Artificial Neural Networks Lecture Notes

Stephen Lucci, PhD

Part 3

About this file:

- This is the printer-friendly version of the file "*lecture03.htm*".
- 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

Threshold Logic Units (Cont'd)

- **Recurrent Networks**
  - Previously, we spoke of feed-forward netowrks. By Contrast,
  - **Recurrent Networks** are those whose partial computations are recycled through the network itself.
  - See. figure 2.21 in Rojas.
  - Assumption: Computation of the activation of each unit consumes one time unit, Δt.
  - In fig. 2.21, The network takes a string in binary as input, and turns any sequence '11' into '10'. That is, any two consecutive ones are transformed into the sequence 10.

For example: S = 00110110

R = 00100100

- **The Mealy Model of Finite Automata**
  - Refer to fig. 2.23 in the textbook.
  - The finite automaton may be formally described as a structure $M_t = <Q, S, R, f, g, q_I>$ where,

Q:	is the set of finite states of the FA
S	is the Input alphabet
R	is the output alphabet
f	is the next-state function $f: Q \times S \rightarrow Q$
g	is the output function $g: Q \times S \rightarrow R$
q_I	is the Initial state of the machine (the state it is in, before receiving input)

  - The machine M_t acts as a time-delay machine, that outputs at time *t*+1 the input given to it at time *t*.
  - We can write: $x \in S^* \rightarrow \boxed{M_t} \rightarrow y \in R^*$.

The Perceptron

On McCulloch-Pitts Units

- McCulloch-Pitts units are similar to logic gates.
- They have no free parameters that can be adjusted.
- Learning can only be implemented by modifying the network topology and thresholds, which is more complex than merely adjusting weights.

The Perceptron

Frank Rosenblatt developed the perceptron model in 1958, which features

- *Numeric weights*, and
- a *special interconnection pattern*.

In the original model

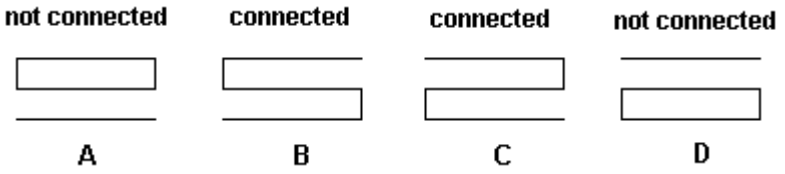
- Units are threshold elements,
- Connectivity is determined stochastically.

In the 1960's, Minsky & Papert refined and perfected this model.

See figure 3.1 (Rojas, p. 56), The Classical Perceptron.

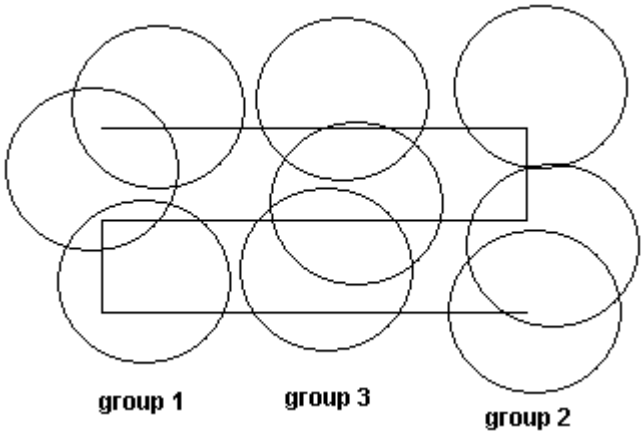
Limitations of the Perceptron
Minsky & Papert studied the limitations of various models of the perceptron.
Example:
Diameter Limited Perceptrons: The receptive field of each predicate has a limited diameter.

Proposition 3.1.1 (Rojas, p. 58)
No diameter limited perceptron can decide whether a geometric figure is connected or not.



Proof By Contradiction
Diameters of receptive fields are limited - hence no single receptive field contains points from both the left and the right ends of the patterns.

Assume that we have three different groups of predicates.



Threshold element performs the computation:
$S = \sum_{Pi \in Group\ 1} w_{1i}P_i + \sum_{Pi \in Group\ 2} w_{2i}P_i + \sum_{Pi \in Group\ 3} w_{3i}P_i - \theta >= 0$.
If S >= 0, then figure is recognized as connected.
If S < 0, then figure is recognized as disconnected.

