

Sistemas Conexionistas

REDES NEURAIS ARTIFICIAIS

Profa. Dra. Gina Maira Barbosa de Oliveira

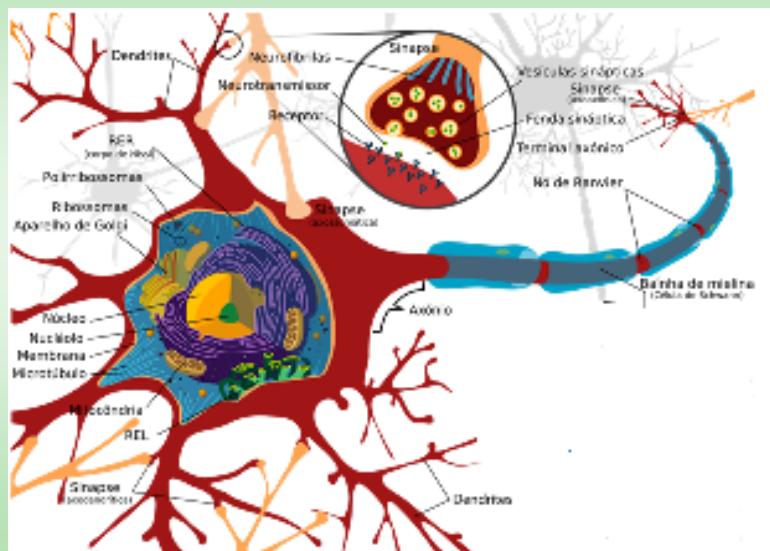
Redes Neurais Artificiais (RNAs)

- Técnica da IA que se inspira em um modelo biológico para a inteligência: o cérebro.
- Modelagem do cérebro: como ele é organizado e sua arquitetura elementar
- Analogia ao cérebro: as RNAs são formadas por um número individual de elementos simples (os neurônios) que se interconectam uns aos outros, formando redes capazes de armazenar e transmitir informações vindas do exterior.

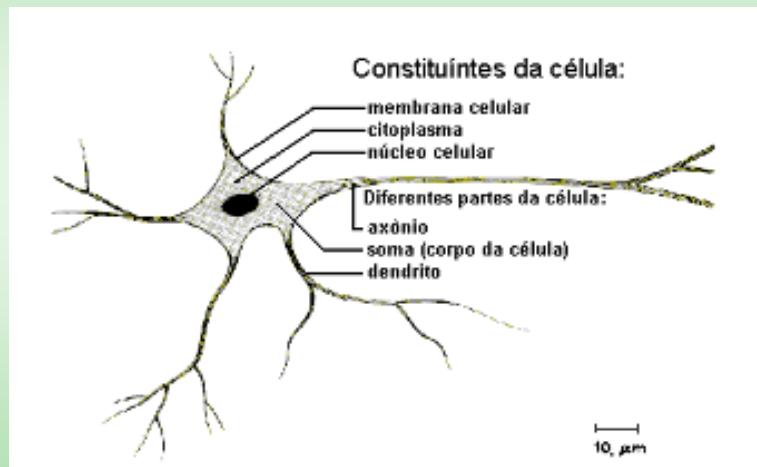
Sistema Nervoso e Redes Neurais Naturais

- O mais fascinante processador baseado em carbono.
- Composto por aproximadamente 10 bilhões de neurônios.
- Estão conectados entre si através de sinapses e juntos formam uma grande rede: Rede Neural

Sistema Nervoso e Redes Neurais Naturais Neurônio Biológico



Sistema Nervoso e Redes Neurais Naturais Neurônio Biológico



Sistema Nervoso e Redes Neurais Naturais Neurônio Biológico

- Dendritos: receber estímulos transmitidos por outros neurônios.
- Corpo do Neurônio ou Soma: coletar e combinar informações vindas de outros neurônios.
- Axônio: transmitir estímulos para outros neurônios
- Sinapse: “conexão” entre o axônio de um neurônio (através dos terminais) e o dendrito de outro neurônio. Transmitem estímulos através de diferentes concentrações de Sódio (Na^+) e Potássio (K^+). Na verdade, o terminal e o dendrito não se tocam (fenda sináptica).

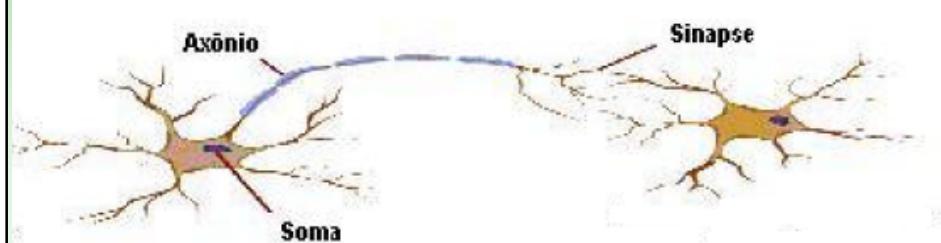
Sistema Nervoso e Redes Neurais Naturais

Neurônio Biológico

- Os neurônios podem estar em dois estados:
 - Ativo ou Excitado: envia estímulos através do axônio até os neurônios ligados a ele pelas sinapses.
 - Inativo ou Inibitório: não envia estímulos

Sistema Nervoso e Redes Neurais Naturais

Sinapse



- Os dendritos podem estabelecer 2 tipos de sinapses:
 - Excitatórias: excitam o neurônio receptor e esse tende a enviar estímulos pelo axônio, tendendo a passar o estímulo adiante.
 - Inibitórias: inibem o neurônio receptor e ele bloqueia a propagação de estímulo

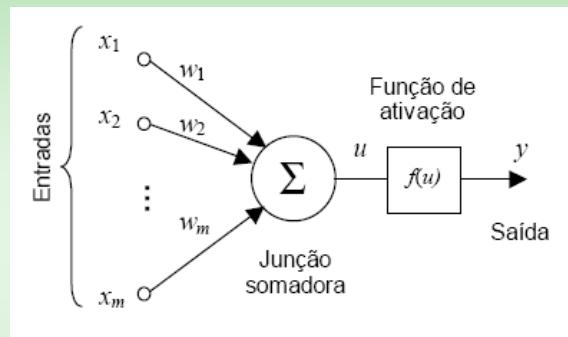
Sistema Nervoso e Redes Neurais Naturais

Neurônio Biológico

- O corpo de um neurônio está ligado a vários outros, que podem estar enviando estímulos tanto excitatórios quanto inibitórios.
- Quando as entradas chegam ao corpo do neurônio, ocorre um processo de integração (SOMA) dos impulsos e como resultado pode ser gerado um impulso (excitatório ou inibitório) que será propagado a outros neurônios.
- Produção do impulso: função de limiar que produz uma saída caso a soma das entradas seja maior ou igual a um dado limiar (Limiar de Ativação)

Modelo de McCulloch e Pitts:

Neurônio Artificial



Elementos básicos do neurônio artificial:

- a) Conjunto de Sinapses cada uma peso sináptico w_j e uma entrada x_j
- b) Somador: combinador linear das entradas, gerando u
- c) Função de Ativação aplicada ao valor de u , gerando y

Modelo de McCulloch e Pitts: Neurônio Artificial

Matematicamente:

$$u = \sum_{j=1}^m w_j x_j$$

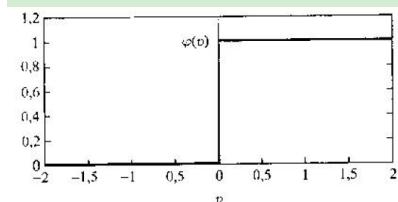
$$y = \varphi(u)$$

Modelo de McCulloch e Pitts: Neurônio Artificial

Tipos de função de ativação $\varphi(v)$:

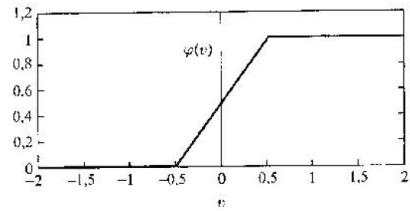
a) **Função de Limiar (ou step):**

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$



b) **Função Linear por partes:**

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 1/2 \\ v, & \text{se } 1/2 > v > -1/2 \\ 0 & \text{se } v \leq -1/2 \end{cases}$$



Modelo McCulloch e Pitts: Neur.Artificial

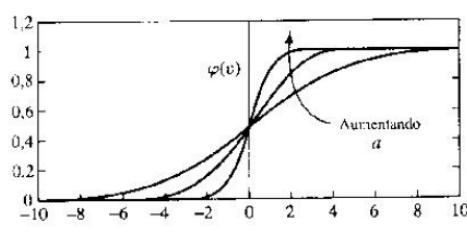
c) **Função Sigmóide:** Gráfico em forma de “S”. Exemplos:

Função Logística: $\varphi(v) = \frac{1}{1 + \exp(-av)}$ (varia de 0 a 1)

onde: a — parâmetro de inclinação da função sigmóide

Função Tangente Hiperbólica: $\varphi(v) = \tanh(v)$

(pode assumir valores negativos)



Exemplo: Rede Neural como memória associativa

Suponha que desejamos associar nomes de vários esportistas brasileiros às suas modalidades:

	Futebol	Automobilismo
Pelé	x	
Zico	x	
Emerson		x
Piquet		x

Para isso, podemos definir um código para associar cada esportista a uma entrada binária e cada modalidade a uma saída binária. Por exemplo:

00	Pelé
01	Zico
10	Emerson
11	Piquet

0	Futebol
1	Automobilismo

Exemplo: Rede Neural como memória associativa

Podemos usar um neurônio artificial para associar cada entrada com a saída desejada:

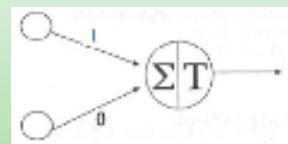
		0	1
		Futebol	Automobilismo
00	Pelé	x	
01	Zico	x	
10	Emerson		x
11	Piquet		x

Exemplo: Rede Neural como memória associativa

Podemos usar um neurônio artificial para associar cada entrada com a saída desejada:

		0	1
		Futebol	Automobilismo
00	Pelé	x	
01	Zico	x	
10	Emerson		x
11	Piquet		x

Uma possível solução:



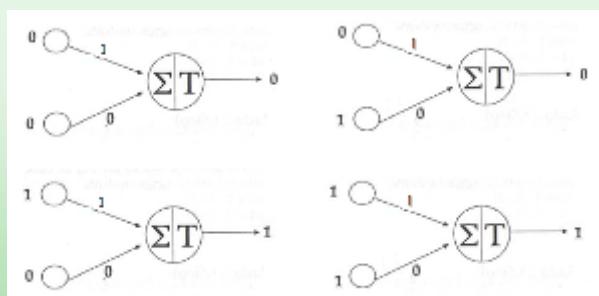
Ativação

$$y = \begin{cases} 1 & \text{se } \text{soma} > 0 \\ 0 & \text{se } \text{soma} \leq 0 \end{cases}$$

Exemplo: Rede Neural como memória associativa

Testando a solução:

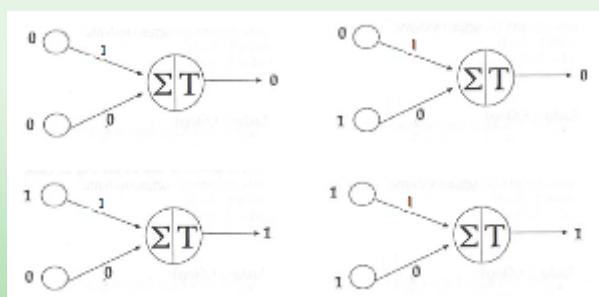
	0	1
	Futebol	Automobilismo
00 Pelé	x	
01 Zico	x	
10 Emerson		x
11 Piquet		x



Exemplo: Rede Neural como memória associativa

Testando a solução:

	0	1
	Futebol	Automobilismo
00 Pelé	x	
01 Zico	x	
10 Emerson		x
11 Piquet		x

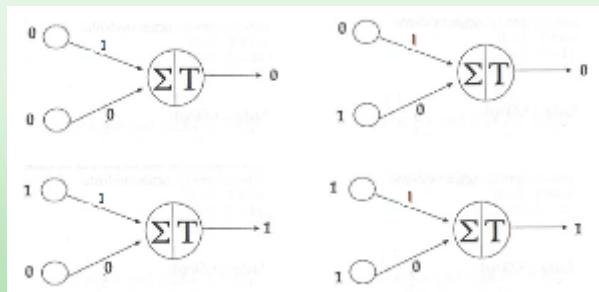


Problema:
Como encontrar os pesos adequados?

Exemplo: Rede Neural como memória associativa

Testando a solução:

		0	1
		Futebol	Automobilismo
00	Pelé	x	
01	Zico	x	
10	Emerson		x
11	Piquet		x



Solução:
Algoritmo
de
Treinamento

Histórico

- Modelo de McCulloch-Pitts (1943): primeiro modelamento matemático de um neurônio biológico .
- Lei de aprendizagem de Hebb (1949): primeiro método de treinamento de redes neurais.
- Primeiro esforço conjunto para estudar IA: foi o encontro no “Dartmouth College”, em 1956. Nasceram simultaneamente dois primeiros paradigmas da IA: simbólica e conexionista.
- Primeira arquitetura de RNA: Perceptron, proposta por Rosenblatt (1957).

Histórico

- Segunda arquitetura de RNA (Adaline/ Madaline) e o metodo de treinamento Regra Delta, propostos por Widrow e Hoff (1958/1960).
- Progressivamente as duas correntes de IA se separaram e as pesquisas em RNAs (corrente conexionista) andaram lentamente enquanto a corrente da manipulação simbólica se acelerou.
- Principal razão: livro de Minsky & Papert (1969). “Uma rede perceptron de uma única camada não capaz de resolver o problemas linearmente não separáveis”. “Problema do OU Exclusivo”

Histórico

- De 1969 a 1982: pouquíssimas publicações
- Algoritmos de predição por gradiente reverso proposto por Werbos (1974)
- Redes ART por Grossberg (1980)
- Redes Auto-organizáveis por Kohonen (1982)
- Redes recorrentes baseadas em funções de energia por Hopfield (1982)
- Algoritmo para treinamento de redes com múltiplas camadas (*Backpropagation*) por Rumelhart, Hinton e Williams (1986): resolvido o “Problema do OU Exclusivo”

Histórico

1943	McCulloch e Pitts
1949	Hebb
1957	Rosenblatt
1958	Widrow e Hoff
...	...
1969	Minsky e Papert
...	...
1960-1980	Kohonen, Grossberg, Widrow, Anderson, Caianiello, Fukushima,
...	...
1974	Werbos
...	...
1982	Hopfield
1986	Rumelhart

Perceptron - Histórico

1943: trabalho pioneiro de McCulloch e Pitts

- McCulloch: psiquiatra e neuroanatomista
- Pitts: matemático
- descrição do modelo formal de um neurônio, nodos MCP
- acreditavam que um número suficiente de neurônios atuando de forma adequada poderiam, a princípio, computar qualquer função computável.

Perceptron - Histórico

1948: Hebb publica o livro *The Organization of Behavior*

- proposta pela primeira vez uma regra de aprendizagem através da modulação (ou modificação) de pesos sinápticos.
- propôs que a efetividade de uma sinapse aumenta devido a ativação repetida de um neurônio (por outro neurônio).
- Regra de Hebb generalizada: propõe que mudanças nos pesos das conexões são dadas pelo produto da atividade pré-sináptica e pós-sináptica:

$$\Delta w_{ij}(t) = \alpha y_i(t) x_j(t),$$

$\Delta w_{ij}(t)$ é a mudança a ser aplicada no neurônio i ,
 α é um fator multiplicativo denominado de taxa de aprendizagem,
 y_i é a saída do neurônio i ,
 x_j é a entrada do neurônio j ,
 t é o índice de tempo.

Perceptron - Histórico

- o peso de um neurônio é atualizado de acordo com a seguinte regra:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t).$$

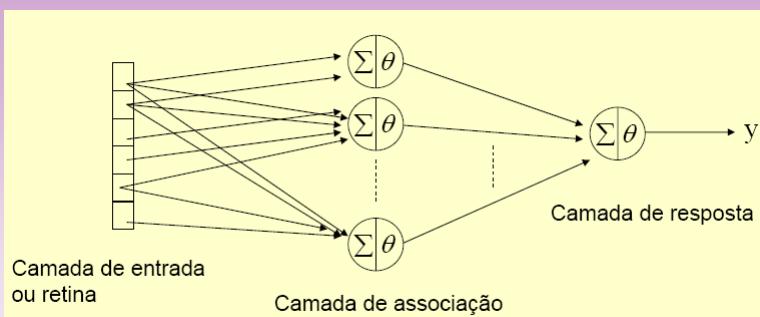
Perceptron - Histórico

1958: Rosenblatt propõe o Perceptron

- Rosenblatt introduziu o perceptron como a arquitetura mais simples de rede neural capaz de classificar padrões linearmente separáveis.
- O algoritmo de treinamento do perceptron foi o primeiro modelo de treinamento supervisionado.
- Basicamente, o perceptron consiste em uma única camada de neurônios com pesos sinápticos e bias ajustáveis.
- Rosenblatt demonstrou o teorema da convergência: se os padrões de entrada forem linearmente separáveis, o algoritmo de treinamento do perceptron possui convergência garantida, ou seja, é capaz de encontrar um conjunto de pesos que classifica corretamente os dados.

