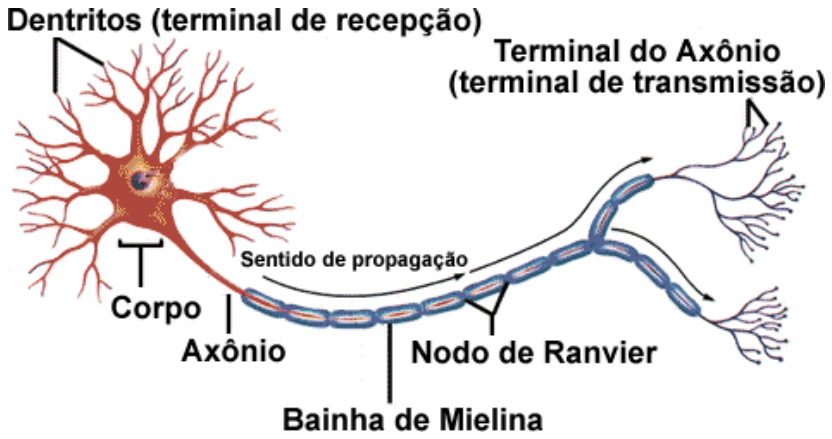
REDES NEURAIS ARTIFICIAIS

Prof. Igor da Penha Natal

Redes Neurais Artificiais

- Usam como modelo a estrutura física do cérebro.
- Tem a habilidade de:
  - Executar computação distribuída.
  - Tolerar entradas ruidosas.
  - Aprender.
- É uma das formas mais populares e efetivas de sistemas de aprendizagem.

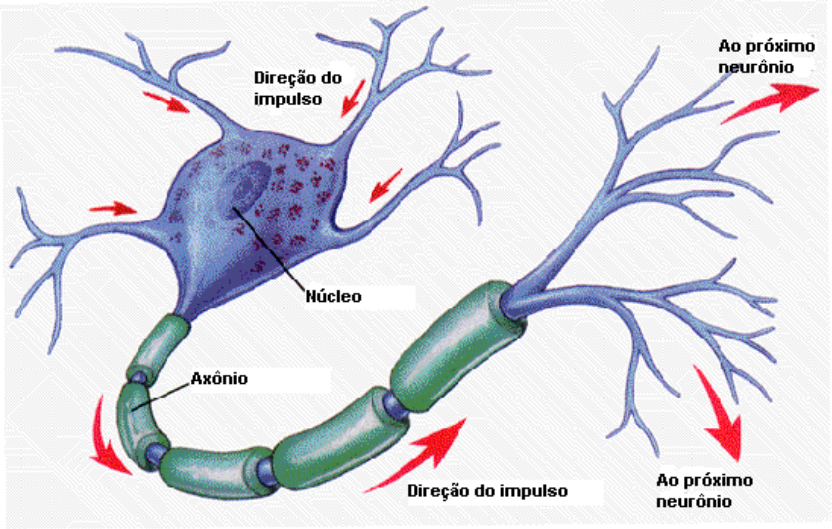
O Neurônio Biológico



NEURÔNIO: é a célula do sistema nervoso responsável pela condução do impulso nervoso, sendo composta de:

- dendritos
- corpo celular
- axônio
- telodendritos (nodo de Ranvier)

O Neurônio Biológico

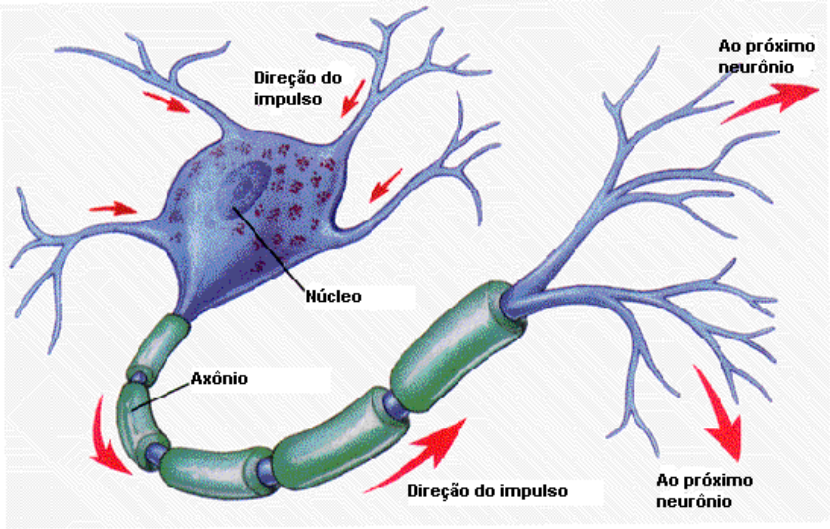


O fluxo da informação ocorre sempre no sentido:

Dendritos ➔ Corpo Celular ➔ Axônio

- **DENDRITOS**: ramificações correspondentes aos canais de entrada de informação (sinais elétricos, escala mVolts).
- **CORPO CELULAR**: local onde é feito o balanço energético da célula nervosa (soma das contribuições de energia).
- **AXÔNIO**: canal de saída do neurônio, ou seja, caminho de propagação dos impulsos nervosos em direção a outros neurônios ou músculos.

O Neurônio Biológico

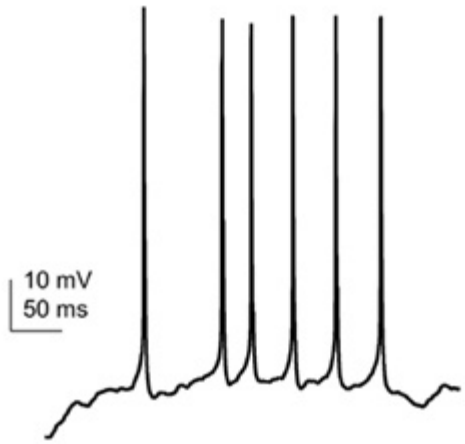


- O axônio emite um impulso elétrico (potencial de ação) apenas se o balanço energético realizado no corpo celular for maior que um certo **LIMIAR**.
- Neste caso, diz-se que o neurônio disparou ou está **ATIVADO**.

O fluxo da informação ocorre sempre no sentido:

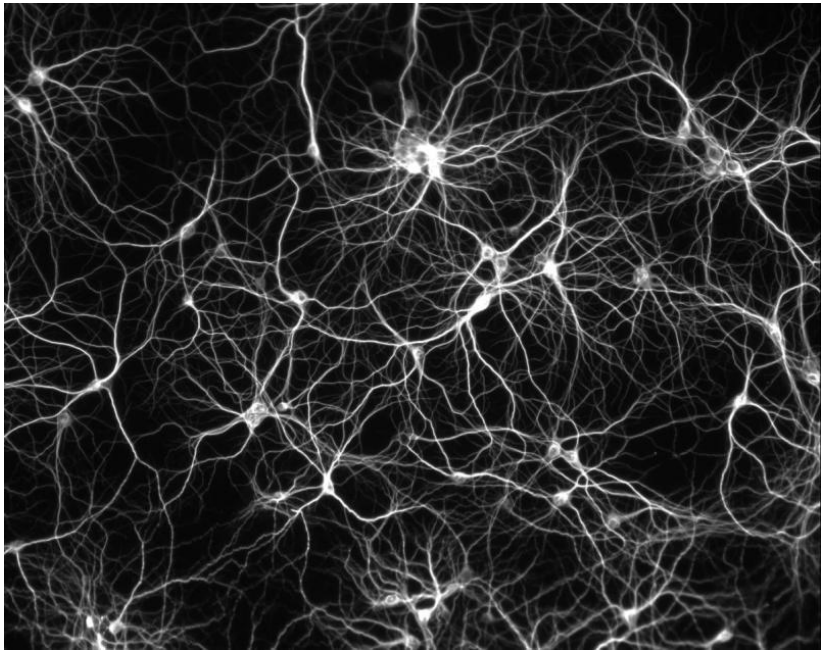
Dendritos ➔ Corpo Celular ➔ Axônio

O Neurônio Biológico



- Um neurônio devidamente estimulado emite um trem de potenciais de ação ao longo de seu axônio.
- A informação é, então, codificada na frequência dos potenciais de ação!

O Neurônio Biológico



- Os neurônios podem se conectar diretamente a outros neurônios (formando **REDES**).

Funcionamento de uma Rede Neural Biológica

- Por meio das sinapses os dendritos recebem sinais excitatórios ou inibitórios.
- Esses sinais chegam ao neurônio por meio dos dendritos, e são somados no corpo celular.
- Quando a soma atinge um **LIMIAR** o neurônio dispara um sinal para os outros (por meio do axônio).
- As sinapses regulam a forma de como a informação passa pelo neurônio.

Propagação da informação em uma Rede Neural Biológica

- Não é qualquer informação presente na entrada que se propaga para a saída.
- É necessário uma combinação correta entre transmissores e receptores nervosos.
- Quando esta combinação acontece, o neurônio dispara (propaga informação).
- A eficácia dessa combinação é determinada pela presença em quantidade suficiente do neurotransmissor na conexão nervosa.

Ajuste das sinapses em uma Rede Neural Biológica

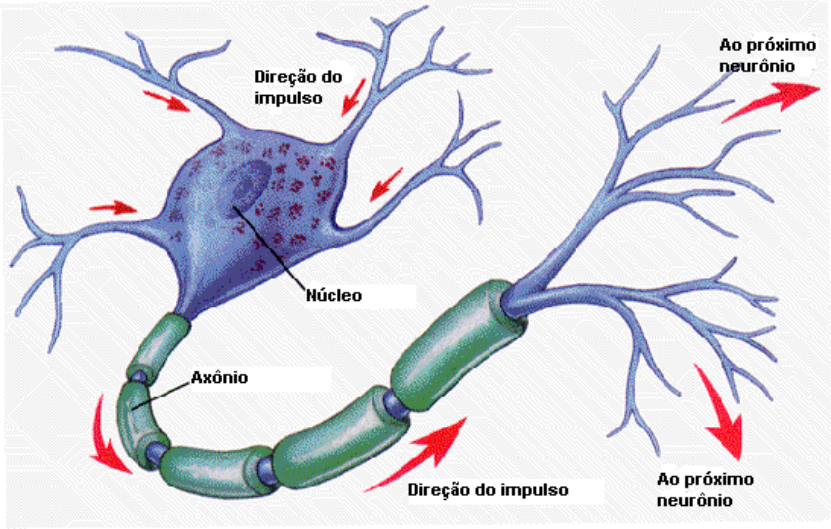
- As sinapses são ajustadas primeiramente baseadas no genótipo.
- No decorrer da vida as sinapses são ajustadas através do treino ou aprendizado.
  - Quanto mais uma conexão é utilizada, maior é a quantidade de neurotransmissores liberados, e mais forte se torna a conexão.
    - Conexões são ajustadas – se tornam mais fracas ou mais fortes.
    - Conexões são construídas.

O Neurônio ARTIFICIAL

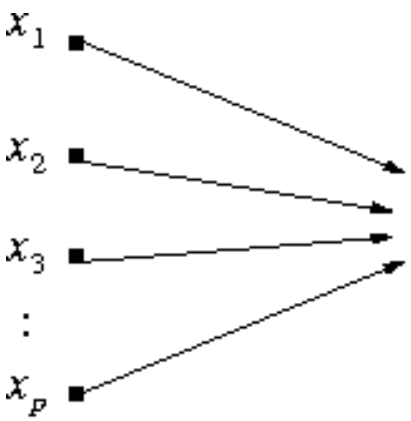
- Foi proposto em 1943 por W. S. McCulloch e W. Pitts.
- Trata-se de um **modelo matemático aproximado** e útil do comportamento real de um neurônio para aplicação em algoritmos computacionais.
- Conhecido por **NEURÔNIO M-P**.

O Neurônio ARTIFICIAL

- Os **dendritos** são modelados como **linhas** ou **CANAL DE TRANSMISSÃO** por onde flui a informação de entrada.

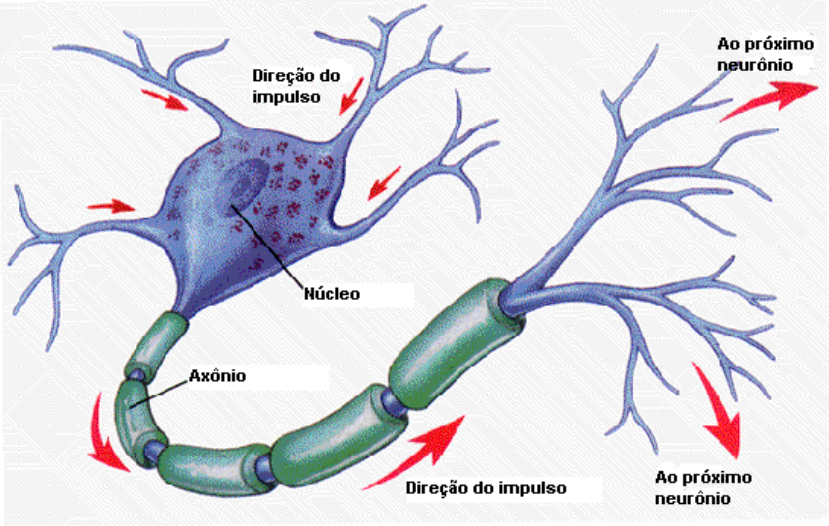


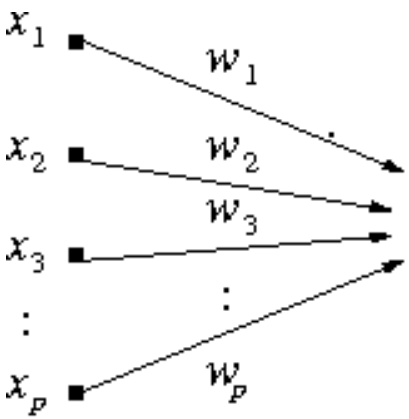
- $(x_j, j=1, \ldots, p)$.



O Neurônio ARTIFICIAL

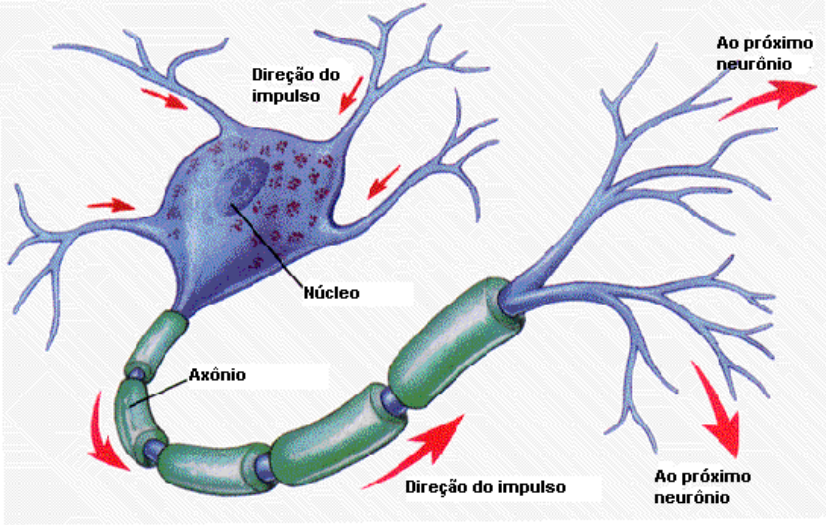
- A **força** (ou eficiência) **das conexões sinápticas** de um dendrito é modelada como um fator (**PESO SINÁPTICO**), cujo papel é modular o fluxo de sinais.

