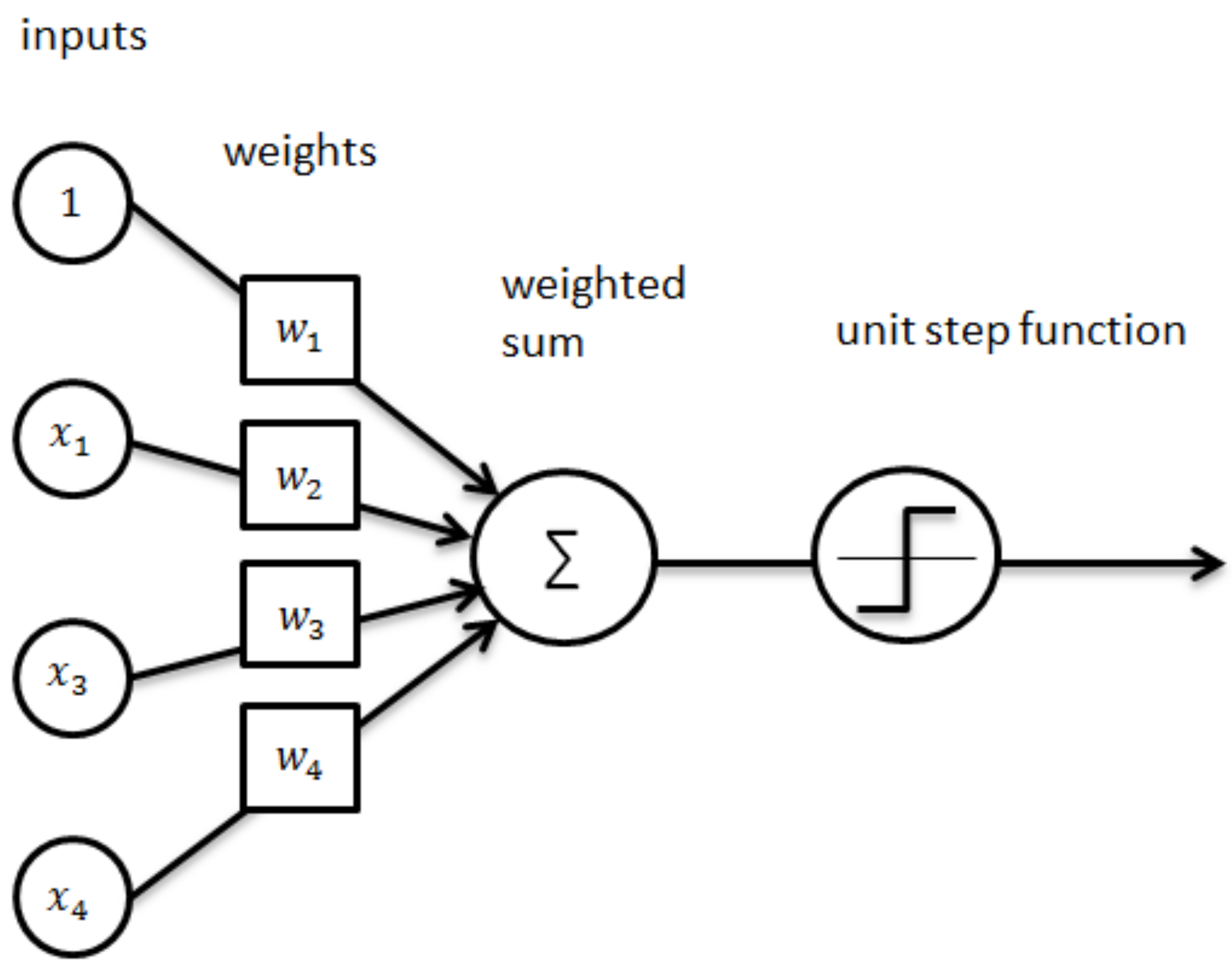
Introduction to neural networks



http://16385.courses.cs.cmu.edu/

16-385 Computer Vision
Spring 2022, Lecture 15 & 16

Overview of today’s lecture

- Perceptron.
- Neural networks.
- Training perceptrons.
- Gradient descent.
- Backpropagation.
- Stochastic gradient descent.

Slide credits

Most of these slides were adapted from:

- Kris Kitani (16-385, Spring 2017).
- Noah Snavely (Cornell University).
- Fei-Fei Li (Stanford University).
- Andrej Karpathy (Stanford University).

Perceptron

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

$$x_j = \sum_i y_i w_{ji} \tag{1}$$

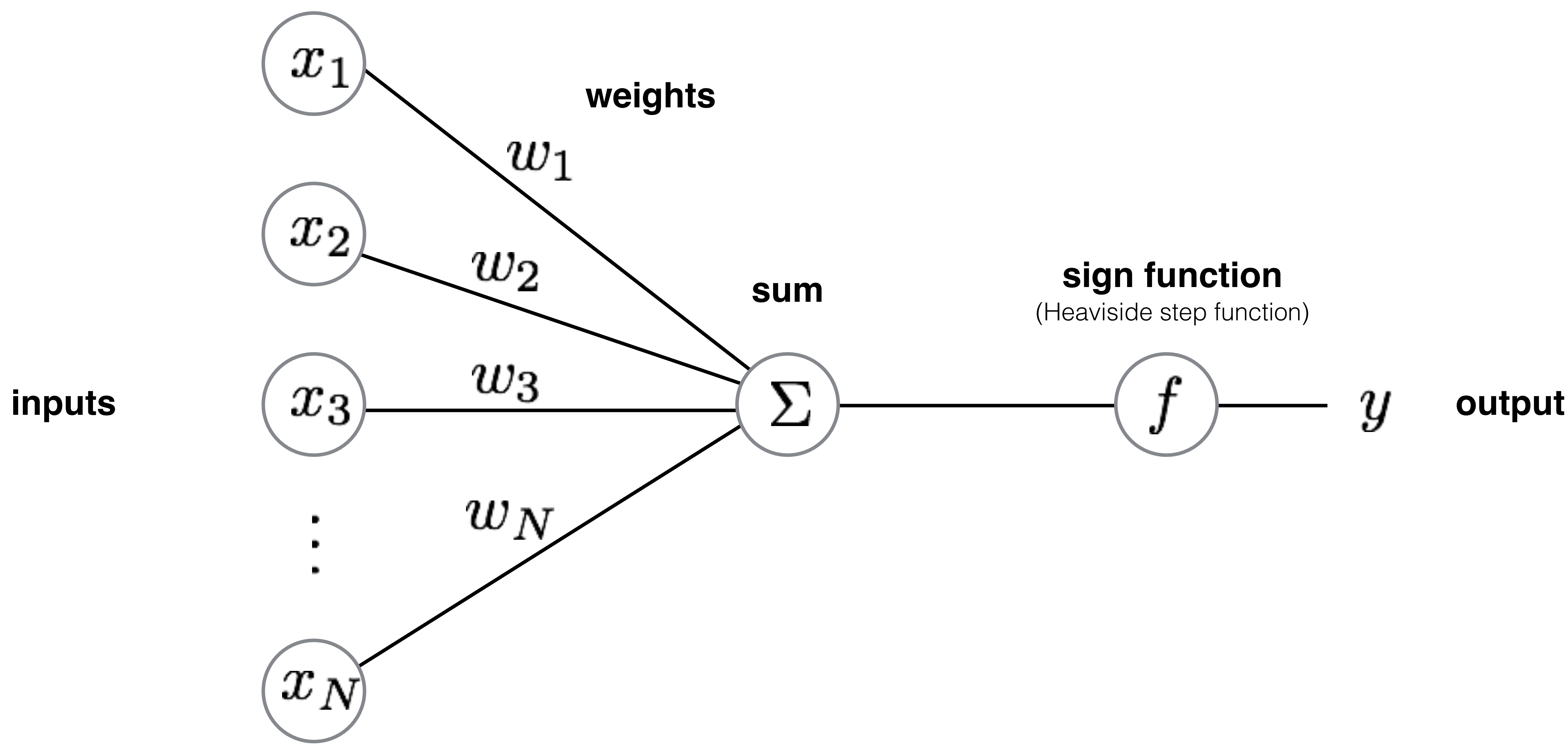
Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

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

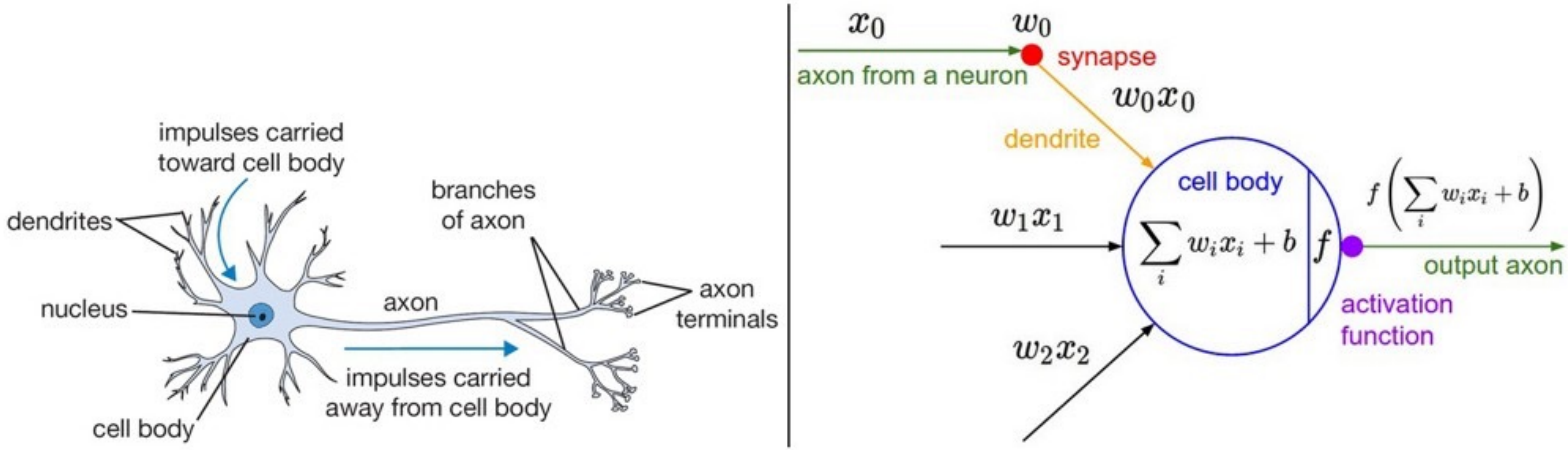
$$y_j = \frac{1}{1+e^{-x_j}} \tag{2}$$

† To whom correspondence should be addressed.

The Perceptron



Aside: Inspiration from Biology



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Neural nets/perceptrons are **loosely** inspired by biology.

But they certainly are **not** a model of how the brain works, or even how neurons work.

function PERCEPTRON ALGORITHM

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

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

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

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

perceptron is just one line of code!

sign of zero is +1

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

$y \in \{1, -1\}$

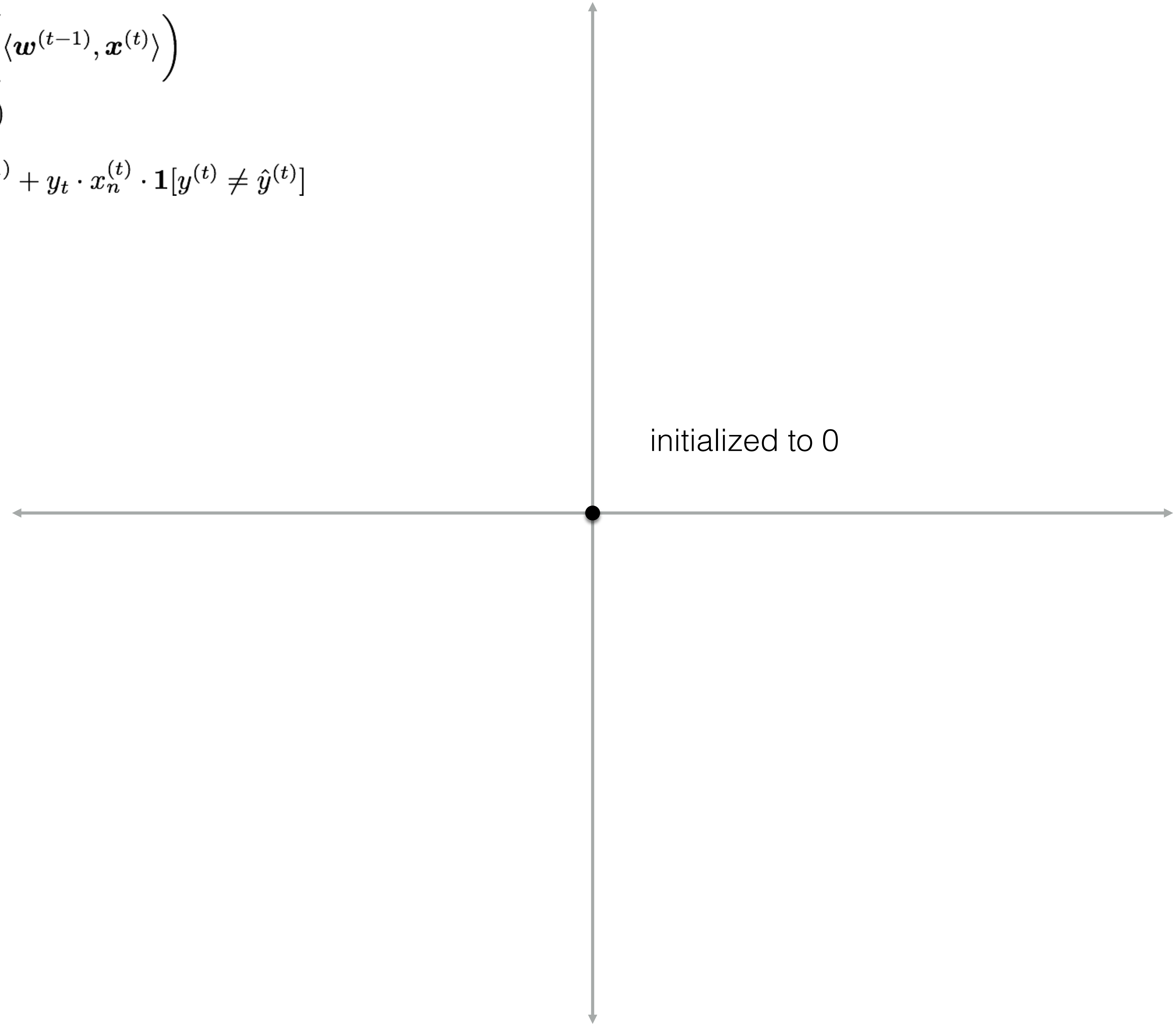
$\qquad w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]$

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

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

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

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

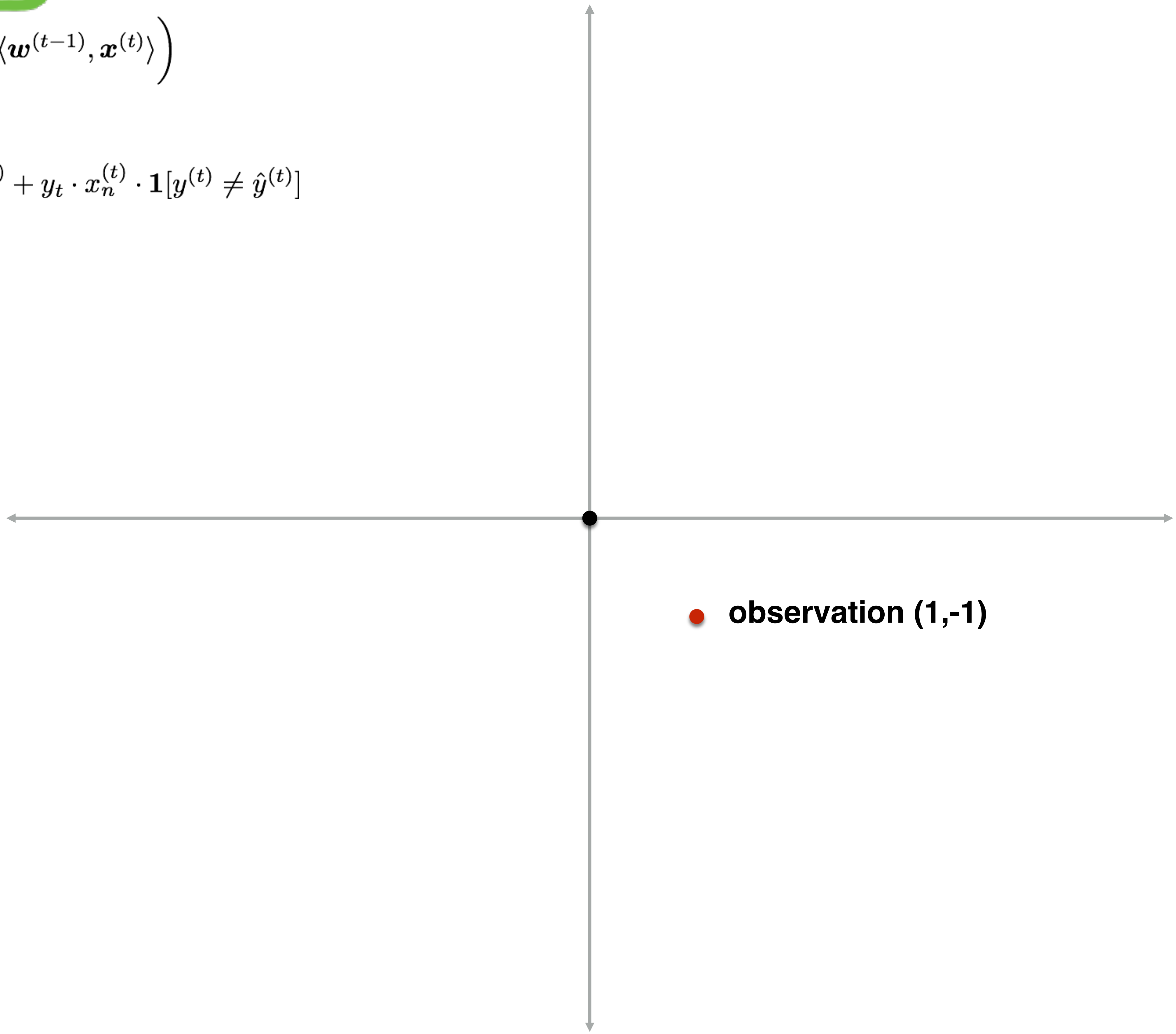


$\boxed{\textsc{Receive}(\boldsymbol{x}^{(t)})}$

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

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

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

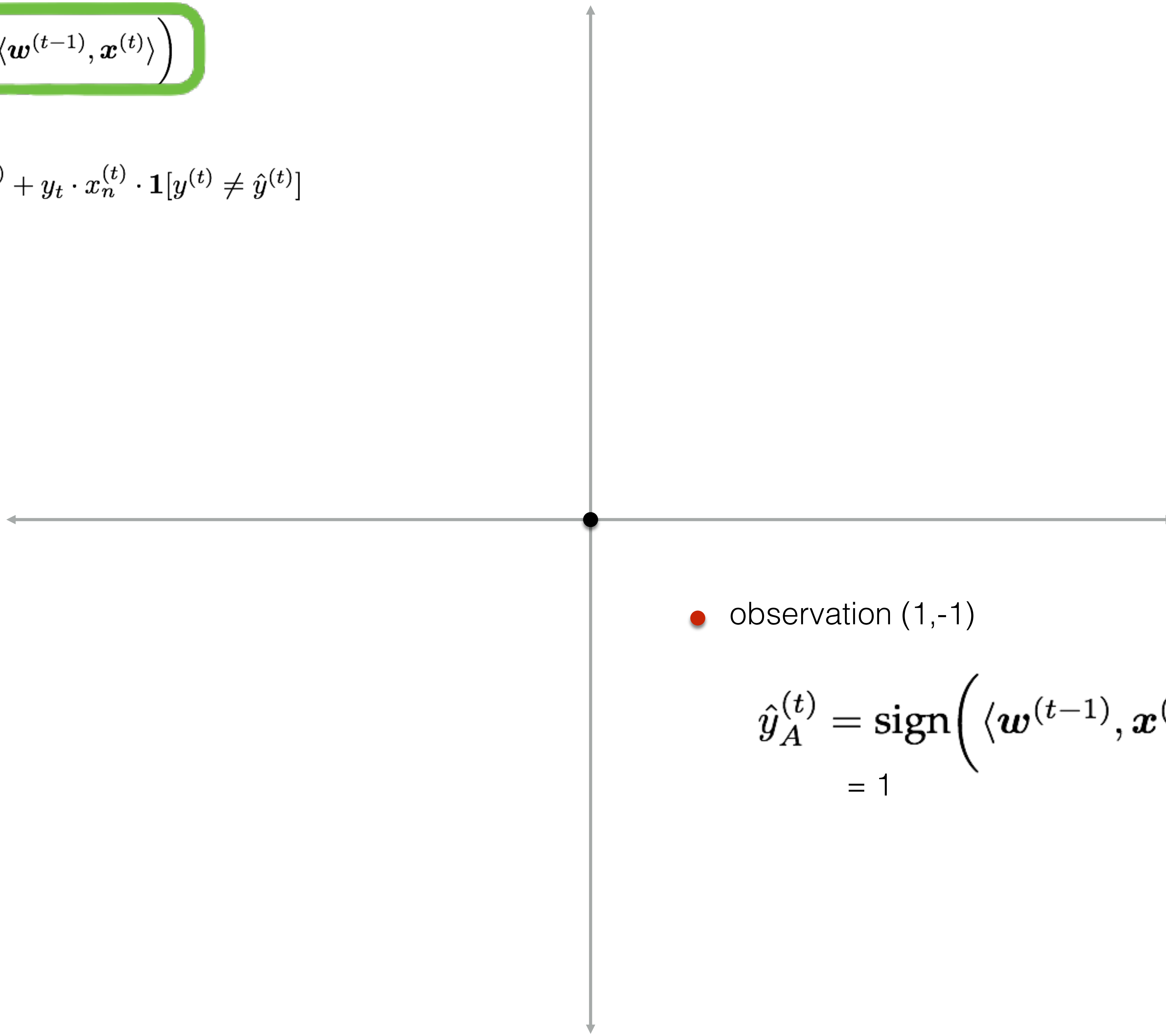


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

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

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

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



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

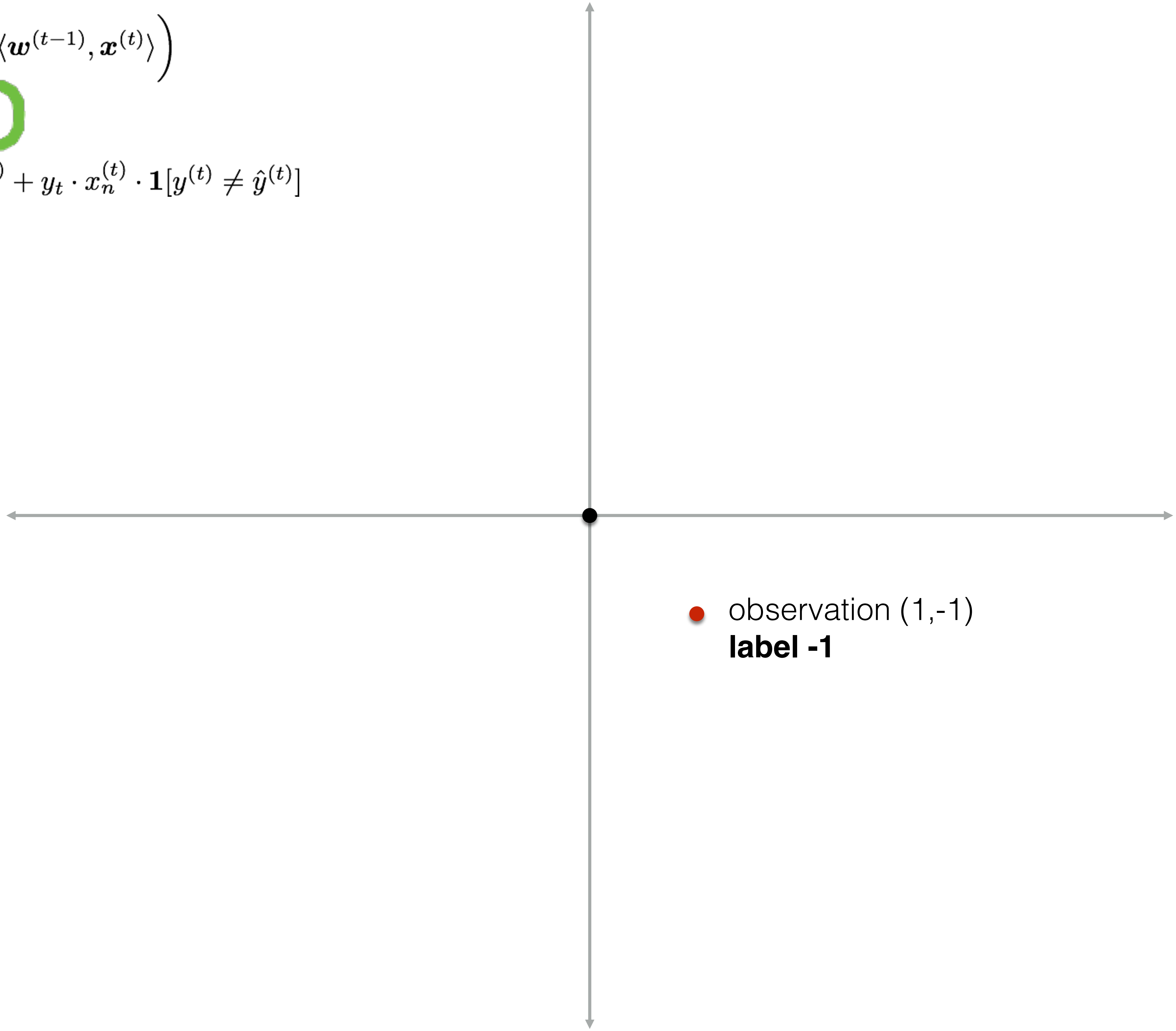
= 1

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

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

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

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



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

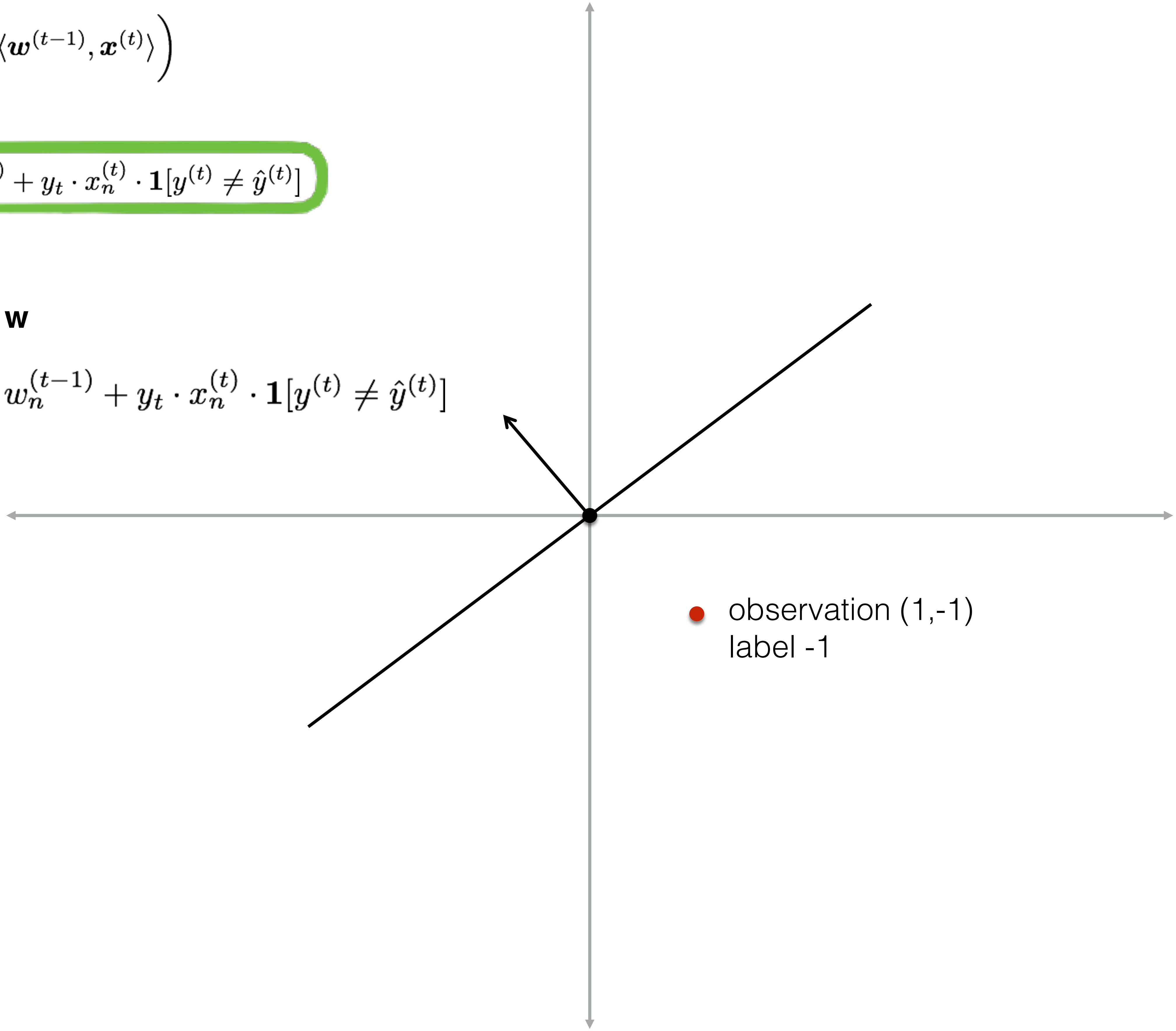
$$\hat{y}_A^{(t)} = \mathrm{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

