

# REDES NEURAIS

---

Prof. André Backes | @progdescomplicada

# Sistema Nervoso

- O que é?
  - É um conjunto complexo de células que determina o funcionamento e comportamento dos seres vivos
    - Engloba o cérebro
- Sua unidade fundamental é o neurônio
  - Se diferencia de outras células por apresentar excitabilidade

# Cérebro

- Funciona de forma inteiramente diferente dos computadores convencionais
  - É lento
    - O tempo de propagação de um impulso no axônio é da ordem de milissegundos!
    - A frequência de disparo de um neurônio é da ordem de kHz!
  - Mas sua lentidão é compensada pela maciça conexão entre neurônios
    - Cada neurônio tem cerca de 10.000 sinapses com outros neurônios

# Cérebro

- Ainda, apesar de sua lentidão...
  - É mais preciso (e possivelmente mais rápido) que computadores convencionais na execução de tarefas complexas
    - Visão, audição, reconhecimento, ...

# Cérebro

- Possui a capacidade de construir suas próprias regras
  - uso da experiência
    - Um milhão de sinapses por segundo são desenvolvidas nos dois primeiros anos de vida
- Composto por várias regiões especializadas
  - Cada uma com funções específicas

# Redes Neurais Artificiais (NRA)

- Motivação
  - Computadores são eficientes em várias áreas, mas a computação convencional não tem obtido desempenho próximo da natureza em vários domínios
  - Vantagens da natureza
    - Seres humanos: reconhecer um rosto familiar em ambiente estranho
    - Morcegos: seu sonar pode reconhecer alvos (distância e velocidade)

# Redes Neurais Artificiais (NRA)

- Motivação

- Seu estudo surgiu com o desejo de entender o cérebro
  - Objetivo principal: reproduzir seu funcionamento em diversas tarefas
- Paradigma Bio-Inspirado de Aprendizado de Máquina (AM)
  - Paradigma conexionista: o comportamento inteligente está relacionado com a dinâmica das conexões entre neurônios.
  - Esta dinâmica é capaz de representar o conhecimento

# Redes Neurais Artificiais (NRA)

- São modelos de computação inspirados no cérebro humano
  - Compostas por várias unidades de processamento
    - Neurônios
  - Interligadas por um grande número de conexões
    - sinapses
- Como modelo, é apenas uma aproximação do fenômeno ou objeto real que se pretende estudar

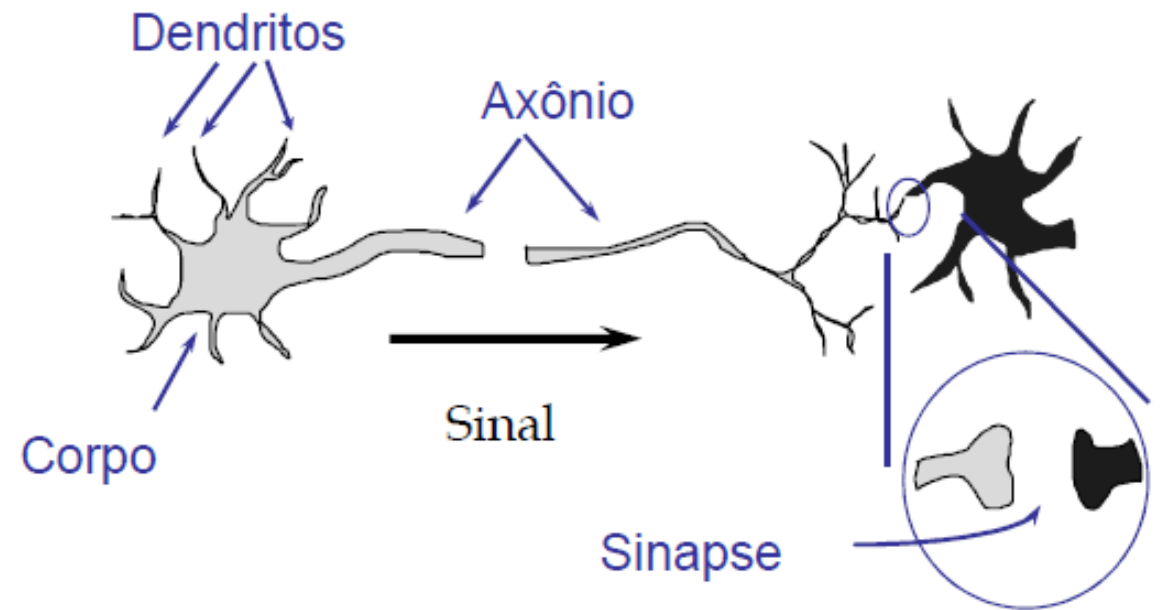


# Redes Neurais Artificiais (NRA)

- Propriedades particulares:
  - Aprender
  - Adaptar
  - Generalizar
  - Eventualmente organizar
- Eficientes em várias aplicações
  - Regressão, classificação, problemas não lineares, etc

# Neurônio

- O neurônio é o bloco construtivo básico de algoritmos de redes neurais
  - Neurônio natural simplificado

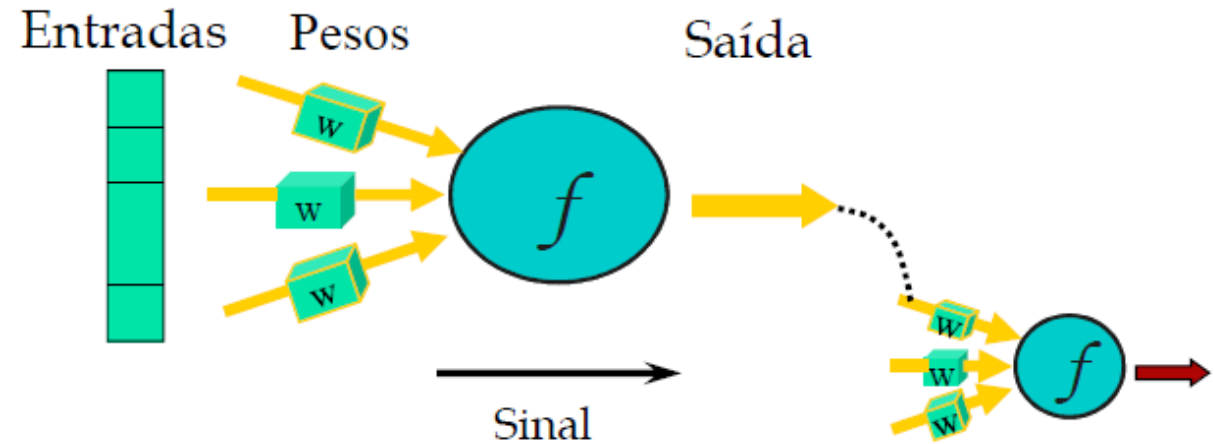


# Neurônio

- Neurônio natural simplificado
  - Dendritos
    - recebem impulsos nervosos oriundos de outros neurônios
  - Corpo da Célula
    - Processa a informação e gera novos impulsos
  - Axônio
    - Transmite os impulsos gerados para outros neurônios
  - Sinapse
    - Ponto de contato entre os axônios e os dendritos de dois neurônios
    - Controla a transmissão de impulsos, proporcionando a capacidade de adaptação do neurônio

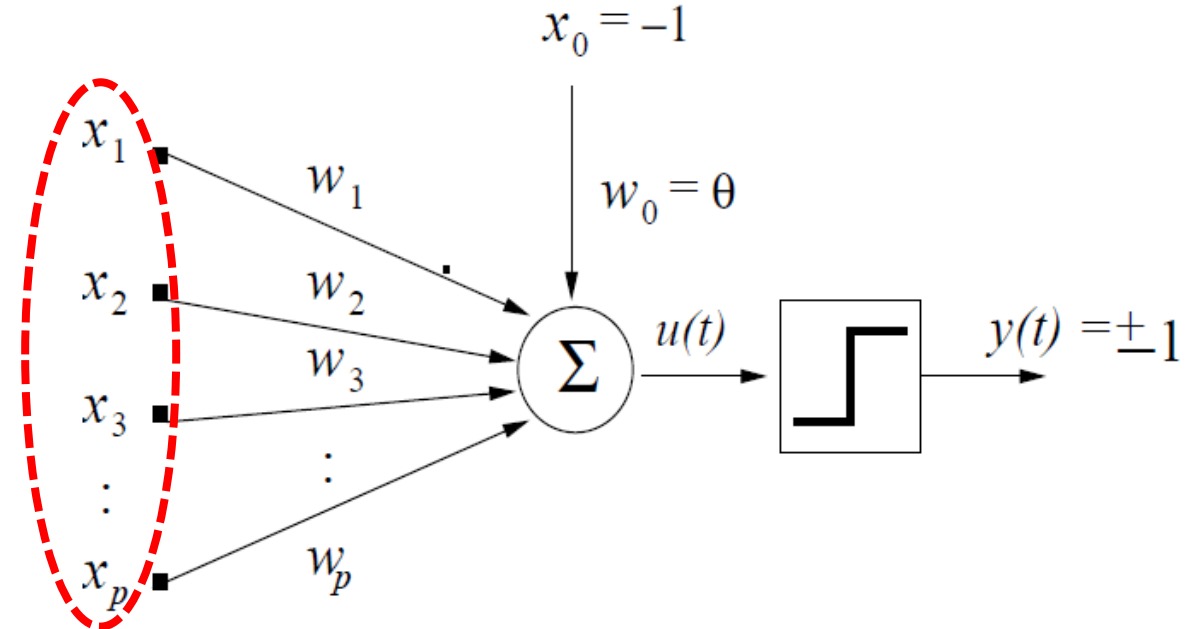
# Neurônio Artificial

- Modelo matemático de um neurônio biológico
  - Proposto inicialmente por McCulloch & Pitts (1943)
  - É uma aproximação útil de um neurônio real



