Artificial Neural Networks
Lecture Notes

Stephen Lucci, PhD

Part 4

About this file:

- If you have trouble reading the contents of this file, or in case of transcription errors, email gi0062@bcmail.brooklyn.cuny.edu
- Acknowledgments:
  Background image is from http://www.anatomy.usyd.edu.au/online/neuroanatomy/tutorial1/tutorial1.html (edited) at the University of Sydney Neuroanatomy web page. Mathematics symbols images are from metamath.org's GIF images for Math Symbols web page. Other image credits are given where noted, the remainder are native to this file.

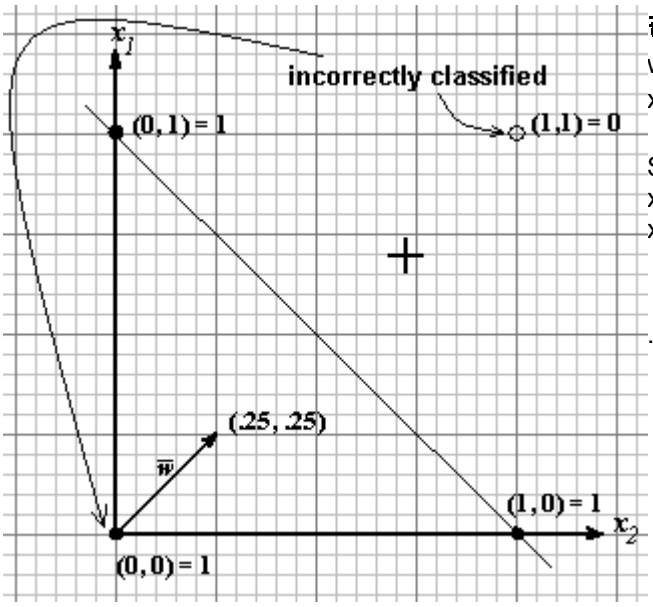
Contents

Perceptron Learning - An example

A Two-Input NAND

x_1	x_2	x_1 NAND x_2
0	0	1
0	1	1
1	0	1
1	1	0

Let $w_1 = w_2 = \theta = 0.25$ to begin.



$\overline{w} \cdot \overline{x} = \theta$

$w_1x_1 + w_2x_2 = \theta$

$x_2 = -(w_1 / w_2)x_1 + (\theta / w_2)$

Substituting, we obtain

$x_2 = -(0.25 / 0.25)x_1 + (0.25 / 0.25)$

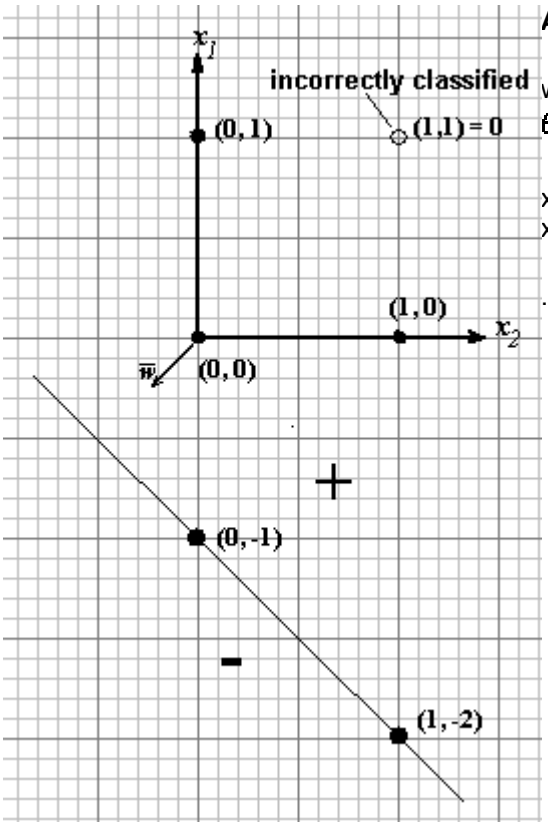
$x_2 = -x_1 + 1$

. i.e.,

x_1	x_2
0	1
1	0

The First Epoch:

w_1	w_2	θ	x_1	x_2	a	y	t	α(t-y)	Δw_1	Δw_2	$\Delta \theta$
.25	.25	.25	0	0	0	0	1	.5(1-0)=.5	0	0	-.5
.25	.25	-.25	0	1	.25	1	1	.5(1-1)=0	0	0	0
.25	.25	-.25	1	0	.25	1	1	0	0	0	0
.25	.25	-.25	1	1	.5	1	0	.5(0-1)=-.5	-.5	-.5	.5



