Lecture 5: “Simple Neural Networks”

Dr. Ehab Essa

Computer Science Department, Mansoura University

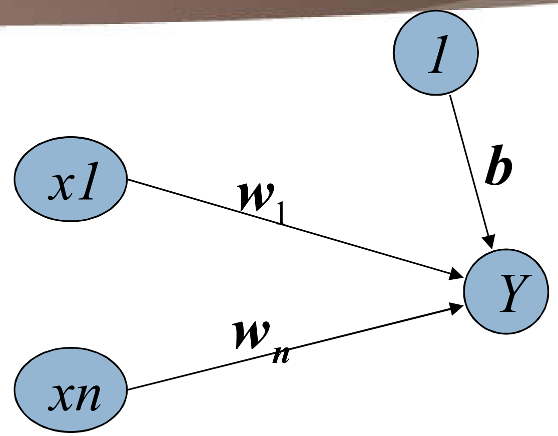
Today

- Hebb nets
- Perceptron
- Adaline

- Lecture Notes:
  - **Chapter 2**, Fundamentals of Neural Networks: Architectures, Algorithms, and Applications by Laurene Fausett

General architecture

Single layer



net input to *Y*: $$y\_in = b + \sum_{i=1}^{n} x_i w_i$$

bias ***b*** is treated as the weight from a special unit with constant output 1. threshold $\boldsymbol{\theta}$ related to *Y*

output $$y = f(y\_in) = \begin{cases} 1 & \text{if } y\_\text{in} \geq \theta \\ -1 & \text{if } y\_\text{in} < \theta \end{cases}$$

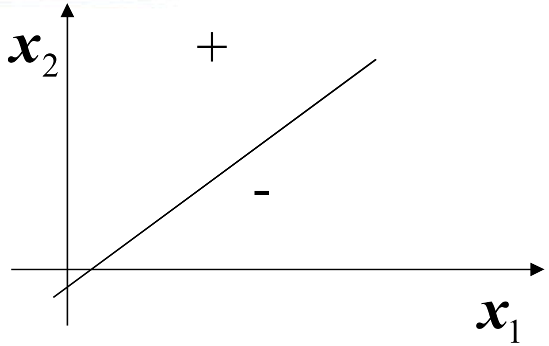
classify $(\boldsymbol{x}_1, \ldots\ldots \boldsymbol{x}_n)$ into one of the two classes

Decision region/boundary

n = 2, b != 0, θ = 0

$$\boldsymbol{b} + \boldsymbol{x}_1\boldsymbol{w}_1 + \boldsymbol{x}_2\boldsymbol{w}_2 = 0 \text{ or}$$

$$\boldsymbol{x}_2 = -\frac{\boldsymbol{w}_1}{\boldsymbol{w}_2}\boldsymbol{x}_1 - \frac{\boldsymbol{b}}{\boldsymbol{w}_2}$$



is a line, called *decision boundary*, which partitions the plane into two decision regions

If a point/pattern $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is in the positive region, then

$\boldsymbol{b} + \boldsymbol{x}_1\boldsymbol{w}_1 + \boldsymbol{x}_2\boldsymbol{w}_2 \geq 0$, and the output is one (belongs to class one)

Otherwise, $\boldsymbol{b} + \boldsymbol{x}_1\boldsymbol{w}_1 + \boldsymbol{x}_2\boldsymbol{w}_2 < 0$, output –1 (belongs to class two)

n = 2, b = 0, θ != 0 would result a similar partition

Hebb Nets

- Hebb, in his influential book *The organization of Behavior* (1949), claimed
  - Behavior changes are primarily due to the changes of synaptic strengths (w_{ij}) between neurons I and j
  - w_{ij} increases only when both I and j are "on": the **Hebbian learning law**
  - In ANN, Hebbian law can be stated: w_{ij} increases only if the outputs of both units x_i and y_j have the same sign.
  - In our simple network (one output and n input units)

$$w_{ij}(new) = w_{ij}(old) + x_i y$$

Hebb Nets

- Hebb net (supervised) learning algorithm (p.49)

```
Step 0.  Initialization: b = 0, w_i = 0, i = 1 to n
Step 1.  For each of the training sample s:t do steps 2 -4
         /* s is the input pattern, t the target output of the sample */
Step 2.      x_i := s_i , (i = 1 to n)             /* set s to input units */
Step 3.      y := t                                /* set y to the target */
Step 4.      w_i := w_i + x_i * y, (i = 1 to n)     /* update weight */
             b := b + y                            /* update bias */
```

Notes:
1) each training sample is used only once.