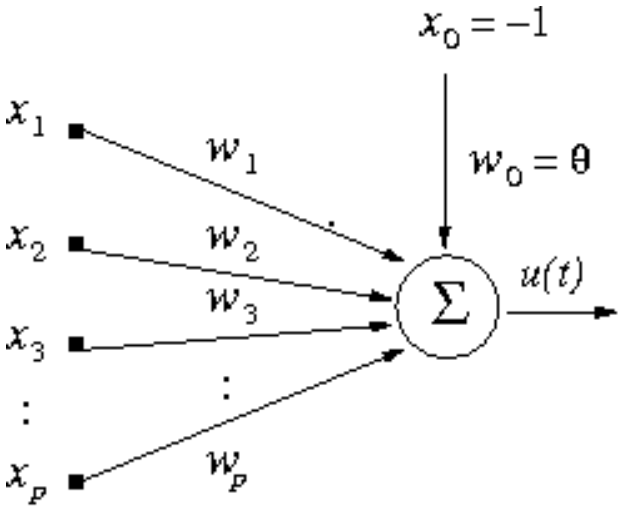




O Neurônio ARTIFICIAL

- A função do corpo celular de realizar o **balanço** ou **acúmulo energético** é modelada por uma **OPERAÇÃO DE SOMATÓRIO** sobre as entradas moduladas pelos pesos sinápticos.





$$u = w_1x_1 + w_2x_2 + ... + w_px_p - \theta$$

x_1, x_2: entradas

w_1, w_2: pesos sinápticos

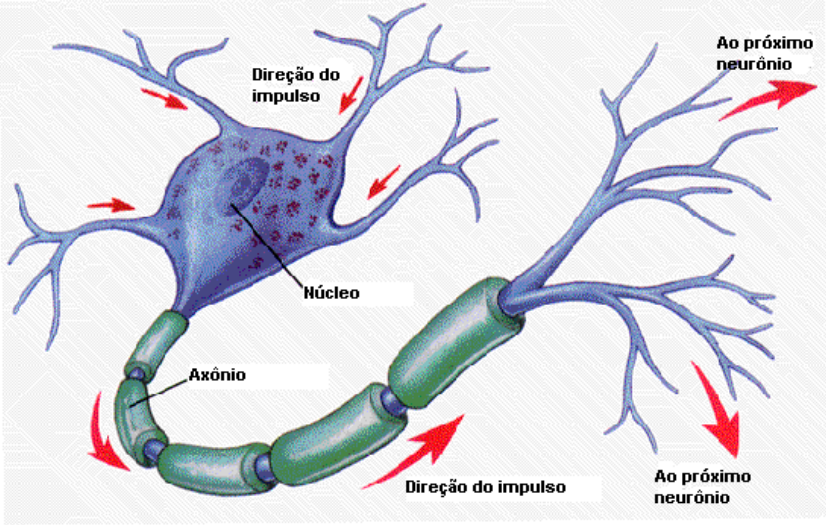
θ: limiar (*bias*)

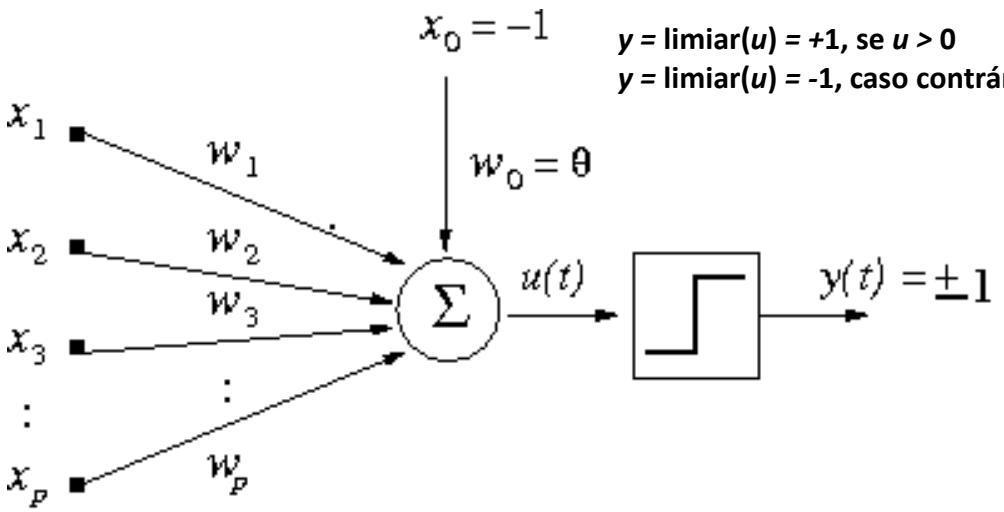
u : ativação

O Neurônio ARTIFICIAL

- O **axônio** é modelado como uma **FUNÇÃO DE ATIVAÇÃO**, i.e. uma chave ON/OFF que indica se o neurônio respondeu ao estímulo atual.

- Em outras palavras, se houve ou não o envio de um potencial de ação.





y = limiar(u) = +1, se $u > 0$
y = limiar(u) = -1, caso contrário.

Funções de Ativação

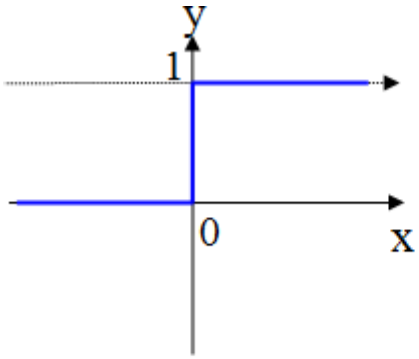
- A **FUNÇÃO DE ATIVAÇÃO** caracteriza o neurônio em **linear** ou **não-linear**.

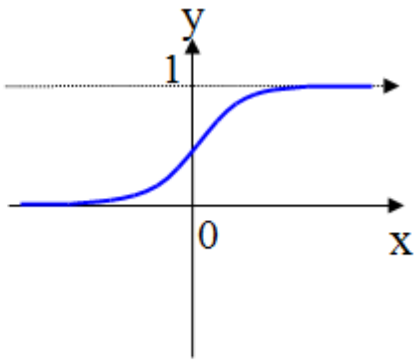
- Modelo não-linear mais simples: **FUNÇÃO DE LIMIAR**
  - Quando o somatório ponderado das entradas atinge um certo valor (normalmente zero), o neurônio dispara.
  - Conhecido como neurônio binário.

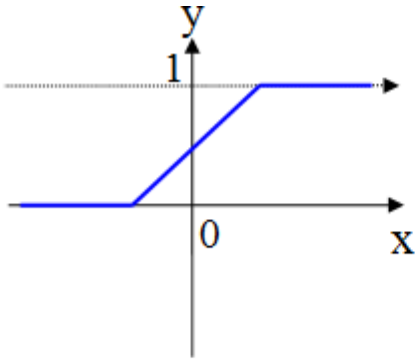
- Modelo não-linear mais utilizado: **FUNÇÃO SIGMOIDE**
  - Saída é proporcional a soma ponderada das entradas.
  - Função de ativação que mais se aproxima de um neurônio real.

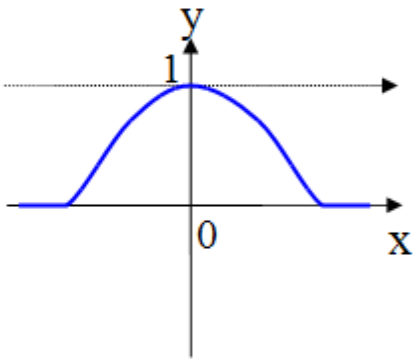
Formas de função de ativação

(ou função de transferência)

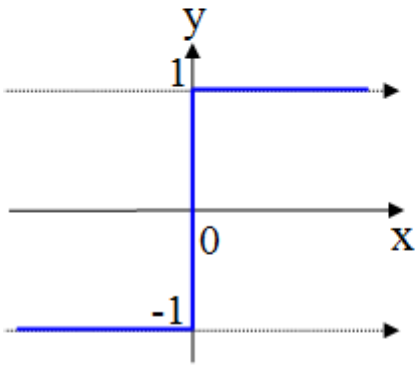




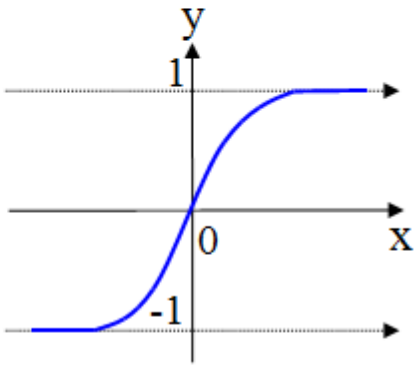




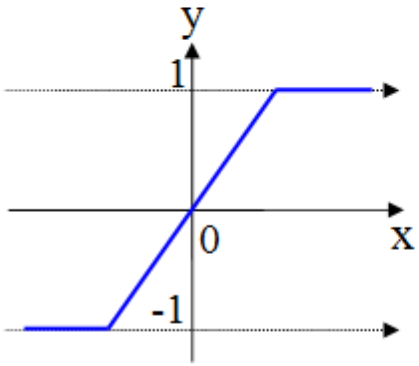
Radial básica



limiar



Sigmóide



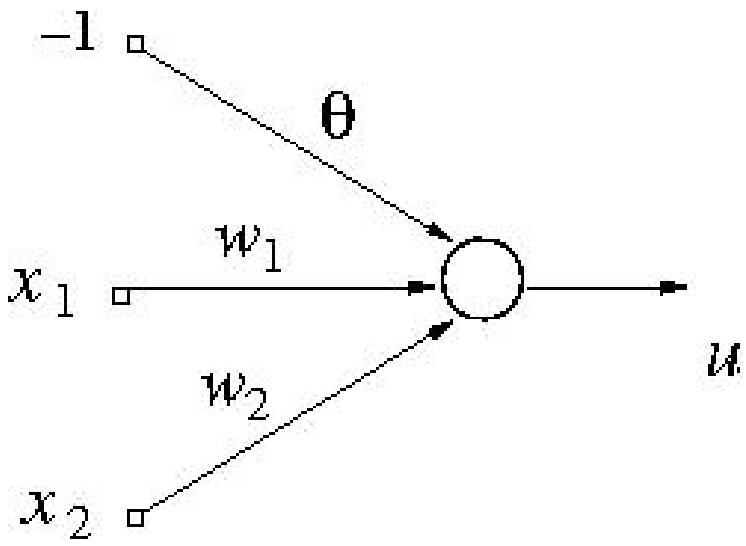
Rampa-limitada

Análise Matemática/Geométrica do Neurônio Artificial

- Tomemos este neurônio, com apenas duas entradas x_1 e x_2.
- Sua **FUNÇÃO DE ATIVAÇÃO** (u) será dada por

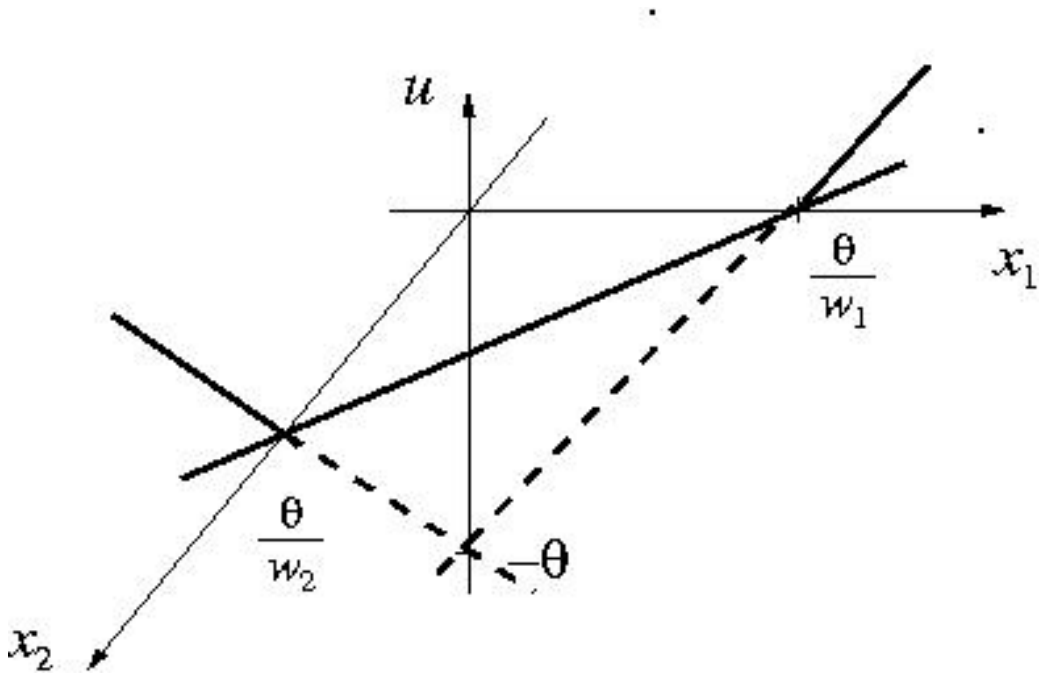
$$u = x_1 w_1 + x_2 w_2 - \theta$$

- onde θ é o **limiar de ativação**.



Análise Matemática/Geométrica do Neurônio Artificial

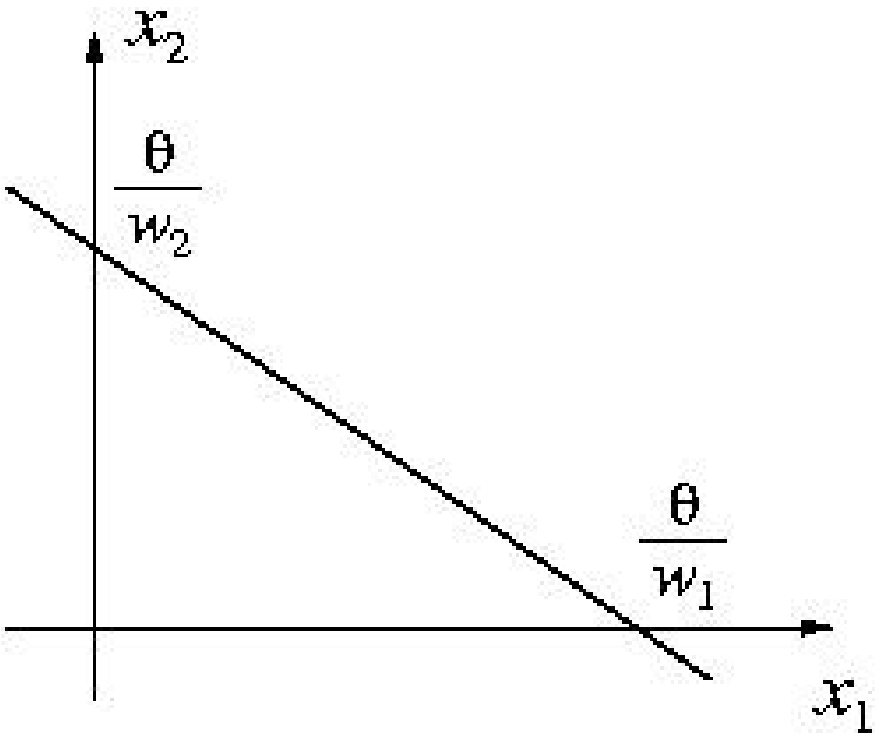
- Se pensarmos num gráfico 3D, veremos que x_1 e x_2 formam um plano.