After the First Epoch

$w_1 = -0.25 = w_2$

$\theta = +0.25$

$x_2 = -(w_1 / w_2)x_1 + (\theta / w_2)$

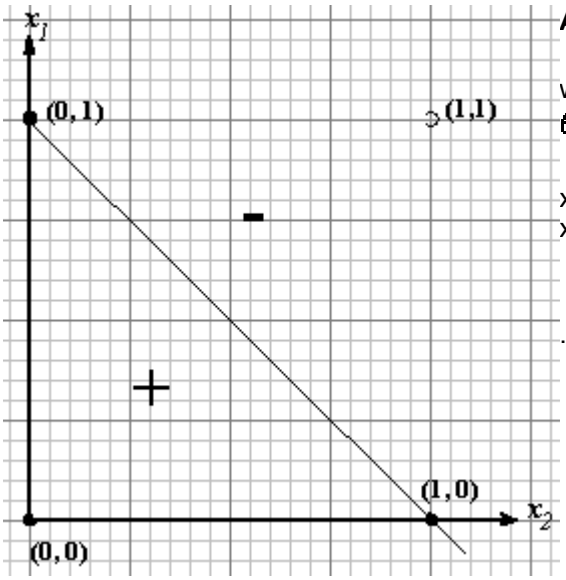
$x_2 = -x_1 - 1$

. i.e.,

x_1	x_2
0	-1
1	-2

Second Epoch:

w_1	w_2	θ	x_1	x_2	a	y	t	α(t-y)	Δw_1	Δw_2	$\Delta \theta$
-.25	-.25	.25	0	0	0	0	1	.5(1-0)=.5	0	0	-.5
-.25	-.25	-.25	0	1	-.25	1	1	0	0	0	0
-.25	-.25	-.25	1	0	-.25	1	1	0	0	0	0
-.25	-.25	-.25	1	1	-.5	0	0	0	0	0	0



After the Second Epoch

$w_1 = w_2 = -0.25$
$\theta = -0.25$

$x_2 = -(w_1 / w_2)x_1 + (\theta / w_2)$
$x_2 = -x_1 + 1$

. i.e.,

x_1	x_2
0	1
1	0

Third Epoch:

w_1	w_2	θ	x_1	x_2	a	y	t	α(t-y)	Δw_1	Δw_2	$\Delta \theta$
-.25	-.25	-.25	0	0	0	1	1	0	0	0	0
-.25	-.25	-.25	0	1	-.25	1	1	0	0	0	0
-.25	-.25	-.25	1	0	-.25	1	1	0	0	0	0
-.25	-.25	-.25	1	1	-.25	0	0	0	0	0	0

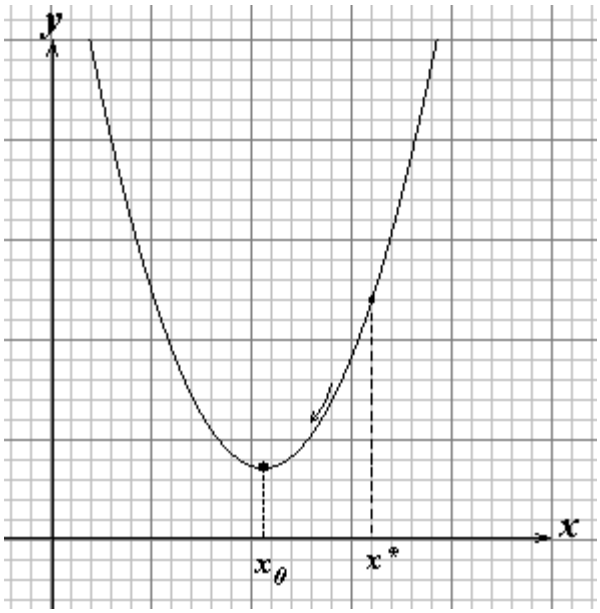
Since there have been no changes, ∴Halt!

The Delta Rule

We desire:

1. Capability to train all the weights in multilayer nets with no a priori knowledge of the training set.
2. Based on defining a measure of the difference between the actual network output and target vector.
3. This difference is then treated as an error to be minimized by adjusting the weights.

Finding the Minimum of a Function: Gradient Descent (*informed hillclimbing?*)



Suppose that quantity y depends on a single variable x.