- If A changed to B - weights in group 2 must increase.
- If A changed to C - weights in group 1 must increase.

However - What if A changed to D?
D is indistinguishable from C in group 1, and
D is indistinguishable from B in group 2
∴D would be categorized as connected which it is *not*.

Trainging TLU's: The Perceptron Rule

- In order for a TLU to perform a given classification it must have the desired decision surface.
- This is determined by the weight vector and the threshold.

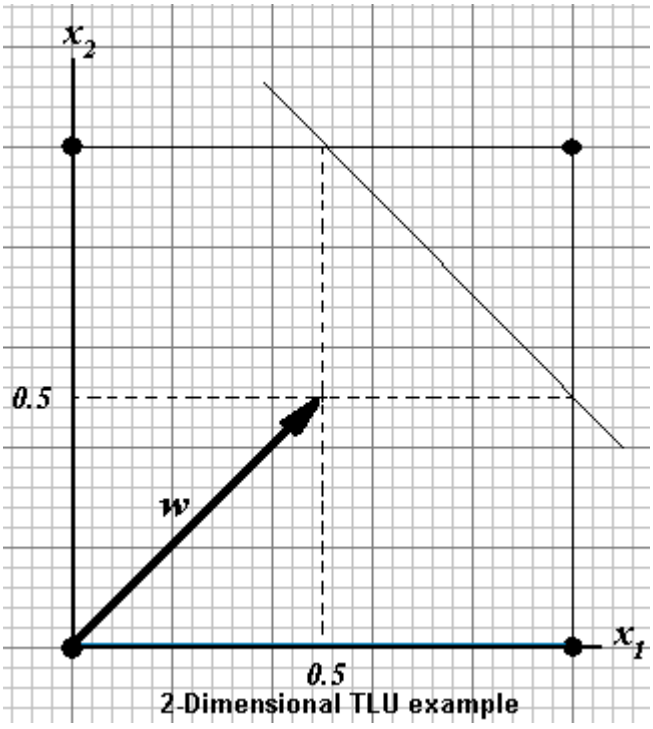
- Hence, it is *necessary to adjust the weights and threshold*. This is usually done via an iterative procedure.
- At each presentation, small changes are made to weights and thresholds.
- This process is known as **training the net**, and the set of examples is the **training set**.
- From the network's viewpoint, it *undergoes a process of learning*, or *adapting to the training set*, and the *prescription* for how to change the weights at each step is **the learning rule**.

$\overline{w}\cdot\overline{x} >= \theta$	Condition for an output of 1. i.e. :
$\overline{w}\cdot\overline{x} - \theta >= 0$	or:
$\overline{w}\cdot\overline{x} + (-\theta) >= 0$	we may think of the threshold as an extra weight tied to an input set to -1.

- **Bias**: The negative of the threshold.
- **Weight Vector**: Originally of dimension *n*
- **Augmented Weight Vector**: (n+1)-dimensional vector, $w_1, w_2, w_3, \ldots, w_n, \theta$
- Thus,

$\overline{w}\cdot\overline{x} >= 0$	$\Rightarrow$y = 1
$\overline{w}\cdot\overline{x} < 0$	$\Rightarrow$y = 0

- A two-input TLU that outputs a "1" with input (1,1), and "0" for all other inputs:



Suitable (non-augmented) weight vector (1/2, 1/2) with threshold 3/4.

The decision line goes through the points x_1 = (1/2, 1) and x_2 =(1, 1/2)

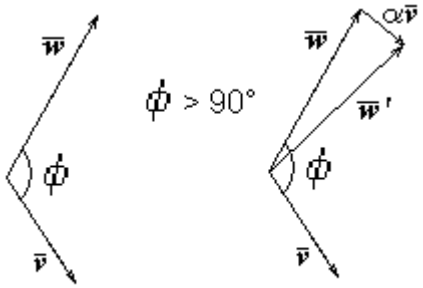
Since $w \cdot x_1 = w \cdot x_2 = 3/4 = \theta$.

