3 REDES CLÁSSICAS – PERCEPTRON E ADALINE

3.1-Redes com funções de ativação de limiar

Uma rede simples de uma camada, consiste em um ou mais neurônios de saída *j* conectados às entradas *i* através de pesos *Wij*. No caso mais simples a rede possui um só neurônio *k*, como indicado na figura 11.

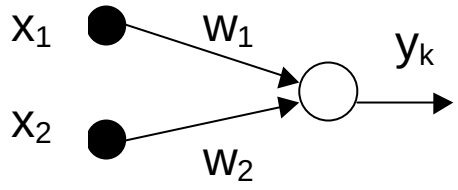


FIG. 16- NEURÔNIO SIMPLES COM DUAS ENTRADAS E UMA SAÍDA

A saída y_k do neurônio, é dada por:

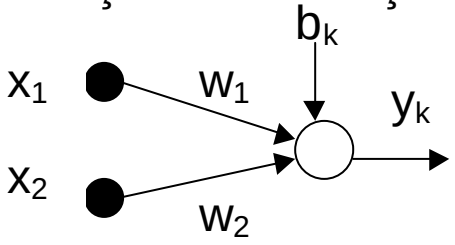
$$y_k = j\left(u_k\right) \quad \text{Onde:} \quad u_k = \sum_{i=1}^{n} w_{ki} x_i$$

A função de ativação j proposta originalmente por McCullock e Pitts, é uma função de limiar, isto é,

$$y_k = j\left(u_k\right) = \begin{cases} 1, se \quad u_k \geq 0 \\ 0, se \quad u_k < 0 \end{cases}$$

Para este caso, o limiar de ativação do neurônio (acima do qual dizemos que o neurônio dispara) é zero, mas o modelo permite estabelecer um limiar de ativação $\boldsymbol{q}$ diferente de zero.

Uma modificação introduzida neste modelo do neurônio foi a criação de um parâmetro polarizador (*bias*) b_k por função de ativação, ou seja, um polarizador por neurônio, cuja função é aumentar ou diminuir a entrada líquida u_k, de forma a transladar a função de ativação no eixo de u_k.



A saída agora é dada por:

$$y_k = j\left(u_k + b_k\right)$$

O polarizador pode ser tratado como “mais um peso”. Para isso basta considerarmos que a entrada do neurônio k é dada por:

$$u_k = \sum_{i=0}^{n} w_{ki} x_i$$ ao invés de: $$u_k = \sum_{i=1}^{n} w_{ki} x_i$$

Voltamos, então, a usar: $$y_k = j\left(u_k\right)$$

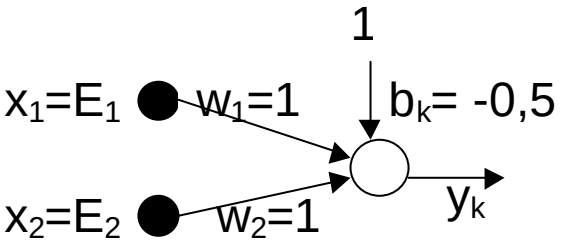
Tudo o que fizemos foi acrescentar uma nova entrada do tipo $x_0 = 1$ com um peso associado $w_{k0} = b_k$

Este neurônio pode ser empregado para separar em classes distintas os padrões de entradas. Se a entrada líquida for maior que o limiar, o padrão dessa entrada pertence à classe 1, caso contrário, pertence à classe 0. Portanto, podemos usa-lo para emular o comportamento de uma porta lógica.

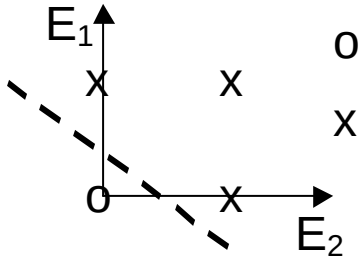
Por exemplo, para modelarmos o comportamento de uma porta porta lógica “ou”, cuja tabela verdade é:

Entrada E_1	Entrada E_2	Saída	
0	0	0	} Classe 0
0	1	1	} Classe 1
1	0	1	
1	1	1	

Poderíamos usar uma rede neural com um neurônio de duas entradas e uma saída, como na figura ao lado.



A rede de fato modela uma função discriminante linear que separa as classes do problema, como mostrado abaixo.



o - Classe 0

x - Classe 1

A separação entre as duas classes é dada pela equação:

E1w1 + E2w2 + b = 0

Que é a equação de uma família de retas que podem separar as classes.