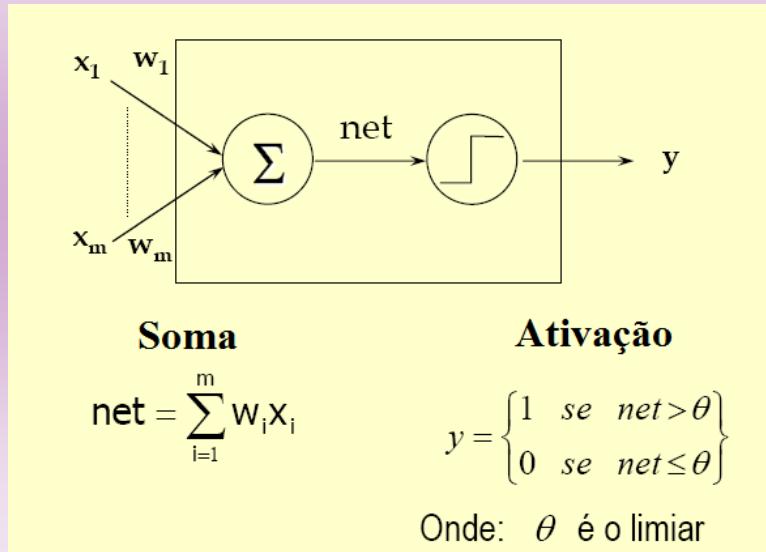
Perceptron - Arquitetura

- Os neurônios do perceptron são similares ao neurônio de McCulloch & Pitts (função de ativação tipo degrau), mas possuem pesos associados, incluindo o bias.
- O perceptron originalmente consistia em uma rede de três camadas, sendo que só a última camada possui pesos ajustáveis.



Perceptron - Arquitetura

- Perceptron com um único neurônio:



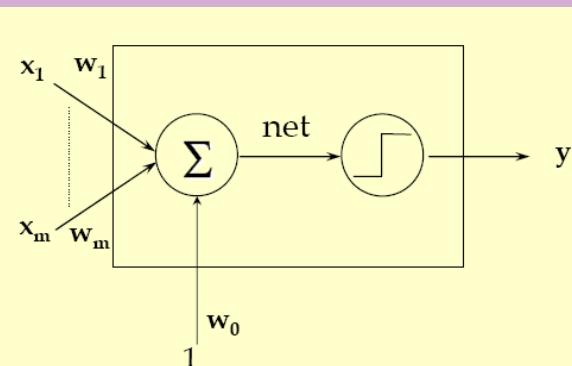
Perceptron - Arquitetura

- O treinamento de uma rede neural consiste em ajustar os seus parâmetros livres, que no caso do perceptron são os pesos w_1, w_2, \dots, w_m e o limiar θ .
- A função de ativação pode ser reescrita da seguinte forma:
 $\text{net} > \theta \rightarrow \text{net} - \theta > 0$.
- Se o termo $-\theta$ for incorporado ao net , chegaremos as seguintes funções de soma e ativação:

Soma	Ativação
$\text{net} = \sum_{i=0}^m w_i x_i$	$y = \begin{cases} 1 & \text{se } \text{net} > 0 \\ 0 & \text{se } \text{net} \leq 0 \end{cases}$
Onde: $w_0 = -\theta, x_0 = 1$	

Perceptron - Arquitetura

- O peso w_0 está associado a uma entrada x_0 cujo valor é sempre 1.
- O peso w_0 é também conhecido como bias e é também denotado pela letra b .



Perceptron - Treinamento

- Durante o treinamento cada padrão de entrada é apresentado ao perceptron e seus pesos são ajustados seguindo a regra criada por Rosenblatt (1958).
 - Seja x_i o i -ésimo padrão de treinamento, y_i a resposta do perceptron para x_i e d_i o valor desejado como resposta.
 - Caso $d_i = y_i$ o algoritmo de treinamento não irá fazer nenhum ajuste aos pesos do neurônio
 - Caso contrário, existem duas possibilidades:
 - 1) $d_i = 1$ e $y_i = 0$
 - 2) $d_i = 0$ e $y_i = 1$
- Em ambos os casos o algoritmo de treinamento precisa ajustar os pesos do Perceptron para que a saída se iguale ao valor desejado.

Perceptron - Treinamento

- Resumo do treinamento: $w = w + \Delta w$
 $d_i = y_i$ então $\Delta w = 0$
Se $d_i = 1$ e $y_i = 0$ então $\Delta w = \eta * x_i$
Se $d_i = 0$ e $y_i = 1$ então $\Delta w = -\eta * x_i$
- Caso simples: se considerarmos $\eta = 1$:
 $d_i = y_i$ então $\Delta w = 0$
Se $d_i = 1$ e $y_i = 0$ então $\Delta w = x_i$ e $w = w + x_i$
Se $d_i = 0$ e $y_i = 1$ então $\Delta w = -x_i$ e $w = w - x_i$
- Caso geral: se chamarmos $e = d_i - y_i$ a atualização dos pesos pode ser escrita de forma mais compacta:
 $w = w + e * \eta * x_i$

Perceptron – Treinamento ($\eta=1$)

Algoritmo de Treinamento do Perceptron

- 1º passo: arbitrar pesos para as conexões entre a camada de entrada e a unidade sigma;
- 2º passo: aplicar um padrão de sinais de entrada x_i e calcular a soma ponderada S ;
- 3º passo: passar a soma ponderada S para a função de transferência:
 - a) se a saída y_i estiver correta, voltar ao 2º passo
 - b) se a saída y_i estiver errada e for 0, adicionar cada peso das conexões com os sinais de entrada relativos a elas,
 - c) se a saída y_i estiver errada e for 1, subtrair de cada peso das conexões os sinais de entrada relativos a elas;
- 4º passo: voltar ao 2º passo.

Perceptron – Treinamento (η qualquer)

- O algoritmo abaixo pode ser utilizado para treinar o perceptron de um único neurônio ($f(\cdot)$ é a função degrau):

```
procedure [w] = perceptron(Max,E, $\eta$ ,X,d)
    initialize w // por simplicidade, inicialize com 0
    t ← 1
    while t < Max & E > 0 do,
        for i from 1 to N do, //para cada padrão de entrada
             $y_i \leftarrow f(wx_i)$  //determine a saída para  $x_i$ 
             $e_i \leftarrow d_i - y_i$  //determine o erro para  $x_i$ 
            w ← w +  $\eta e_i x_i$ //atualize o vetor de pesos
        end for
        E ← sum ( $e_i$ )
        t ← t + 1
    end while
end procedure
```

Perceptron – Exemplo de Treinamento

Para um melhor entendimento da rede neural Perceptron, vamos desenvolver um pequeno exemplo de aprendizado. O exemplo será composto por poucos elementos para simplificar a demonstração. No nosso exemplo desejamos que a rede aprenda a distinguir ídolos do esporte nacional. Vamos treinar a rede para reconhecer dois grandes jogadores de futebol (Pelé e Zico) e dois grandes pilotos de automobilismo (Emerson e Piquet).

	Futebol	Automobilismo
Pelé	x	
Zico	x	
Emerson		x
Piquet		x

Perceptron – Exemplo de Treinamento

Isso quer dizer que, quando a rede estiver treinada, poderemos apresentar qualquer um dos ídolos acima à rede e a mesma deverá identificar qual a atividade exercida pelo personagem apresentado.

Para tanto, antes de prosseguir com a rede neural, sabemos da necessidade de codificar a informação, tanto para as entradas quanto para as saídas desejadas. Essa necessidade parte do princípio que em uma rede neural tipo Perceptron, os valores devem assumir condições binárias, ou seja, 1 ou 0. Neste caso, uma proposta de identificação dos conjuntos da tabela acima pode ser codificada da seguinte forma:

		0	1
		Futebol	Automobilismo
00	Pelé	x	
01	Zico	x	
10	Emerson		x
11	Piquet		x

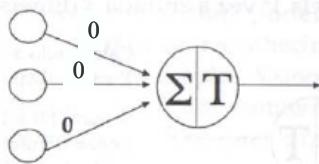
Perceptron – Exemplo de Treinamento

Acrescentando o VIÉS (1) ao conjunto de entrada temos, então, o conjunto com o seguinte aspecto:

100	Pelé
101	Zico
110	Emerson
111	Piquet

Podemos, agora, traçar o desenho da rede neural baseando-nos nas tabelas de dados (conjunto de treinamento). Pelas tabelas acima, podemos notar que a saída se constitui de apenas duas posições, 1 (0) ou automobilismo (1), logo, podemos usar um neurônio de saída que, identificando 100 (Pelé) ou 101 (Zico) produza 0 (futebol), e, identificando 110 (Emerson) ou 111 (Piquet) produza 1 (automobilismo) como saída. Para as entradas usaremos 3 elementos de entrada, um para o viés (1) e os outros dois para as informações relativas aos ídolos (00, 01, 10 e 11).

