Inteligência Artificial

BCC35G

Diego Bertolini
diegobertolini@utfpr.edu.br
http://www.inf.ufpr.br/diegob/

Aula 011

- **Aula Anterior:**
  - **Aprendizagem Supervisionada – Support Vector Machine**
- **Aula de Hoje:**
  - **Aprendizagem Supervisionada – Redes Neurais Artificiais**

Objetivo

O que vocês devem saber ao final da aula:

Conceitos básicos de RNA.

Formas de Aprendizado

Aprendizado Supervisionado

- Árvores de Decisão.
- K-Nearest Neighbor (KNN).
- Support Vector Machines (SVM).
- **Redes Neurais.**

Aprendizado Não Supervisionado

Aprendizado Por Reforço

Redes Neurais Artificiais

- **Proposta Inicial meados de 1940**
- **Perceptron (1 camada de pesos ajustáveis)**
- **Descrédito a partir do final da década 60**
- **Impulso a partir da década de 80**

Computador x cérebro

Velocidade de processamento

CPU: Clock

Neural: disparo neurônio ms

Ordem de processamento

CPU: serial

Neural: paralelo

Abundância e complexidade:

CPU: um ou poucos processadores

Neural: 10^{11}-10^{14} neurônios; 10^{3}-10^{4} conexões/neurônio

Computador x cérebro

Armazenamento do conhecimento

CPU: estritamente realocável

Neural: adaptativo

Tolerância a falhas

CPU: mínima, senão inexistente

Neural: boa

Controle do processamento

CPU: controle autocrático, centralizado

Neural: controle anárquico, distribuído

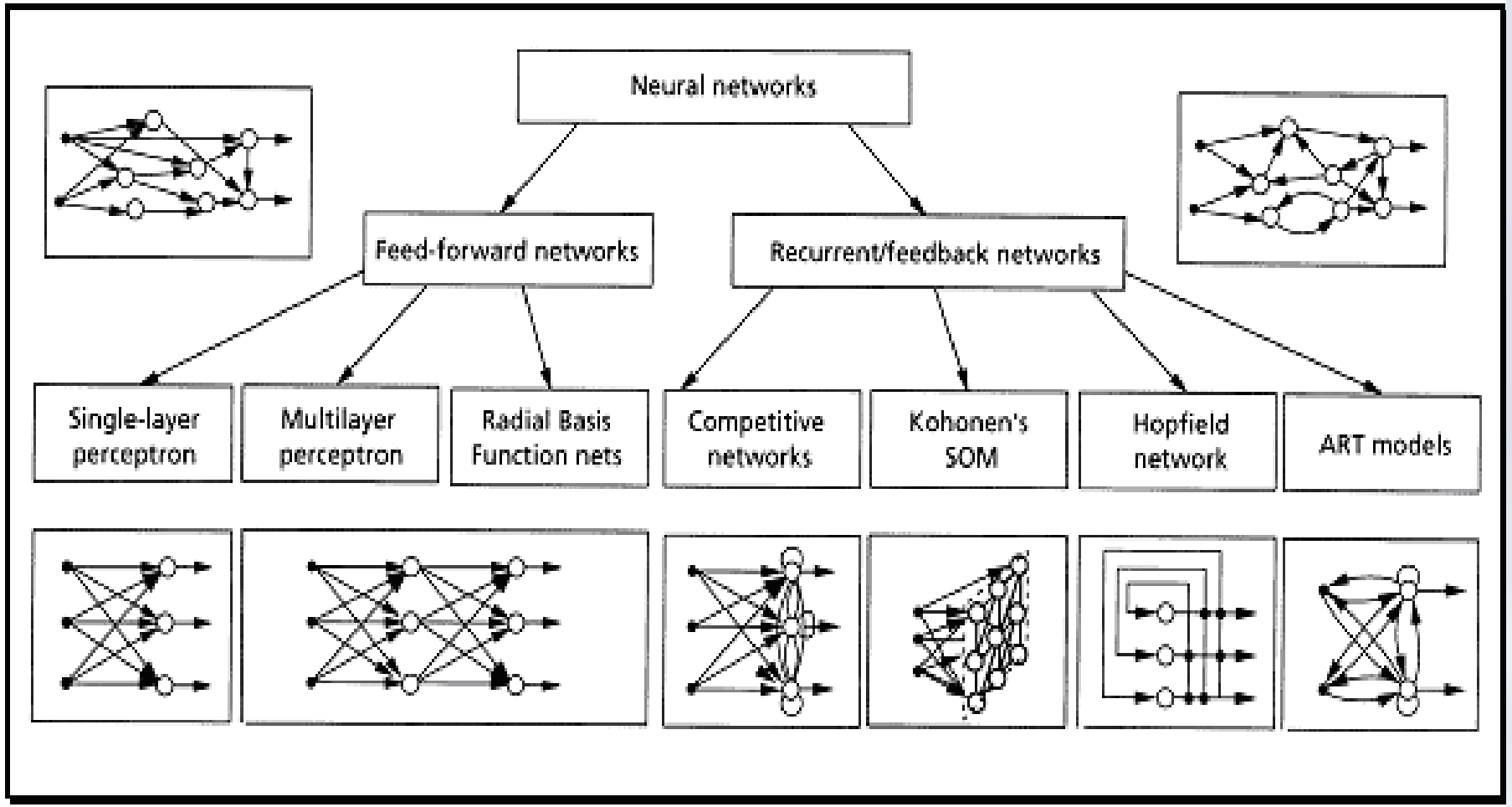
Forma Matemática Simplificada

- **Dendritos: entradas**
- **Corpo celular:**
  - Soma ponderada
  - Função não-linear
- **Axônio:** distribuição aos neurônios