Bem, mas como foram encontrados os valores dos pesos e do bias para o caso acima? Ou seja, como treinar a rede para encontrar esses valores? A seguir veremos dois métodos de treinamento.

3.2-Regra de treinamento de Hebb

Os algoritmos de treinamento basicamente calculam a cada passo um novo valor para os pesos e para os bias dos nós (que, como vimos, podem também ser considerados como pesos). Ou seja: $w_{novo} = w_{antigo} + \Delta w$

Hebb propos uma regra de cálculo para cada Δw que é proporcional à entrada associada e uma função de transferência que é ligeiramente diferente da função de limiar vista anteriormente. Esta função é conhecida como função de limiar bipolar:

$$y_k = j(u_k) = \begin{cases} 1, se \quad u_k \geq 0 \\ -1, se \quad u_k < 0 \end{cases}$$

O algoritmo de treinamento é o seguinte:

1. Inicie todos os pesos e bias com zero
2. Para cada padrão (entrada e saída desejada) faça:
   2.1 Para cada conexão faça: (d é a saída desejada do nó)
      2.1.1 $\Delta w_i = x_i * d$; (considerar $w_0 = b$, $x_0 = 1$ e $\Delta w_0 = \Delta b$)
      2.1.2 $w_i = w_i + \Delta w$

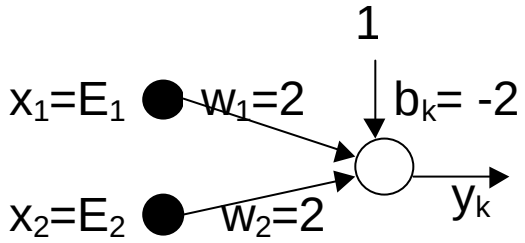
Ao final do treinamento espera-se que y = d (ou seja, a saída obtida seja igual à desejada). Vamos testar aplicando a regra a uma operação lógica AND bipolar, dada pela tabela:

Entrada E_1	Entrada E_2	Saída	
1	1	1	} Classe 0
1	-1	-1	} Classe 1
-1	1	-1	
-1	-1	-1	

Vamos aplicar o algoritmo e fazer as contas, indicando-as na tabela abaixo (lembrando-nos de incluir o bias):

E1	E2		Saída (d)	$\Delta w_0 =$ (1*d)	$\Delta w_1 =$ (E1*d)	$\Delta w_2 =$ (E2*d)	novo b	novo w_1	novo w_2
							0	0	0
1	1	1	1	1	1	1	1	1	1
1	-1	1	-1	-1	-1	1	0	0	2
-1	1	1	-1	-1	1	-1	-1	1	1
-1	-1	1	-1	-1	1	1	-2	2	2

Ou seja, a rede final encontrada foi:



Testando as entradas, vemos que a rede está treinada.

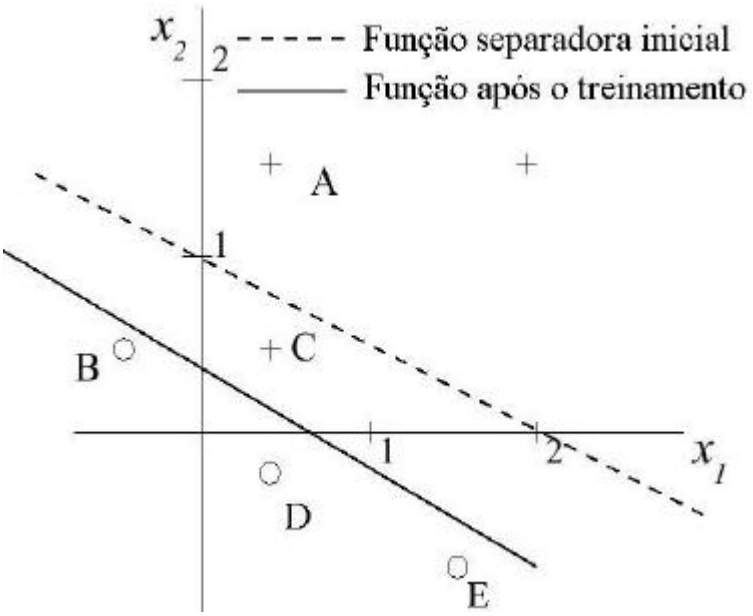
3.3-Regra de treinamento PERCEPTRON

Esta regra foi proposta por Rosemblatt (1959) e apresenta algumas mudanças em relação à regra de Hebb. A função de ativação passou a poder ter um limiar θ diferente de zero. Foi introduzido um fator de aprendizado η ($0 < \eta \leq 1$), que regula a velocidade de modificação dos pesos. O algoritmo de treinamento foi modificado, de forma que não é feita correção nos pesos quando a rede responde corretamente ao padrão. O algoritmo completo é o seguinte:

1 Inicie aleatoriamente os pesos e bias (= 0, por simplicidade)
2 Repita
  Para cada padrão (entrada e saída desejada) faça
    se $y \neq d$ faça
      Para cada conexão faça
        $\Delta w_i = \mathbf{h} * \mathbf{x_i} * d$
        $w_i = w_i + \Delta w_i$
até que não tenham sido feitas alterações para os padrões

Vamos analisar um exemplo, ilustrado na figura abaixo.

Na figura ao lado, os ponto marcados com x pertencem a uma classe e os pontos marcados com círculo, pertencem a outra classe. O PERCEPTRON deve ser capaz de achar a função separadora das classes. A tabela com os valores das entradas, saídas e dos pesos e bias encontra-se abaixo.



Os pesos foram iniciados com valores arbitrários diferentes de zero e que correspondem à reta separadora inicial (reta pontilhada na figura). A taxa de aprendizado é 1 e o limiar 0.

+ o	x_1	x_2	1	Saída (d)	$\Delta w_0 = (1*d)$	$\Delta w_1 = (x_1*d)$	$\Delta w_2 = (x_2*d)$	Novo B	novo w_1	novo w_2
								-2	1	2
A	0,5	1,5	1	1	—	—	—	—	—	—
B	-0.5	0.5	1	-1	—	—	—	—	—	—
C	0,5	0,5	1	1	1	0,5	0,5	-1	1,5	2,5
D	0,5	-0,3	1	-1	—	—	—	—	—	—
E	1,5	-1	1	-1	—	—	—	—	—	—

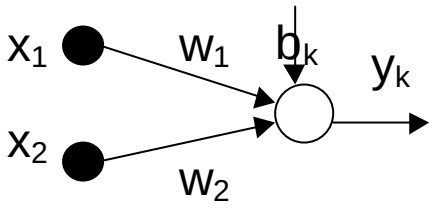
Como houve uma mudança neste ciclo (época) o algoritmo ainda calcularia todos os padrões uma segunda vez e só então, como não haveria mudanças, o algoritmo pararia. A reta separadora, correspondente aos novos parâmetros de pesos e bias encontrados é a reta cheia da figura acima.

Na prática pode acontecer da convergência para todos os padrões não ser possível, ou seja, o erro não atingir zero. Para tanto, pode-se modificar o algoritmo para parar quando um percentual desejado de respostas corretas tiver sido atingido

(90%, por exemplo). Entretanto, Rosemblat provou que o algoritmo é convergente, isto é, o erro sempre diminui.

3.4-Redes ADALINE

Uma importante alteração foi proposta por Widrow e Hoff em 1960. A rede por eles proposta é uma rede adaptativa (ADAptative Linear Element), treinada por um algoritmo de mínimos quadrados, também conhecido como Regra Delta.
A arquitetura do neurônio tem a mesma forma do Perceptron:



Também da mesma forma que no Perceptron, a rede pode ter vários neurônios recebendo as mesmas entradas em uma única camada de saída.
Entretanto, se os nós são combinados de forma que a saída de um esteja conectada à entrada de outro, temos uma rede de múltiplas camadas, chamada rede MADALINE (Many ADAptative Linear Elements). A saída de cada nó é:

$$y_k = \sum_{i=0}^{M} w_{ki} x_i \qquad (3.1)$$

Onde M é o número de entradas da rede.
A idéia básica da regra Delta é ajustar os pesos a partir do erro na saída da rede. O erro (também chamado custo) para um nó, dado um determinado padrão p, é expresso como:

$$e_k^p = d_k^p - y_k^p \qquad (3.2)$$

Onde: d_k é a saída desejada (fornecida no padrão *p* de entrada/saída) e y_k é a saída realmente obtida (fornecida pela rede quando é apresentado o padrão *p*). Se temos mais de um nó na saída, precisamos considerar todos os erros dos diferentes nós de saída. Entretanto, se simplesmente somarmos os erros, a tendência é que erros positivos anulem os negativos, causando uma falsa visão do erro total. Assim, a proposta é minimizar a função de erro dada pelo somatório dos quadrados dos erros na saída.

