



Departamento de Ciência da Computação (DCC)

# **Redes Neurais Artificiais - Parte II**

## **GCC128 - Inteligência Artificial**

**Vinicius Ruela Pereira Borges**

`viniciusrpb@icmc.usp.br`

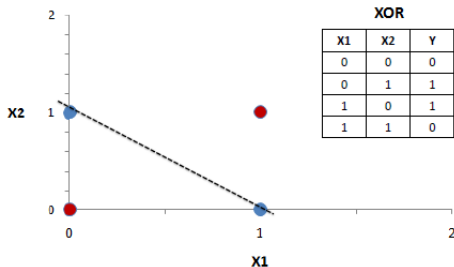
*Lavras, 29 de julho de 2016*

# Roteiro - Redes Neurais Artificiais (RNA)

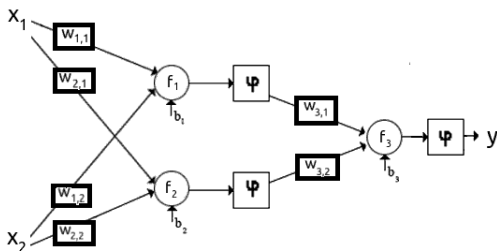
- Redes Neurais Multilayer Perceptron
- Exercícios

# Redes Perceptron Simples

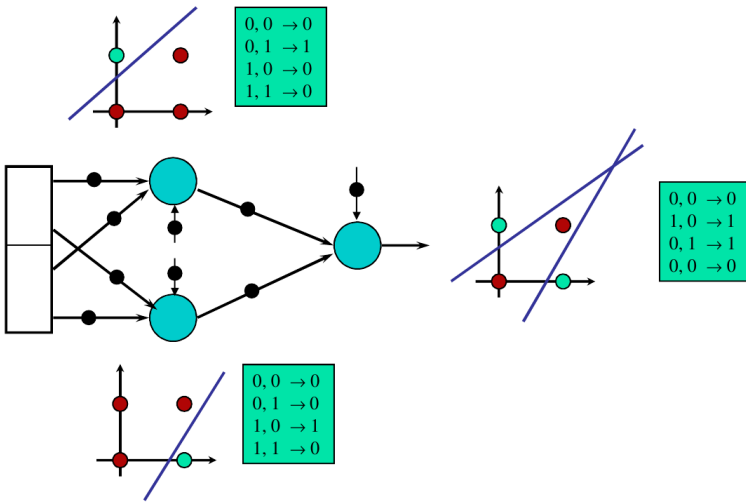
- Como resolver os problemas não linearmente separáveis?
- Problema XOR (Ou Exclusivo)



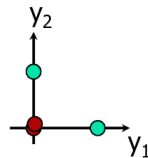
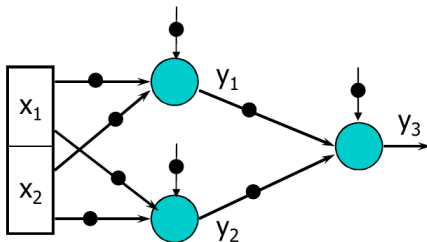
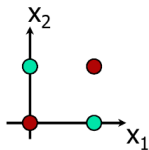
- Utilizar mais camadas de neurônios dentro da rede
  - Camada 1: uma rede Perceptron para cada problema linearmente separável
  - Camada 2: uma rede que combina as saídas das redes da primeira camada, produzindo a saída final



# Modificações



# Modificações



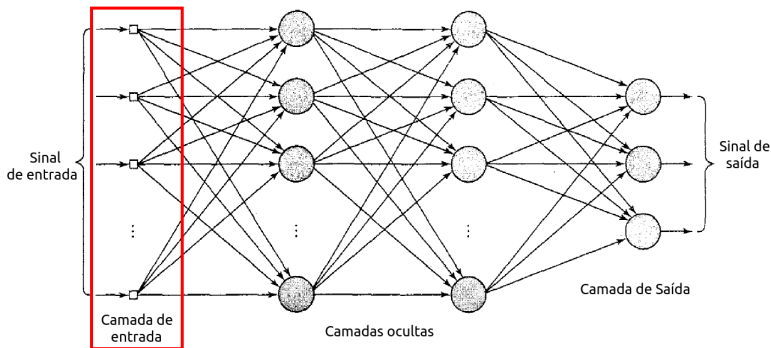
$x_1$	$x_2$	$y_1$	$y_2$	$y_3$
0	0	0	0	$\rightarrow 0$
0	1	1	0	$\rightarrow 1$
1	0	0	1	$\rightarrow 1$
1	1	0	0	$\rightarrow 0$

