Neural Networks

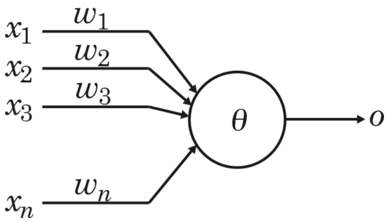
Single Neuron Models

Dr. Petr Musilek

Department of Electrical and Computer Engineering
University of Alberta

Fall 2019

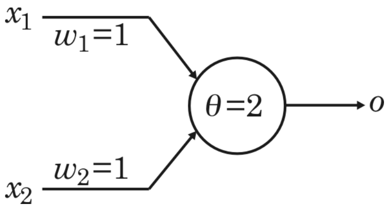
McCulloch-Pitts Neuron Model



$$w_i = \pm 1;\ x_i = \{0, 1\}$$

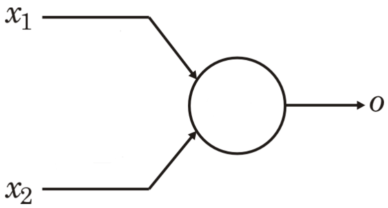
$$o = \begin{cases} 1 & \text{if} \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0 & \text{if} \sum_{i=1}^{n} w_i x_i < \theta. \end{cases}$$

Boolean Logic Implementation: AND Logic Function



x_1	x_2	$\rightarrow$	o
1	1		1
1	0		0
0	1		0
0	0		0

Boolean Logic Implementation: OR Logic Function



x_1	x_2	$\rightarrow$	o
1	1		1
1	0		1
0	1		1
0	0		0

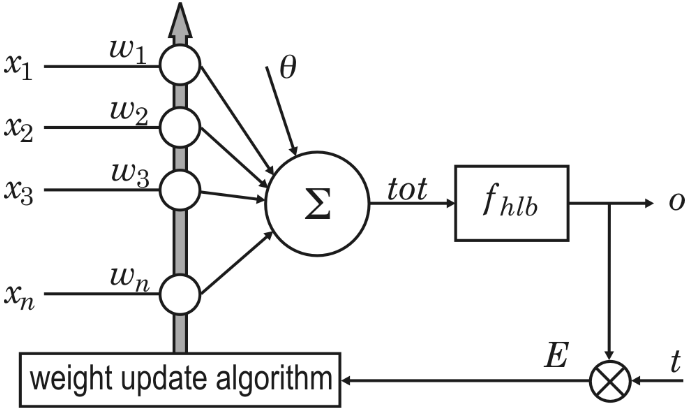
Perceptron

- 1958 (Rosenblatt); training algorithm that provided the first procedure for training a simple ANN
- The simplest form of a neural network - single neuron with adjustable synaptic weights and a hard limiter activation function (based on the McCulloch and Pitts neuron model).

Perceptron

- Operation: The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to $+1$ if its input is positive and -1 if it is negative (sometimes values $[0, 1]$ are used).
- Learning: making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron.
- To allow realization of mappings that require position of hard limit different from 0, a threshold term θ is added.
- Initial weights and threshold are randomly assigned, usually in the range $[-0.5, 0.5]$, and then updated to obtain the output consistent with the training examples.

Perceptron



x_1
x_2
x_3
x_n
w_1
w_2
w_3
w_n
θ
Σ
tot
f_{hlb}
o
E
t
weight update algorithm

Perceptron: training algorithm

1. Initialization. Set initial weights w_i and threshold θ to small random numbers (e.g. in the range $[-0.5, 0.5]$)
2. Activation. From a training set T, select an input pattern $\mathbf{x}(k) \in T$. Apply it to activate the perceptron and calculate the actual output $o(k)$

$$o(k) = f_{hlb}\left(\sum_{i=1}^{m} w_i x_i(k) - \theta\right),$$

   where m is the number of the perceptron inputs (equal to the dimension of input patterns), and f_{hlb} is the bipolar step activation function.
3. Determine error using the desired output (target) $t(k) \in T$:

$$E(k) = t(k) - o(k)$$

Perceptron: training algorithm

4. Weight update. Update the weights according to the following rule
   $w_i^{\text{new}} = w_i^{\text{old}} + \Delta w_i$
   $$\begin{array}{ll} \text{If } E(k) > 0: & \text{increase perceptron output } o(k), \\ \text{If } E(k) < 0: & \text{decrease } o(k), \end{array}$$
   i.e. $\Delta w_i(k) = \eta \cdot x_i(k) \cdot E(k)$
5. Iteration.Steps 2-4 compose an *iteration*. Perform one iteration for each training pattern $k = 1, \ldots, n$.
6. Repetition. One set of updates of all the weights for all the training patterns is called one *epoch* of training. Repeat the process until convergence.