Ou seja, queremos minimizar a função de erro dada por:

$$E = \frac{1}{2}\sum_{k=1}^{N} e_k^2 = \frac{1}{2}\sum_{k=1}^{N}\left(d^p - y^p\right)^2 \qquad (3.3)$$

Onde *N* é o número total de nós na camada de saída. O fator ½ é unicamente para ajuste do algoritmo de treinamento. Como o erro *E* é função da saída da rede y_k, e esta é função dos pesos w_{ki}, podemos dizer que o erro *E* é função (ou seja, varia com) os pesos da rede. Ou seja: $E = f(w_{k1}, w_{k2}, w_{k3}, \ldots w_{kN})$. Assim, a idéia é minimizar o erro ajustando os pesos, ou seja, encontrar o mínimo da função de erro. Este mínimo é dado pela derivada parcial da função de erro em relação a cada peso: $\dfrac{\partial E}{\partial w_{ki}}$

O gradiente da função de erro é o vetor composto pelas derivadas parciais da função de erro em relação a cada peso e aponta para a direção de crescimento da função, da mesma forma que a derivada em uma função monovalorada *f(x)*. Assim, a idéia é variar os pesos de forma a caminhar na direção contrária ao gradiente, que é a direção de maior diminuição do valor da função de erro. Ou seja queremos:

$$\Delta w_{ki} = -\eta \frac{\partial E}{\partial w_{ki}} \qquad (3.4)$$

Onde η é uma constante de proporcionalidade, também chamada de taxa de treinamento. A derivada parcial da equação (3.4) pode ser decomposta, pela regra da cadeia, em:

$$-\eta \frac{\partial E}{\partial w_{ki}} = -\eta \frac{\partial \frac{1}{2}\sum_{k=1}^{N}\left(d^p - y^p\right)^2}{\partial w_{ki}} = \eta \left(d^p - y^p\right)\frac{\partial y^p}{\partial w_{ki}} \qquad (3.5)$$

Da equação (3.1), temos que: $\partial y_k / \partial w_{ki} = x_i$, o que resulta:

$$\Delta w_{ki} = \eta \left(d^p - y^p\right) x_i \qquad (3.6)$$

Que é a regra Delta para atualização dos pesos.

A regra é capaz de atualizar corretamente os pesos a cada iteração, com entradas binárias ou contínuas.

O algoritmo de treinamento pode ser assim resumido:

1 Inicie os pesos e bias com valores aleatórios pequenos

2 Repita

Para cada padrão (entrada e saída desejada) faça

Calcule y^p

Para cada conexão faça

$$\Delta w_i = \eta * x_i * (d^p - y^p)$$

$$w_i = w_i + \Delta w_i$$

até que um erro pequeno tenha sido atingido

Caso desejemos obter saídas binárias, podemos introduzir uma função de limiar na saída do nó. Um exemplo, seria para uma aplicação de emulação de uma porta lógica. Podemos definir uma função de saída bipolar com limiar 0 e aplicar a tabela verdade de uma função OR bipolar no treinamento.

Entrada E_1	Entrada E_2	Saída
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

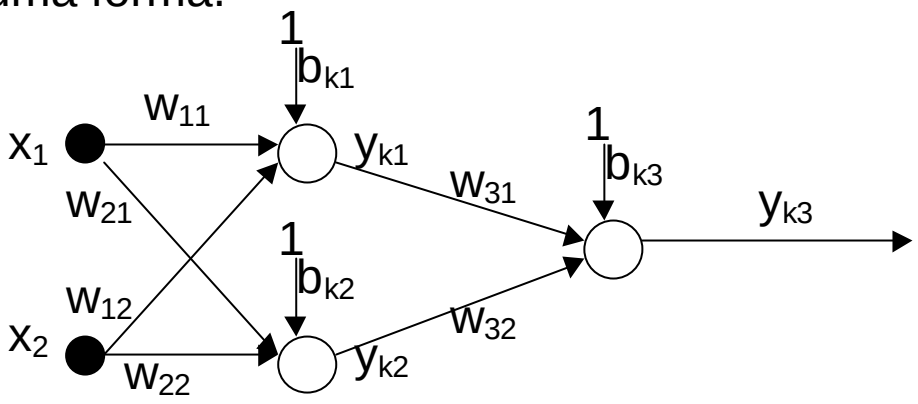
Usando um nó com duas entradas e uma saída, iniciando os pesos (inclusive o bias) com o valor 0.1 e η= 0.25 , temos:

#	1	E1	E2	d	y	d = (d-y)	Δw_0 = 1*d*η	Δw_1 = E1*d*η	Δw_2 = E2*d*η	novo w_0(b)	Novo w_1	novo w_2
										0.10	0.10	0.10
1	1	1	1	1	0.30	0.70	0.18	0.18	0.18	0.28	0.28	0.28
	1	1	-1	1	0.28	0.72	0.18	0.18	-0.18	0.46	0.46	0.09
	1	-1	1	1	0.09	0.91	0.23	-0.23	0.23	0.68	0.23	0.32
	1	-1	-1	-1	0.13	-1.13	-0.28	0.28	0.28	0.40	0.51	0.60
2	1	1	1	1	1.51	-0.51	-0.13	-0.13	-0.13	0.27	0.38	0.47
	1	1	-1	1	0.18	0.82	0.21	0.21	-0.21	0.48	0.59	0.26
	1	-1	1	1	0.15	0.85	0.21	-0.21	0.21	0.69	0.38	0.47
	1	-1	-1	-1	-0.16	-0.84	-0.21	0.21	0.21	0.48	0.59	0.68

No segundo passo a saída y já está correta (usando o limiar).

3.5-Redes MADALINE

Nas redes com múltiplos ADALINE (Many ADALINE), estes são conectados em mais de uma camada de forma a combiná-los de alguma forma.



Esta combinação pode ser uma “regra da maioria”, em que o ADALINE da saída não possui pesos e somente propaga a saída dos ADALINES ocultos que formem uma maioria.

Pode ainda ser uma combinação lógica dos ADALINES ocultos, em que o ADALINE de saída possui pesos fixos responsáveis por essa função lógica.

Por exemplo, com w_{31}=0,5 , w_{32}=0,5 e b_{k3}=0,5 temos uma combinação correspondente a uma porta OU.

Já com w_{31}=0,5 , w_{32}=0,5 e b_{k3} = -0,5 temos uma combinação correspondente a uma porta AND.

O algoritmo MR1 foi desenvolvido por Widrow e Hoff (1960) para treinamento de redes MADALINE com entradas bipolares.

Uma extensão desse algoritmo, conhecida como MRII foi desenvolvida por Widrow, Winter e Baxter (1987) que também permite ajuste nos pesos da camada de saída, o que permite maior flexibilidade ao algoritmo. Entretanto, o melhor algoritmo para treinamento de redes com múltiplas camadas é o conhecido como Backpropagation, que será explicado na próxima sessão.

A versão do algoritmo de treinamento MR1, originalmente proposto pelos citados autores, possui a seguinte forma:

1 Inicie os pesos e bias com valores aleatórios pequenos para os nós ocultos e com os valores fixos para o nó de saída, de acordo com a função combinatória desejada (OR/AND)

2 Repita

Para cada padrão (entrada e saída desejada) faça

Calcule u^p (entrada líquida) e y^p para todos os nós

Se $y_{\text{nó saída}} \neq d$ então

Caso a função do nó de saída seja

OR:

Se d = 1 então

Para cada conexão do nó k cujo valor da entrada líquida u_k é o mais próximo de zero, faça

$$\Delta w_{ki} = \eta * x_i * (1 - u_k)$$

$$w_{ki} = w_{ki} + \Delta w_{ki}$$

Se d = -1 então

Para cada conexão de nó k cujo valor da entrada líquida u_k seja positivo, faça

$$\Delta w_{ki} = \eta * x_i * (-1 - u_k)$$

$$w_{ki} = w_{ki} + \Delta w_{ki}$$

AND:

Se d = 1 então

Para cada conexão de nó k cujo valor da entrada líquida u_k seja negativo, faça

$$\Delta w_{ki} = \eta * (1 - u_k)$$

$$w_{ki} = w_{ki} + \Delta w_{ki}$$

Se d = -1 então

Para cada conexão do nó k cujo valor da entrada líquida u_k é o mais próximo de zero, faça

$$\Delta w_{ki} = \eta * x_i * (-1 - u_k)$$

$$w_{ki} = w_{ki} + \Delta w_{ki}$$

até que um erro pequeno tenha sido atingido, os pesos não estejam mais variando ou execute um número fixo de vezes.