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

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

update w

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



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

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

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

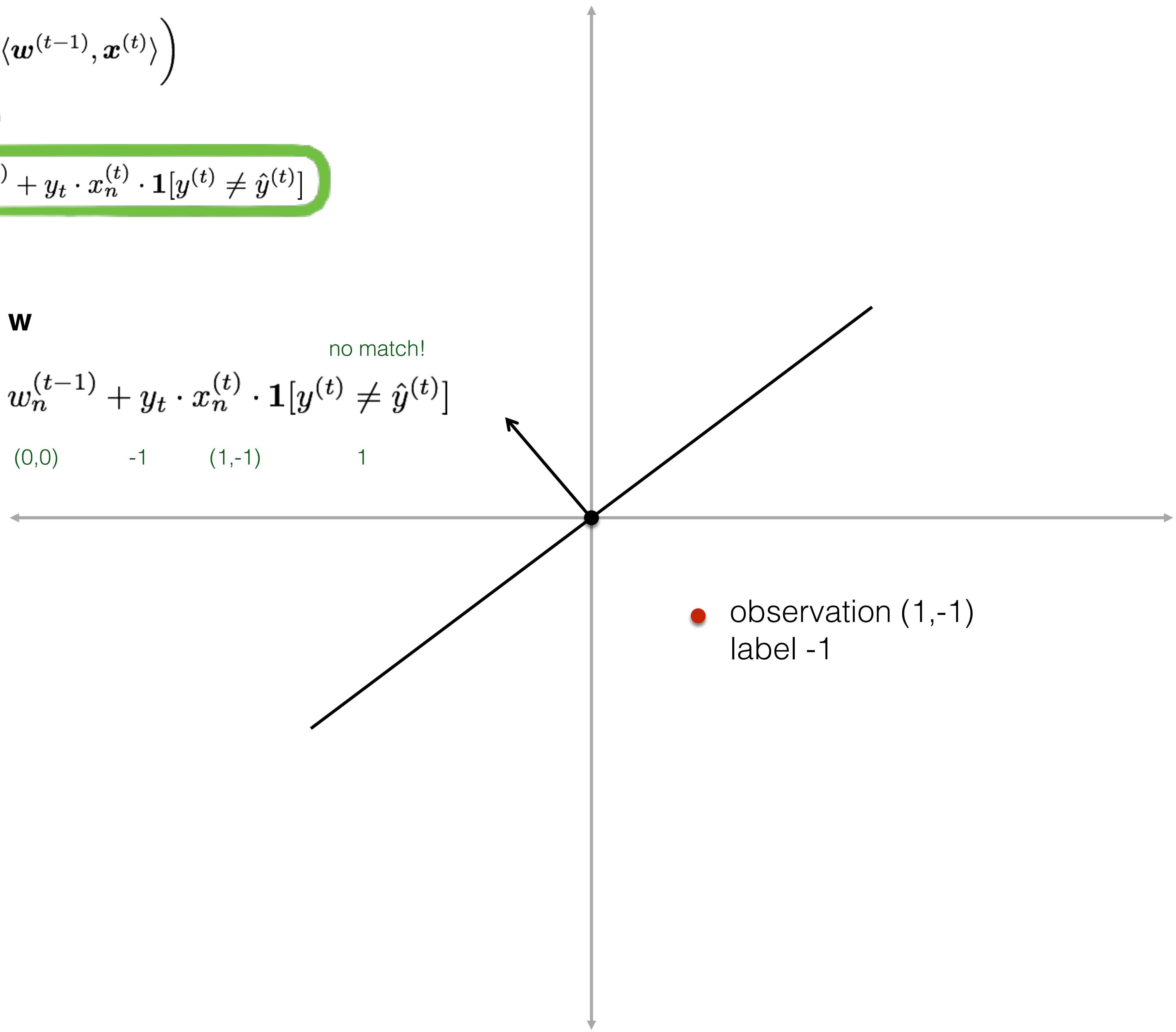
$$\boxed{w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]}$$

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



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

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

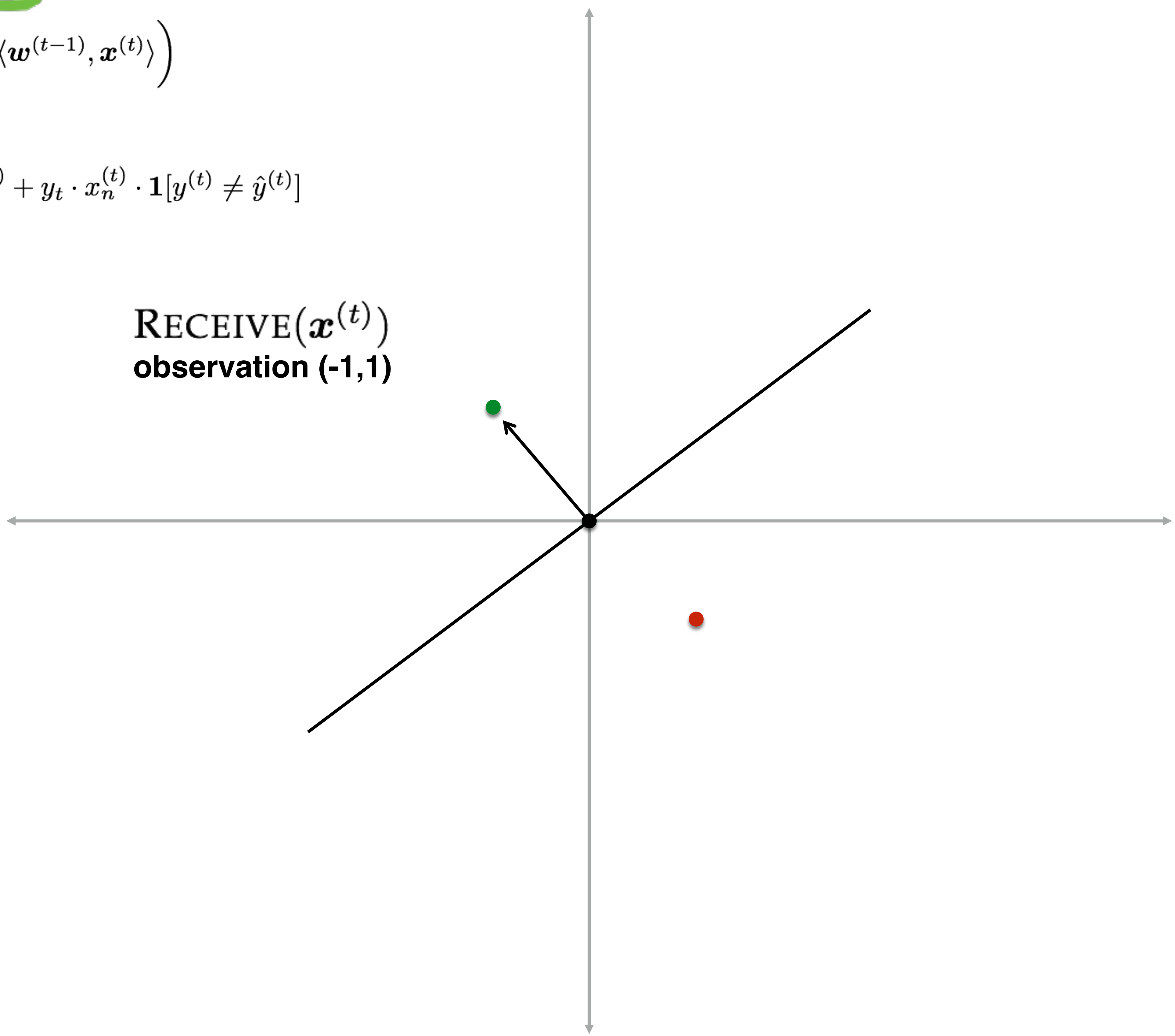
$\textsc{Receive}(y^t)$

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

(-1,1)

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

observation (-1,1)



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

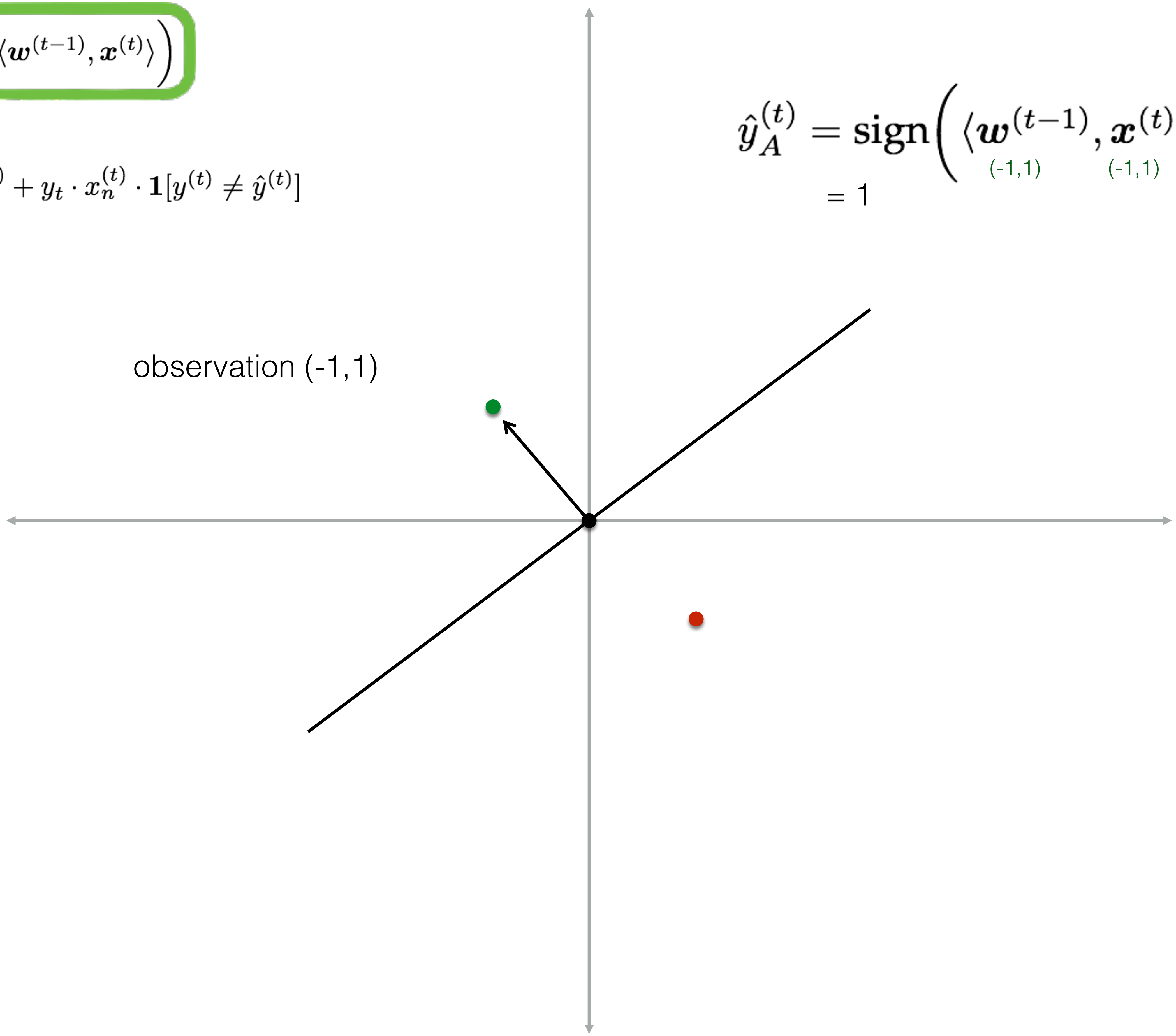
$$\boxed{\hat{y}_A^{(t)} = \text{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)}$$

RECEIVE(y^t)

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

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

$= 1$



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

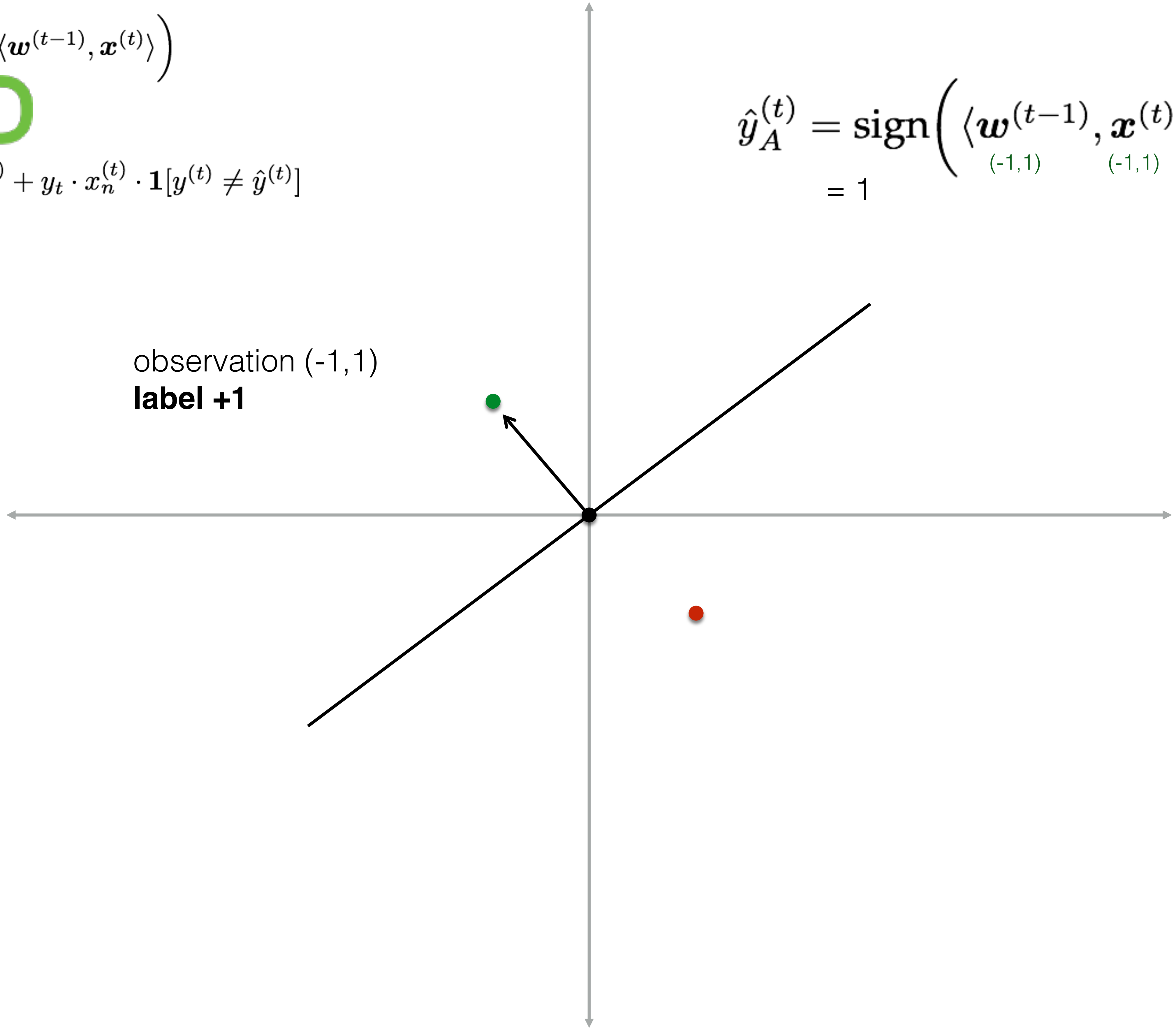
$$\hat{y}_A^{(t)} = \mathrm{sign}\Big(\langle \boldsymbol{w}^{(t-1)}, \boldsymbol{x}^{(t)} \rangle\Big)$$

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

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

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

$= 1$



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

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

$\text{RECEIVE}(y^t)$

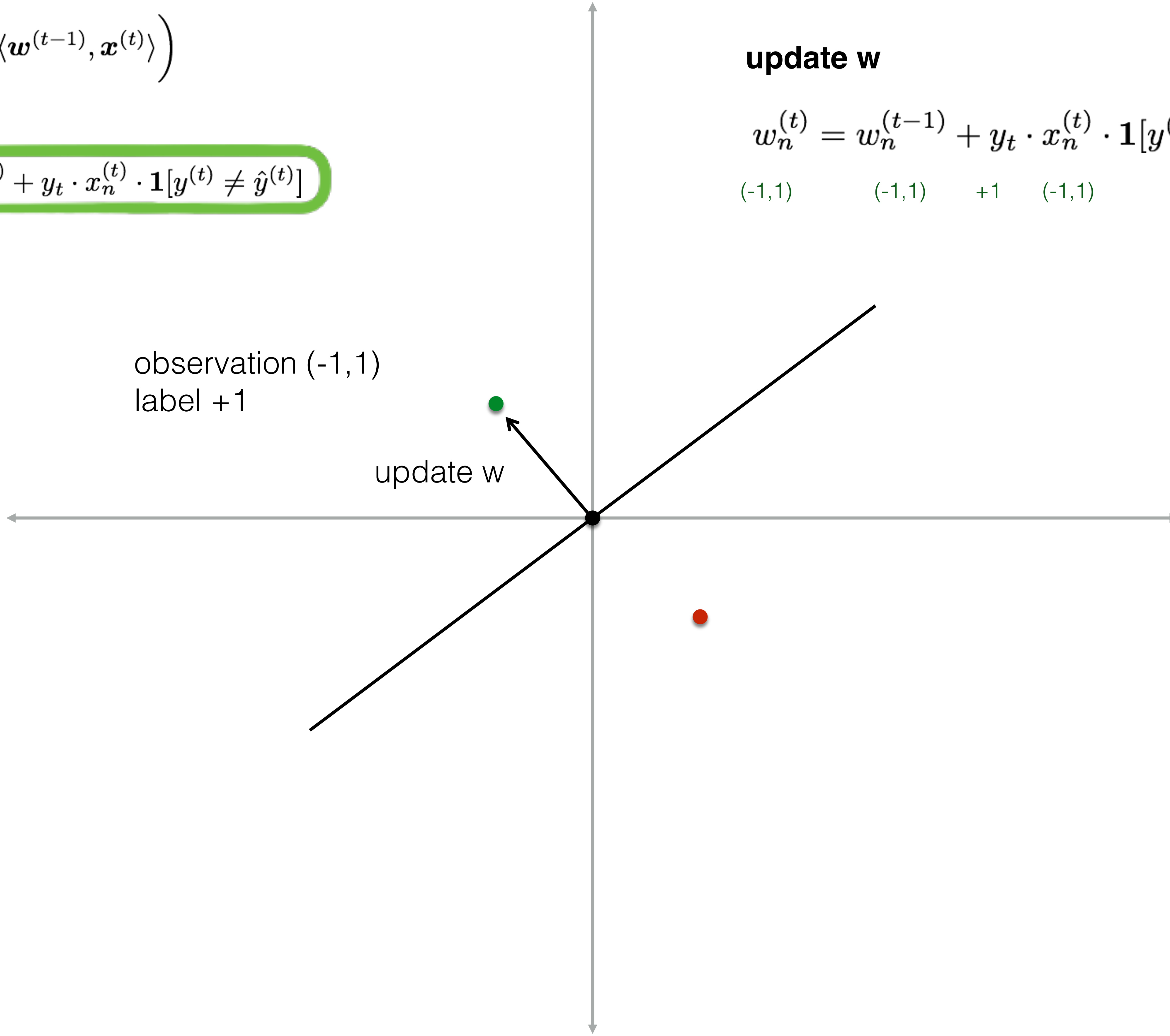
$$\boxed{w_n^{(t)} = w_n^{(t-1)} + y_t \cdot x_n^{(t)} \cdot \mathbf{1}[y^{(t)} \neq \hat{y}^{(t)}]}$$

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

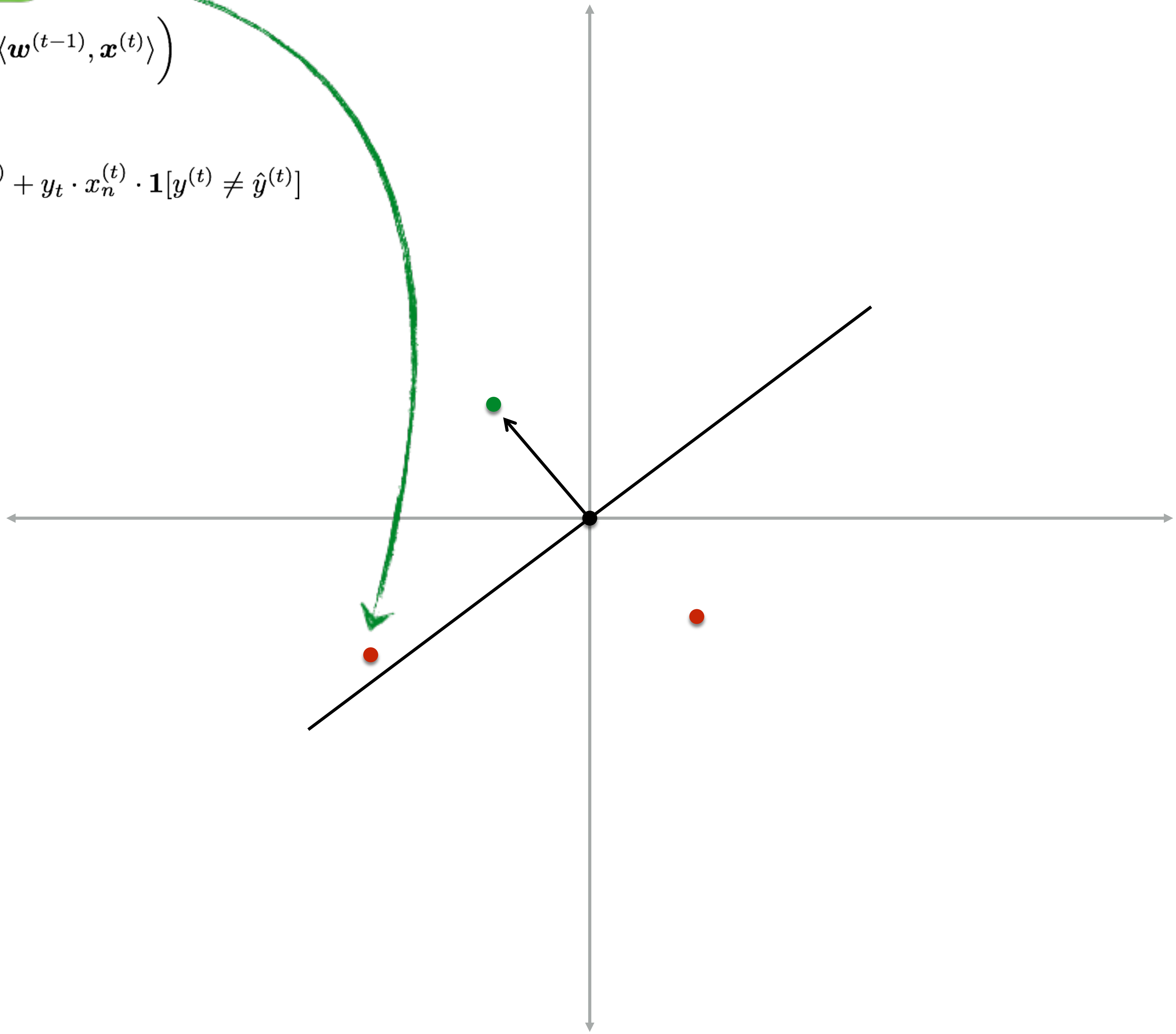


$\boxed{\textsc{Receive}(\boldsymbol{x}^{(t)})}$

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

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

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

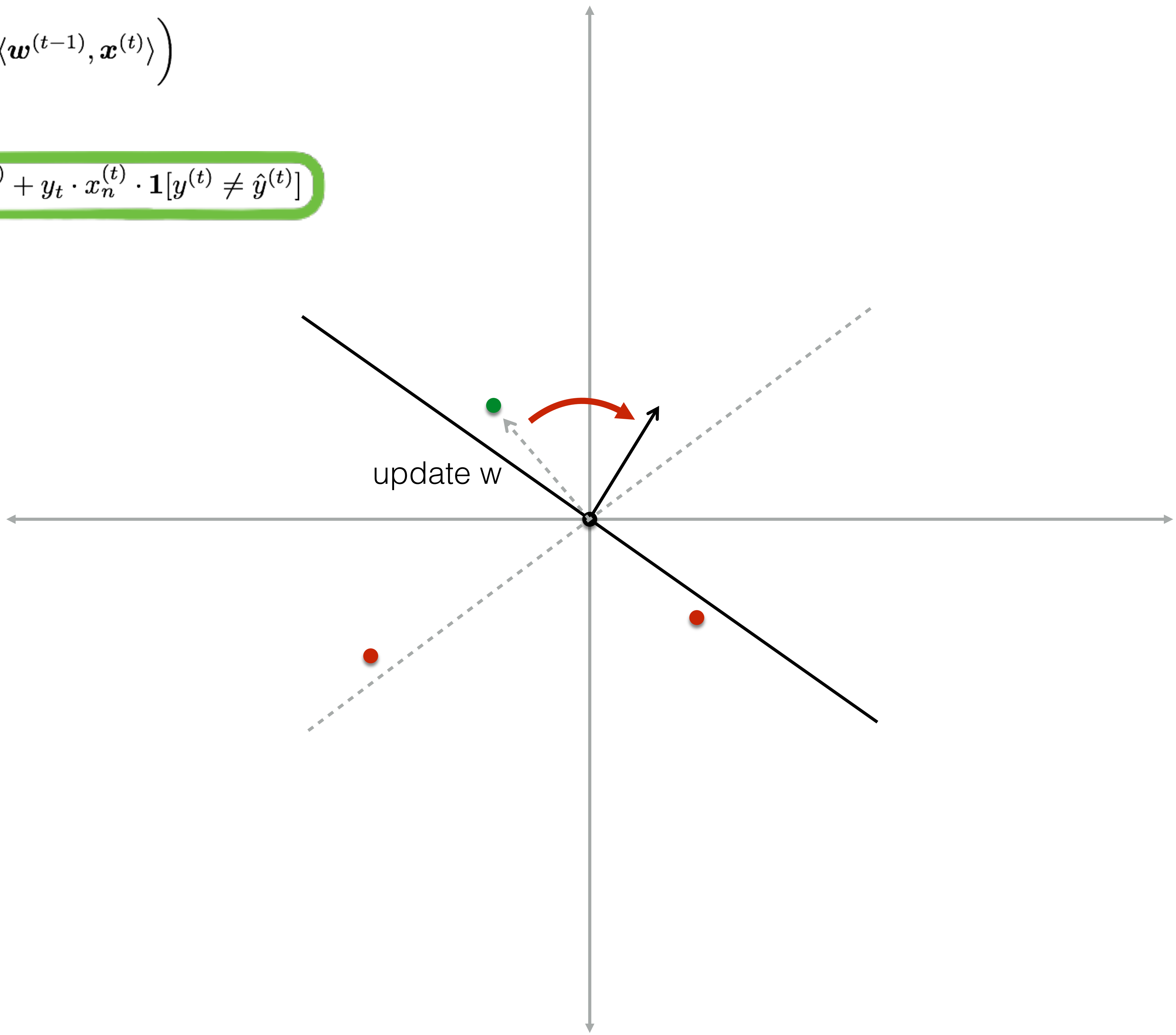


RECEIVE($\boldsymbol{x}^{(t)}$)

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

RECEIVE(y^t)

