Adaline Model

Adaline Model (Widrow & Hoff)

"Adaline" = Adaptive Linear Neuron
or
"Element"

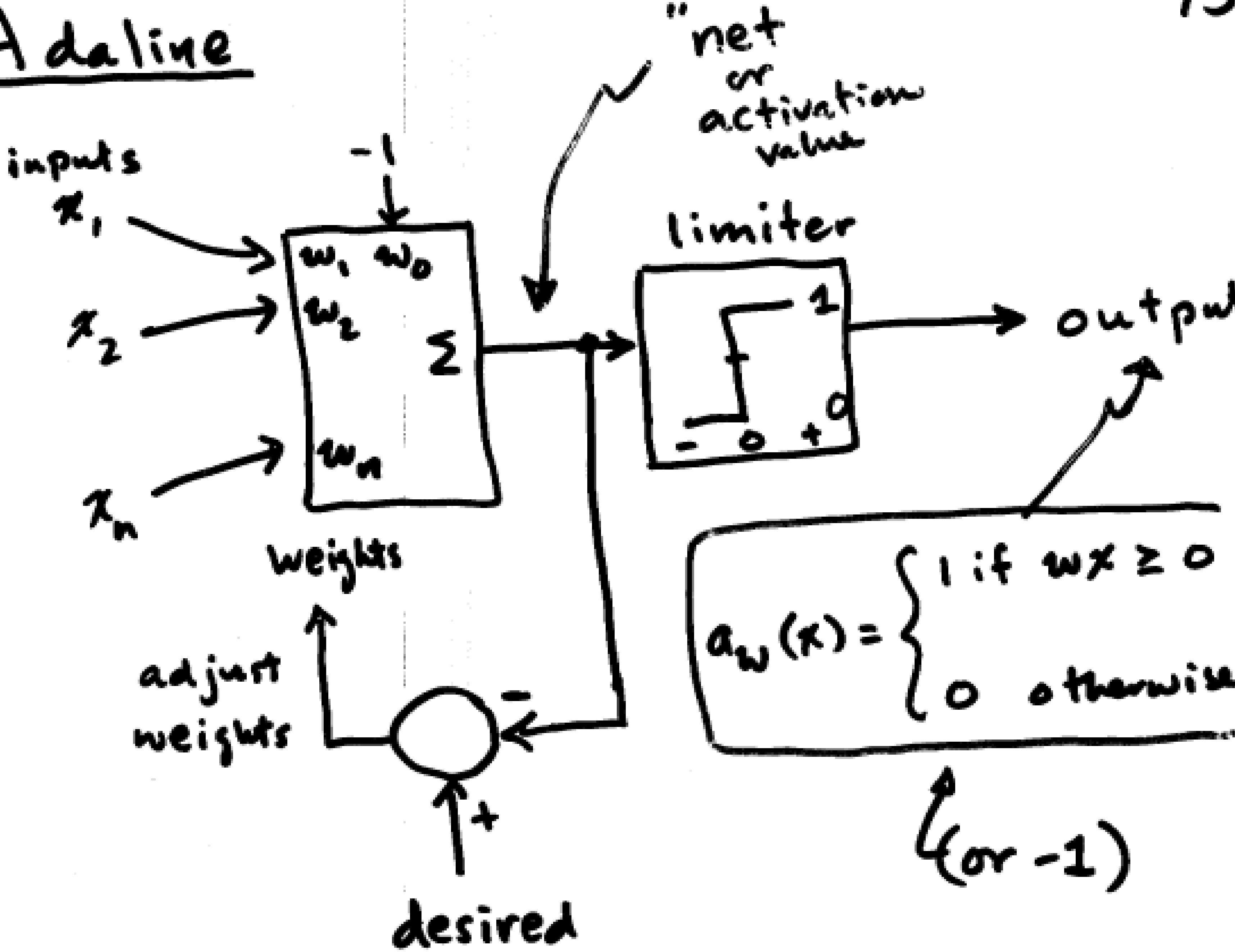
It is essentially the same as a perceptron, but the training model is different.