Como exemplo, implementaremos uma rede MADALINE para emular uma porta lógica XOR que, como já vimos, não pode

ser representada com uma rede de uma só camada, dado que a função não é linearmente separável. Podemos definir uma função de saída bipolar com limiar 0 e aplicar a tabela verdade de uma função XOR bipolar no treinamento.

Entrada E_1	Entrada E_2	Saída
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

O problema como um todo apresenta duas entradas e uma saída. Usaremos uma rede com dois nós ocultos e um nó de saída com uma função de combinação OR. Como a função do nó de saída é OR, temos que:

w_{31}=0,5 , w_{32}=0,5 e b_{k3}=0,5

Como é necessário iniciar os pesos (inclusive o bias) para os demais nós com valores pequenos e aleatórios, vamos escolher arbitrariamente:

w_{10}=b_{k1}=0,30 , w_{11}=0,05 e w_{12}=0,20

w_{20}=b_{k2}=0,15 , w_{21}=0,10 e w_{22}=0,20

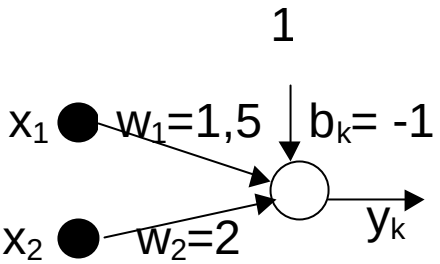
Também devemos determinar uma taxa de aprendizado, de forma que arbitrariamente escolhemos η= 0.5. Os cálculos:

#	1	**E 1**	**E 2**	d	**U_1**	**U_2**	**Y_3**	Novo w_{10}	Novo w_{11}	Novo w_{12}	Novo w_{20}	Novo w_{21}	Novo w_{22}
								0.30	**0.05**	**0.20**	**0.15**	**0.10**	**0.20**
1	**1**	**1**	**1**	**-1**	**0.30+ 0.05+ 0.20 =0.55**	**0.15+ 0.10+ 0.20 =0.45**	**1**	**0.30+ 0.50* -1.55 =- .48**	**0.05+ 0.50* -1.55 =- .73**	**0.20+ 0.50* -1.55 =- .58**	**0.15+ 0.50* -1.45 =- .58**	**0.10+ 0.50* -1.45 =- .63**	**0.20+ 0.50* -1.45 =- .53**
	1	**1**	**-1**	**1**	**- .48+ - .73+ 0.58= - 0.63**	**- .58+ - .63+ 0.53= - 0.68**	**-1**	**- .48+ 0.50* 0.37 =- .30**	**- .73+ 0.50* 0.37 =- .55**	**- .58+ 0.50* -0.37 =- .77**	**-**	**-**	**-**
	1	**-1**	**1**	**1**	**:**	**:**	**:**	**:**	**:**	**:**	**:**	**:**	**:**
	1	**-1**	**-1**	**-1**	**:**	**:**	**:**	**:**	**:**	**:**	**:**	**:**	**:**

Após quatro épocas de treinamento, os pesos assumiram os valores $w_{10}=b_{k1}=-0,99$, $w_{11}=-0,73$, $w_{12}=1,27$, $w_{20}=b_{k2}=-1,09$, $w_{21}=1,53$ e $w_{22}=-1,33$ e as saídas convergiram adequadamente.

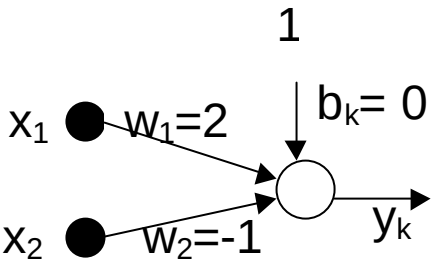
EXERCÍCIOS

1 – Considere a rede ao lado, com função de ativação de McCullock-Pitts e com limiar **q** = 0. Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.

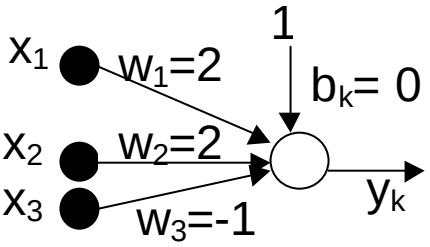


2 – Recalcule a mesma rede, agora com entradas bipolares (-1 e 1) e limiar **q** = 1.