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

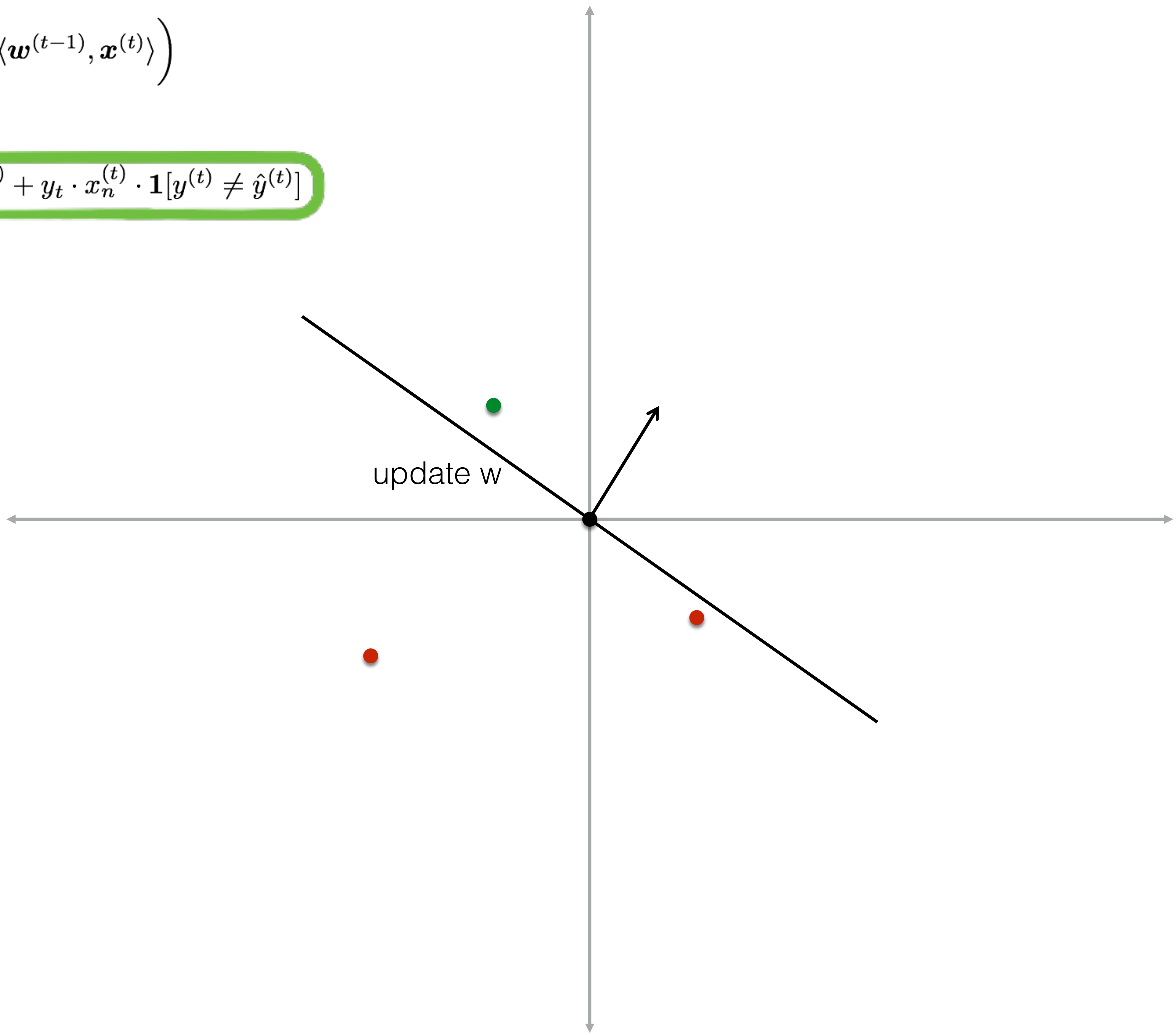


$\textsc{Receive}(\boldsymbol{x}^{(t)})$

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

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

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

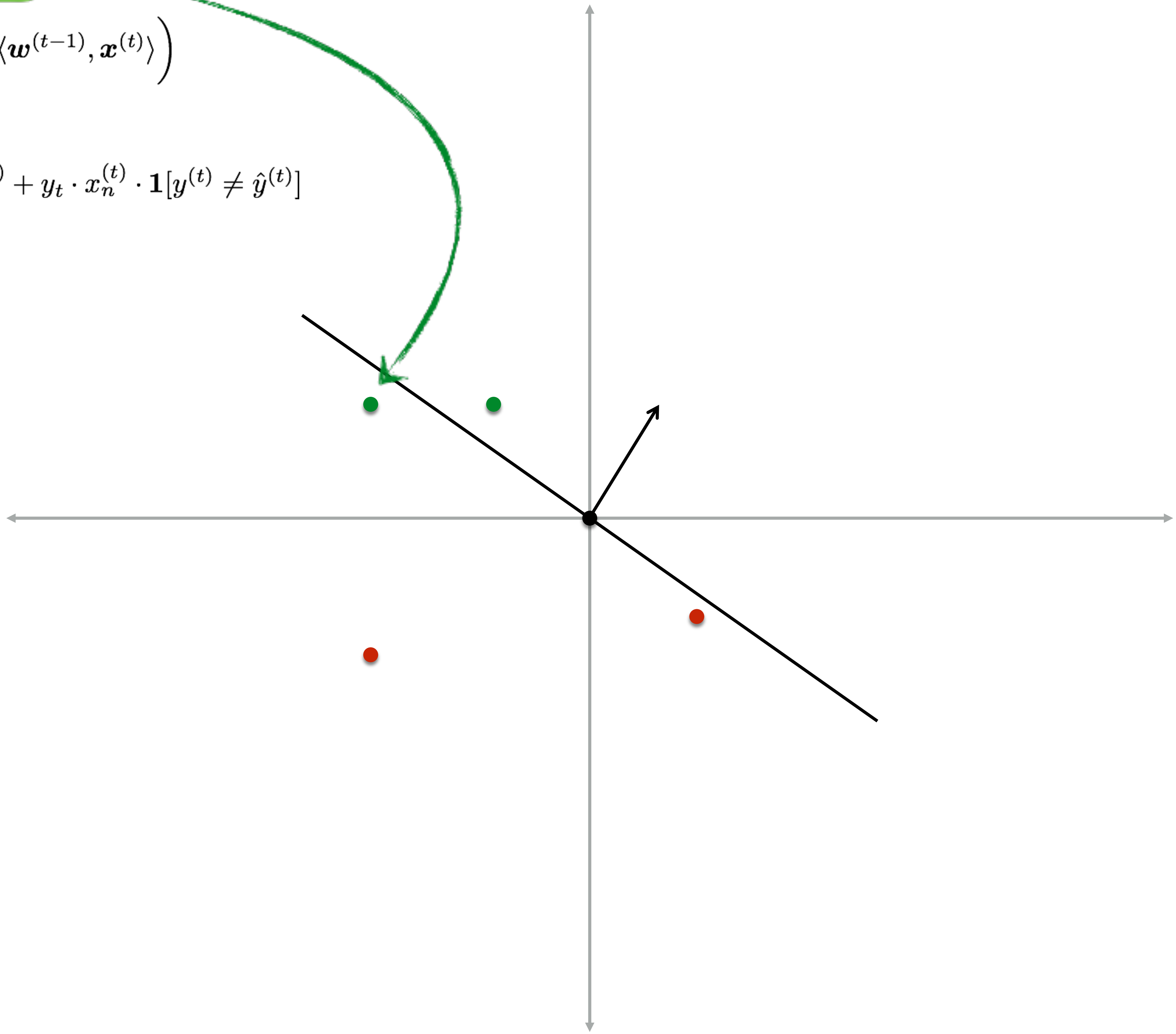


$\boxed{\textsc{Receive}(\boldsymbol{x}^{(t)})}$

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

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

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

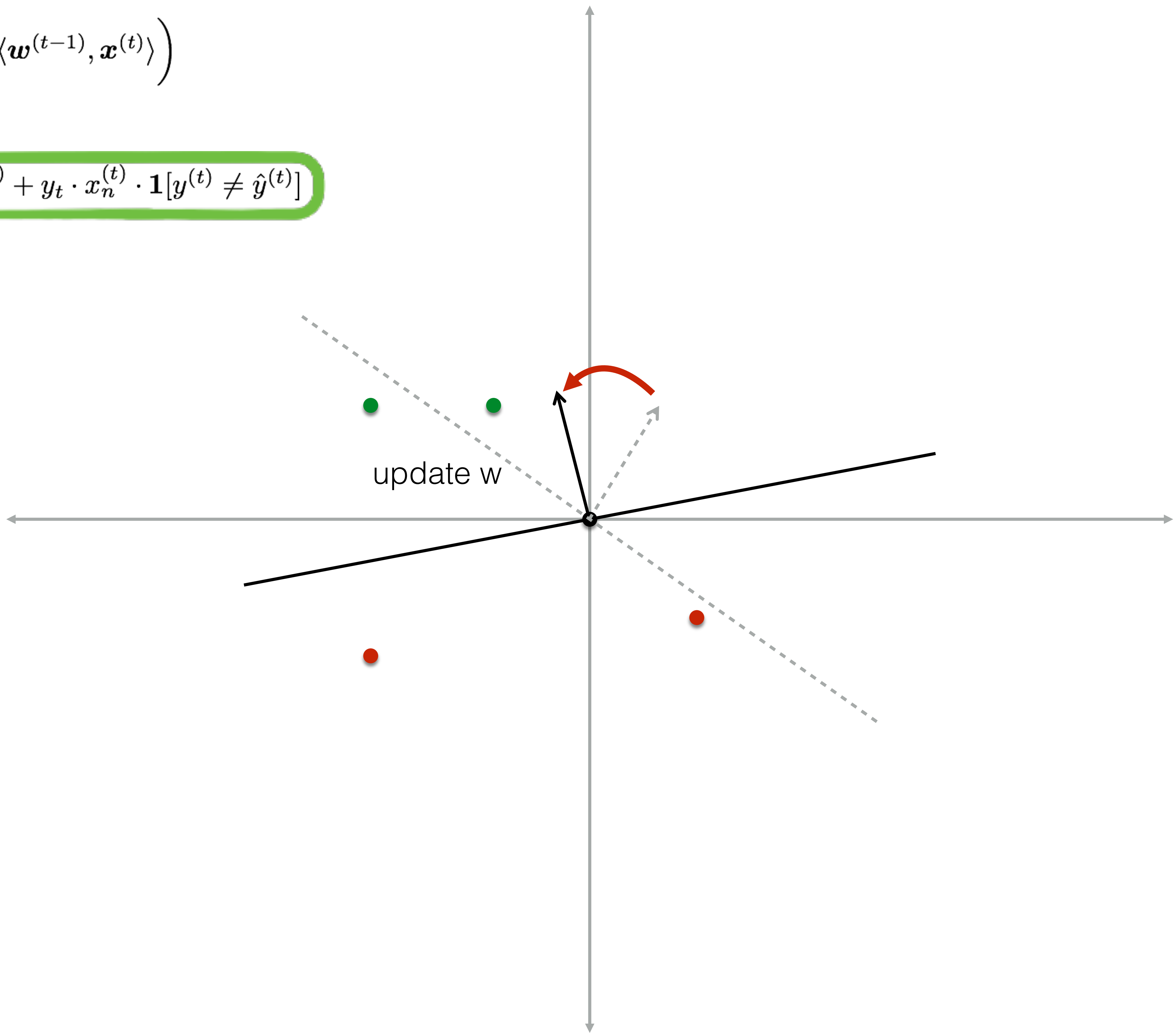


RECEIVE($\boldsymbol{x}^{(t)}$)

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

RECEIVE(y^t)

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

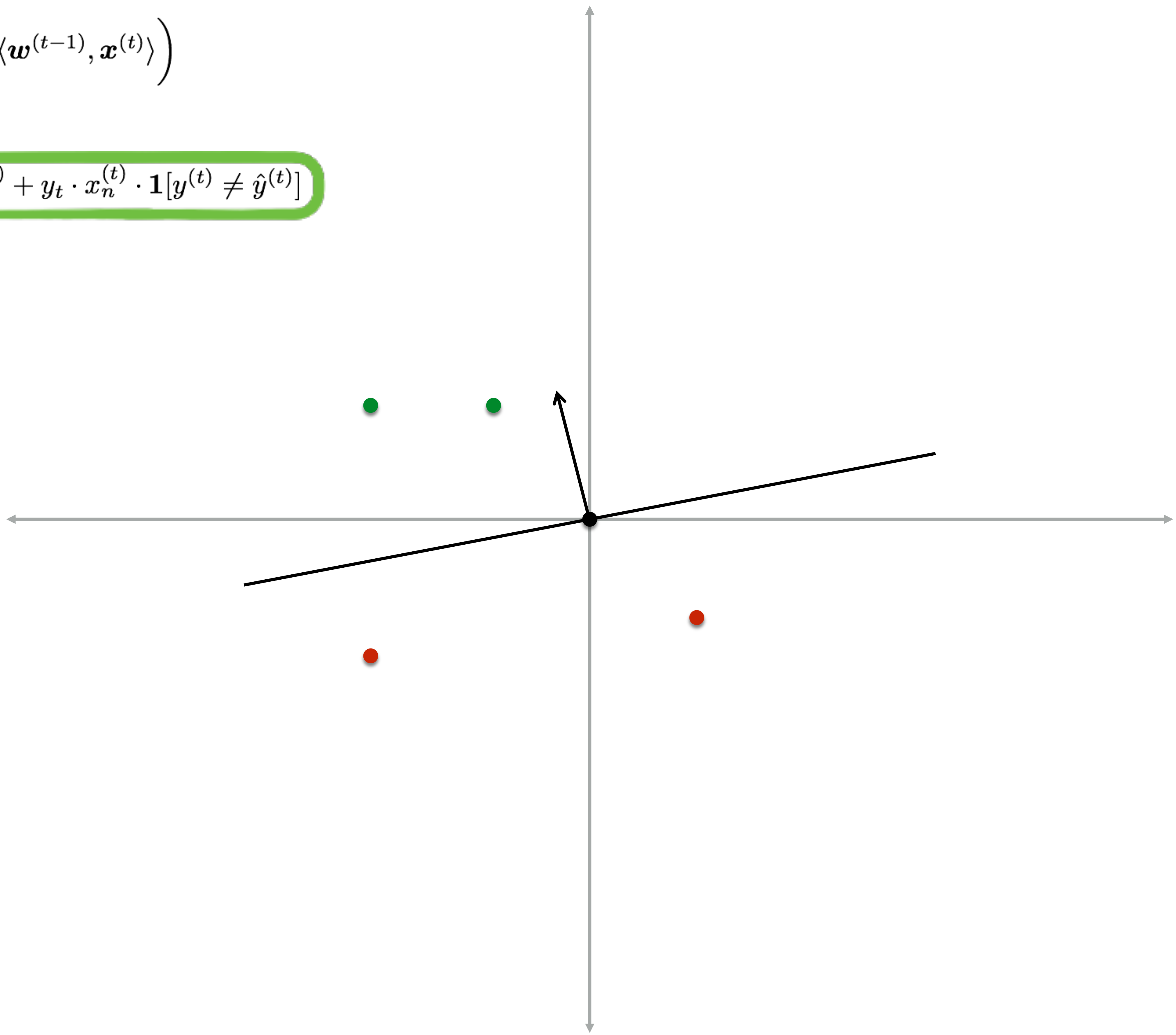


RECEIVE($\boldsymbol{x}^{(t)}$)

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

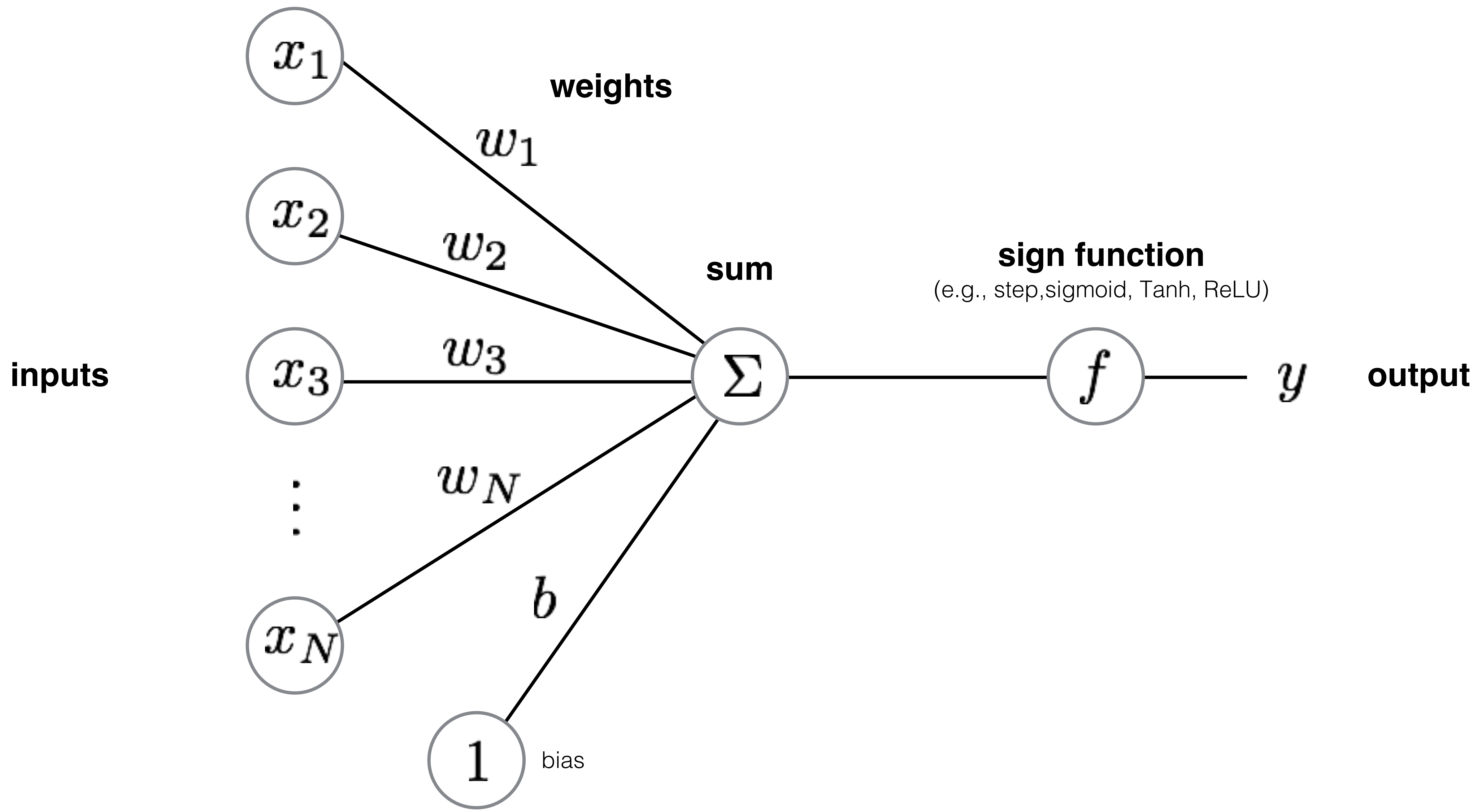
RECEIVE(y^t)

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

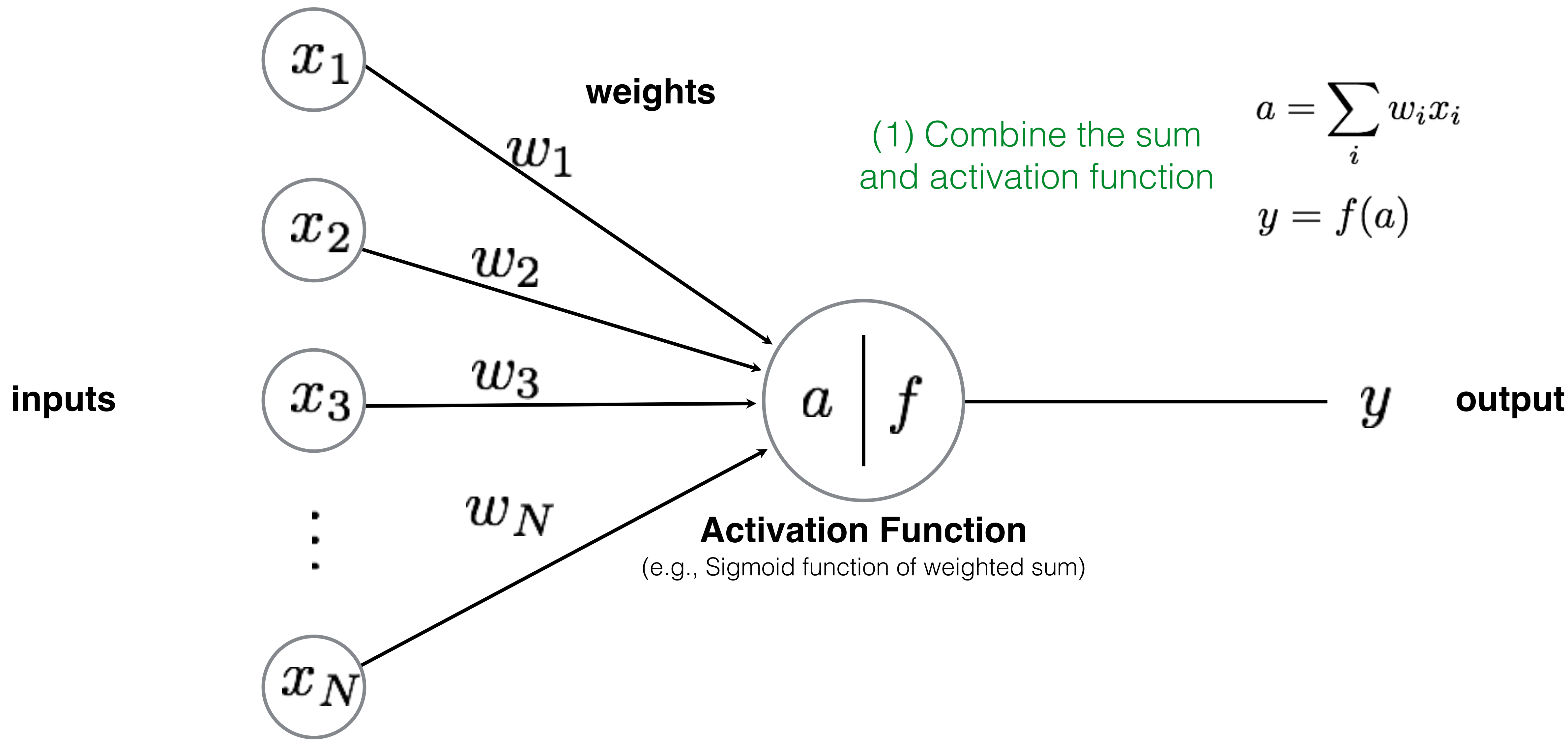


repeat …

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)