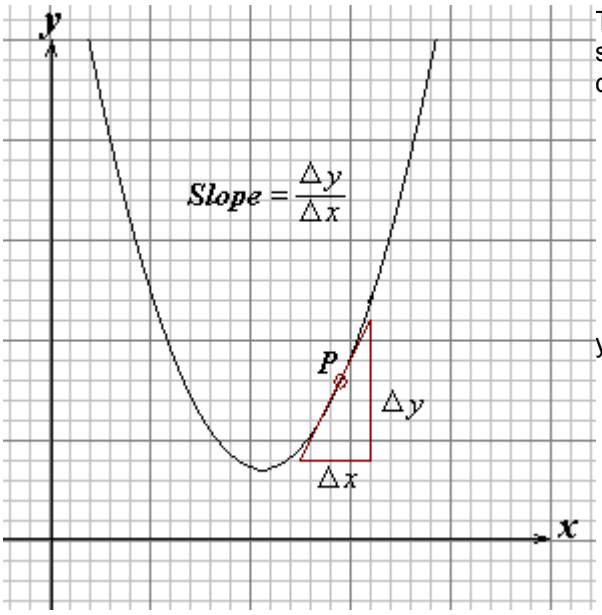
i.e., y = y(x).

We wish to find x_0 which minimizes x.

i.e., $y(x_0) <= y(x)$, ∀x.

- Let x* be current best estimate for x_0.
- To obtain a better estimate for x_0, choose ∆x so as to follow the function downhill.
- We need to know the *slope* of the function at x*:

Slope of a Function



The **slope** at any point x is just the slope of a straight line, the **tangent**, which just grazes the curve at that point.

1. Here, one may draw the function on graph paper
2. Draw the tangent at the point P
3. Measure the sides ∆x, ∆y, or merely calculate:

y'(x = P).

If Δx is small enough, $\delta y = \Delta y$.

Dividing Δy by Δx, and then multiplying by Δx leaves Δy unchanged.

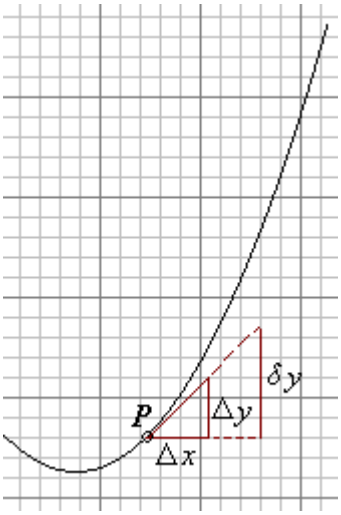
$\Delta y = (\Delta y / \Delta x) \Delta x$

Furthermore, $\delta y \approx \Delta y$.

Hence, we may write: $\delta y = \text{slope} \cdot \Delta x$.
That is,

$$\delta y = (dy / dx) \cdot \Delta x. \qquad (*)$$

That is, the derivative of y with respect to x.



Suppose we can evaluate the slope or derivative of y and put

$\Delta x = -\alpha(dy/dx)$, where $\alpha > 0$ and is small enough to ensure that $\delta y \approx \Delta y$.

Then, substituting this is in (*), we get

$$dy \approx -\alpha(dy/dx)^2 \qquad (**)$$

The quantity $(dy/dx)^2$ is positive.
Hence, the quantity $-\alpha(dy/dx)^2$ must be negative.

$\therefore \delta y < 0$.

i.e., we have "*traveled down*" the curve towards the minimal point.

If we keep repeating steps such as (**), then we should approach the value x_0 associated with the function minimum.

This is **Gradient Descent**.

Its effectiveness hinges on the ability to calculate or make estimates of dy/dx.

Functions of More Than One Variable

Suppose $y = y(x_1, x_2, \ldots, x_n)$.

One may speak of the slope of the function, or its rate of change, with respect to each of these variables independently.

The slope or derivative of a function y with respect to the variable x_i is:

$\delta y / \delta x_i$ The partial derivative.

The equivalent is then

$\Delta x_i = -\alpha(\delta y / \delta x_i)$.

There is an equation like this for each variable, and all of them must be used to ensure that $\delta y < 0$ and there is gradient descent.

Gradient Descent on an Error.

- Consider a network consisting of a single TLU.
- Assume *supervised learning*.
  i.e., for every input pattern, *p*, in the training set there is a corresponding target t^p.
- The augmented weight vector, $\vec{w}$, completely characterizes the behavior of the network.
- Any function, E, that expresses the discrepency between desired and actual network output, may be considered as a *function of the weights*.
  i.e.,
  $E = E(w_1, w_2, w_{n+1})$.