Análise Matemática/Geométrica do Neurônio Artificial

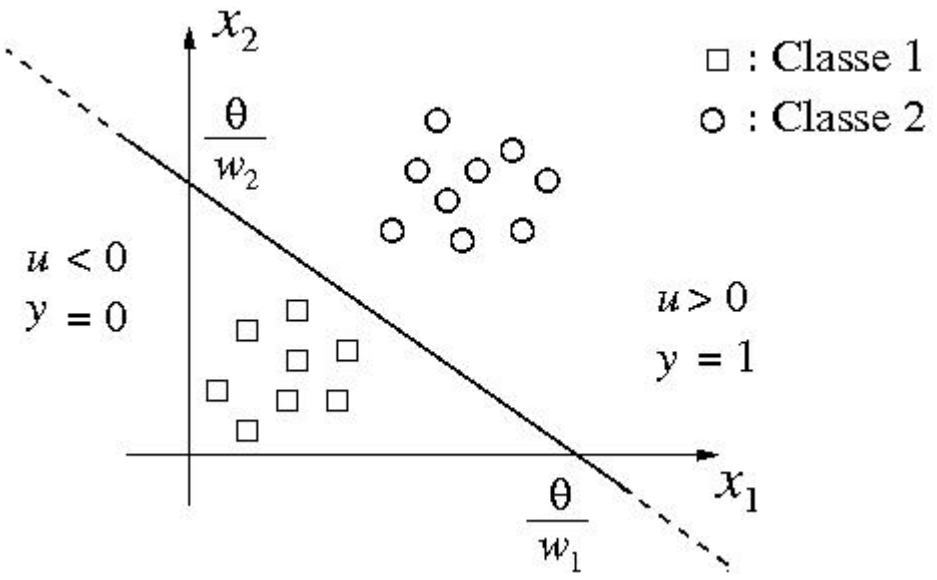
- Pra simplificar, tomemos u=0 e trabalhando apenas no plano x_1x_2 teremos uma reta de equação

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$$



Análise Matemática/Geométrica do Neurônio Artificial

- Isto nos permite, por exemplo, utilizar este neurônio para separar com eficiência duas classes bem isoladas.

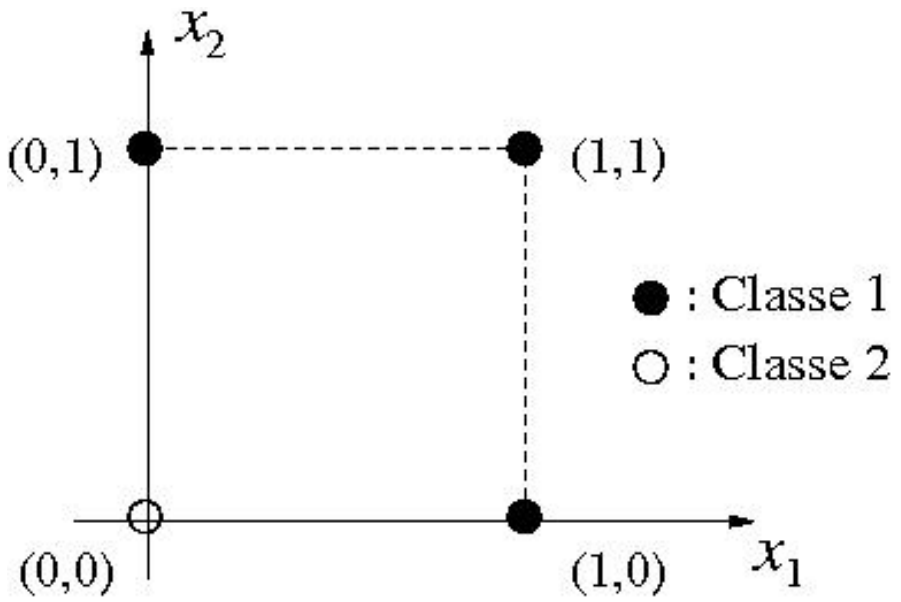


Simulando portas lógicas com Neurônios M-P

- Se plotarmos a tabela-verdade da porta OR num plano cartesiano, teremos o seguinte resultado:

Porta OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

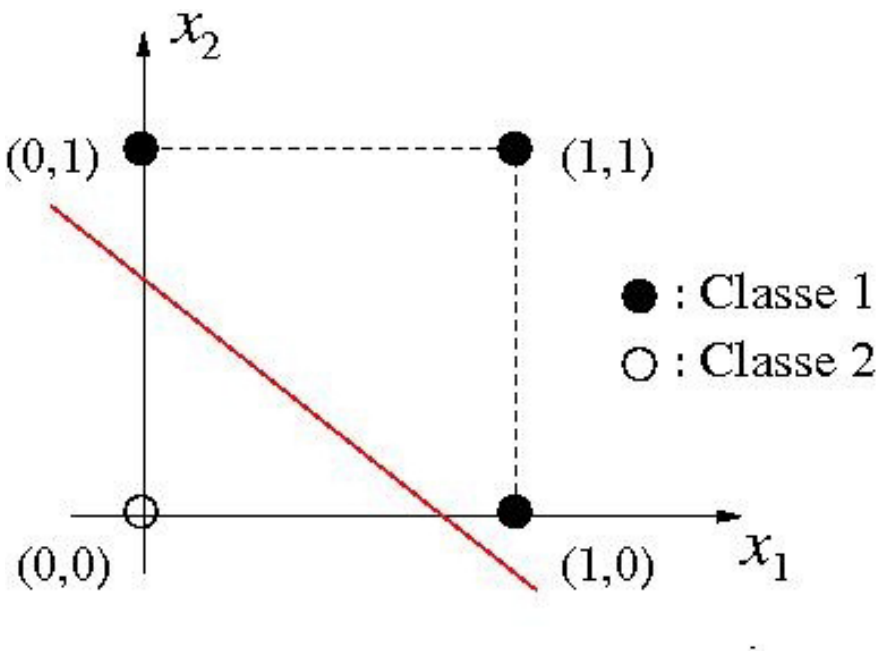


Simulando portas lógicas com Neurônios M-P

- Acho que podemos encontrar pelo menos uma reta separando as classes y=1 e y=0

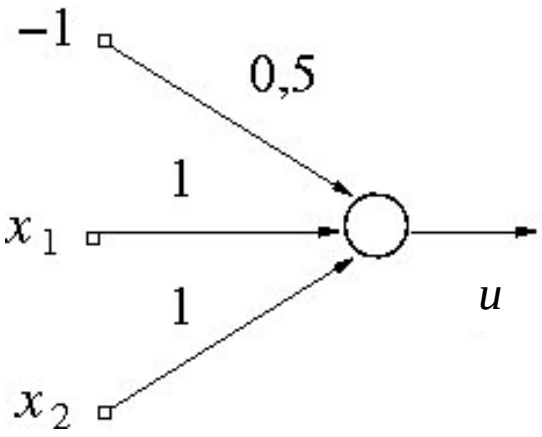
Porta OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



Simulando portas lógicas com Neurônios M-P

- e isto pode ser feito com o seguinte neurônio:



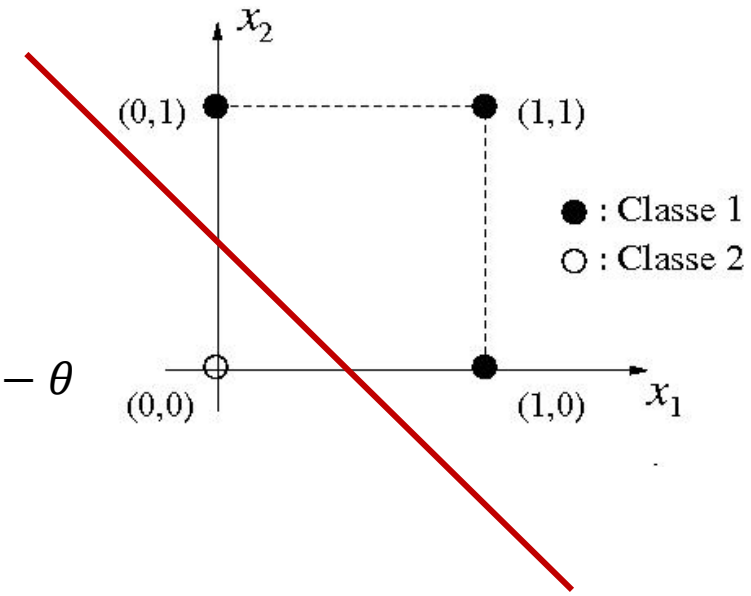
$$w_1 = w_2 = 1$$

$$\theta = 0{,}5$$

$$u = x_1 w_1 + x_2 w_2 - \theta$$

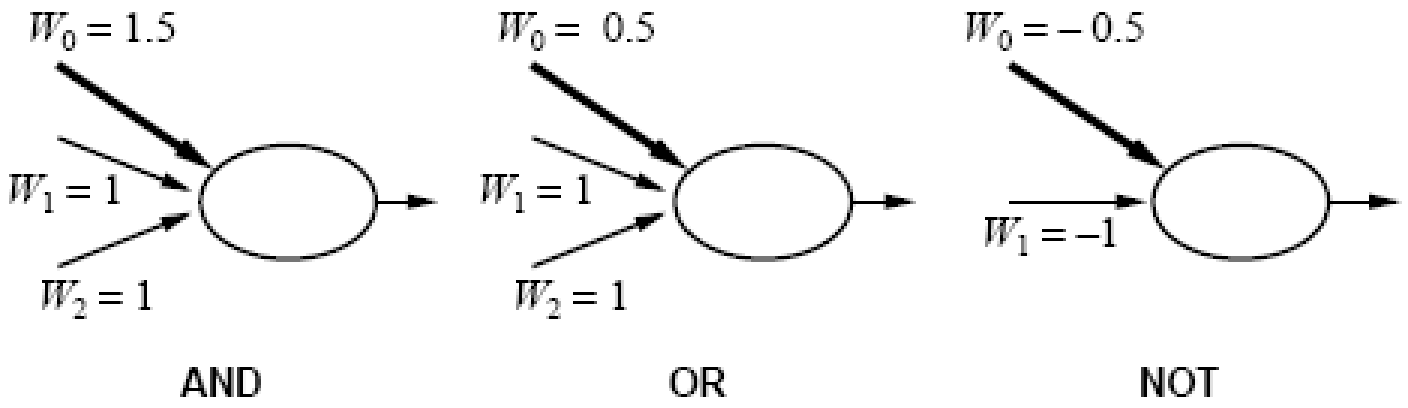
$$u = x_1 + x_2 - 0{,}5$$

$$\boldsymbol{x_2 = -x_1 + \frac{1}{2}}$$



Neurônios como portas lógicas

- Usam uma função de ativação de limiar.



Considerações sobre o Neurônio M-P

- No contexto de agentes de aprendizagem gostaríamos que o próprio neurônio possa determinar os pesos e o limiar adequado para o problema, isso se consegue incluindo na modelagem uma regra de aprendizagem.

- A **REGRA DE APRENDIZAGEM** é uma equação que permitirá o ajustes dos pesos e limiar de acordo com os erros cometidos durante a execução da tarefa de classificação, na fase de treinamento do neurônio.

Ajuste das sinapses em uma Rede Neural Artificial

- As sinapses, normalmente, possuem **VALORES RANDÔMICOS INICIALMENTE**.
  - Algum conhecimento prévio do domínio pode ajudar a definir os pesos sinápticos iniciais.
- Enquanto a rede aprende, as sinapses são ajustadas para fornecer saídas de melhor qualidade.
  - Em algumas tipos de redes apenas são feitos **ajustes dos pesos sinápticos** já existentes.
  - Em outros tipos de redes **criam-se neurônios e conexões** entre eles para melhorar a qualidade da saída.

Modelo de uma Rede Neural Artificial

- É caracterizado por:
  - Uma quantidade de **neurônios** (ou elementos processadores).
  - A forma das **conexões** entre eles (topologia da rede).
  - Seu **esquema de aprendizagem**.

Modelo de uma Rede Neural Artificial

- Cada entrada de um neurônio está associada a um **PESO SINÁPTICO**:
  - Excitatório (quando positivo) ou
  - Inibitório (quando negativo).

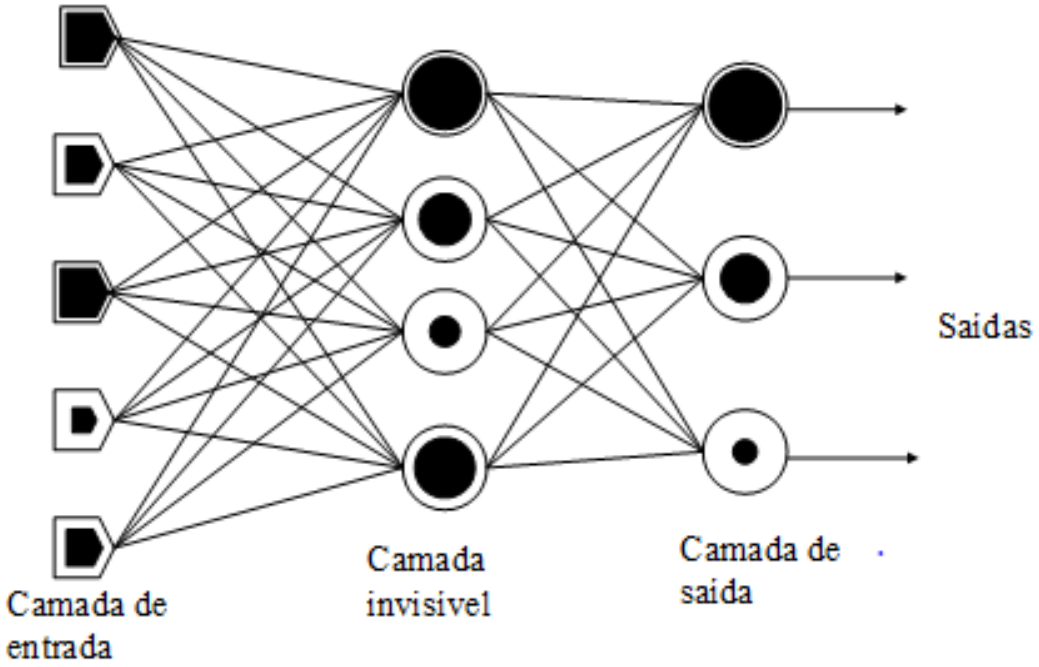
- O sinal de entrada pode assumir uma variação:
  - **Contínua** (desde –1 até 1) – grau de veracidade.
  - **Discreta** (valores binários 0 e 1) – falso ou verdadeiro.