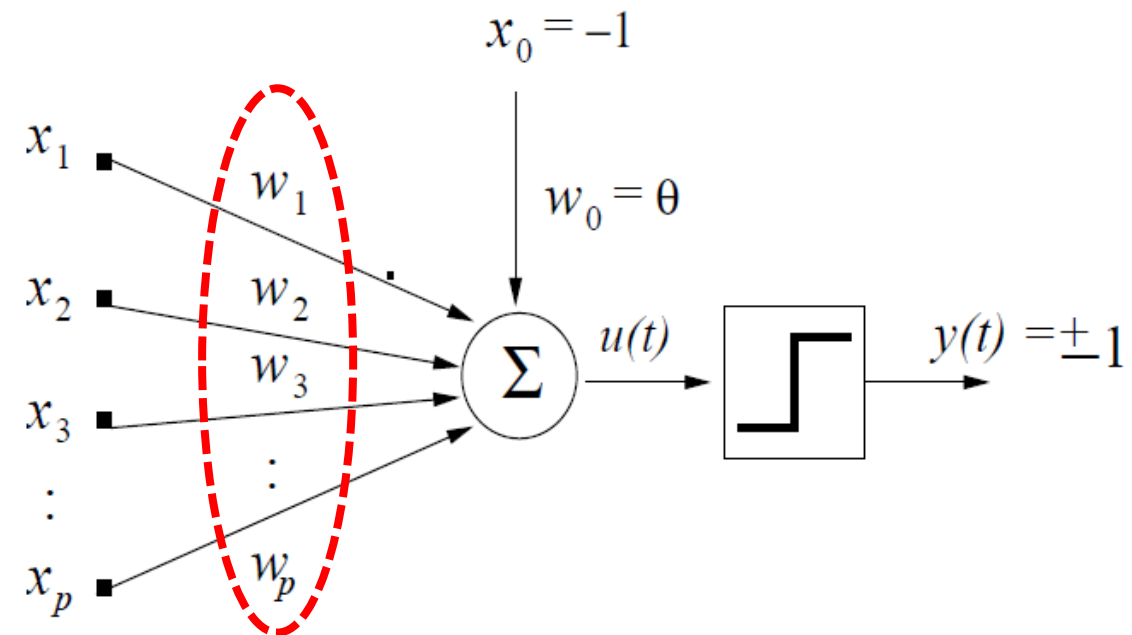
# Neurônio Artificial | Estrutura básica

- Os dendritos são modelados como uma linha ou canal de transmissão por onde flui a informação de entrada
- $x_i, i=1, \dots, p$



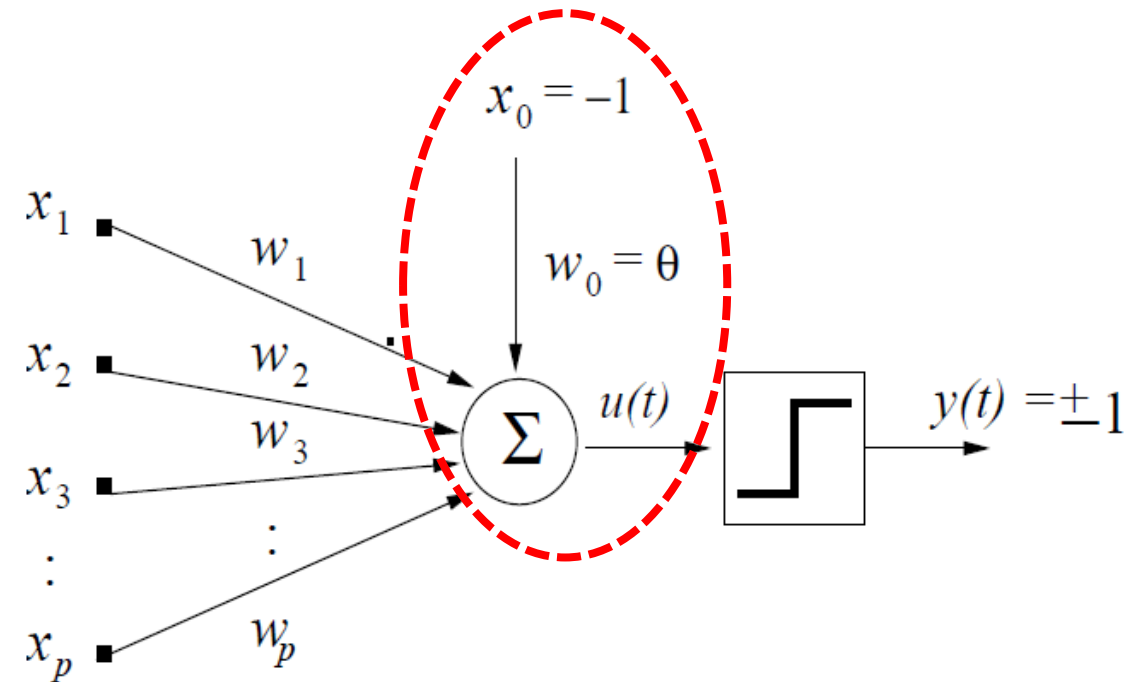
# Neurônio Artificial | Estrutura básica

- A força das conexões sinápticas dos dendritos é modelada como um fator (peso sináptico), cujo papel é modular o fluxo de sinais passando por eles
- $w_i, i=1, \dots, p$



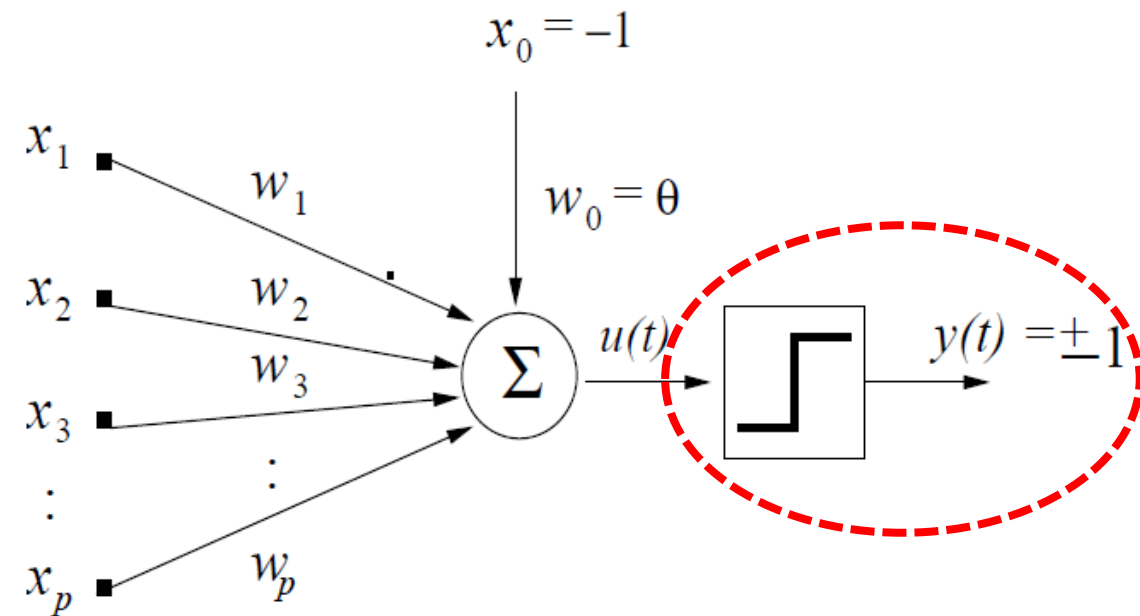
# Neurônio Artificial | Estrutura básica

- O corpo celular realiza o acúmulo energético
  - Somatório das entradas moduladas pelos pesos sinápticos:  $u = x_1 * w_1 + x_2 * w_2 + \dots + x_p * w_p - \Theta$
  - $\Theta$  : limiar (bias)



# Neurônio Artificial | Estrutura básica

- O axônio funciona como uma *função de ativação* (chave ON-OFF)
  - Indica se o neurônio respondeu ao estímulo atual
  - Indica se houve ou não o envio de um potencial de ação
    - $y = \text{sinal}(u) = +1$ , se  $u > 0$
    - $y = \text{sinal}(u) = -1$ , caso contrário





# Conceitos Básicos

- Principais aspectos das RNAs
  - Arquitetura
    - Unidades de Processamento
    - Topologia
  - Aprendizado
    - Paradigmas de aprendizado
    - Algoritmos de aprendizado

# Conceitos Básicos

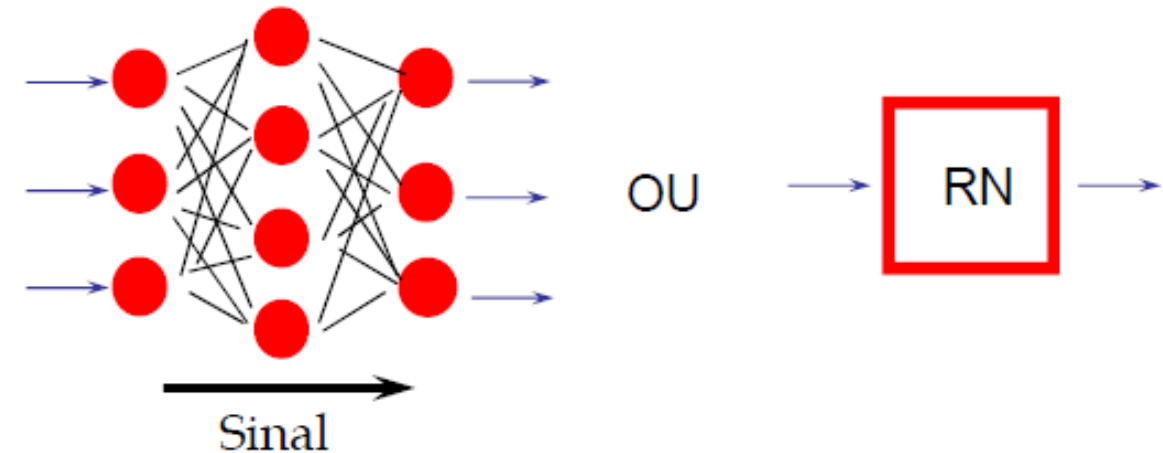
- Unidades de Processamento
  - Diferentes tipos de neurônios
    - Estáticos ou Dinâmicos
    - Atemporais ou Temporais
    - Lineares ou Não Lineares

# Conceitos Básicos

- Topologia
  - Diferentes quantidades de camadas
    - Uma camada: Perceptron, Adaline
    - Multi-camadas: Multi Layer Perceptron (MLP), Funções de Base Radial (RBF)

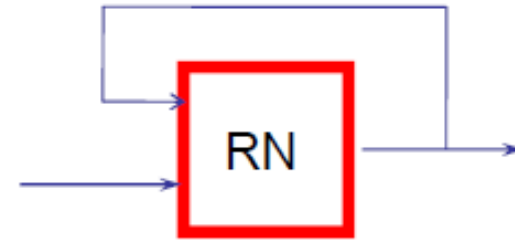
# Conceitos Básicos | Topologia

- Diferentes arranjos das conexões
  - Redes *feedforward*: não existem loops de conexões. É o tipo mais comum