Adjusting The Weight Vector

- Training a TLU with augmented weight vector training set: $\bar{v}$(the input), t (the target).
- Input vector $\bar{v}$presented to TLU, where the desired response, or target, is t = 1.
- However, it produces an output y = 0).
- TLU has thus **misclassified** - and we must adjust the weights!
- To produce a "0", the activation must have been negative when it should have been positive.

  $\therefore$the dot product $\bar{w}\cdot\bar{v}$was negative, and the two vectors were pointing away from each other.
- Thus, we need to rotate $\bar{w}$so that it points more in the direction of $\bar{v}$. How do we do this?
- We add a fraction of $\bar{v}$to $\bar{w}$(so as not to upset previous learning).
- i.e.,

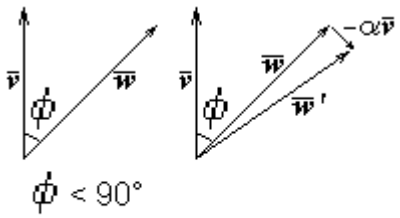
  $\bar{w}' = \bar{w} + \alpha\bar{v}$. **(*)**



TLU misclassification 1 - 0

- suppose now that target t = 0, but y = 1
- This means the activation was positive when it should have been negative.
- Thus, we need to rotate $\bar{w}$away from $\bar{v}$, which we can accomplish by substracting a fraction of $\bar{v}$from $\bar{w}$.
- Thus, we have

  $\bar{w}' = \bar{w} - \alpha\bar{v}$ **(**)**



TLU misclassification 0 -1

- Combining **(*)** and **(**)** we obtain

  $\bar{w}' = \bar{w} + \alpha(t - y)\bar{v}$. **(***)**

  or

  $\Delta\bar{w} = \bar{w}' - \bar{w}$

  $\Delta\bar{w} = \alpha(t - y)\bar{v}$

  or, in terms of components,

  $\Delta w_i = \alpha(t - y)v_i$, i = 1 to n+1, where $w_{n+1} = \theta$and $v_{n+1} = -1$ (*always*.)
- This is called the ***Perceptron Learning Rule*** because historically it was first used with perceptrons.

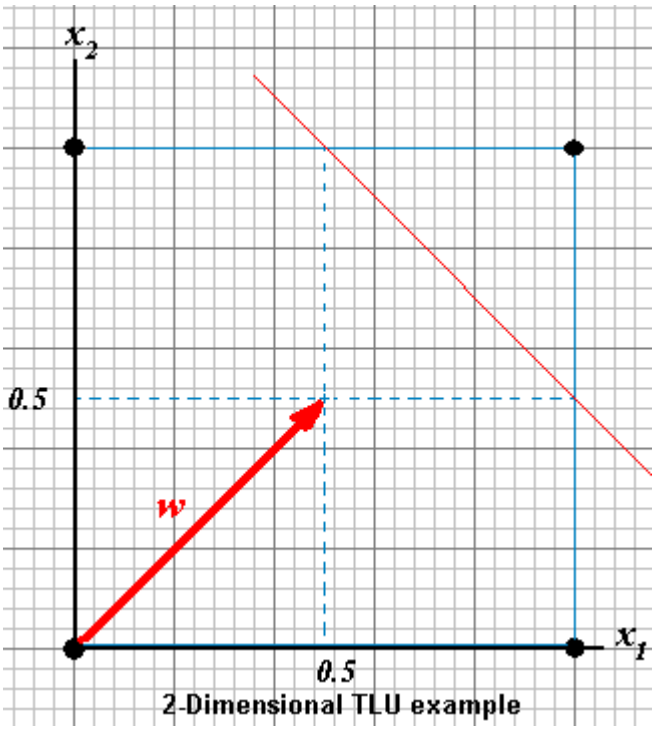
The Perceptron Learning Rule

```
Begin
Repeat
        For each training vector pair (V, t)
                Evaluate the output y when V is input to the TLU
                If y != t Then
                        Form a new weight vector W' according to (***)
                Else
                        Do nothing
                End If
        End For
Until y = t for all vectors
End
```

- The algorithm will generate a valid weight vector for the problem at hand, if one exists.