Estruturas das Redes Neurais Artificiais

- **REDES ACÍCLICAS** ou de alimentação direta (***feedforward***).
  - Representa uma função do seu estado atual, não existe nenhum estado interno além dos pesos sinápticos.
- **REDES CÍCLICAS** ou recorrentes (***feedback***).
  - Utiliza sua saída para realimentar suas próprias entradas.
  - Os níveis de ativação da rede formam um sistema dinâmico.
  - A resposta da rede a uma determinada entrada depende de seu estado inicial, que pode depender de entradas anteriores (memória de curto prazo).
  - São mais próximas do modelo do cérebro, mas são mais difíceis de compreender.

Estrutura de uma rede (*feedforward*)

- Normalmente os nós entre as camadas são completamente ligados.

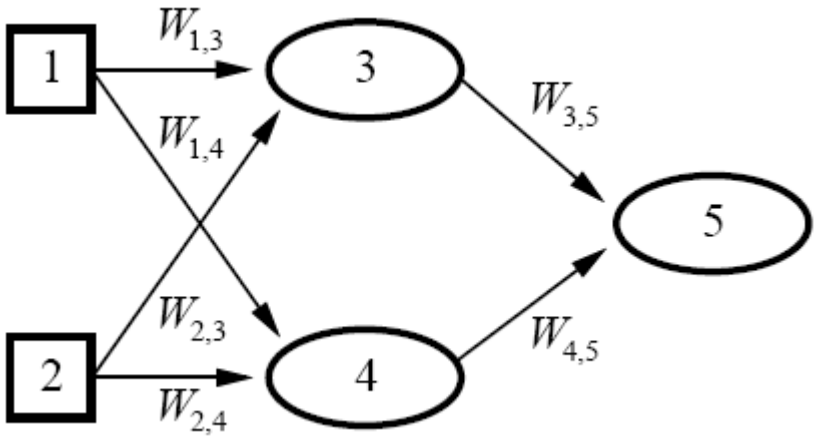


Exemplo: Uma rede de alimentação direta representa uma função de suas entradas

- Seja o vetor de entrada $x = (x_1, x_2)$, as ativações de entrada são definidas como $(a_1, a_2) = (x_1, x_2)$ e a rede calcula:

$$y_5 = g(W_{3,5}*y_3 + W_{4,5}*y_4)$$

$$y_5 = g(W_{3,5}*g(W_{1,3}*x_1 + W_{2,3}*x_2) + W_{4,5}*g(W_{1,4}*x_1 + W_{2,4}*x_2))$$



Redes de alimentação direta (*feedforward*)

- Normalmente são representadas em **CAMADAS**:
  - Os neurônios que recebem sinais de excitação são chamados de **camada de entrada** ou primeira camada.
  - Os neurônios que tem sua saída como saída da rede são a **camada de saída** ou última camada.
  - Os neurônios que não pertencem nem à camada de saída e nem a camada de entrada são chamados e neurônios internos (ou *hidden*) podendo se organizar em uma ou mais **camadas internas** (*hidden layers*).
  - Cada camada recebe apenas entradas de unidades situadas na camada imediatamente precedente.

Uma rede de alimentação direta representa uma função de suas entradas

$$y_5 = g(W_{3,5}*g(W_{1,3}*x_1 + W_{2,3}*x_2) + W_{4,5}*g(W_{1,4}*x_1 + W_{2,4}*x_2))$$

- A saída da rede como um todo é uma função de suas entradas.
- Os pesos da rede atuam como parâmetros dessa função.
- Ajustando os pesos, mudamos a função que a rede representa.
  - Essa é a forma de aprendizagem das Redes Neurais Artificias.

Perceptron Simples

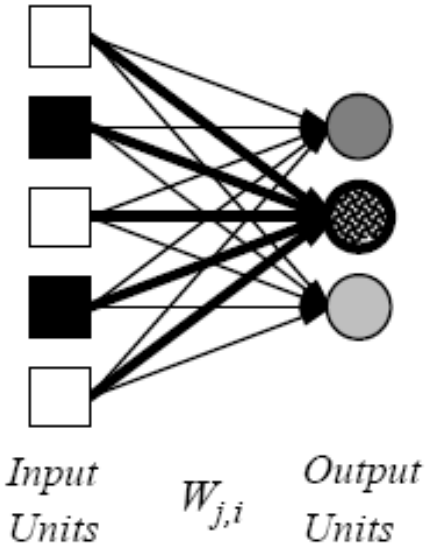
Rede de *Perceptron* Simples (PS)

- Considerado o primeiro algoritmo de Redes Neurais Artificiais (RNA), proposto por Frank Rosenblatt em 1958.
- A grande sacada de Rosenblatt foi associar uma regra de aprendizagem aos neurônios M-Ps.
- *Perceptron* é uma rede de neurônios que pode ter uma camada.

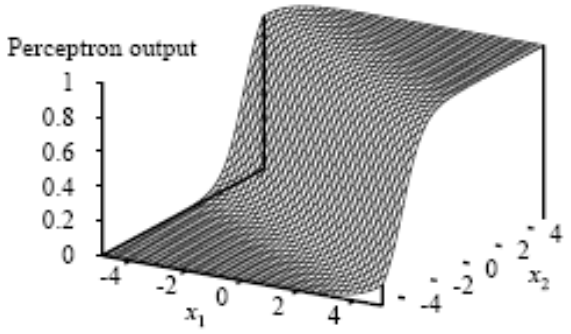
Rede de *Perceptron* Simples (PS)

- É uma rede com todas as entradas conectadas diretamente às saídas.
- Cada unidade de saída é independente das outras – cada peso afeta apenas uma das saídas.

Uma rede de *perceptrons* com três unidades de saída que compartilham cinco entradas



Saída de um *perceptron* de duas entradas com uma função de ativação sigmoide

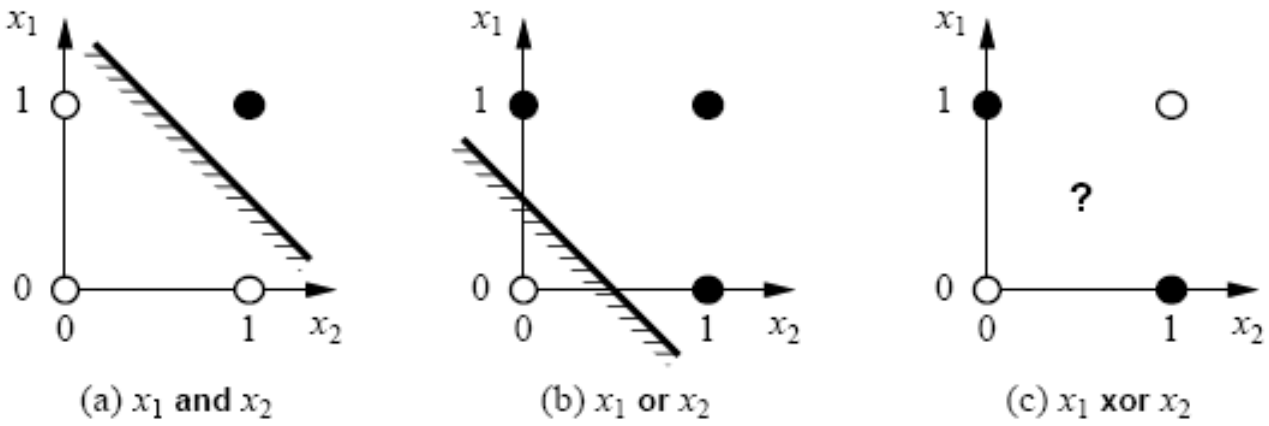


Rede de *Perceptron* Simples (PS)

- Uma rede com um única unidade de saída:
  - Com uma **função de ativação de limiar** podemos visualizar o *Perceptron* como uma representação de uma função booleana (E, OU, NOT, como vimos).
  - Pode representar algumas funções booleanas bastante "complexas" de maneira compacta.
    - **Exemplo**: **função maioria** com cada $W_j = 1$ e limiar $W_0 = n/2$

O que os *Perceptrons* não podem representar

- Podemos ver a equação do *Perceptron* de limiar como uma reta que separa as entradas.
- Por esta razão o ***Perceptron* é um separador linear**.
- Podendo representar apenas funções linearmente separáveis.



(a) x_1 and x_2 (b) x_1 or x_2 (c) x_1 xor x_2

Rede *Perceptron* Simples (PS)

- Costuma-se representar as variáveis de entradas e os pesos como vetores:

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{pmatrix} = \begin{pmatrix} -1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix} \qquad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{pmatrix} = \begin{pmatrix} \theta \\ w_1 \\ \vdots \\ w_p \end{pmatrix}$$

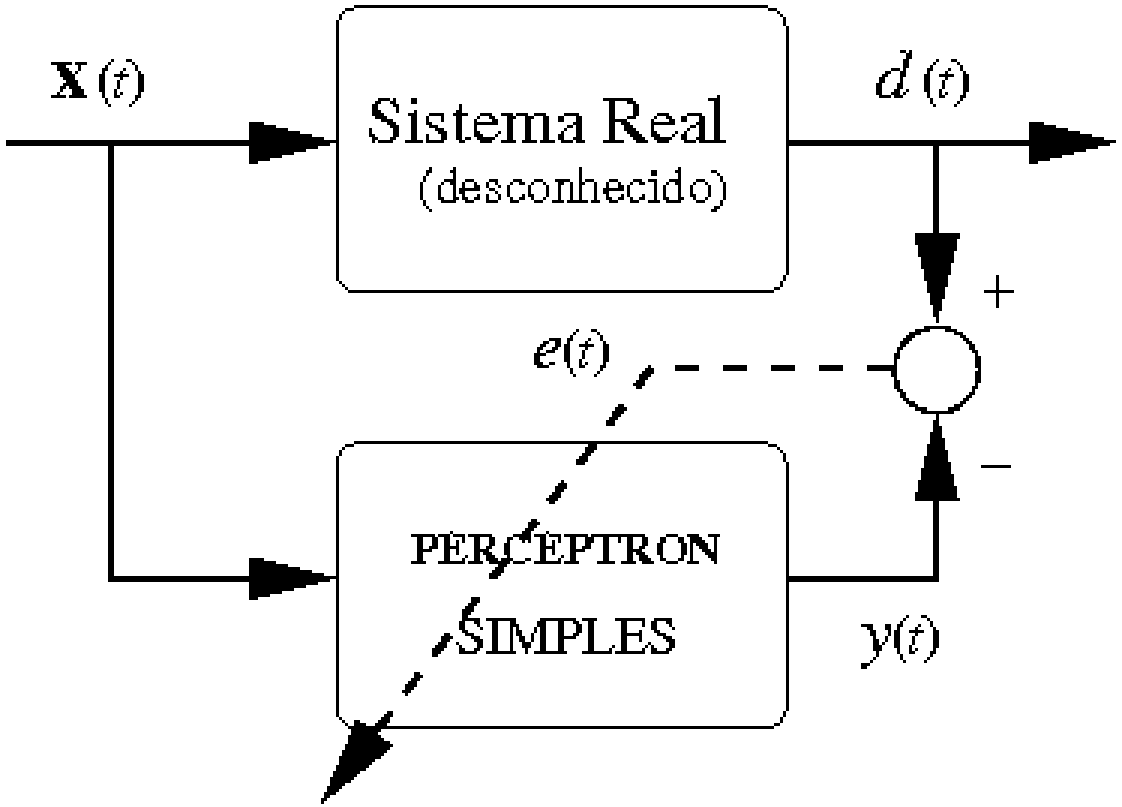
Regra de aprendizagem do *PS*

- A função vetorial da ativação (***u***) nos permitirá obter uma regra de aprendizagem para o neurônio M-P.
- **A aprendizagem se dará modificando os pesos e o limiar do neurônio** até conseguirmos solucionar o problema de interesse.
- A regra será controlada por dois fatores:
  - O **ERRO** entre a saída desejada (***d***) e a gerada (***y***).

    $$e = d - y$$

  - As informações fornecidas pelo **VETOR DE ENTRADA** (***x***).

Regra de aprendizagem do *PS*



Regra de aprendizagem do *PS*

- Uma regra de aprendizagem tem a seguinte forma:

$$w^{t+1} = w^t + \Delta w^t$$

  - w^t = conhecimento atual
  - Δw^t = incremento no conhecimento (informação adquirida)
  - w^{t+1} = novo conhecimento

Regra de aprendizagem do *PS*

- Para nosso modelo baseado nos neurônios M-P, teremos:

$$\Delta w^t = g(e^t, x^t)$$

  - isto é, o ganho de conhecimento será dado por uma função em relação ao erro encontrado com os valores de entrada fornecidos.

