# Classificação de Dados

## Parte 2

Prof. Guilherme Palermo Coelho

CET0621 - Aprendizado de Máquina na Análise de Dados

# Roteiro

- Perceptrons Multicamadas:
  - Introdução e inspiração biológica;
  - Neurocomputação e o neurônio artificial;
  - Redes Neurais de Múltiplas Camadas;
  - Aprendizado em MLPs;
  - Outros aspectos a serem considerados.

**Nota**: Este material está parcialmente baseado nas notas de aula dos cursos "Redes Neurais", desenvolvidas pelo Prof. Fernando J. Von Zuben (FEEC/Unicamp), e "Mineração de Dados", desenvolvidas pelo Prof. Fabrício Olivetti de França (CMCC/UFABC).

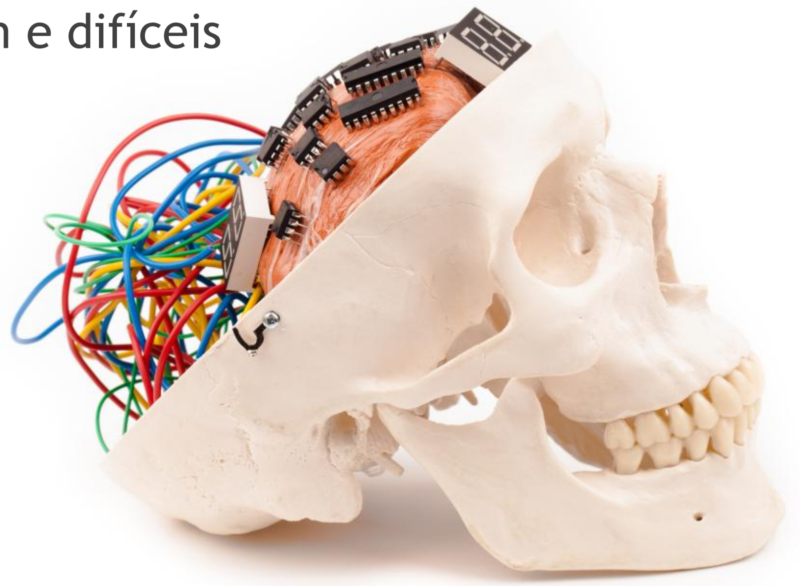
# Introdução e Inspiração Biológica

# Introdução

- O cérebro apresenta características muito interessantes sob o ponto de vista de resolução de problemas:
  - Adaptabilidade por intermédio de aprendizado;
  - Comportamento sensível ao contexto;
  - Tolerância a erros;
  - Capacidade de operar com conhecimento parcial;
  - Grande capacidade de memória;
  - Capacidade de processamento paralelo e em tempo real;
- É desejável que tais características estejam presentes em sistemas computacionais voltados para resolução de problemas reais.

# Introdução

- No princípio da computação, os computadores eletrônicos eram chamados de “cérebros eletrônicos”:
  - Acreditava-se que eles representavam um caminho direto para a reprodução da inteligência;
  - Com o passar dos anos eles não atenderam às expectativas de reprodução de comportamentos inteligentes.
  - Muitos são os exemplos de tarefas que são fáceis para o homem e difíceis para a máquina, e vice-versa.



# Introdução

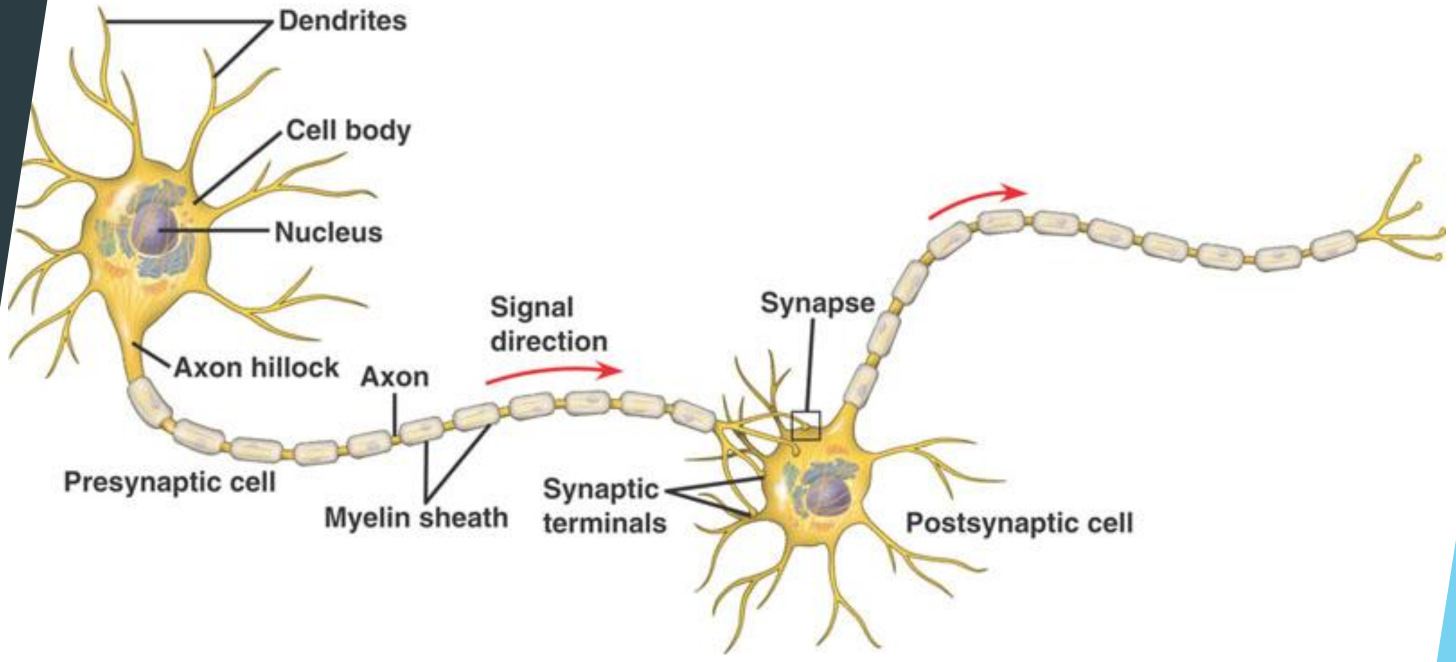
- Neste contexto, nos primórdios da IA havia uma linha de pensamento que defendia que, para alcançarmos a **simulação da inteligência humana**, seria necessário imitar o funcionamento do próprio cérebro:
  - Avançamos muito no conhecimento da arquitetura fisiológica do cérebro;
  - Mas ainda não conhecemos exatamente como o cérebro realiza computação de alto nível.

# Introdução

- Percebeu-se então que o caminho contrário talvez pudesse levar a melhores resultados:
  - Tentar extrair conhecimento sobre o funcionamento do cérebro a partir de simulações, em computadores digitais, de modelos matemáticos de redes neurais.
- Este caminho alternativo levou ao desenvolvimento das **redes neurais artificiais (RNA).**

# Inspiração Biológica

- As RNAs têm sua inspiração nos neurônios biológicos e suas interconexões;
- **Neurônios**: unidades básicas de processamento do sistema nervoso de vertebrados;
  - São responsáveis por ***processar sinais***, vindos tanto do meio externo quanto de outros neurônios;
  - Estão conectados uns aos outros através das ***conexões sinápticas***;
  - São capazes de gerar **sinais elétricos**, usados para transmitir informações a outras células.



# Inspiração Biológica

- O processo de **transmissão de sinais entre neurônios** é fundamental para a capacidade de processamento de informação do cérebro;
  - A efetividade da transmissão de sinais pode ser **modulada**, permitindo que o cérebro se adapte a diferentes situações.
- A **plasticidade sináptica**, ou seja, a capacidade das sinapses sofrerem modificações, é essencial para o aprendizado da maioria das RNAs;
  - A **memória** também é resultado de um processo adaptativo das sinapses;
  - Estas alterações resultam em caminhos novos ou facilitados de desenvolvimento e transmissão de sinais através dos circuitos neurais;
  - Um dos resultados de um **processo de aprendizagem** é a criação de um padrão de conexões sinápticas mais permanente.

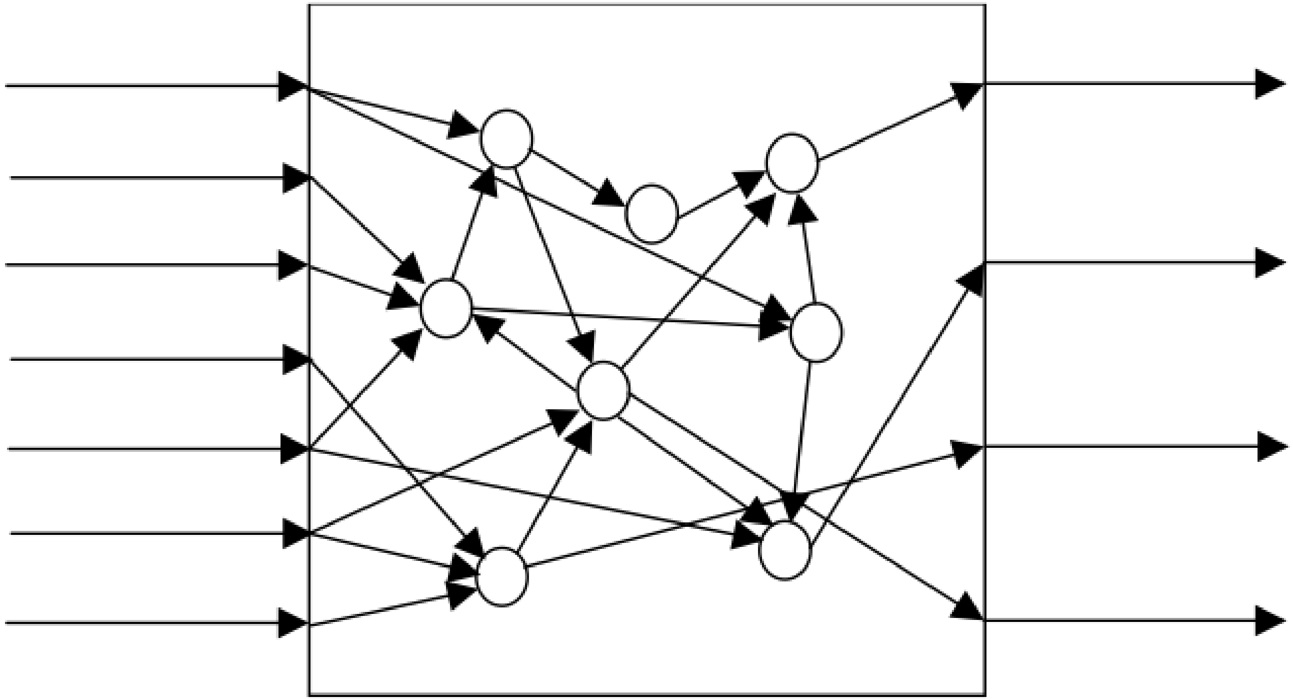
# Neurocomputação e o Neurônio Artificial

# Neurocomputação

- A computação realizada pelo cérebro não segue os mesmos princípios associados a máquinas de Von Neumann;
- Máquinas Von Neumann:
  - Realizam **processamento** e **armazenamento** de dados em dispositivos fisicamente distintos;
- Redes Neurais Artificiais:
  - O processamento ocorre de forma paralela e distribuída;
  - Deve ser possível ocorrer aprendizado;
  - Usam o mesmo dispositivo físico para armazenamento e processamento de dados.

# Neurocomputação

- Arquitetura neurocomputacional é baseada na **interconexão de unidades de processamento:**
  - Simples e similares (neurônios artificiais);
  - Dotadas de grande poder de adaptação (paradigma conexionista).



