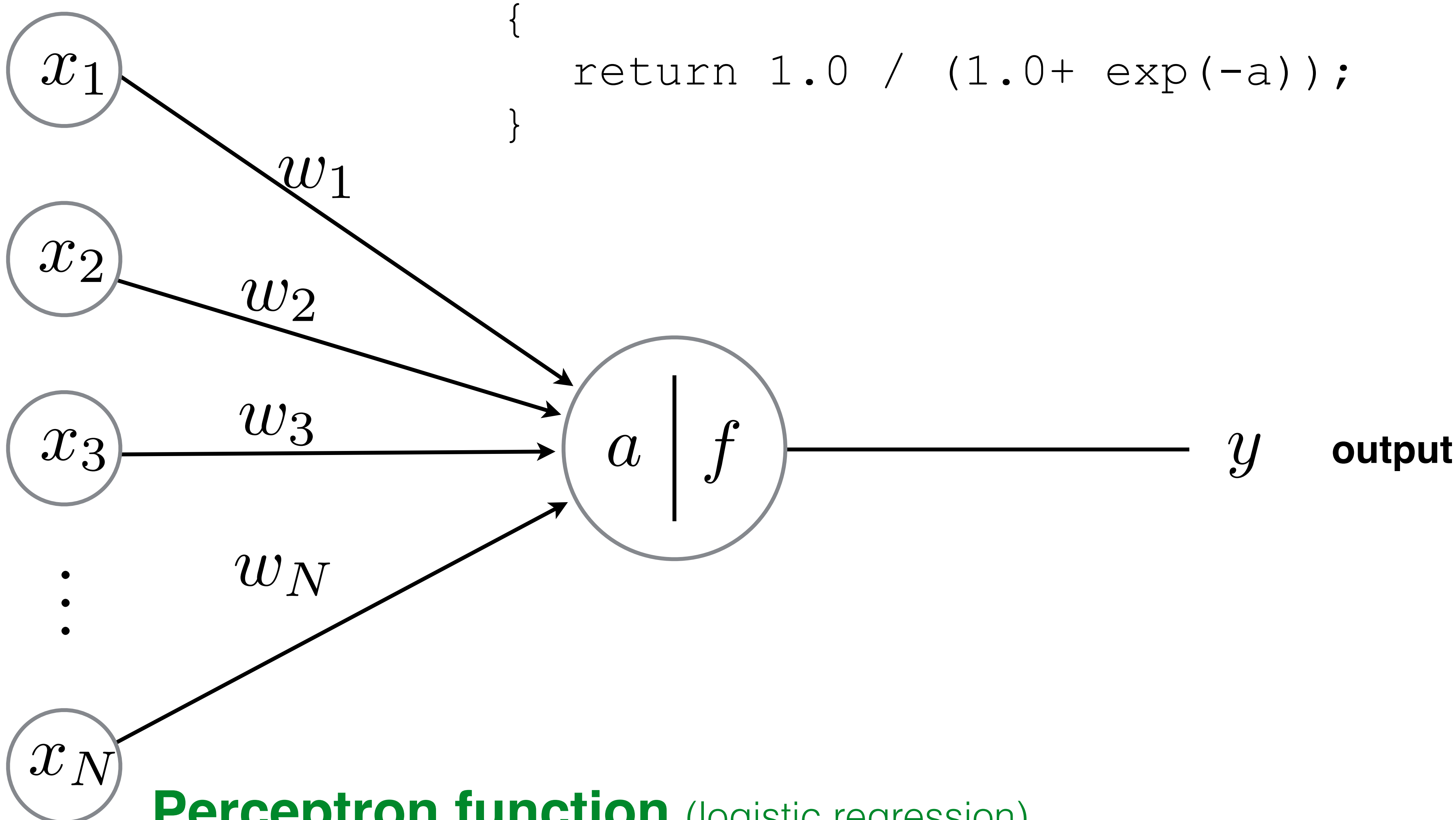
$$x_N = 1$$

$$w_N = b$$

Programming the 'forward pass'

Activation function (sigmoid, logistic function)

```
float f(float a)
{
   return 1.0 / (1.0+ exp(-a));
}
```



Perceptron function (logistic regression)

```
float perceptron(vector<float> x, vector<float> w)
{
   float a  = dot(x,w);
   return f(a);
}
```