

# Introdução a Redes Neurais Artificiais

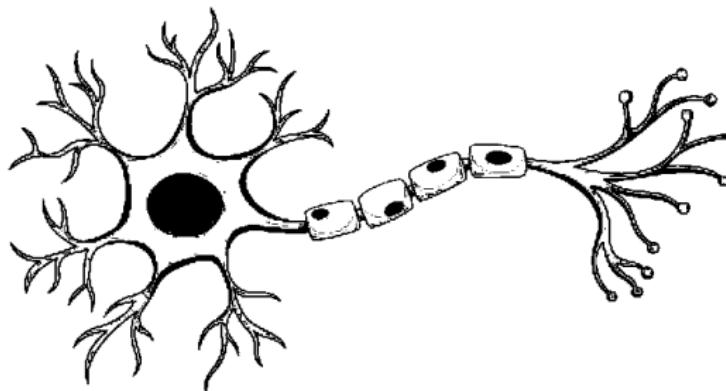
Prof. Jefferson T. Oliva

Aprendizado de Máquina e Reconhecimento de Padrões (AM28CP)  
Engenharia de Computação  
Departamento Acadêmico de Informática (Dainf)  
Universidade Tecnológica Federal do Paraná (UTFPR)  
Campus Pato Branco



- Redes Neurais Artificiais
- Perceptron
- Perceptron Multicamadas

- Redes neurais artificiais são modelos matemáticos inspirados na estrutura neural biológica
  - Compostas por unidades de processamento simples, denominados neurônios artificiais
  - Os neurônios artificiais são interconectados e comumente organizados em camadas



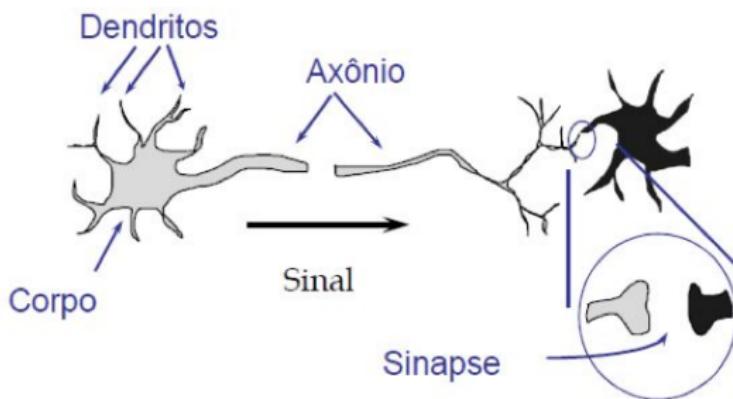
- Breve histórico

- McCulloch & Pitts (1943): modelo matemático do neurônio
- Hebb (1949): formulação explícita de uma regra fisiológica para modificação sináptica (“Postulado de Aprendizado de Hebb”)
- Rosenblatt (1958): rede Perceptron
- Minsky & Papert (1969): demonstraram limitações dos Perceptrons de uma única camada
- Hopfield (1982): “física com redes neurais”
- Kohonen (1982): mapas auto-organizáveis
- Rumelhart, Hinton & Williams (1986): Backpropagation

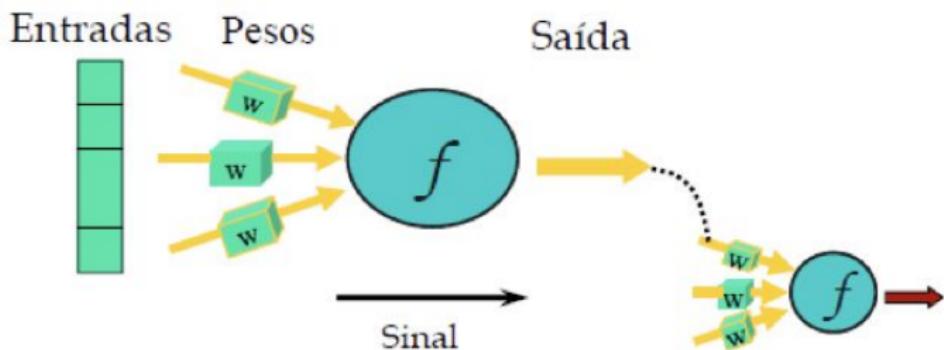
## **Redes Neurais Artificiais**

- Redes Neurais Artificiais (RNAs) fornecem um método geral e prático para a aprendizagem de funções de valor real e de valor discreto a partir de exemplos
- A aprendizagem de redes neurais é robusta a erros e ruídos nos dados de treinamento
- RNA é um modelo inspirado na aprendizagem de sistemas biológicos redes complexas de neurônios interconectados
- Trabalhos em RNAs começaram com o desejo de entender o cérebro

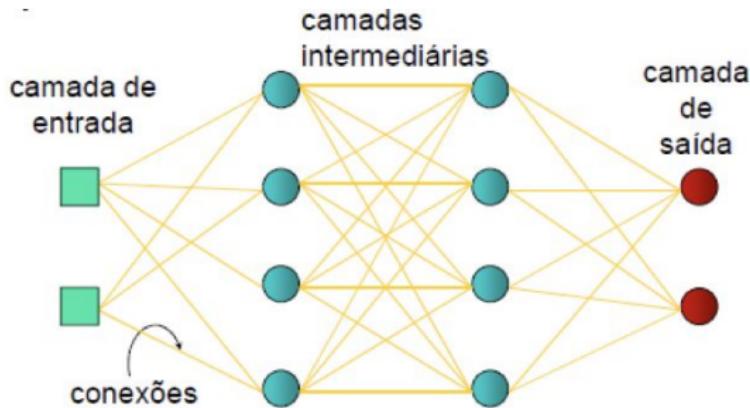
- Atualmente, o objetivo principal (ainda) de uma RNA é reproduzir seu funcionamento em diversas tarefas: paradigma conexionista