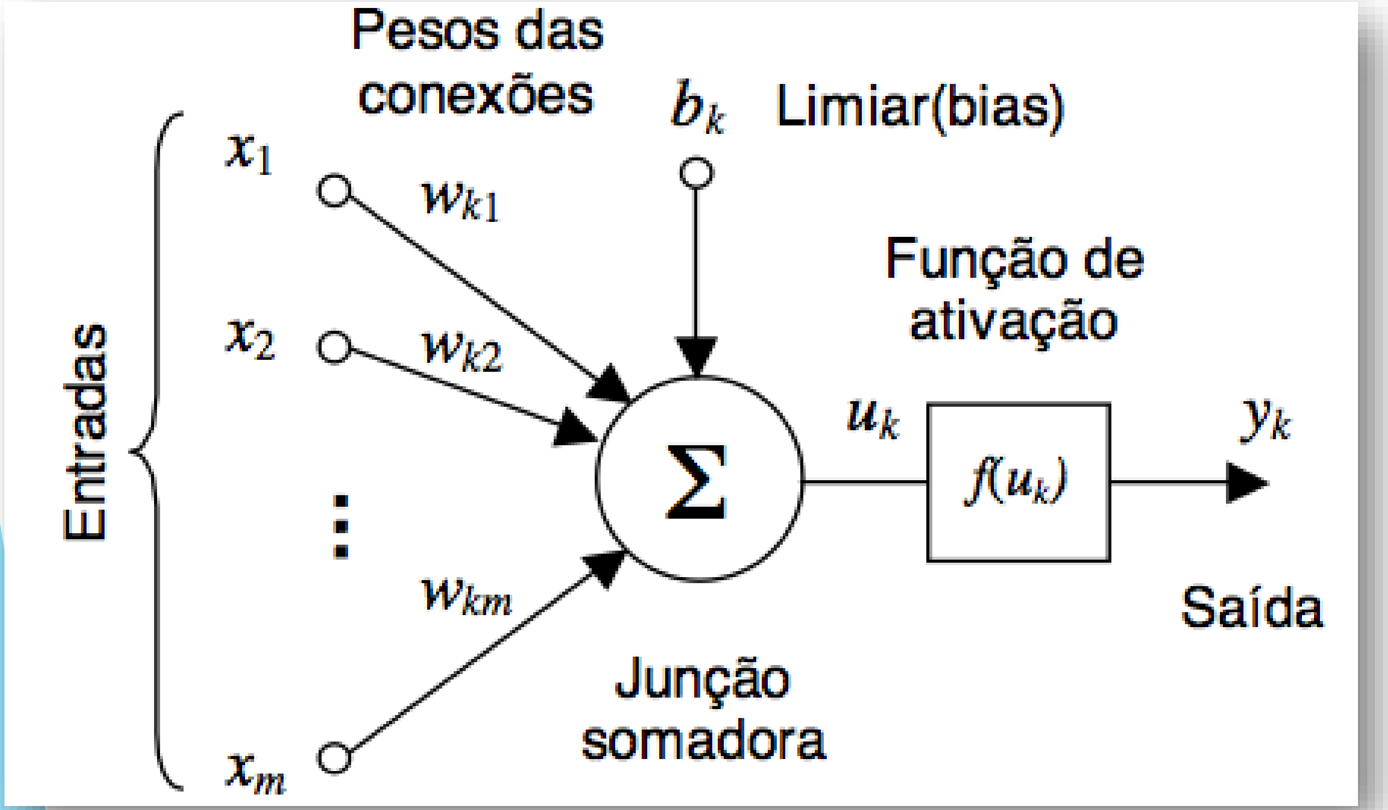
# Neurocomputação

- Motivação por trás da neurocomputação:
  - Possibilidade de elaborar mecanismos distintos de solução para problemas intratáveis ou ainda não-resolvidos com base na computação convencional;
  - Criar condições para reproduzir habilidades cognitivas e de processamento de informação muito desejadas em aplicações de engenharia, mas apresentadas apenas por algumas espécies animais.

- Existem múltiplas propostas de redes neurais artificiais, voltadas para diferentes aplicações e contextos;
  - Neste curso veremos apenas um tipo: **Perceptrons Multicamadas;**
  - Voltados para predição de dados (estimação e classificação).

# O Neurônio Artificial

- **Um dos** modelos matemáticos de neurônio artificial mais usados em RNAs é o *perceptron*, dado por:

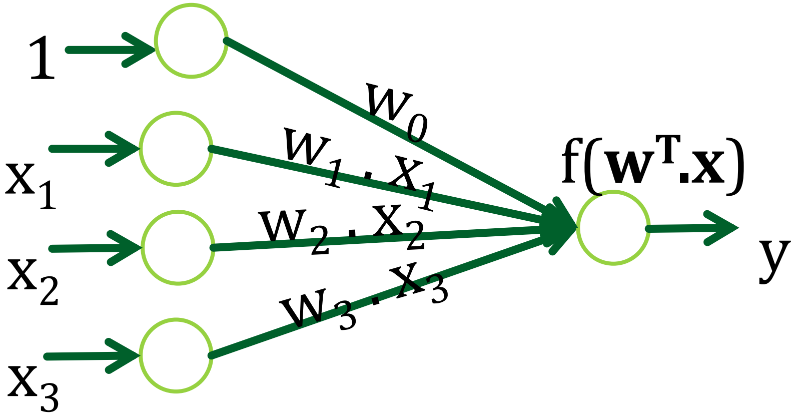



Saída:

$$y_k = f(u_k) = f\left(\sum_{j=1}^{m} w_{kj}x_j + b_k\right)$$

# O Neurônio Artificial

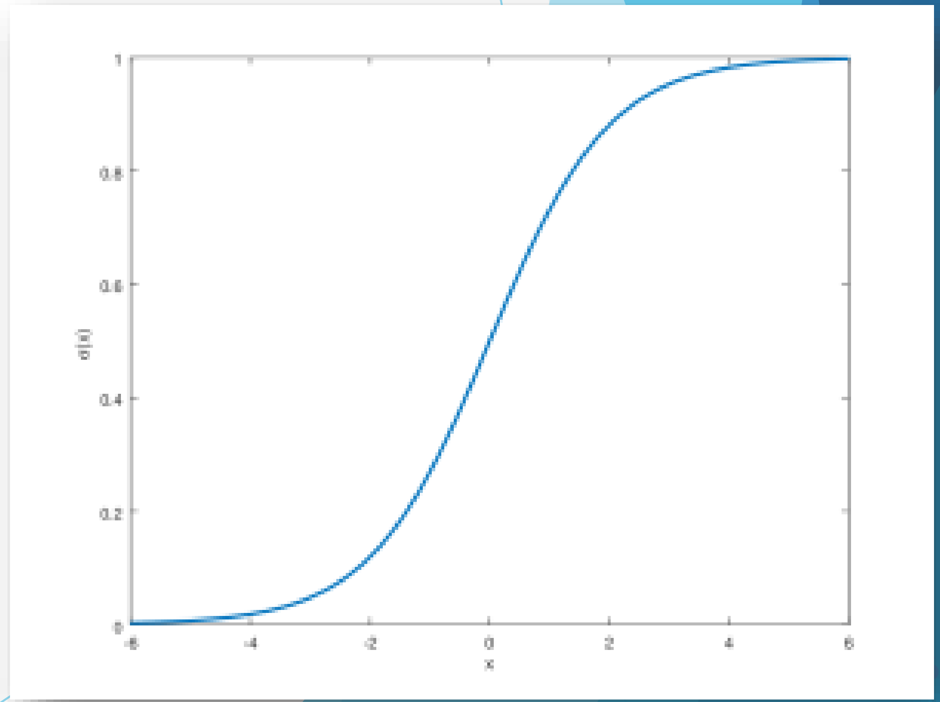
- A ideia geral do *perceptron* é que o neurônio deve gerar um **sinal** (*saída*) a partir dos **estímulos** (*entradas*) recebidos;
  - Cada entrada $i$ recebida é **amplificada** ou **atenuada** por um peso $w_i$.
  - Tais entradas amplificadas/atenuadas são **combinadas** e **deslocadas** (por um *bias*) em um módulo integrador (**"junção somadora"**);
  - À saída do integrador é aplicada uma **função de ativação** $f$.
- Uma simplificação da formulação anterior é considerar o *bias* como um peso $w_{kj} = b_k$, aplicado a uma entrada constante:




$$y_k = f(u_k) = f\left(\sum_{j=0}^{m} w_{kj} x_j\right) = f(\boldsymbol{w}^T \boldsymbol{x})$$

# O Neurônio Artificial – Função de Ativação

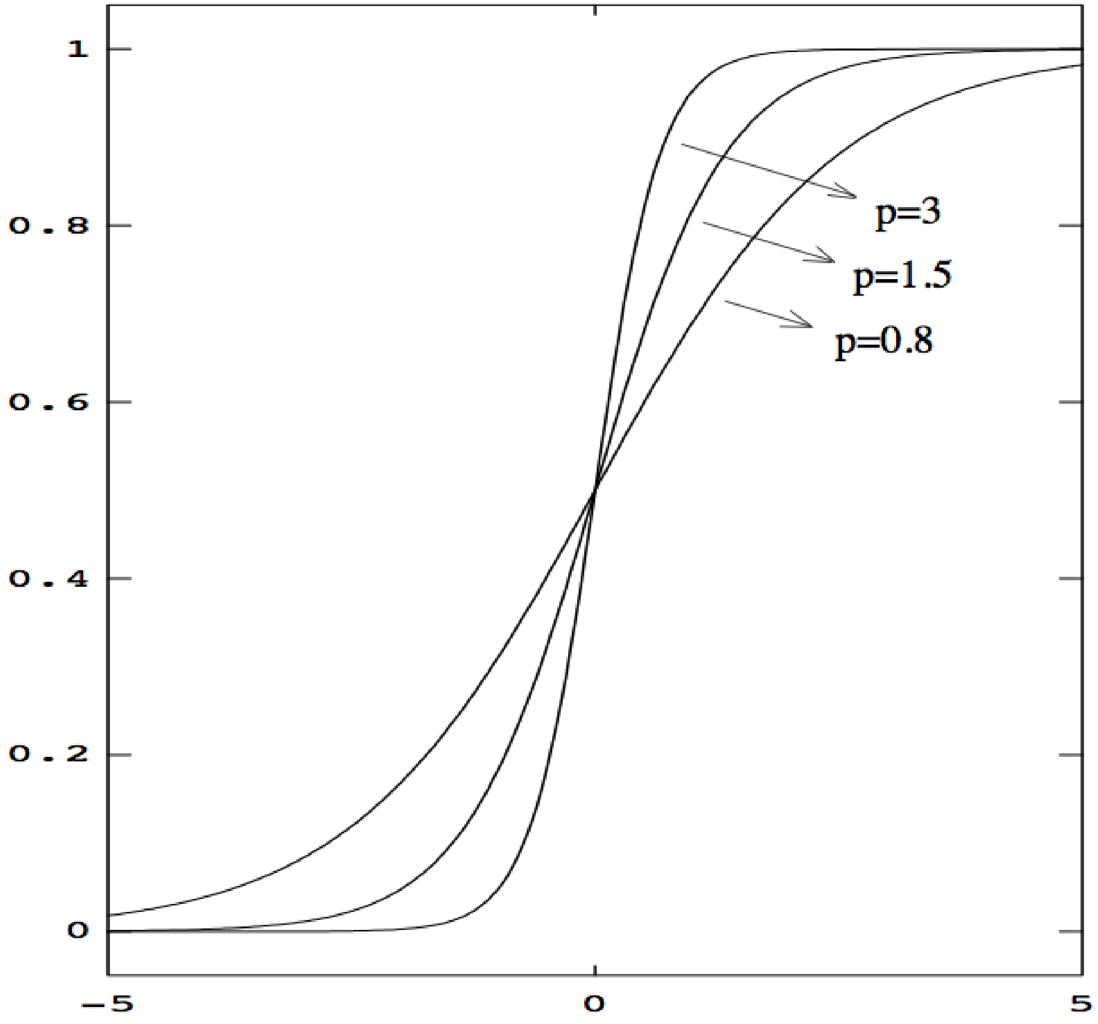
- As funções de ativação são funções que recebem a saída do integrador e determinam a "intensidade" da saída do neurônio;
  - Ou seja, a **ativação** do neurônio;
- Geralmente são usadas funções que realizam um mapeamento **monotonicamente crescente** entre entradas e saídas;
- Mais especificamente, funções de **formato sigmoide**:
  - Monotônicas e que atendem às seguintes condições:
    - $f(-\infty) = 0$ ou $f(-\infty) = -1$;
    - $f(+\infty) = 1$.
- Em alguns casos, são usadas funções lineares.
- Na maioria das aplicações devem ser **diferenciáveis.**