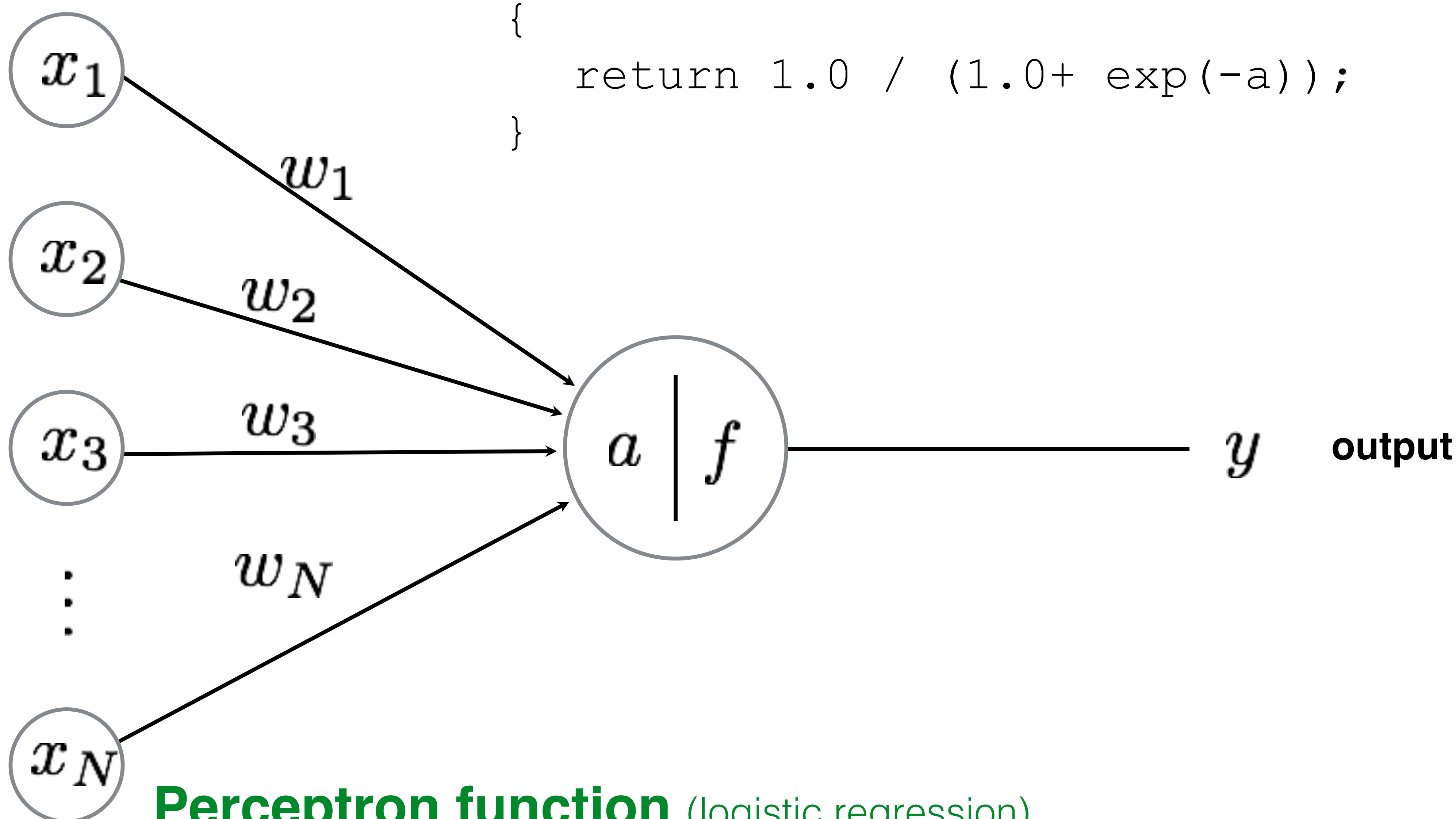
$$x_N = 1$$

$$w_N = b$$

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);
}
```

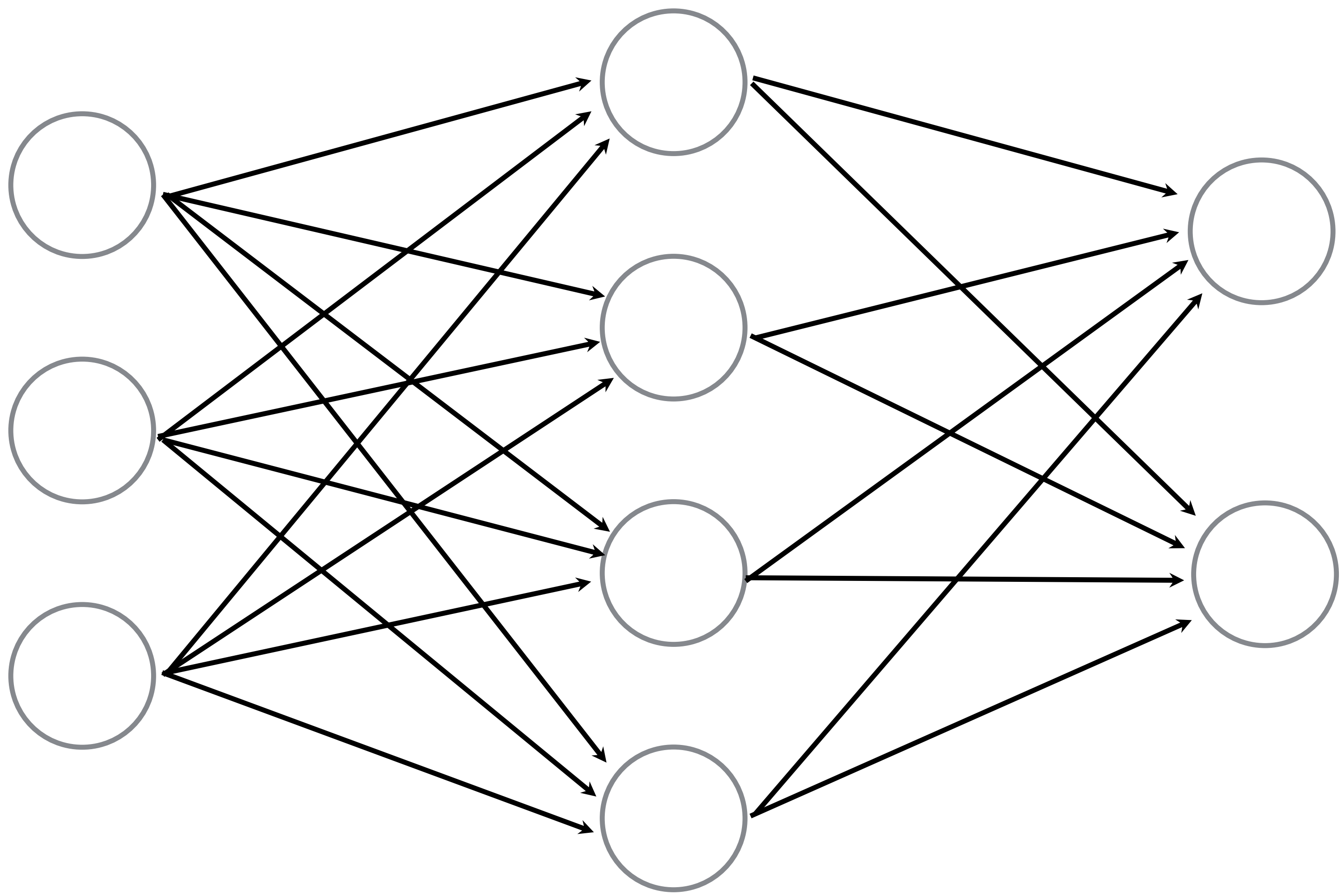
Neural networks

Connect a bunch of perceptrons together …

Connect a bunch of perceptrons together …

Neural Network

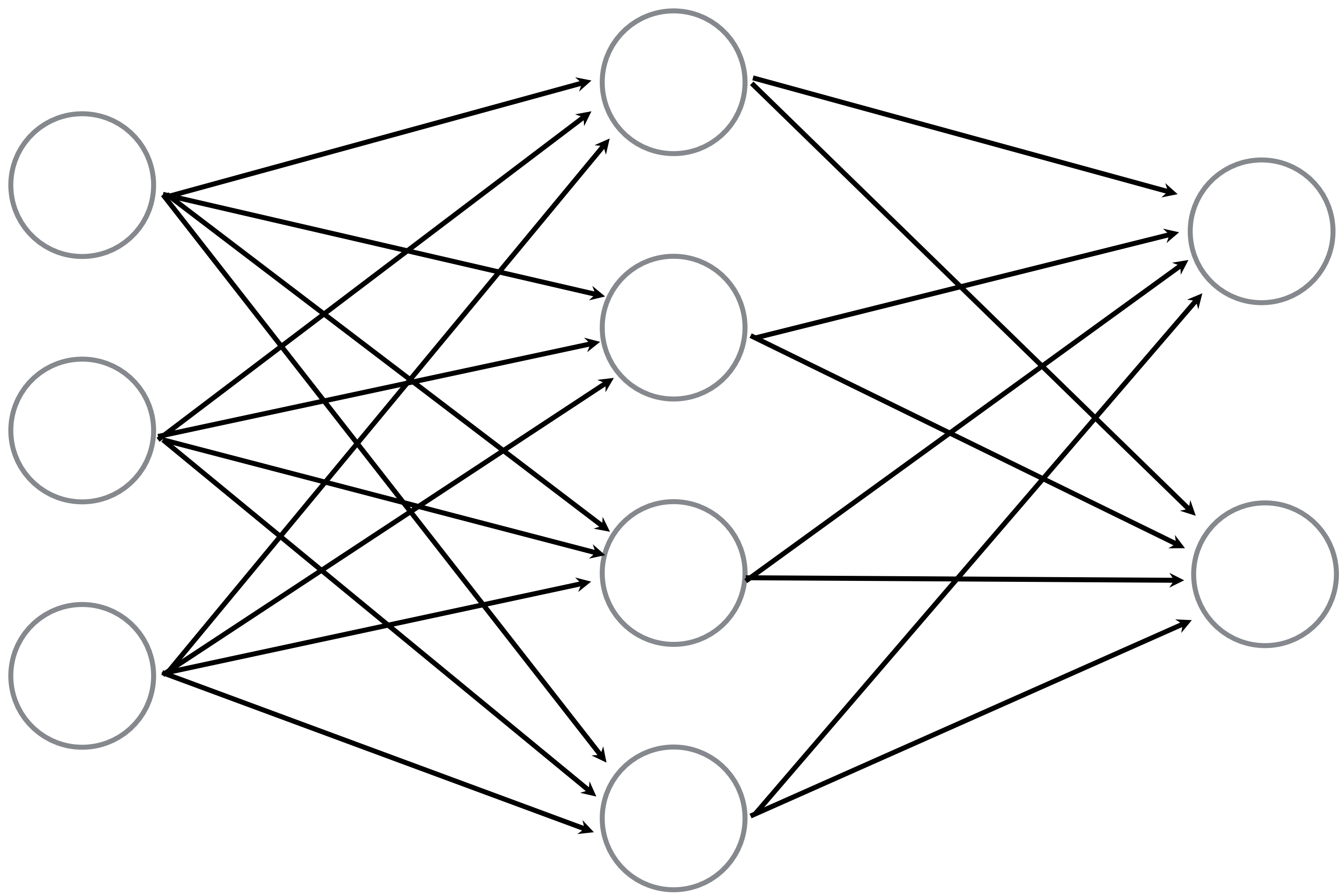
a collection of connected perceptrons



Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

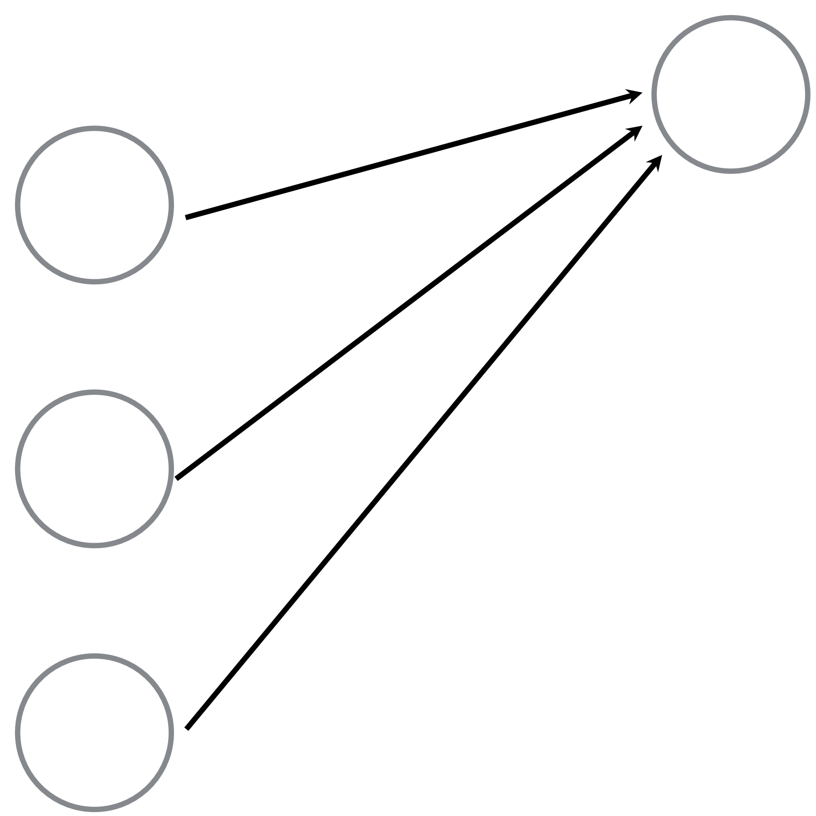


How many perceptrons in this neural network?

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

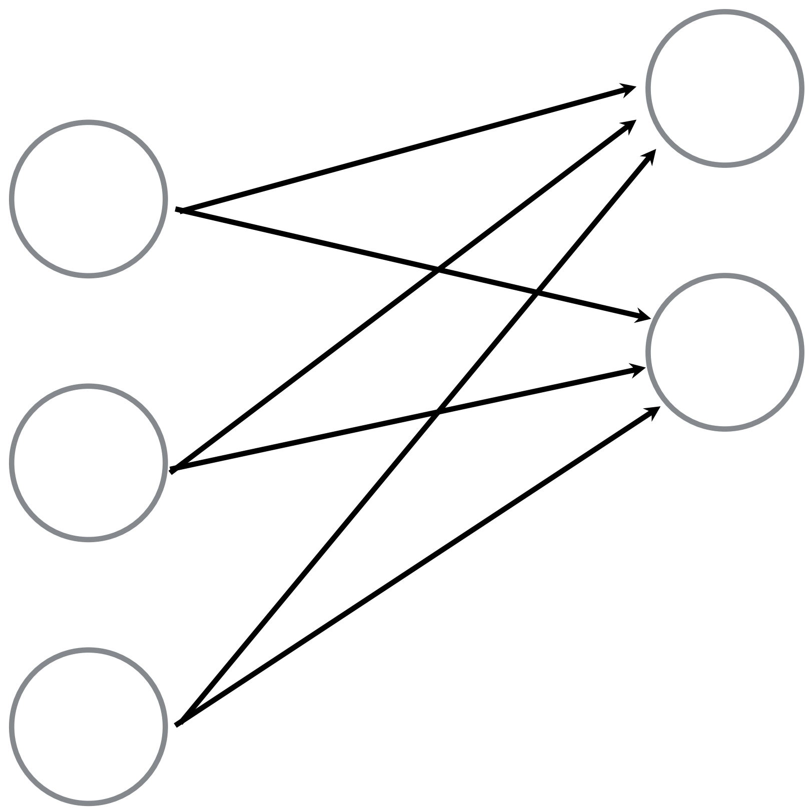


‘one perceptron’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

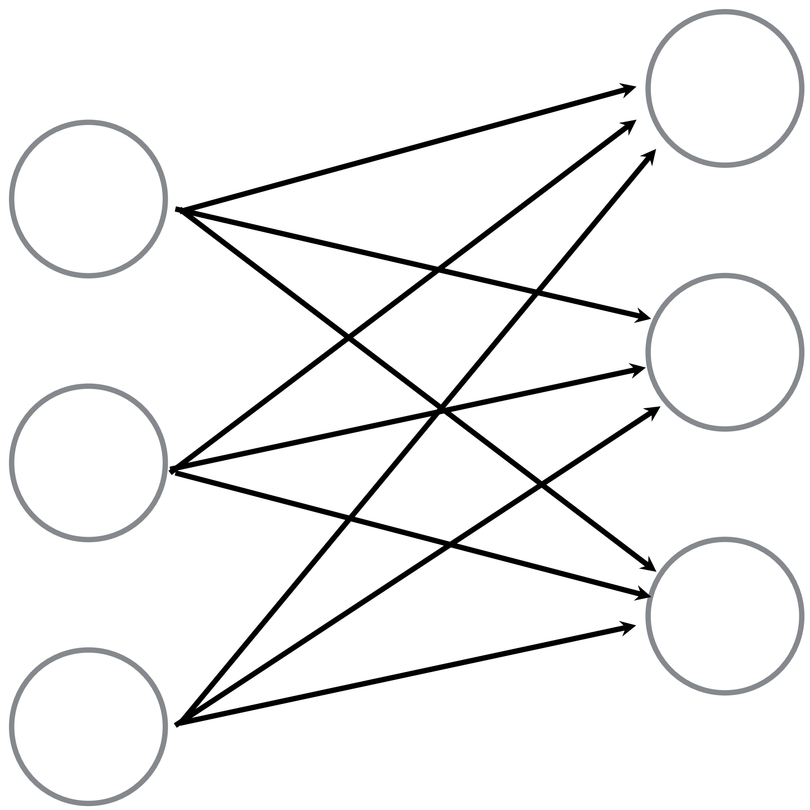


‘two perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

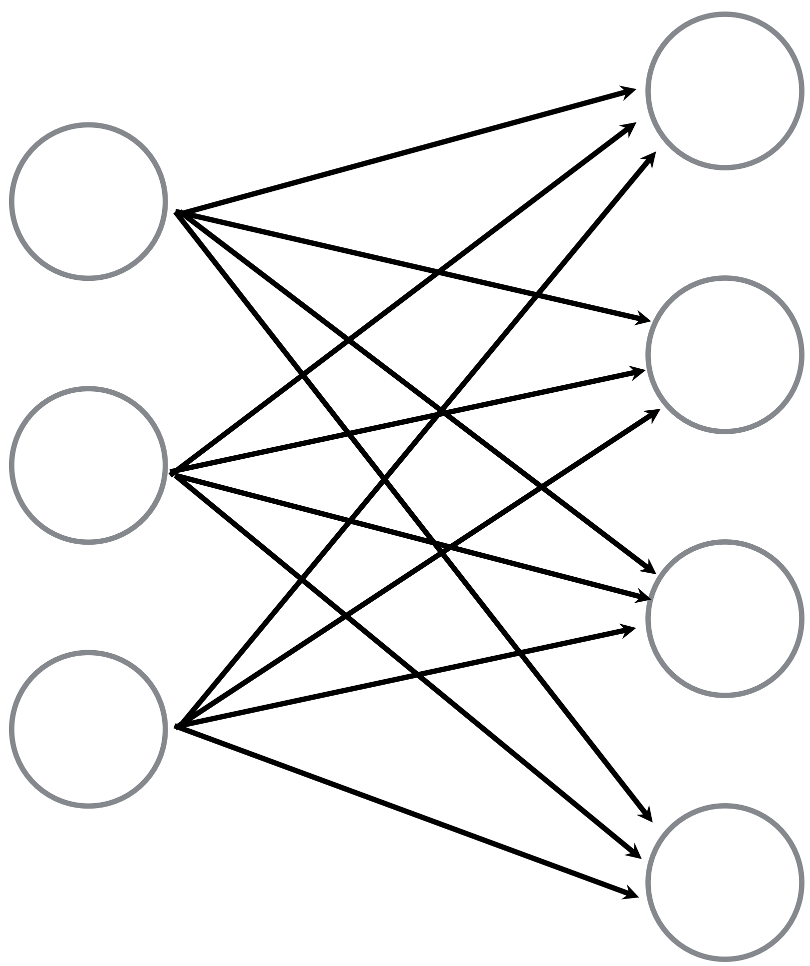


‘three perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

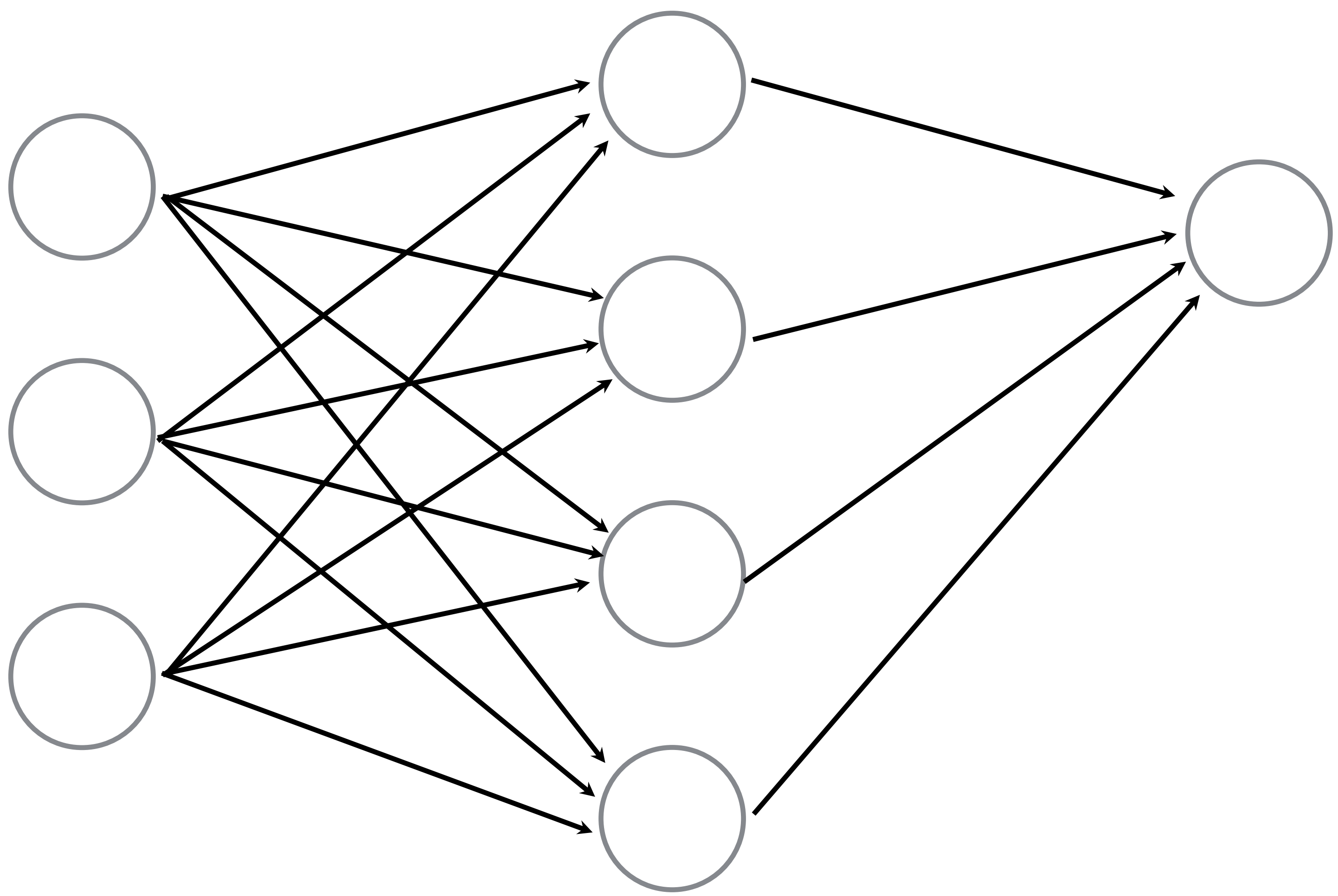


‘four perceptrons’

Connect a bunch of perceptrons together …

Neural Network

a collection of connected perceptrons

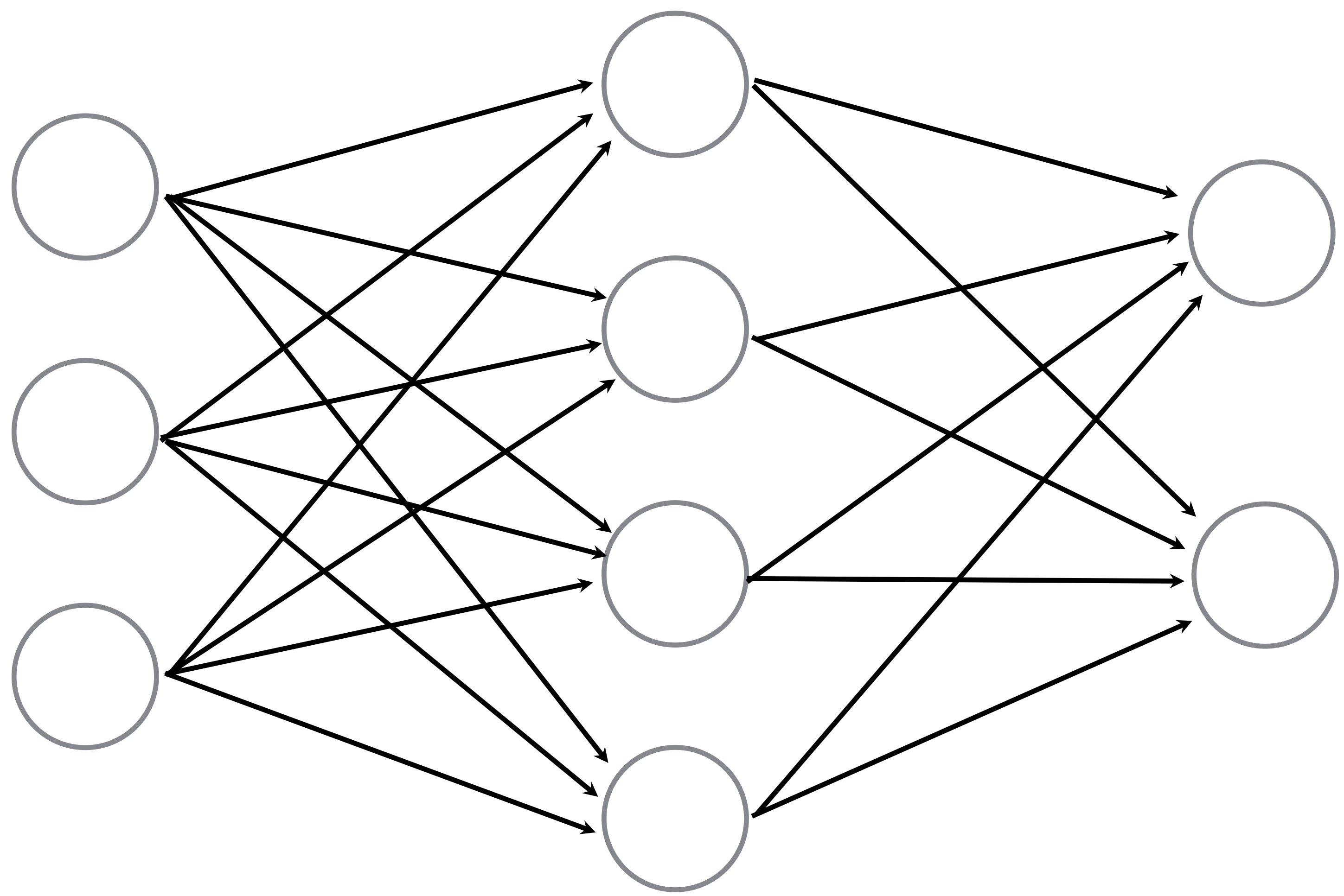


‘five perceptrons’

Connect a bunch of perceptrons together …

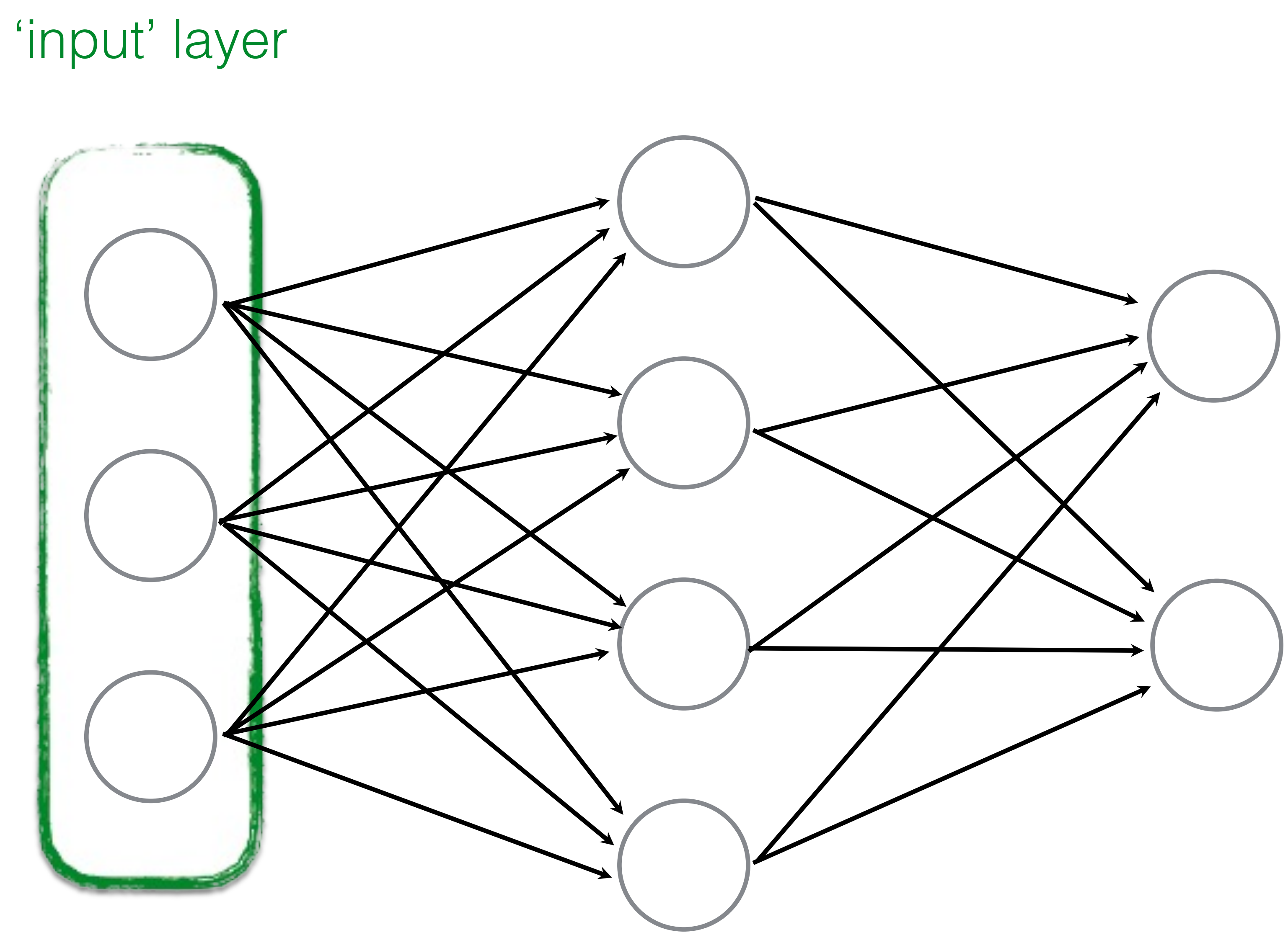
Neural Network

a collection of connected perceptrons



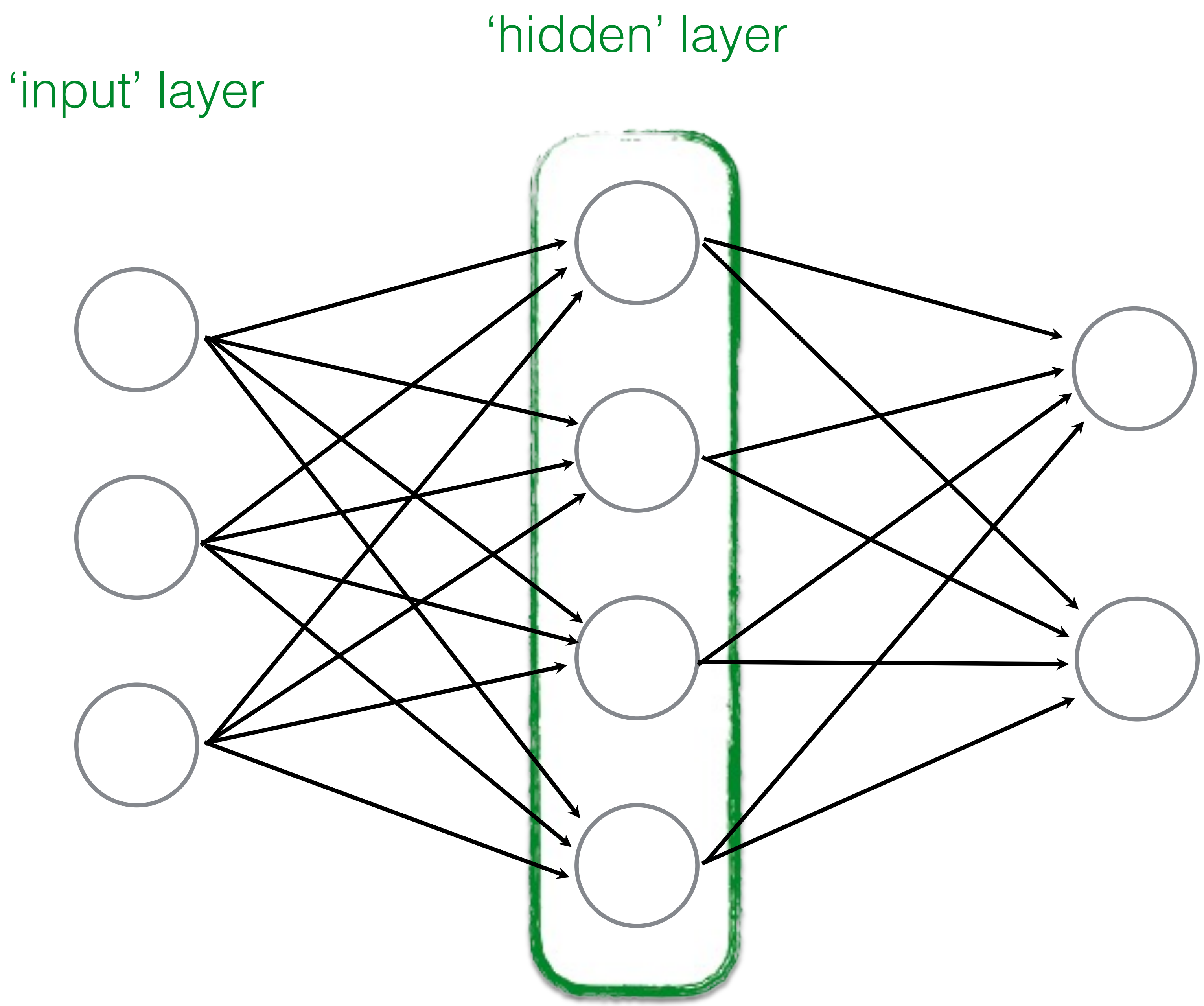
‘six perceptrons’

Some terminology…



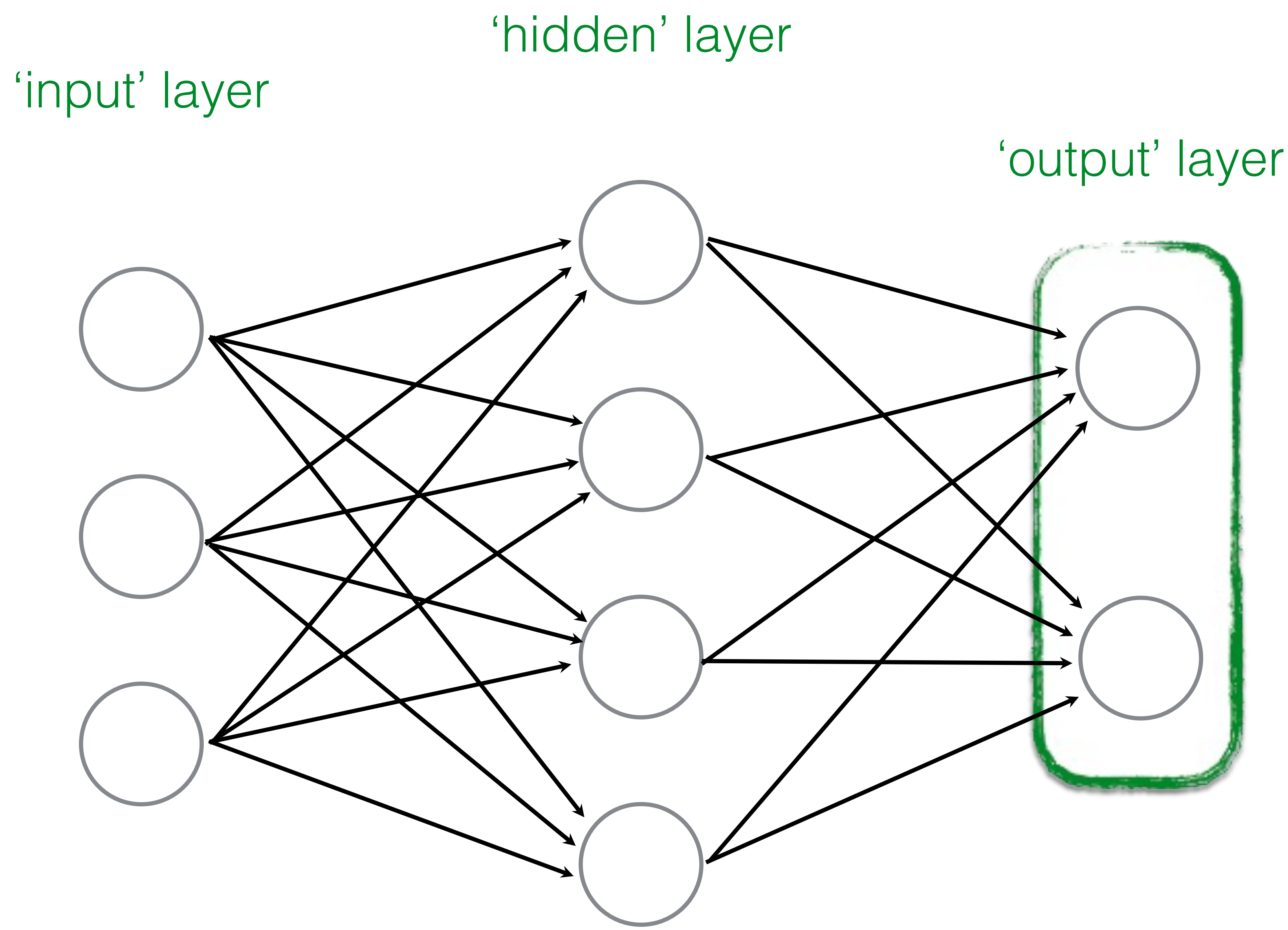
…also called a **Multi-layer Perceptron** (MLP)

Some terminology…



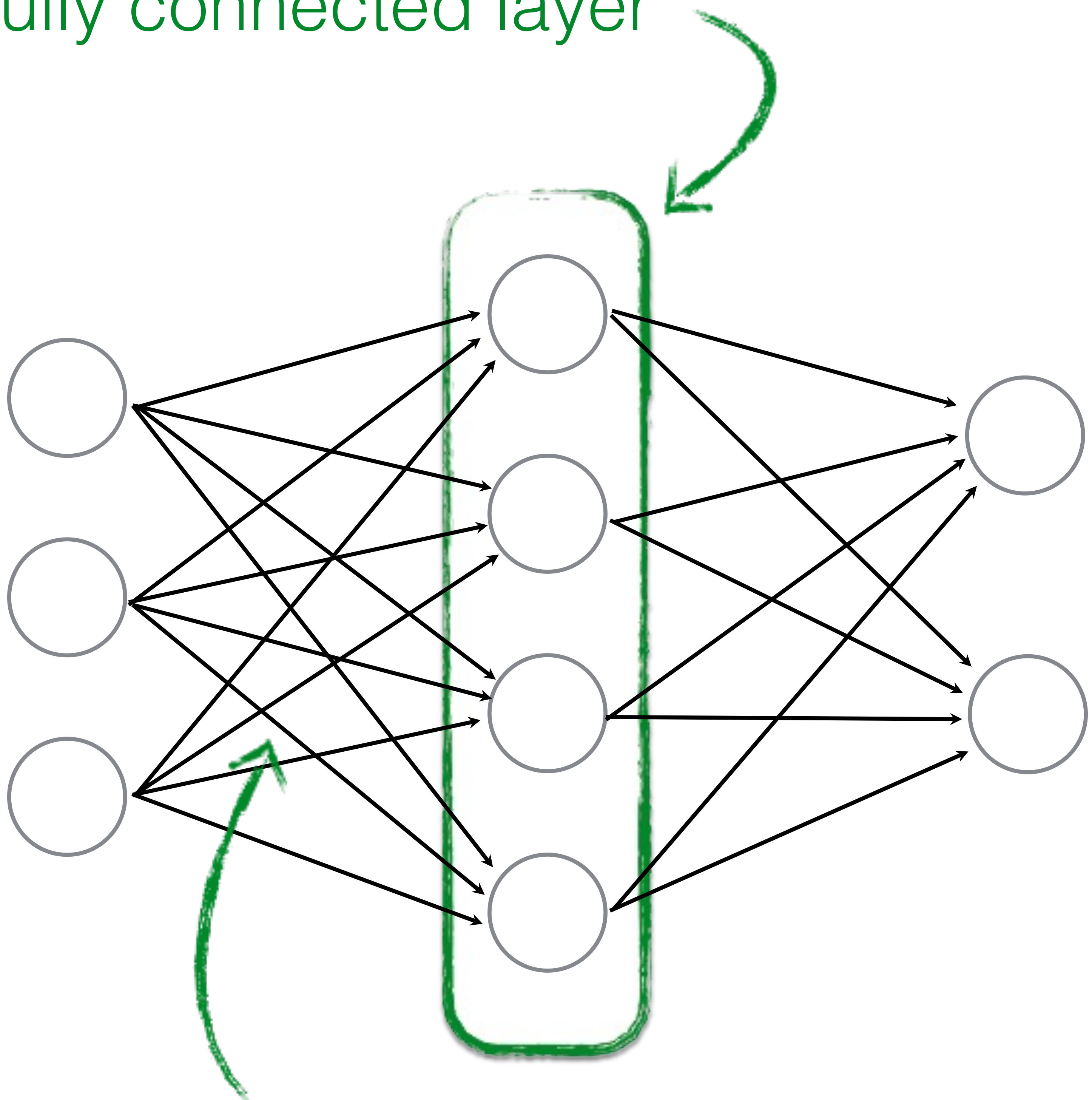
…also called a **Multi-layer Perceptron** (MLP)

Some terminology…

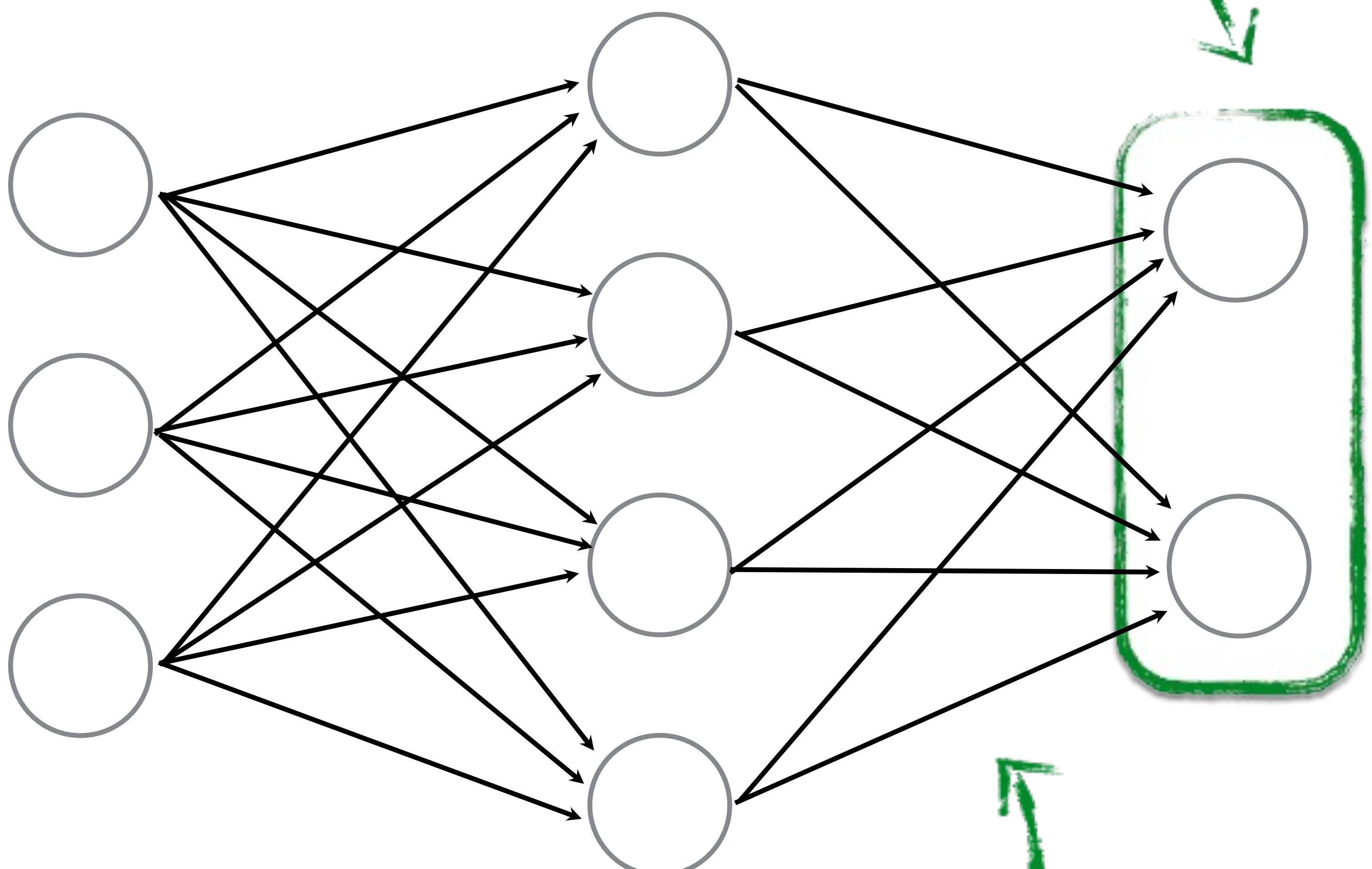


…also called a **Multi-layer Perceptron** (MLP)

this layer is a
‘fully connected layer’



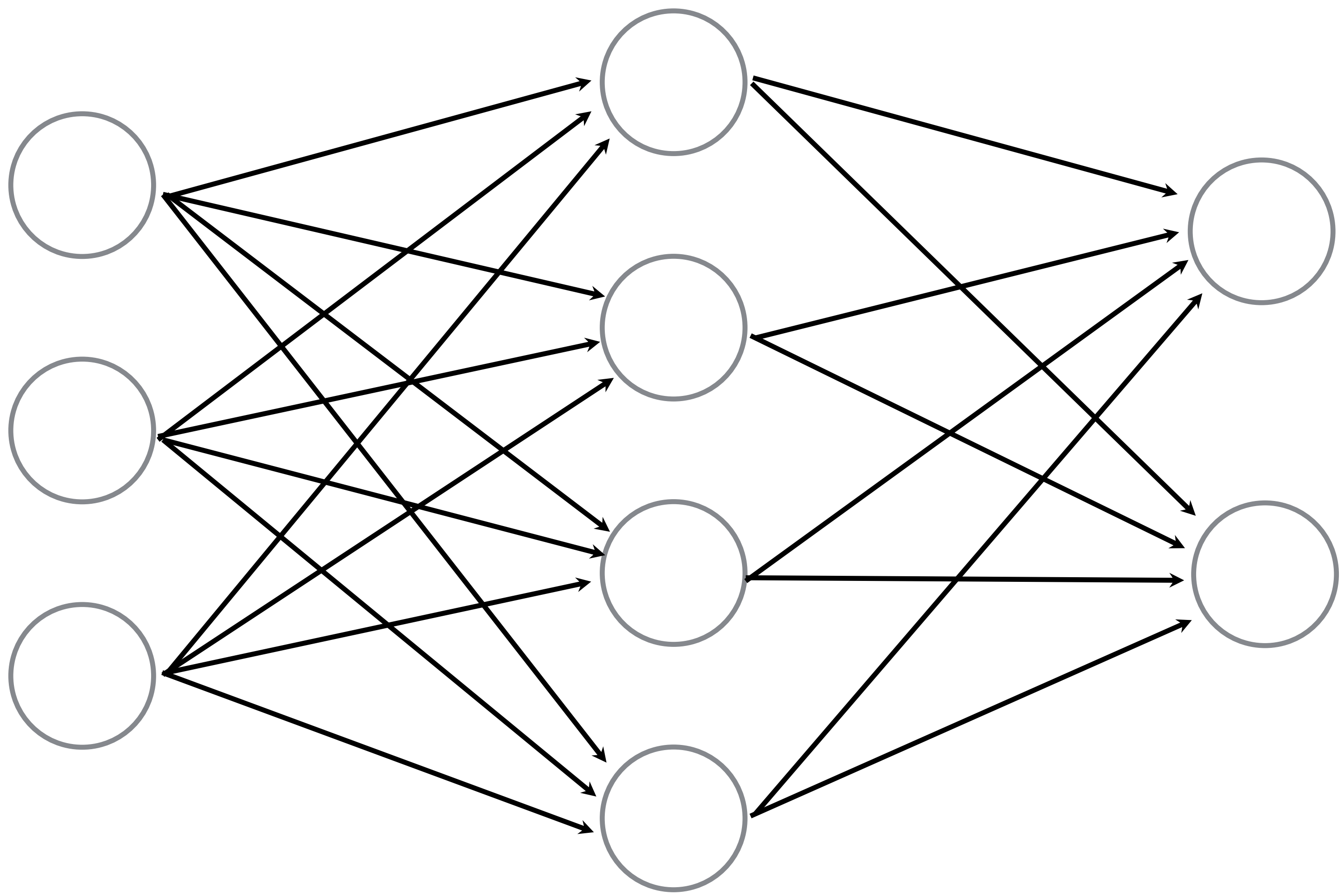
all pairwise neurons <u>between</u> layers are connected so is this



all pairwise neurons between layers are connected

How many neurons (perceptrons)?

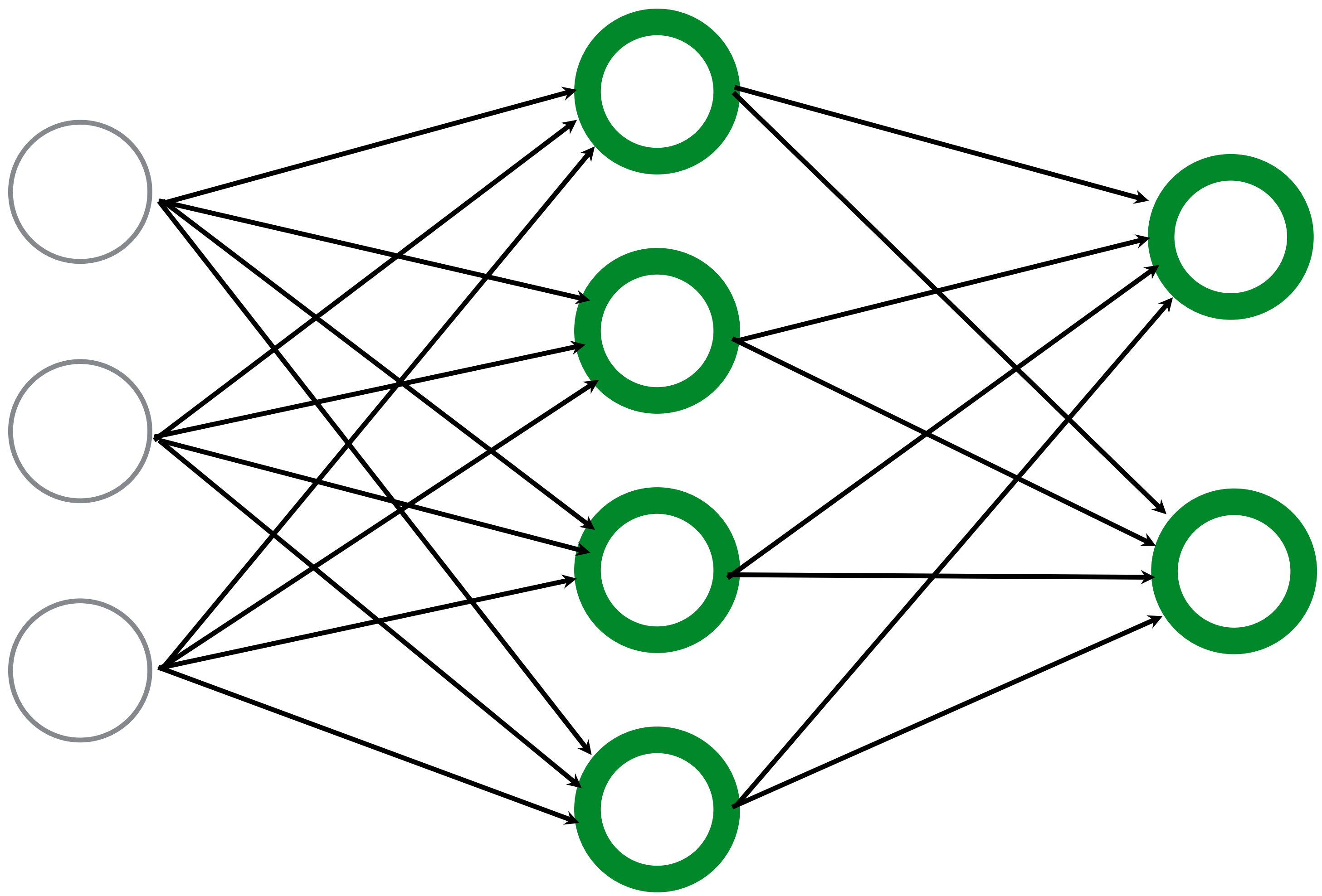
How many weights (edges)?



How many learnable parameters total?

How many neurons (perceptrons)? 4 + 2 = 6

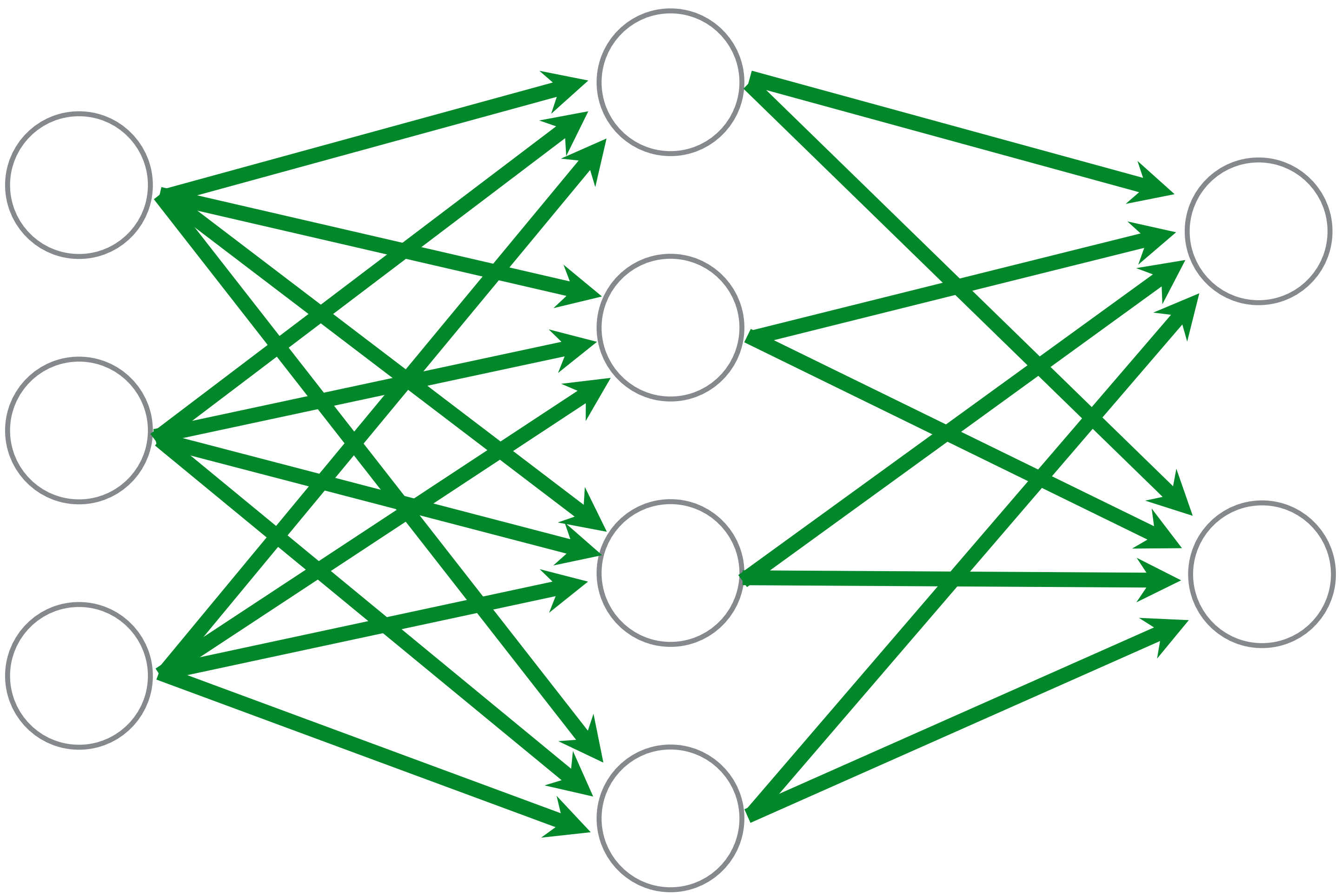
How many weights (edges)?



How many learnable parameters total?

How many neurons (perceptrons)? $4 + 2 = 6$

How many weights (edges)? $(3 \times 4) + (4 \times 2) = 20$



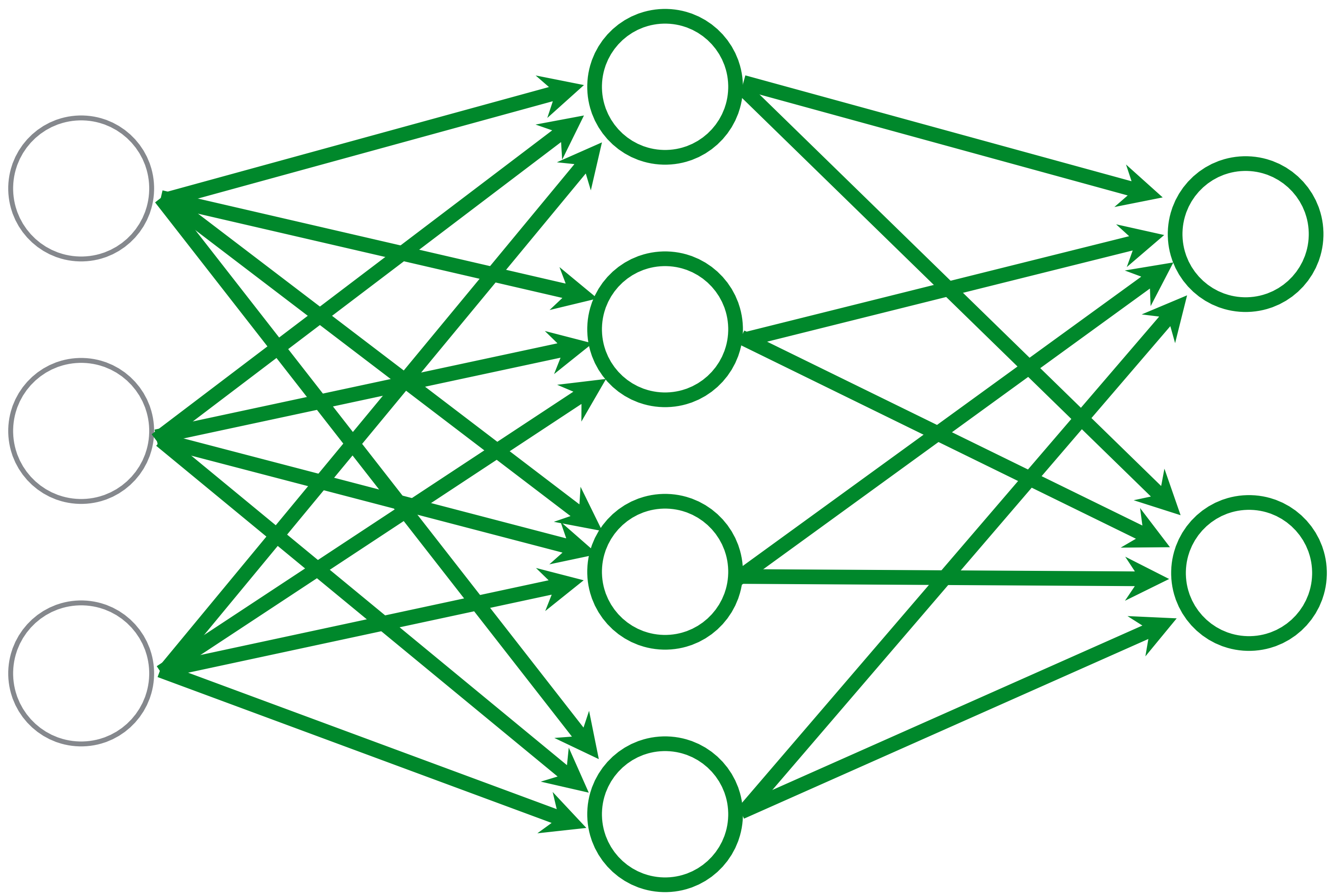
How many learnable parameters total?

How many neurons (perceptrons)?

4 + 2 = 6

How many weights (edges)?

(3 x 4) + (4 x 2) = 20



How many learnable parameters total?

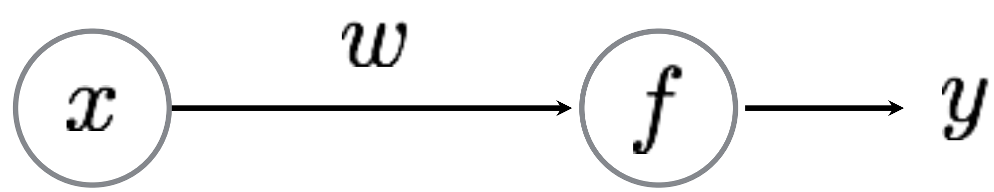
$$\underbrace{20 + 4 + 2}_{} = 26$$

bias terms

Training perceptrons

Let’s start easy

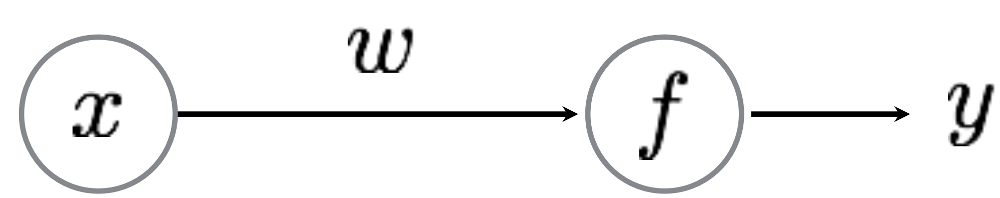
world’s smallest perceptron!



$$y = wx$$

What does this look like?

world's smallest perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\mathrm{PER}}(x; w)$$

Estimate the parameters of the Perceptron

$$w$$

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\text{PER}}(x; w)$$

what is this activation function?

Estimate the parameters of the Perceptron

$$w$$

Learning a Perceptron

Given a set of samples and a Perceptron

$$\{x_i, y_i\}$$

$$y = f_{\mathrm{PER}}(x; w)$$

what is this activation function?

linear function! $f(x) = wx$

Estimate the parameters of the Perceptron

$$w$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

Given training data:

x	y
10	10.1
2	1.9
3.5	3.4
1	1.1

What do you think the weight parameter is?

$$y = wx$$

not so obvious as the network gets more complicated so we use …

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