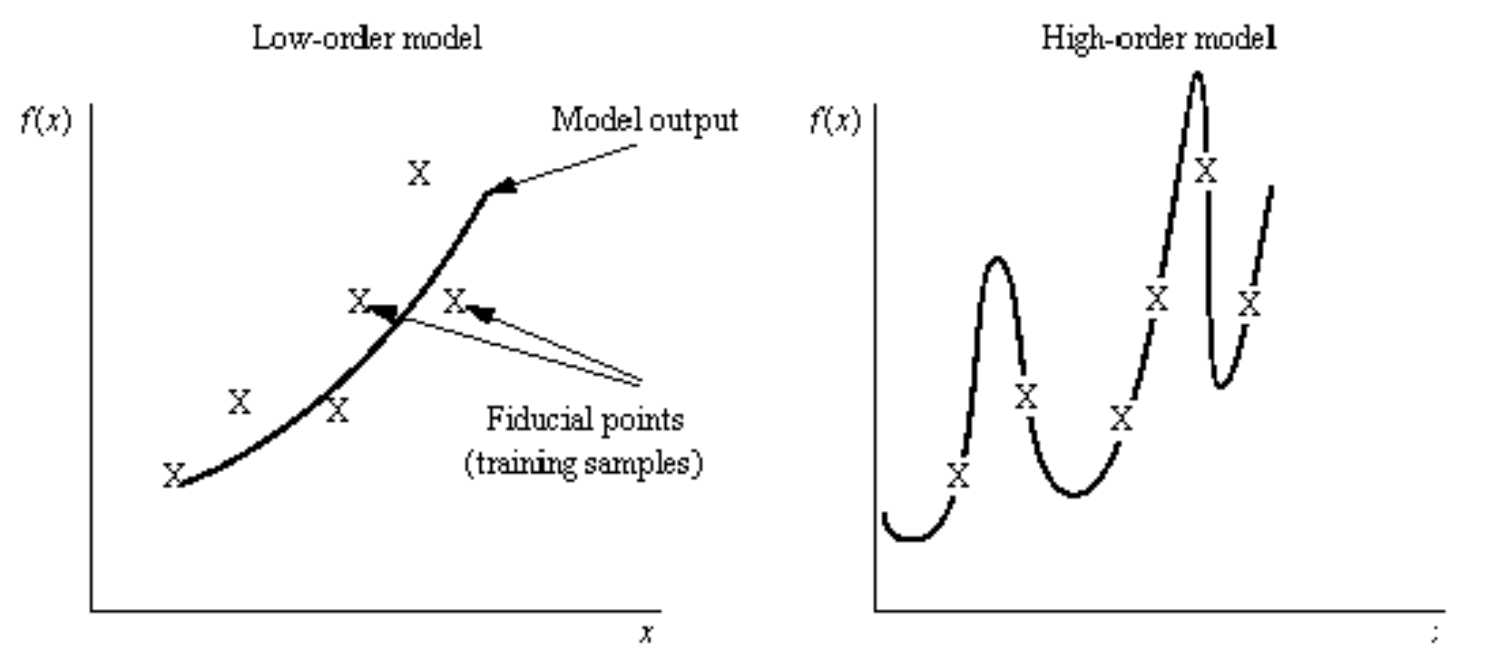
Etapas de Projeto

- Definir o problema;
- Escolher informação
  - Obter dados;
  - Criar arquivos rede;
- Treinar a rede;
- Testar a rede;
- Uso em campo;

Classificação das RNAs



Sub- e sobre-ajuste do polinômio



Implementações

- **Características-chave:**
  - Computacionalmente intensivas
  - Massivamente paralelas
  - Grandes requisitos de memória
- **Possibilidades de implementação**
  - Computadores convencionais
  - Computadores dedicados
  - Implementação em hardware específico

Redes Neurais Artificiais

- **Razões para utilização**
  - Paralelismo
  - Capacidade de adaptação
  - Memória distribuída
  - Capacidade de generalização
  - Facilidade de construção

Redes Neurais Artificiais

- **Características de uma boa aplicação**
  - Regras de resolução do problema desconhecidas ou difíceis de formalizar
  - Dispõe-se de um grande conjunto de exemplos e suas soluções
  - Necessita-se de grande rapidez na resolução do problema, p.ex. Tempo real
  - Não existem soluções tecnológicas atuais

Redes Neurais Artificiais

- **Domínios de aplicação privilegiados**
  - Reconhecimento de formas
  - Tratamento de sinal
  - Visão, fala
  - Previsão e modelagem
  - Auxílio à decisão
  - Robótica

Introdução

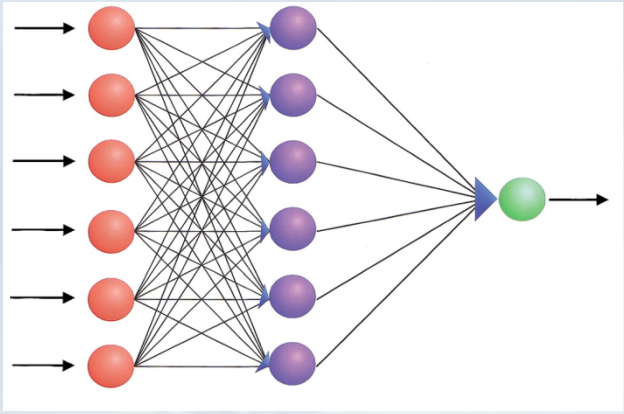
Redes Neurais podem ser consideradas um paradigma diferente de computação.

Inspirado na arquitetura paralela do cérebro humano.

Elementos de processamento simples.

Grande grau de interconexões.

Interação adaptativa entre os elementos.



Introdução

- **No cérebro, o comportamento inteligente é uma propriedade emergente de um grande número de unidades simples (ao contrário do que acontece com regras e algoritmos simbólicos).**

- **Neurônios ligam e desligam em alguns milissegundos, enquanto o hardware atual faz o mesmo em nano segundos.**

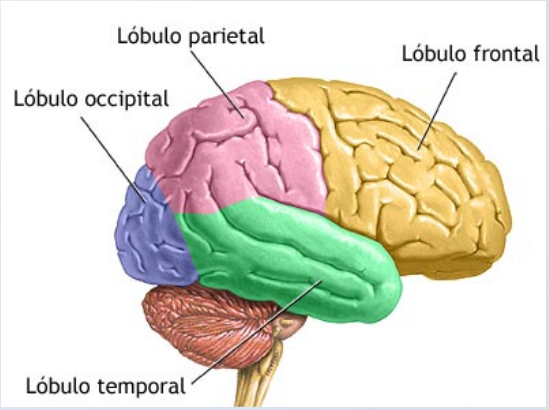
  - Entretanto, o cérebro realiza tarefas cognitivas complexas (visão, reconhecimento de voz) em décimos de segundo.

- **O cérebro deve estar utilizando um paralelismo massivo.**

Introdução

- **O cérebro humano tem sido extensamente estudado, mas ainda não somos capazes de entender completamente o seu funcionando.**

- **O cérebro é muito complexo, até mesmo o comportamento de um simples neurônio é bem complexo.**



Neurônio

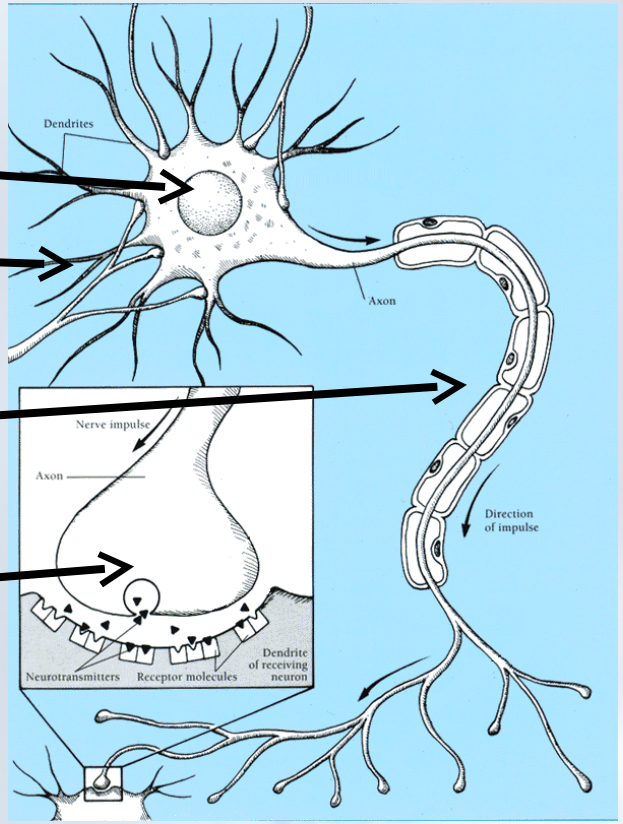
Estrutura de um Neurônio:

Corpo celular

Dendritos

Axônio

Terminais sinápticos



Funcionamento de um Neurônio

- **Através dos dentritos, o neurônio recebe sinais de outros neurônios a ele conectados por meio das sinapses.**
- **Os sinais são acumulados no corpo do neurônio.**
- **Quando a soma dos sinais passa de um certo limiar (~ 50mV) um sinal é propagado no axônio.**
- **As sinapses tem um peso que pode ser:**
  - excitatório: incrementam a soma dos sinais.
  - inibidor: decrementam.