Perceptron for Classification

- Can be used for binary classification.
- Given training examples of classes C_1, C_2 we train the perceptron in such a way that it classifies correctly the training examples:
  - If the output is +1 then the input is assigned to C_1
  - If the output is -1 then the input is assigned to C_2

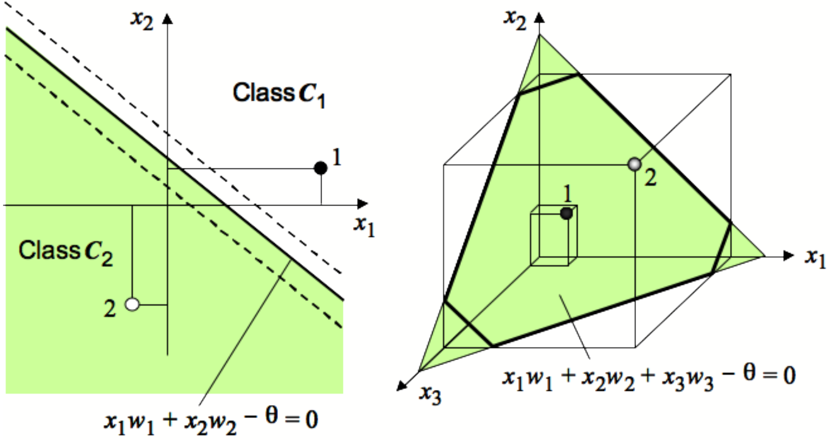
Perceptron for Classification – Learning

Numerically: finding suitable values for the weights in such a way that the training examples are correctly classified.

Geometrically: finding a hyper-plane that separates the examples of the two classes.

$$\sum_{i=1}^{m} w_i x_i - \theta = 0$$

Perceptron - Geometric View

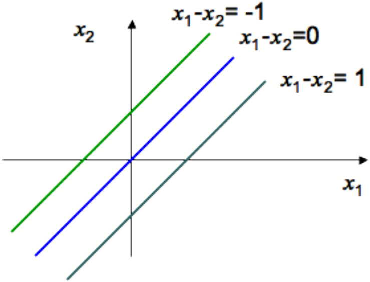


Bias of a Neuron

The bias θ has the effect of applying affine transformation to the weighted sum

$$tot = \sum_{i=1}^{m} w_i x_i - \theta,$$

tot is sometimes called *net input* to the neuron



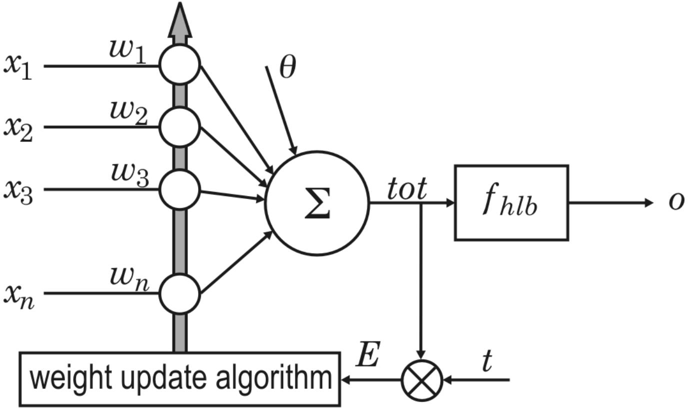
Note

m inputs, $m+1$ weights to incorporate bias (variable threshold)

i.e. augmented $\mathbf{x} = [1, x_1, \ldots, x_m]$, and $\mathbf{w} = [\theta, w_1, \ldots, w_m]$.

ADLINE

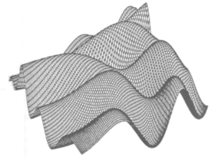
Widrow and Hoff, 1960s



Difference compared to perceptron learning?

ADALINE Learning: LMS and Gradient Descent

- LMS (Least Mean Squares): form of gradient descent through weight space
- Waves in the ocean: How to find the bottom of a wave using only local info?



ADLINE Learning

Gradient Descent

Determine error: $E = \sum_{k=1}^{n} (t_k - tot_k)^2$

Move the weights in the negative direction of the gradient

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

Determine error: $E = \sum_{k=1}^{n} (t_k - tot_k)^2$

Move the weights in the negative direction of the gradient

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

Move the weights

How to do this ?