true label

An Incremental Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

what does this mean?

true label

Before diving into gradient descent, we need to understand …

Loss Function

defines what it means to be

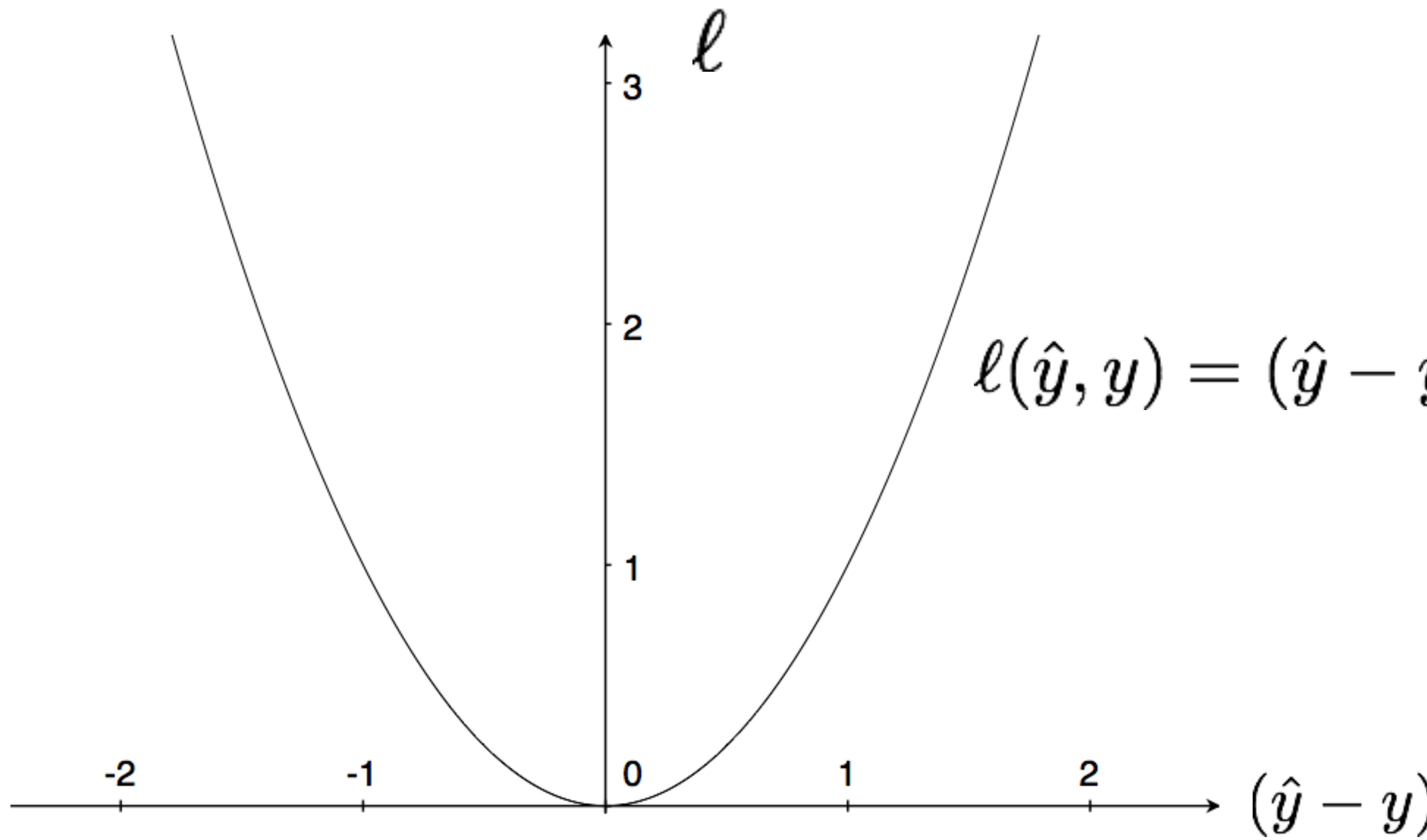
close to the true solution

YOU get to chose the loss function!

(some are better than others depending on what you want to do)

Squared Error (L2)

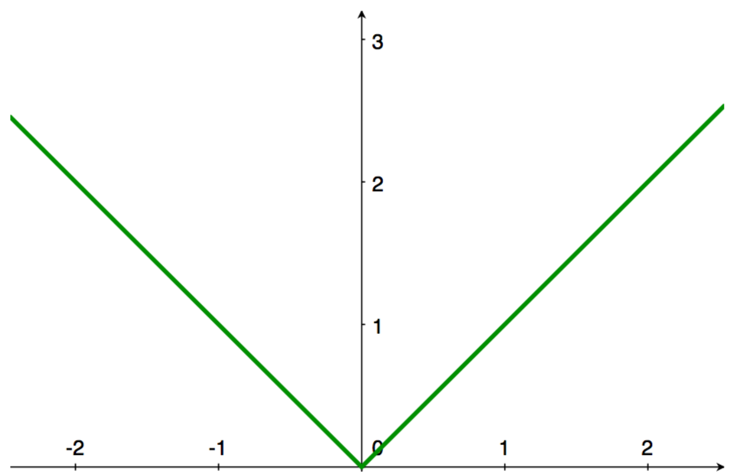
(a popular loss function) ((why?))



$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$

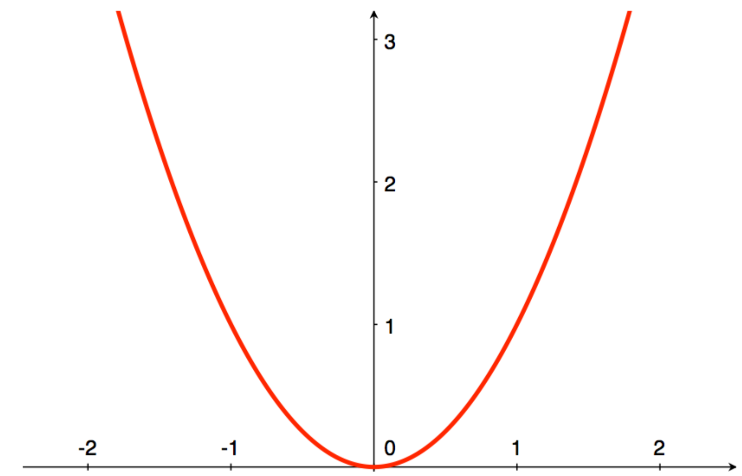
L1 Loss

$$\ell(\hat{y}, y) = |\hat{y} - y|$$



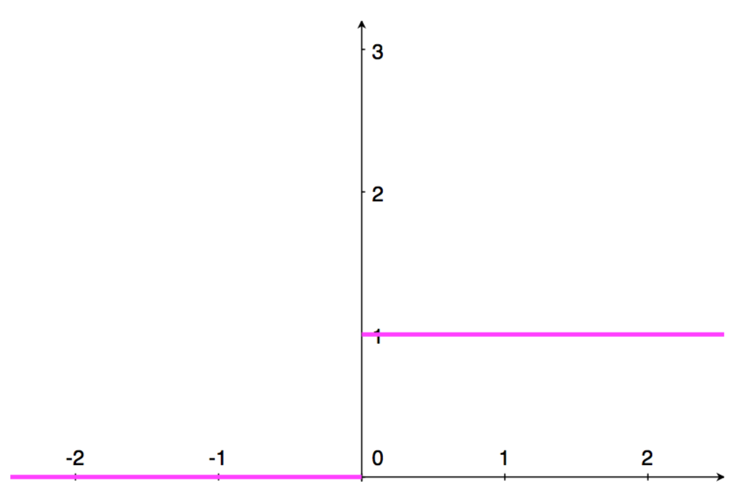
L2 Loss

$$\ell(\hat{y}, y) = (\hat{y} - y)^2$$



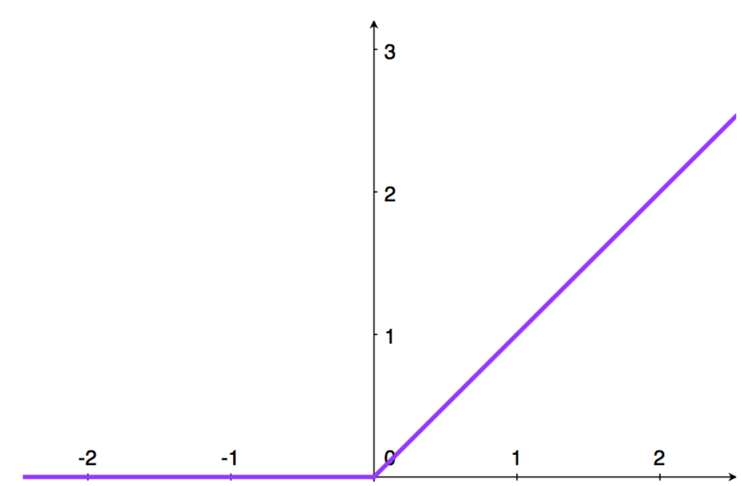
Zero-One Loss

$$\ell(\hat{y}, y) = \mathbf{1}[\hat{y} = y]$$



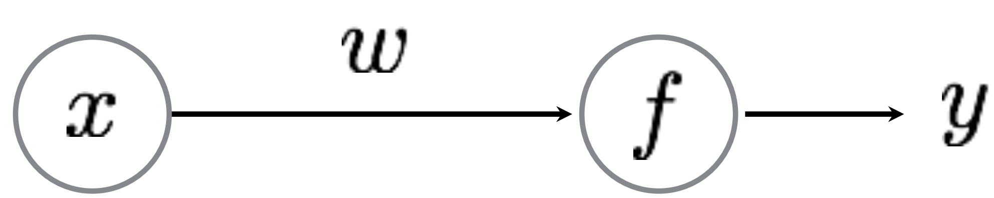
Hinge Loss

$$\ell(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$$



back to the…

World's Smallest Perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

function of **ONE** parameter!

Learning Strategy

(gradient descent)

Given several examples

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$$

and a perceptron

$$\hat{y} = wx$$

Modify weight w such that $\hat{y}$ gets **'closer'** to y

perceptron parameter

perceptron output

true label

Code to train your perceptron:

$$\textbf{for } n = 1 \ldots N$$
$$\quad w = w + (y_n - \hat{y})x_i;$$

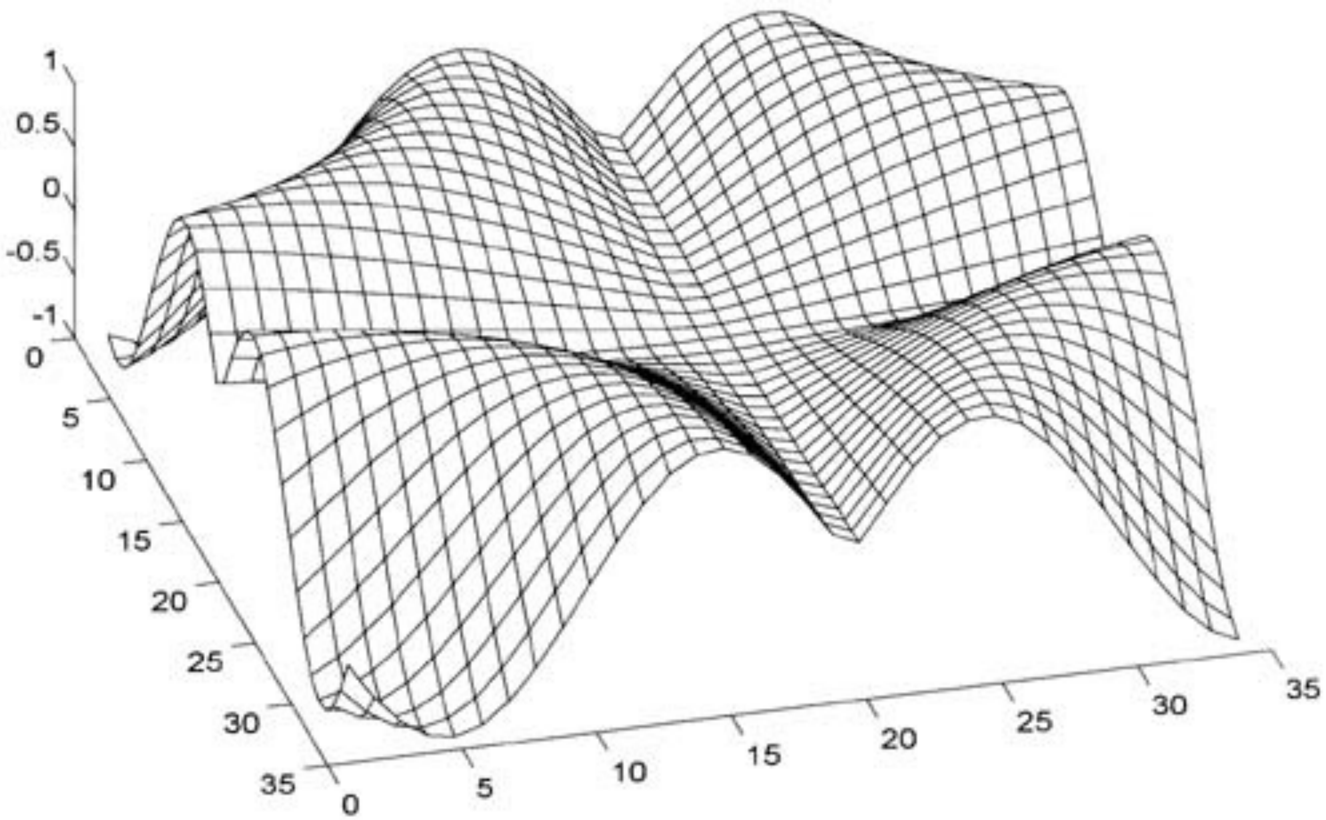
just one line of code!

Now where does this come from?

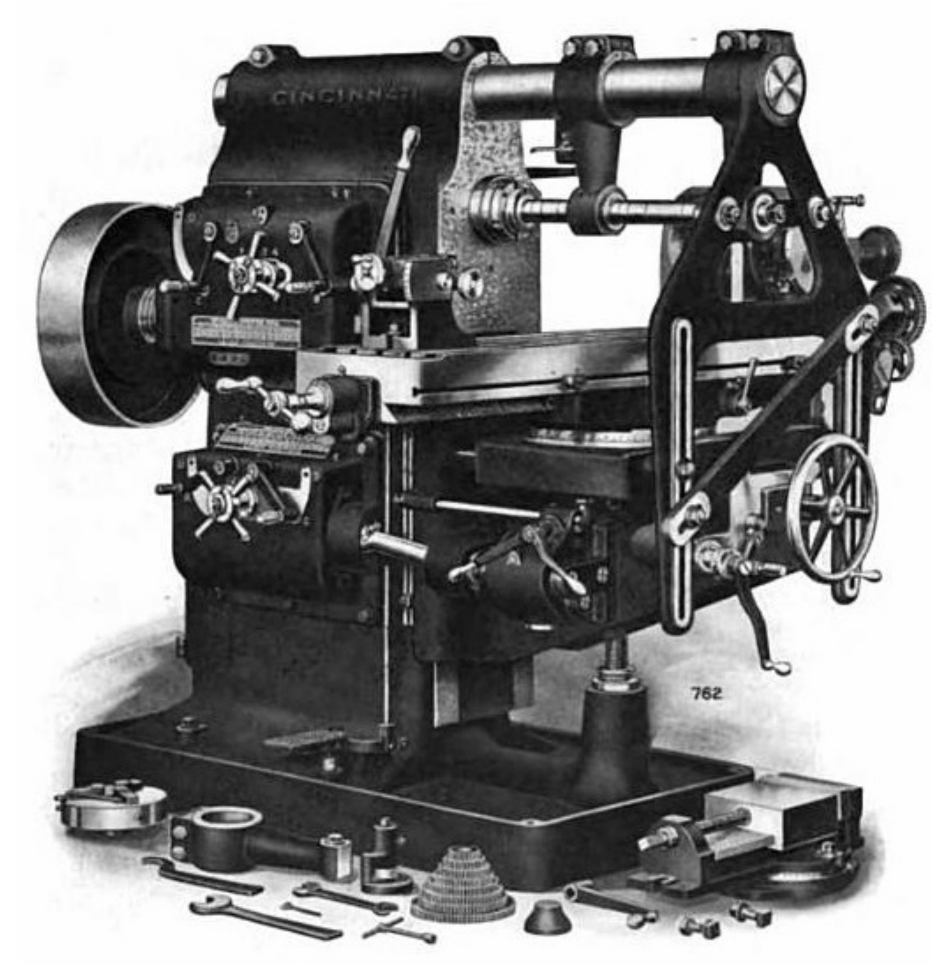
Gradient descent

(partial) derivatives tell us how much one variable affects another

Two ways to think about them:

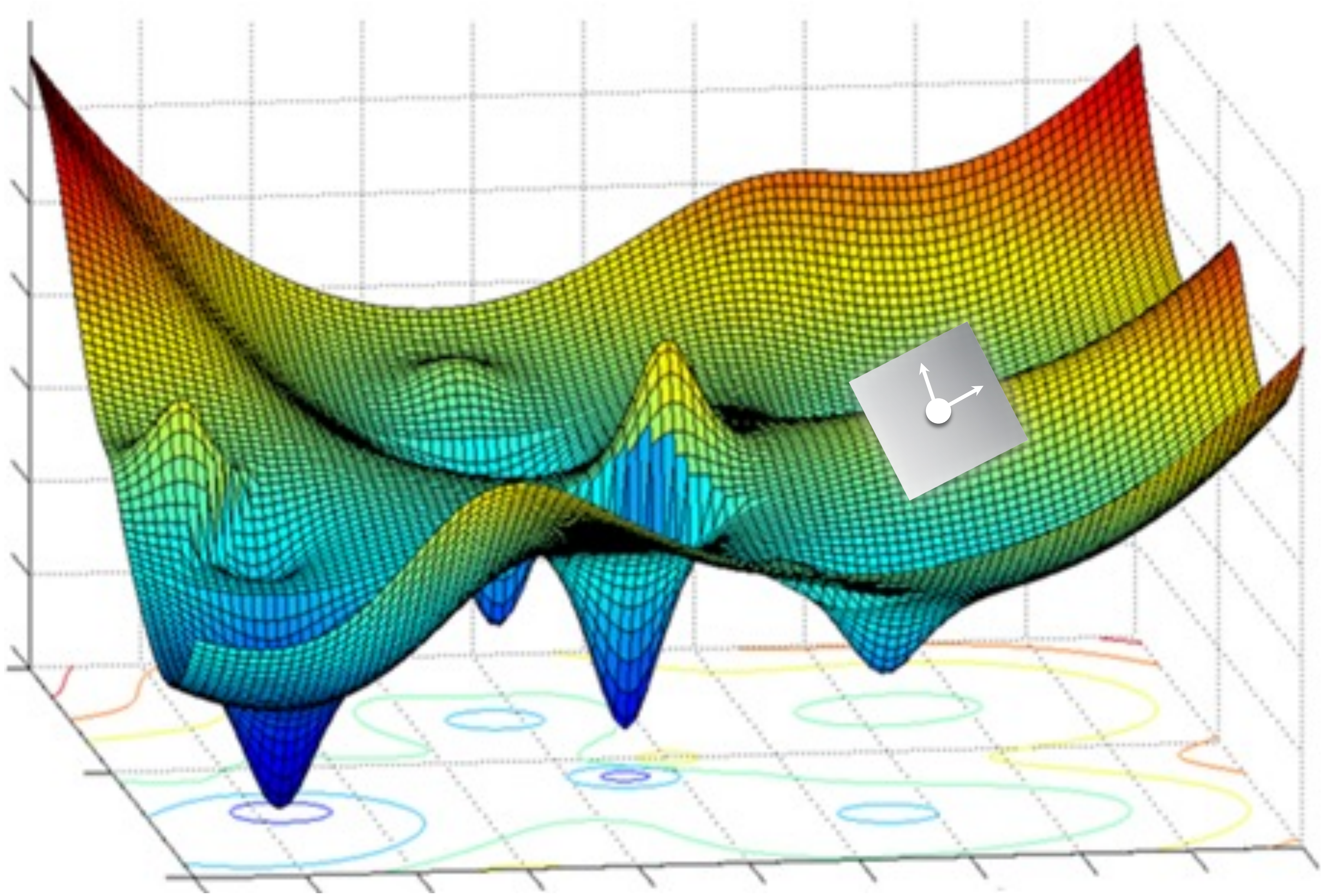


Slope of a function



Knobs on a machine

1. Slope of a function:



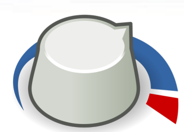
$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \left[\frac{\partial f(\boldsymbol{x})}{\partial x}, \frac{\partial f(\boldsymbol{x})}{\partial y}\right]$$

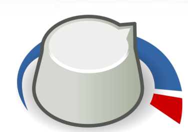
describes the slope around a point

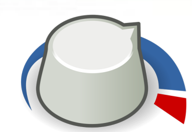
2. Knobs on a machine:

input
$\boldsymbol{x}$

output
$f(\boldsymbol{x}; \boldsymbol{w})$





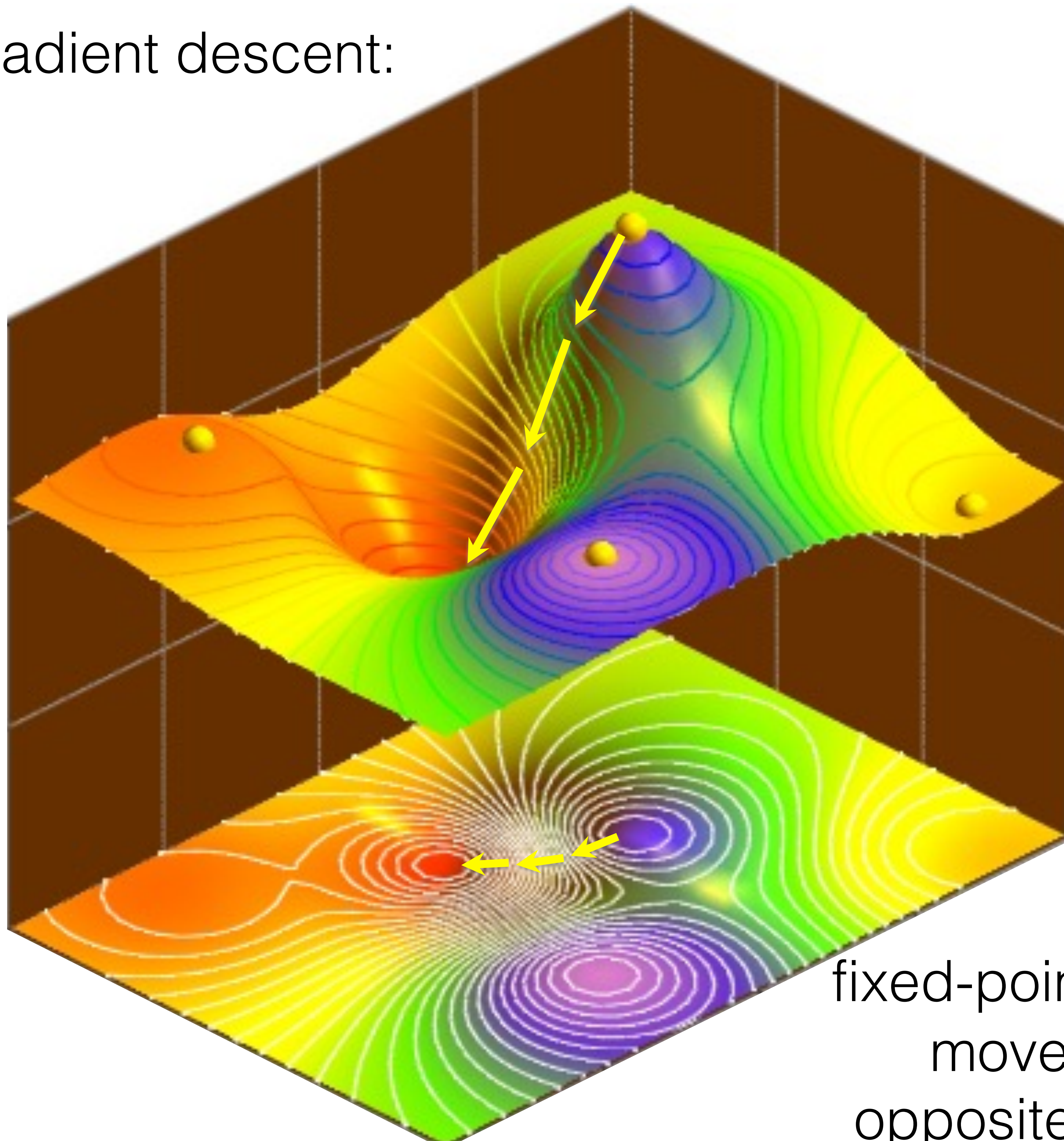


describes how each 'knob' affects the output

$$\frac{\partial f(x)}{\partial w_1} \qquad \frac{\partial f(x)}{\partial w_2} \qquad \frac{\partial f(x)}{\partial w_3}$$

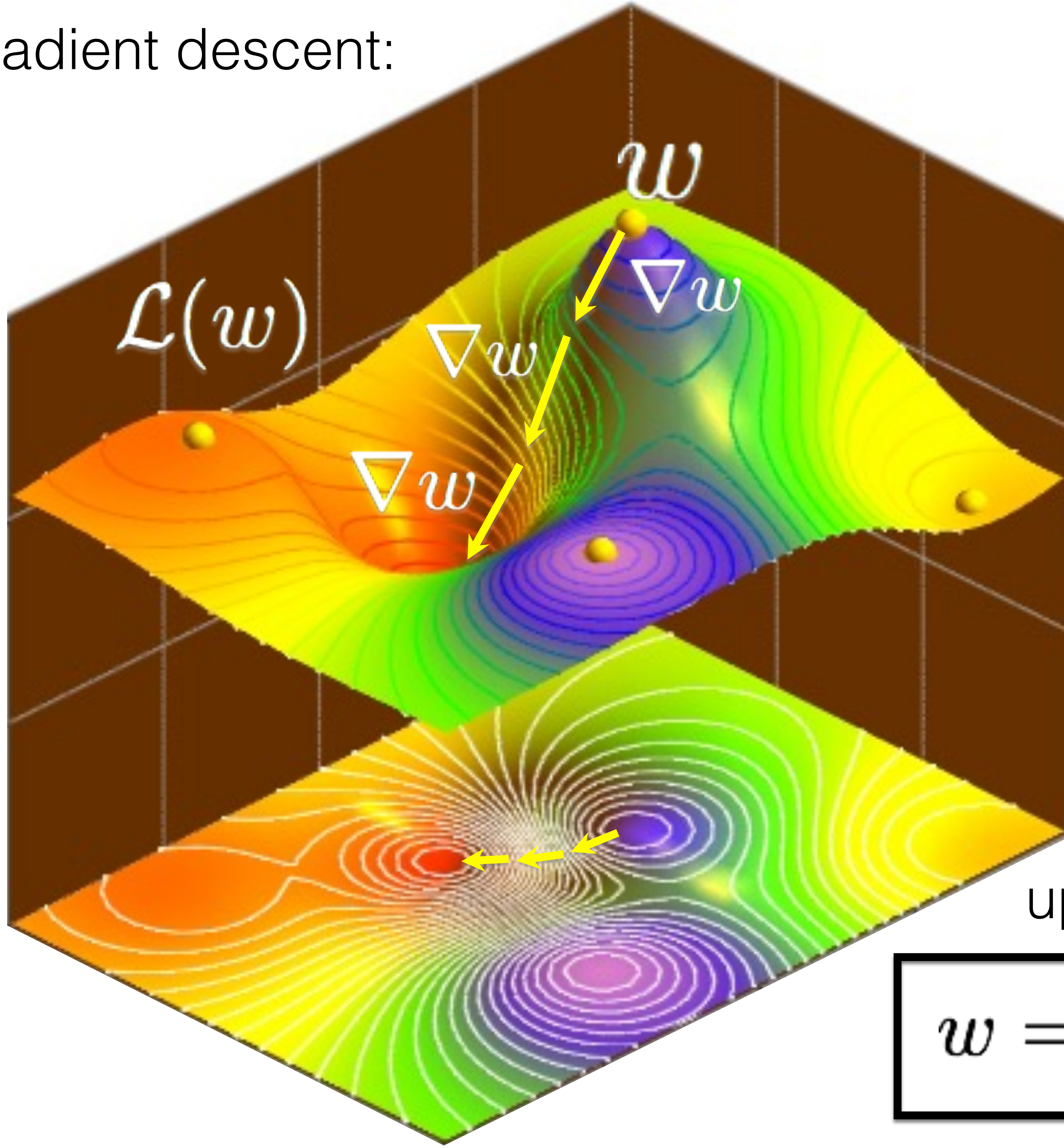
small change in parameter Δw_1 → output will change by $\dfrac{\partial f(x)}{\partial w_1} \Delta w_1$

Gradient descent:



Given a
fixed-point on a function,
move in the direction
opposite of the gradient

Gradient descent:



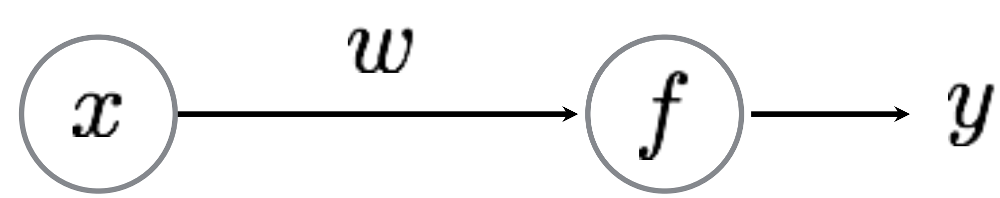
update rule:

$$\boxed{w = w - \nabla w}$$

Backpropagation

back to the…

World's Smallest Perceptron!



$$y = wx$$

(a.k.a. line equation, linear regression)

function of **ONE** parameter!

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$

$$w = w + \underline{(y_n - \hat{y})x_i};$$

This is just gradient descent, that means…

this should be the gradient of the loss function

$$\frac{d\mathcal{L}}{dw}$$ …is the rate at which **this** will change…

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

the loss function

… per unit change of **this**

$$y = wx$$

the weight parameter

Let's compute the derivative…

Compute the derivative

$$\frac{d\mathcal{L}}{dw} = \frac{d}{dw}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{dwx}{dw}$$