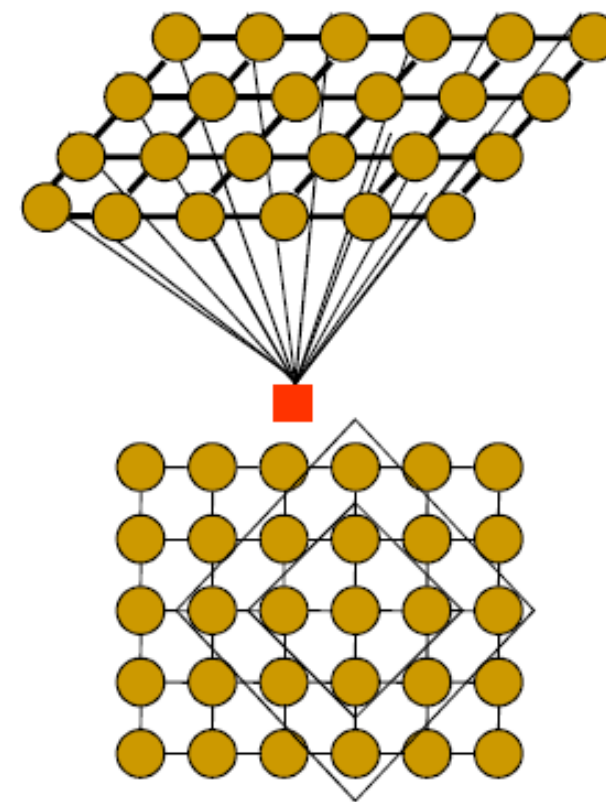
# Conceitos Básicos | Topologia

- Diferentes arranjos das conexões
  - Redes recorrentes: conexões apresentam loops, isto é, possuem conexões ligando neurônios de uma camada a neurônios de camada(s) anterior(es)
  - Podem “lembrar” excitações passadas



# Conceitos Básicos | Topologia

- Diferentes arranjos das conexões
  - Redes em mapas (ou grades): matriz n-dimensional de neurônios, com relação de vizinhança espacial



# Conceitos Básicos

- Paradigmas de aprendizado
  - Indicam como a RNA se relaciona com o ambiente externo
  - Principais Paradigmas
    - Supervisionado
    - Não supervisionado
    - Reforço

# Conceitos Básicos

- Algoritmos de aprendizado
  - Conjunto de regras bem definidas usadas para ensinar a rede a resolver um certo problema
  - Algumas das principais categorias
    - Correção de Erro
    - Competitivo
    - Hebbiano
  - Divergem na maneira como os pesos são ajustados

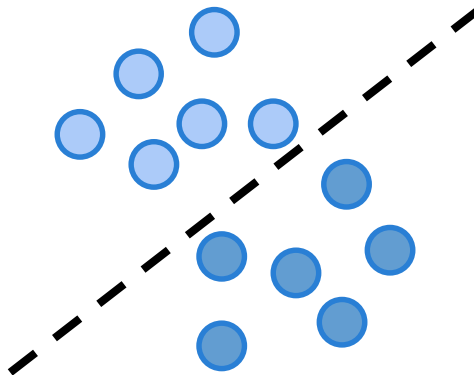


# Rede Perceptron Simples

- É considerada o primeiro algoritmo de NRA
  - Desenvolvida por Rosenblatt em 1958
  - Utiliza modelo de neurônio de McCulloch-Pitts como unidade de processamento com saída em  $\{-1, +1\}$ 
    - Perceptron Simples = Neurônio + Regra de Aprendizagem

# Rede Perceptron Simples

- É a rede mais simples para classificação de padrões linearmente separáveis
  - Para classificação binária (2 classes), resume-se a um neurônio com pesos ajustáveis
  - A regra de aprendizagem é o mecanismo que torna a rede Perceptron Simples um dispositivo inteligente



# Rede Perceptron Simples

- Regra de Aprendizado
  - Fornecem a base para o entendimento dos métodos de treinamento para redes formadas por várias unidades
  - Consiste na modificação dos pesos e do limiar do neurônio
    - Até que ele resolva o problema de interesse
    - Ou até que o período de aprendizagem tenha terminado

# Rede Perceptron Simples

- Regra de Aprendizado
  - Pesos são inicializados aleatoriamente
  - Pesos são então ajustados sempre que a rede classifica equivocadamente um exemplo de treinamento
  - Esse processo se repete até que um determinado critério de parada seja alcançado

# Rede Perceptron Simples

- Regra de Aprendizado
  - É um classificador linear ótimo
    - Sua regra de aprendizagem conduz à minimização de uma função-custo
    - Tenta encontrar a melhor fronteira linear que separa os dados
- Possível função-custo
  - Quantificar a probabilidade média de erros de classificação
  - Buscamos minimizar o erro de classificação dos dados de entrada

# Rede Perceptron Simples

- Treinamento
  - Treinamento supervisionado
    - Padrões desejados de saída  $\mathbf{d}$
  - Dado um padrão de entrada  $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_p]$  tem-se a correção de erro para cada peso ( $\mathbf{w}_i, i=1, \dots, p$ )
    - $\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \Delta \mathbf{w}_i(t)$
  - Onde
    - $\mathbf{w}_i(t)$ : peso atual
    - $\Delta \mathbf{w}_i(t)$ : incremento no peso
    - $\mathbf{w}_i(t+1)$ : peso modificado

# Rede Perceptron Simples

- Treinamento: incremento do peso  $\Delta w_i(t)$ 
  - Gradiente descendente
    - A direção do passo futuro dependerá da direção do passo anterior pois sempre são ortogonais
  - Classificação correta ( $d = y$ )
    - $\Delta w_i(t) = 0$
  - Classificação incorreta ( $d \neq y$ )
    - $\Delta w_i(t) = \eta x_i (d - y)$
  - Fator  $\eta$ 
    - Tornar o processo de ajuste mais estável (também chamado de passo de aprendizagem)
    - $0 < \eta \ll 1$

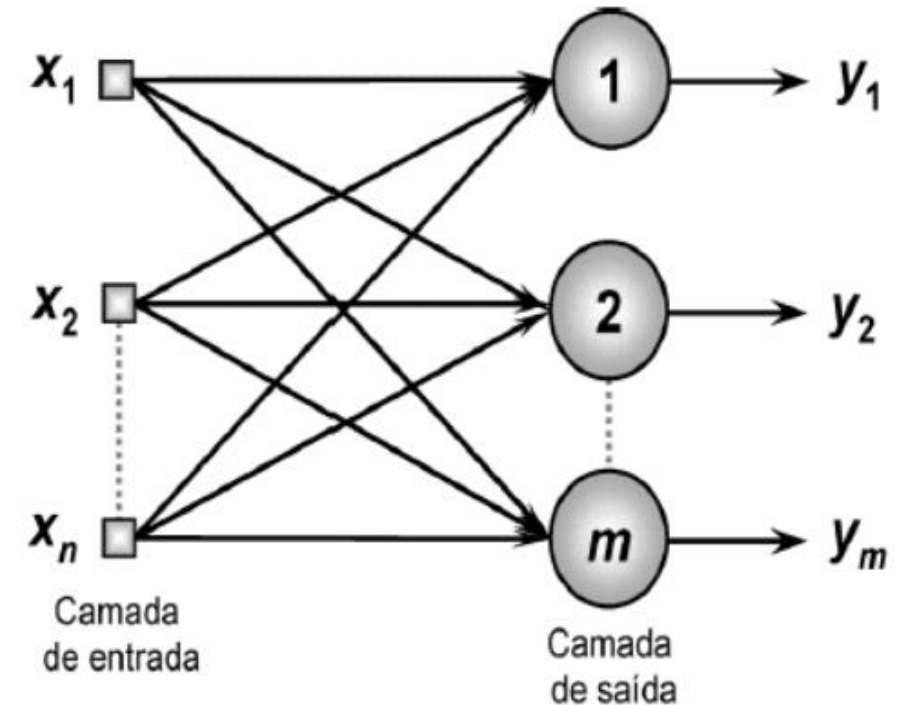
# Rede Perceptron Simples

- Algoritmo de Treinamento
  - Iniciar todos os pesos  $w_i$
  - Repita
    - Para cada par de treinamento  $(\mathbf{x}, \mathbf{d})$ 
      - Calcular a saída  $\mathbf{y}$
      - Se  $(\mathbf{d} \neq \mathbf{y})$  Então
        - Atualizar os pesos dos neurônios
  - Até o erro ser aceitável



# Rede Perceptron Simples

- O que fazer se tivermos mais de 2 classes?
  - Um único neurônio nos permite categorizar apenas duas classes de dados
  - Para problemas com múltiplas classes, deve-se utilizar vários neurônios em paralelo



# Rede Perceptron Simples

- Classificação em múltiplas classes
  - O funcionamento de cada neurônio é o mesmo individualmente
  - Todos possuem
    - Seu próprio vetor de pesos
    - Ajuste de pesos
    - Sinal de saída
  - Numa rede com  $Q$  neurônios teremos  $Q$  regras de aprendizagem

# Rede Perceptron Simples

- Classificação em múltiplas classes
  - Como especificar o número de neurônios  $Q$  ?
  - **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  $Q > 2,45 = 3$

# Rede Perceptron Simples

- Classificação em múltiplas classes
  - **Método 1:** Codificação binária simples
    - Exemplo: Se  $C = 6$  classes, então  $Q > 2,45 = 3$
    - Classe 1:  $d = [0 \ 0 \ 1]^T$
    - Classe 2:  $d = [0 \ 1 \ 0]^T$
    - Classe 3:  $d = [0 \ 1 \ 1]^T$
    - Classe 4:  $d = [1 \ 0 \ 0]^T$
    - Classe 5:  $d = [1 \ 0 \ 1]^T$
    - Classe 6:  $d = [1 \ 1 \ 0]^T$

# Rede Perceptron Simples

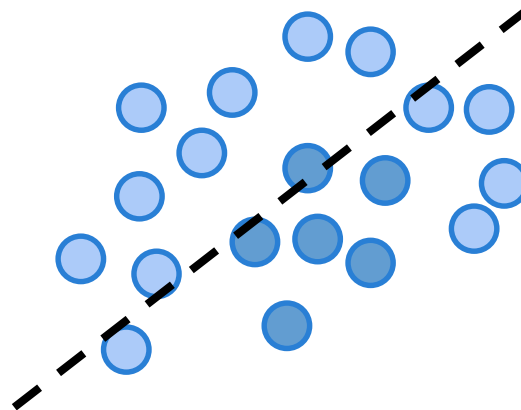
- Classificação em múltiplas classes
  - Como especificar o número de neurônios  $Q$  ?
  - **Método 2:** Codificação 1-out-of- $Q$ 
    - Se tenho  $C$  classes, então  $Q = C$
    - Exemplo: Se  $C = 4$  classes, então  $Q = 4$
  - Apenas uma das componentes do vetor tem valor igual a 1
    - Os vetores são ortogonais, isto é, produto escalar entre eles é nulo

# Rede Perceptron Simples

- Classificação em múltiplas classes
  - **Método 1:** Codificação 1-out-of-Q
    - Exemplo: Se  $C = 4$  classes, então  $Q = 4$
    - Classe 1:  $\mathbf{d} = [0 \ 0 \ 0 \ 1]^T$
    - Classe 2:  $\mathbf{d} = [0 \ 0 \ 1 \ 0]^T$
    - Classe 3:  $\mathbf{d} = [0 \ 1 \ 0 \ 0]^T$
    - Classe 4:  $\mathbf{d} = [1 \ 0 \ 0 \ 0]^T$