The optimal weight vector is found by minimizing this function E by gradient descent.

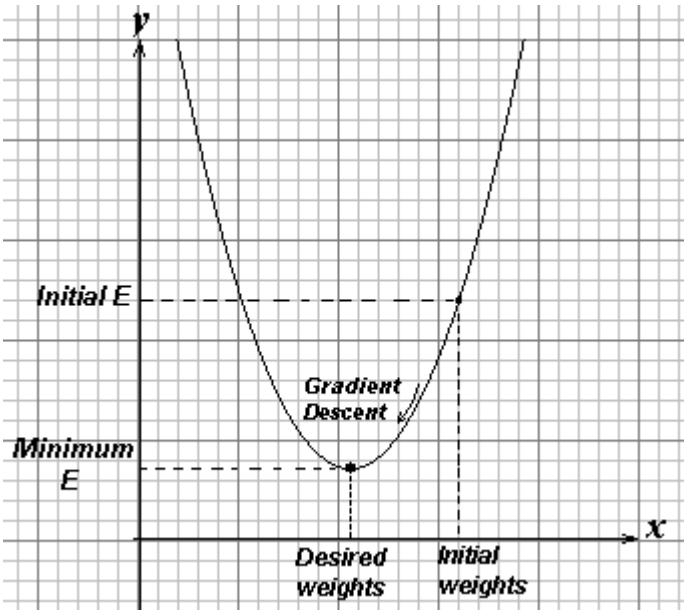
$\Delta w_i = -\alpha(\delta E / \delta w_i)$.

We need to find a suitable error E.

Suppose we assign *equal importance* to the error for each pattern, so that if e^p is the error for training pattern *p*, then the total error E is just the average or mean over all N patterns.

$$E = 1/N \{ \sum_{p=1}^{N} e^p \}.$$

One attempt to define e^p as simply the difference $e^p = t^p - y^p$, where y^p is the TLU output in response to p.



However ... then the error is smaller for $t^{p=0}$, $y^{p=1}$ than for $t^{p=1}$, $y^{p=0}$. *They're equally wrong*.

We next try

$e^p = (t^p - y^p)^2$.

- A subtle problem remains:
  With gradient descent, it is assumed that the function to be minimized depends on its variables in

a smooth, continuous fashion.
First, the activation a^p is simply the weighted sum of inputs. This is smooth and continuous.
But, the output depends on a^p via the discontinuous step function.

- One remedy:
  $e^p = (t^p - a^p)^2$.

We must be careful how we define the targets.
We have used {0,1} heretofore.

When using the augmented weight vector, the output changes as the activation changes sign i.e.,

$a >= 0 \Rightarrow y = 1$.

- $\therefore$ As long as activation takes on the correct sign, the target output is guaranteed and we are free to choose two arbitrary numbers, one positive, and one negative, as the activation targets.

{1, -1} are customary.

- One last modification:
  A factor of 1/2 is added to the error expression - simplifies the resulting slope or derivative.

$\therefore e^p = 1/2 (t^p - a^p)^2$.

and thus,

$$E = \frac{1}{N} \sum_{p=1}^{N} \frac{1}{2} (t^p - a^p)^2 .$$

The Delta Rule.

- The Error E depends on all the patterns. So do all its derivatives. Hence ,the whole training set needs to be presented in order to evaluate the gradients $\delta E / \delta w_i$
- This is **batch training** - results in *true* gradient descent, but is computationally intensive.
- Instead ... adapt the weights based on the presentation of each pattern individually.

i.e., we present the net with a pattern *p*,
evaluate $\delta e^p / \delta w_i$,
and use this as an estimate of the true gradient $\delta E / \delta w_i$

- Recall that:
  $e^p = 1/2 (t^p - a^p)^2$

and

$a^p = w_1 x_1^p + w_2 x_2^p + ... + w_{n+1} x_{n+1}^p$

$\delta e^p / \delta w_i = -(t^p - a^p) x_i^p$, where x_i^p is the i^{th} component of pattern *p*.

1. The gradient must depend in some way on ($t^p - a^p$). The larger this is, the larger we expect the gradient to be.
   If this difference is zero, then the gradient is also zero, since we have found the minimum value of e^p.
2. The gradient must depend on the input x_i^p, for if this is zero, then the i^{th} input is making no contribution to the activation for the p^{th} pattern - and cannot affect the error. No matter how w_i changes, it makes no difference to e^p.