# Redes Multilayer Perceptron

- Tipo de RNA utilizada para aprender problemas não-linearmente separáveis
- Rede do tipo feed-forward: alimentação ocorre em uma direção
- O estímulo é transmitido uni-direcionalmente, neurônio por neurônio, da camada de entrada até a camada de saída
- É formada pela combinação das funções implementadas por neurônios das camadas anteriores

# Redes Multilayer Perceptron

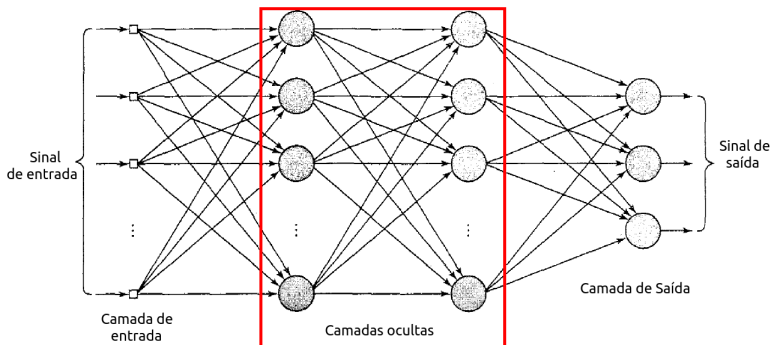
- Camada de entrada: conjunto de unidades sensoriais que recebem os dados (estímulos)
- “Linhas convexas no espaço de decisão”





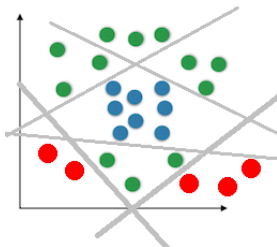
# Redes Multilayer Perceptron

- Camada(s) oculta(s): os neurônios não se conectam dentro de uma mesma camada e não geram a saída da rede
- “Regiões convexas” → número de lados = número de unidades da camada anterior



# Redes Multilayer Perceptron

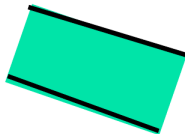
- **Camada(s) oculta(s):** gera uma codificação interna baseada em uma transformação não-linear dos dados
- Cada neurônio da camada oculta é uma perceptron simples
- Portanto, é uma combinação de vários problemas linearmente separáveis



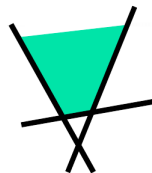
# Regiões Convexas



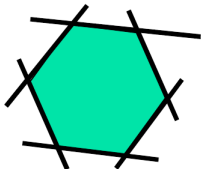
Aberta



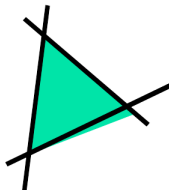
Aberta



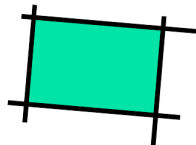
Aberta



Fechada

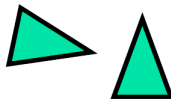
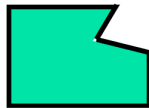


Fechada



Fechada

# Regiões Convexas



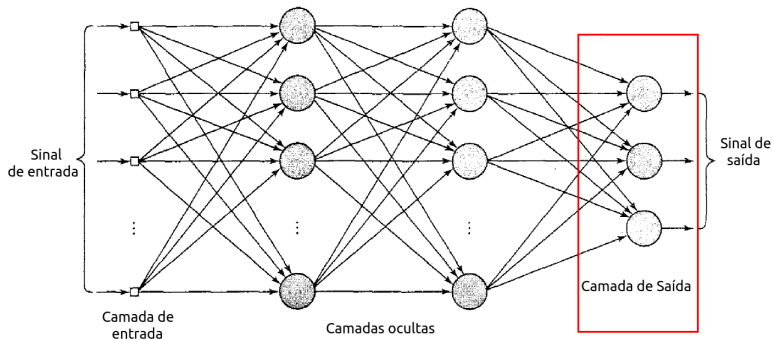
- **Camada(s) oculta(s):** quantos neurônios utilizar na camada oculta?
- Depende do problema, do número de atributos, da quantidade de classes...
- Em geral, duas camadas são suficientes para uma rede neural

# Regiões Convexas

Perceptron structure	XOR problem	Meshed classes	General region
1 layer			
2 layer			
3 layer			