Perceptron – Exemplo de Treinamento



Iniciando os pesos do neurônio com 0 temos o seguinte conjunto de pesos:

$$w = \{w_0, w_1, w_2\} = \{0, 0, 0\}$$

A função de transferência do neurônio é igual a:

se soma ponderada > 0 então saída = 1
senão saída = 0

O conjunto de treinamento constitui os seguintes pares:

entrada 1= $\{x_0, x_1, x_2\} = \{1, 0, 0\}$, cuja saída é saída 1= $\{y\} = \{0\}$

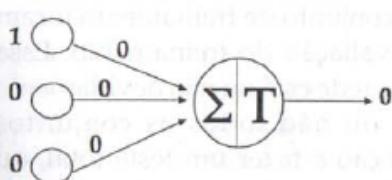
entrada 2= $\{x_0, x_1, x_2\} = \{1, 0, 1\}$, cuja saída é saída 2= $\{y\} = \{0\}$

entrada 3= $\{x_0, x_1, x_2\} = \{1, 1, 0\}$, cuja saída é saída 3= $\{y\} = \{1\}$

entrada 4= $\{x_0, x_1, x_2\} = \{1, 1, 1\}$, cuja saída é saída 4= $\{y\} = \{1\}$

Perceptron – Exemplo de Treinamento

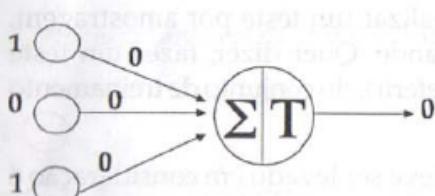
Apresentando pela 1^a vez a entrada 1 (Pelé):



Calculando a saída para a entrada 1
 $\Sigma x.w = x_0w_0 + x_1w_1+x_2w_2$

Soma = $1.0 + 0.0 + 0.0 = 0$ e $FT(0) = 0$
saída = 0 (futebol) e
saída desejada = 0 (futebol), logo, a
saída está correta

Apresentando pela 1^a vez a entrada 2 (Zico):

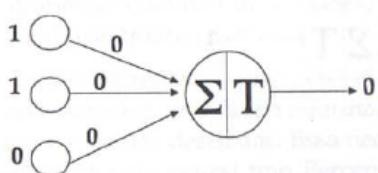


Calculando a saída para a entrada 2
 $\Sigma x.w = x_0w_0 + x_1w_1+x_2w_2$

Soma = $1.0 + 0.0 + 1.0 = 0$ e $FT(0) = 0$
saída = 0 (futebol) e
saída desejada = 0 (futebol), logo, a
saída está correta

Perceptron – Exemplo de Treinamento

Apresentando pela 1^a vez a entrada 3 (Emerson):

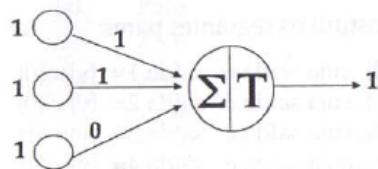


Calculando a saída para a entrada 3
 $\Sigma x.w = x_0w_0 + x_1w_1 + x_2w_2$

Soma = $1.0 + 1.0 + 0.0 = 0$ e $FT(0) = 0$
saída = 0 (futebol) e
saída desejada = 1 (automobilismo),
logo, a saída não está correta

Como saída = 0 vamos adicionar a
cada peso a entrada correspondente:
 $w_0 = 0 + 1 = 1$
 $w_1 = 0 + 1 = 1$
 $w_2 = 0 + 0 = 0$

Apresentando pela 1^a vez a entrada 4 (Piquet):



Calculando a saída para a entrada 4
 $\Sigma x.w = x_0w_0 + x_1w_1 + x_2w_2$
Soma = $1.1 + 1.1 + 1.0 = 2$ e $FT(2) = 1$
saída = 1 (automobilismo) e
saída desejada = 1 (automobilismo),
logo, a saída está correta

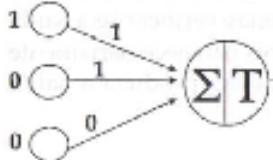
Agora que todas as entradas do conjunto de treinamento foram apresentadas, deve-se fazer uma avaliação do treinamento. Essa avaliação poderá ajudar a decidir se a rede está ou não devidamente treinada, ou seja, ela reconhece ou não todos os conjuntos apresentados. Uma forma de avaliação é fazer um teste total, ou seja, apresentar todo o conjunto de treinamento e quando uma entrada não for reconhecida, ainda durante a apresentação do conjunto, interromper o teste, ajustar os pesos e continuar o treinamento. Outra proposta é realizar um teste por amostragem, quando o conjunto for muito grande. Quer dizer, fazer um teste com 10%, ou 20% (como o leitor preferir), do conjunto de treinamento e verificar o seu sucesso.

Outro ponto importante que deve ser levado em consideração é a precisão da rede. É certo que, dependendo do conjunto de treinamento e da arquitetura da rede neural, talvez ela não reconheça 100% dos casos apresentados à ela, mas reconhece uma percentagem razoável. Essa decisão deve ser tomada em tempo de treinamento.

Para o nosso caso em particular, podemos decidir pelo treinamento total, ou seja, 100% de reconhecimento, visto que o conjunto e a rede são pequenos e simples. Vamos, então, reiniciar o treinamento já que houve, em algum momento do treinamento anterior, um reajuste de pesos. Isso quer dizer que em algum momento, alguma entrada não foi reconhecida apropriadamente, e, dessa forma, vamos repassar o conjunto de treinamento para eliminar a possibilidade da rede não reconhecer alguma entrada do conjunto apresentado.

Perceptron – Exemplo de Treinamento

Apresentando pela 2^a vez a entrada 1 (Pelé):

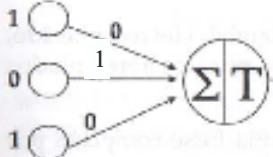


Calculando a saída para a entrada 1
 $\Sigma x.w = x_0w_0 + x_1w_1 + x_2w_2$

Soma = $1.1 + 0.1 + 0.0 = 1$ e $FT(1) = 1$
saída = 1 (automobilismo) e
saída desejada = 0 (futebol), logo, a saída
não está correta

Como saída = 1 vamos subtrair de cada
peso a entrada correspondente:
 $w_0 = 1 - 1 = 0$
 $w_1 = 1 - 0 = 1$
 $w_2 = 0 - 0 = 0$

Apresentando pela 2^a vez a entrada 2 (Zico):

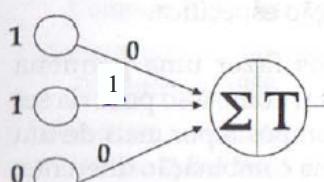


Calculando a saída para a entrada 2
 $\Sigma x.w = x_0w_0 + x_1w_1 + x_2w_2$

Soma = $1.0 + 0.1 + 1.0 = 0$ e $FT(0) = 0$
saída = 0 (futebol) e
saída desejada = 0 (futebol), logo, a
saída está correta

Perceptron – Exemplo de Treinamento

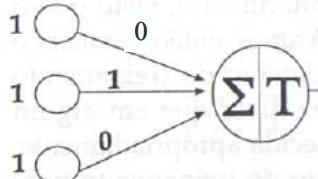
Apresentando pela 2^a vez a entrada 3 (Emerson):



Calculando a saída para a entrada 3
 $\Sigma x.w = x_0w_0 + x_1w_1 + x_2w_2$

Soma = $1.0 + 1.1 + 0.0 = 1$ e $FT(1) = 1$
saída = 1 (automobilismo) e
saída desejada = 1 (automobilismo),
logo, a saída está correta

Apresentando pela 2^a vez a entrada 4 (Piquet):



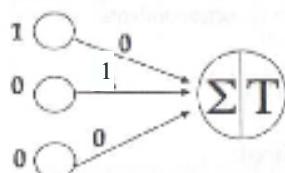
Calculando a saída para a entrada 4
 $\Sigma x.w = x_0w_0 + x_1w_1 + x_2w_2$

Soma = $1.0 + 1.1 + 1.0 = 1$ e $FT(1) = 1$
saída = 1 (automobilismo) e
saída desejada = 1 (automobilismo),
logo, a saída está correta

Perceptron – Exemplo de Treinamento

Conforme houve um ajuste de pesos logo no início deste segundo treinamento, poderíamos reiniciar pela 3^a vez todo o treinamento da rede neural. Entretanto, podemos dispensar todo este retreino vendo que as entradas 2, 3 e 4 produzem saídas corretas mediante os pesos { 0, 1, 0 } da rede. Precisamos apenas verificar se a saída produzida para a entrada 1 está correta, pois não necessariamente ajustando os pesos uma vez fará com que se produza a saída esperada.