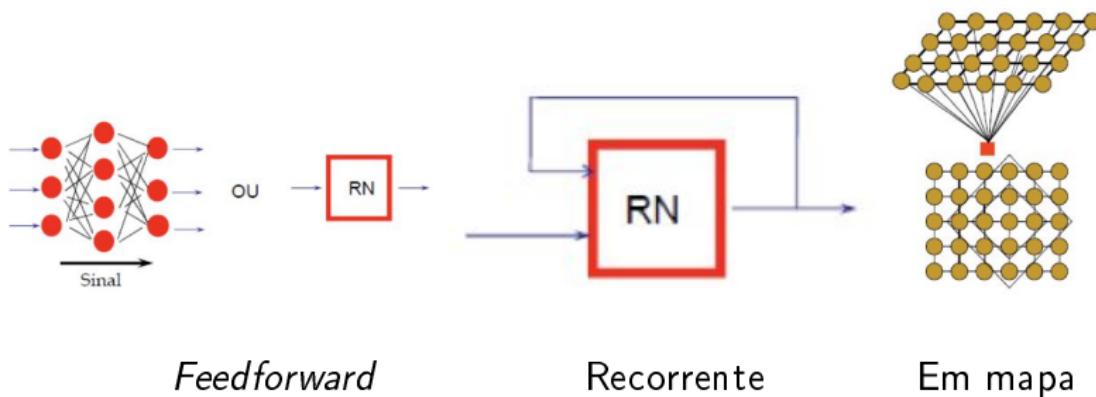
- Neurônio artificial



- Estrutura genérica de uma RNA



- Diferentes arranjos, topologia
  - *Feedforward*, recorrente, em mapa



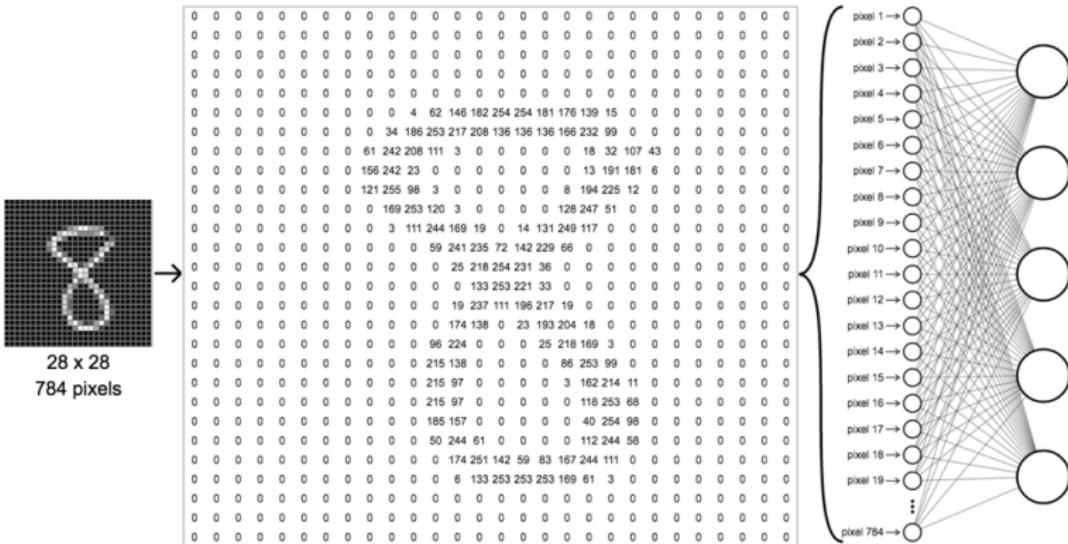
*Feedforward*

Recurrente

Em mapa

## Redes Neurais Artificiais

- Exemplo

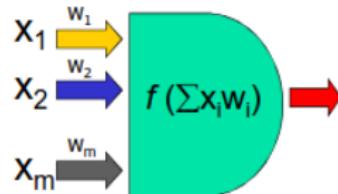


- Ciclo do projeto de uma RNA
  - ① Obtenção de dados
  - ② Escolha de características
  - ③ Treinamento (aprendizagem)
  - ④ Avaliação
  - ⑤ Complexidade computacional

- Principais aspectos das RNA
  - Arquitetura
    - Neurônios artificiais (unidades de processamento)
    - Conexões
    - Topologia
  - Aprendizado
    - Algoritmos
    - Paradigmas

- Unidades de processamento
  - Recebimento entradas de conjuntos de unidades
  - Aplicação de uma função sobre as entradas
  - Envio do resultado da função para um outro conjunto de unidades
  - Exemplos:

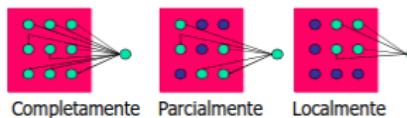
$$u = \sum_{i=1}^m x_i w_i$$



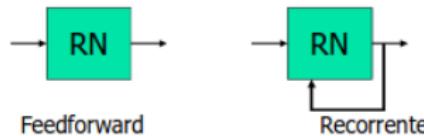
- Conexões
  - Definem a interligação entre os neurônios
  - Codificação do conhecimento da rede
  - Conexão inibitória: ( $w_{ik}(t) < 0$ )
  - Conexão excitatória: ( $w_{ik}(t) > 0$ )