The model is a special case of gradient descent which is used in a wide variety of networks for training.

Unlike Perceptron training, an Adaline continues to learn even when it gives the <u>right</u> answer.

This is done by having the training be based on the "<u>net</u>" or <u>activation</u> value, rather than on the output.

Adaline



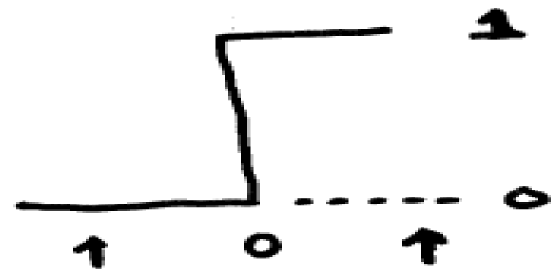
$$a_w(x) = \begin{cases} 1 \text{ if } wx \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

(or -1)

A problem to be addressed is:

What is the <u>desired</u> <u>activation</u> <u>value</u> for a sample, since desired <u>outputs</u> are expressed instead?

Since the output is a limiter function applied to the activation, we can pick <u>nominal</u> activation values to get the desired output.



Picking desired activation of, say −0.5, 0.5 might be ok.

Picking 0, 1 would not be so good, since 0 is right on the boundary.

Adaline Training

Adaline Training

Given input vector x,

desired net value d

the error is defined as

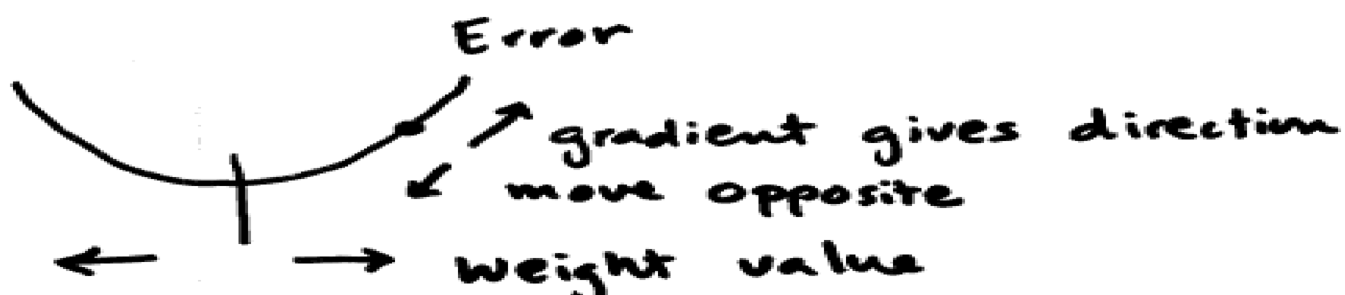
$$E = d - wx$$
$$= d - \sum w_i x_i$$

Since the error is signed, it is common to minimize the squared error, so the system converges toward zero error regardless of sign.

$$E^2 = (d - wx)^2$$

Gradient Descent

To move the squared error toward 0, we can compute the gradient which gives the direction of error increase, and move the weights in the opposite direction.



The gradient is computed treating the squared-error function E as a function of $\underline{w}$, with the input(s) fixed:

$$E^2 = E^2(\underline{w})$$

$$\nabla_{w} E^2 = \left(\frac{\partial E^2}{\partial w_0}, \frac{\partial E^2}{\partial w_1}, \dots, \frac{\partial E^2}{\partial w_n} \right)$$

vector of partial derivatives

The gradient is computed by computing each component

$$\frac{\partial E^2}{\partial w_i}$$

$$= \frac{\partial}{\partial w_i}(d - wx)^2$$

$$= 2\underbrace{(d - wx)}_{= E}\underbrace{\frac{\partial}{\partial w_i}(d - wx)}_{= -\underbrace{\frac{\partial}{\partial w_i} wx}}$$

$$= \frac{-\partial}{\partial w_i}(w_0 \cdot -1 + w_1 x_1 + w_2 x_2 + \ldots)$$

$$= -x_i$$ (where $x_0 = -1$)

Summarizing

$$\frac{\partial E^2}{\partial w_i} = -2 E x_i$$

is the gradient component for the $i\underline{th}$ weight.