2) Weight change can be written $\Delta w_{ij} = x_i y$

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

- Activation function If binary data : binary step function If bipolar data : bipolar sign function

Examples: AND function

- Hebb net for **and function** : binary input and output

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1						
(1,0,1)	0						
(0,1,1)	0						
(0,0,1)	0						

bias unit

Examples: AND function

$$\Delta w_{ij} = x_i y$$

$$w_{ij}(new) = w_{ij}(old) + \Delta w_{ij}$$

- Hebb net for **and function** : binary input and output

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1	1	1	1	1	1	1
(1,0,1)	0	0	0	0	1	1	1
(0,1,1)	0	0	0	0	1	1	1
(0,0,1)	0	0	0	0	1	1	1

bias unit

Examples: AND function

- Is it correctly learned after using each sample once?

- **Testing second pattern**
- (1,0) : x1=1 ,,, x 2 = 0 ,,, w1 =1 ,, w2 = 1 ,, b =1 ,, $\theta = 0$
- X1*w1 + x2*w2 + b = 1 + 0 + 1 = 2 $> \theta$ $\rightarrow$ output = 1 which is wrong

Example 2: AND function

- Bipolar units (1, -1)

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1	1	1	1	1	1	1
(1,-1,1)	-1	-1	1	-1	0	2	0
(-1,1,1)	-1	1	-1	-1	1	1	-1
(-1,-1,1)	-1	1	1	-1	2	2	-2

Testing (1,1) : x1=1 ,,, x 2 = 1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$
X1w1 + x2w2 + b = 2 + 2 - 2 = 2 > θ → output = 1

Testing (1,-1) : x1=1 ,,, x 2 = -1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$

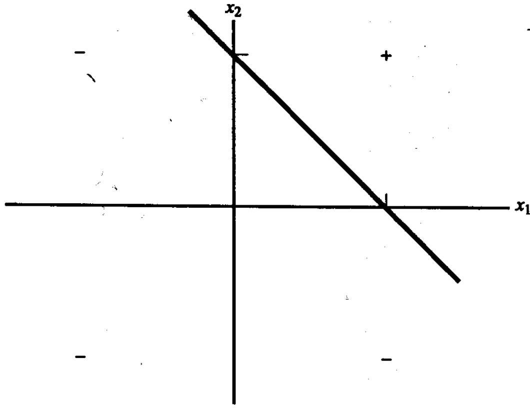
X1w1 + x2w2 + b = 2 - 2 - 2 = -2 < θ → output = -1

Example 2: AND function

- Bipolar units (1, -1)

Input	target	Weight changes			weights		
(x1,x2,1)	Y=t	Δw1	Δw2	Δb	w1	w2	b
					0	0	0
(1,1,1)	1	1	1	1	1	1	1
(1,-1,1)	-1	-1	1	-1	0	2	0
(-1,1,1)	-1	1	-1	-1	1	1	-1
(-1,-1,1)	-1	1	1	-1	2	2	-2

A correct boundary
$-1 + x1 + x2 = 0$
is successfully learned



Testing (-1,1) : x1=-1 ,,, x 2 = 1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$
X1w1 + x2w2 + b = -2 + 2 - 2 = -2 < θ → output = -1

Testing (-1,-1) : x1=-1 ,,, x 2 = -1 ,,, w1 =2 ,, w2 = 2 ,, b =-2 ,, $\theta = 0$

X1w1 + x2w2 + b = -2 -2 - 2 = -6 < θ → output = -1

Examples: Character Recognition

- Convert the patterns to input vectors by replace each # with 1 and . with -1

```
# . . . #        . # # # .
. # . # .        # . . . #
. . # . .   and  # . . . #
. # . # .        # . . . #
# . . . #        . # # # .
```

Pattern 1 **Pattern 2**

ect. **Pattern 1 then becomes**

1 −1 −1 −1 1, −1 1 −1 1 −1, −1 −1 1 −1 −1, −1 1 −1 1 −1,
1 −1 −1 −1 1,

and pattern 2 becomes

−1 1 1 1 −1, 1 −1 −1 −1 1, 1 −1 −1 −1 1, 1 −1 −1 −1 1, −1 1 1 1 −1,

- The correct response for the first pattern is +1 and the second pattern is -1

Hebb Nets

- It will fail to learn x1 ^ x2 ^ x3, even though the function is linearly separable.