# O Neurônio Artificial – Função de Ativação

- Funções sigmoides muito usadas em RNAs:
  - **Função Logística:**

$$y_k = f(u_k) = \frac{1}{1 + e^{-p \cdot u_k}}$$

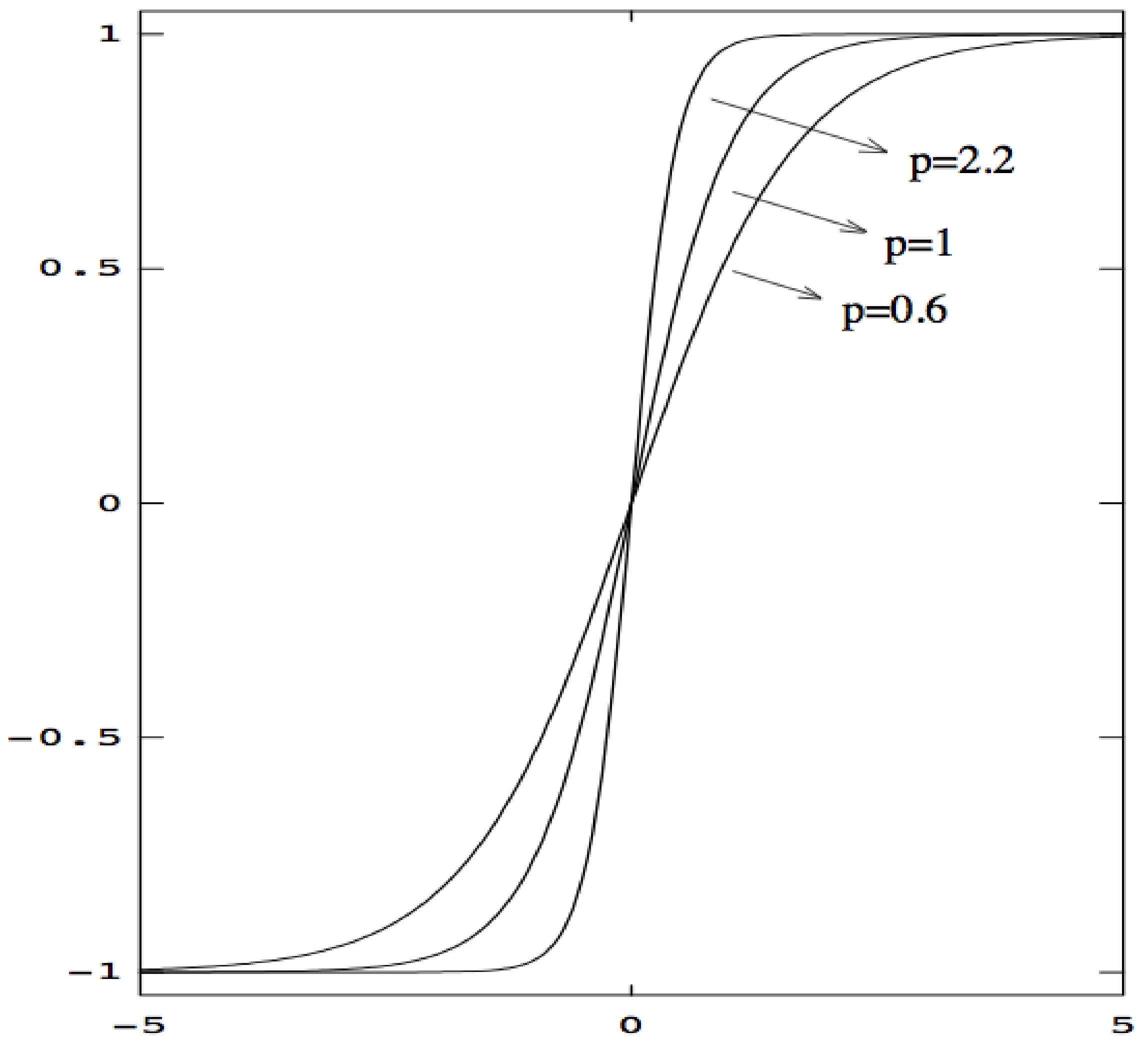


# O Neurônio Artificial – Função de Ativação

- Funções sigmoides muito usadas em RNAs:
  - **Função Tangente Hiperbólica:**

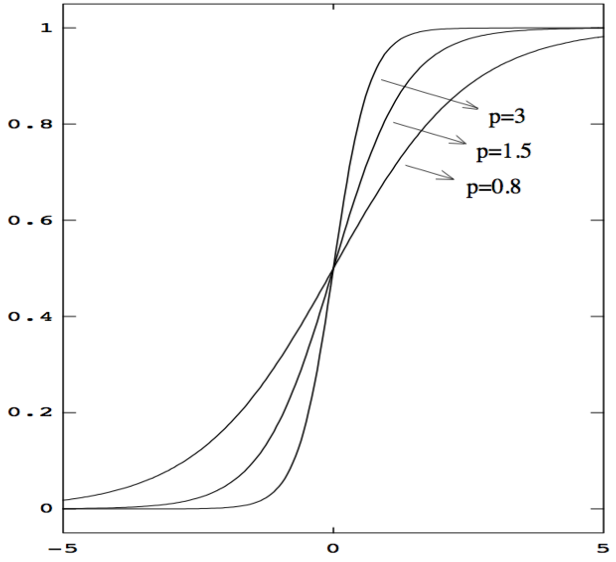
$$y_k = f(u_k) = \tanh(p \cdot u_k)$$

$$y_k = f(u_k) = \frac{e^{p \cdot u_k} - e^{-p \cdot u_k}}{e^{p \cdot u_k} + e^{-p \cdot u_k}}$$

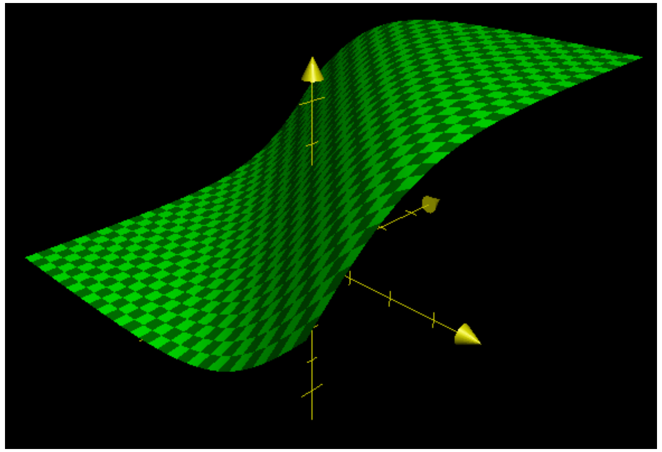


# Geometria de um neurônio artificial

- O mapeamento entre entradas e saída, realizado por um *perceptron*, é uma **função de expansão ortogonal** (*ridge function*);
  - Ou seja, funções que têm:
  - Forma **sigmoide** em uma direção;
  - Constantes nas demais direções ortogonais a esta única direção em que a forma da função se manifesta.

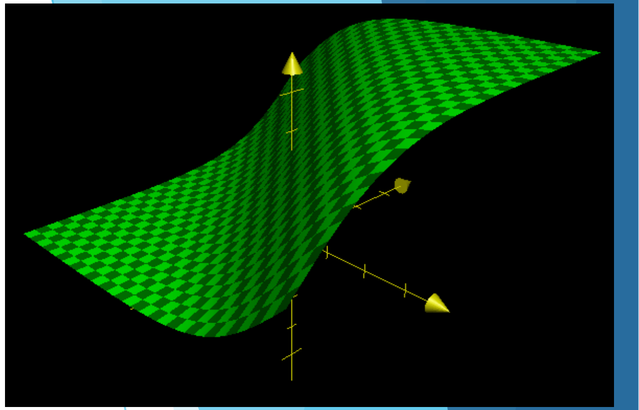



$$y_k = f(u_k) = \frac{1}{1+e^{-p \cdot u_k}}$$



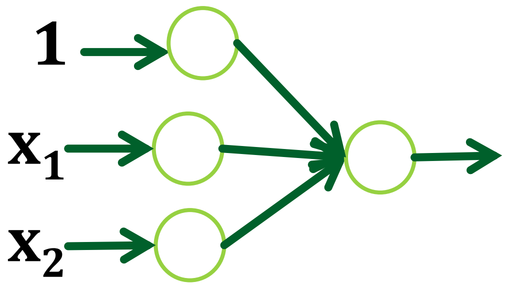
$$y_k = f(\vec{x}) = \frac{1}{1+e^{-[1 \quad 1] \cdot \vec{x}}}$$

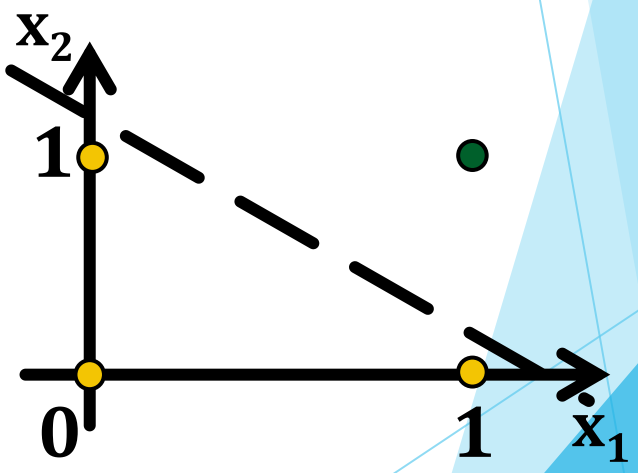
# Geometria de um neurônio artificial



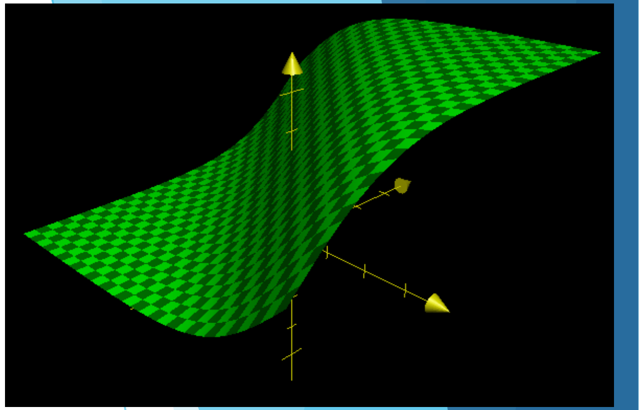
- RNAs com um **único neurônio artificial como o visto anteriormente** podem ser usadas para aprender a tratar problemas **linearmente separáveis**;
- **Ex.**: função lógica E (AND)

| $x_1$ | $x_2$ | $x_1$ E $x_2$ |
| --- | --- | --- |
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |



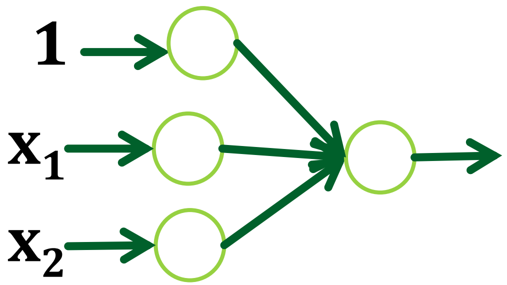



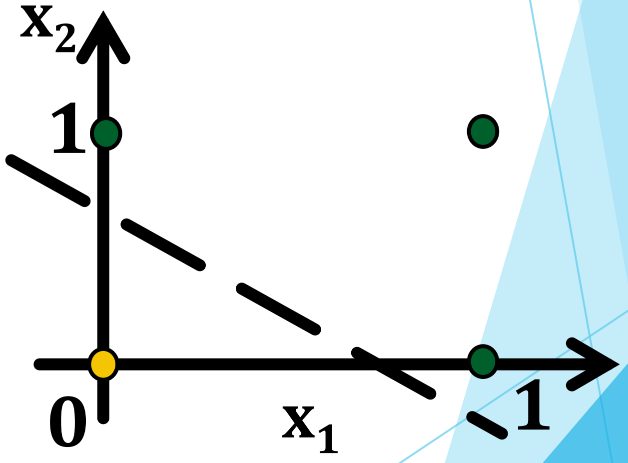
# Geometria de um neurônio artificial



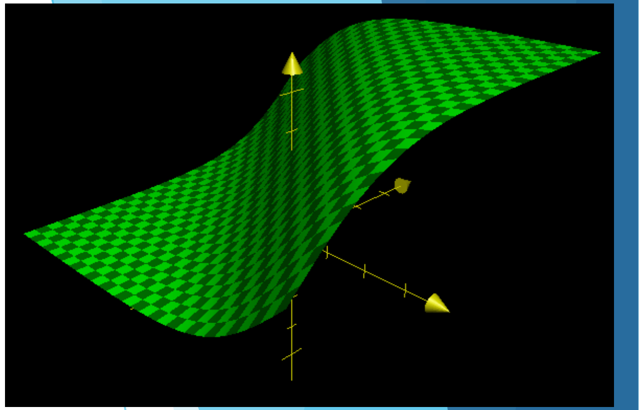
- RNAs com um **único neurônio artificial como o visto anteriormente** podem ser usadas para aprender a tratar problemas **linearmente separáveis**;
- **Ex.**: função lógica OU (OR)

| $x_1$ | $x_2$ | $x_1$ OU $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |



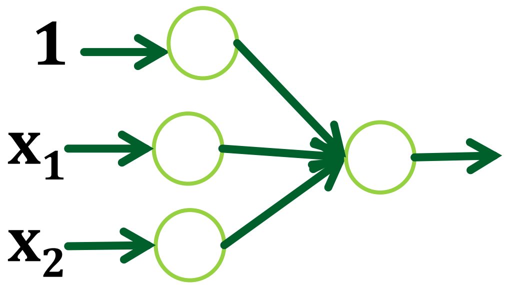



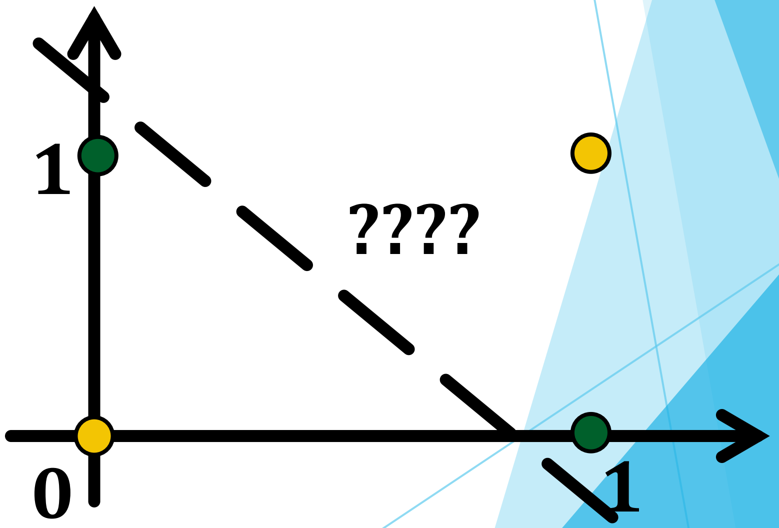
# Geometria de um neurônio artificial



- RNAs com um **único neurônio artificial como o visto anteriormente** podem ser usadas para aprender a tratar problemas **linearmente separáveis**;
- **Ex.**: função lógica XOR (OU Exclusivo)

| $x_1$ | $x_2$ | $x_1$ XOR $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |



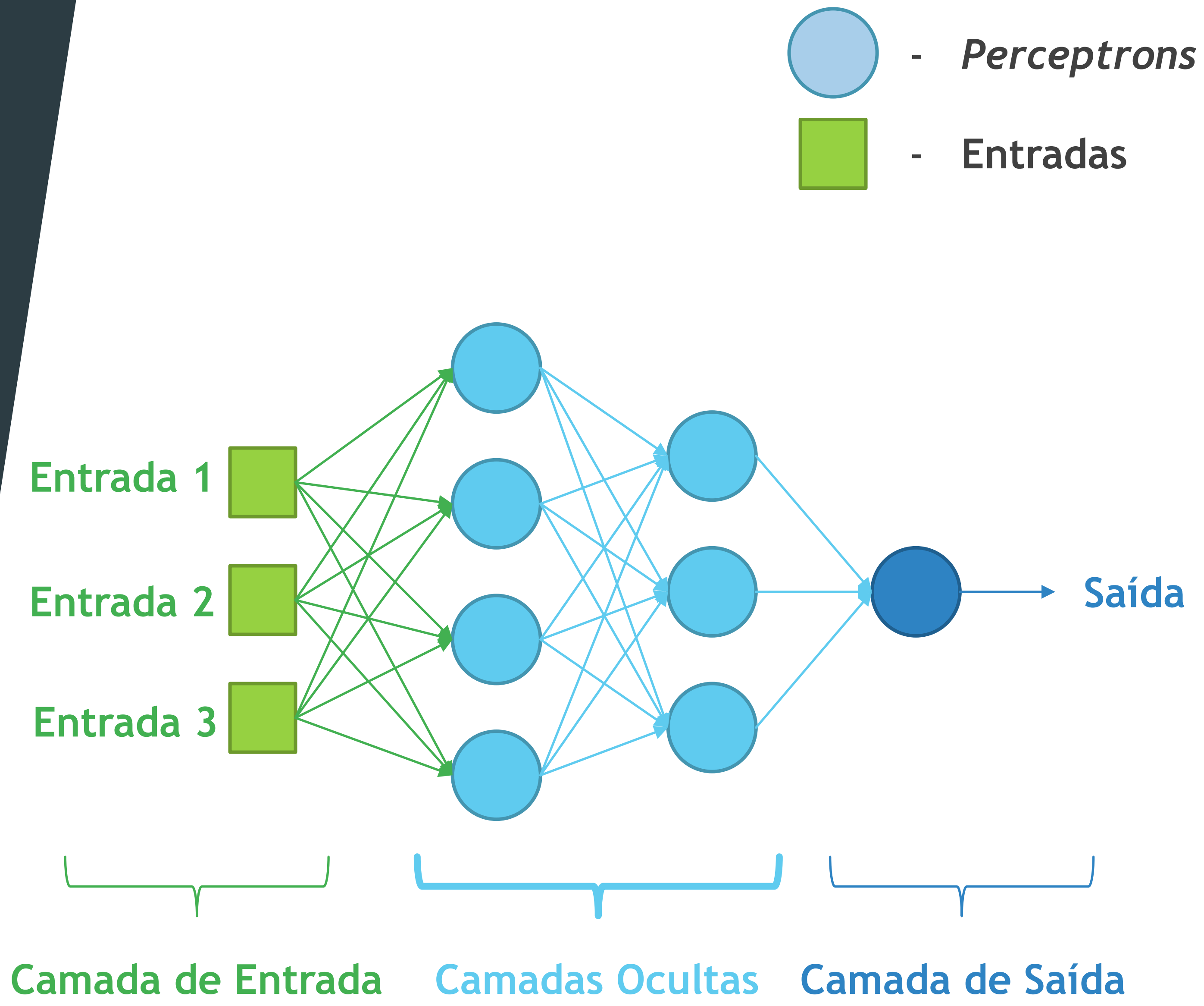




- Para realizar mapeamentos **não-lineares**, é necessário combinar múltiplos neurônios em **camadas** → Redes Neurais de Múltiplas Camadas.

# Redes Neurais de Múltiplas Camadas

# Redes Neurais de Múltiplas Camadas

- O processo de conexão entre neurônios artificiais leva à geração de sinapses e à construção de redes neurais artificiais.
- As estruturas mais conhecidas são em camadas (*layers*):
  - A **saída** de cada neurônio de uma camada precedente é **entrada** para todos os neurônios da camada seguinte (redes *feed-forward*).
  - **Ex.**: *Multilayer Perceptrons* (MLPs)

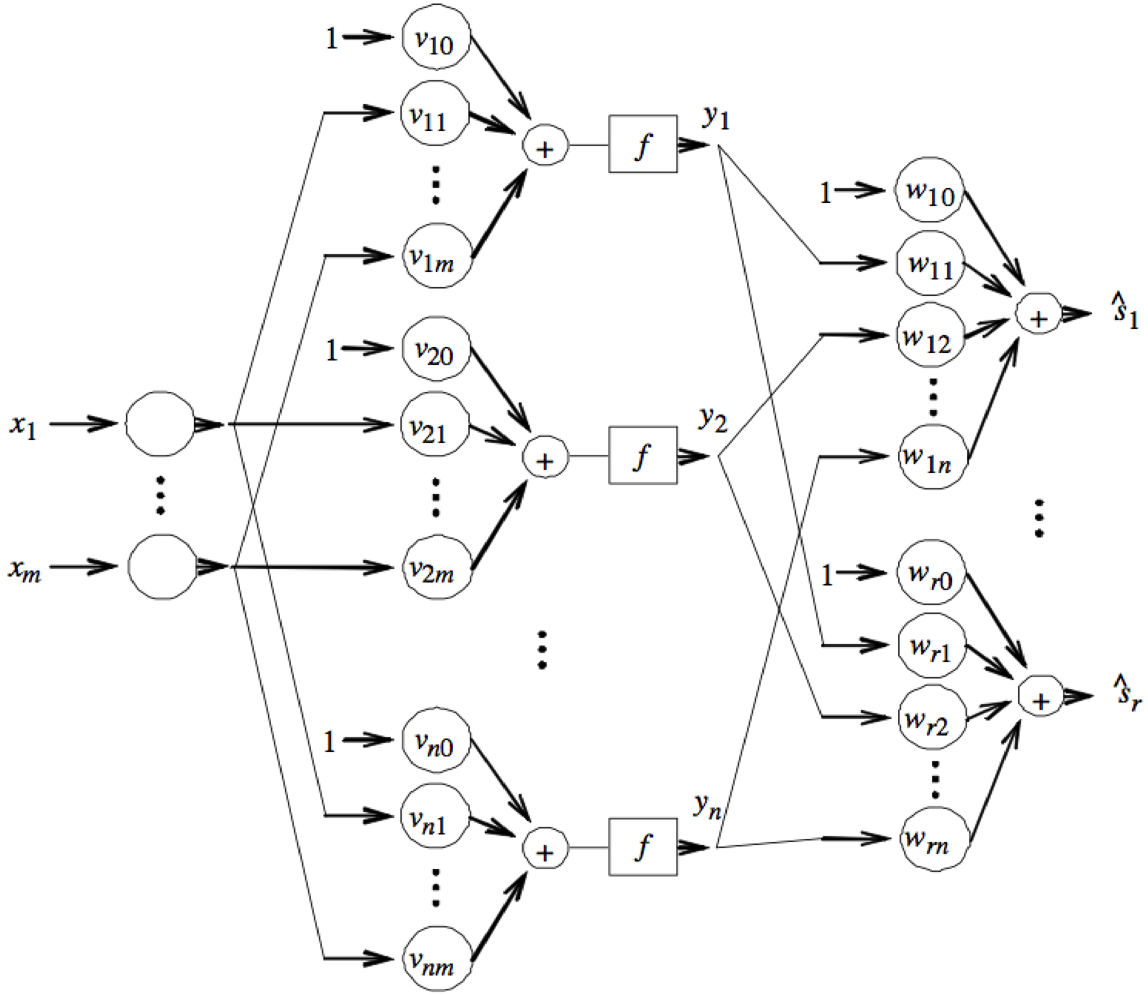


# *Multilayer Perceptrons* (MLP)

- Nas MLPs, cada neurônio corresponde a um *perceptron* como os vistos anteriormente;
- O número de camadas (*layers*) e o número de neurônios em cada camada deve ser definido de acordo o problema a ser tratado;
  - Deve-se considerar a capacidade de generalização da rede (será discutido adiante);
- A maioria das aplicações emprega MLPs com uma **única camada intermediária** (oculta):
  - Com apenas uma camada oculta a rede neural já apresenta capacidade de **aproximação universal** (CYBENKO, 1989; HORNIK et al., 1989; HORNIK et al., 1990; HORNIK et al., 1994).