Introdução

Características do Cérebro Humano:

- 10^{11} neurônios.
- Cada neurônio tem em media 10^{4} conexões.
- Milhares de operações por segundo.
- Neurônios morrem frequentemente e nunca são substituídos.
- Reconhecimento de faces em aproximadamente 0.1 segundos.

Introdução

- **O cérebro humano é bom em:**
  - Reconhecer padrões,
  - Associação,
  - Tolerar ruídos...
- **O computador é bom em:**
  - Cálculos,
  - Precisão,
  - Lógica.

Introdução

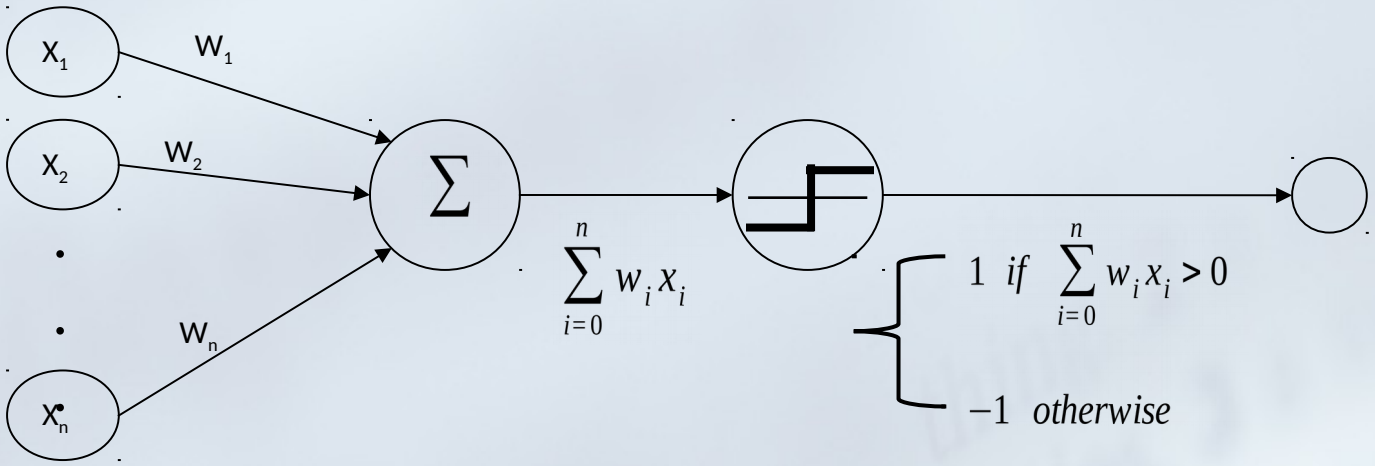
- **Formas mais básicas de aprendizado em Redes Neurais:**
  - **Perceptron**: Algoritmo para aprendizagem de redes neurais simples (uma camada) desenvolvido nos anos 50.
  - **Backpropagation**: Algoritmo mais complexo para aprendizagem de redes neurais de múltiplas camadas desenvolvido nos anos 80.

Aprendizagem de Perceptron

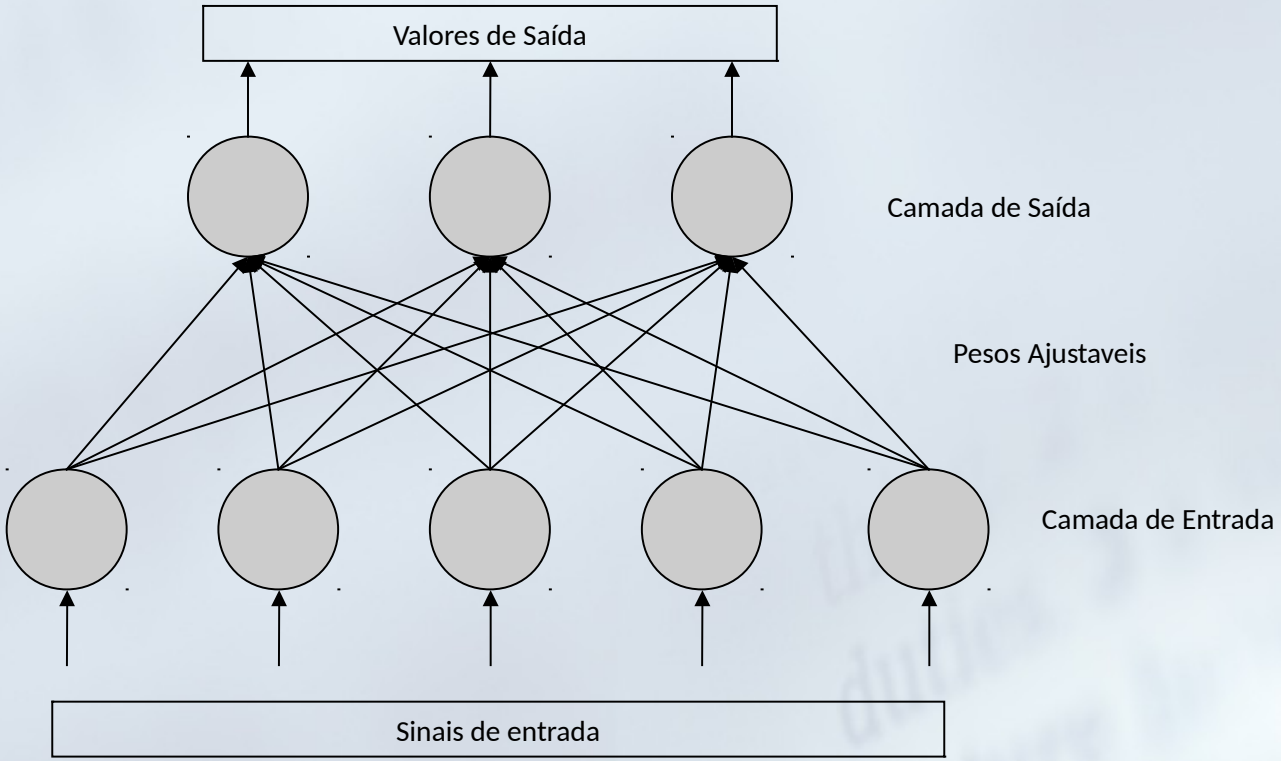
- **Usa-se um conjunto de exemplos de treinamento que dão a saída desejada para uma unidade, dado um conjunto de entradas.**
- **O objetivo é aprender pesos sinápticos de tal forma que a unidade de saída produza a saída correta pra cada exemplo.**
- **O algoritmo faz atualizações iterativamente até chegar aos pesos corretos.**

Perceptron

Unidade de Threshold Linear



Rede de Perceptrons



Aprendizado de Perceptrons

- **Para que um perceptron possa aprender uma função deve-se mudar o valor dos pesos ajustáveis por um quantidade proporcional a diferença entre a saída desejada e atual saída do sistema.**

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

				Saída desejada: t
x_1	x_2	...	x_n	o
x_1	x_2	...	x_n	t

***t* = saída desejada.**

***o* = atual saída do perceptron.**

$\sum$ = Learning rate.

Aprendizado de Perceptrons

- **Regra de aprendizado:**

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

- **Se a saída do perceptron não estiver correta (*t* != *o*):**
  - Os pesos w_i são alterados de forma que a saída do perceptron para os novos pesos seja próxima de *t*.