- Topologia

- Número de camadas
- Cobertura das conexões



- Arranjo das conexões



- Aprendizado
  - Define como os parâmetros de rede são ajustados
  - Principais formas de ajuste
    - Correção de erro
    - Hebbiano
    - Competitivo
    - Termodinâmico (Boltzmann)
- Paradigma de aprendizado
  - Define as informações externas que são passadas para a rede durante o processo de aprendizado
  - Exemplos de abordagens: supervisionado, não-supervisionado, semi-supervisionado, reforço

A mostly complete chart of

## Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

Backfed Input Cell

Input Cell

Noisy Input Cell

Hidden Cell

Probabilistic Hidden Cell

Spiking Hidden Cell

Output Cell

Match Input Output Cell

Recurrent Cell

Memory Cell

Different Memory Cell

Kernel

Convolution or Pool

Perceptron (P)

Feed Forward (FF)

Radial Basis Network (RBF)

Recurrent Neural Network (RNN)

Long / Short Term Memory (LSTM)

Gated Recurrent Unit (GRU)

Auto Encoder (AE)

Variational AE (VAE)

Denoising AE (DAE)

Sparse AE (SAE)

Markov Chain (MC)

Hopfield Network (HN)

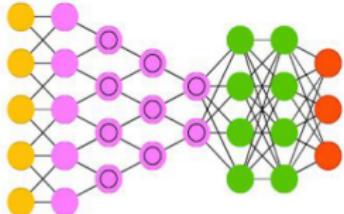
Boltzmann Machine (BM) Restricted BM (RBM)

Deep Belief Network (DBN)

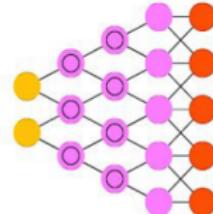


# Redes Neurais Artificiais

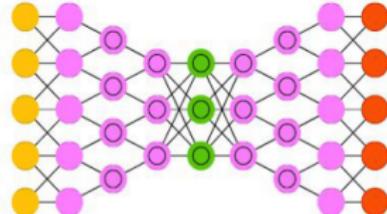
Deep Convolutional Network (DCN)



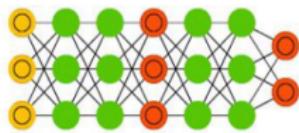
Deconvolutional Network (DN)



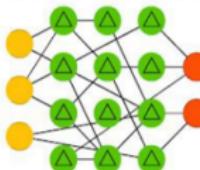
Deep Convolutional Inverse Graphics Network (DCIGN)



Generative Adversarial Network (GAN)



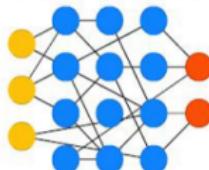
Liquid State Machine (LSM)



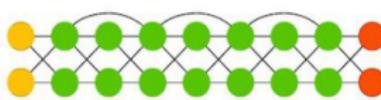
Extreme Learning Machine (ELM)



Echo State Network (ESN)



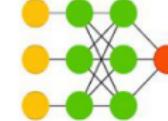
Deep Residual Network (DRN)



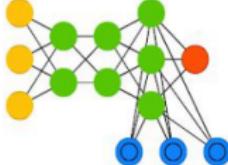
Kohonen Network (KN)



Support Vector Machine (SVM)



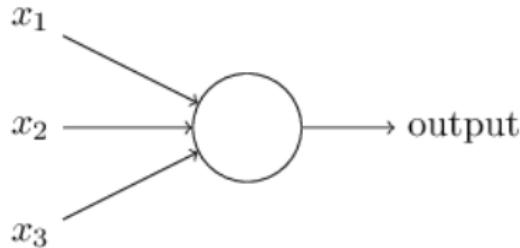
Neural Turing Machine (NTM)



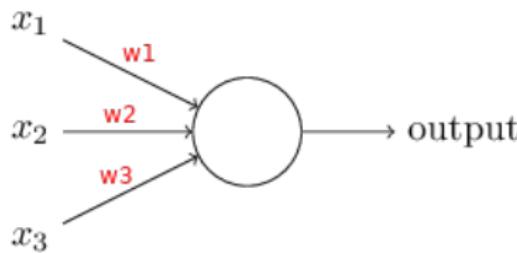
## Perceptron

# Perceptron

- Rede neural elementar baseada em uma unidade chamada Perceptron criada por Rosenblatt em 1958
  - O Perceptron permite uma compreensão clara de como funciona uma rede neural em termos matemáticos
- Um Perceptron é um modelo matemático que recebe várias entradas,  $x_1, x_2, \dots$  e produz uma única saída binária