# *Multilayer Perceptrons* (MLP)

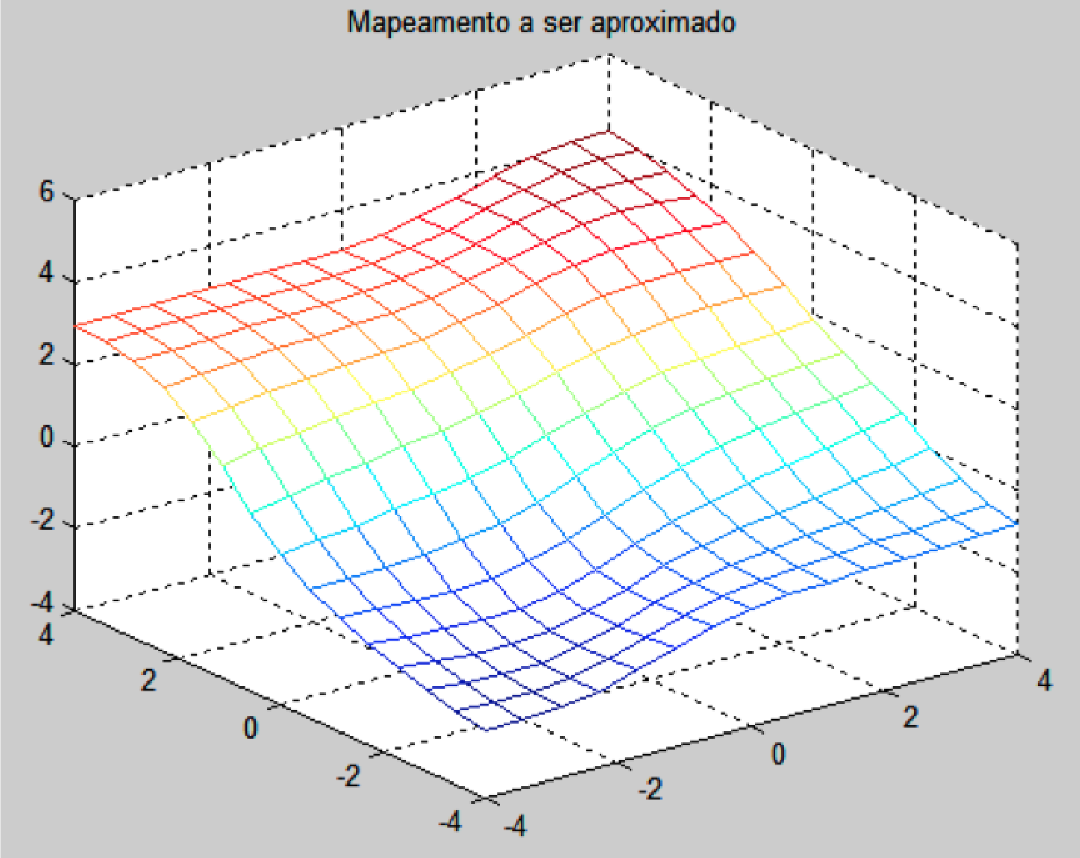
- Exemplo:
  - MLP com uma camada oculta;
  - Neurônios na camada de saída com função de ativação linear;
  - Saídas são uma combinação linear das entradas;




$$s_k = \sum_{j=0}^{n} w_{kj} y_j = \sum_{j=0}^{n} w_{kj} f\left(\sum_{i=0}^{m} v_{ji} x_i\right) = \sum_{j=0}^{n} w_{kj} f(\mathbf{v}_j^T \mathbf{x})$$

# Contribuição de cada Neurônio (MLP)

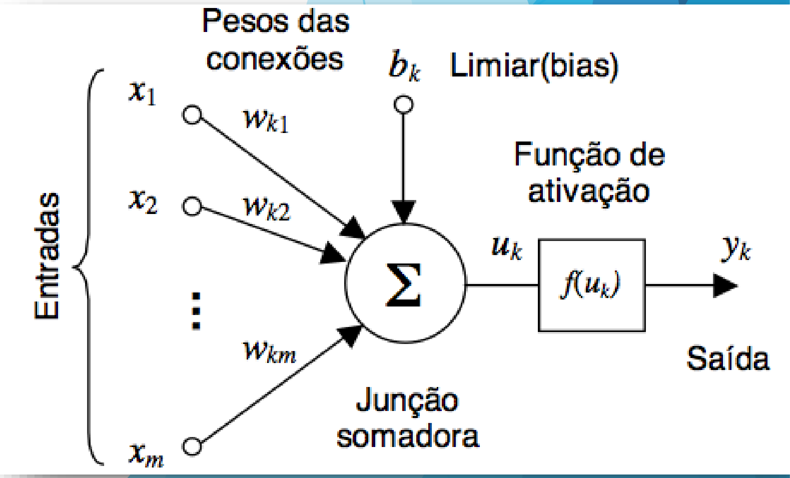
- O mapeamento não-linear realizado por uma rede neural do tipo MLP como a do *slide* anterior é uma **combinação linear** de **funções de expansão ortogonal** (*ridge functions*);
- Para ilustrar, considere o seguinte mapeamento de $\mathbb{R}^2$ (duas entradas) para $\mathbb{R}^1$ (uma saída), que deve ser feito por uma rede MLP com 5 neurônios na camada intermediária e funções de ativação tangente hiperbólica:



# Contribuição de cada Neurônio (MLP)

- **Exemplo**: continuação
  - Número de entradas $(n_i)$: 2
  - Número de camadas ocultas: 1;
  - Número de neurônios na camada oculta $(n_h)$: 5;
  - Número de saídas $(n_o)$: 1
  - Considerando que todos os neurônios têm *bias*, o total de pesos a serem ajustados é:

$$(n_i+1) \times n_h + (n_h + 1) \times n_o = (2 + 1) \times 5 + (5 + 1) \times 1 = 15 + 6 = 21$$



# Contribuição de cada Neurônio (MLP)

- **Exemplo**: continuação (pesos)
  - Pesos obtidos para cada neurônio da **camada intermediária,** após o treinamento (um neurônio por coluna, *bias* na primeira linha):

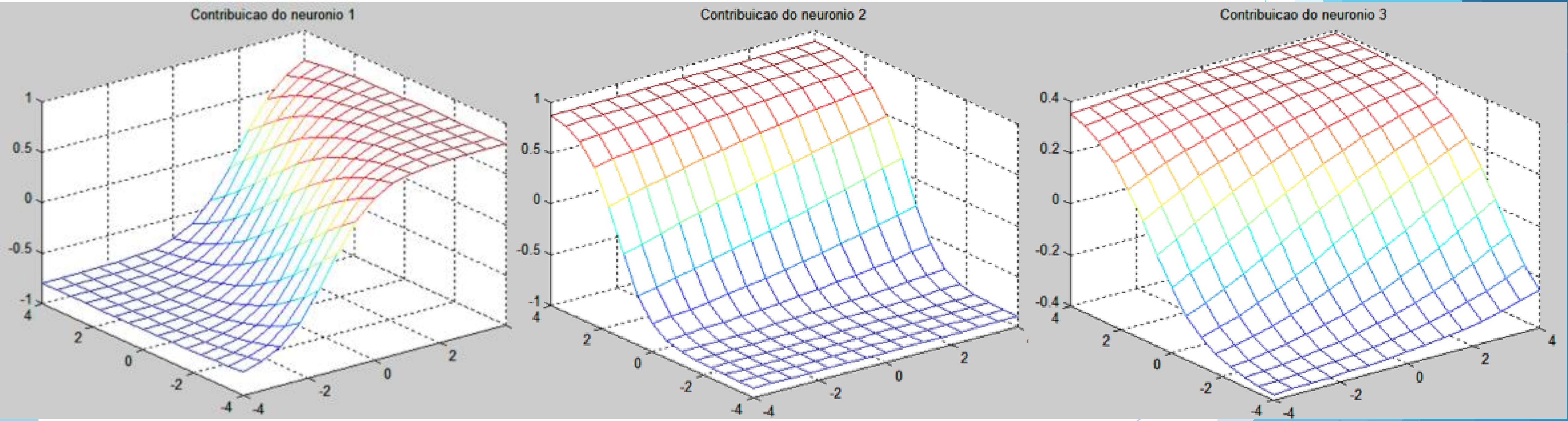
| | | | | |
|---|---|---|---|---|
| -0.2000893971446 | -0.7005190801004 | 0.3969922184411 | -0.1000386326727 | 0.6960626246728 |
| 0.7001816852893 | 0.1001586041766 | 0.1986002882348 | -0.2999619530380 | 0.2986911223548 |
| -0.3000639814659 | 0.8002220985579 | 0.4937240042168 | 0.5000542722296 | 0.8951501213136 |

  - Pesos da camada saída:

0.99989340388393
0.79971888341317
0.90007841696146
0.38564988369799
0.79996881679466
0.71442550587375