# Rede Perceptron Simples

- O que fazer se as classes não puderem ser separadas por uma reta?
  - Perceptron Simples resolve apenas problemas linearmente separáveis
    - Grande número de aplicações importantes não são linearmente separáveis



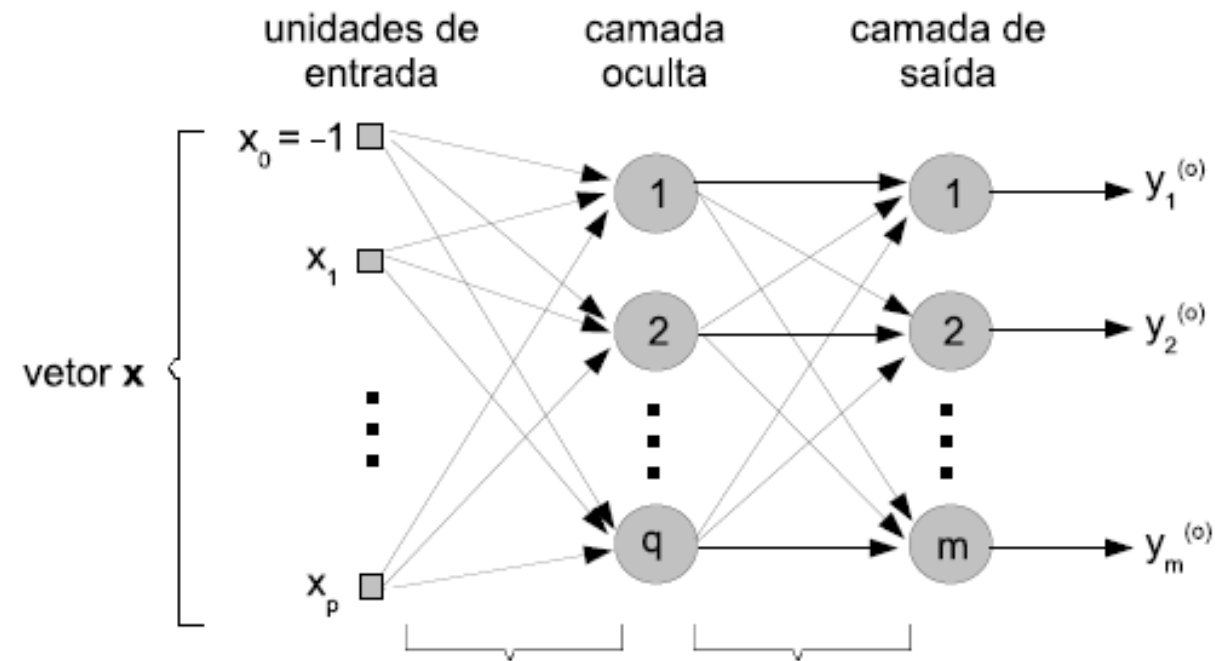
# Perceptron de Múltiplas Camadas

- Podemos modificar a rede Perceptron
  - Solução: utilizar mais de uma camada de neurônios dentro da rede
- Rede Perceptron de Múltiplas Camadas
  - Multilayer Perceptron – MLP



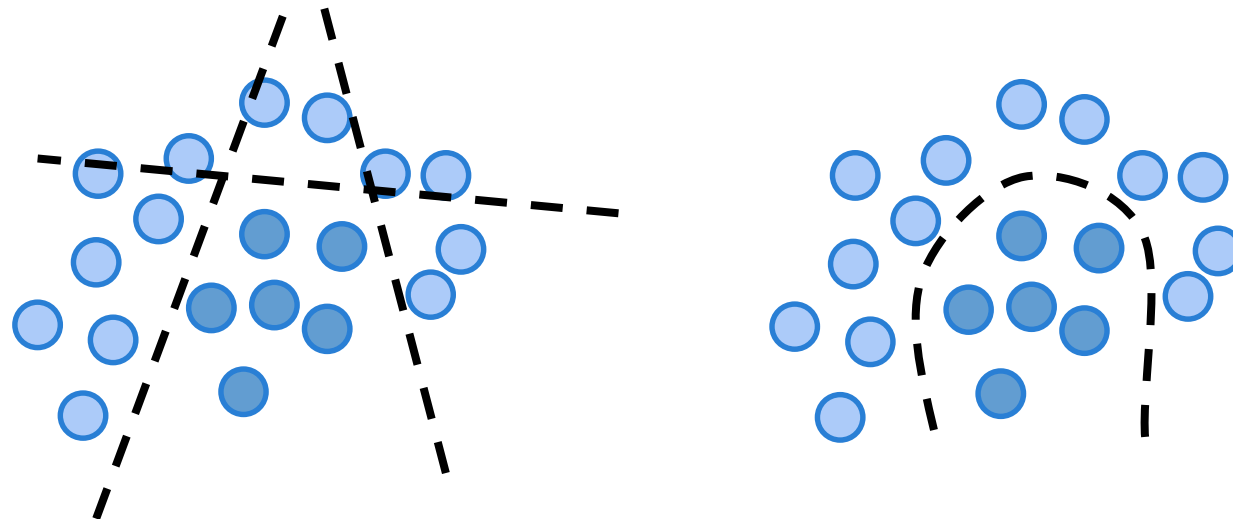
# Perceptron de Múltiplas Camadas

- Estrutura básica de uma MLP
  - Conjunto de unidades de entrada
    - Recebem os sinais (ou dados)
  - Uma ou mais camadas ocultas
    - Não possuem acesso direto a saída da rede
    - Neurônios não se conectam dentro de uma mesma camada
  - Uma camada de saída:
    - combina as saídas produzindo a classificação final



# Perceptron de Múltiplas Camadas

- Qual a função das camadas ocultas?
  - Elas realizam uma transformação não linear nos dados
    - Cada camada é uma rede Perceptron para cada grupo de entradas linearmente separáveis
  - Facilitam a tarefa de classificação



# Perceptron de Múltiplas Camadas

- Notação de uma MLP
  - Uma rede MLP com 1 camada oculta é representada por
    - $MLP(p, q_1, m)$
  - Onde
    - $p$  é o número de variáveis de entrada
    - $q_1$  é o número de neurônios ocultos
    - $m$  é o número de neurônios de saída
  - Com 2 camadas
    - $MLP(p, q_1, q_2, m)$ , onde  $q_2$  é o número de neurônios ocultos na segunda camada

# Perceptron de Múltiplas Camadas

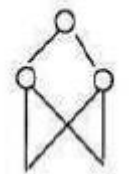
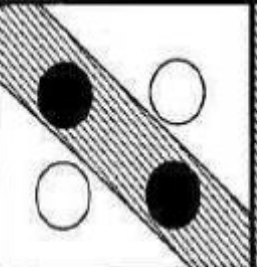


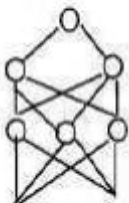
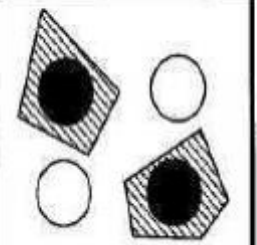
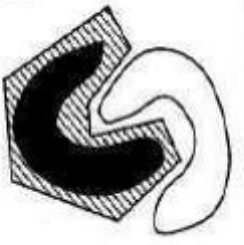
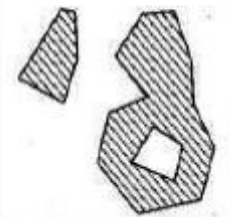
- Notação de uma MLP
  - Desse modo, uma rede MLP com
    - 4 variáveis de entrada, 10 neurônios ocultos e 2 neurônios de saída é representada como MLP(4,10,2)
    - 15 variáveis de entrada, 20 neurônios na 1ª camada oculta, 10 neurônios na 2ª camada oculta e 4 neurônios de saída é representada como MLP(15,20,10,4)

# Perceptron de Múltiplas Camadas

- Quantas camadas ocultas usar?
- E quantos neurônios por camada oculta?
  - Não há regras determinadas para isso
    - Podemos ter qualquer número de camadas e neurônios
  - Em geral, depende da natureza do problema
    - Muitas camadas e/ou neurônios podem comprometer o desempenho da rede

# Perceptron de Múltiplas Camadas

- De modo geral, uma ou duas camadas ocultas são suficientes
  - Uma camada oculta: regiões de decisão convexas
  - Duas camadas oculta: regiões de decisão não convexas

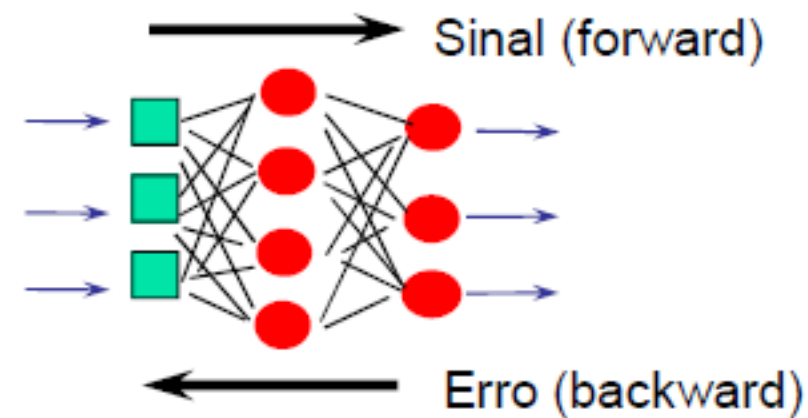
Rede MLP	Descrição das regiões de decisão	Problema do OU-exclusivo	Classes com regiões não-convexas	Formas das regiões de decisão
 1 camada oculta	Arbitrária (complexidade limitada pelo número de neurônios ocultos)			
 2 camadas ocultas	Arbitrária (complexidade limitada pelo número de neurônios ocultos)			

# Perceptron de Múltiplas Camadas

- Como treinar uma rede MLP?
  - Neurônios em camadas ocultas não tem acesso a saída da rede
    - Não é possível calcular o erro associado a esse neurônio
  - O que fazer então?
    - Uma solução foi “inventar” uma espécie de erro para os neurônios ocultos
    - Propagar os erros dos neurônios de saída em direção a todos os neurônios das camadas ocultas (caminho inverso ao do fluxo da informação)