# Redes Multilayer Perceptron



- Camada de saída: combina as saídas produzindo o resultado final
- “Combinações das figuras convexas produzindo figuras mais complexas” → número de figuras convexas = número de unidades da camada anterior

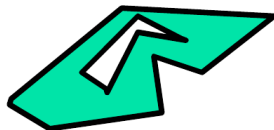




# Exercício

- Determine o número de camadas e pelo menos quantos neurônios em cada camada possui a rede que divide o espaço de entradas das formas abaixo:



 classe 1  
 classe 2

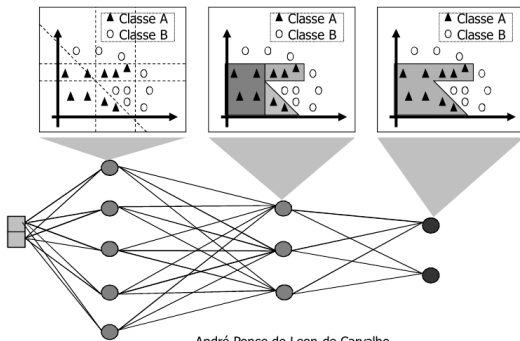


 classe 1  
 classe 2



# Exercício

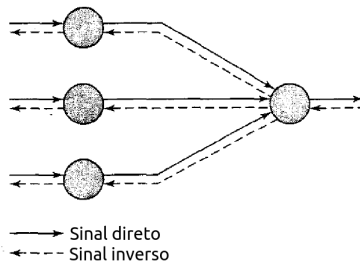
- Analise os gráficos abaixo e descreva o papel de cada camada de neurônios



- Como treinar uma rede MLP?
- Na fase de treinamento, o aprendizado é supervisionado
- Estender a técnica de gradiente descendente para calcular os pesos de cada neurônio
- Regra delta para MLPs  $\rightarrow$  Backpropagation para MLPs

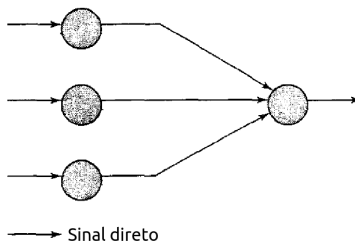
# Redes Multilayer Perceptron - Backpropagation

- Algoritmo é composto por duas fases:
  - Fase de propagação (forward)
  - Fase de retropropagação (backward)



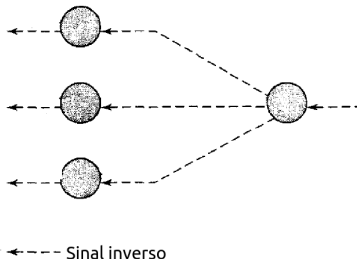
# Redes Multilayer Perceptron - Backpropagation

- Na fase de propagação, os estímulos (dados de entrada) são aplicados aos neurônios e os resultados são propagados, camada por camada, até o cálculo final da saída da rede
  - Os pesos sinápticos são mantidos fixos



# Redes Multilayer Perceptron - Backpropagation

- Na fase de retropropagação, ocorre a propagação do erro da camada de saída até as camadas iniciais
  - Os pesos sinápticos são ajustados de acordo com uma regra de correção de erro
  - Cálculo de derivadas

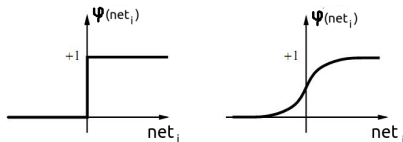


# Redes Multilayer Perceptron - Backpropagation

- Um sinal de erro é computado com base na diferença entre as respostas real e esperada da rede
  - Tal sinal é propagado no sentido contrário aos estímulos na rede
- Durante o treinamento, o valor dos pesos sinápticos são ajustados e reajustados visando diminuir o erro
  - ... isto é, aproximar a resposta de saída da rede com a resposta esperada

# Redes Multilayer Perceptron

- **Problema:** O cálculo das derivadas demanda funções diferenciáveis
  - O uso de funções de ativação do tipo sinal e degrau são funções descontínuas
- **Solução:** Uso de funções de ativação contínuas
  - Aproximações suaves das funções “hard”



- **Função de ativação:** Por quê uma função de ativação não-linear e contínua?
- O sinal de erro é facilmente calculado na última camada. No entanto, isso não acontece com as camadas ocultas
- O uso de uma função de ativação “hard” tornaria a avaliação precisa do erro muito difícil:
  - As distâncias entre as respostas que estão sendo geradas pelos neurônios das camadas internas e da camada de saída da rede e os respectivos valores desejados não oferecem precisão
  - Tais funções de ativação lineares calculam valores “discretos”