Representando a entrada 1 (Pelé):



Calculando a saída para a entrada 1
 $\Sigma x_i w_i = x_0 w_0 + x_1 w_1 + x_2 w_2$
Soma = $1.0 + 0.1 + 0.0 = 0$ e $FT(0) = 0$
saída = 0 (futebol) e
saída desejada = 0 (futebol), logo, a
saída está correta

Conforme o teste realizado acima, Pelé também foi reconhecido, o que significa que a rede reconheceu todos os casos apresentados do conjunto de treinamento.

Perceptron Aspectos Práticos do Treinamento

- Condição inicial: verificou-se que diferentes conjuntos iniciais de pesos para o perceptron podem levar a diferentes superfícies de decisão.
- O problema de ajuste supervisionado de pesos pode ser visto como um processo de busca por um conjunto de pesos que otimizam uma determinada superfície de erro. Uma escolha inadequada da condição inicial da rede pode levar o algoritmo a uma convergência para ótimos locais desta superfície de erro.
- Critério de convergência: no caso do perceptron, é possível garantir que, dado um conjunto de padrões linearmente separáveis, o algoritmo é capaz de encontrar uma superfície de decisão capaz de classificar corretamente os dados.

Perceptron

Aspectos Práticos do Treinamento

- É possível utilizar como critério de convergência para o perceptron simples (classificação binária) a determinação de erro igual a zero para todos os padrões de entrada.
- Outro critério que pode ser utilizado é a adaptação por uma quantidade finita de iterações, denominadas de *épocas de treinamento*.
- Parâmetros de treinamento: o algoritmo de treinamento do perceptron possui basicamente o parâmetro de treinamento η (taxa de aprendizagem) que deve ser definido pelo usuário.

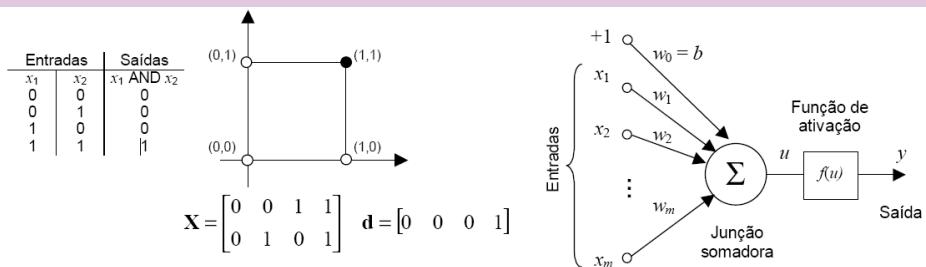
Perceptron

Aspectos Práticos do Treinamento

- Treinamento versus aplicação da rede: é importante diferenciar entre o processo de treinamento e aplicação da rede.
- O treinamento da rede corresponde ao processo de ajuste de pesos quando exemplos são apresentados como entrada, sabendo-se para cada exemplo a saída desejada (supervisionado).
- Após treinada, a rede poderá ser aplicada ao mesmo problema, de forma a verificar a qualidade do aprendizado, ou a outro problema, de forma a verificar sua *capacidade de generalização*.

Perceptron – Motivação Geométrica

- Considere o problema de utilizar o perceptron com um único neurônio para representar a função lógica AND.



A saída y_i do neurônio para o vetor de dados x_i pode ser representada na forma: $y_i = f(wx_i + b)$

- Para quaisquer valores de w e b , a função $f(u)$ separa o espaço de entradas em duas regiões, sendo que a curva de separação (superfície de decisão) é uma linha reta.

Perceptron – Motivação Geométrica

- A equação desta reta é dada por:

$$w_1 x_1 + w_2 x_2 + b = 0$$

- Se a função de ativação do tipo degrau possui $\theta = 0$, então $w_1 x_1 + w_2 x_2 + b \geq 0$ resultará em uma saída positiva.

- Inicializando todos os pesos em zero $w = [0 \ 0]$ e $b = 0$ com $\eta = 1$, o algoritmo de treinamento do perceptron fornece:

$w_1 = 2$; $w_2 = 1$; $b = -3$, definindo a reta:

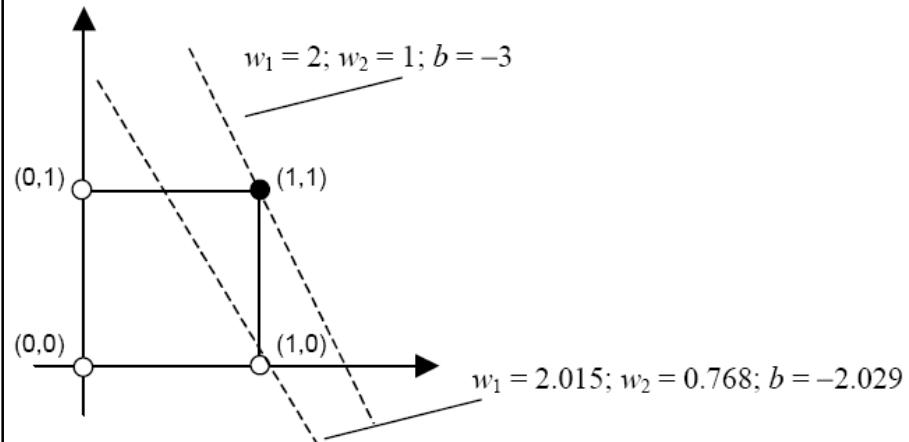
$$2x_1 + 1x_2 - 3 = 0.$$

- Inicializando os pesos com valores aleatórios pequenos, a reta de decisão obtida seria diferente. Por exemplo se os pesos iniciais forem $w_1 = 0.015$; $w_2 = 0.768$; $b = 0.971$, obtem-se $w_1 = 2.015$; $w_2 = 0.768$; $b = -2.029$, definindo a reta:

$$2.015x_1 + 0.768x_2 - 2.029 = 0.$$

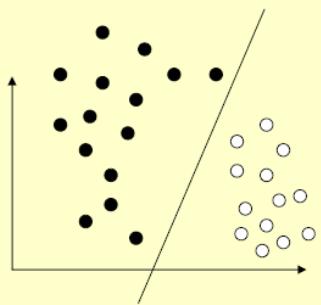
Perceptron – Motivação Geométrica

- A figura a seguir mostra as duas superfícies de decisão e os pesos e bias determinados pelo algoritmo de treinamento.

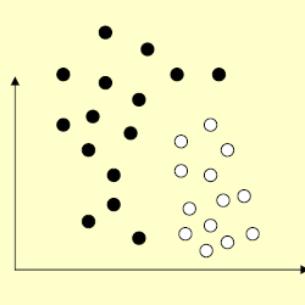


Perceptron – Limitações

O perceptron só é capaz de separar classes através de funções lineares. Mas para que ele consiga isso é necessário que as classes sejam linearmente separáveis, o que é uma condição não garantida para a maioria dos problemas de classificação.



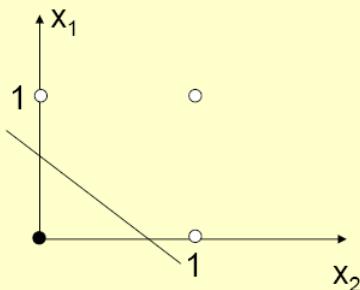
Linearmente separável



Não linearmente separável

Perceptron – Limitações

O perceptron é capaz de funcionar como uma porta **ou**, já que o **ou** é linearmente separável.

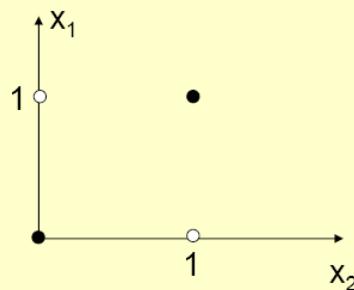


Função **ou**

x_1	x_2	$x_1 \text{ ou } x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Perceptron – Limitações

O **ou exclusivo** não é linearmente separável, então o perceptron não pode funcionar como esta porta lógica.



Função **ou exclusivo**

x_1	x_2	$x_1 \text{ xor } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron - Histórico

1958: Rosenblatt propõe o Perceptron

1962: Rosenblatt demonstra que o algoritmo de treinamento do Perceptron sempre converge se as classes forem linearmente separáveis (Teorema de Convergência do Perceptron).