We adjust the weight <u>opposite</u> to this, so we have the rule

$$\text{new } w_i = w_i + \eta E x_i$$

η ← learning rate (incorporating the 2)

or in vector form

$$\boxed{\text{new } w = w + \eta E x}$$

Adaline learning rule

The Adaline training algorithm applies the rule

$$w = w + \Delta w;$$

for $\Delta w = \eta E x$

training input

error in training input

learning rate

<u>until</u>

(a) all samples correctly classified

or (b) a specified MSE over all samples is achieved

or (c) a specified number of epochs reached

The Adaline training algorithm presented,

$$\Delta w = \eta \cdot \text{error} \cdot x$$

change in weight vector

desired - actual

input vector

is called the <u>LMS algorithm</u>

"least mean-square"

A modification, which improves training is called the

<u>α-LMS</u> or <u>Widrow-Hoff rule</u>

$$\Delta w = \eta \cdot \text{error} \cdot \frac{x}{\|x\|}$$

length x

The motivation for this modification is that $\frac{1}{\|x\|}$ adjusts the value of w in favor of classifying x with little effect on classification of other inputs.

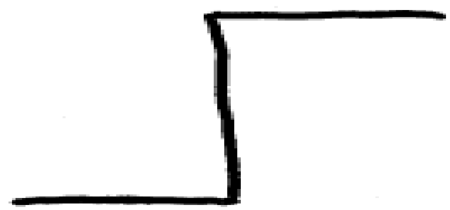
Generalizing Adaline Training

(moving toward backpropagation)

Recall that for discrete output we were forced to choose target activation values such as -0.5 , 0.5 to get outputs 0 1

Not all choices will work equally well. It would be nice if the neuron learned the choice, instead.

Continuous approximation to a hard limiter



limiter
(discrete)

Sigmoid
(continuous)

sigmoid with
greater spread

limited linear

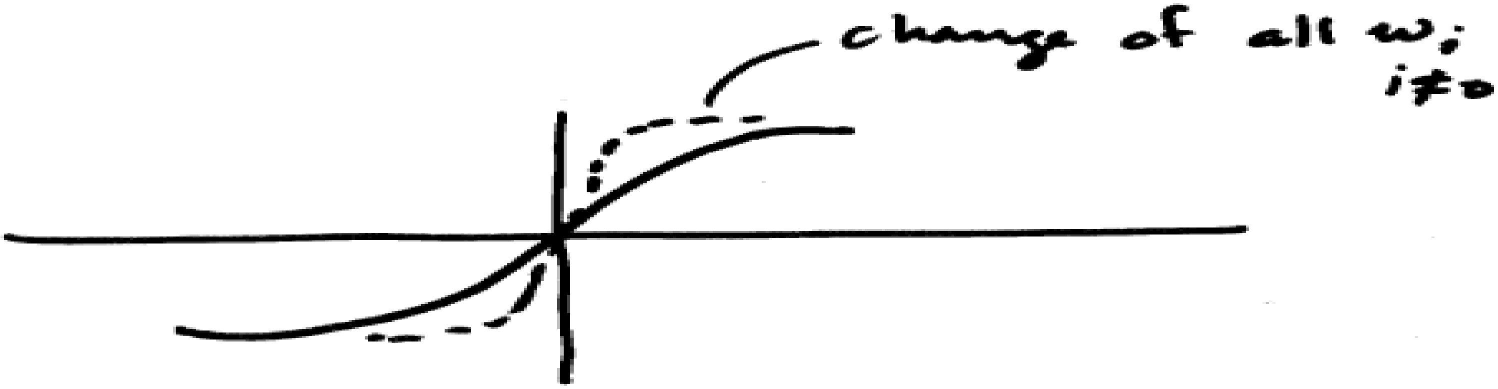
The sigmoid curve is capable of "imitating" a range of behaviors depending on the slope.