- Stronger learning methods are needed.
  - **Error driven**: for each sample s:t, compute y from s based on current W and b, then compare y and t
  - **Use training samples repeatedly**, and each time only change weights slightly (a << 1)
  - Learning methods of Perceptron and Adaline are good examples

Perceptrons

- By Rosenblatt (1958,1962)
  - Three layers of units: ***S**ensory,* ***A**ssociation,* and ***R**esponse*
  - Learning occurs only on weights from ***A*** units to ***R*** units (weights from ***S*** units to ***A*** units are fixed).
  - Simple perceptron used binary or bipolar activations for the sensory and associator units and activation of +1, 0 or -1 for response unit:

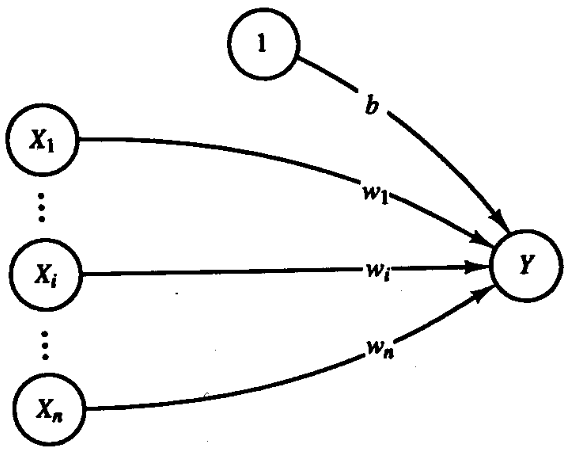
$$f(y\_in) = \begin{cases} 1 & \text{if } y\_in > \theta \\ 0 & \text{if } -\theta \le y\_in \le \theta \\ -1 & \text{if } y\_in < -\theta \end{cases}$$

  - For a given training sample s:t, change weights only if the computed output y is different from the target output t (thus **error driven**)

Architecture for simple Perceptron

- The output from the associator units was binary.

- Since only the weights from the associator units to the output unit could be adjusted, we limit our consideration to the single layer net.



Perceptron learning algorithm (p.62)

Step 0. Initialization: $b = 0$, $w_i = 0$, $i = 1$ to n
Step 1. While stop condition is false do steps 2-5

Step 2. For each of the training sample s:t do steps 3 -5
Step 3. $xi := s_i$, $i = 1$ to n
Step 4. compute y

$$y\_in = b + \sum xi\, wi$$

$$y = \begin{cases} 1 & , \quad y\_in > \theta \\ 0 & , \quad -\theta <= y\_in <= \theta \\ -1 & , \quad y\_in < -\theta \end{cases}$$

Step 5. If error occurs (y != t) update weight

$$w_i := w_i + \alpha * x_i * t, \qquad i = 1 \text{ to } n$$

$$b := b + \alpha * t$$

Perceptrons

- Notes:
  - Learning occurs only when a sample has y != t
  - Two loops, a completion of the inner loop (each sample is used once) is called an epoch
- Stop condition
  - When no weight is changed in the current epoch, or
  - When pre-determined number of epochs is reached

Examples: AND function

$w_i := w_i + \alpha * x_i * t$

- Example **and function** : **binary input and bipolar target $\alpha = 1$, $\theta = 0.2$**
- **First epoch**

Input	y_in	y	target	Weight changes			weights		
(x1,x2,1)			t	Δw1	Δw2	Δb	w1	w2	b
							0	0	0
(1,1,1)	0	0	1	1	1	1	1	1	1
(1,0,1)	2	1	-1	-1	0	-1	0	1	0
(0,1,1)	1	1	-1	0	-1	-1	0	0	-1
(0,0,1)	-1	-1	-1	0	0	0	0	0	-1