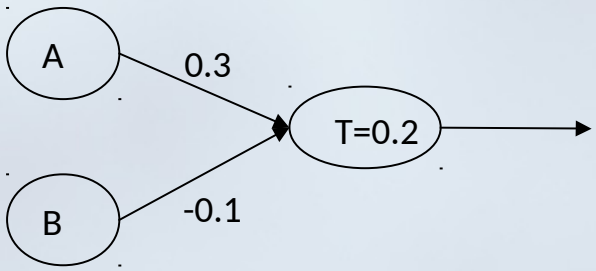
- **O algoritmo vai convergir para a correta classificação se:**
  - O conjunto de treinamento é linearmente separável.
  - η é suficientemente pequeno.

Treinando um Neurônio

Operador And

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Threshold = 0.2
Learning Rate = 0.1



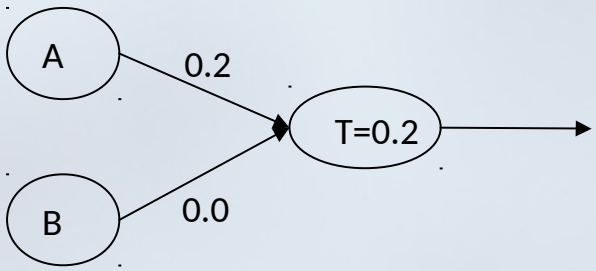
A	B	Somatório	Saída	Erro
0	0	(0*0.3)+(0*-0.1) = 0	0	0
0	1	(0*0.3)+(1*-0.1) = -0.1	0	0
1	0	(1*0.3)+(0*-0.1) = 0.3	1	-1
1	1	(1*0.3)+(1*-0.1) = 0.2	1	0

Treinando um Neurônio

Operador And

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Threshold = 0.2
Learning Rate = 0.1



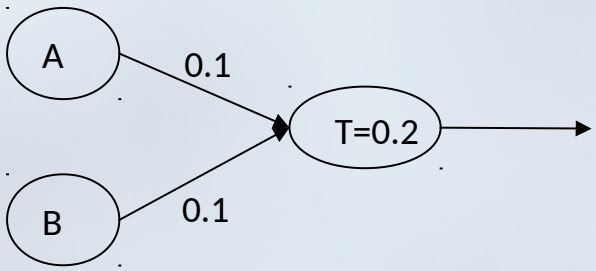
A	B	Somatório	Saída	Erro
0	0	(0*0.2)+(0*0.0) = 0	0	0
0	1	(0*0.2)+(1*0.0) = 0	0	0
1	0	(1*0.2)+(0*0.0) = 0.2	1	-1
1	1	(1*0.2)+(1*0.0) = 0.2	1	0

Treinando um Neurônio

Operador And

A	B	Saída
0	0	0
0	1	0
1	0	0
1	1	1

Threshold = 0.2
Learning Rate = 0.1

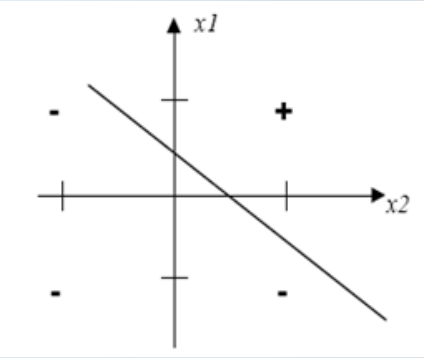


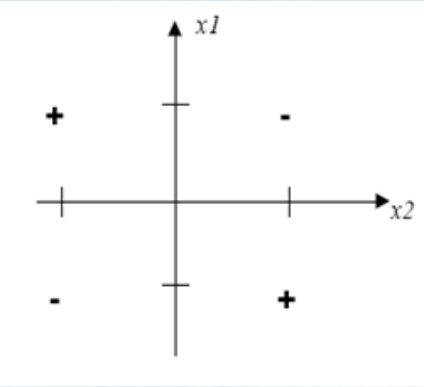
A	B	Somatório	Saída	Erro
0	0	(0*0.1)+(0*0.1) = 0	0	0
0	1	(0*0.1)+(1*0.1) = 0.1	0	0
1	0	(1*0.1)+(0*0.1) = 0.1	0	0
1	1	(1*0.1)+(1*0.1) = 0.2	1	0

Limitações

- **Um único Perceptron consegue resolver somente funções linearmente separáveis.**

- **Em funções não linearmente separáveis o perceptron não consegue gerar um hiperplano para separar os dados.**





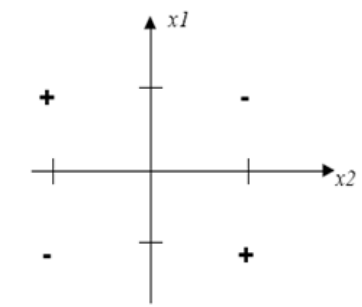
Redes Multicamadas

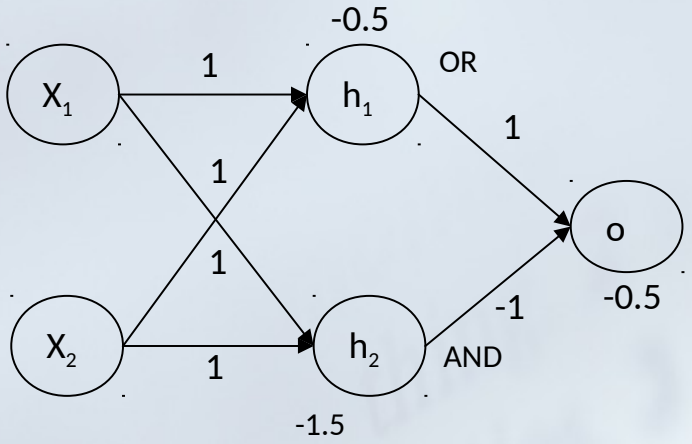
- Perceptrons expressam somente superfícies de decisão linear.
- Entretanto, é possível combinar vários perceptrons lineares para gerar superfícies de decisão mais complexas.
- Dessa forma podemos, por exemplo, gerar uma superfícies de classificação para o operador XOR.