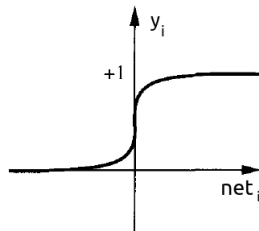
- **Função de ativação:** Por quê uma função de ativação não-linear?
- Os neurônios das camadas internas da rede recebem uma estimativa muito mais precisa do erro calculado na camada de saída, uma vez que os valores variam gradativamente dentro de um intervalo, fazendo com que os pesos internos sejam ajustados da melhor maneira possível na retropropagação
- Além disso, as funções contínuas e não-lineares são diferenciáveis

- Função sigmoidal:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

- Derivadas são fáceis de calcular

$$f'(x) = x(1 - x) \quad (2)$$



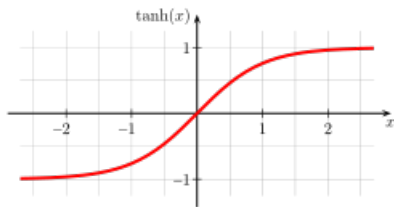
# Redes Multilayer Perceptron

- Função tangente hiperbólica:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3)$$

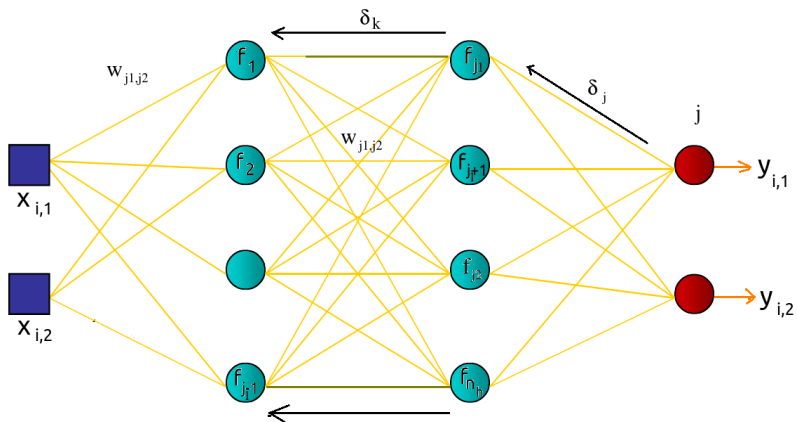
- Derivadas são fáceis de calcular

$$f'(x) = 1 - \tanh^2(x) \quad (4)$$



- **Função de ativação:** Por quê uma função de ativação não-linear?
- Saída da função de ativação é interpretada como:
  - Taxa média de disparo (TMD) de um neurônio

# Notações



# Redes Multilayer Perceptron - ALGORITMO

## Backpropagation

- 1 Inicializar aleatoriamente os pesos  $\{w_{j_1,j_2}\}$  e os bias  $\{b_{j_1,j_2}\}$  e o valor da taxa de aprendizado  $\eta$
- 2 Para cada par de treinamento  $(\mathbf{x}_i, l(\mathbf{x}_i))$ , faça  
// Fase de propagação (forward)
- 3     Aplicar o padrão de entrada à camada de entrada da rede;
- 4     Para cada camada  $c$  ( $c > 1$ ), processar o sinal de entrada e calcular a saída de seus neurônios, aplicando-a como entrada na camada  $c + 1$ ;
- 5     Calcular a saída  $y_{i,j}$  para cada neurônio da camada de saída  $j$
- 6     Calcular os erros produzidos na camada de saída da rede.

# Redes Multilayer Perceptron - ALGORITMO

## Backpropagation

// Fase de propagação (forward) // Fase de retropropagação (backward)

- 7 Para cada neurônio da camada de saída, calcule o erro

$$\delta_j \leftarrow y_{i,j}(1 - y_{i,j})(l(\mathbf{x}_i) - y_{i,j}) \quad (5)$$

- 8 Para cada neurônio da camada oculta, calcule o erro

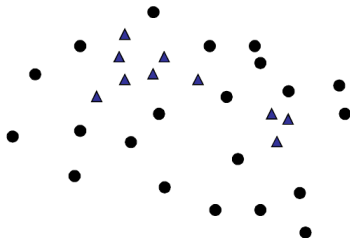
$$\delta_k \leftarrow y_{i,k}(1 - y_{i,k}) \sum_{j \in \text{saída}} w_{j,k} \delta_j \quad (6)$$