For an individual pattern (particular k)

$$E = (t - tot)^2 = \left(t - \sum_{i=0}^{m} x_i w_i\right)^2$$

differentiating

$$\frac{\partial E}{\partial w_j} = 2\,(t - tot)\,\frac{\partial}{\partial w_j}(-tot) = -2\,(t - tot)\,x_j$$

ADALINE Learning
Moving the weights

Now change the weights in the opposite direction of $\frac{\partial E}{\partial w_j}$

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = -\eta \left(t - tot\right) x_j$$

Now change the weights in the opposite direction of $\frac{\partial E}{\partial w_j}$

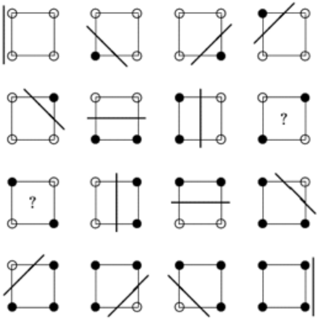
$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j} = -\eta \, (t - tot) \, x_j$$

... and finally

$$w_j^{\mathrm{new}} = w_j^{\mathrm{old}} - \Delta w_j = w_j^{\mathrm{old}} + \eta \, (t - tot) \, x_j$$

- Possible input values $x_i = \pm 1 : d = 2$
- Dimension of inputs $m = 2$
- Number of possible m-element vectors $k = dm = 4$
- Number of possible partitions into two classes ($y = \pm 1$): $2k = 16$

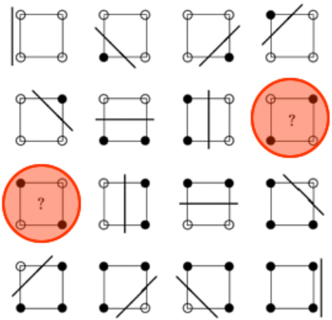
- Possible input values $x_i = \pm 1 : d = 2$
- Dimension of inputs $m = 2$
- Number of possible m-element vectors $k = dm = 4$
- Number of possible partitions into two classes ($y = \pm 1$): $2k = 16$



Linear separability of patterns

Separation of Input Space

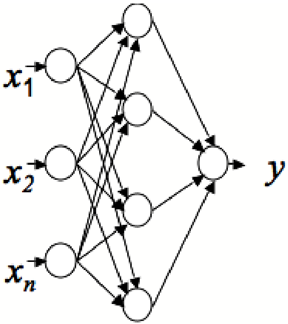
- Possible input values $x_i = \pm 1 : d = 2$
- Dimension of inputs $m = 2$
- Number of possible m-element vectors $k = dm = 4$
- Number of possible partitions into two classes ($y = \pm 1$): $2k = 16$



MADALINE

3-layer network with Many ADALINEs

- ADALINE - only linear decision boundary
- MADALINE - with multiple ADALINEs, a decision boundary can be “carved” into more complex shapes

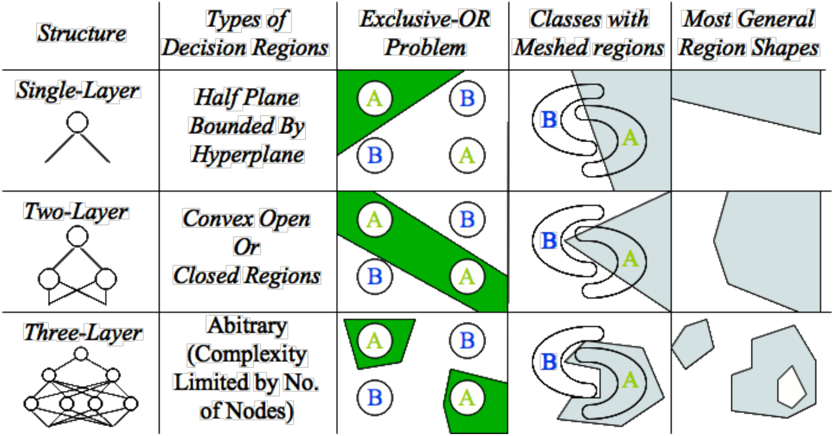


Unfortunately

MADALINEs are hard to teach . . .

ADLINE – MADALINE

Non linearly separable problems



Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer	Half Plane Bounded By Hyperplane	A B B A	B A	
Two-Layer	Convex Open Or Closed Regions	A B B A	B A	
Three-Layer	Abitrary (Complexity Limited by No. of Nodes)	A B B A	B A	