Operador XOR

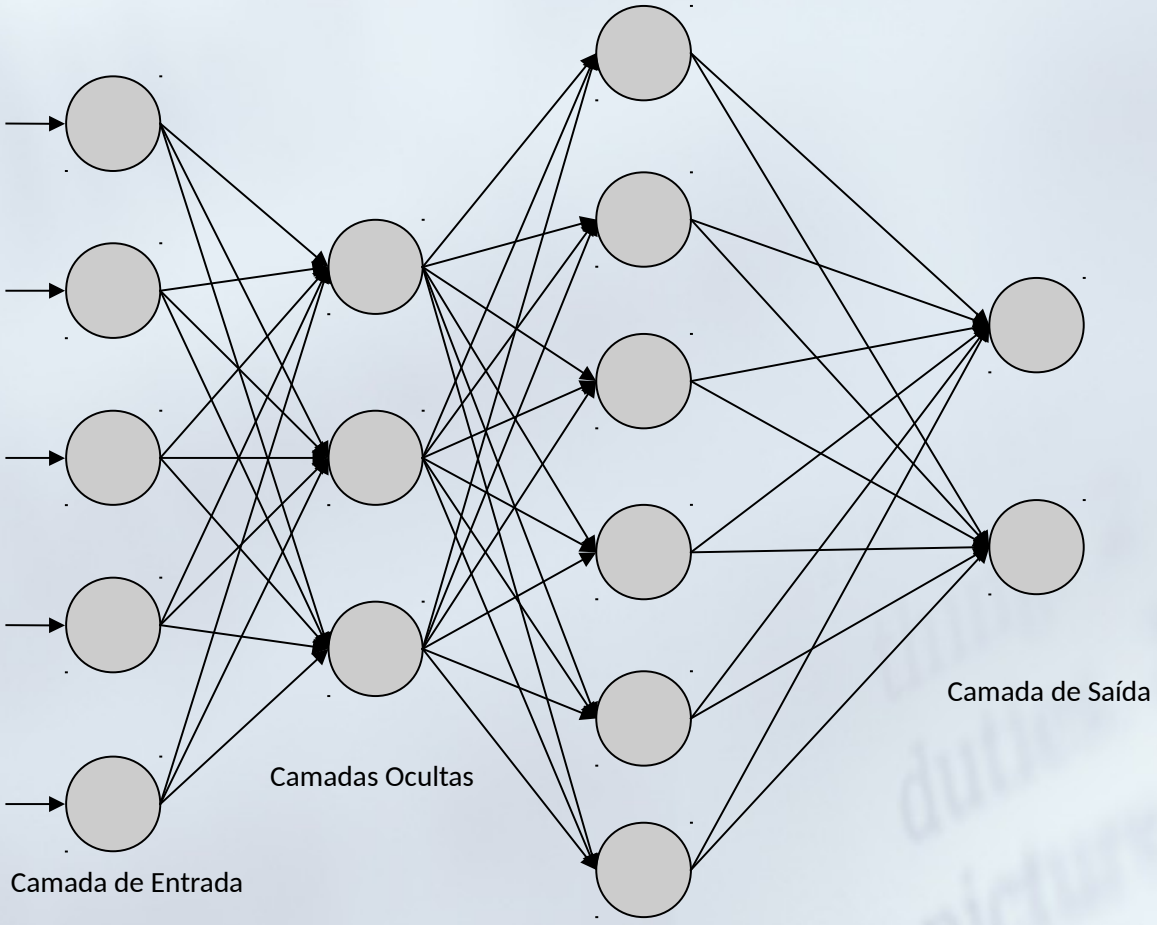
Operador XOR

A	B	Saída
0	0	0
0	1	1
1	0	1
1	1	0



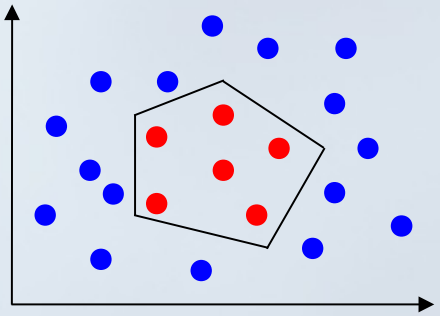


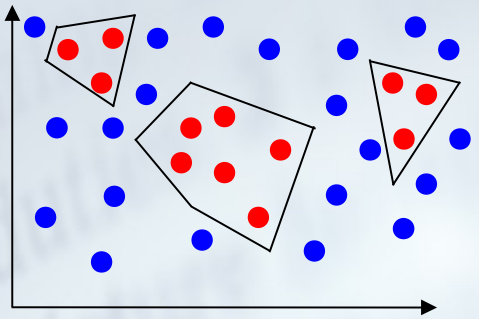
Redes Multicamadas



Redes Multicamadas

- Adicionar uma camada oculta a rede permite que a rede possa gerar uma função de convex hull.
- Duas camadas ocultas permite a rede gerar um função com diferentes convex hulls.





Redes Multicamadas

- Unidades lineares são capazes gerar funções lineares, dessa forma função de uma rede multicamada também será linear.

- Entretanto, existem muitas funções que não podem ser modeladas por funções lineares.

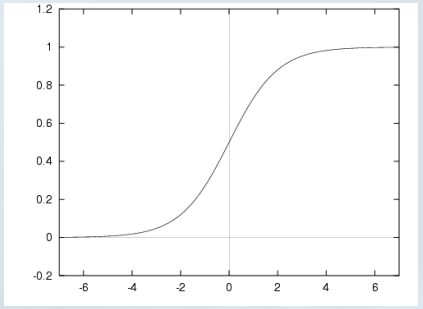
- Por esse motivo é necessário utilizar uma outra função de ativação.

Redes Multicamadas

Funções de ativação mais comuns:

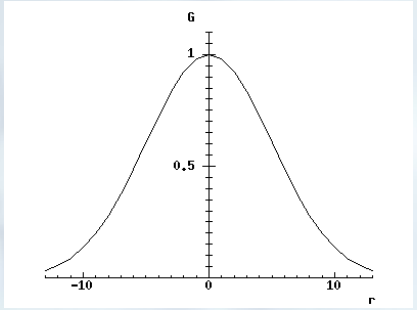
Sigmoidal:

$$y = f\left(h = w_0 \cdot 1 + \sum_{i=1}^{n} w_i \cdot x_i ; p\right) = \frac{1}{1 + e^{-hp}}$$



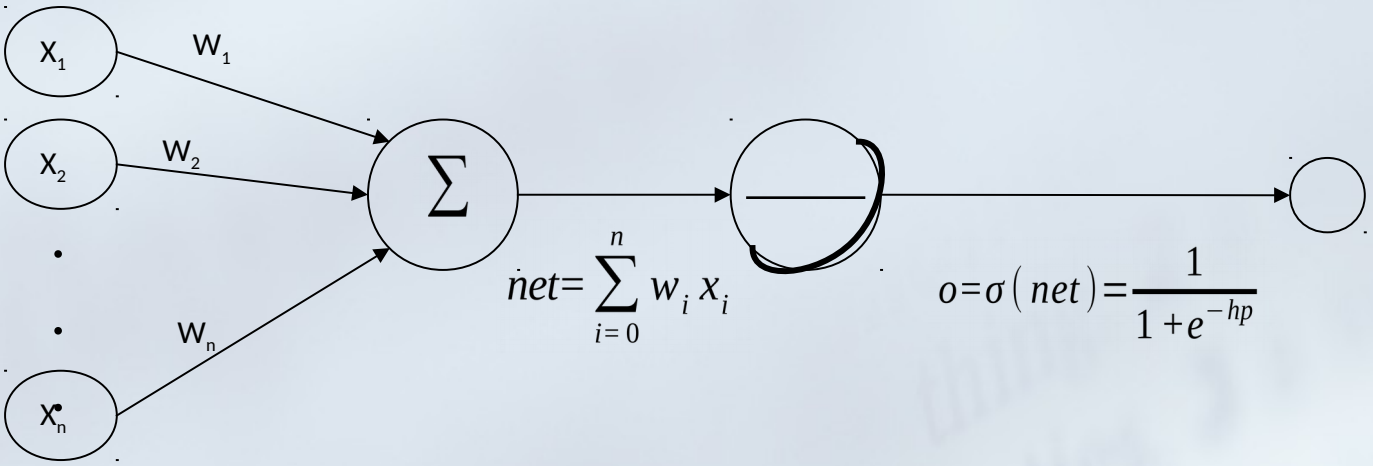
Radial (Gausiana):

$$y = f\left(h = \sum_{i=1}^{n} (x_i \cdot w_i)^2 ; \sigma = w_0\right) = \frac{1}{2\pi\sigma} e^{-\frac{h^2}{2\sigma^2}}$$