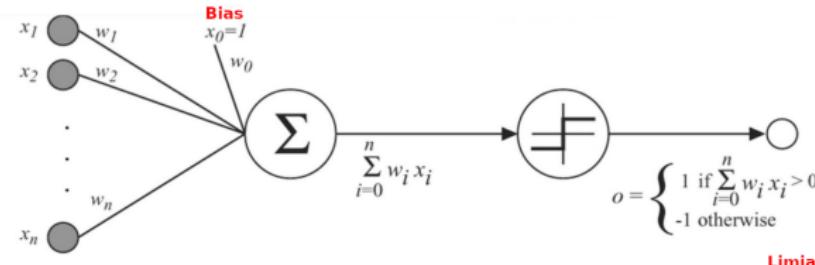
# Perceptron



- Nesse exemplo, o Perceptron possui três entradas:  $x_1, x_2, x_3$
- Rosenblatt propôs uma regra simples para calcular a saída: introdução de pesos,  $w_1, w_2, \dots$ , números reais expressando a importância das respectivas entradas para a saída
- A saída do neurônio, 1 ou -1, é determinada pela soma ponderada,  $\sum_j w_j x_j$ , menor ou maior do que algum valor limiar (*threshold*)

# Perceptron

- Modelo desenvolvido por Rosenblatt em 1958

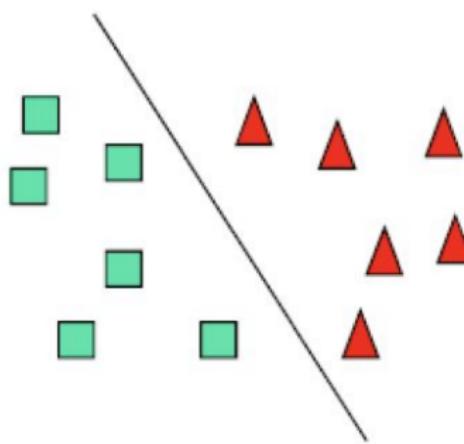


$$o(x_1, \dots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \dots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Limiar

- Onde:
  - Cada elemento  $w_i$  é uma constante de valor real, ou peso, que determina a contribuição da entrada  $x_i$  na saída do perceptron
  - A aprendizagem do perceptron envolve a escolha dos pesos de  $w_0$  a  $w_n$
  - A camada de entrada deve possuir uma unidade especial conhecida como bias, que é uma entrada de valor constante associada a um peso  $w_i$  em cada neurônio  $i$

- Podemos “ver” o perceptron como uma superfície de separação em um espaço de instâncias
  - O perceptron fornece “1” para instâncias dispostas em um lado do hiperplano e “-1” para instâncias dispostas no outro lado
  - Um único perceptron consegue separar somente conjuntos de exemplo linearmente separáveis



# Perceptron

## Regras de treinamento de perceptron

- Como aprender os pesos para um perceptron?
  - Problema: determinar um vetor de pesos que faça o perceptron produzir a saída correta (-1 ou +1) para cada um dos exemplos de treinamento
  - Solução: Começar com um vetor de pesos aleatórios e aplicar iterativamente a regra perceptron para cada exemplo de treinamento, modificando os pesos cada vez que ele classificar um exemplo erroneamente
    - Este processo é repetido várias vezes até que o perceptron classifique todos os exemplos de treinamento corretamente ou a taxa de erro ser aceitável

# Perceptron

## Regras de treinamento de perceptron

- Os pesos do perceptron são modificados a cada passo de acordo com a regra de treinamento do perceptron, que modifica o peso  $w_i$  associado a entrada  $x_i$  de acordo com a regra:

$$w_i = w_i + \Delta w_i$$

- Onde

$$\Delta w_i = \eta(t - o)x_i$$

- $t$  é o valor alvo para o exemplo de treinamento (alvo = classe)
- $o$  é a saída gerada pelo perceptron (o que ele classificou)
- $\eta$  é uma constante pequena (0.1) chamada de taxa de aprendizagem

# Perceptron

## Regras de treinamento de perceptron

- Se o exemplo de treinamento é classificado corretamente:

$$(t - o) = 0 \quad \Delta w_i = 0$$

- Se o exemplo de treinamento é classificado incorretamente, o valor de  $\Delta w_i$  é alterado:

- Se  $x_i = 0,8$ ,  $\eta = 0,1$ ,  $t = 1$ ,  $o = -1$

- A atualização do peso é dado por:

$$\Delta w_i = \eta(t - o)x_i = 0,1(1 - (-1))0,8 = 0.16$$

- Pode-se provar que este procedimento de aprendizagem converge dentro de um número finito de passos quando:
  - As classes dos dados de treinamento são linearmente separáveis
  - $\eta$  é suficientemente pequeno

# Perceptron

## Regras de treinamento de perceptron

1 Iniciar todos os pesos  $w_i$

2 Repita

Para cada par de treinamento ( $x, d$ )

Calcular a saída  $y$

Se ( $d \neq y$ ) Então

Atualizar pesos dos neurônios

Até o erro ser aceitável

1 Apresentar padrão  $x$  a ser reconhecido

2 Calcular a saída  $y$

3 Se ( $y = -1$ ) Então

$x \in$  classe -1

Senão

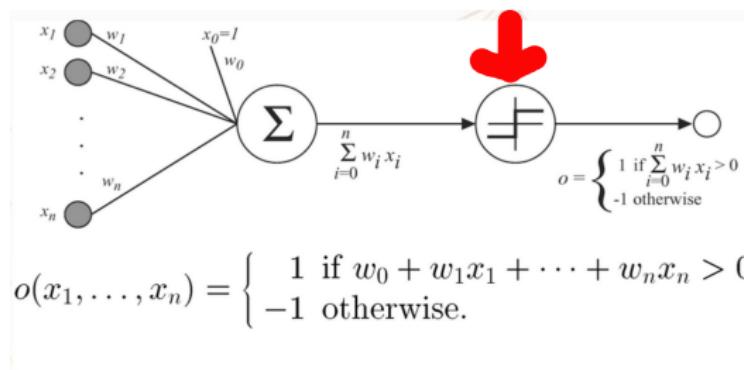
$x \in$  classe 1

# Perceptron

## Regras de treinamento de perceptron

- Função de ativação

- A função de ativação basicamente decidem se um neurônio deve ser ativado ou não