$$= -(y-\hat{y})x = \nabla w$$ just shorthand

That means the weight update for **gradient descent** is:

$$w = w - \nabla w$$ move in direction of negative gradient
$$= w + (y-\hat{y})x$$

Gradient Descent (world's smallest perceptron)

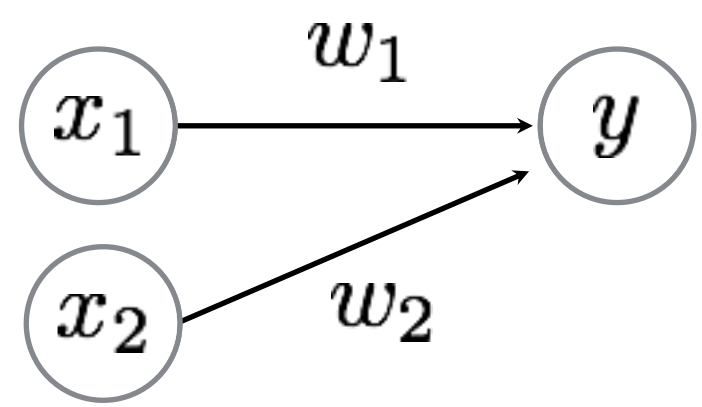
For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = wx_i$
   b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})^2$
2. Update
   a. Back Propagation $\frac{d\mathcal{L}_i}{dw} = -(y_i - \hat{y})x_i = \nabla w$
   b. Gradient update $w = w - \nabla w$

Training the world's smallest perceptron

$$\text{for } n = 1 \ldots N$$
$$w = w + (y_n - \hat{y})x_i;$$

world's (second) smallest **perceptron**!



function of **two** parameters!

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass
   b. Compute Loss
2. Update
   a. Back Propagation
   b. Gradient update

we just need to compute partial derivatives for this network

Derivative computation

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1}$$
$$\boxed{= -(y-\hat{y})x_1 = \nabla w_1}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\}$$
$$= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2}$$
$$= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1}$$
$$= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2}$$
$$\boxed{= -(y-\hat{y})x_2 = \nabla w_2}$$

Why do we have partial derivatives now?

Derivative computation

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_1} &= \frac{\partial}{\partial w_1}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\ &= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial w_1 x_1}{\partial w_1} \\ &= -(y-\hat{y})x_1 = \nabla w_1\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_2} &= \frac{\partial}{\partial w_2}\left\{\frac{1}{2}(y-\hat{y})^2\right\} \\ &= -(y-\hat{y})\frac{\partial \hat{y}}{\partial w_2} \\ &= -(y-\hat{y})\frac{\partial \sum_i w_i x_i}{\partial w_1} \\ &= -(y-\hat{y})\frac{\partial w_2 x_2}{\partial w_2} \\ &= -(y-\hat{y})x_2 = \nabla w_2\end{aligned}$$

Gradient Update

$$\begin{aligned}w_1 &= w_1 - \eta \nabla w_1 \\ &= w_1 + \eta (y-\hat{y})x_1\end{aligned}$$

$$\begin{aligned}w_2 &= w_2 - \eta \nabla w_2 \\ &= w_2 + \eta (y-\hat{y})x_2\end{aligned}$$

Gradient Descent

For each sample $\{x_i, y_i\}$

1. Predict

    a. Forward pass $\hat{y} = f_{\mathrm{MLP}}(x_i; \theta)$

    b. Compute Loss $\mathcal{L}_i = \frac{1}{2}(y_i - \hat{y})$

    (side computation to track loss. not needed for backprop)

2. Update

    two lines now

    a. Back Propagation

    $$\nabla w_{1i} = -(y_i - \hat{y})x_{1i}$$
    $$\nabla w_{2i} = -(y_i - \hat{y})x_{2i}$$

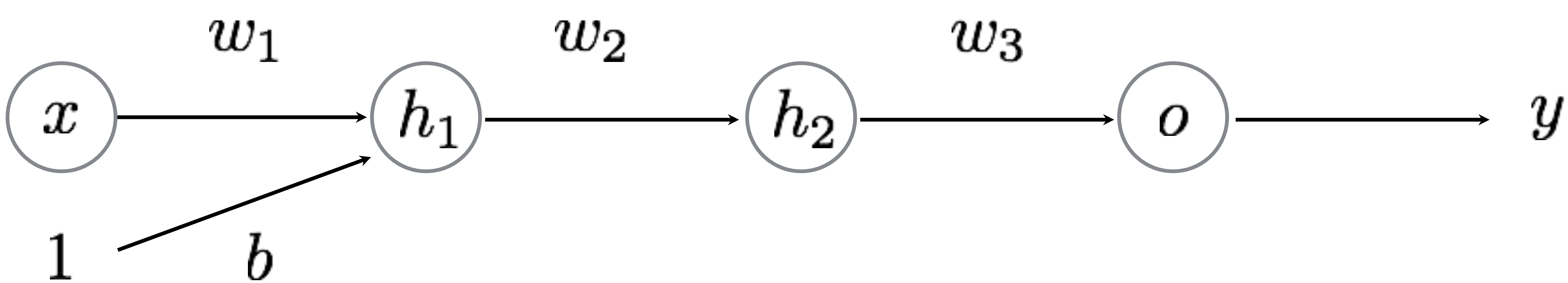
    b. Gradient update

    $$w_{1i} = w_{1i} + \eta(y - \hat{y})x_{1i}$$
    $$w_{2i} = w_{2i} + \eta(y - \hat{y})x_{2i}$$

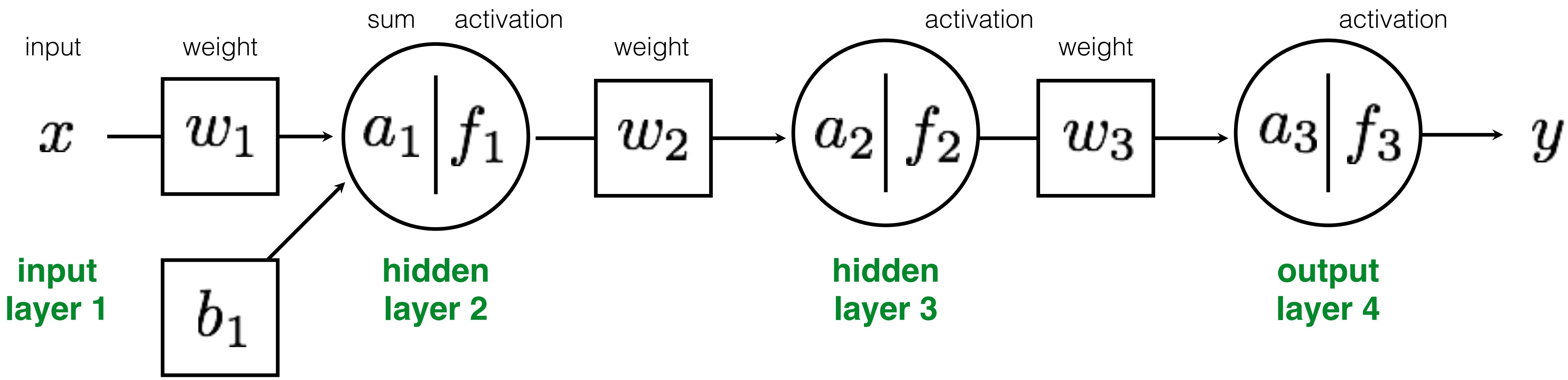
    (adjustable step size)

We haven’t seen a lot of ‘propagation’ yet
because our perceptrons only had <u>one</u> layer…

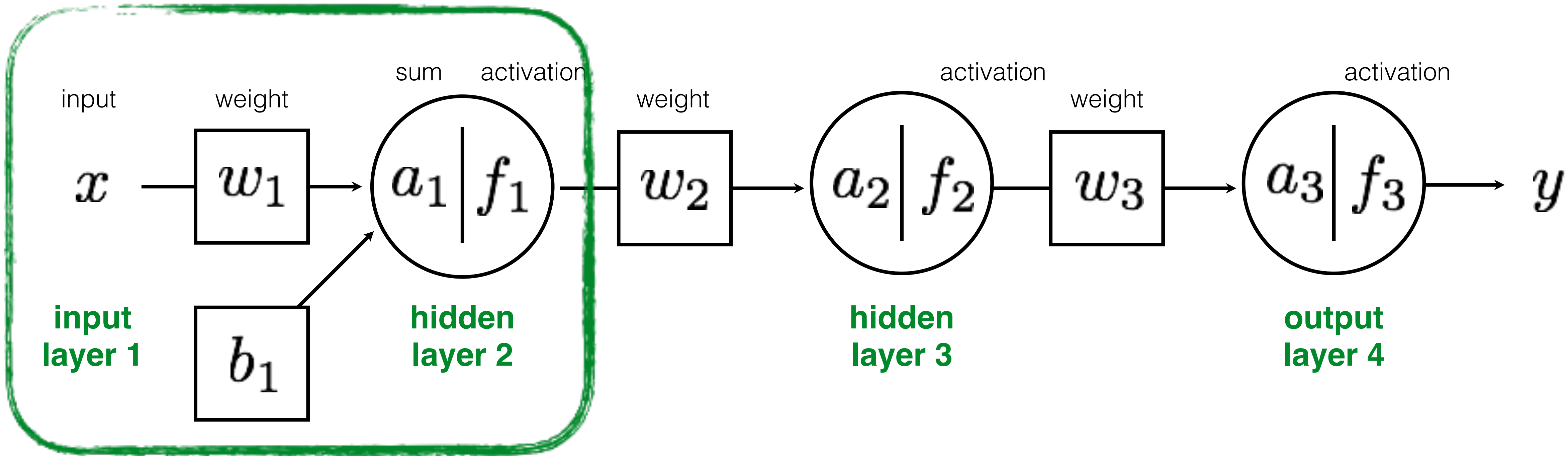
multi-layer perceptron



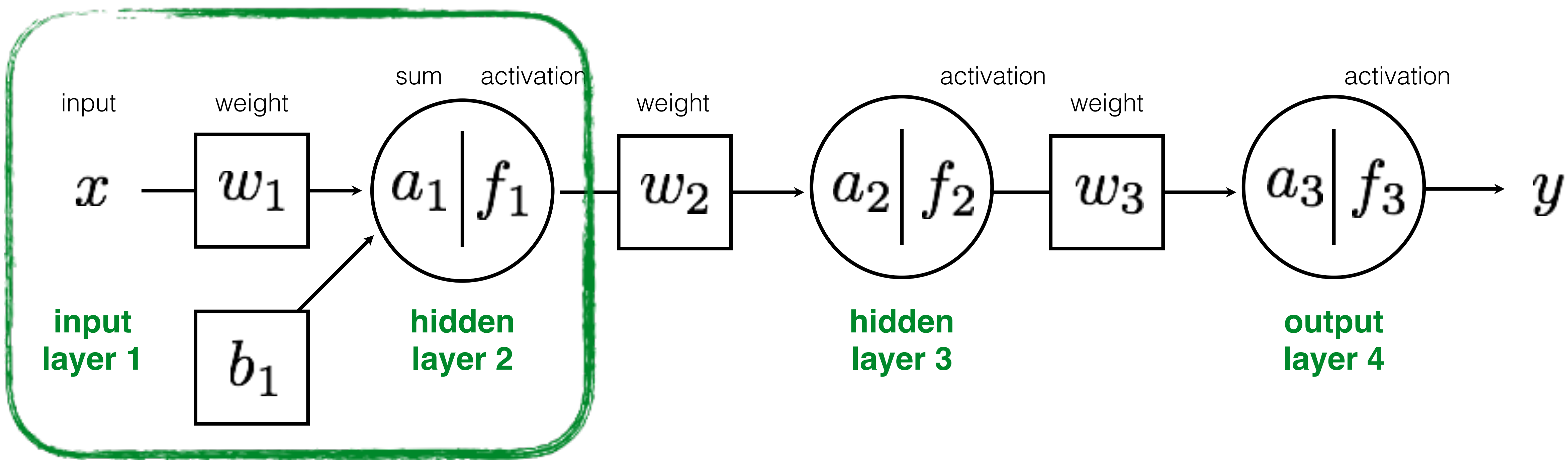
function of **FOUR** parameters and **FOUR** layers!



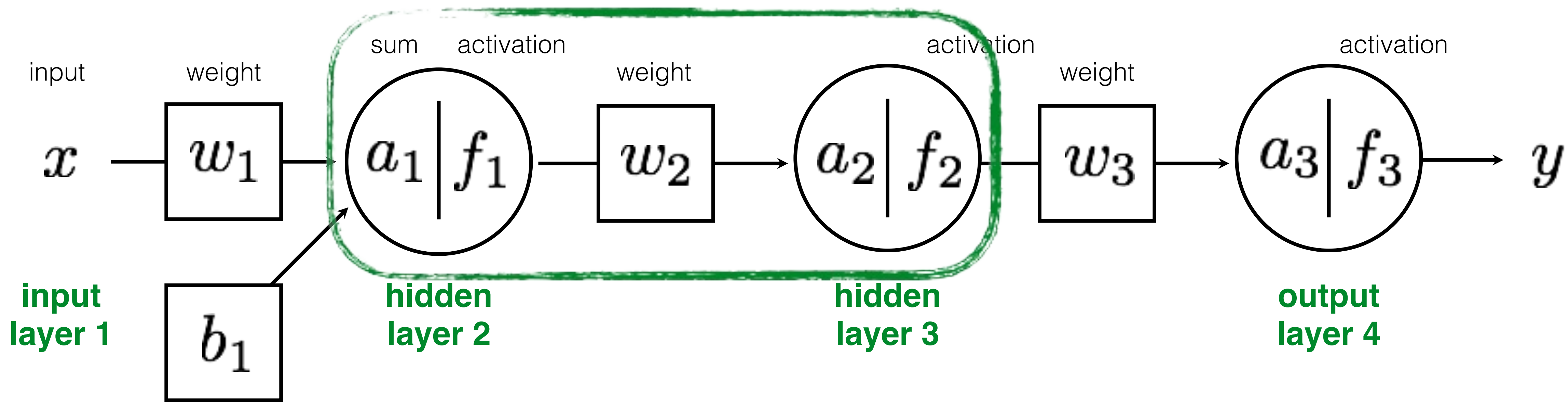
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1
f_1
w_2
a_2
f_2
w_3
a_3
f_3
y
b_1
input
layer 1
hidden
layer 2
hidden
layer 3
output
layer 4



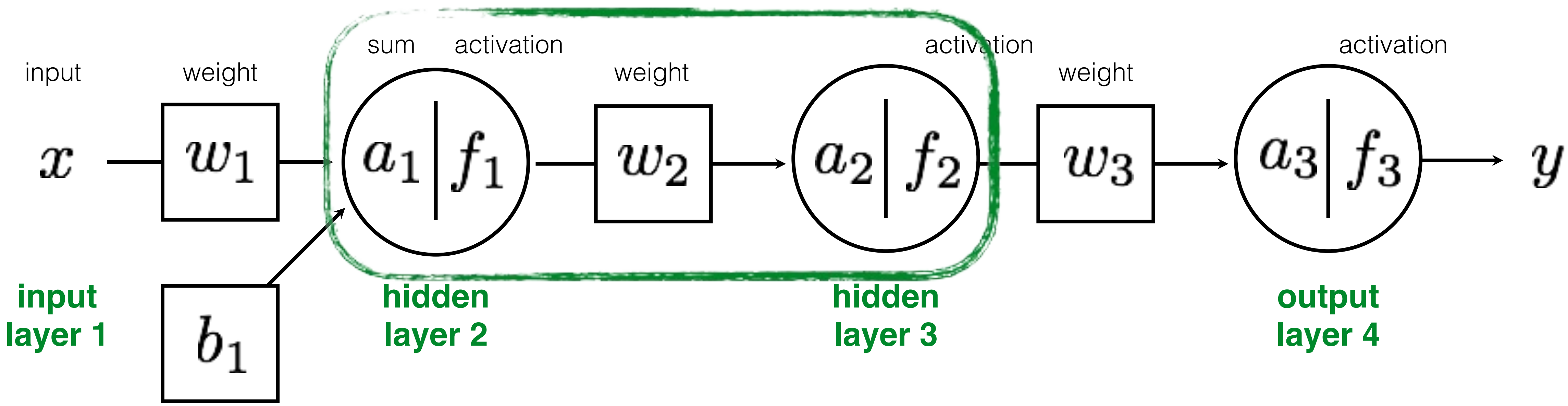
input
weight
sum
activation
weight
activation
weight
activation
x
w_1
a_1 f_1
w_2
a_2 f_2
w_3
a_3 f_3
y
b_1
input layer 1
hidden layer 2
hidden layer 3
output layer 4