- 9 Atualize cada peso da rede  $w_{j_1,j_2}$

$$\begin{aligned} w_{j_1,j_2}^{t+1} &\leftarrow w_{j_1,j_2}^t + \Delta w_{j_1,j_2}^t \\ \Delta w_{j_1,j_2}^t &\leftarrow \eta \delta_{j_1} y'_{j_1,j_2} \end{aligned}$$

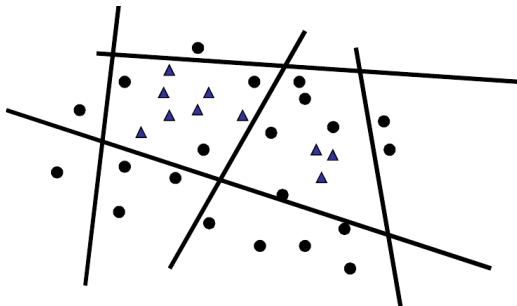
em que  $y'_{j_1,j_2}$  é a saída do neurônio  $j_1$  e entrada no neurônio  $j_2$

# Redes Multilayer Perceptron - Treinamento

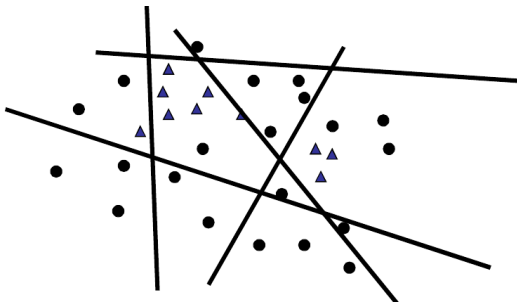




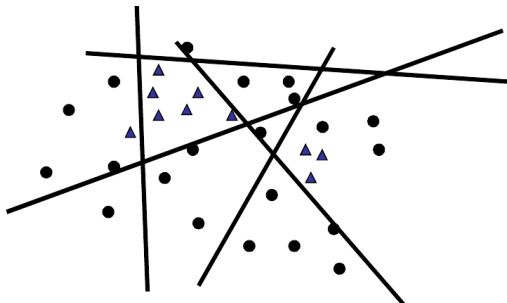
# Redes Multilayer Perceptron - Treinamento



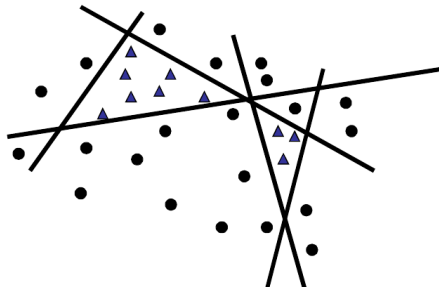
# Redes Multilayer Perceptron - Treinamento



# Redes Multilayer Perceptron - Treinamento



# Redes Multilayer Perceptron - Treinamento



# Redes Multilayer Perceptron - Problemas no aprendizado

- Ocorrências de mínimos locais
  - Soluções nem sempre são as melhores
  - Maneiras de evitar: modo de inicialização dos pesos
- Número de neurônios por camada
  - Número de exemplos de treinamento
  - Complexidade da função a ser aprendida
  - Quantidade de ruído

- Instabilidade numérica
  - Problemas na aproximação das funções de ativação
- Isso ocorre devido ao cálculo das derivadas
  - Métodos numéricos (otimização mais robusta)
  - Normalizar dados minimizam tais efeitos indesejados

- Overfitting
  - Depois de um certo ponto do treinamento, a rede piora ao invés de melhorar
  - Se especializa em padrões de treinamento, incluindo suas peculiaridades (piora generalização)
- Alternativas
  - Encerrar treinamento mais cedo (early stop)
  - Poda de conexões e neurônios irrelevantes (pruning)
  - Penalização dos valores (norma) dos pesos (weight decay)

- RUSSELL, Stuart J.; STUART, J. Norvig. Artificial Intelligence: A Modern Approach, Chapter 10 p. 320-374, 2003.
- TAN, Pang-Ning et al. Introduction to data mining. Boston: Pearson Addison Wesley, 2006.
- Alguns slides foram adaptados do material do Prof. André Backes (FACOM/UFU) <sup>1</sup>
- Alguns slides foram adaptados do material do Prof. João Garcia (ICMC/USP) <sup>2</sup>

---

<sup>1</sup>Acessado em 17 de fevereiro de 2016

<http://www.facom.ufu.br/~backes/pgc204/Aula07-RedesNeurais.pdf>

<sup>2</sup>Acessado em 16 de fevereiro de 2016

<http://wiki.icmc.usp.br/images/6/66/SCC5809Cap1.pdf>