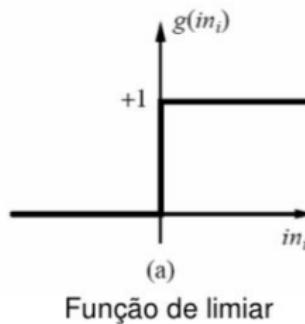


# Perceptron

## Regras de treinamento de perceptron

- Função de ativação

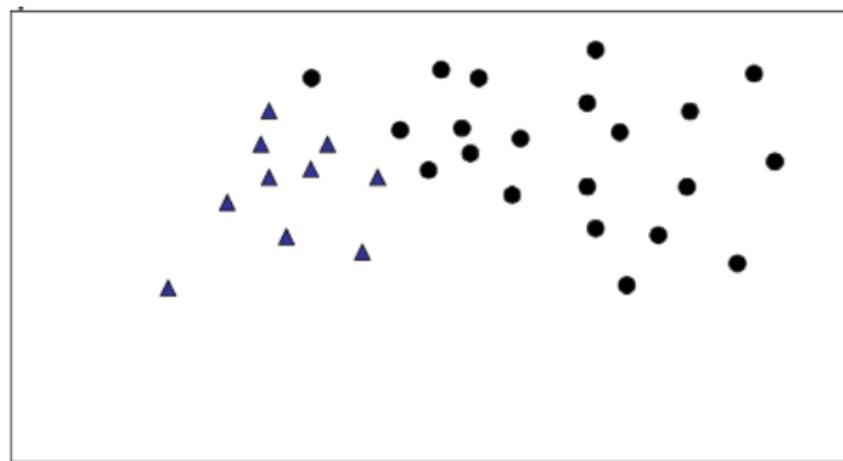
- Existem vários tipos de função de ativação, cada qual usado em diferentes situações
- No caso do Perceptron, é uma função de limiar (também chamado degrau)
  - Nesse caso depois de calculado a somatória de  $x_i w_i$ , é verificado o limiar para encontrar qual classe pertence