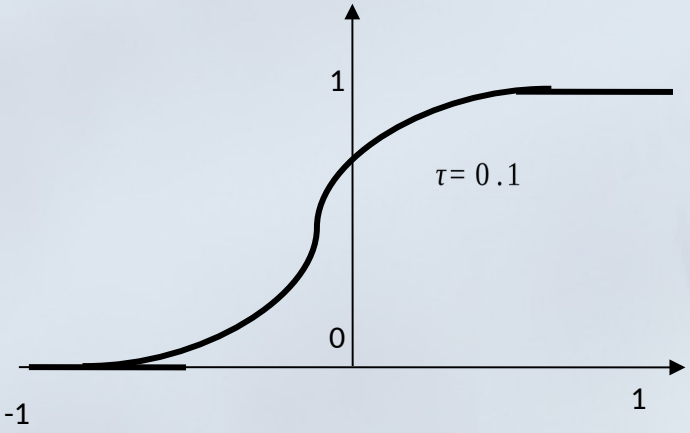
Redes Multicamadas

Unidade Sigmoid



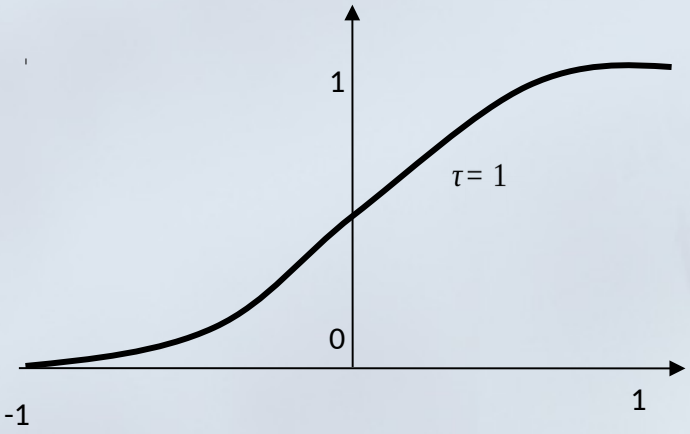
Função Sigmoidal

$$f_i(net_i(t)) = \frac{1}{1 + e^{-(net_i(t) - \alpha)/\tau}}$$



Função Sigmoidal

$$f_i(net_i(t)) = \frac{1}{1 + e^{-(net_i(t) - \alpha)/\tau}}$$



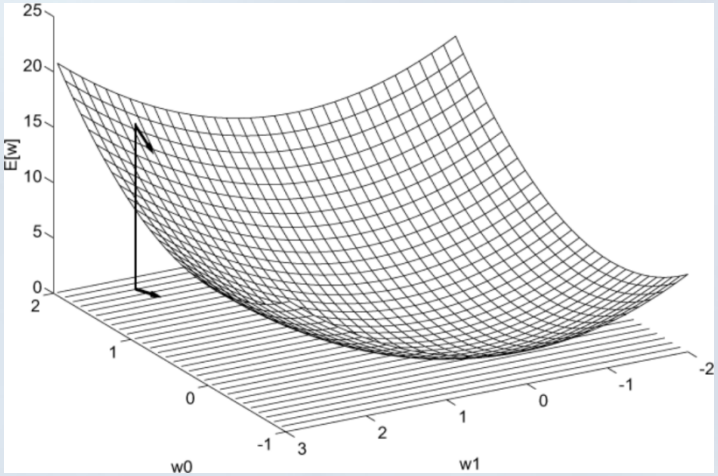
Backpropagation

- Aprende os pesos para uma rede multicamadas, dada uma rede com um número fixo de unidades e interconexões.

- O algoritmo backpropagation emprega a descida do gradiente para minimizar o erro quadrático entre a saída da rede e os valores alvos para estas saídas.

Descida do Gradiente

- A **descida do gradiente** busca determinar um vetor de pesos que minimiza o erro.



- Começando com um vetor inicial de **pesos arbitrário** e modificando-o repetidamente em pequenos passos.

- A cada passo, o vetor de pesos é alterado na direção que produz a maior queda ao longo da superfície de erro.

Backpropagation

- Aprende os pesos para uma rede multicamadas, dada uma rede com um número fixo de unidades e interconexões.

- O algoritmo backpropagation emprega a descida do gradiente para minimizar o erro quadrático entre a saída da rede e os valores alvos para estas saídas.

Valor de saida da rede →

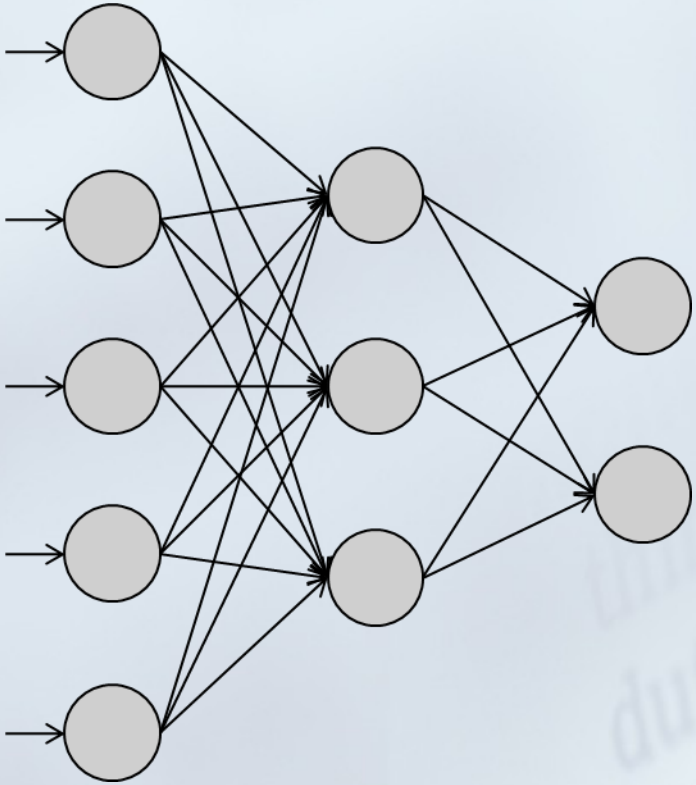
0.119 0.059 0.253 0.246

0 0 1 0

← Valor da saida buscada

Erro (E) = (Valor da saida buscada) – (Valor de saida da rede)

Backpropagation



Backpropagation

Inicializa cada peso w_i com um pequeno valor randômico.

Enquanto condição de parada não for atingida **faça**

{

Para cada exemplo de treinamento **faça**

{

Entre com os dados do exemplo na rede e calcule a saída da rede (o_k)

Para cada unidade de saída k **faça**

{

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

}

Para cada unidade oculta h **faça**

{

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{h,k}\delta_k$$

}

Para cada peso w_j da rede **faça**

{

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

$$where \quad \Delta w_{i,j} = \eta \delta_j x_{i,j}$$

}

}

}

Backpropagation

- O backpropagation não é um algoritmo ótimo e não garante sempre a melhor resposta.
- O algoritmo de descida do gradiente pode ficar preso em um erro mínimo local.
- É possível refazer o treinamento variando os valores iniciais dos pesos.
- Backpropagation é o algoritmo de aprendizagem mais comum, porém existem muitos outros.