1969: Minsky e Papert demonstram as limitações do Perceptron – Problema do XOR

Anos 70: Como o perceptron trata apenas de problemas linearmente separáveis e não existe um algoritmo de treinamento para redes de múltiplas camadas, o seu uso é muito limitado. Com isso as pesquisas em redes neurais ficaram paradas por mais de 10 anos.

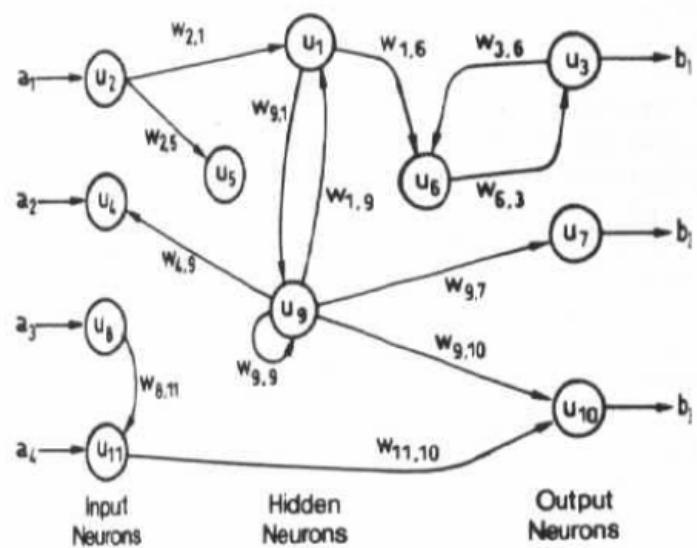
Arquitetura, Topologia e Treinamento

- Arquitetura de uma RNA: o padrão geral das conexões sinápticas que define a estrutura geral do arranjo de neurônios;
- Topologia: dentro da arquitetura considerada, define a composição estrutural que a RNA assume. Por exemplo, quantos neurônios, qual a função de ativação, etc.
- Treinamento: algoritmos de ajuste sináptico ou algoritmo de aprendizagem. Muitas vezes específico para a arquitetura adotada.

Arquiteturas de Rede

- Camadas mais usuais das arquiteturas: Camada de Entrada, Camada(s) Intermediária(s) (ou Oculta ou Invisível), Camada de Saída
- Principais Arquiteturas:
 - Redes Diretas (Feedforward) de Camada Simples
 - Redes Diretas (Feedforward) de Camadas Múltiplas
 - Redes Recorrentes ou Realimentadas
 - Redes Reticuladas

Arquiteturas de Rede

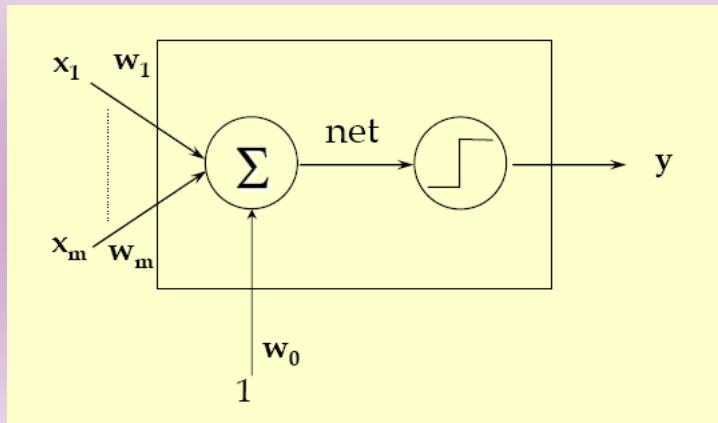


Exemplo de uma rede neural artificial arbitrária

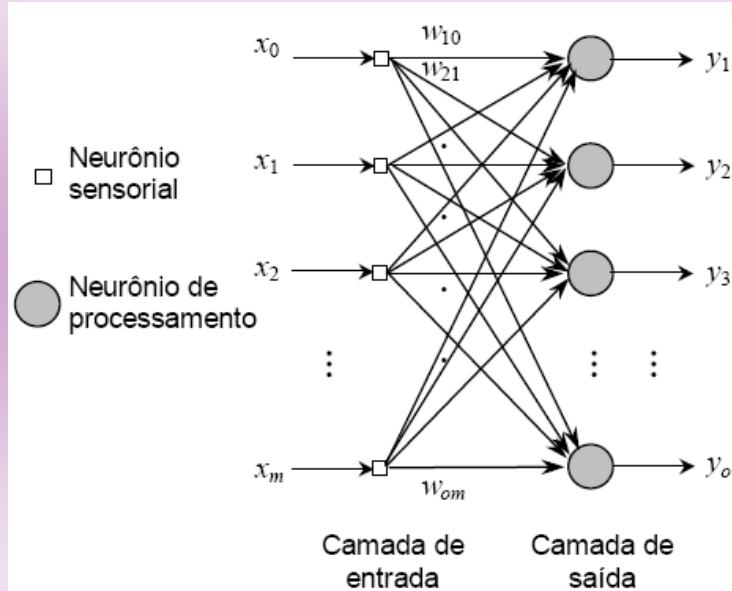
Redes Diretas de Camada Simples

- Possui apenas a “camada” de entrada (neurônios sensoriais) e a camada de saída (única com neurônios processadores)
- O fluxo de informações (estímulos) flui em uma única direção: da entrada para a saída
- Aplicação em classificação de padrões e filtragem linear.
- Principais exemplos: Perceptrons e Adalines
- Algoritmos de Treinamento: Regra de Hebb (Perceptrons) e Regra Delta (Adaline)

Redes Diretas de Camada Simples



Redes Diretas de Camada Simples



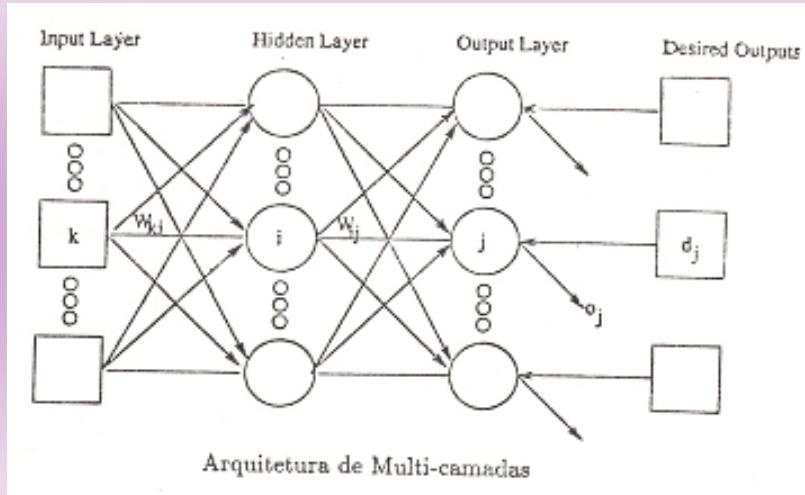
Redes Diretas de Múltiplas Camadas

- Além das camadas de entrada e de saída possui uma ou mais camadas escondidas (com elementos processadores).
- O fluxo de informações (estímulos) flui em uma única direção: da entrada para a saída.
- O número necessário de camadas escondidas e neurônios em cada camada dependem do tipo e complexidade do problema, além da quantidade e qualidade dos dados disponíveis para treinamento.

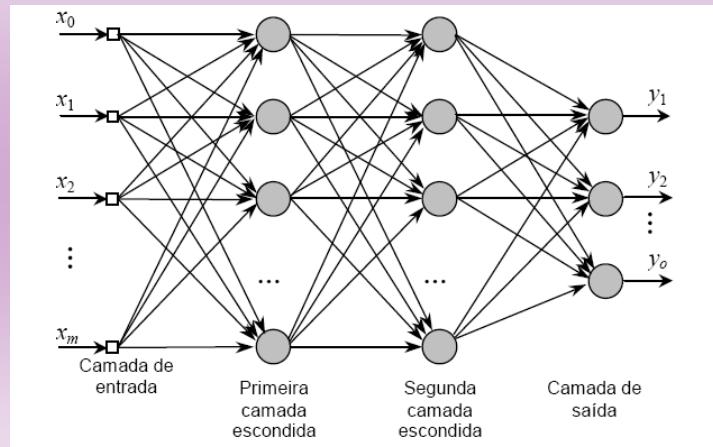
Redes Diretas de Múltiplas Camadas

- Aplicação: aproximação de funções, classificação de padrões, identificação de sistemas, otimização, robótica, controle de processos e filtragem não-linear
- Principais exemplos: MLP (Perceptrons Multicamadas) e Redes de Base Radial (RBF)
- Algoritmos de Treinamento: Regra Delta Generalizada ou Backpropagation (MLP) e Regra Delta/Competitiva (RBF)