$$a_1 = w_1 \cdot x + b_1$$

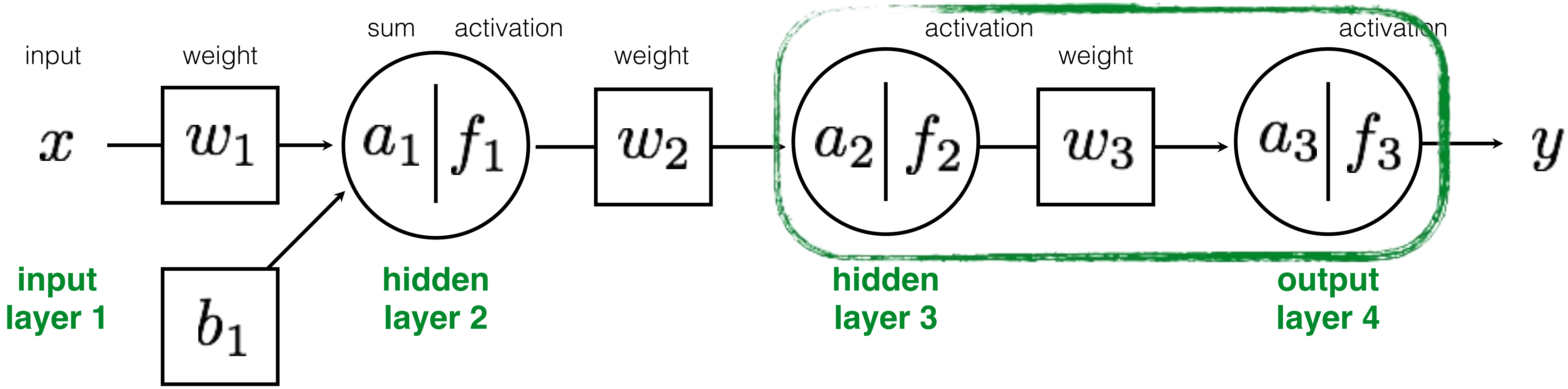


$$a_1 = w_1 \cdot x + b_1$$



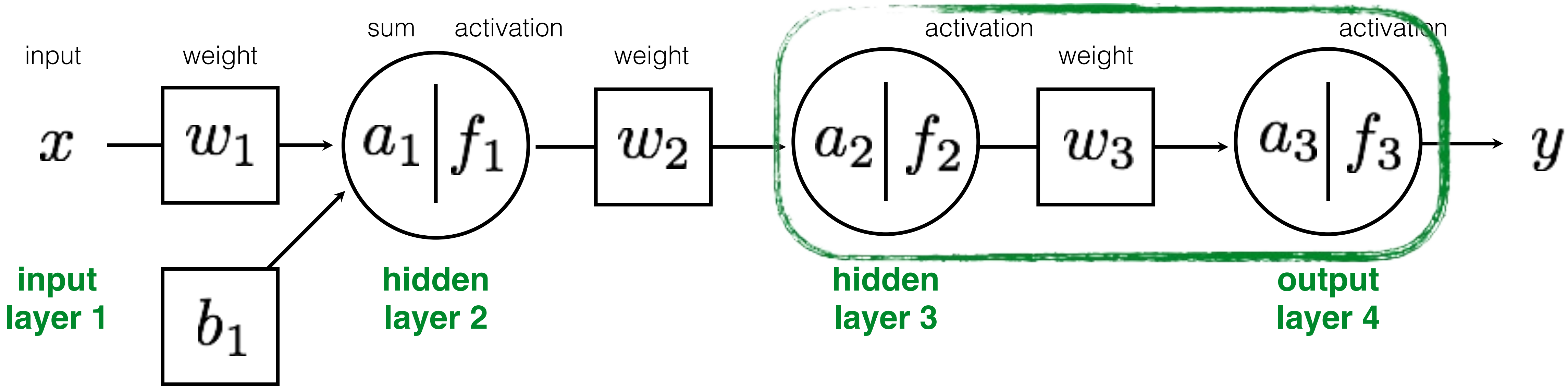
$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

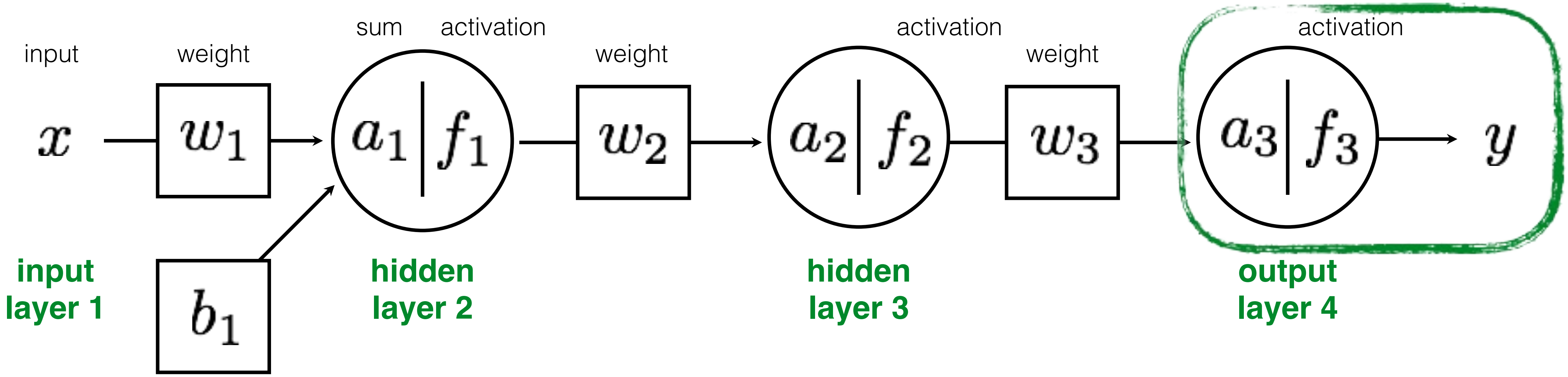
$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

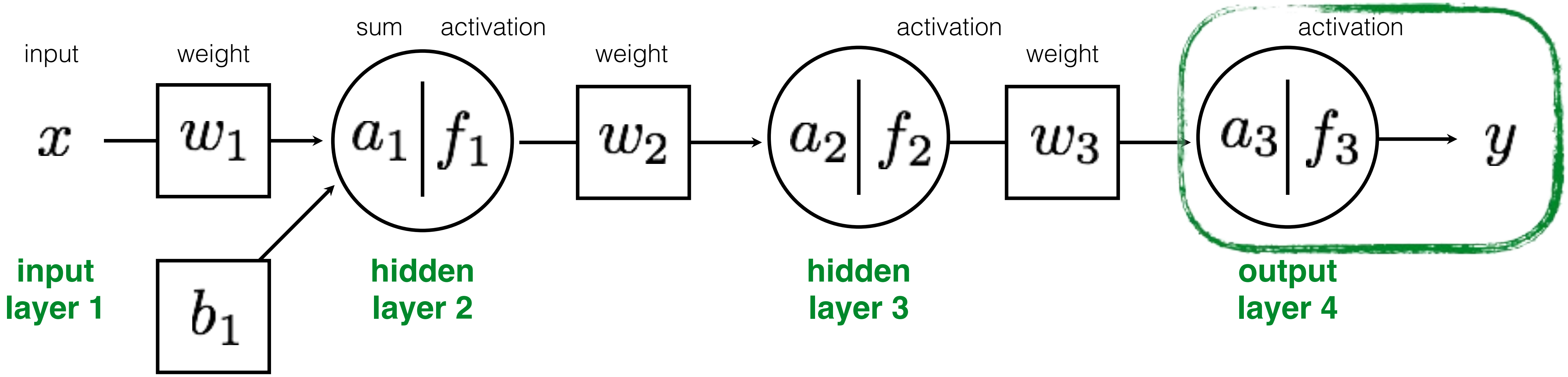
$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$



$$a_1 = w_1 \cdot x + b_1$$

$$a_2 = w_2 \cdot f_1(w_1 \cdot x + b_1)$$

$$a_3 = w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1))$$

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

Entire network can be written out as one long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

known

We need to train the network:

What is known? What is unknown?

Entire network can be written out as a long equation

$$y = f_3(w_3 \cdot f_2(w_2 \cdot f_1(w_1 \cdot x + b_1)))$$

activation function sometimes has parameters

unknown

We need to train the network:

What is known? What is unknown?

Learning an MLP

Given a set of samples and a MLP

$$\{x_i, y_i\}$$

$$y = f_{\text{MLP}}(x; \theta)$$

Estimate the parameters of the MLP

$$\theta = \{f, w, b\}$$

Gradient Descent

For each **random** sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$ vector of parameter partial derivatives
   b. Gradient update $\theta \leftarrow \theta - \eta \nabla \theta$

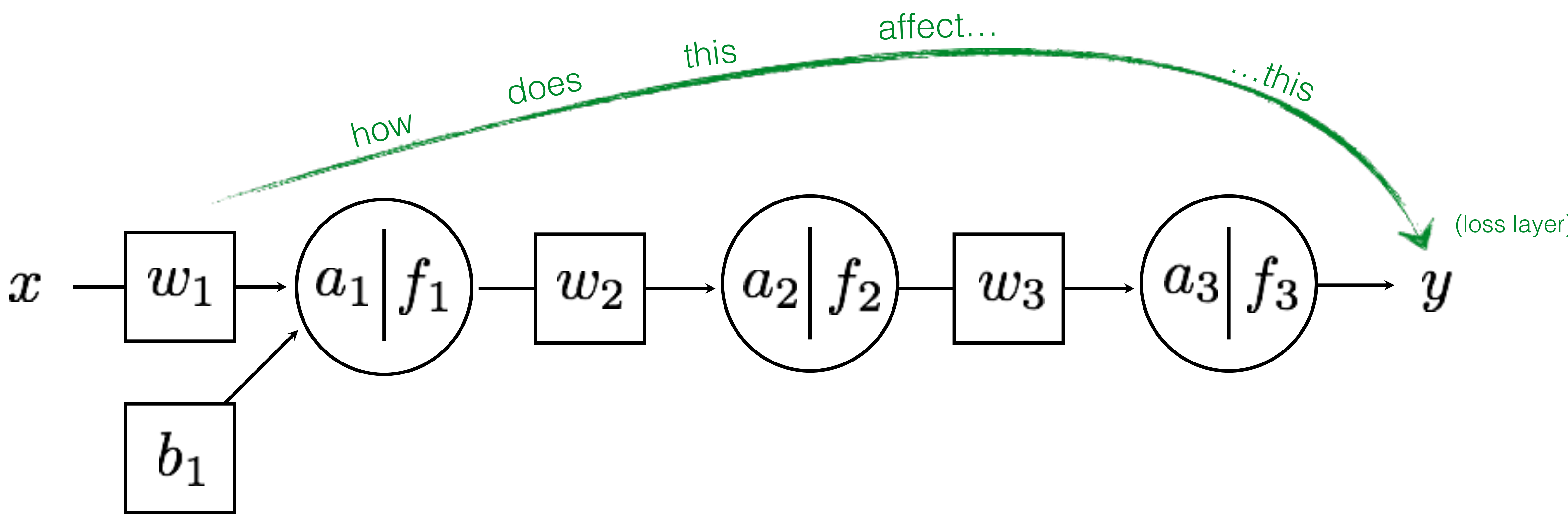
vector of parameter update equations

So we need to compute the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = \left[\frac{\partial \mathcal{L}}{\partial w_3} \frac{\partial \mathcal{L}}{\partial w_2} \frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial b} \right]$$

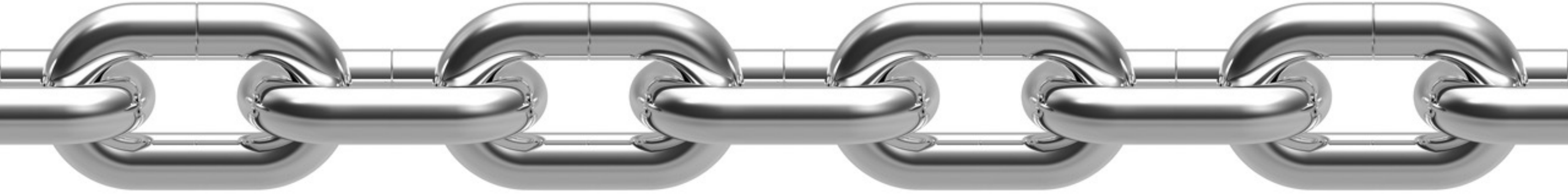
Remember,

Partial derivative $\dfrac{\partial L}{\partial w_1}$ describes…



So, how do you compute it?

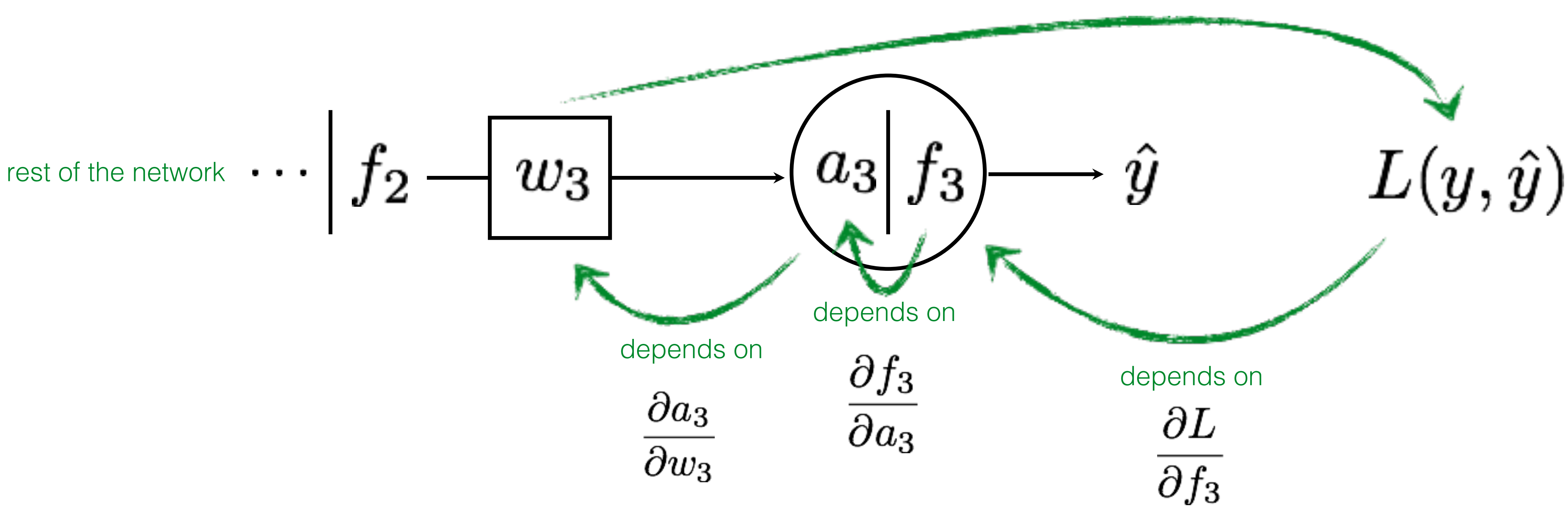
The Chain Rule

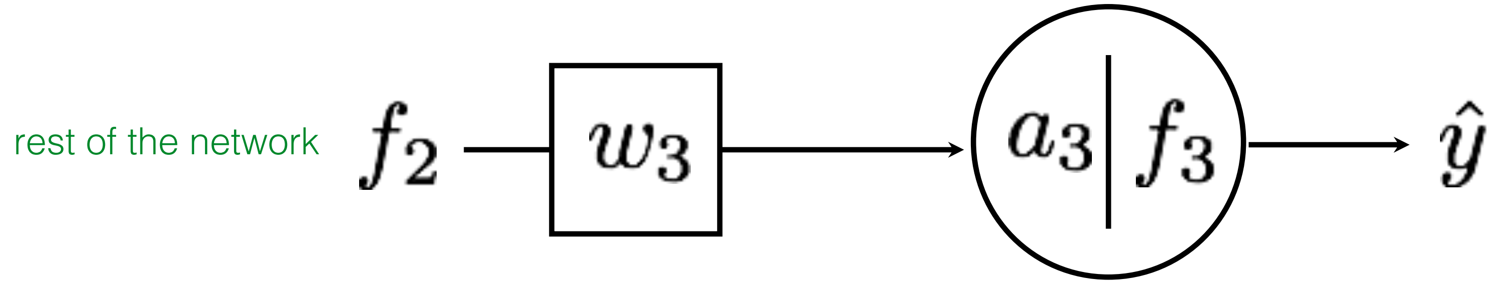


According to the chain rule…

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

Intuitively, the effect of weight on loss function : $\frac{\partial L}{\partial w_3}$

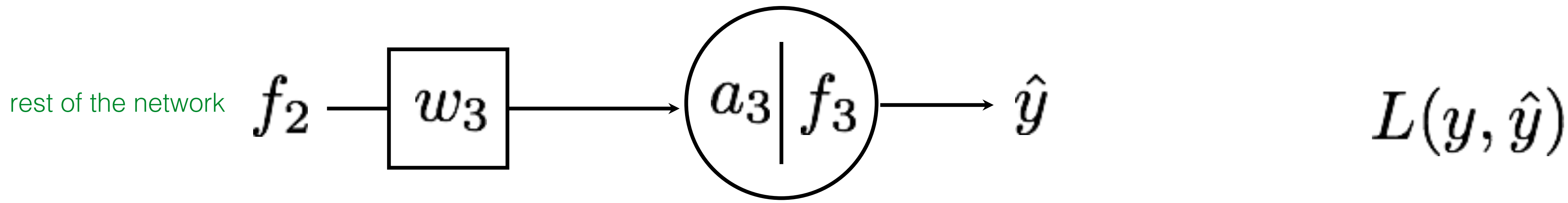




$$L(y, \hat{y})$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3}$$

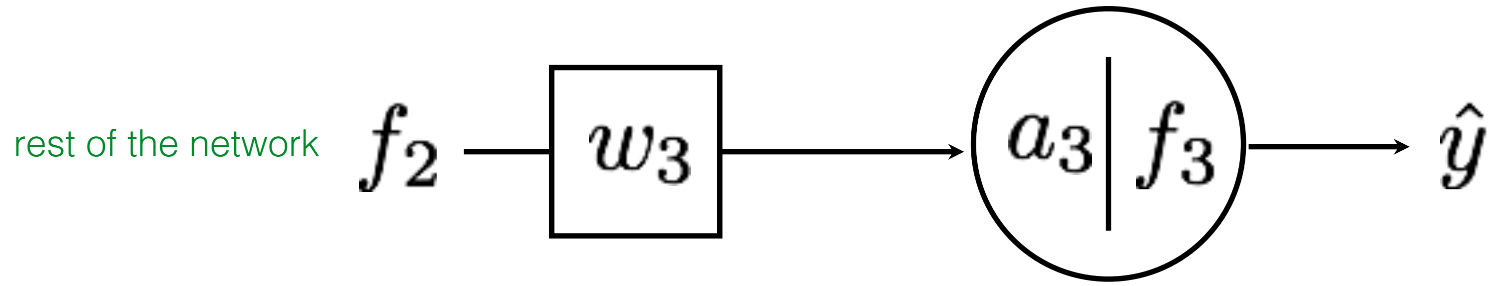
Chain Rule!



$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Just the partial
derivative of L2 loss



$L(y, \hat{y})$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

Let's use a Sigmoid function

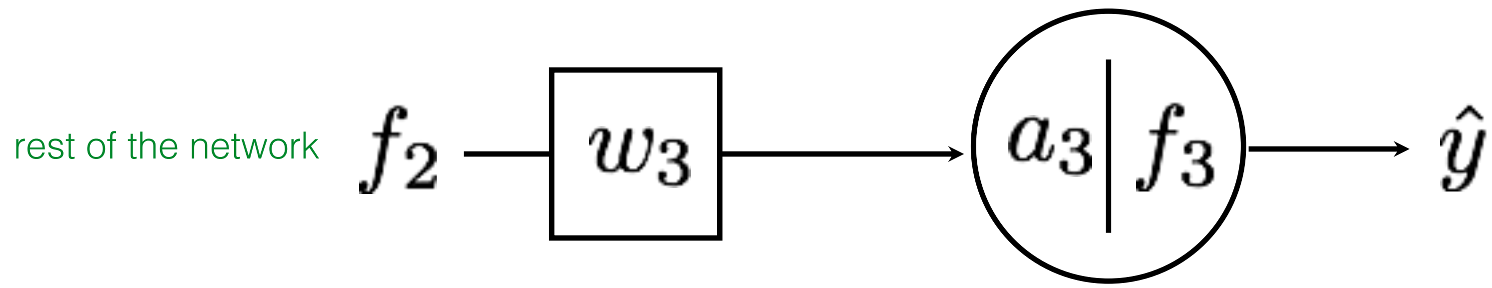
$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

rest of the network $f_2 \longrightarrow \boxed{w_3} \longrightarrow (a_3 \mid f_3) \longrightarrow \hat{y}$ $\qquad L(y, \hat{y})$

$$
\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y}) f_3(1 - f_3) \frac{\partial a_3}{\partial w_3}
\end{aligned}
$$

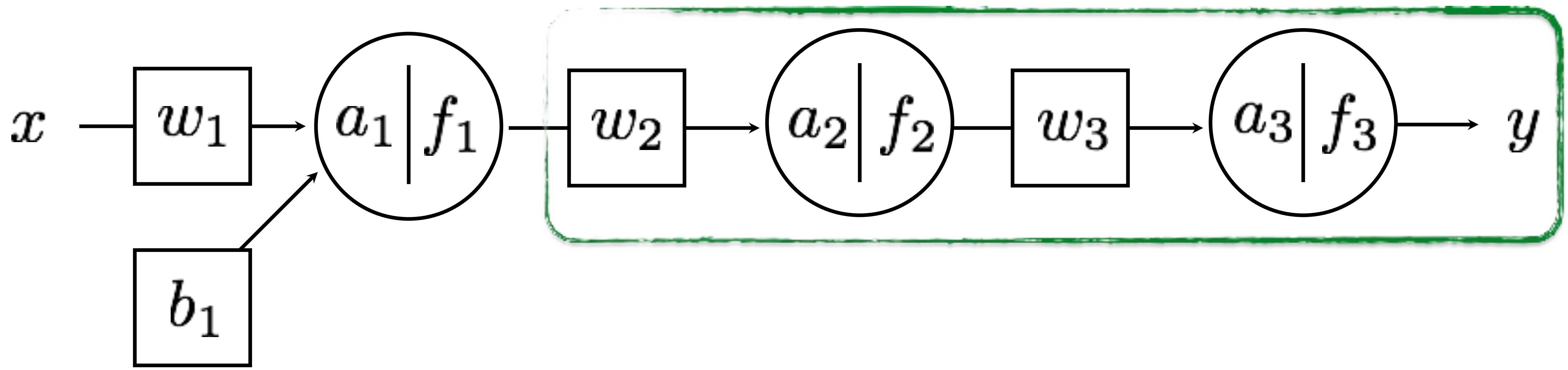
Let's use a Sigmoid function

$$\frac{ds(x)}{dx} = s(x)(1 - s(x))$$

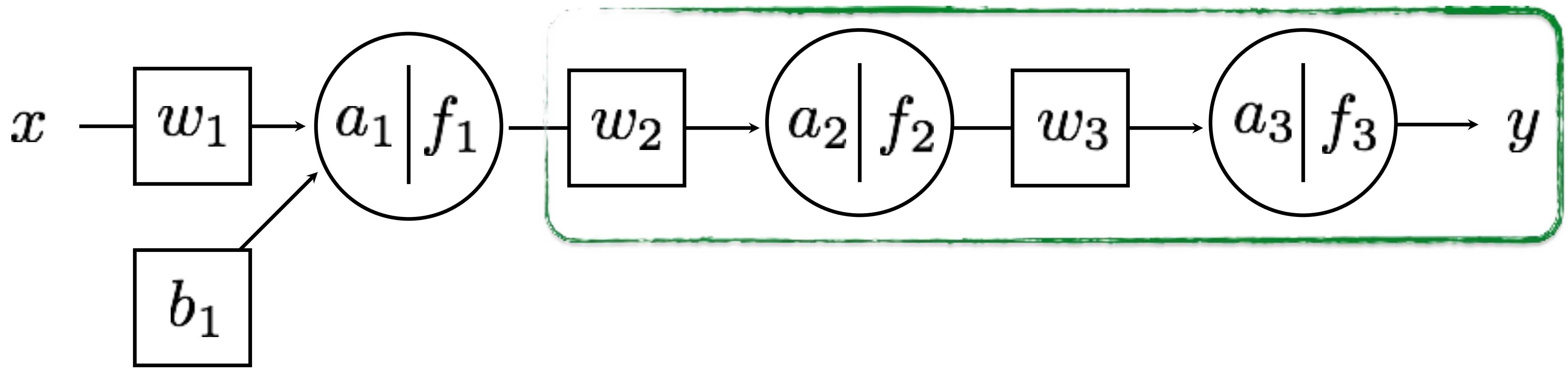


$$L(y, \hat{y})$$

$$\begin{aligned}
\frac{\partial L}{\partial w_3} &= \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})f_3(1 - f_3)\frac{\partial a_3}{\partial w_3} \\
&= -\eta(y - \hat{y})f_3(1 - f_3)f_2
\end{aligned}$$



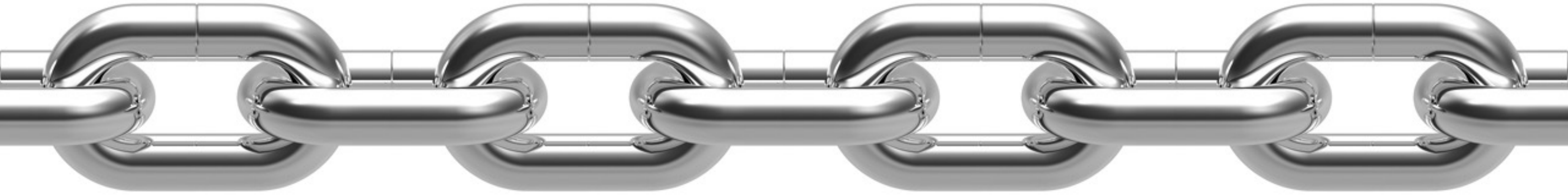
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$



$$\frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial f_3} \frac{\partial f_3}{\partial a_3}} \frac{\partial a_3}{\partial f_2} \frac{\partial f_2}{\partial a_2} \frac{\partial a_2}{\partial w_2}$$