# Perceptron

Regras de treinamento de perceptron

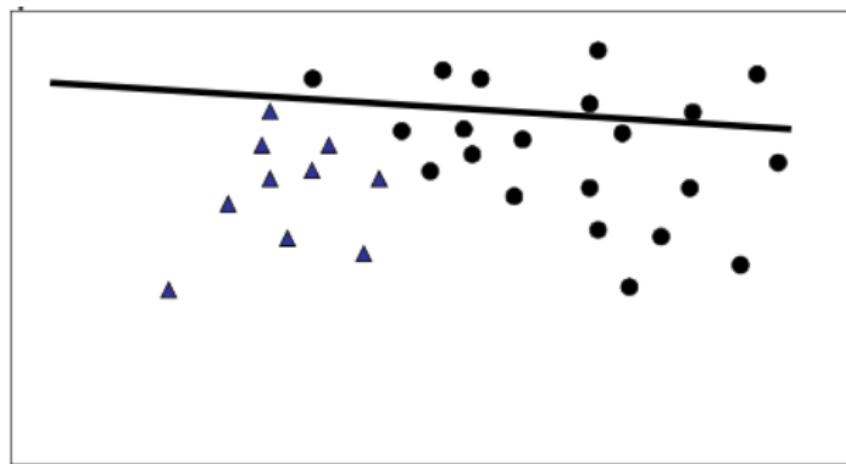
- Modificando fronteiras



# Perceptron

## Regras de treinamento de perceptron

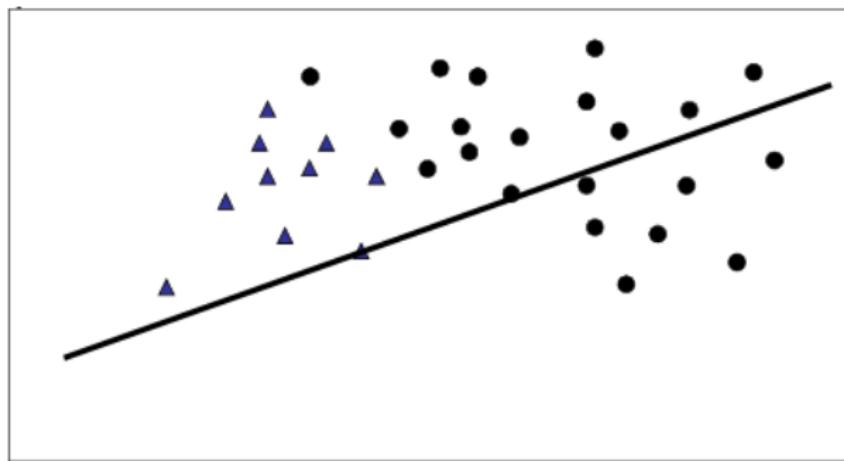
- Modificando fronteiras



# Perceptron

Regras de treinamento de perceptron

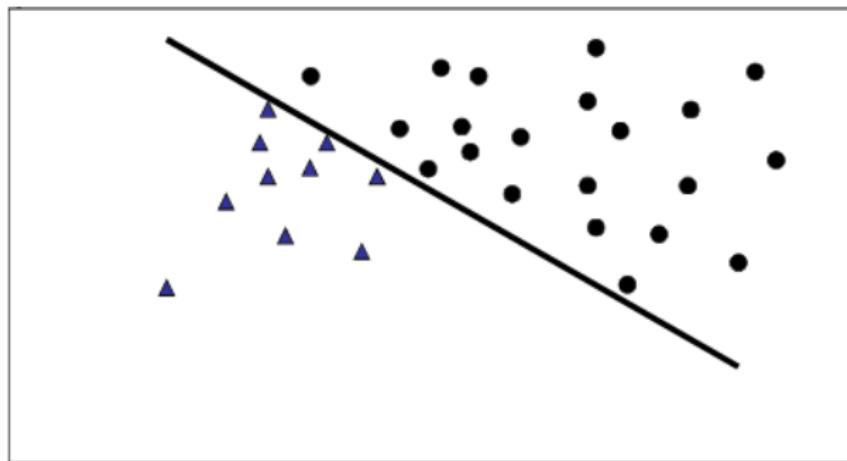
- Modificando fronteiras



# Perceptron

Regras de treinamento de perceptron

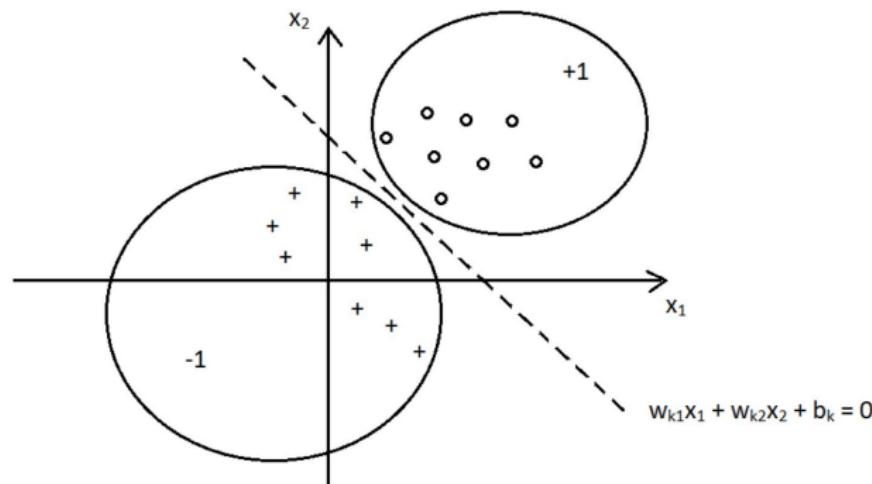
- Modificando fronteiras



# Perceptron

## Regras de treinamento de perceptron

- Objetivo do perceptron com função de ativação

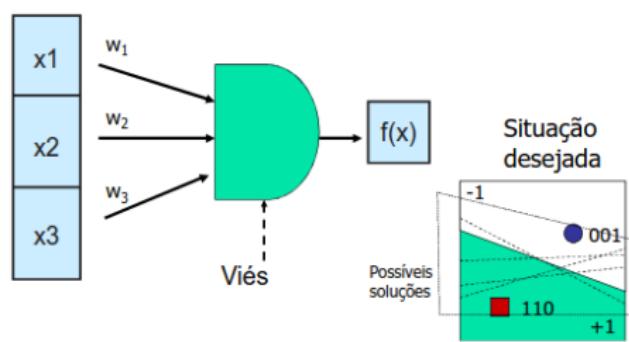


# Perceptron

## Regras de treinamento de perceptron

- Exemplo

- Dada uma rede perceptron com três entradas, pesos  $w_1 = 0,4$ ,  $w_2 = -0,6$  e  $w_3 = 0,6$ , e limiar (viés)  $\theta = 0,5$
- Ensinar a rede com os exemplos  $(001, -1)$  e  $(110, +1)$ , considerando uma taxa de aprendizado  $\eta = 0,4$
- Classificar os exemplos: 111, 000, 100 e 011



# Perceptron

## Regras de treinamento de perceptron

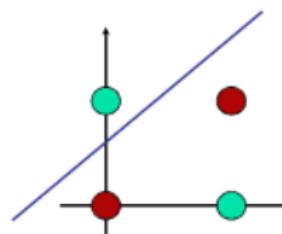
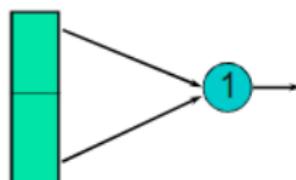
- Exemplo: treinamento para o exemplo 001, cuja classe é  $y = -1$ 
  - Passo 1: determinar a saída da rede
    - $u = -1(0, 5) + 0(0, 4) + 0(-0, 6) + 1(0, 6) = 0, 1$
    - $f(u) = +1 \quad (0, 1 \geq 0)$
  - Passo 2: atualizar os pesos ( $y \neq f(u)$ ):  $w_i = w_i + x_i \eta(t - o)$ 
    - $w_0 = 0, 5 + 0, 4(-1)(-1 - (+1)) = 1, 3$
    - $w_1 = 0, 4 + 0, 4(0)(-1 - (+1)) = 0, 4$
    - $w_2 = -0, 6 + 0, 4(0)(-1 - (+1)) = -0, 6$
    - $w_3 = 0, 6 + 0, 4(1)(-1 - (+1)) = -0, 2$