Examples: AND function

- **Second epoch**

Input	y_in	y	target	Weight changes			weights		
(x1,x2,1)			Y=t	Δw1	Δw2	Δb	w1	w2	b
							0	0	-1
(1,1,1)	-1	-1	1	1	1	1	1	1	0
(1,0,1)	1	1	-1	-1	0	-1	0	1	-1
(0,1,1)	0	0	-1	0	-1	-1	0	0	-2
(0,0,1)	-2	-1	-1	0	0	0	0	0	-2

- **Tenth epoch**

Input	y_in	y	target	Weight changes			weights		
(x1,x2,1)			Y=t	Δw1	Δw2	Δb	w1	w2	b
							2	3	-4
(1,1,1)	1	1	1	0	0	0	2	3	-4
(1,0,1)	-2	-1	-1	0	0	0	2	3	-4
(0,1,1)	-1	-1	-1	0	0	0	2	3	-4
(0,0,1)	-4	-1	-1	0	0	0	2	3	-4

Adaline(**Ada**ptive **Line**ar Neuron)

- By Widrow and Hoff (1960)
  - The same architecture of our simple network
  - *Usually uses bipolar activations for its input and output target.*
  - During training, the activation of the unit is its net input, i.e., the activation function is the identity function.
  - Learning method: **delta rule** (another way of error driven), also called Widrow-Hoff learning rule
    - $b := b + \alpha * (t - y\_in)$
    - $w_i := w_i + \alpha * (t - y\_in) * x_i$
  - Delta rule is a consequence of trying to reduce the squared error of an arbitrary training pattern.

Derivation of the delta rule

- Error for all P training samples: mean square error

$$E = \frac{1}{P}\sum_{p=1}^{P}(t(p) - y\_in(p))^2$$

E is a function of W = {w1, ... wn}

- Learning takes **gradient descent** approach to reduce E by modify W
  - the gradient of E: $\nabla E = (\frac{\partial E}{\partial w_1}, \ldots\ldots \frac{\partial E}{\partial w_n})$
  - $\Delta w_i \propto -\frac{\partial E}{\partial w_i}$

$$\frac{d}{dx}[f(x)]^2 = 2f(x)\frac{d}{dx}f(x)$$

  - $\frac{\partial E}{\partial w_i} = [\frac{2}{P}\sum_{p=1}^{P}(t(p) - y\_in(p))]\frac{\partial}{\partial w_i}(t(p) - y\_in(p)$

$$= -[\frac{2}{P}\sum_{p=1}^{P}(t(p) - y\_in(p))]x_i$$

  - There for $\Delta w_i \propto -\frac{\partial E}{\partial w_i} \quad = [\frac{2}{P}\sum_{1}^{P}(t(p) - y\_in(p))]x_i$

Adaline algorithm

Step 0. Initialize weights.
(Small random values are usually used.)
Set learning rate α.
(See comments following algorithm.)

Step 1. While stopping condition is false, do Steps 2–6.

Step 2. For each bipolar training pair s:t, do Steps 3–5.

Step 3. Set activations of input units, $i = 1, \ldots, n$:

$$x_i = s_i.$$

Step 4. Compute net input to output unit:

$$y\_in = b + \sum_i x_i w_i.$$

Step 5. Update bias and weights, $i = 1, \ldots, n$:

$$b(\text{new}) = b(\text{old}) + \alpha(t - y\_in).$$

$$w_i(\text{new}) = w_i(\text{old}) + \alpha(t - y\_in)x_i.$$

Step 6. Test for stopping condition:
If the largest weight change that occurred in Step 2 is smaller than a specified tolerance, then stop; otherwise continue.

MADALine [1960,1987]

- MADALine consists of many Adaline neurons arranged in a multilayer net.
- There are two training algorithms (MR and MRII).
  - In MR, only weights for the hidden Adalines (Z_1 & Z_2) are adjusted, the weights for the output unit are fixed.
  - In MRII, all the weights in the net are adjusted.
- Thresholding activation function is used during training and testing.

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ -1 & \text{if } x < 0. \end{cases}$$