  - Analisando os possíveis valores de ***e***, temos:

    - Caso 1: $e = d\text{-}y$ = +1 *(d=1 e y=0)*

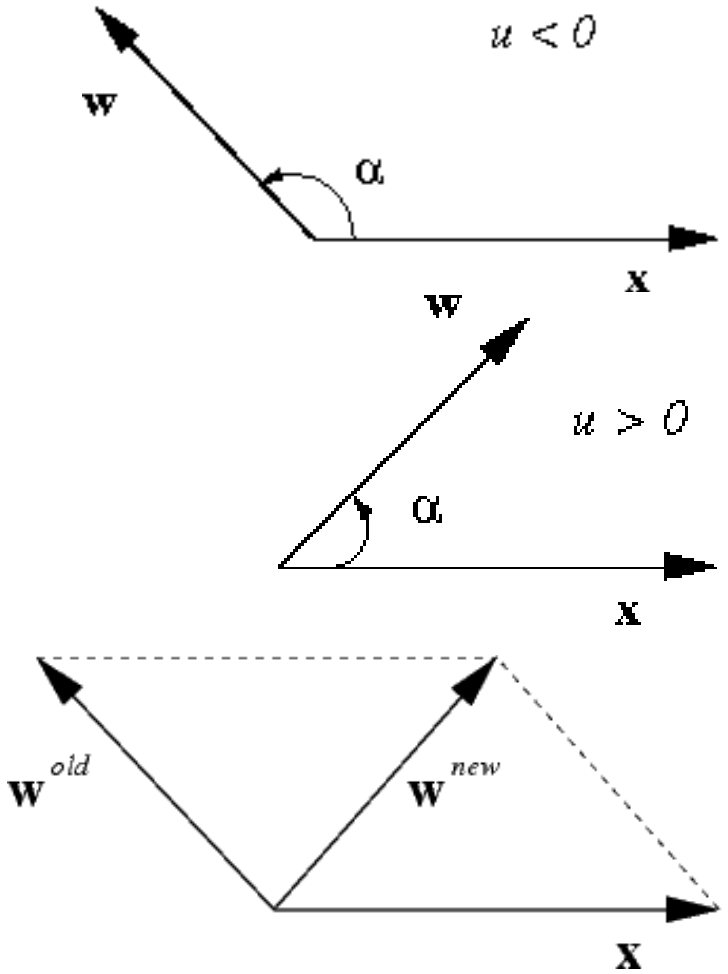
    - Caso 2: $e = d\text{-}y$ = -1 *(d=0 e y=1)*

    - Caso 3: $e = d\text{-}y$ = 0 *(d=1 e y=1) ou (d=0 e y=0)*

Regra de aprendizagem do *PS*

- *Caso 1, e=1*
  - *Situação ocorrida:*
    u < 0, y=0
  - *Situação desejada:*
    u>0, y=0
  - *Ajuste:*
    *Fazer o vetor **w** aproximar-se de **x***

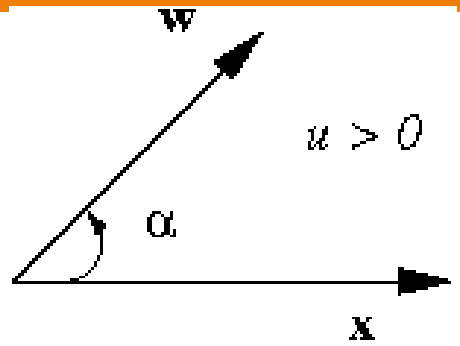
$$w^{t+1} = w^t + x^t$$

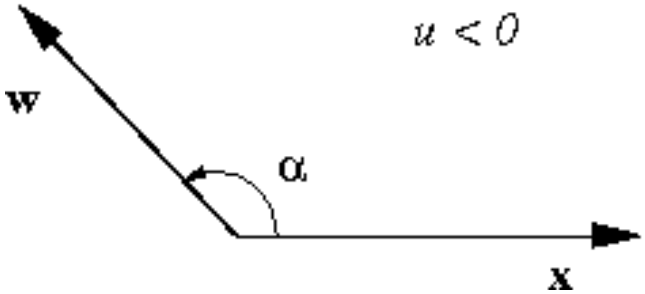


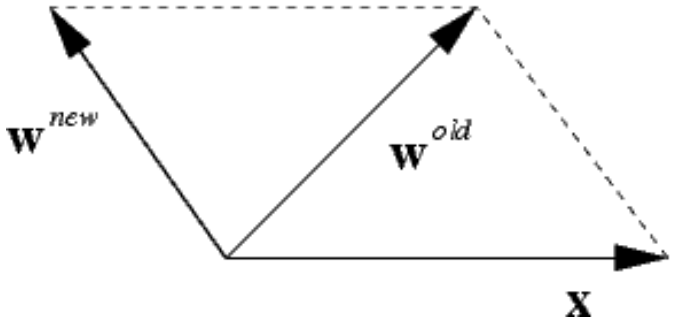
Regra de aprendizagem do *PS*

- *Caso 2, e=-1*
  - *Situação ocorrida:*
    u > 0, y=+1
  - *Situação desejada:*
    u<0, y=0
  - *Ajuste:*
    *Fazer o vetor **w** afastar-se de **x***

$$w^{t+1} = w^t - x^t$$



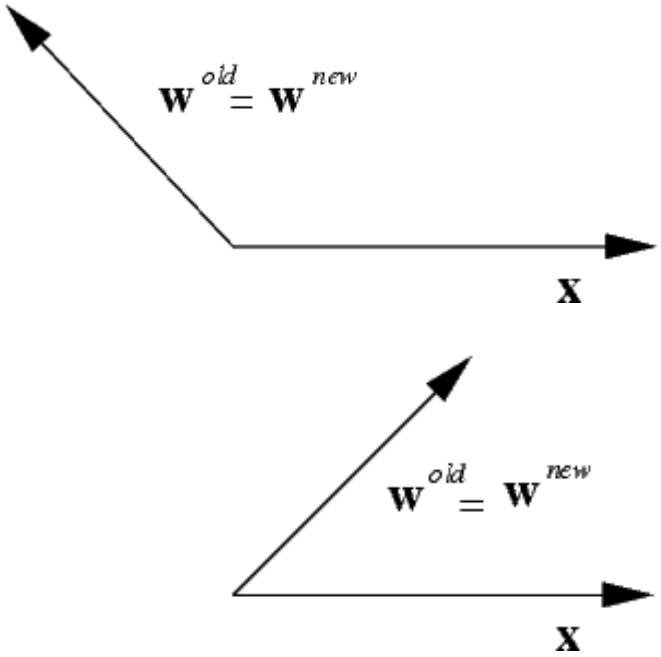




Regra de aprendizagem do *PS*

- *Caso 3, e=0*
  - *A) Situação ocorrida = Situação desejada*
    u > 0, y = +1
  - *B) Situação ocorrida = Situação desejada*
    u < 0, y = 0
  - *Ajuste:*
    não é necessário

$$w^{t+1} = w^{t}$$



Regra de aprendizagem do *PS*

- Para facilitar a tomada de decisão, vamos combinar os 3 casos em uma mesma equação, trazendo o erro (***e***):

$$w^{t+1} = w^t - e^t . x^t$$

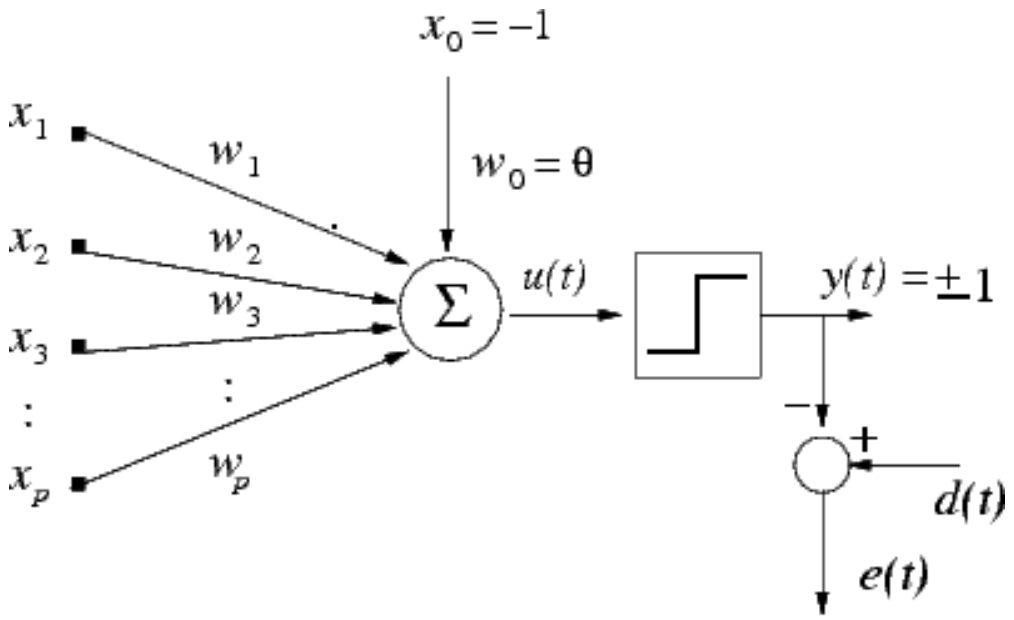
- Para que o processo também seja mais estável, costuma-se introduzir uma **TAXA DE APRENDIZAGEM**, denominado aqui por η:

$$w^{t+1} = w^t - \eta . e^t . x^t$$

na qual $0 < \eta \leq 1$.

Perceptron Simples (PS)

- Resumindo, o modelo do *Perceptron* Simples:
  - Neurônio M-P + regra de aprendizagem
  - $w^{t+1} = w^t - \eta.e^t.x^t$



Época de Treinamento

- O aprendizado é resultado da apresentação repetitiva de todas as amostras do conjunto de treinamento.
- Cada apresentação de todo o conjunto de treinamento é denominada **ÉPOCA**.
- O processo de aprendizagem é repetido época após época, até que um **CRITÉRIO DE PARADA** seja satisfeito.
- É recomendável que a **ordem de apresentação das amostras seja aleatória** de uma época para outra. Isso tende a fazer com que o ajuste de pesos tenha um caráter estocástico ao longo do treinamento.

Algoritmo *Perceptron* Simples (PS)

Inicialização (t=0)

- Define-se valor de η
- Inicia vetor w(0) com valores aleatórios (ou nulos)

Ciclo de Treinamento

- Seleciona vetor de entrada x^0
- Calcula ativação, u^t
- Calcula saída, y^t
- Calcula o erro: $e^t = d^t - y^t$
- Ajusta pesos via regra de aprendizagem
- Verifica critério de parada
  - Se atendido, finalizar treinamento
  - Senão, t = t+1, volta para início do ciclo de treinamento

Exemplo

- Vamos aplicar *Perceptron* Simples para fazer um neurônio M-P aprender o comportamento da porta lógica OR
  - Os vetores de entrada, ***x*** serão as 4 combinações:
    - $x(-1, x_1, x_2)$
  - Inicialização: (t=0)
    - vetor ***w*** com zeros: $w(\theta, w_1, w_2) = w(0,0,0)$
    - escolhemos $\eta = 0{,}5$

Porta OR

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	1

Exemplo

t=1: Calcular saída para $\mathbf{w}(1) = [0 \;\; 0 \;\; 0]$ e $\mathbf{x}(1) = [-1 \;\; 0 \;\; 0]$.

$$u(1) = (0)(-1) + (0)(0) + (0)(0) = 0 \Rightarrow y(1) = 0,\ e(1) = 0.$$

$$\mathbf{w}(2) = \mathbf{w}(1) + \eta\, e(1)\mathbf{x}(1) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + (0{,}5)(0)\begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} = \mathbf{w}(1)$$

t=2: Calcular saída para $\mathbf{w}(2) = [0 \;\; 0 \;\; 0]$ e $\mathbf{x}(2) = [-1 \;\; 0 \;\; 1]$.

$$u(2) = (0)(-1) + (0)(0) + (0)(1) = 0 \Rightarrow y(2) = 0,\ e(2) = 1$$

$$\mathbf{w}(3) = \mathbf{w}(2) + \eta\, e(2)\mathbf{x}(2) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + (0{,}5)(1)\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -0{,}5 \\ 0 \\ 0{,}5 \end{bmatrix}$$

Exemplo

t=3: Calcular saída para $\mathbf{w}(3) = [-0{,}5 \;\; 0 \;\; 0{,}5]$ e $\mathbf{x}(3) = [-1 \;\; 1 \;\; 0]$.

$u(3) = (-0{,}5)(-1) + (0)(1) + (0{,}5)(0) = 0{,}5 \Rightarrow y(3) = 1, e(1) = 0.$

$\mathbf{w}(4) = \mathbf{w}(3)$

t=4: Calcular saída para $\mathbf{w}(4) = [-0{,}5 \;\; 0 \;\; 0{,}5]$ e $\mathbf{x}(4) = [-1 \;\; 1 \;\; 1]$.

$u(4) = (-0{,}5)(-1) + (0)(1) + (0{,}5)(1) = 1 \Rightarrow y(4) = 1, e(4) = 0.$

$\mathbf{w}(5) = \mathbf{w}(4)$

t=5: Calcular saída para $\mathbf{w}(5) = [-0{,}5 \;\; 0 \;\; 0{,}5]$ e $\mathbf{x}(5) = [-1 \;\; 0 \;\; 0]$.

$u(5) = (-0{,}5)(-1) + (0)(0) + (0{,}5)(0) = 0{,}5 \Rightarrow y(5) = 1, e(5) = -1.$

$$\mathbf{w}(6) = \mathbf{w}(5) + \eta\, e(5)\mathbf{x}(5) = \begin{bmatrix} -0{,}5 \\ 0 \\ 0{,}5 \end{bmatrix} + (0{,}5)(-1)\begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0{,}5 \end{bmatrix}$$

Exemplo

t=6: Calcular saída para $\mathbf{w}(6) = [0 \;\; 0 \;\; 0{,}5]$ e $\mathbf{x}(6) = [-1 \;\; 0 \;\; 1]$.

$u(6) = (0)(-1) + (0)(0) + (0{,}5)(1) = 0{,}5 \Rightarrow y(6) = 1, e(6) = 0.$

$\mathbf{w}(7) = \mathbf{w}(6)$

t=7: Calcular saída para $\mathbf{w}(7) = [0 \;\; 0 \;\; 0{,}5]$ e $\mathbf{x}(7) = [-1 \;\; 1 \;\; 0]$.

$u(7) = (0)(-1) + (0)(1) + (0{,}5)(0) = 0 \Rightarrow y(7) = 0, e(7) = 1.$

$$\mathbf{w}(8) = \mathbf{w}(7) + \eta e(7)\mathbf{x}(7) = \begin{bmatrix} 0 \\ 0 \\ 0{,}5 \end{bmatrix} + (0{,}5)(1)\begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ 0{,}5 \end{bmatrix}$$

t=8: Calcular saída para $\mathbf{w}(8) = [-0{,}5 \;\; 0{,}5 \;\; 0{,}5]$ e $\mathbf{x}(8) = [-1 \;\; 1 \;\; 1]$.

$u(8) = (-0{,}5)(-1) + (0{,}5)(1) + (0{,}5)(1) = 0{,}5 \Rightarrow y(8) = 1, e(8) = 0.$

$\mathbf{w}(9) = \mathbf{w}(8)$

Exemplo

t=9: Calcular saída para $\mathbf{w}(9) = [-0{,}5\ 0{,}5\ \ 0{,}5]$ e $\mathbf{x}(9) = [-1\ \ 0\ \ 0]$.

$u(9) = (-0{,}5)(-1) + (0{,}5)(0) + (0{,}5)(0) = 0{,}5 \Rightarrow y(9) = 1,\ e(9) = -1.$

$$\mathbf{w}(10) = \mathbf{w}(9) + \eta\, e(9)\, \mathbf{x}(9) = \begin{bmatrix} -0{,}5 \\ 0{,}5 \\ 0{,}5 \end{bmatrix} + (0{,}5)(-1) \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0{,}5 \\ 0{,}5 \end{bmatrix}$$

t=10: Calcular saída para $\mathbf{w}(10) = [0\ \ 0{,}5\ \ 0{,}5]$ e $\mathbf{x}(10) = [-1\ \ 0\ \ 1]$.

$u(7) = (0)(-1) + (0{,}5)(0) + (0{,}5)(1) = 0{,}5 \Rightarrow y(10) = 1,\ e(10) = 0.$

$\mathbf{w}(11) = \mathbf{w}(10)$

t=11: Calcular saída para $\mathbf{w}(11) = [0\ \ 0{,}5\ \ 0{,}5]$ e $\mathbf{x}(11) = [-1\ \ 1\ \ 0]$.