# Backpropagation

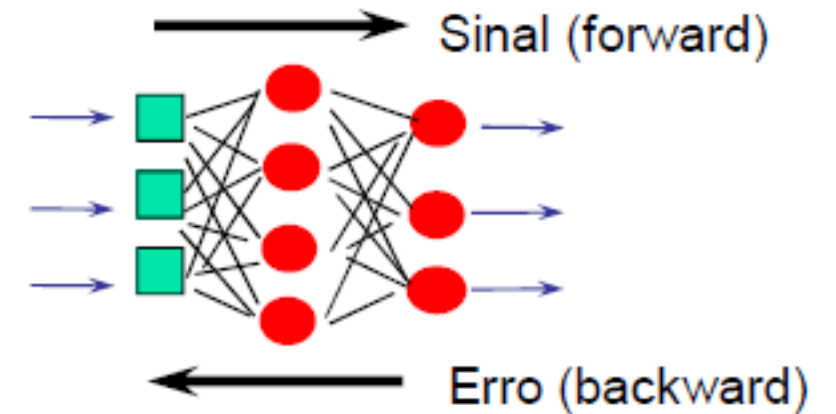
- Algoritmo de *backpropagation*
  - Uma das mais populares técnicas de aprendizado para redes MLP
  - Envolve dois sentidos de propagação de sinais na rede
    - Sentido direto (forward)
    - Sentido inverso (backward)





# *Backpropagation*

- Algoritmo de *backpropagation*
  - Sentido direto (forward):
    - Cálculo da saída e do erro
  - Sentido inverso (backward)
    - Propagação do erro
      - Envolve o cálculo de derivadas



# *Backpropagation*

- Propagação do erro
  - A idéia básica é propagar o sinal de erro calculado na etapa de treinamento de volta para todos os neurônios
    - Coeficientes dos pesos utilizados para propagar os erros para trás são iguais aos utilizados durante o cálculo de valor de saída
    - Apenas a direção do fluxo de dados é alterado
    - Esta técnica é aplicada em todas as camadas de rede

# *Backpropagation*

- Propagação do erro
  - Sinal de erro é calculado para cada neurônio
    - Seus coeficientes de peso podem ser modificados
  - Esse cálculo envolve a derivada da função de ativação do neurônio

# *Backpropagation*

- Problema
  - Derivadas demandam funções diferenciáveis
    - Funções de ativação dos neurônios intermediários são descontínuas
- Solução
  - Funções de ativação contínuas
    - Utilizar aproximações das funções de ativação

# *Backpropagation*

- Função de ativação
  - A função de ativação do neurônio artificial é do tipo Degrau
    - Não-linearidade dura ou hard
    - A saída é uma variável do tipo ON-OFF (binária  $[0,1]$  ou bipolar  $[-1,+1]$ )
  - Substituímos ela por uma função de ativação do tipo Sigmoidal
    - Não-linearidade suave ou soft
    - A saída passa a ser uma variável do tipo Real ou Analógica (qualquer valor entre  $[0,1]$  ou  $[-1,+1]$ )

# *Backpropagation*

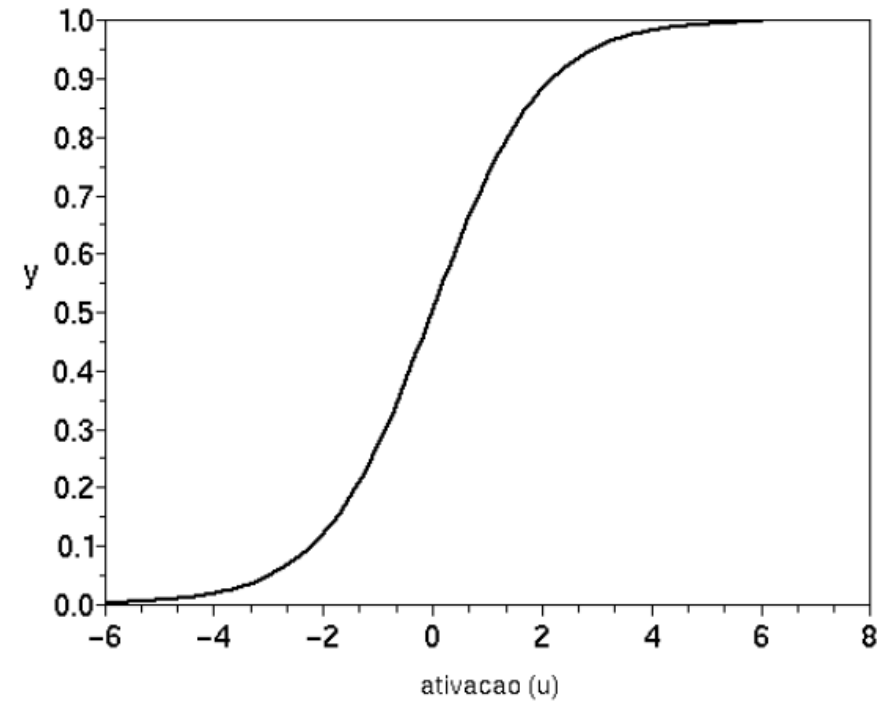
- Função de ativação do tipo Sigmoidal
  - Vantagens
    - Derivadas fáceis de calcular
    - Não-linearidade fraca (trecho central é quase linear)
    - Interpretação da saída como taxa média de disparo (mean firing rate), em vez de simplesmente indicar se o neurônio está ou não ativado (ON-OFF)
  - Desvantagens
    - Elevado custo computacional para implementação em sistemas embarcados devido à presença da função exponencial

# Backpropagation

- Função de ativação
  - Sigmóide Logística

$$y_i(t) = \frac{1}{1 + \exp(-u_i(t))}$$

$$y_i(t) \in (0,1)$$

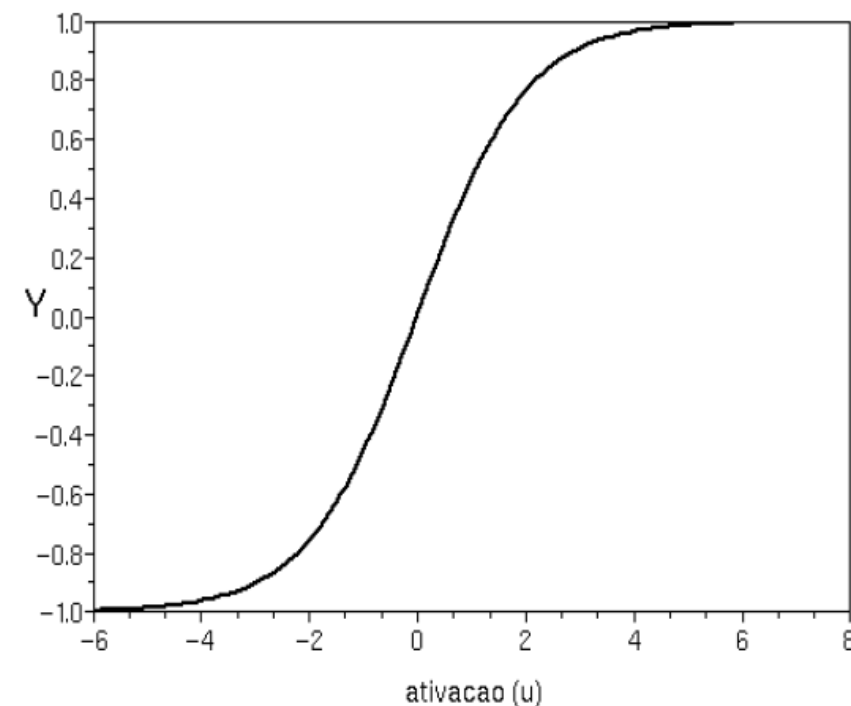


# *Backpropagation*

- Função de ativação
  - Tangente Hiperbólica

$$y_i(t) = \frac{1 - \exp(-u_i(t))}{1 + \exp(-u_i(t))}$$

$$y_i(t) \in (-1, 1)$$





# *Backpropagation*

- Existem muitas variações do *Backpropagation*
  - Momentum
  - Quickprop
  - Newton
  - Gradiente Conjugado
  - Levenberg-Marquardt
  - Super Self-Adjusting Backpropagation

# Problemas no aprendizado

- Ocorrência de mínimos locais
  - A solução estável que não é a melhor
  - Incidência desse problema pode ser reduzida
    - Uso de *backpropagation seqüencial* (estocástico)
    - Múltiplas inicializações dos pesos
- Lentidão da rede em superfícies complexas
  - Podemos amenizar o problema
    - Uso de variantes do *backpropagation*

# Problemas no aprendizado

- *Overfitting* (sobreajustamento)
  - A partir de um certo ponto do treinamento, o desempenho da rede piora ao invés de melhorar
  - A rede se especializa nos padrões de treinamento, incluindo suas peculiaridades
    - Piora a sua capacidade de generalização
    - Incapacita a rede de reconhecer dados diferentes dos usados no seu treinamento

# Problemas no aprendizado

- *Overfitting* (sobreajustamento)
  - O que fazer nesse caso?
    - Podemos encerrar treinamento mais cedo (*early stop*)
    - Fazer a poda de conexões e neurônios irrelevantes (*pruning*)
    - Penalizar os valores dos pesos (*weight decay*)

# Problemas no aprendizado

- *Underfitting* (subajustamento)
  - Arquitetura da rede tem poucos parâmetros
    - O modelo é muito simples
  - Falta de representatividade das classes
  - É possível que a rede sequer aprenda o padrão
    - Baixa capacidade de generalização
- Pode-se resolver esse tipo de problema com um conjunto de treinamento de bom tamanho
  - Técnicas de amostragem ajudam

# Atualização dos pesos da rede

- Existem diversas abordagens para a atualização dos pesos da rede
  - Por ciclo (batelada ou *batch*)
  - Por padrão (sequencial ou *on-line*)
- A escolha da melhor abordagem depende da aplicação

# Atualização dos pesos da rede

- Atualização por ciclo (batelada ou *batch*)
  - Atualiza os pesos depois que todos os padrões de treinamento forem apresentados
  - Vantagens
    - Estimativa mais precisa do vetor gradiente
    - Mais estável
  - Desvantagem
    - Mais lento

# Atualização dos pesos da rede

- Por padrão (sequencial ou *on-line*)
  - Atualiza os pesos após apresentação de cada padrão em ordem aleatória
  - Vantagens
    - Requer menos memória
    - Mais rápido
    - Menos susceptível a mínimos locais
  - Desvantagens
    - Pode se tornar instável
    - Requer controle da taxa de aprendizado



# Redes RBF

- Redes RBF (*Radial Basis Functions*)
  - Redes com função de base radial
  - Classe de redes com arquitetura feedforward
    - Os valores das entradas se propagam na rede em um único sentido
  - Possui camada oculta como as redes MLPs
    - Apenas 1 camada oculta
    - Podem resolver problemas não linearmente separáveis

# Redes RBF

- Redes RBF (*Radial Basis Functions*)
  - Utilizam um modelo de neurônio diferente das MLPs
    - Neurônios da camada oculta com resposta radial a excitações
    - Esse tipo de neurônio modela o conceito biológico de campo receptivo (*local receptive fields*)
  - Os neurônios de saída são neurônios comuns