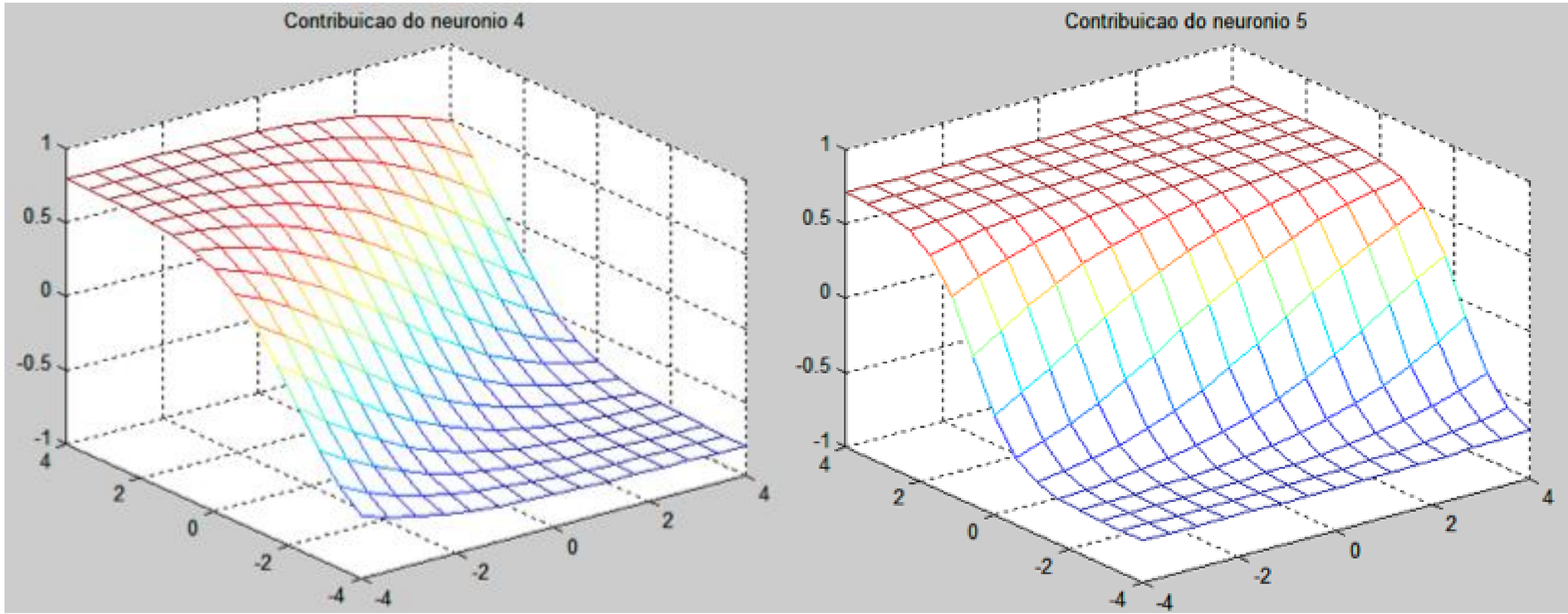
# Contribuição de cada Neurônio (MLP)

- **Exemplo**: continuação (*ridge functions* dos neurônios da camada oculta)



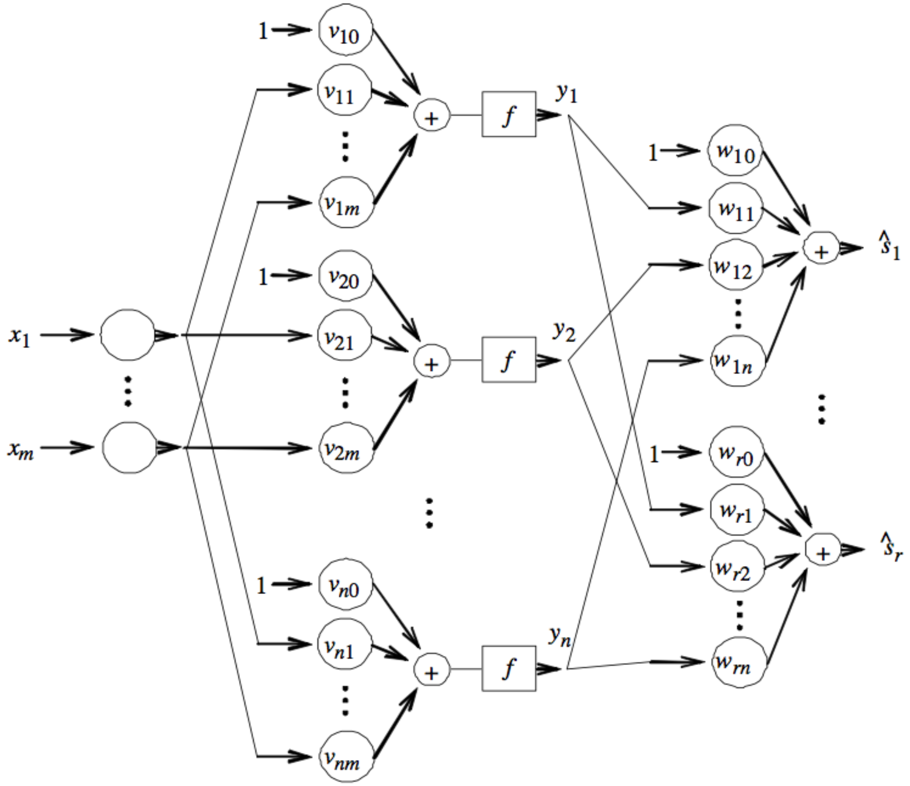
# Contribuição de cada Neurônio (MLP)

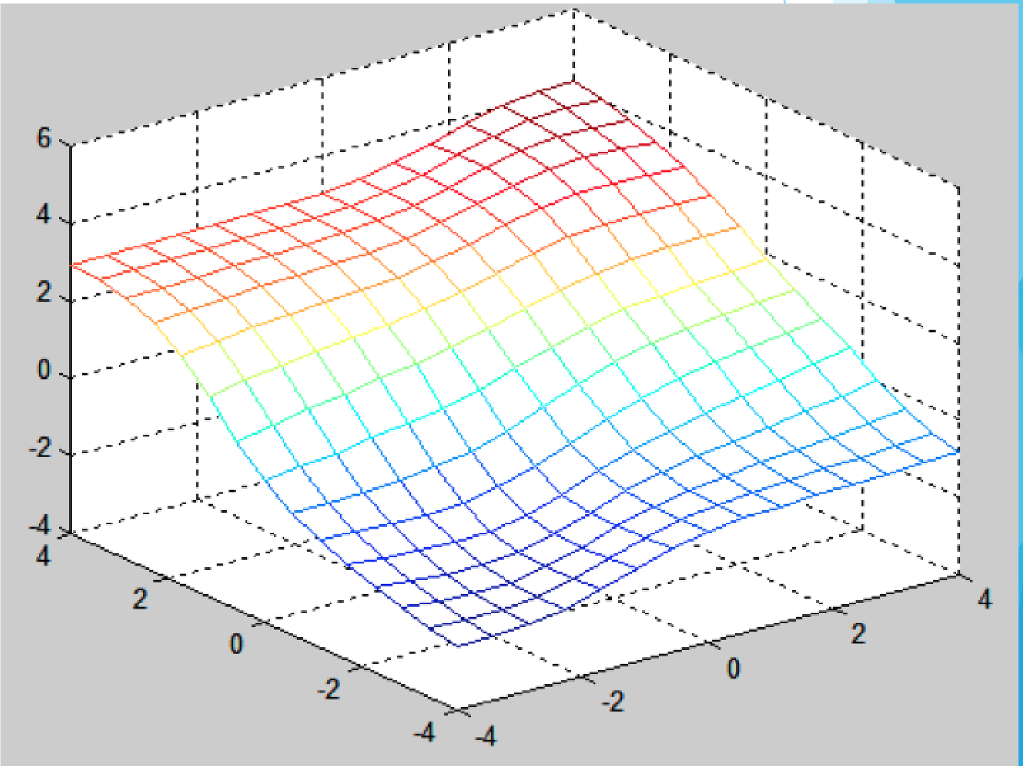
- **Exemplo**: continuação (*ridge functions* dos neurônios da camada oculta)



# Contribuição de cada Neurônio (MLP)

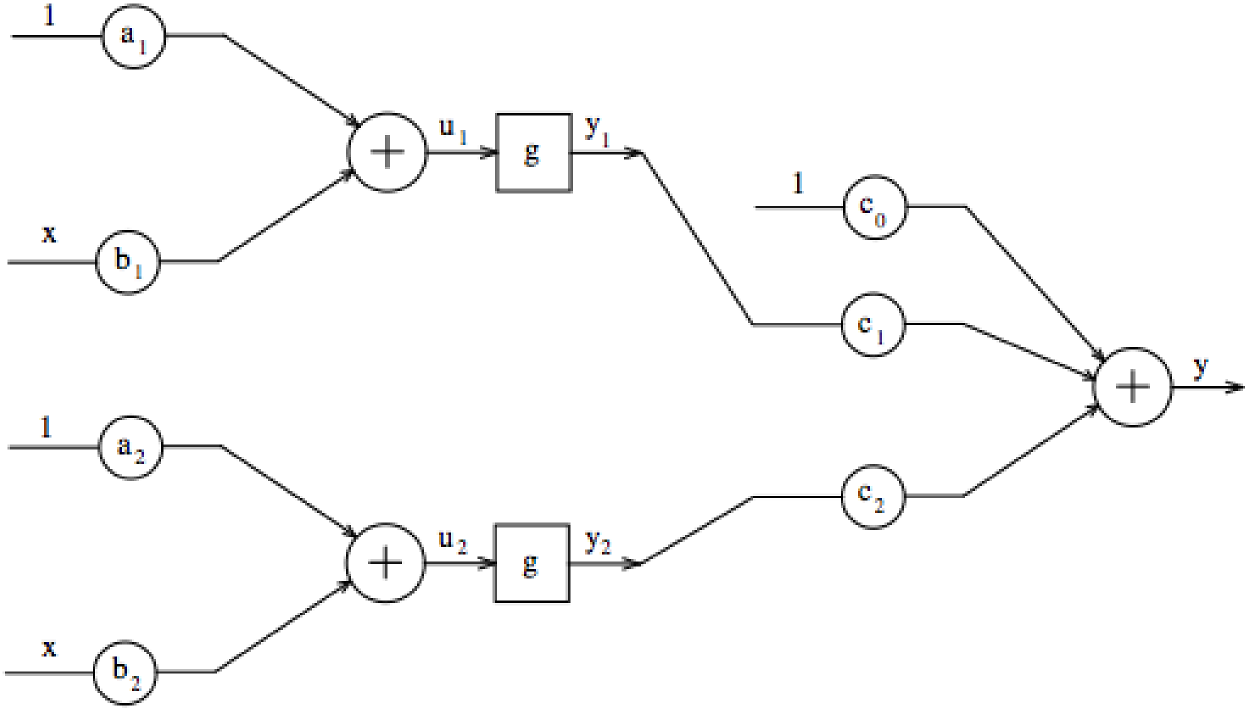
- **Exemplo**: continuação
  - Na camada de saída, estas contribuições individuais são ponderadas e combinadas como uma única superfície final de aproximação:



# O Papel dos Pesos Sinápticos

$$y = c_0 + \sum_{n=1}^{p} c_n g(b_n x + a_n)$$



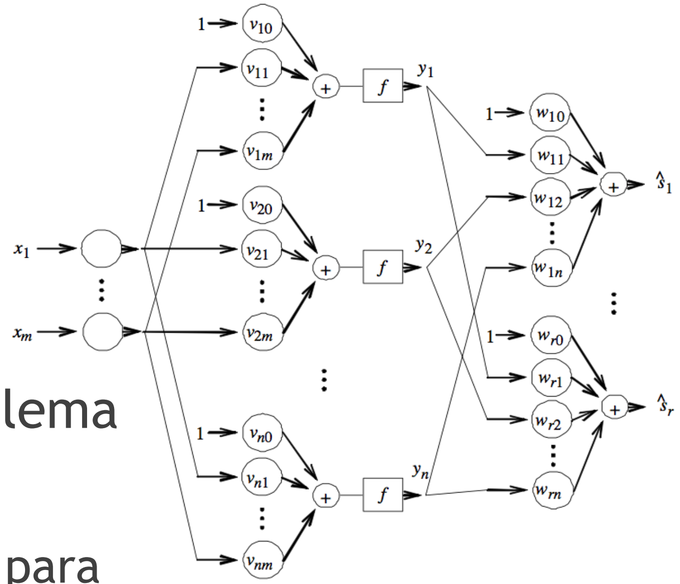

$$y = c_0 + c_1 g(b_1 x + a_1) + c_2 g(b_2 x + a_2) \Rightarrow \begin{cases} \text{a : deslocamento no eixo x} \\ \text{b : inclinação da sigmóide} \\ \text{c : amplitude da sigmóide} \end{cases}$$

# Aprendizado em MLPs

# Aprendizado em MLPs

- Até aqui nós vimos que, dada uma rede neural do tipo MLP com uma camada intermediária e um número suficiente de neurônios nesta camada, qualquer superfície pode ser aproximada:
  - Basta que, para isso, sejam definidos os pesos sinápticos de forma adequada;
- Este processo de ajuste de pesos sinápticos é conhecido como **treinamento da rede**:
  - Dado um conjunto de amostras que representam o mapeamento a ser obtido, os pesos sinápticos devem ser definidos de forma que a rede **aprenda** a aproximar tais amostras;
    - Aprendizado supervisionado!
  - O ajuste dos pesos é feito de forma a **minimizar o erro** entre a saída da rede e a saída desejada.