$u(11) = (0)(-1) + (0{,}5)(1) + (0{,}5)(0) = 0{,}5 \Rightarrow y(11) = 1,\ e(11) = 0.$

$\mathbf{w}(12) = \mathbf{w}(11)$

Exemplo

t=12: Calcular saída para $\mathbf{w}(12) = [0 \;\; 0{,}5 \;\; 0{,}5]$ e $\mathbf{x}(12) = [-1 \;\; 1 \;\; 1]$.

$u(12)=(0)(-1)+(0{,}5)(1) + (0{,}5)(1) = 1 \Rightarrow y\,(12)= 1, e(12)=0.$

$\mathbf{w}(13)=\mathbf{w}(12)$

t=13: Calcular saída para $\mathbf{w}(13) = [0 \;\; 0{,}5 \;\; 0{,}5]$ e $\mathbf{x}(13) =[-1 \;\; 0 \;\; 0]$.

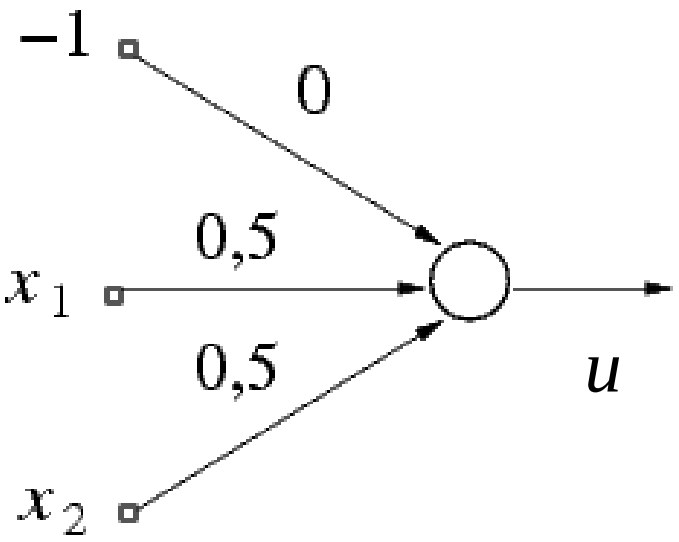
$u(13) = (0)(-1) + (0{,}5)(0) + (0{,}5)(0) = 0 \Rightarrow y(13) = 0, e(13)=0.$

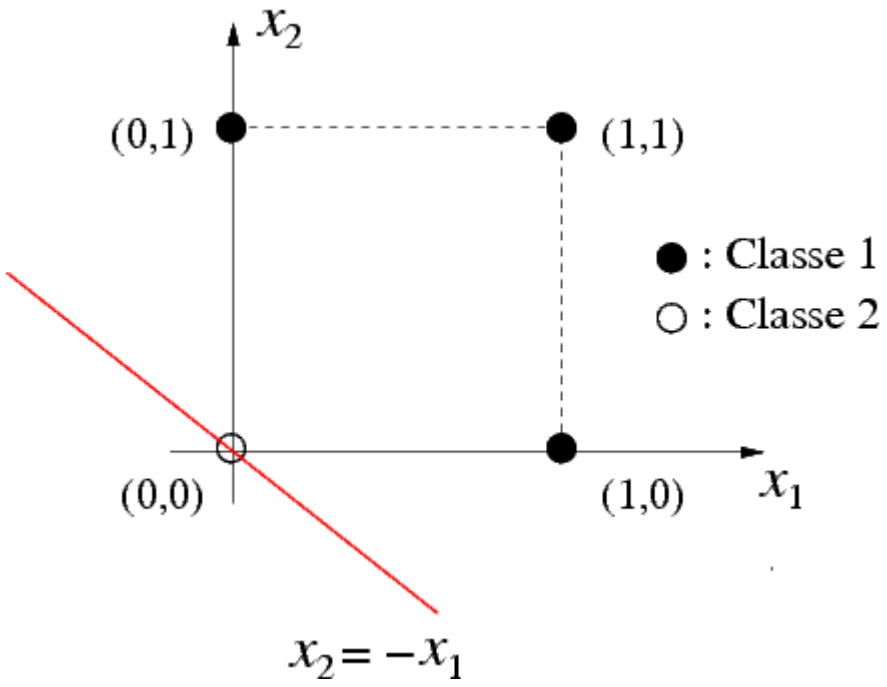
$\mathbf{w}(14)=\mathbf{w}(13)$

Fim do Treinamento!

Exemplo

- Solução encontrada:





- **Não é a melhor solução**. Provavelmente se os pesos iniciais não fossem nulos, uma outra reta resultado poderia ter sido apresentada.

Regra de aprendizado de ***Widrow-Hoff*** (ou ***Delta***) para *Perceptrons* de limiar

- Ignora a função de ativação fazendo $g(\mathbf{X}) = \mathbf{X}$.

$$u \leftarrow \sum_{j=0}^{n} w_j x_j$$

$$e \leftarrow d - g(u)$$

$$w_j{}^{t+1} \leftarrow w_j{}^{t} + \eta * e * x_j$$

- Uma rápida análise:
  - Quando o erro é positivo, a saída da rede é pequena demais, e os pesos devem ser aumentados.
  - Quando o erro é negativo, a saída da rede é grande demais e os pesos devem ser diminuídos.
- Depois de encontrar os pesos que minimizam o erro quadrático, podemos reinserir a função de limiar para produzir os valores necessários.

Aprendizagem de um *Perceptron* com **função de ativação sigmóide**

- O **erro quadrático** para um único exemplo de treinamento ***X*** e saída verdadeira ***d*** é escrito como:

$$e = \frac{1}{2}(d - g(u))^2$$

- A **descida de gradiente** é usada para reduzir o **erro quadrático**. Calculando a derivada parcial de ***e*** em relação a cada peso temos:

$$\frac{\partial e}{\partial w_j} = -e * g'(u) * x_j$$

- **Atualizamos o peso** como a seguir:

$$w_j{}^{t+1} \leftarrow w_j{}^{t} + \eta * g'(u) * x_j$$

na qual η é a taxa de aprendizagem.

Regra de aprendizado **Delta Generalizada** para *Perceptrons* com função sigmoide

- A derivada da função sigmoide $g'(u) = g.(1-g)$ resultando em:

$$u \leftarrow \sum_{j=0}^{n} w_j x_j$$

$$e \leftarrow y - g(u)$$

$$w_j{}^{t+1} \leftarrow w_j{}^{t} + \eta * e * g(u)(1 - g(u)) * x_j$$

- Para *Perceptrons* de limiar onde $\boldsymbol{g'(u)}$ é indefinido, a regra do *Perceptron* original simplesmente omite este valor.

Algoritmo de aprendizagem para *Perceptron*

função APRENDIZAGEM-DE-PERCEPTRON (exemplos, rede) retorna: um vetor de pesos

entrada: exemplos, um conjunto de exemplos, cada um com entrada $\boldsymbol{X} = x_1,..., x_n$ e ***y***

saída: rede, um *Perceptron* com pesos $\boldsymbol{W_{j,j}} = 0...n$ e função de ativação ***g*** (diferenciável)

repita

para cada ***e*** em exemplos ***faça***

$$u \leftarrow \sum_{j=0}^{n} w_j x_j$$

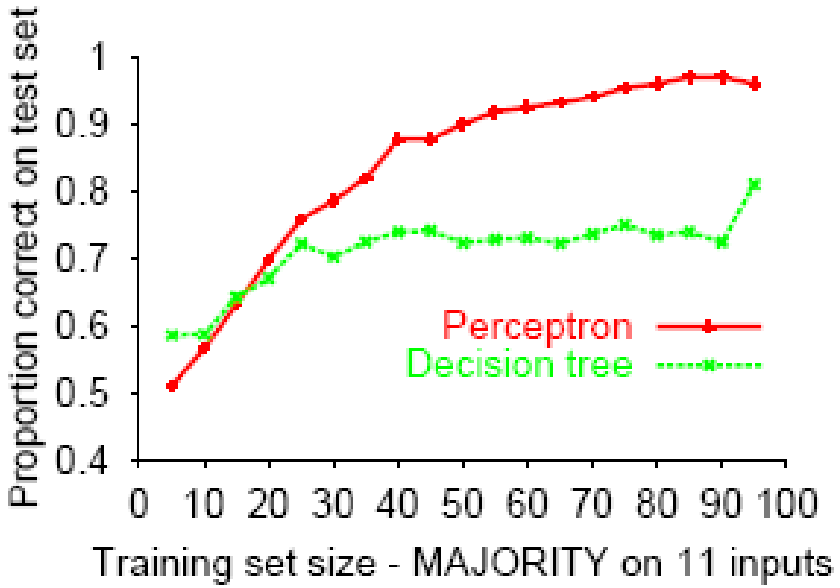
$$e \leftarrow d - g(u)$$

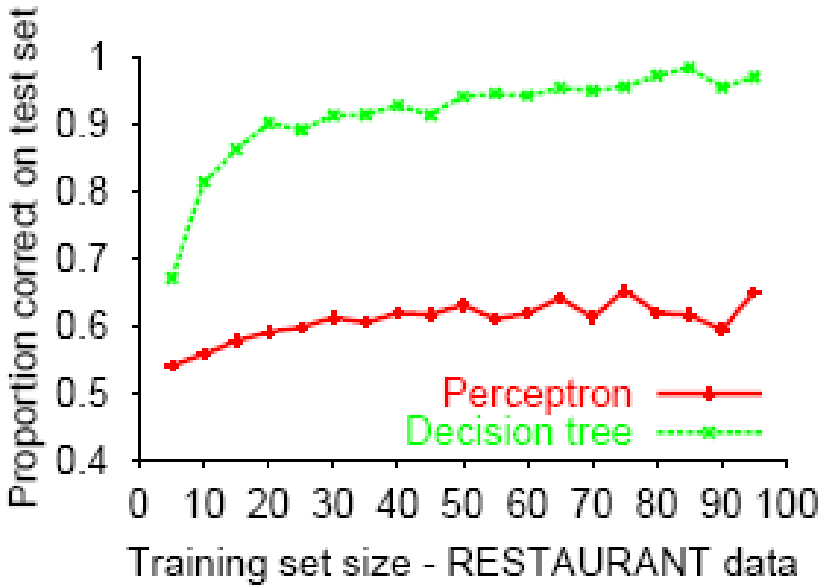
$$w_j{}^{t+1} \leftarrow w_j{}^{t} + \eta * g'(u) * x_j$$

até que algum critério de parada seja satisfeito

retornar HIPÓTESE-DA-REDE-NEURAL(rede)

Curva de aprendizagem
Perceptrons X Árvores de Decisão



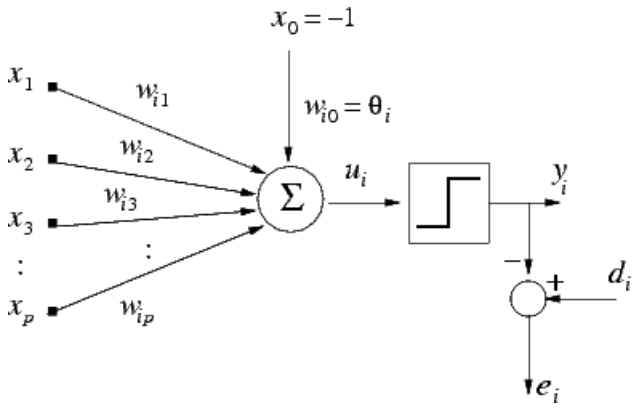


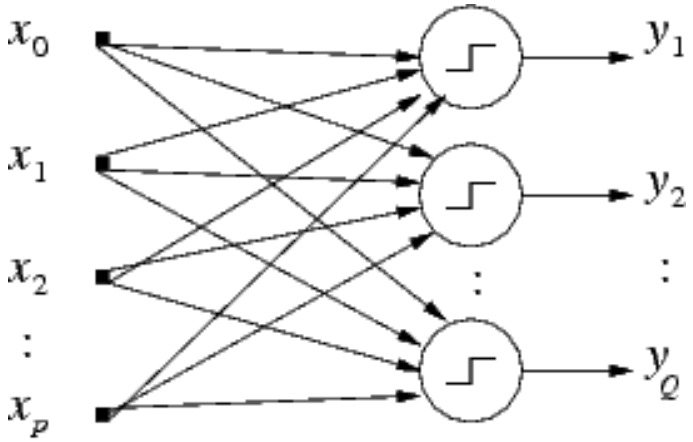
Exercício

- Considere um *Perceptron* com duas unidades de entrada com valores reais e uma unidade de saída.
  - Todos os pesos iniciais e o desvio (*bias*) são iguais a 0,5.
  - A saída deve ser 1 para os valores de entrada $x_1 = 0,7$ e $x_2 = -0,3$.
- Mostre como a regra de aprendizado de *Widrow-Hoff* altera os pesos do *Perceptron* para que a rede tenha a saída correta para este exemplo.
- Considere $\eta = 0,3$.

Perceptron Simples (PS) – Múltiplas classes

- Um único neurônio M-P categoriza apenas duas classes de dados. Em **problemas com múltiplas classes, deve-se utilizar vários neurônios em paralelo**.





- O funcionamento de cada neurônio individualmente é o mesmo, pois cada neurônio tem seu próprio vetor de pesos e respectiva regra de aprendizagem.

Perceptron Simples (PS) – Múltiplas classes

- Como saber quantos neurônios (***Q***) utilizar?

- **MÉTODO 1** – Codificação Binária simples
  - Se tenho *C* classes, então *Q* é o maior inteiro igual a ou menor que $\sqrt{C}$.
    - **Exemplo**: Se C = 6 classes, então Q > 2,45 = 3.
  - Os vetores de saída desejados são construídos do seguinte modo:
    - Classe 1: d(0 0 1), Classe 2: d(0 1 0), Classe 3: d(0 1 1), ..., Classe 6: d(1 1 0)

Perceptron Simples (PS) – Múltiplas classes

- Como saber quantos neurônios (***Q***) utilizar?

- **MÉTODO 2**: 1-out-of-Q
  - Se tenho *C* classes, então *Q*=*C*.
    - **Exemplo**: Se C = 6 classes, então Q = 6.
  - Neste método apenas uma das componentes do vetor de saídas desejadas tem valor igual a 1, i.e. Os vetores d são ortogonais.
  - Classe 1: d(1 0 0 0 0 0), Classe 2: d(0 1 0 0 0 0)
    Classe 3: d(0 0 1 0 0 0), Classe 4: d(0 0 0 1 0 0)
    Classe 5: d(0 0 0 0 1 0), Classe 6: d(0 0 0 0 0 1)

Perceptron Simples (PS) - Algumas Dicas

- Usar taxa de aprendizagem pequena, η=0,01 ou η=0,1 por exemplo.
- Usar valores de saída $y_i \in \{-1,+1\}$, em vez de $y_i \in \{0,+1\}$.
- Mudar a ordem de apresentação dos vetores de treinamento a cada época de treinamento, tornando-a aleatória.
- Usar o método 2 para determinar o número de neurônios *(Q)* e a representação dos vetores de saídas desejadas (***d***).
- Normalizar os vetores de entrada se as variáveis apresentarem ordens de grandeza muito díspares.