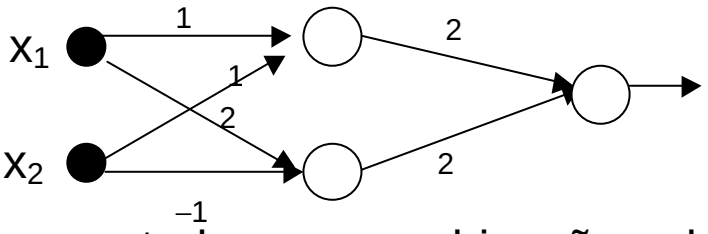
3 – Considere a rede ao lado, com função de ativação de McCullock-Pitts e com limiar **q** = 2. Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.



4 – Considere a rede ao lado, com função de ativação de McCullock-Pitts e com limiar **q** = 4. Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.



5 – Considere a rede com função de ativação de McCullock-Pitts, com limiar **q** = 2 e com polarização b=0 para todos os nós.



Apresente a saída da rede para todas as combinações de entradas binárias (0 e 1) possíveis.

6 – Construa uma rede de com função de ativação de McCullock-Pitts e com limiar **q** = 0, que responda como uma função OR NOT bipolar. Use para treinar a regra de Hebb.

7 – Construa uma rede com função de ativação de McCullock-Pitts, com limiar **q** = 0, que responda como uma função AND NOT bipolar. Use para treinar a regra de Hebb.

8 – Use a regra de Hebb para classificar os vetores a = (1, 1, 1, 1) e b = (–1, 1, –1, –1) em um grupo (saída = 1), e os vetores c = (1, 1, 1, –1) e d = (1, –1, –1, 1) em outro (saída = -1). Use limiar **q** = 0. Faça um esquema da rede, construa a tabela de treinamento e preencha a tabela usando a regra de Hebb.

9 – Repita o problema anterior usando a regra de treinamento Perceptron, com **h**= 1 e **q** = 0. Inicie todos os pesos com zero.

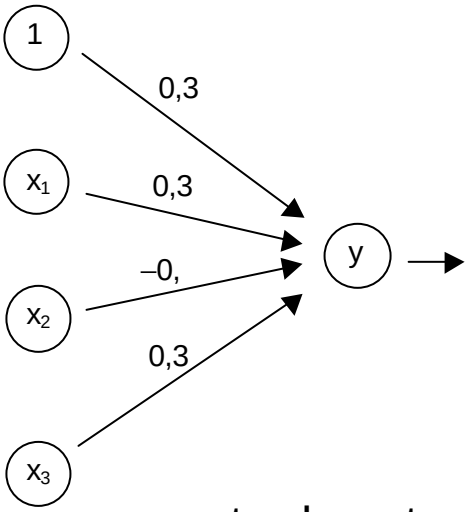
10 – Usando regra Perceptron, treine uma rede para emular uma porta AND.

11 – Construa um perceptron de duas saídas para classificar os vetores a = (1, 1, 1) e b = (1, –1, –1) em uma classe, e c = (1, 1, –1) e d = (–1 ,1, –1) em outra. Faça **h** = 1, θ = 0, e inicialize os pesos e viés com zero. Construa a tabela de treinamento, faça um esquema gráfico da rede e teste–a para as entradas dadas e para os vetores e = (–1, –1, 1), f = (0, –1, 1), g = (–1, 1, 0) e h = (1, 1, 0) . Discuta os resultados.

12 – Construa uma rede Adaline para a função lógica AND, utilizando entradas e saídas bipolares. Inicie todos os pesos com o valor 0.1 e **h**= 0.25.

13 – Construa uma rede Adaline para a função lógica OR NOT, utilizando entradas e saídas bipolares. Inicie todos os pesos com o valor 0.1 e η= 0.25.

14 – Foi construída uma rede Adaline, com **h** = 0,25, com os pesos e bias iniciados em 0,1, para classificar os vetores a = (1, –1, 1) e b = (1, 1, –1) na classe de saída 1, e c = (1, –1, –1) na classe de saída –1. Imediatamente antes de entrar o vetor b, a rede apresentou a configuração ao lado.



Forneça os valores dos pesos após o processamento do vetor de entrada b.