- Exemplo: teste para classificar os exemplos 111, 000, 100 e 011
  - Pesos aprendidos: 0,5, 1,2, 0,2 e -0,2
  - Exemplo 111
    - $u = -1(0,5) + 1(1,2) + 1(0,2) + 1(-0,2) = 0,7$
    - $f(u) = +1 \ (0,7 \geq 0)$
  - Exemplo 000
    - $u = -1(0,5) + 0(1,2) + 0(0,2) + 0(-0,2) = -0,6$
    - $f(u) = -1 \ (0,5 < 0)$

# Perceptron

- O perceptron falha em convergir se as classes forem linearmente não-separáveis
  - Exemplo: XOR

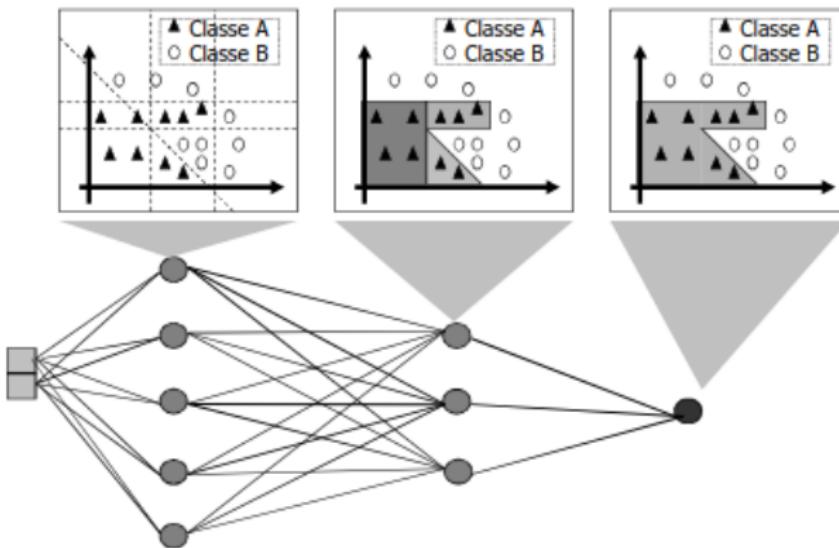
0, 0 → 0
0, 1 → 1
1, 0 → 1
1, 1 → 0



## Perceptron Multicamadas

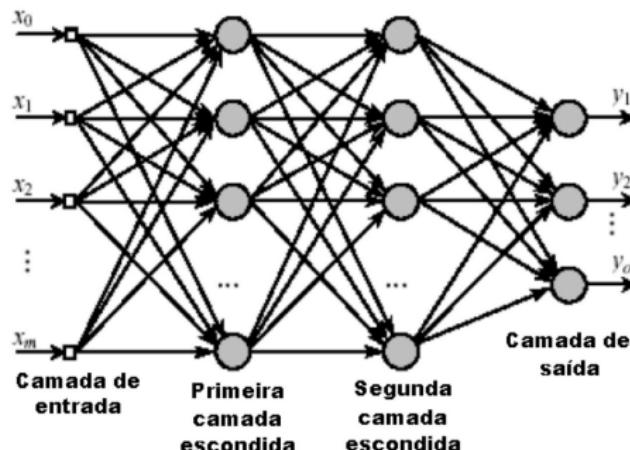
# Perceptron Multicamadas

- É um perceptron que se une a perceptrons adicionais, empilhados em várias camadas, para resolver problemas complexos



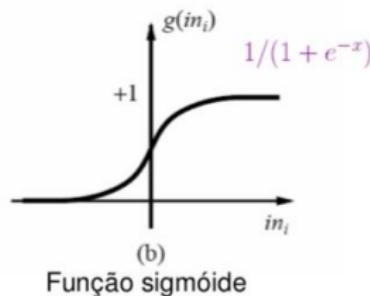
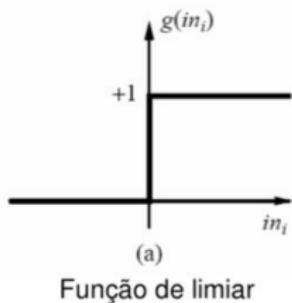
# Perceptron Multicamadas

- Uma rede neural com ao menos uma camada intermediária e um modelo denominada perceptron de múltiplas camadas (MLP – *multilayer perceptron*)
- Cada entrada envia vários sinais
  - Um sinal indo para cada perceptron na próxima camada
  - Para cada sinal, o perceptron usa pesos diferentes



# Perceptron Multicamadas

- Vimos que a estrutura de um perceptron clássico inclui uma função de ativação com limiar que, tendo em vista a natureza do problema de classificação, é abrupta
- MLPs utilizam funções de ativação sigmoidais, que possuem um perfil de "S", ou seja, de um degrau suave

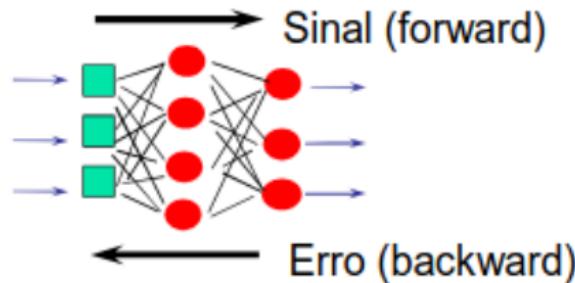


- No início do treinamento do classificador, a rede comete erros o tempo todo!
- Gradualmente, o desempenho da rede melhora no decorrer do processo de treinamento
- Após a ocorrência de erros propagados entre as camadas durante o treinamento, a rede volta atrás, a partir da última camada até a primeira, para a correção de erros
  - Esse processo iterativo de passar uma amostra pela rede e voltar atrás para se corrigir é conhecido como aprendizado profundo, e é o que torna as redes com inteligência semelhante à humana
  - O treinamento de redes MLPs é feito utilizando o algoritmo *backpropagation*