The Perceptron Convergence Theorem

- ***If two classes of vectors, X, Y are linearly separable, then application of the perceptron training algorithm will eventually result in a weight vector $\overline{w}_0$, such that $\overline{w}_0$ defines a TLU whose decision hyperplane separates X and Y.***
- This theorem was first proven by Rosenblatt in 1962.
- Once $\overline{w}_0$ has been found, it remains stable and no further changes are made to the weights.
- Returning to our previous example,



initial weights are $w_1$0 and $w_2$0.4

initial threshold θ= 0.3.

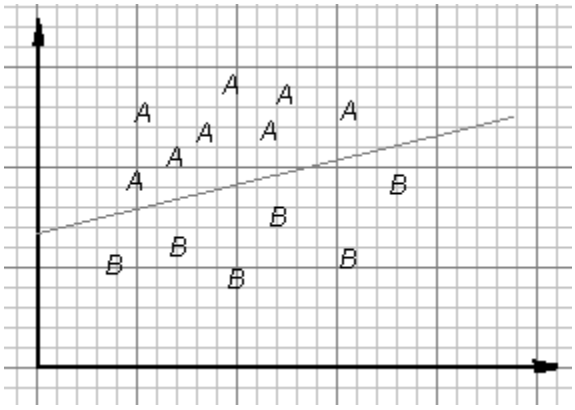
The ***learning rate*** α= 0.25.

- HOMWORK: Write a program to implement the Perceptron Learning Rule and solve this problem:

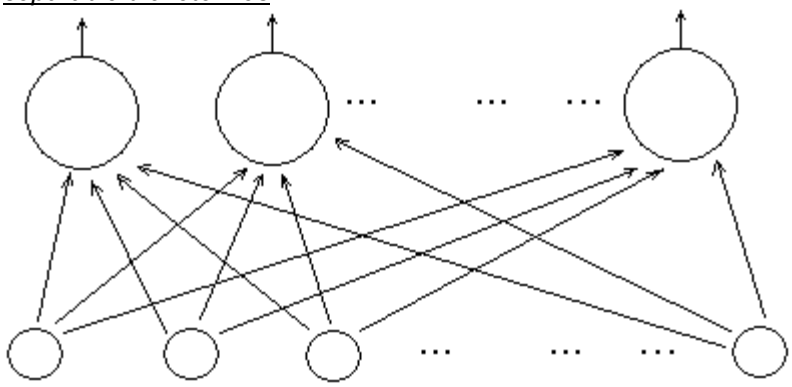
x_1	x_2	w_1	w_2	θ	α	y	t	$\alpha(t-y)$	Δw_1	Δw_2	???
0	0	0.00	0.40	0.30	0.00	0	0	0.00	0	0	0
0	1	0.00	0.40	0.30	0.40	1	0	-0.25	0	-0.25	0.25
1	0	0.00	0.15	0.55	0.00	0	0	0.00	0	0	0
1	1	0.00	0.15	0.55	0.15	0	1	0.25	0.25	0.25	-0.25
0	0	0.25	0.40	0.30	0.00	0	0	0.00	0	0	0
0	1	0.25	0.40	0.30	0.40	1	0	-0.25	0	-0.25	0.25
1	0	0.25	0.15	0.55	0.25	0	0	0.00	0	0	0
1	1	0.25	0.15	0.55	0.40	0	1	0.25	0.25	0.25	-0.25
0	0	0.50	0.40	0.30	0.00	0	0	0.00	0	0	0
0	1	0.50	0.40	0.30	0.40	1	0	-0.25	0	-0.25	0.25
1	0	0.50	0.15	0.55	0.50	1	0	-0.25	-0.25	0	0.25
1	1	0.50	0.15	0.70	0.40	0	1	0.25	0.25	0.25	-0.25
0	0	0.50	0.40	0.45	0.00	0	0	0.00	0	0	0
0	1	0.50	0.40	0.45	0.40	0	0	0.00	0	0	0
1	0	0.50	0.40	0.45	0.50	1	0	-0.25	-0.25	0	-0.25
1	1	0.50	0.40	0.70	0.90	0	1				

Single Layer Nets

- We may use a single perceptron or TLU to classify two linearly separable classes A and B.
- Patterns may have many inputs "cartoon" picture or schematic.