# Redes RBF

- Redes RBF (*Radial Basis Functions*)
  - Neurônios com resposta radial
    - Respondem seletivamente a um intervalo finito do espaço de sinais de entrada
  - Aprendizagem
    - Busca uma superfície em um espaço de dimensão qualquer que produza o melhor ajuste os dados de treinamento
    - Treinamento, em geral, muito mais rápido que as MLP

# Redes RBF

- Resposta Radial
  - Presentes em alguns tipos de células nervosas
    - Células auditivas possuem maior sensibilidade a frequências próximas a um determinado tom
    - Células da retina maior sensibilidade a excitações luminosas próximas ao centro do seu campo receptivo
  - Modelo matemático
    - Função de Base Radial

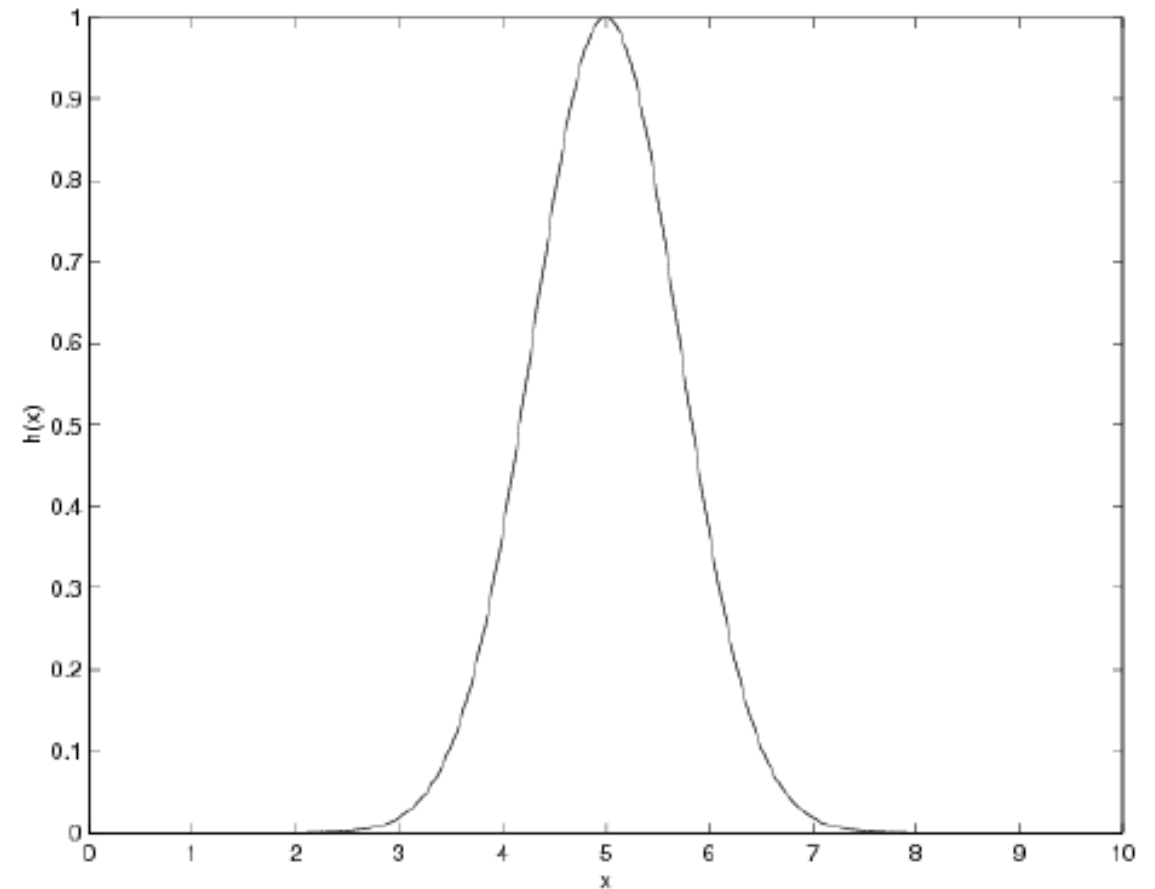
# Modelo do neurônio RBF

- Existem diferentes modelos matemáticos possíveis para uma função de base radial
  - Gaussiana
    - $h(x) = \exp(-\frac{(x-\mu)^2}{\sigma^2})$
  - Multi-Quadrática Inversa
    - $h(x) = \frac{1}{\sqrt{(x-\mu)^2 + \sigma^2}}$
  - Chapéu Mexicano
    - $h(x) = \begin{cases} \frac{\sin((x-\mu)/\sigma)}{((x-\mu)/\sigma)}, & \text{se } x \neq \mu \\ 1, & \text{caso contrário} \end{cases}$

# Modelo do neurônio RBF

- Gaussiana

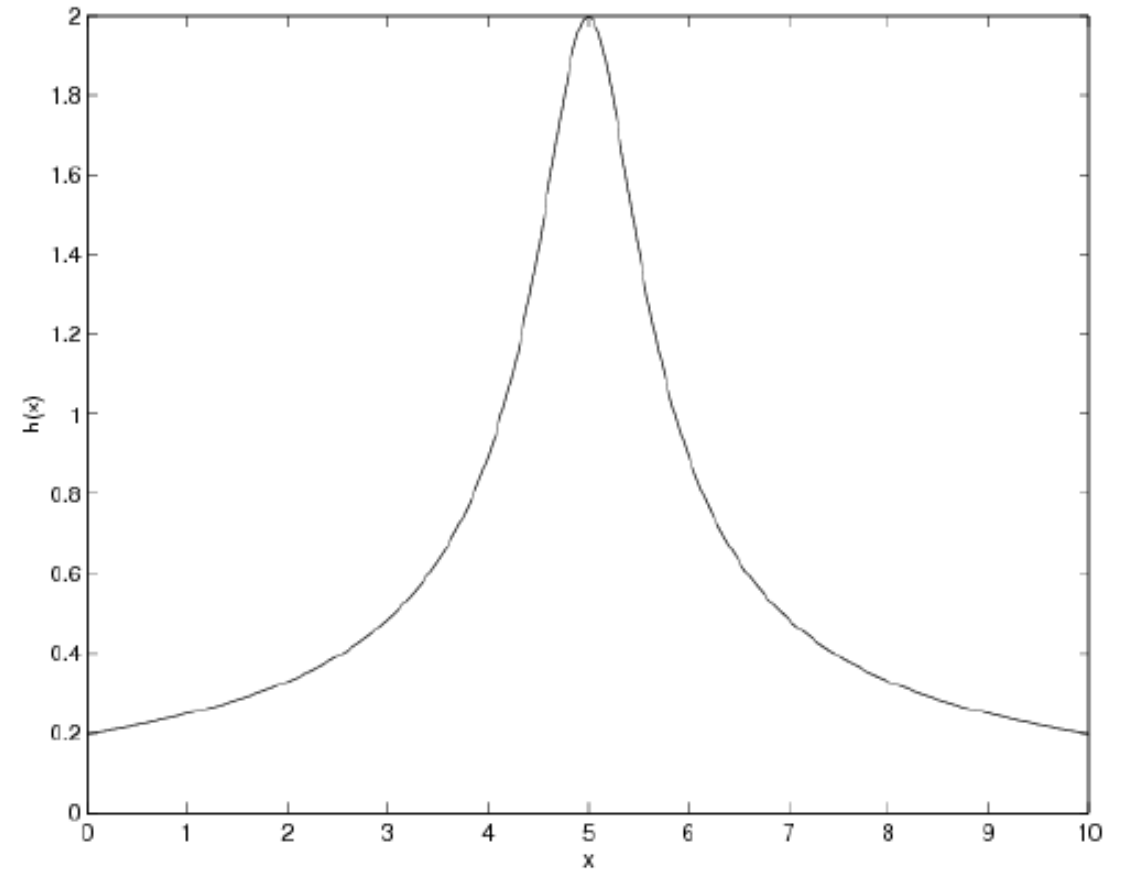
- $h(x) = \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right)$



# Modelo do neurônio RBF

- Multi-Quadrática Inversa

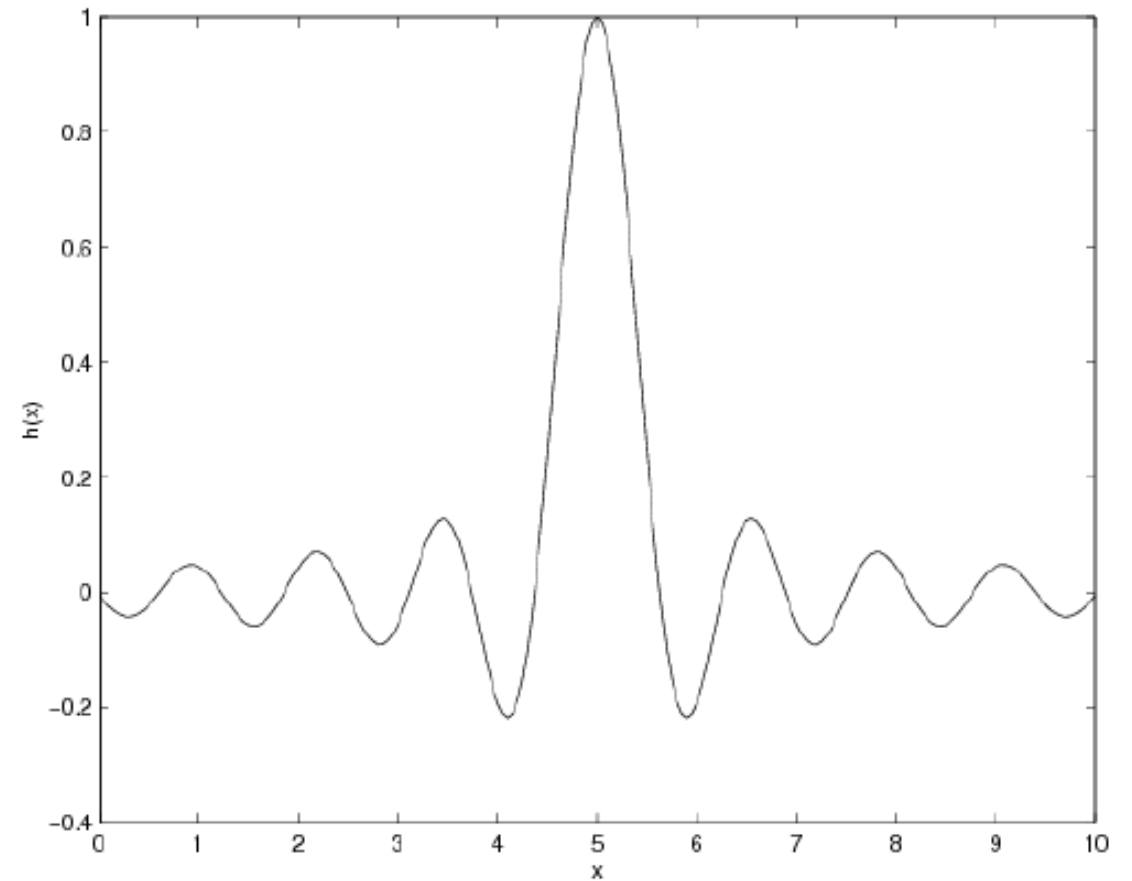
- $$h(x) = \frac{1}{\sqrt{(x-\mu)^2 + \sigma^2}}$$