There are actually many curves of sigmoid character. A common one is the <u>logistic function</u> or "logsig"

$$f(y) = \frac{1}{1 + e^{-y}}$$

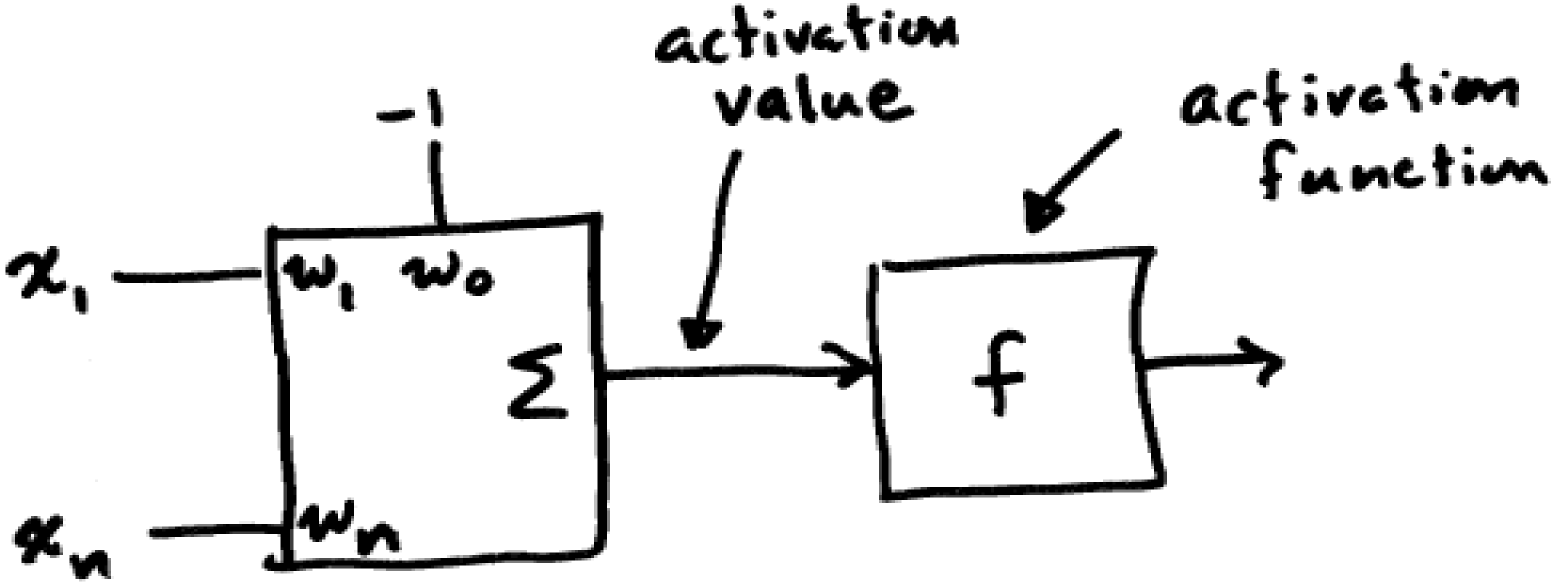
Since y will be the activation value $\sum_i w_i x_i$ the effective slope is adjusted by the weights.

The sigmoid can also effectively be translated by the constant term in $\sum w_i x_i$ $(x_0 = -1)$



As w_i's $\to \infty$, sigmoid steepens.

Our generalized Adaline looks like



where f is, say, a sigmoid.

How to <u>train</u> the general model?

Generalized Adaline Training

Training Adaline with Generalized Activation Function using Gradient Descent

$$\text{Error} = E(w) = d - f(y)$$

desired

activation function

activation value

$= w \cdot x$

$= \sum_i w_i x_i$

identity for regular adaline, could be sigmoid

$E(w)$ because inputs regarded as fixed during training

Gradient descent depends on computing or estimating the <u>gradient</u>

$$\nabla_w E^2$$

squared error function (sum of squared errors over all inputs)

which is a <u>vector</u> with one component per weight w_i

$$\frac{\partial}{\partial w_i} E^2$$

From calculus differentiation rule for square of a function:

$$\frac{\partial}{\partial w_i} E^2 = 2E \frac{\partial E}{\partial w_i}$$

Since $E = d - f(y)$

$$\frac{\partial E}{\partial w_i} = \underbrace{\frac{\partial d}{\partial w_i}}_{=0} - \frac{\partial}{\partial w_i} f(y)$$

constant

So $$\frac{\partial}{\partial w_i} E^2 = -2E \frac{\partial}{\partial w_i} f(y)$$