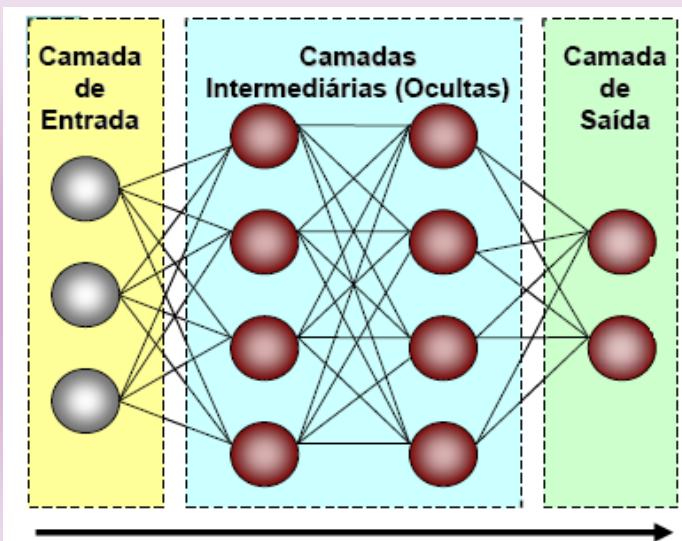
Redes Diretas de Múltiplas Camadas



Redes Diretas de Múltiplas Camadas: Exemplo com 2 camadas escondidas

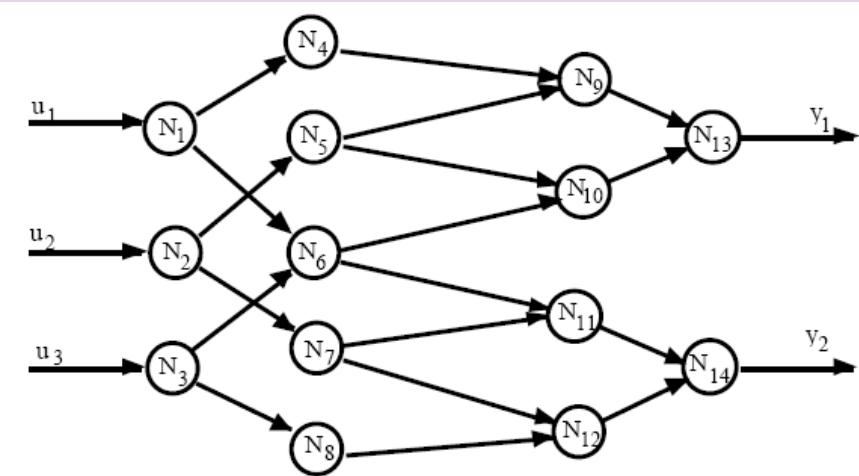


Redes Diretas de Múltiplas Camadas



Sentido único de propagação do sinal

Redes Diretas de Múltiplas Camadas



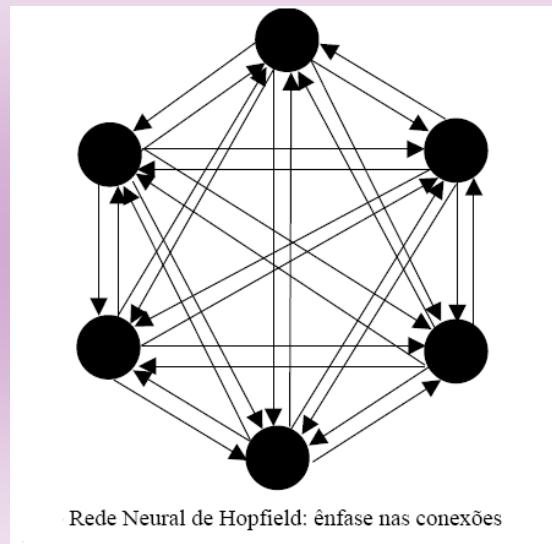
Redes Recorrentes ou Realimentadas

- Além do fluxo de sinais da entrada para a saída, as saídas de alguns neurônios são retroalimentadas como entradas de neurônios de camadas anteriores .
- Elas possuem, pelo menos, um laço realimentando a saída de neurônios para outros neurônios da rede.
- A retroalimentação qualifica tais redes para processamento dinâmico de informações ou seja, para aplicação em problemas variantes com o tempo como previsão de series temporais, controle de processos, etc.

Redes Recorrentes ou Realimentadas

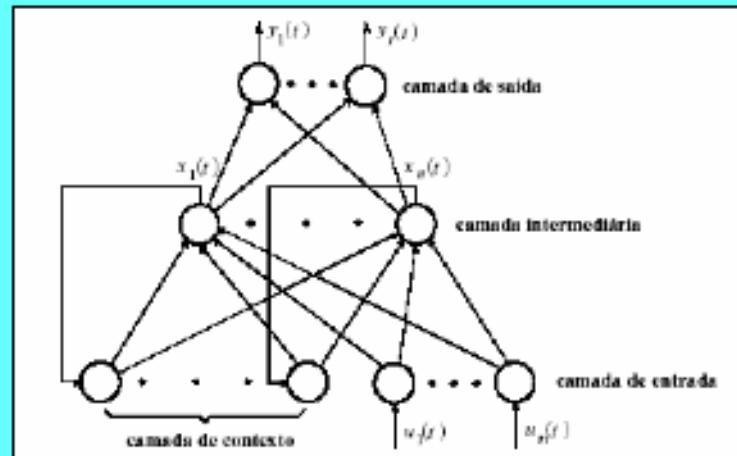
- Principais exemplos: rede de Hopfield e rede Perceptron multicamadas com retroalimentação.
- Algoritmos de treinamento: minimização de funções de energia (rede de Hopfield) e regra delta generalizada/*Backpropagation* (rede MLP com retroalimentação)

Redes Recorrentes ou Realimentadas



Redes Recorrentes ou Realimentadas

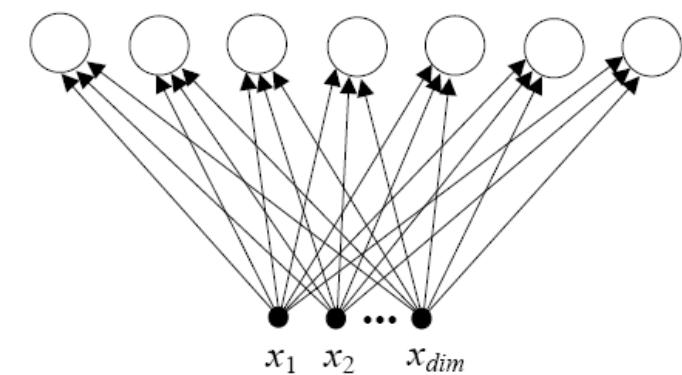
Rede Elman



Redes Reticuladas

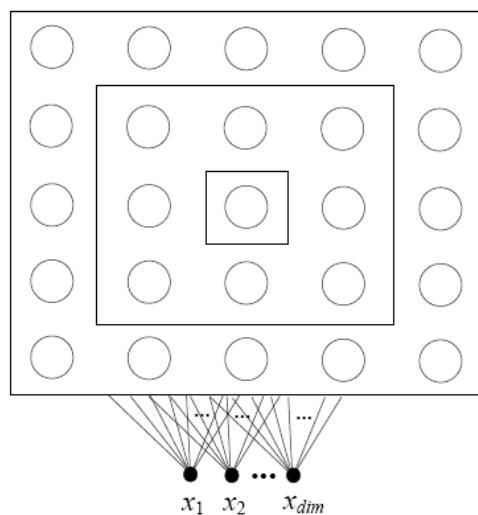
- As estruturas reticuladas consideram a disposição espacial dos neurônios visando propósitos de extração de características. Ou seja, a localização do neurônio (dada pela sua vizinhança) está diretamente relacionada com o processo de ajuste de pesos.
- Aplicações: especialmente em problemas de agrupamentos (clustering).
- Principal exemplo: rede de Kohonen
- Algoritmo de treinamento baseado em um processo competitivo

Redes Reticuladas



Rede de Kohonen em arranjo unidimensional: ênfase na vizinhança

Redes Reticuladas



Rede de Kohonen em arranjo bidimensional: ênfase na vizinhança

Tipos de Treinamento

- **Aprendizado supervisionado:** baseado em um conjunto de exemplos de estímulo/resposta (ou entrada-saída).
- **Aprendizado não-supervisionado:** baseado apenas nos estímulos recebidos pela rede neural. A rede deve aprender a “categorizar” os estímulos.
- **Aprendizado por reforço:** o comportamento da rede é avaliado apenas com base em alguma critério numérico, fornecido em instantes espaçados de tempo;