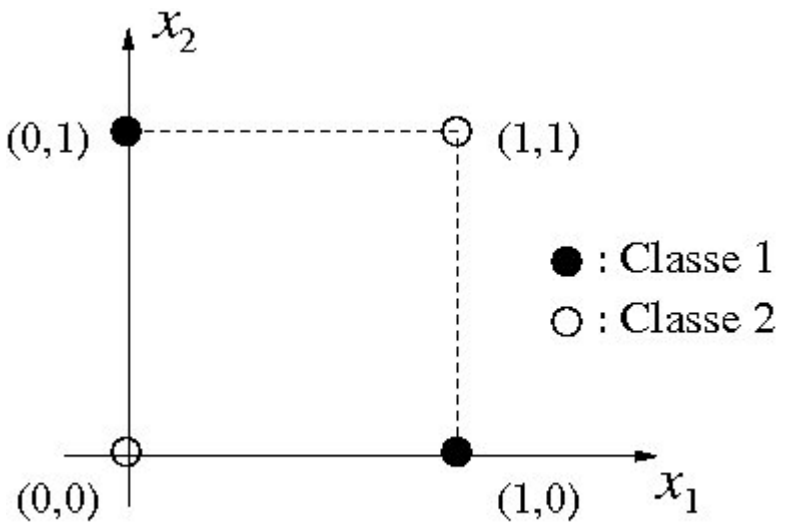
Redes de Multiplas Camadas

Redes Multicamadas (*FeedForward*)

- Voltando ao problema de implementar a porta XOR com uma rede *Perceptron*.

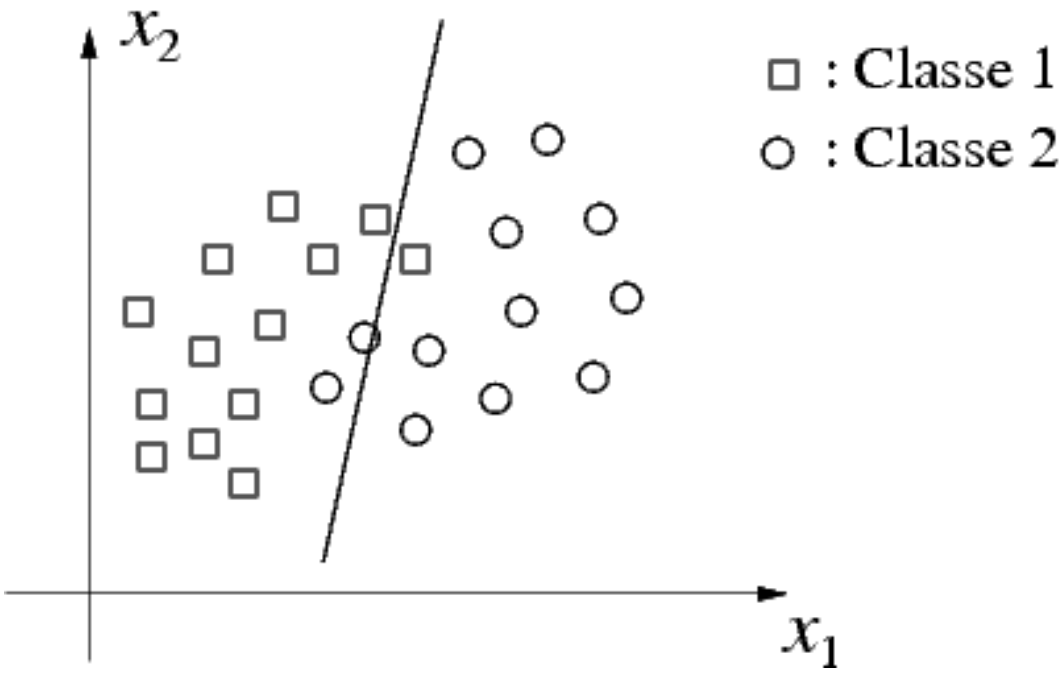
Porta XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

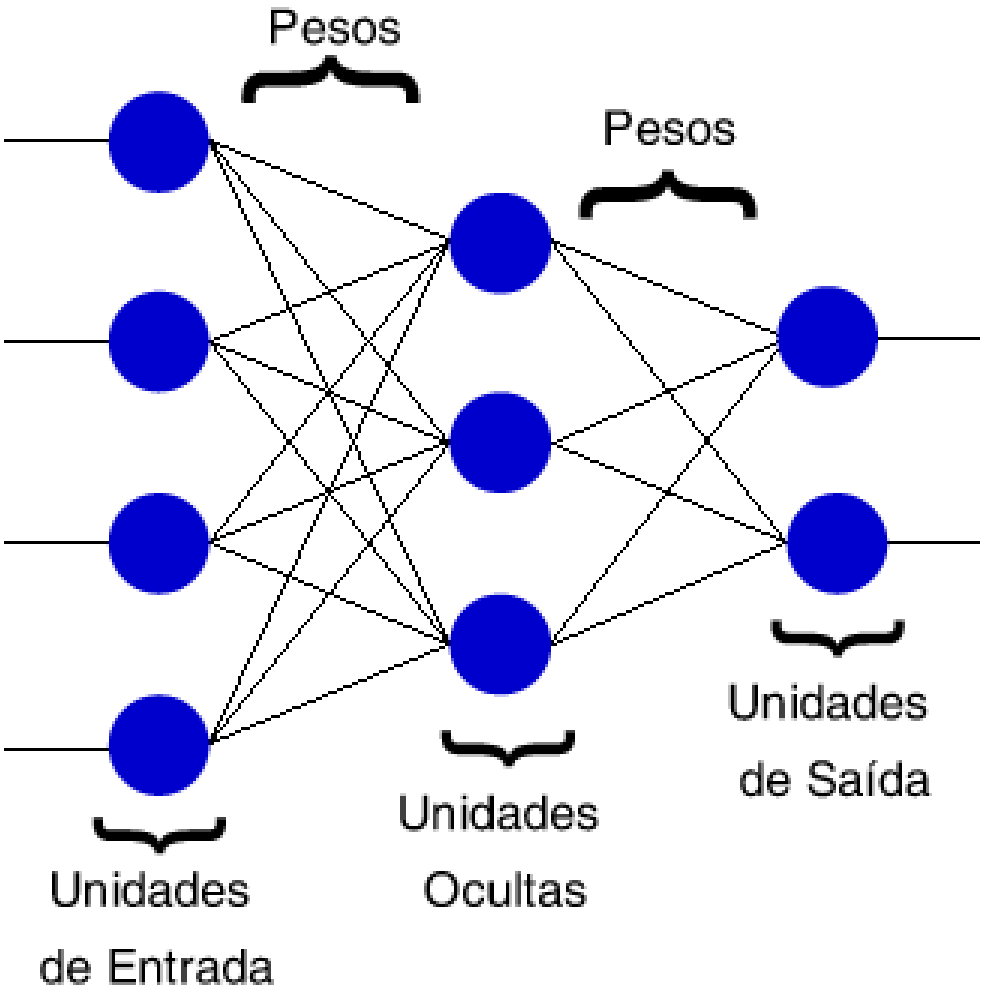


Redes Multicamadas (*FeedForward*)

- Problemas linearmente inseparáveis exigirão a utilização de mais neurônios.

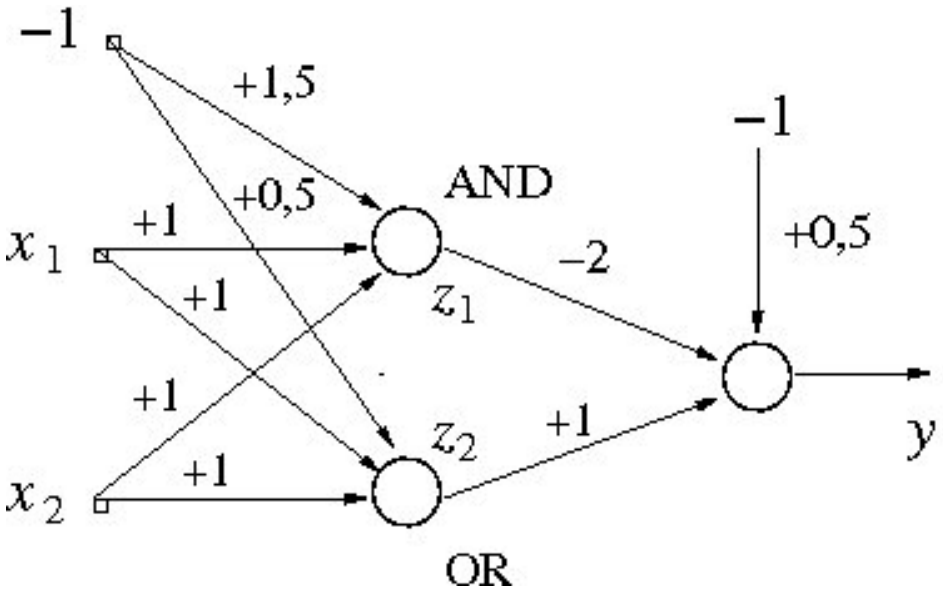


Redes Multicamadas (*FeedForward*)



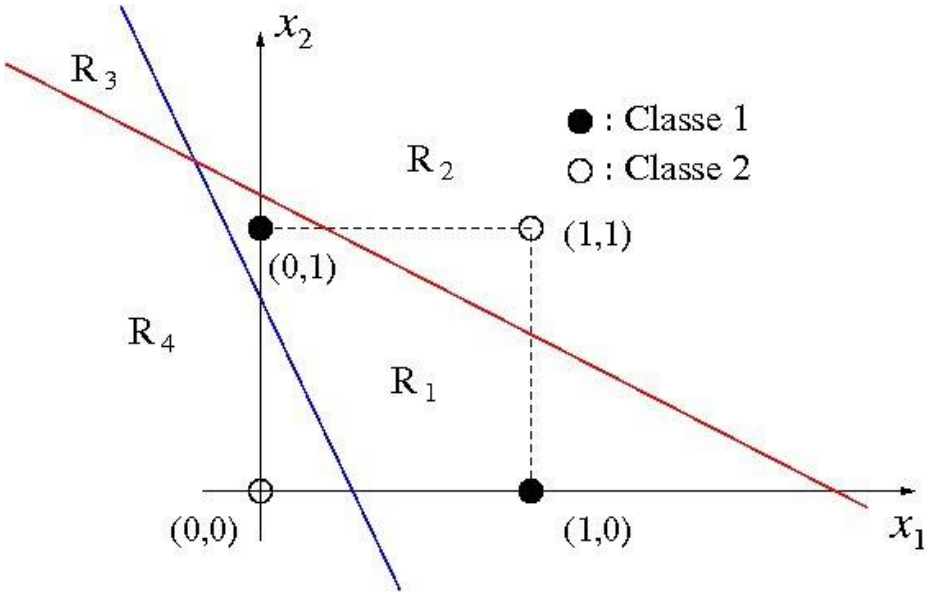
Redes Multicamadas (*FeedForward*)

- Desta forma, o problema da porta XOR pode ser resolvido com três neurônios, dispostos em duas camadas, como uma possível solução apresentada a seguir.



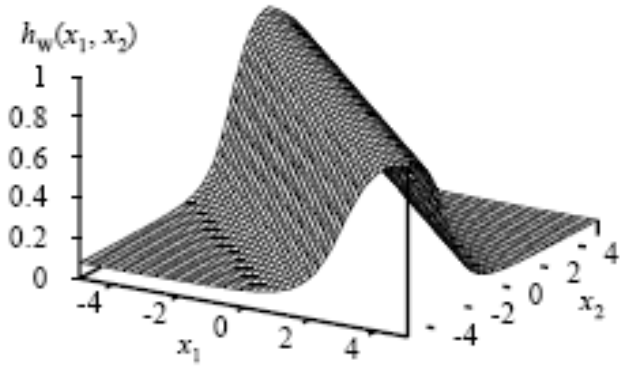
Redes Multicamadas (*FeedForward*)

- Redes com várias camadas conseguem resolver problemas mais complexos, pois ampliam o poder de separação das instâncias do problema.



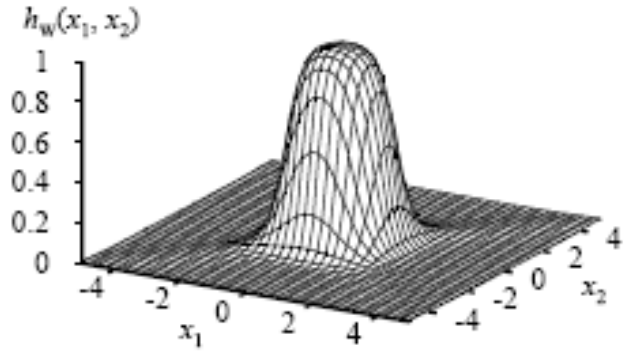
Redes Multicamadas (*FeedForward*)

- A vantagem em **adicionar camadas ocultas** é que ela **aumenta o espaço de hipóteses** que a rede pode representar.



Combinação de duas funções de limiar opostas para obtener um cume.

(Uma camada oculta que combina a saída de dois *Perceptrons*)



Combinação de dois cumes para formar uma coluna.