Backpropagation

- **Minimiza o erro sobre os exemplos de treinamento**
  - Generalizará bem sobre exemplos subseqüentes?
  - O treinamento pode levar milhares de iterações → vagaroso
  - A utilização da rede após o treinamento → muito rápida

Exemplo

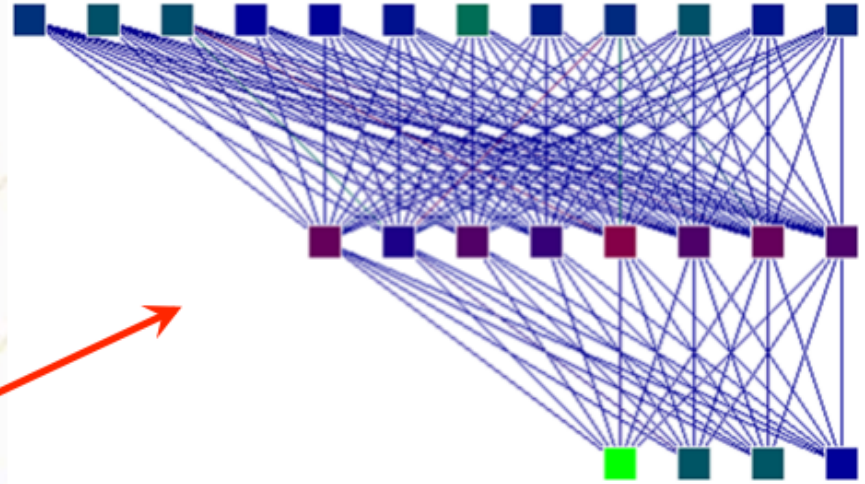
- Dada a imagem de um personagem, ele deve ser classificado corretamente, ou seja, se a imagem for do personagem Bart, ela deve ser classificada pelo algoritmo de aprendizagem como sendo o personagem Bart.

Exemplo

vetor de características

0.43 0.03 0.40 0.19 0.12 0.16 0.04 0.01 0.00 0.01 0.40 0.02
0 0 1 0

valor do conceito alvo associado ao vetor



rede neural treinada

0.119 0.059 0.253 0.569

valor do conceito alvo estimado

Erro = (valor do conceito alvo real) – (valor do conceito alvo estimado)

Condiçao de Parada

A condição de parada do algoritmo backpropagation foi deixada em aberto.

- Quando devemos parar o treinamento, i.e. parar de atualizar os pesos?

– Escolha óbvia: continuar o treinamento até que o erro (E) seja menor do que um valor pré-estabelecido.

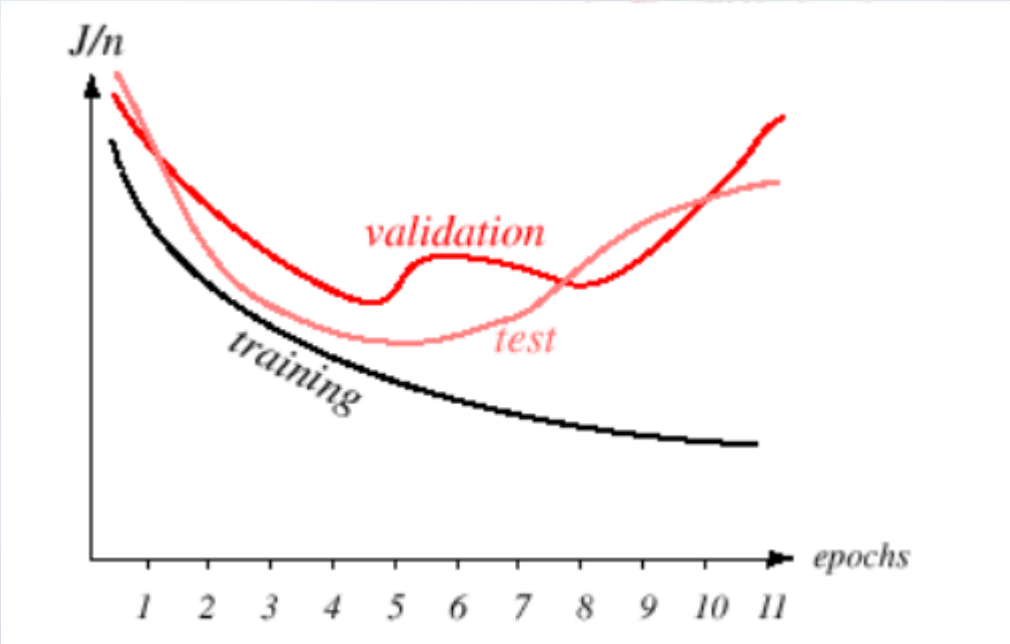
– Porém, isto implica em sobre ajuste (overfitting) !!!

Condiçao de Parada

O algoritmo backpropagation é susceptível a sobre ajustar a rede aos exemplos de treinamento ao preço de reduzir a generalização sobre exemplos novos.

- A Figura a seguir ilustra o perigo de minimizar o erro sobre os dados de treinamento em função do número de iterações (atualização dos pesos).

Condiçao de Parada



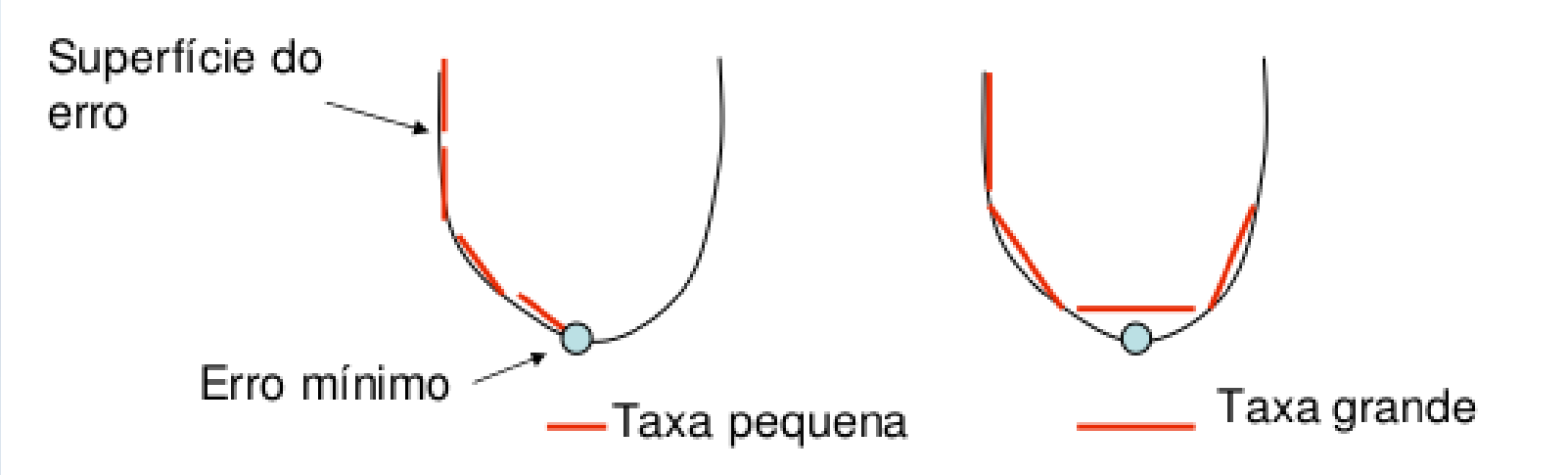
Aspectos Práticos

- **Alguns aspectos práticos devem ser considerados na utilização de redes neurais MLP.**
  - Taxa de aprendizagem
  - Momentum
  - Online vs batch
  - Shuffle
  - Normalização
  - Inicialização dos pesos
  - Generalização

Taxa de Aprendizagem

Taxas muito pequenas tornam o processo bastante lento.