# Aprendizado em MLPs



- O aprendizado supervisionado em MLPs pode ser visto como um problema de **otimização**:
  - Busca-se **minimizar o erro** entre a saída de rede e os valores esperados para cada amostra dos dados;
- Como visto antes, a saída de uma MLP com uma camada oculta é dada por:

$$\breve{s} = \sum_{j=0}^{n} w_j y_j = \sum_{j=0}^{n} w_j f\left(\sum_{i=0}^{m} v_{ji} x_i\right) = \sum_{j=0}^{n} w_j f\left(\mathbf{v}_j^T \mathbf{x}\right)$$

- Sendo assim, o ***erro quadrático médio*** da RNA para as *N* amostras será dado por:

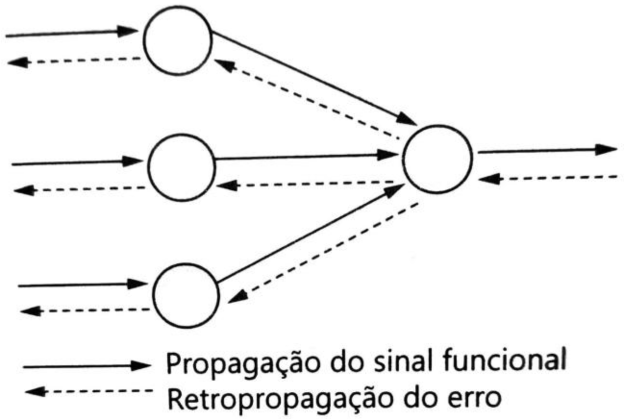
$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{l=1}^{N}(\breve{s}_l - s_l)^2 = \frac{1}{N}\sum_{l=1}^{N}\left(\sum_{j=0}^{n} w_j f\left(\mathbf{v}_j^T \mathbf{x}_l\right) - s_l\right)^2$$

$N$: num. amostras de treinamento

$\boldsymbol{\theta}$: pesos da MLP

# Aprendizado em MLPs

- O problema de minimização de $J(\boldsymbol{\theta})$ consiste em encontrar o vetor de pesos $\boldsymbol{\theta} \in \mathbb{R}^{\boldsymbol{p}}$ que leve ao menor erro de treinamento.
- Se as funções de ativação forem diferenciáveis, é possível utilizar métodos fechados de otimização baseados em gradiente → ***Algoritmo Backpropagation*** (HAYKIN, 1999; HAYKIN, 2008);
  - Sinal (entradas) são propagadas pela MLP até a saída, com os pesos sendo mantidos fixos;
  - O erro observado na saída é ***retropropagado*** **(sentido oposto ao anterior)**, em um processo que leva ao ajuste dos pesos.



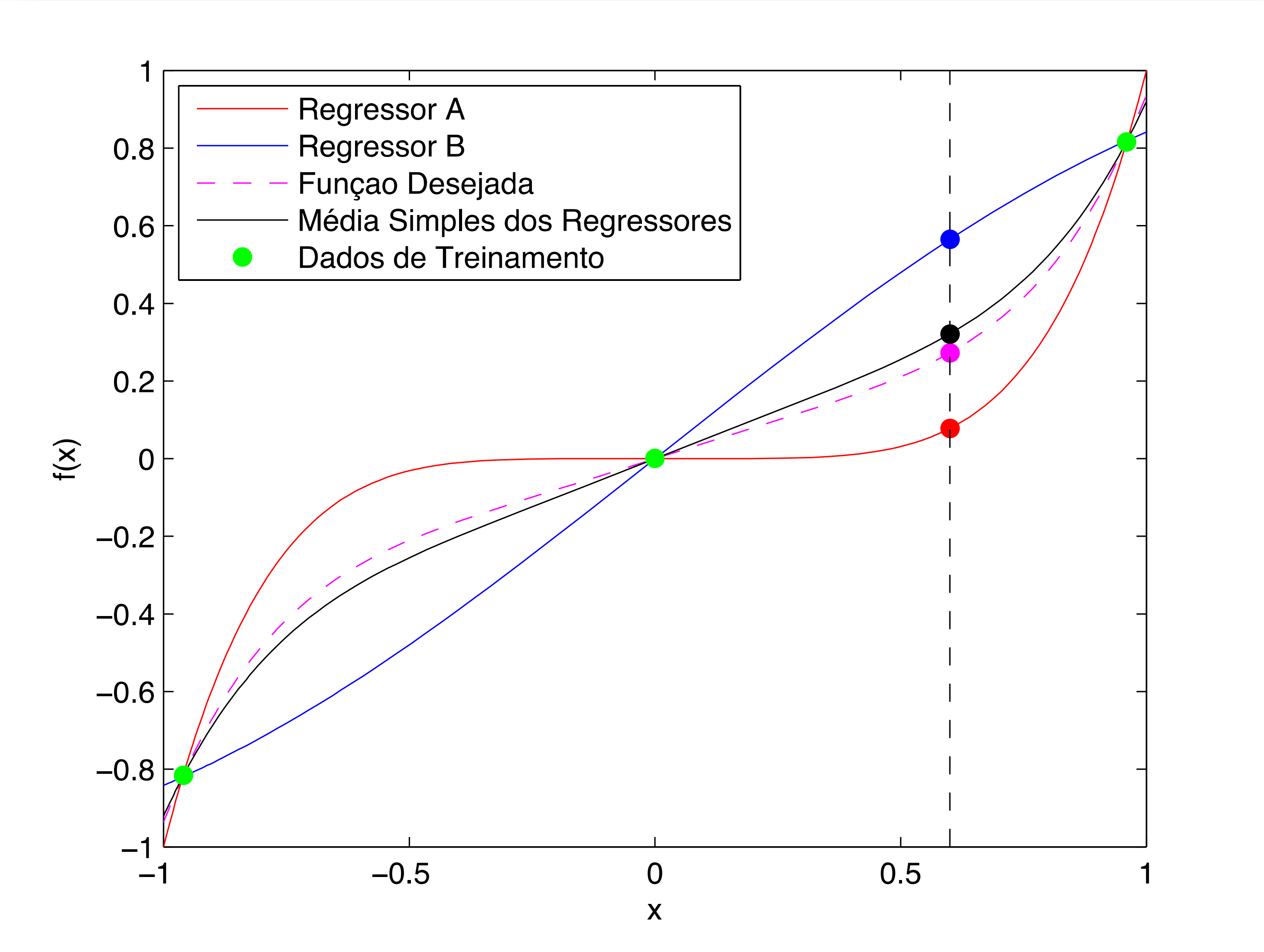

Outras estratégias de busca também podem ser usadas: algoritmos evolutivos, inteligência de enxame ...

# Capacidade de Generalização em MLPs

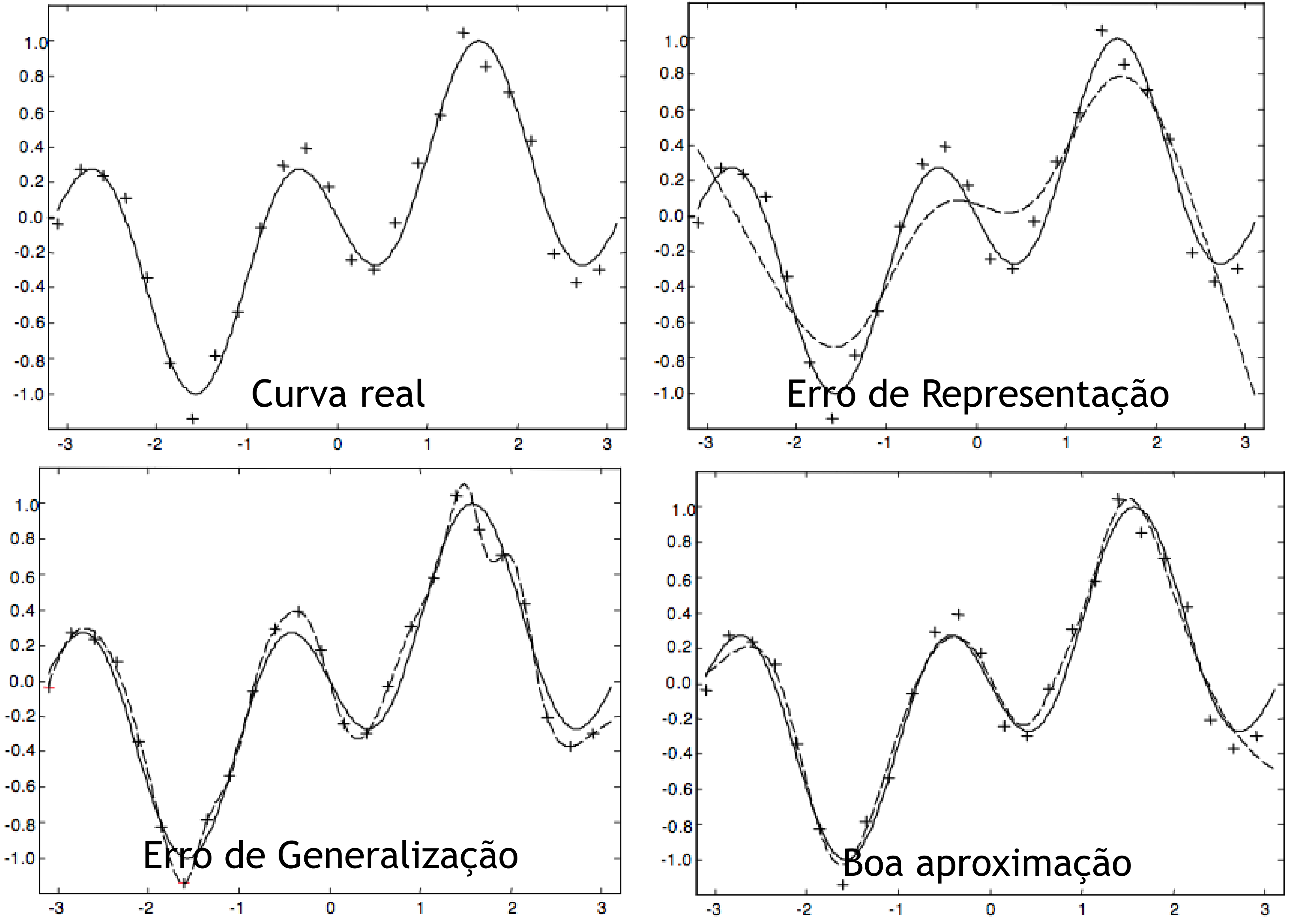
- Um problema comum a todos os modelos de aproximação de funções que possuem capacidade de aproximação universal (como as redes MLP): a necessidade de controlar adequadamente o seu ***grau de flexibilidade***;
  - Erro de Representação x Erro de Generalização (Dilema *bias-variância*);

- O conjunto de amostras disponível para treinamento supervisionado é finito:
  - Infinitos mapeamentos podem produzir o mesmo desempenho de aproximação, independentemente do critério de desempenho adotado.
  - Esses mapeamentos alternativos vão diferir justamente onde não há amostras disponíveis para diferenciá-los.