Perceptron com Múltiplos Neurônios

- A regra de aprendizagem do perceptron é do tipo supervisionada, empregando a aprendizagem por correção de erro.
- Esta regra pode ser facilmente estendida para atualizar os pesos de uma rede de neurônios em uma única camada.
- Neste caso, para cada vetor de entrada \mathbf{x}_i , haverá um vetor de saídas da rede: $\mathbf{y}_i = f(\mathbf{W}_{xi})$;
- Existe um vetor de erros para cada padrão de entrada:
$$\mathbf{e}_i = \mathbf{d}_i - \mathbf{y}_i$$

Perceptron com Múltiplos Neurônios

```
procedure [W] = perceptron(Max,η,X,D)
    initialize W //for simplicity set it to zero
    t ← 1
    while t < Max do,
        for i from 1 to N do, //para cada padrão de entrada
            yi ← f(Wxi ) //determine a saída da rede para xi
            ei ← di - yi //determine a saída da rede para xi
            W ← W + η ei xi //atualize a matriz de pesos
        end for
        t ← t + 1
    end while
end procedure
```

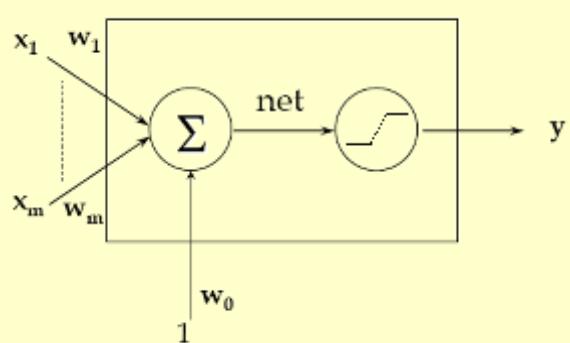
Adaline - Histórico

1958: Widrow & Hoff desenvolveram o *algoritmo dos quadrados mínimos (least square) ou regra delta*, praticamente ao mesmo tempo em que Rosenblatt propôs o perceptron.

Eles introduziram a rede Adaline (*Adaptive Linear Element*), muito similar ao perceptron, porém com função de ativação linear ao invés de função sinal.

O objetivo do algoritmo de treinamento é minimizar o erro quadrático médio (MSE ou EQM) entre a saída da rede e a saída desejada.

Adaline - Arquitetura



Adaline - A Regra Delta

- Seja $\{x_1, d_1\}, \{x_2, d_2\}, \dots, \{x_N, d_N\}$, um conjunto de pares de entrada-saída, onde x_j é o vetor de entradas j , e d_j seu correspondente vetor de saídas (desejadas).
- A regra delta ajusta os pesos e o limiar da rede neural de forma a minimizar o erro (diferença) entre a saída da rede e a saída desejada para todos os padrões de treinamento.
- A soma dos erros quadráticos (SEQ) para um determinado padrão é dada por:

$$\mathfrak{J} = \sum_{i=1}^o e_i^2 = \sum_{i=1}^o (d_i - y_i)^2$$

Adaline - A Regra Delta

- O gradiente de \mathfrak{J} , também denominado de *índice de desempenho ou função custo*, é o vetor que consiste das derivadas parciais de \mathfrak{J} em relação a cada um dos pesos.
- Este vetor fornece a direção de crescimento mais rápido do erro (\mathfrak{J}). Portanto, a direção oposta ao gradiente de \mathfrak{J} é a direção de maior decrescimento da função custo.
- Sendo assim, o erro pode ser reduzido ajustando-se os pesos da rede da seguinte forma:

$$w_{IJ} = w_{IJ} - \eta \cdot \frac{\partial \mathfrak{J}}{\partial w_{IJ}}$$

onde w_{IJ} é um peso arbitrário conectando o neurônio pré-sináptico J ao neurônio pós-sináptico I , e η é a taxa de aprendizagem.

Adaline - A Regra Delta

- Torna-se necessário portanto, determinar de forma explícita o gradiente do erro em relação ao peso arbitrário.
- Como o peso w_{IJ} influencia apenas a unidade I , o gradiente do erro é dado por:

$$\frac{\partial \mathfrak{J}}{\partial w_{IJ}} = \frac{\partial}{\partial w_{IJ}} \sum_{i=1}^o (d_i - y_i)^2 = \frac{\partial}{\partial w_{IJ}} (d_I - y_I)^2$$

Mas, $y_I = f(\mathbf{w}_I \cdot \mathbf{x}) = f(\sum_j w_{Ij} x_j) = \sum_j w_{Ij} x_j$ (ativação linear)

$$\frac{\partial \mathfrak{J}}{\partial w_{IJ}} = -2(d_I - y_I) \frac{\partial y_I}{\partial w_{IJ}} = -2(d_I - y_I)x_J$$

Adaline - A Regra Delta

- Portanto a regra delta para atualizar o peso do neurônio w_{IJ} é dada por:

$$w_{IJ} = w_{IJ} + \eta (d_I - y_I) x_J.$$

- Em notação matricial:

$$\mathbf{W} = \mathbf{W} + \eta \mathbf{e}_i \mathbf{x}_i^T$$

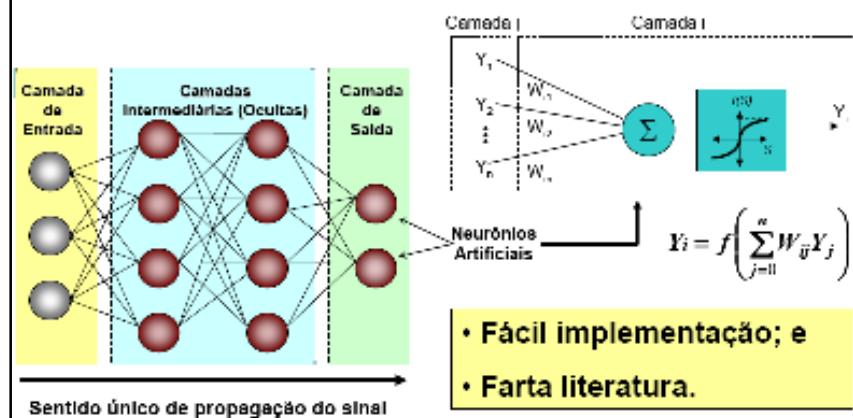
Adaline - A Regra Delta

- A beleza deste algoritmo é que, a cada iteração, ele calcula uma aproximação do vetor gradiente do erro multiplicando o erro pelo vetor de entradas, e esta aproximação pode ser utilizada em um algoritmo do tipo gradiente descendente com taxa de aprendizagem fixa.

Redes MLP (Perceptron Multi-Camadas)

- Minsky e Papert demonstraram que redes de uma camada não são capazes de solucionar problemas que não sejam linearmente separáveis. Como não acreditavam na possibilidade de se construir um método de treinamento para redes com mais de uma camada, eles concluíram que as redes neurais seriam sempre suscetíveis a essa limitação.
- O desenvolvimento do algoritmo de treinamento backpropagation, por Rumelhart, Hinton e Williams em 1986, mostrou que é possível treinar eficientemente redes com camadas intermediárias, resultando no modelo de Redes Neurais Artificiais mais utilizado atualmente, as redes Perceptron Multi-Camadas (MLP), treinadas com o algoritmo backpropagation.

Redes MLP (Perceptron Multi-Camadas)



Redes MLP (Perceptron Multi-Camadas)

- Nessas redes, cada camada tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói o padrão que será a resposta. As camadas intermediárias funcionam como extratoras de características, seus pesos são uma codificação de características apresentadas nos padrões de entrada e permitem que a rede crie sua própria representação, mais rica e complexa, do problema.

UFU1

Redes MLP (Perceptron Multi-Camadas)

- ✓ Ajuste Sináptico: $W_{ij}(t+1) = W_{ij}(t) + \Delta W_{ij}$
- ✓ Modo Supervisionado ($Y^d - Y$):

$$\text{Backpropagation: } \Delta W_{ij} = \eta \delta_i Y_j$$

Erro da saída

$$\text{Camada de saída: } \delta_i = \overbrace{(Y_i^d - Y_i)}^{\text{Erro da saída}} \underbrace{f\left(\sum_j W_{ij} Y_j\right)}_{\text{Derivada da função de ativação}}$$

Derivada da função de ativação

$$\text{Demais Camadas: } \delta_i = \overbrace{f\left(\sum_j W_{ij} Y_j\right)}^{\text{Próxima camada}} \sum_k \delta_k w_{ki}$$

Slide 88

UFU1 FACOM; 03/12/2010

Referências adicionais utilizadas na preparação desse material

Material didático:

Prof. Dr. Fernando Von Zuben

DCA/FEEC/Unicamp

Prof. Dr. A. C. G. Thomé
NCE/UFRJ