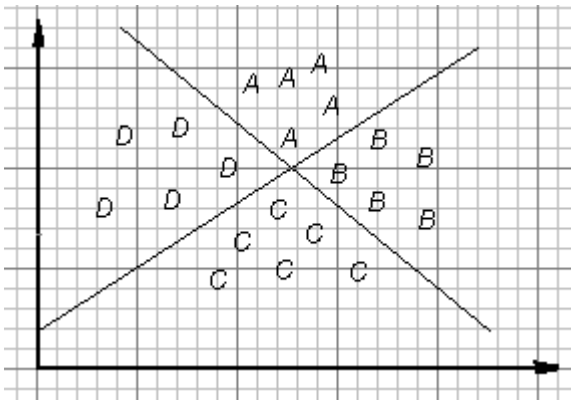
- It is possible to train *multiple nodes* on the input space to obtain a set of *linearly separable dichotomies*



Example:
Classifying handwritten alphabetic characters where 26 dichotomies are required, each one separating one letter class from the rest of the alphabet.
e.g, A's from Not A's; B's from Not B's; etc.

Non-Linearly Separable Classes

- Four classes, A, B, C and D separated by two planes in pattern space:



Note: that this is a two-dimensional schematic or representation of an n-dimensional space.

- Here, we could not use a single layer net!
  Class A, for example, is not linearly separable from the others taken together.
- We can, however, "chop" the pattern space into linearly separable regions, and look for particular combinations of overlap within these regions.
- Class of A and B (combined, AB) is linearly separable from C and D together (CD). So are AD and BC.

Class	y_1
AB	1
CD	0

Unit U_1

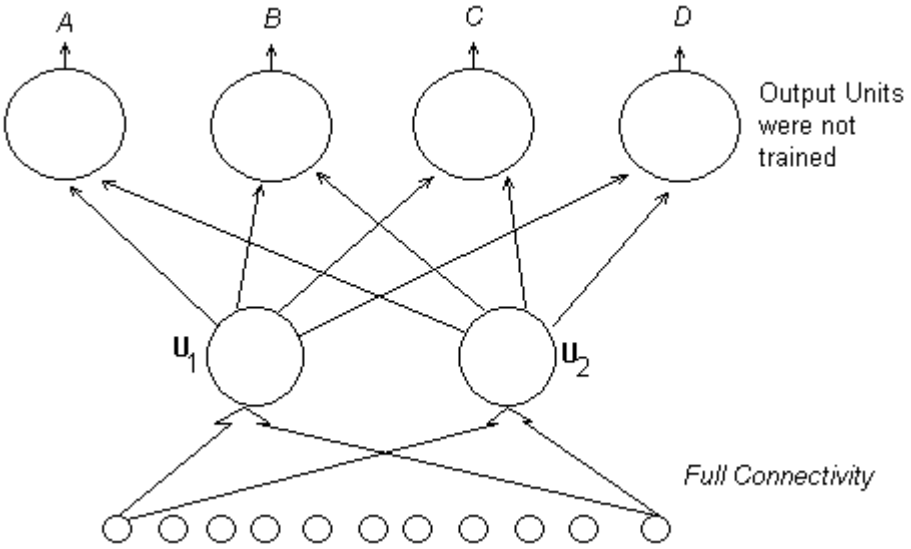
Class	y_2
AD	1
BC	0

Unit U_2

- If a member of class A is input to each of U_1 and U_2, then $y_1 = 1$, $y_2 = 1$.

- Thus, $y_1 = 1$, $y_2 = 1$ iff input is in A.
- We obtain a unique code for each of the boxes:

y_1	y_2	Class
0	0	C
0	1	D
1	0	B
1	1	A



- Note: The less information we have to supply ourselves, the more useful a network will be.
- Here, we required two pieces of information:
  i) The four classes may be separated by two hyperplanes.
  ii) AB was linearly separable from CD and AD was linearly separable from BC.
- We will need a new training algorithm to accomplish this.

HOMEWORK:
Using the perceptron learning rule,
Find the weights required to perform the following classifications:

58. Vectors (1,1,1,1) and (0,1,0,0) are members of the class, and therefore have target value 1.
59. Vectors (1,1,1,0) and (1,0,0,1) are not members of the class, and have target value 0.

Use a learning rate of 1, and starting weights of 0. Using each of the training $\vec{x}$ vectors as input, test the responses of the net.