Conversely, if x_i^p is large, then the i^{th} input is correspondingly sensitive to the value of w_i.

$$\delta e^p / \delta w_i = -(t^p - a^p)x_i^p$$

use as an estimate ,

$$\Delta w_i = -\alpha(\delta E / \delta w_i)$$

we obtain,

$$\Delta w_i = -\alpha(t^p - a^p) x_i^p$$

- Pattern Training Regime: weight changes are made after each vector presentation.
- We are using estimates for the true gradient. The progress in the minimization of E is noisy. i.e., weight changes are sometimes made which increase E.
- This is the **Widrow-Hoff Rule**, now refered to as the **Delta Rule** (or δ-rule.)
- Widrow and Hoff first proposed this training regime (1960.) They trained ADALINES (ADAptive LINear ElementS,) which is a TLU, except that the input and output signals were *bipolar* (i.e., {-1,1}.)
- If the learning rate α is sufficiently small, then the delta rule *converges*.
  I.e., the weight vector approaches the vector w_0, for which the error is a minimum, and E itself approaches a constant value.
- Note: A solution will *not* exist if the problem is not linearly separable.
- Then w_0 is the best the TLU can do, and some patterns will be incorrectly classified.
- (Note the difference with the Perceptron rule !!!)
- Also note, delta rule will always make changes to weights, no matter how small (because target activation values ± 1 will never be attained exactly.)

The Delta Rule Algorithm

```
Begin
Repeat
       For each training vector pair (V, t)
              Evaluate the activation a when V is input to the TLU
              Adjust each of the weights
       End For
Until the rate of change of the error is sufficiently small
End
```

The Delta Rule - An Example

Train a two-input TLU with initial weights (0, 0.4) and threshold 0.3, using a learning rate α= 0.25. (The AND function)

First Epoch

w_1	w_2	θ	x_1	x_2	a	t	$\alpha\delta$	δw_1	δw_2	
0.00	0.40	0.30	0	0	-0.30	-1.00	-0.17	-0.00	-0.00	(1) 0.17
0.00	0.40	0.48	0	1	-0.08	-1.00	-0.23	-0.00	(2) -0.23	(3) 0.23
0.00	0.17	0.71	1	0	-0.71	-1.00	-0.07	(4) -0.07	-0.00	(5) 0.07
0.07	0.17	0.78	1	1	-0.68	1.00	0.42	(6) 0.42	(7) 0.42	(8) -0.42

After the first epoch, w_1 = 0.35, w_2 = 0.59, θ= 0.36

We employ $\Delta w_i = +\alpha(t^p - a^p)\, x_i^p$.
Note the plus sign before α: Always travel in the opposite direction of gradient.

(1) $\delta\theta$= +0.25(-1.00 - (-0.30))(-1) ⟵-1 is the input to θ.
= -0.25(-0.7) = 0.175 (sign?)!

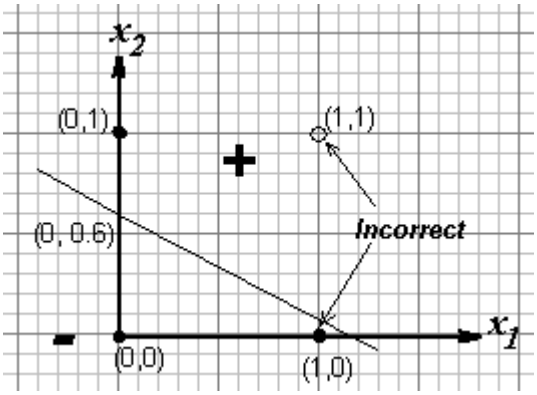
(2) δw_2 = -0.25(-1.00 - (0.08)) * 1
= -0.25(-0.92) = 0.23. ((3) will have the opposite sign.)

(4) δw_1 = -0.25(-1.00 - (-0.71)) * 1
= -0.25(-0.29) = 0.07. ((5) will have the opposite sign.)

(8)$\delta\theta$ = -0.25(1.00 - (-0.68))(-1)
= -0.25 (1.68) = -0.42 ((6), (7) opposite sign.)

Since, after the first epoch, we have
w_1 = 0.35,
w_2 = 0.59,
θ= 0.36 ,

$x_2 = -(0.35 / 0.59)\, x_1 + (0.36 / 0.59) = -0.59\, x_1 + 0.6$
i.e., slope is -0.59.



Second Epoch

w_1	w_2	θ	x_1	x_2	a	t	$\alpha\delta$	δw_1	δw_2	
0.35	0.59	0.36	0	0						
			0	1						
			1	0						
			1	1						