How to find $\frac{\partial}{\partial w_i} f(y)$?

$$y = wx$$

Regard y as a function, say g, of w:

$$y = g(w) = wx$$

We want to compute

$$\frac{\partial}{\partial w_i} \underbrace{f(g(w))}$$

<u>composition</u> of f and g

Chain Rule of Calculus

For any function φ, φ' denotes its **derivative** (assuming it has one).

$f \circ g$ denotes the **composition** of f with g, i.e.

$$(f \circ g)(w) = f(g(w))$$

Chain Rule:

$$\underset{\substack{\uparrow \\ \text{composition}}}{(f \circ g)'}(w) = f'(g(w)) \underset{\substack{\uparrow \\ \text{product}}}{\cdot} g'(w)$$

Our application:

$$\frac{\partial}{\partial w_i} f(g(w))$$

$$= \underbrace{f'(g(w))}_{= f'(y)} \cdot \left(\frac{\partial}{\partial w_i} g\right)(w)$$

activation value

$$g(w) = w_0 x_0 + w_1 x_1 + \dots + w_n x_n$$

$$\frac{\partial}{\partial w_i} g = x_i$$

since $\frac{\partial}{\partial w_i} w_j \cdot x_j = \begin{cases} x_j & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$

variables

So $\boxed{\frac{\partial}{\partial w_i} f(g(w)) = f'(y) \cdot x_i}$

Summary

For $f(y) = \dfrac{1}{1+e^{-y}}$

$$f'(y) = f(y)\,(1 - f(y))$$

So once $f(y)$ is compute, as a value z, say

$f'(y)$ can be computed

algebraically as $z\,(1-z)$

Summary

Gradient component

$$-2E \frac{\partial}{\partial w_i} f(y)$$

Essentially 3.8 in text.

$$= -2E \underbrace{f'(y)}_{\text{derivative of } f \text{ evaluated at activation value}} x_i$$

We only require that f have a <u>derivative</u> in order to train.

Consider case of f being a <u>sigmoid</u>:

$$f(y) = \frac{1}{1+e^{-y}}$$

Using the rule $\frac{d}{dy}\left(\frac{1}{g}\right) = \frac{-g'}{g^2}$

$$f'(y) = \frac{-\frac{d}{dy}(1+e^{-y})}{(1+e^{-y})^2} = \frac{e^{-y}}{(1+e^{-y})^2}$$

$$= \frac{1+e^{-y}}{(1+e^{-y})^2} - \frac{1}{(1+e^{-y})^2}$$

$$= \frac{1}{1+e^{-y}} - \frac{1}{(1+e^{-y})^2}$$

$$= f(y) - f(y)^2 = f(y)(1-f(y))$$

Weight Update for Sigmoid Adeline

$$\Delta w_i = 2E \frac{\partial}{\partial w_i} f(y) \cdot \eta$$

$$= 2E\, f'(y)\, x_i \cdot \eta$$

$$= 2E \cdot f(y)\,(1 - f(y))\, x_i \cdot \eta$$

Incorporating factor of 2 into η

$$\boxed{\Delta w_i = \eta\, E\, f(y)\,(1 - f(y))\, x_i}$$

where η = learning rate

E = error = $\underbrace{d}_{\text{desired}} - f(y)$

y = activation value = $w \cdot x$

x_i = i^{th} input value

Algorithm:

Initialize w to random reals.

Set η to a "nominal" rate, such as 0.1.

```
done = false;
while( ¬done )
    {
    done = true;
    for( j = 1 to N )
        {
        error = y^j - p_w(x^j);
        if( error ≠ 0 )
            {
            done = false;
            w = w + η * error * x^j;
            }
        }
    }
```