# Modelo do neurônio RBF

- Chapéu Mexicano

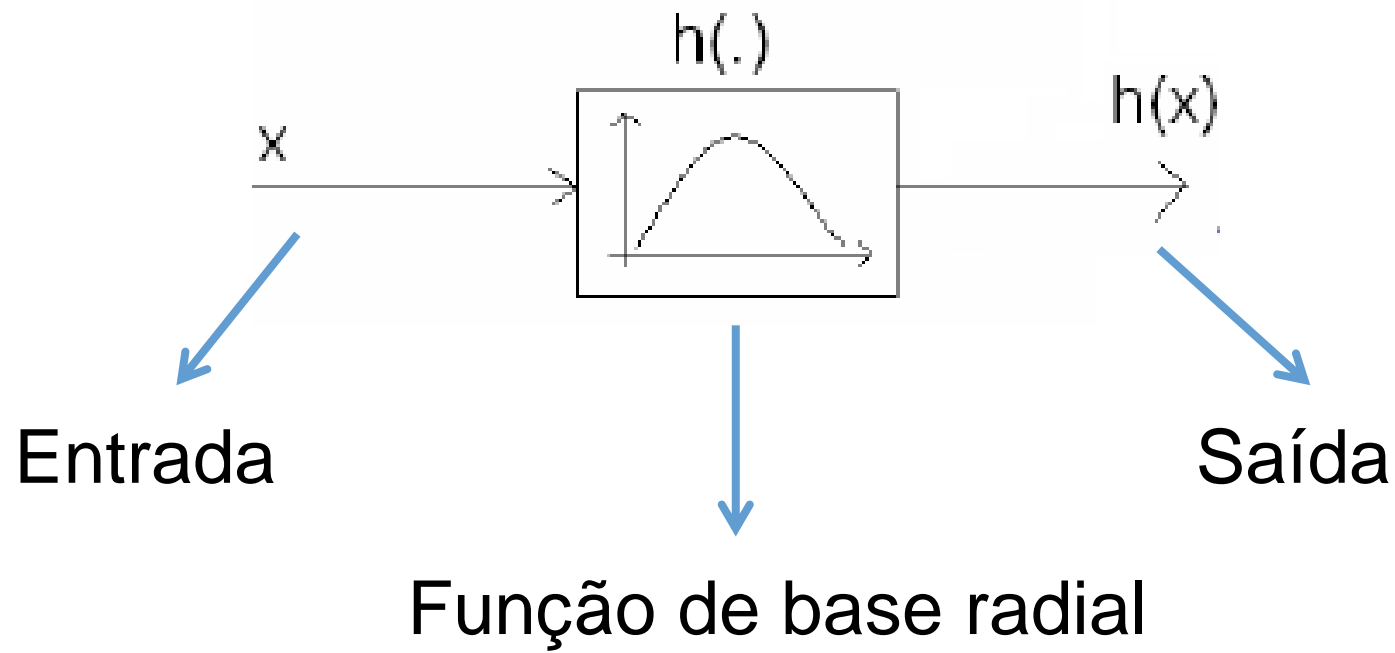
- $$h(x) = \begin{cases} \frac{\sin((x-\mu)/\sigma)}{((x-\mu)/\sigma)}, & \text{se } x \neq \mu \\ 1, & \text{caso contrário} \end{cases}$$





# Modelo do neurônio RBF

- Modelo básico do neurônio

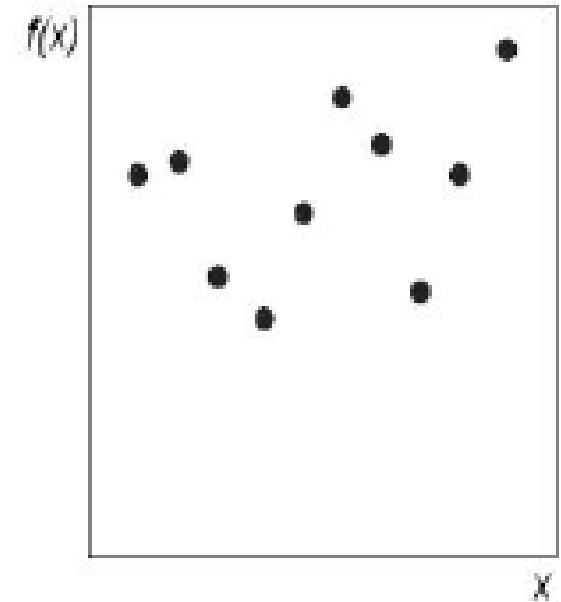


# Histórico

- Podemos construir uma função complexa a partir de funções simples
  - Série de Fourier
  - Transformada de Fourier
  - Transformada Wavelet
  - Redes RBF
- Redes RBF utilizam funções de base radial para aproximar outras funções

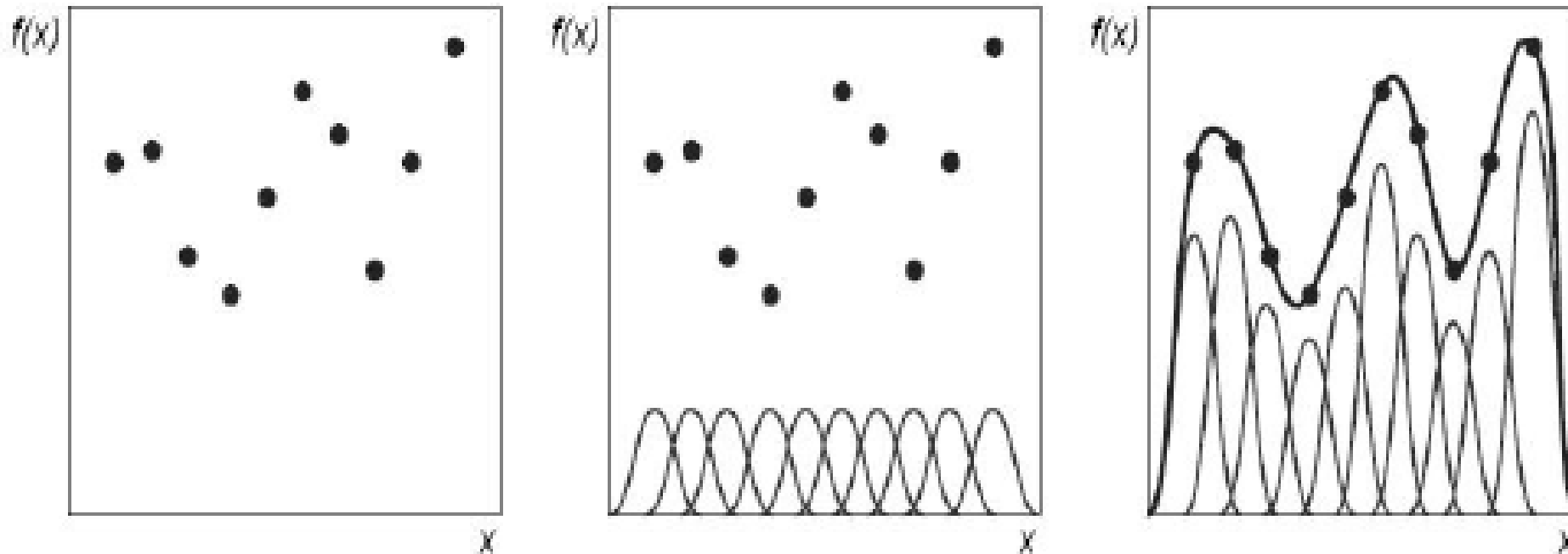
# Histórico

- Sua origem vem de técnicas para realizar interpolação exata de funções
  - Ex.: utilize um conjunto de  **$N$**  funções de base, não-lineares, para calcular a função dada função



# Histórico

- A rede RBF pode aproximar qualquer função contínua através da combinação linear de funções gaussianas com centros em diferentes posições do espaço de entrada.



# Histórico

- Primeiros trabalhos com funções de base radial
  - Interpolação
  - Estimação de densidade
  - Aproximação de funções multivariadas suaves
- Atualmente
  - Os modelos são de natureza adaptativa
  - Utilização de um número relativamente menor de unidades de processamento localmente sintonizadas

# Desempenho da rede RBF

- De modo geral, redes RBF precisam de ao menos **10** vezes mais dados de treinamento para atingir a mesma precisão das redes MLP-BP
- Em tarefas difíceis de classificação, redes RBF podem ser melhores que MLP
  - Necessidade de número suficiente de
    - Padrões de treinamento
    - Neurônios ocultos

# Desempenho da rede RBF

- Apesar da necessidade de maior conjunto de treinamento, o tempo de treinamento é muito menor
  - Apenas uma pequena fração de neurônios ocultos responde a um dado padrão de entrada
    - São unidades localmente sintonizáveis
    - Sensíveis a padrões próximos de seus campos receptivos
  - Numa MLP
    - Todos os neurônios são avaliadas e têm seus pesos ajustados

# Comparação entre as redes

- Redes RBF *versus* Redes MLP
  - RBF só tem uma camada oculta
    - MLP-BP pode ter mais
  - RBF usualmente tem mais neurônios na oculta que a MLP
  - MLP usa funções sigmoidais de ativação
    - RBF usa função de base radial e linear nas camadas oculta e de saída, respectivamente
  - RBF é usualmente menos sensível a inserção de dados novos
  - RBF pode necessitar de maior número de parâmetros ajustáveis