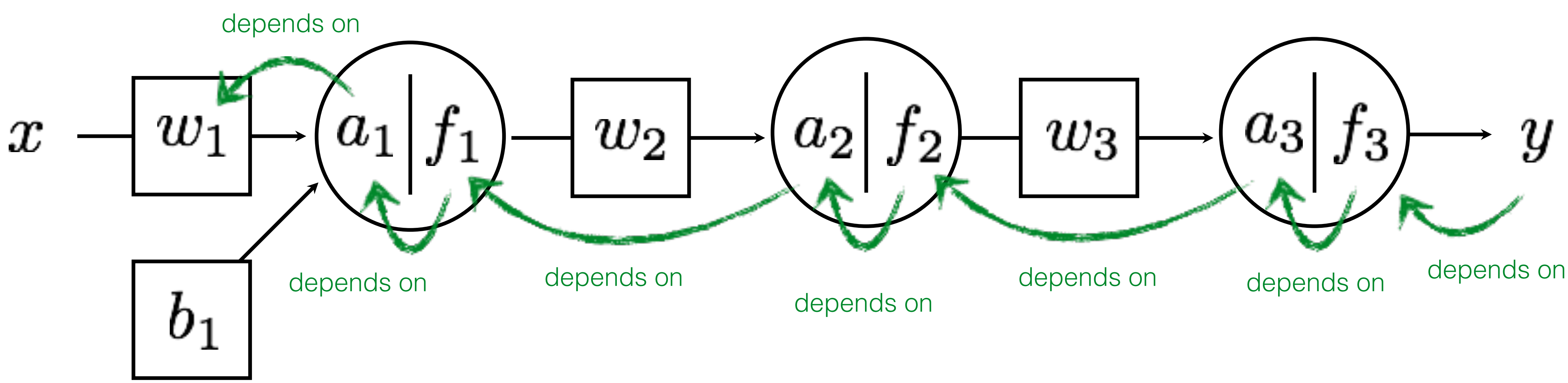
already computed.
re-use (propagate)!

The Chain Rule



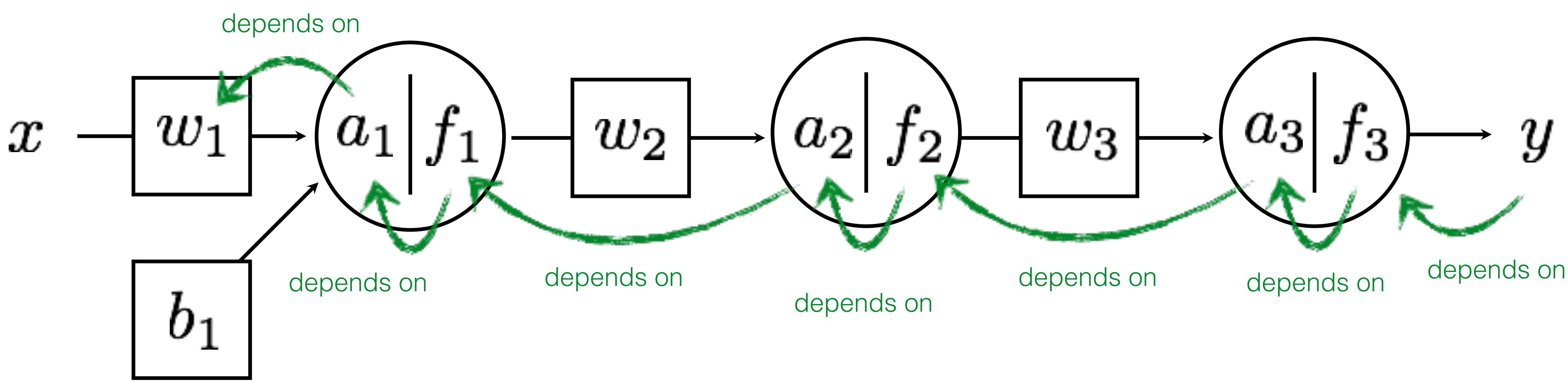
a.k.a. backpropagation

The chain rule says…



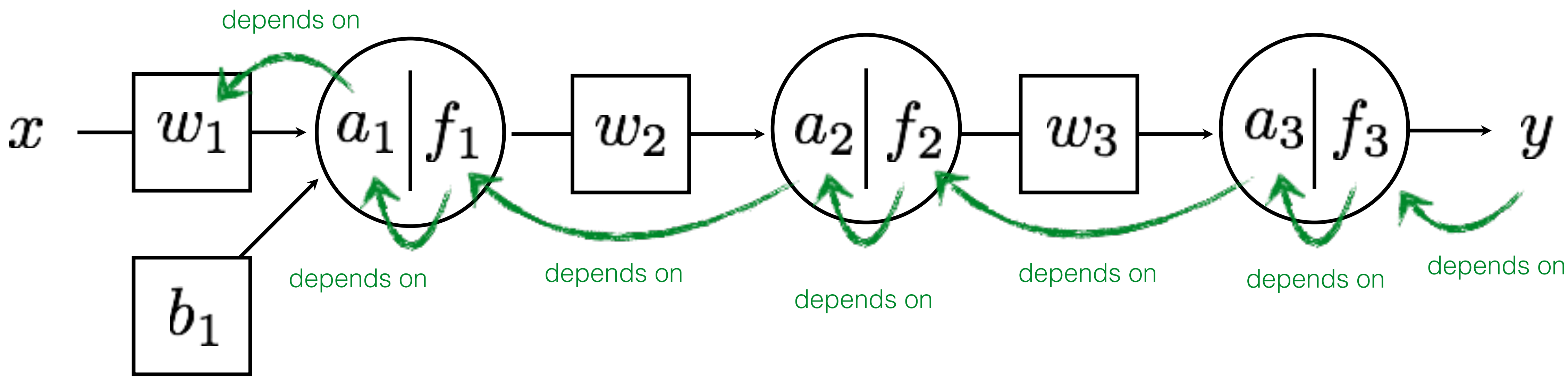
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

The chain rule says…



$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

already computed.
re-use (propagate)!

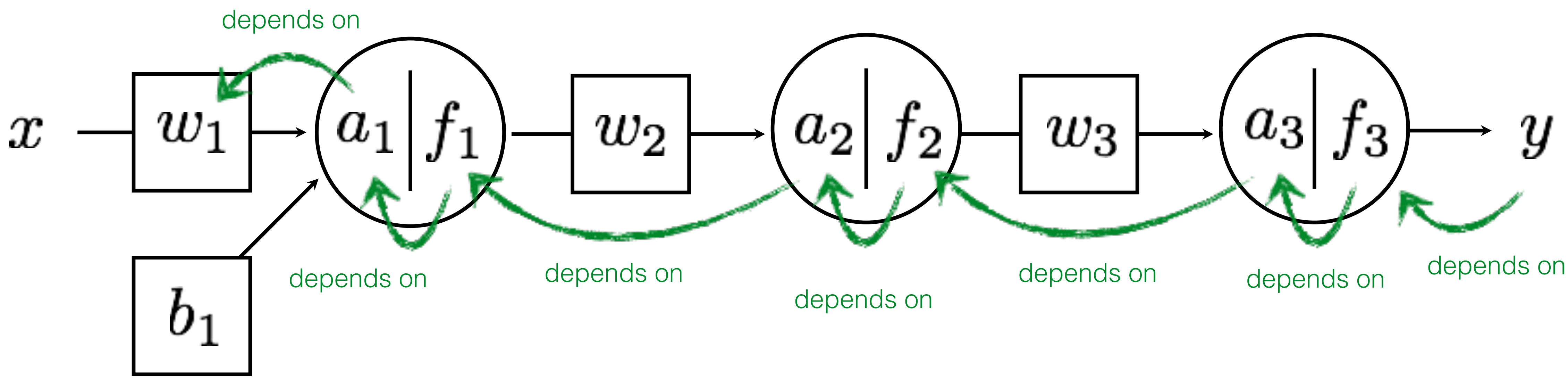


$$\frac{\partial \mathcal{L}}{\partial w_3} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

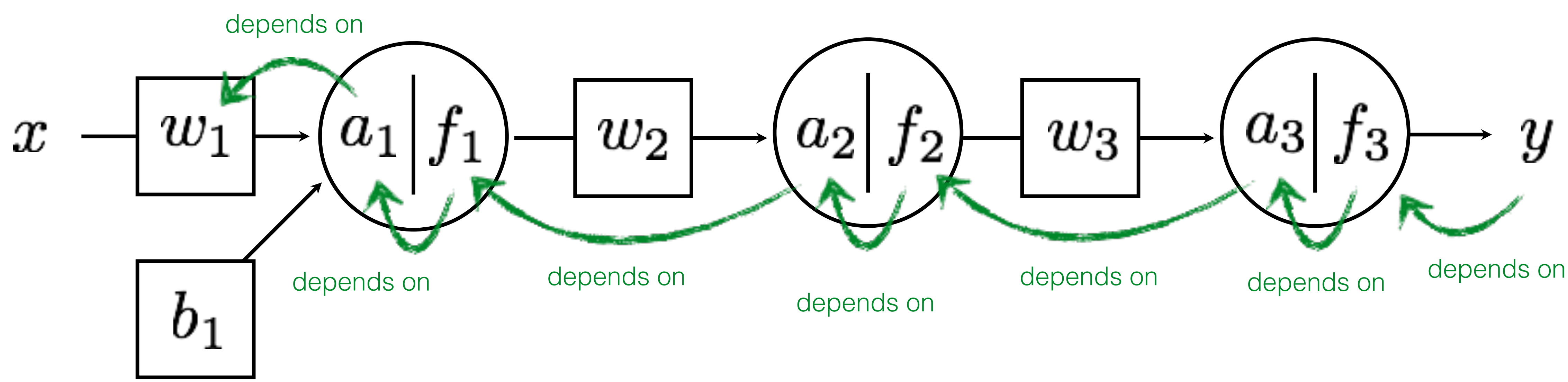


$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$



$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial w_1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \boxed{\frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}}\frac{\partial a_1}{\partial b}$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

   b. Compute Loss $\mathcal{L}_i$

2. Update

   a. Back Propagation

$$\frac{\partial \mathcal{L}}{\partial w_3} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial w_3}$$
$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial w_2}$$
$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial w_1}$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial f_3}\frac{\partial f_3}{\partial a_3}\frac{\partial a_3}{\partial f_2}\frac{\partial f_2}{\partial a_2}\frac{\partial a_2}{\partial f_1}\frac{\partial f_1}{\partial a_1}\frac{\partial a_1}{\partial b}$$

   b. Gradient update

$$w_3 = w_3 - \eta \nabla w_3$$
$$w_2 = w_2 - \eta \nabla w_2$$
$$w_1 = w_1 - \eta \nabla w_1$$
$$b = b - \eta \nabla b$$

Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict

   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$

   b. Compute Loss $\mathcal{L}_i$

2. Update

   a. Back Propagation $\dfrac{\partial \mathcal{L}}{\partial \theta}$

   vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \dfrac{\partial \mathcal{L}}{\partial \theta}$

   vector of parameter update equations

Stochastic gradient descent

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

What we are truly minimizing:

$$\min_{\theta} \sum_{i=1}^{N} L(y_i, f_{MLP}(x_i))$$

The gradient is:

$$\sum_{i=1}^{N} \frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta}$$

What we use for gradient update is:

$$\frac{\partial L(y_i, f_{MLP}(x_i))}{\partial \theta} \quad \text{for some i}$$

Stochastic Gradient Descent

For each example sample $\{x_i, y_i\}$

1. Predict
   a. Forward pass $\hat{y} = f_{\text{MLP}}(x_i; \theta)$
   b. Compute Loss $\mathcal{L}_i$
2. Update
   a. Back Propagation $\frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter partial derivatives

   b. Gradient update $\theta \leftarrow \theta + \eta \frac{\partial \mathcal{L}}{\partial \theta}$

      vector of parameter update equations

How do we select which sample?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size $B < N$.

Why not do gradient descent with all samples?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size $B < N$.

Why not do gradient descent with all samples?

- It’s very expensive when N is large (big data).

Do I lose anything by using stochastic GD?

How do we select which sample?

- Select randomly!

Do we need to use only one sample?

- You can use a *minibatch* of size $B < N$.

Why not do gradient descent with all samples?

- It's very expensive when N is large (big data).

Do I lose anything by using stochastic GD?

- Same convergence guarantees and complexity!
- Better generalization.