# Perceptron Multicamadas

Algoritmo *backpropagation*

- Treinamento em duas fases
  - *Forward*
  - *Backward*



# Perceptron Multicamadas

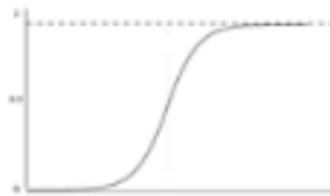
## Algoritmo *backpropagation*

- Treinamento
  - Supervisionado
  - Uso de gradiente descendente
  - Ajuste dos pesos:  $\Delta w_{ij} = \eta x_i \delta_j$ 
    - $\delta_j = f'(net)erro_j$ , se  $j$  for camada de saída
    - $\delta_j = f'(net) \sum w_{jk} \delta_k$ , se  $j$  for camada intermediária
    - $erro_j = \frac{1}{2} \sum_{q=1}^c (y_q - f(net_q))^2$
    - $net = \sum_{i=0}^m x_i w_i$
  - Se  $f(net)$  for uma função sigmoidal,  
 $f'(net) = f(net)(1 - f(net))$
  - Convergência não é garantida no treinamento do modelo!

# Perceptron Multicamadas

## Algoritmo *backpropagation*

- Função de ativação
  - Não linear
  - Diferenciável, contínua e não decrescente (geralmente)
  - Sigmoidal
    - Logística:  $f(x) = \frac{1}{1+e^{-net}}$
    - Tangente hiperbólica:  $f(x) = \frac{1-e^{-net}}{1+e^{-net}}$



# Perceptron Multicamadas

Algoritmo *backpropagation*

- Treinamento

*Iniciar todas as conexões com valores aleatórios  $\in [a,b]$*

*Repete*

*erro = 0;*

*Para cada par de treinamento  $(X, y)$*

*Para cada camada  $k := 1$  a  $N$*

*Para cada neurônio  $j := 1$  a  $M_k$*

*Calcular a saída  $f_{kj}(\text{net})$*

*Se  $k = N$*

*Calcular soma dos erros de seus neurônios;*

*Se  $\text{erro} > \varepsilon$*

*Para cada camada  $k := N$  a 1*

*Para cada neurônio  $j := 1$  a  $M_k$*

*Atualizar pesos;*

*Até  $\text{erro} < \varepsilon$  (ou número máximo de ciclos)*

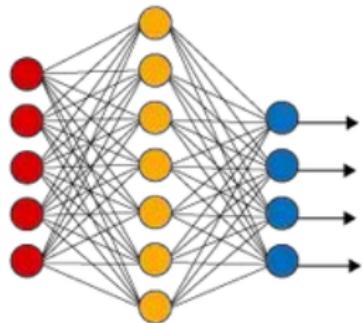
# Perceptron Multicamadas

## Algoritmo *backpropagation*

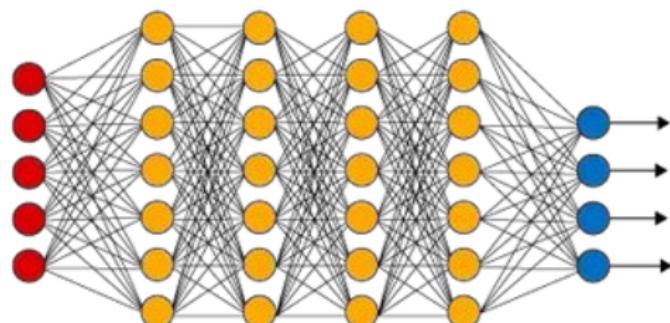
- *Forward* (propagação para frente)
  - Alimentação da camada de entrada
  - Ativação de cada neurônio utilizando ativações e pesos da camada anterior (se houver) e uma função de ativação
    - Essa ativação é feita em camada por camada até a de saída
  - Cálculo do erro gerado na última camada (função de perda)
- *Backward* (propagação para trás)
  - Propagação do erro camada por camada
  - Cálculo da contribuição (gradiente) de cada neurônio para o erro total
  - Atualização dos pesos

# Perceptron Multicamadas

**Simple Neural Network**



**Deep Learning Neural Network**



● Input Layer

● Hidden Layer

● Output Layer

-  BISHOP C.  
Neural Networks for Pattern Recognition.  
Oxford University Press, 1995.
-  DUDA R., Hart P., STORK D.  
Pattern Classification.  
Wiley Interscience, 2002.
-  CARVALHO, A. P. L. F.  
Árvores de Características. Aprendizado de Máquina.  
*Slides*. Ciência de Computação e Matemática Computacional.  
ICMC/USP, 2015.
-  HAYKIN S.  
Neural Networks: A Comprehensive Foundation.  
Prentice Hall, 1988.

-  MITCHELL T.  
Machine Learning.  
WCB McGraw-Hill, 1997.
-  RASCHKA, S.; MIRJALILI, V.  
*Python Machine Learning.*  
Packt, 2017.