# Capacidade de Generalização em MLPs



# Capacidade de Generalização em MLPs

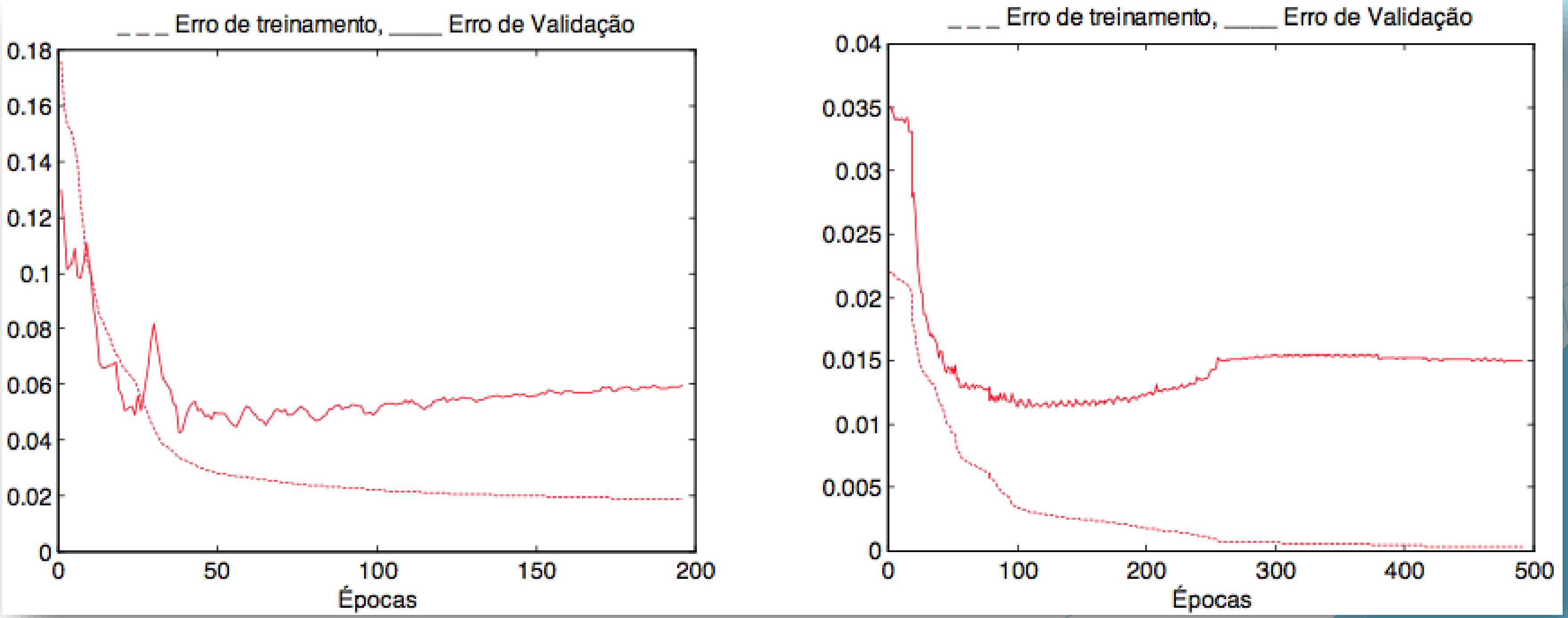


# Capacidade de Generalização em MLPs

- É necessário adotar alguma estratégia, durante o treinamento, para **maximizar a capacidade de generalização** de uma MLP;
  - Ou seja, maximizar sua capacidade de aproximar corretamente amostras de dados inexistentes no treinamento.
- Uma estratégia recomendada é dividir o conjunto de dados disponível em três partes:
  - **Conjunto de Treinamento**: amostras que serão empregadas no ajuste dos pesos;
  - **Conjunto de Validação**: empregado para definir o momento de interrupção do treinamento.
  - **Conjunto de Teste**: usado para avaliar a capacidade de generalização da rede, após o treinamento.
- Deve-se assegurar que **todos os conjuntos sejam suficientemente representativos do mapeamento que se pretende aproximar**.

# Capacidade de Generalização em MLPs

- Nas figuras abaixo pode-se encontrar curvas típicas observadas para os erros quadráticos médios de treinamento e de validação:
  - Como a curva do erro de validação oscila bastante e esboça um comportamento pouco previsível, recomenda-se sobreajustar a rede e armazenar os pesos associados ao mínimo do erro de validação.



# Capacidade de Generalização em MLPs

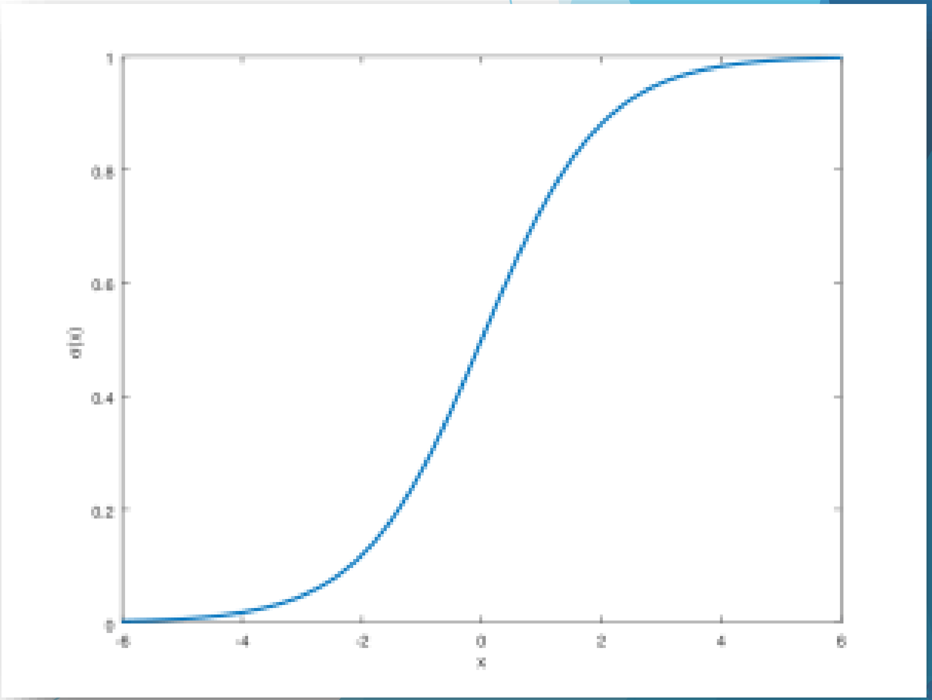
- Não existe um consenso sobre como particionar os dados em **treinamento**, **validação** e **teste**:
- Uma sugestão seria dividir as amostras em:
  - 70% para treinamento;
  - 20% para validação; e
  - 10% para teste.

- A sugestão anterior só é válida se existirem amostras suficientes para representar adequadamente o problema em todas as partições!

# Outros Aspectos

# Valor inicial dos pesos

- A forma mais simples de inicialização dos pesos de uma rede MLP é **atribuir valores pequenos e aleatoriamente distribuídos ao redor de zero;**
- Esta inicialização faz com que a rede inicial tenha as seguintes propriedades:
  - Seu mapeamento se aproxima de um hiperplano, sem nenhuma tendência definida sob o ponto de vista de comportamento não-linear;
  - A ativação de todos os neurônios se encontra **fora da região de saturação** (região em que a sigmoide possui valor +1 ou -1) → facilita o processo de ajuste dos pesos.



# Preparação dos dados de entrada

- O **pré-processamento dos dados** a ser feito antes do treinamento de uma MLP pode afetar significativamente o seu desempenho.
- Nesta etapa deve-se:
  - Definir o que será entrada e o que será saída na rede (como em toda estratégia de aprendizado supervisionado);
  - Tratar os dados faltantes → MLPs não funcionam com uma entrada "ausente";
  - Transformar os dados categóricos em dados numéricos → MLPs só aceitam valores numéricos na entrada:
    - Abordagem mais indicada: converter um atributo categórico que possui *n* categorias em *n* novas entradas binárias da rede;
    - A entrada correspondente à categoria de uma amostra é ajustada como "1" e as demais como "0".

# Preparação dos dados de entrada

- O **pré-processamento dos dados** a ser feito antes do treinamento de uma MLP pode afetar significativamente o seu desempenho.
- Nesta etapa deve-se:
  - Normalizar os dados para o intervalo suportado pelas funções de ativação:
    - Valores de entrada muito altos podem levar os neurônios diretamente para a região de saturação das funções de ativação → dificulta o processo de treinamento;
    - Recomenda-se normalizar cada atributo dos dados no intervalo (0, 1) ou (-1, 1), conforme a função de ativação.
      - E **guardar** as informações usadas na normalização dos dados para que novas amostras dos dados possam ser normalizadas da mesma forma.

- Mais informações: Seção 7.3.1 de ENGELBRECHT (2007).

# Preparação para classificação de dados

- Em problemas de classificação é necessário adotar alguma abordagem para **converter a saída contínua da rede em um rótulo indicativo da classe à qual a amostra apresentada deve ser atribuída**;
- Isto exige o ajuste:
  - Do formato da saída das amostras que serão usadas no treinamento;
  - Da ***camada de saída*** da MLP.
- Estratégias:
  - Se o problema tiver apenas duas classes, pode-se definir uma função de ativação sigmoide também na camada de saída da MLP:
    - Os dados de treinamento e validação devem ser ajustados com saídas “1” para uma das classes e “0” para a outra (ou “-1”, no caso de tangente hiperbólica);
    - As saídas da rede para uma dada amostra devem ser discretizadas.

Função Logística

- Saída 0.6 → 1.0
- Saída 0.2 → 0.0

# Preparação para classificação de dados

- Se o problema tiver **mais de duas classes**, a abordagem mais indicada é definir um neurônio de saída (com função de ativação sigmoide) para cada classe;
  - As amostras dos dados passam a ter número de saídas igual ao número de classes do problema;
  - A saída correspondente ao neurônio associado à classe à qual a amostra pertence deve ser ajustada como “1” na amostra, e as demais como “0” (ou “-1” no caso de tangente hiperbólica);
  - Para identificar a qual classe pertence uma dada amostra de dados, pode-se tomar a **saída que resultar em maior ativação**.

# Preparação para classificação de dados

- Por fim, deve-se tomar cuidados adicionais também no **particionamento** dos dados em conjuntos de treinamento, validação e teste:
  - Deve-se procurar respeitar a **distribuição dos dados** junto a cada classe;
  - Isto é válido independentemente do número de classes do problema!

# Referências Bibliográficas

# Referências Bibliográficas


DE CASTRO, L. N., FERRARI, D. G.. *Introdução à Mineração de Dados – Conceitos Básicos, Algoritmos e Aplicações*. Ed. Saraiva, 2016.

CYBENKO, G. *Approximation by superposition of sigmoidal functions*. Mathematics of Control, Signals and Systems, vol. 2, no. 4, pp. 303-314, 1989.

ENGELBRECHT, A. P. *Computational Intelligence – An Introduction*, Wiley, 2007.

HAYKIN, S. *Neural Networks: A Comprehensive Foundation*, 2nd Ed., Prentice-Hall, 1999.

HAYKIN, S. *Neural Networks and Learning Machines*, 3rd edition, Prentice-Hall, 2008.

HORNIK, K., STINCHCOMBE, M., WHITE, H. *Multi-layer feedforward networks are universal approximators*. Neural Networks, vol. 2, no. 5, pp. 359-366, 1989.

HORNIK, K., STINCHCOMBE, M., WHITE, H. *Universal approximation of an unknown function and its derivatives using multilayer feedforward networks*. Neural Networks, vol. 3, no. 5, pp. 551-560, 1990.

HORNIK, K., STINCHCOMBE, M., WHITE, H., AUER, P. *Degree of Approximation Results for Feedforward Networks Approximating Unknown Mappings and Their Derivatives*. Neural Computation, vol. 6, no. 6, pp. 1262-1275, 1994.