- **Taxas muito grandes tornam o processo rápido.**
  - **Podem não trazer os resultados ideais.**



Taxa de Aprendizagem

- **O ideal é começar com uma taxa grande e reduzir durante as iterações.**
- **Permite a exploração global no início (exploration) a local (exploitation) quando o algoritmo estiver próximo do ótimo global.**
- **Geralmente valores entre 0.05 e 0.75 fornecem bons resultados.**

Momentum

- **É uma estratégia usada para evitar mínimos locais.**
- **Normalmente $0 <= \alpha <= 0.9$;**

Shuffle

- **Redes neurais aprendem melhor quando diferentes exemplos de diferentes classes são apresentados a rede.**
  - Uma prática muito comum consiste em apresentar um exemplo de cada classe a rede
  - Isso garante que os pesos serão atualizados levando-se em consideração todas as classes.

Shuffle

- **Se apresentarmos à rede todos os exemplos de uma classe, e assim por diante, os pesos finais tenderão para a última classe**
  - A rede vai “esquecer” o que ela aprendeu antes.

Normalização

- **A normalização é interessante quando existem características em diversas unidades dentro do vetor de características.**
  - Nesses casos, valores muito altos podem saturar a função de ativação.
  - Uma maneira bastante simples de normalizar os dados consiste em somar todas as características e dividir pela soma
  - Outra normalização bastante usada é a normalização Z-score.

Generalização

- **Um aspecto bastante importante quando treinamos um classificador é garantir que o mesmo tenha um bom poder de generalização.**
  - Evitar overfitting.
  - A maneira clássica de se garantir uma boa generalização consiste em reservar uma parte da base para validar a generalização.

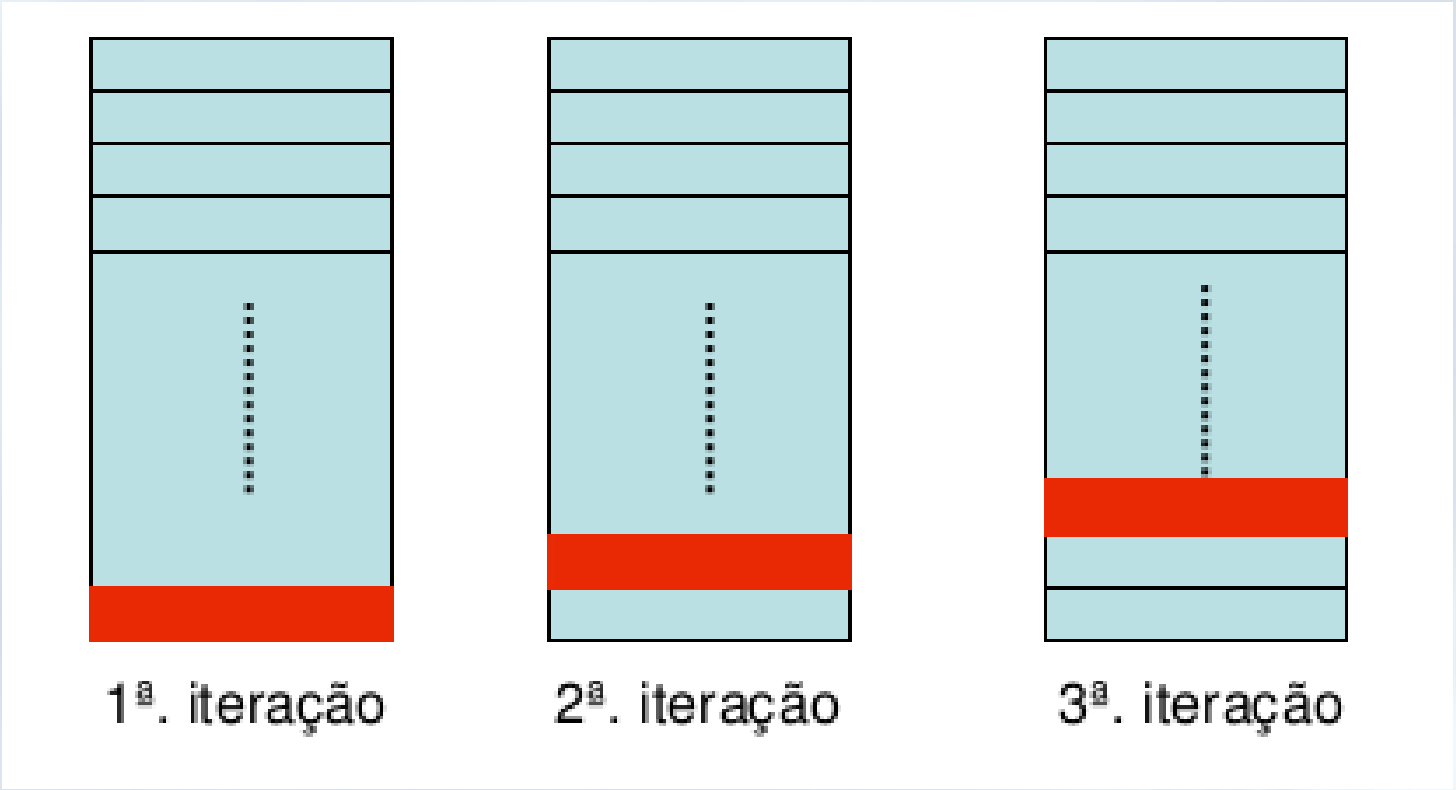
Generalização

- **Uma outra estratégia é a validação cruzada.**
  - Interessante quando a base não é muito grande
  - Separar alguns exemplos para validação pode prejudicar o treinamento.
  - Consiste em dividir a base de aprendizagem em n partições iguais e usar n-1 partições para aprendizagem e uma partição para validação.
- **A cada iteração a partição usada para validação é trocada.**

Tamanho da Rede

- **Geralmente uma camada escondida é suficiente.**
  - Em poucos casos você vai precisar adicionar uma segunda camada escondida.
  - Não existe uma formula matemática para se encontrar o número de neurônios.
    - Empírico
- **Dica prática**
  - Comece com uma rede menor, pois a aprendizagem vai ser mais rápida.

Generalização



Resumindo...

- **Redes Neurais: um método prático para aprendizagem de funções de valor real e vetorial sobre atributos de valor contínuo e discreto.**
- **Robustez a ruídos nos dados de treinamento.**
  - O espaço considerado pelo algoritmo backpropagation é o espaço de todas as funções que podem ser representadas pelos pesos.
  - O backpropagation busca o espaço de todos os pesos possíveis usando a descida do gradiente para reduzir iterativamente o erro em uma rede (ajustar aos dados de treinamento).

Resumindo...

- **Sobre ajuste resulta em redes que não generalizam bem. Utilizar um conjunto de validação para avaliar a generalização**
- **Backpropagation é o algoritmo de aprendizagem mais comum, porém existem muitos outros . . .**

Bibliografia e Materiais.

Estes slides foram adaptados do Livro:

Adaptado das Aulas do Professor Ederley – PUC-RIO

Livro: Duda, R., Hart, P., Stork, D., **Pattern Classification**, John Wiley & Sons, 2000