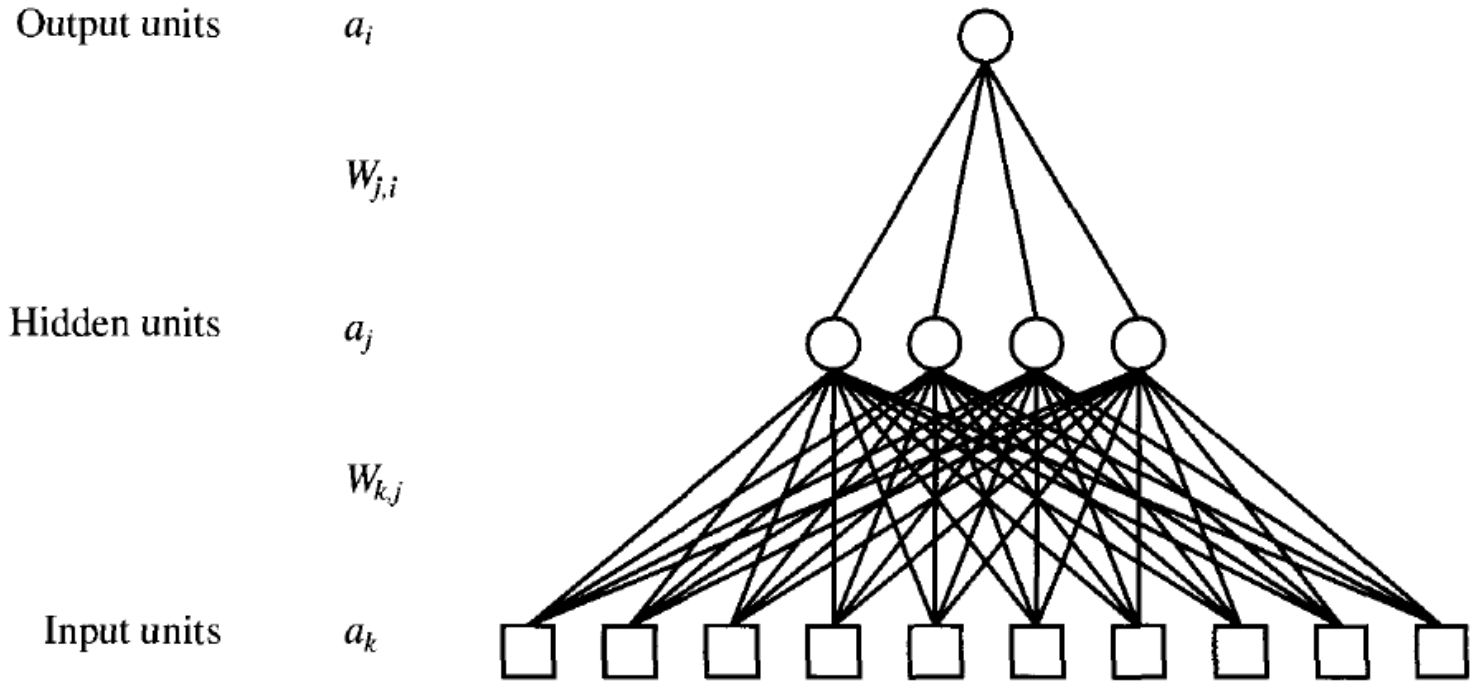
(Uma segunda camada oculta que combina a saída da camada anterior)

Redes Multicamadas (*FeedForward*)

- Requer no mínimo 3 camadas:
  - **DE ENTRADA**: recebe as entradas.
  - **OCULTA** ou intermediária: processam os dados.
  - **DE SAÍDA**: apresenta os resultados da rede.
- Os nós da camada de entrada se conectam somente com os nós da camada oculta.
- E os nós da camada oculta somente com os nós da camada de saída.

Exemplo: Redes Multicamadas (FeedForward)

- Uma rede neural de várias camadas, com uma camada oculta e 10 entradas, adequada para o problema do restaurante.



Aprendizagem em redes multicamadas

- Emprega o algoritmo de aprendizagem ***BACKPROPAGATION***:
  - É o mais difundido em redes multicamadas.
  - Aprendizado supervisionado.
  - Durante a **operação** (uso) da rede **nenhuma informação é retropropagada**.
  - Na fase de **aprendizagem** o **sinal de erro da saída é retropropagado** pela rede modificando os pesos sinápticos para minimizar o erro.
  - Depois do treinamento o processo de aprendizado se encerra e os pesos são armazenados.
- Muitas vezes chamadas de redes *Backpropagation*, mas o termo *Backpropagation* se refere ao método de aprendizagem da rede.

O algoritmo de aprendizado *BackPropagation*

- Emprega duas fases:
  - ***FEED-FORWARD*** → as **entradas** se propagam pela rede, da camada de entrada até a camada de saída.
  - ***FEED-BACKWARD*** → os **erros** se propagam na direção contrária ao fluxo de dados, indo da camada de saída até a primeira camada escondida.
- É repetido até que, para todos os processadores da camada de saída e para todos os padrões de treinamento, o erro seja menor do que o especificado.

O algoritmo de aprendizado *BackPropagation*

- É semelhante ao algoritmo de aprendizagem de *Perceptron*.
- Mas:
  - Podemos ter várias saídas, e assim temos um vetor de saída ao invés de um único valor.
  - Enquanto o erro da camada de saída é claro, o erro nas camadas ocultas parece misterioso.
    - Os dados de treinamento não informam qual valor a saída dos nós ocultos devem ter.
  - Então fazemos uma propagação de retorno do erro da camada de saída para as camadas ocultas.

O algoritmo de aprendizado *BackPropagation* – atualização dos pesos

- Na camada de saída a regra de atualização dos pesos é idêntica à do *Perceptron*.

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

- Seja $\boldsymbol{Err_i}$ o i-ésimo componente do vetor de erro $\boldsymbol{y - g(u)}$ e seja $\boldsymbol{\Delta_i}$ o erro modificado, logo:

$$\Delta_i = Err_i \times g'(in_i)$$

- Para atualizar o erro entre as unidades de entrada e as unidades ocultas, precisamos definir uma quantidade análoga ao termo de erro para os nós de saída:

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i \ .$$

O algoritmo de aprendizado *BackPropagation* – atualização dos pesos

- Assim efetuamos a propagação de retorno do erro.
  - O nó oculto ***j*** é responsável por alguma fração do erro Δ_i em cada um dos nós de saída aos quais ele se conecta.

- A regra de atualização dos pesos entre a entrada e a camada oculta é quase idêntica à regra de atualização para a camada de saída:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

O algoritmo de aprendizado *BackPropagation*

- Resumo do processo de propagação de retorno:
  - Calcular os valores Δ_j para as unidades de saída usando o erro observado.
  - Começando na camada de saída, repetir as etapas a seguir para cada camada da rede, até ser alcançada a camada oculta conectada à camada de entrada:
    - Propagar os valores Δ_j de volta até a camada anterior.
    - Atualizar os pesos entre as duas camadas.

O algoritmo de aprendizado *BackPropagation*

- Para cada exemplo no conjunto de treino, faça:
  - Propague a entrada pela rede e obtenha a saída ***y*** de cada unidade de saída da rede.
  - Propague os erro de volta pela rede:
    - Para cada unidade ***i*** de saída calcule seu erro em termos de Δ_i:

$$\Delta_i = (y_i - a_i) * g'(in_i)$$

    - Para cada unidade escondida ***j***, calcule seu erro em termos de Δ_j :

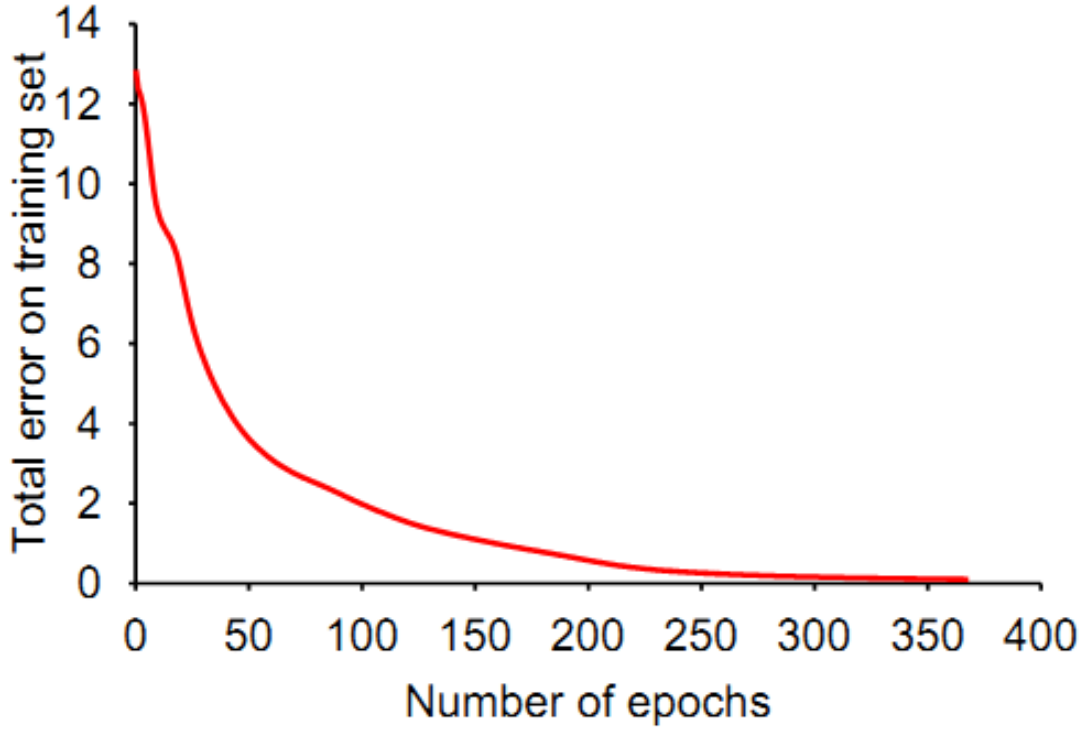
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

    - Atualize cada um dos pesos da rede:

$$W_{k,j} = W_{k,j} + \alpha * a_k * \Delta_j$$

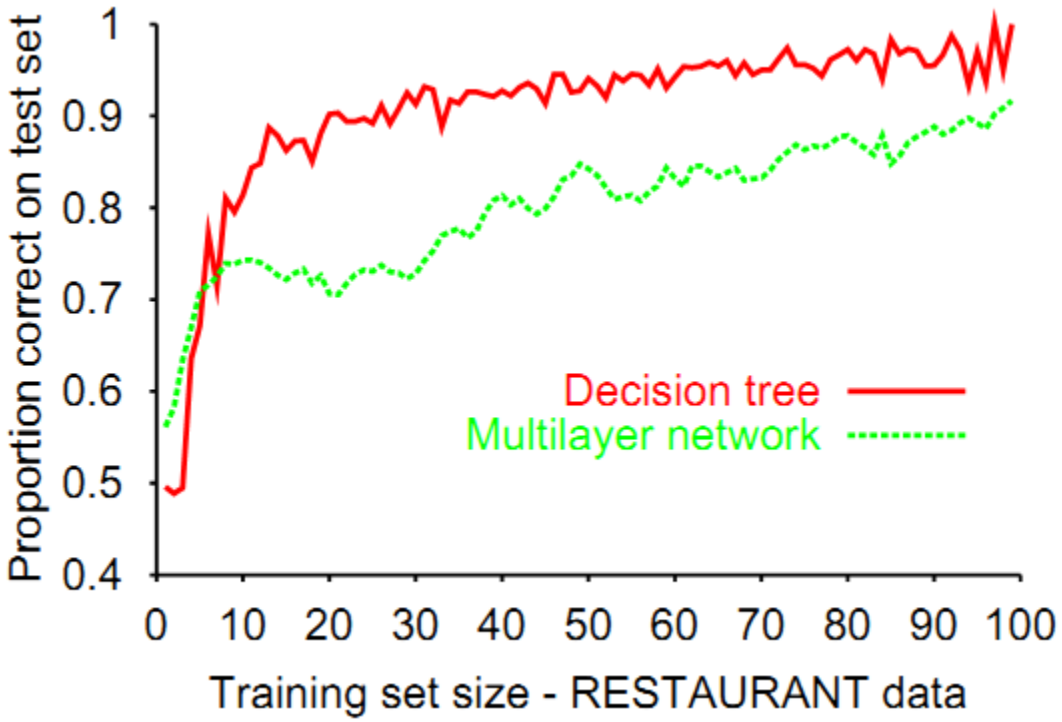
Exemplo

- Curva de treinamento para 100 exemplo do restaurante: acha ajuste exato.



Exemplo

- Curva de aprendizagem para rede neural com 4 unidades na camada oculta.



O processo de treinamento

- Durante o treinamento espera-se que o erro diminua de forma que os pesos w_{ij} alcancem melhores valores (mínimo global).
- Mas a rede pode entrar em um **estado de paralisia** causado pelo próprio processo de otimização de erros pelo gradiente.
  - w_{ij} é deslocado em uma região de gradiente pequeno (**mínimo local**).
  - e o treinamento praticamente para onde não é desejável.
- Pode-se usar a metaheurística de *Simulated Annealing* para fugir deste problema.

Monitoração do processo de aprendizagem

- É importante fazer a monitoração do processo durante o treinamento para evitar a **SUPERADAPTAÇÃO**.
  - Situação em que a rede perde sua capacidade de generalização (memoriza os pares de entrada/saída).
- Para reduzir a superadaptação:
  - Acompanhar o desempenho da rede sobre um conjunto separado de teste.
  - Depois de passar por um conjunto de treino, o conjunto de teste é apresentado à rede e os resultados obtidos são avaliados, mas não propagados.

Monitoração do processo de aprendizagem

- Um **critério** deve ser estabelecido como **ponto de parada**, antes que se alcance a superadaptação.
- A eficiência da rede para o conjunto de teste e treino vai **aumentado** devido ao ajuste dos pesos.
- Até que esta eficiência **atinge um platô** – em torno do qual os pesos ficam oscilando.
- A rede está entrando em superadaptação, e o processo de aprendizagem deve parar.

Fatores que influenciam o aprendizado

- O **número de camadas** invisíveis.
- O **número de neurônios** nas camadas invisíveis.
- O formato do **conjunto de treino**.

Número de camadas invisíveis

- Camadas invisíveis são:
  - Detectores de características.
  - Unidades internas de representação.
- A rede **acumula conhecimento abstraído** do conjunto de treino nestas camadas .
- A rede pode ser treinada para detectar a presença de características nos padrões de entrada.
  - Capacidade para lidar com **dados incompletos**.
- Para se descobrir as características que estão sendo representadas na camada invisível é necessário fazer uma **análise de sensibilidade**.

Número de camadas invisíveis

- O treinamento é iniciado com apenas **uma camada invisível**, e verificando-se o seu desempenho.
- Aumenta-se o número de camadas gradativamente, monitorando-se o desempenho da rede.
- A **capacidade de mapeamento** de padrões complexos **aumenta** com o aumento das camadas.
- Mais camadas invisíveis demandam **maior tempo de treinamento**.
- Para grande parte das aplicações:
  - Redes Neurais *Backpropagation* com **uma camada invisível** são o suficiente.

Formato do conjunto de treino

- Deve ter qualidade, refletir as informações desejadas.
- Pode ser preparado seguindo uma distribuição randômica.
- Ou seguindo uma mesma proporcionalidade para se evitar o aprendizado preferencial.
  - Preferência para os padrões mais frequentes.
- Deve ser completamente representativo das entradas utilizadas para a rede.

Sistemas tradicionais X Conexionistas

- **Sistemas tradicionais**:
  - Tem o problema de aquisição de conhecimento.
- **Redes Neurais**:
  - Necessita de uma boa quantidade de exemplos.
  - Não exige que se defina regras específicas.
  - Desenvolve suas próprias regras.
  - Útil quando não existem regras estabelecidas, ou elas são muito complexas.
  - Podem extrapolar fatos e gerar conclusões.

Sistemas tradicionais X Conexionistas

- **Redes Neurais**:
  - Tem habilidade para trabalhar com dados incompletos ou com ruído (incerteza).
  - Não é fácil obter uma explicação sobre os resultados obtidos.
    - Conhecimento na forma de pesos sinápticos.
    - Explicações sobre os resultados envolvem árdua tarefa de análise de sensibilidade.