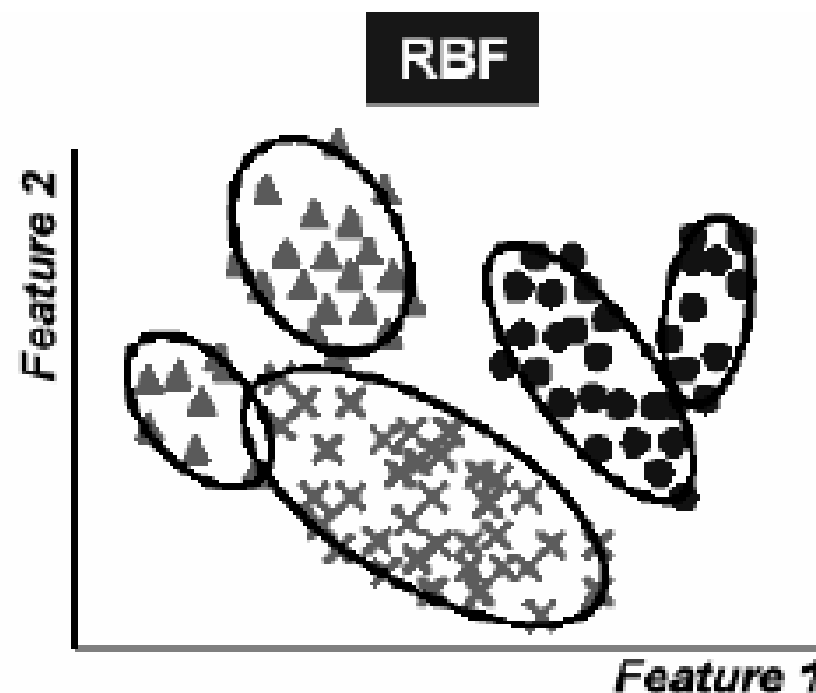
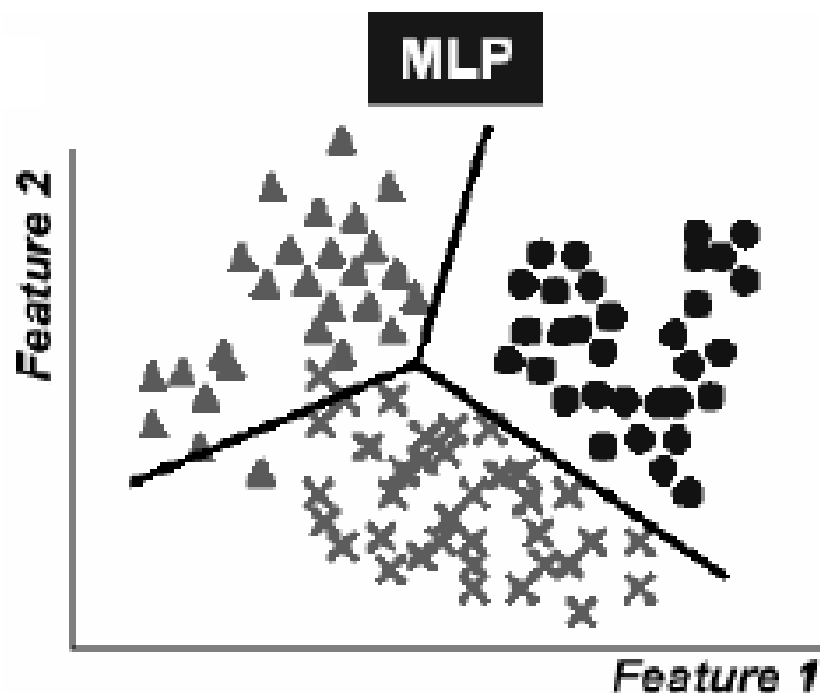


# Comparação entre as redes

- Redes RBF *versus* Redes MLP
  - MLP gera regiões globais de decisão
    - Maior capacidade de generalização (extrapolação) que a RBF. Logo, lida melhor com outliers que a RBF (ajuste local)
  - Quando usar qual?
    - Rede MLP: padrões de entrada são custosos (ou difíceis de se gerar) e/ou quando a velocidade de recuperação é crítica
    - Rede RBF: os dados são baratos e abundantes, necessidade de treinamento online

# Comparação entre as redes

- Redes RBF *versus* Redes MLP
  - RBF separa classes por hiperelipsoides e a MLP-BP por hiperplanos



# Redes Hopfield

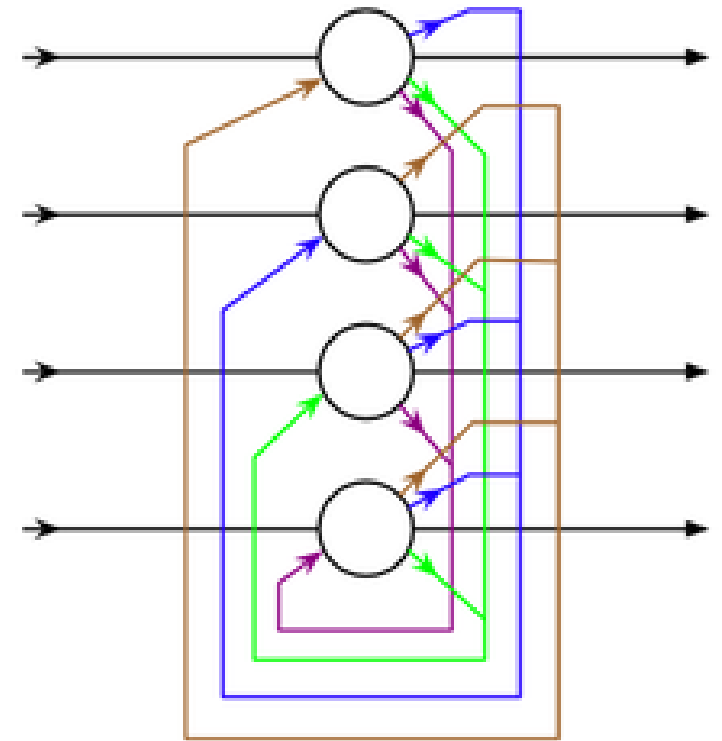
- Modelo de redes neurais auto-associativas
  - Desenvolvidas por J. Hopfield em 1982
- Similar a um modelo de memória auto-associativa
  - Capaz de armazenar e depois recuperar um certo conjunto de padrões

# Redes Hopfield | Motivação

- Sistemas físicos com um grande  $n^o$  de elementos
  - Interações entre estes geram fenômenos coletivos estáveis
- Redes que possuem neurônios que interagem entre si podem levar a fenômenos coletivos equivalentes?
  - Sistemas de neurônios conectados possuem estados estáveis que são atingidos quando a rede é estimulada por estados similares

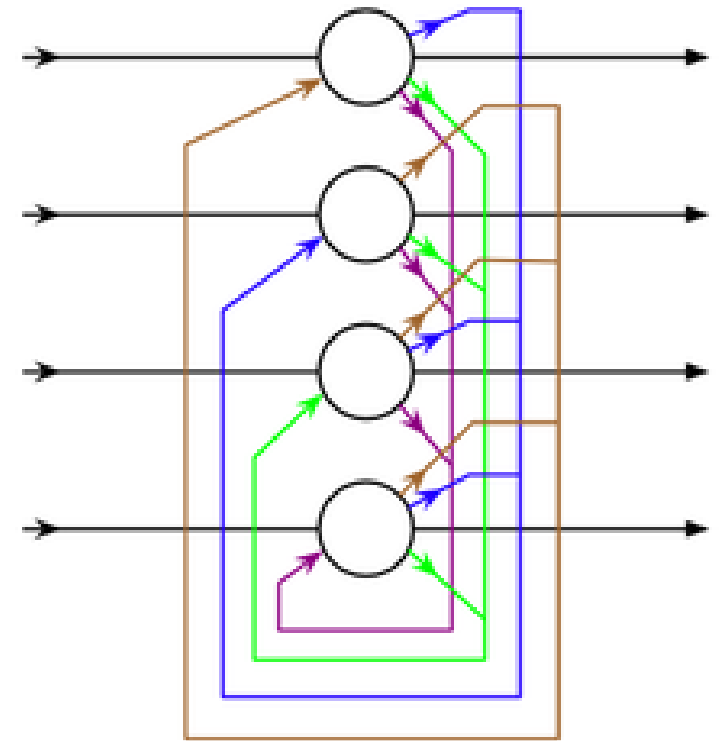
# Redes Hopfield | Características

- Possui uma única camada de neurônios totalmente conectada
- Utiliza neurônios do tipo MCP (McCulloch-Pitts)
  - Unidade de processamento com saída em  $\{-1, +1\}$
- Estrutura recorrente
  - Com *feedback*



# Redes Hopfield | Características

- Unidades são ao mesmo tempo de entrada e de saída
- Conjunto de saídas define o “estado” da rede
- Funcionamento assíncrono



# Redes Hopfield

## □ Funcionamento assíncrono

- Em um determinado instante de tempo apenas uma unidade da rede é escolhida para mudar de estado
- Esse processo se repete até que a rede encontre um ponto de equilíbrio estável
  - A saída de cada unidade da rede se mantém constante

# Redes Hopfield | Funcionamento básico

- Etapa de treinamento
  - A rede memoriza os padrões
- Etapa de uso
  - A rede irá passar por uma sequência de ativações intermediárias até se estabilizar em um padrão previamente treinado

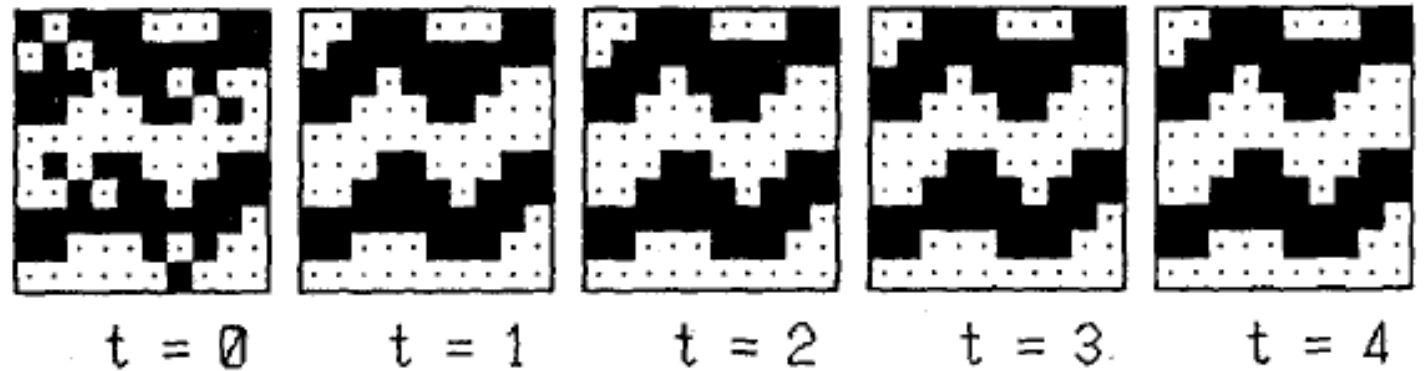


# Redes Hopfield | Funcionamento básico

- Etapa de treinamento



- Etapa de uso



# Redes Hopfield | Aplicações

- Regeneração de padrões



# Redes Hopfield | Aplicações

- Completar um padrão conhecido



# Agradecimentos

- Agradeço aos professores
  - Guilherme de Alencar Barreto - Universidade Federal do Ceará (UFC)
  - Prof. Ricardo J. G. B. Campello – ICMC/USP
  - Aluizio Fausto Ribeiro Araújo - Universidade Federal de Pernambuco (UFPE)
  - Germano C. Vasconcelos - Universidade Federal de Pernambuco (UFPE)
  - Paulo J.L. Adeodato - Universidade Federal de Pernambuco (UFPE)
- pelo material disponibilizado