

Inteligência Artificial

Aula 12 – Redes Neurais

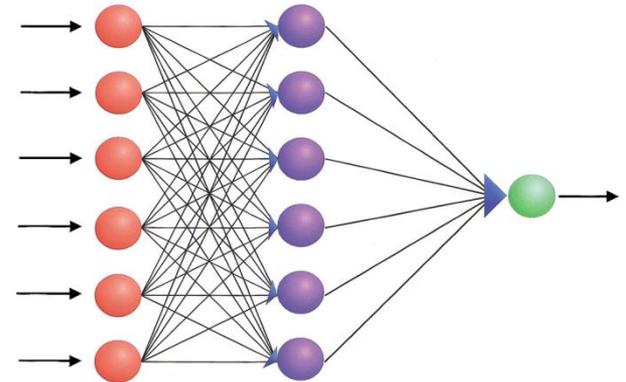
Tiago Alves de Oliveira
tiagofga@gmail.com

Formas de Aprendizado

- **Aprendizado Supervisionado**
 - Árvores de Decisão.
 - K-Nearest Neighbor (KNN).
 - Support Vector Machines (SVM).
 - **Redes Neurais.**
- Aprendizado Não Supervisionado
- Aprendizado Por Reforço

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.

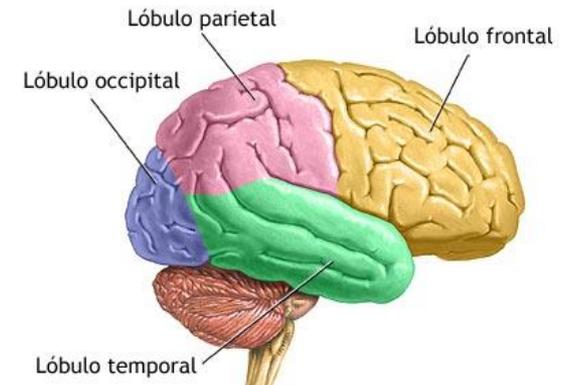


Introdução

- 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

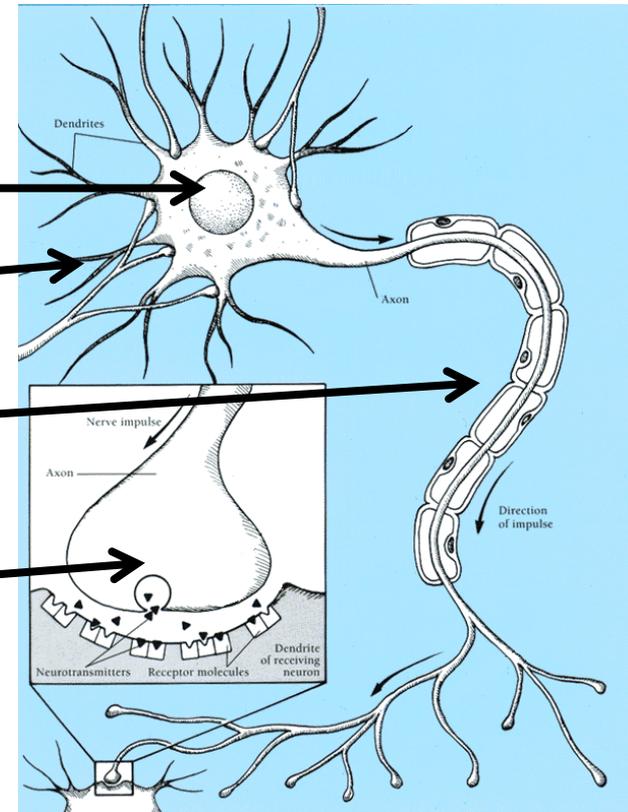
- O **cérebro humano** tem sido extensamente estudado, mas ainda não somos capazes de **entender completamente** o seu funcionamento.
- O cérebro é **muito complexo**, até mesmo o comportamento de um simples **neurônio** é bem complexo.



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 ($\sim 50\text{mV}$) 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

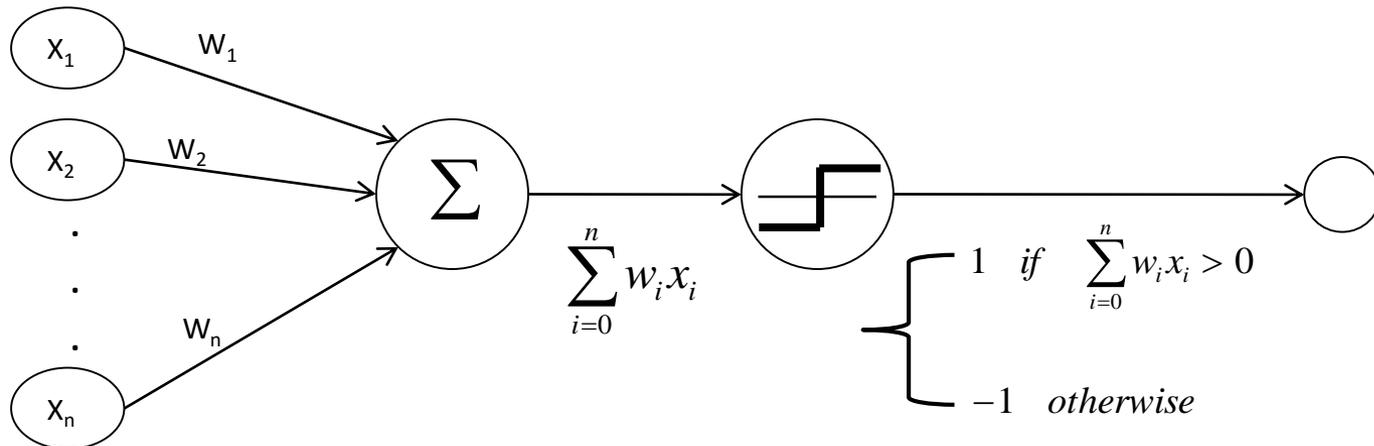
- Formas mais básicas de **aprendizado** em Redes Neurais:
 - **Perceptron**: Algoritmo para aprendizagem de redes neurais simples (uma camada) desenvolvido nos anos 50.
 - **Backpropagation**: Algoritmo mais complexo para aprendizagem de redes neurais de múltiplas camadas desenvolvido nos anos 80.

Aprendizagem de Perceptron

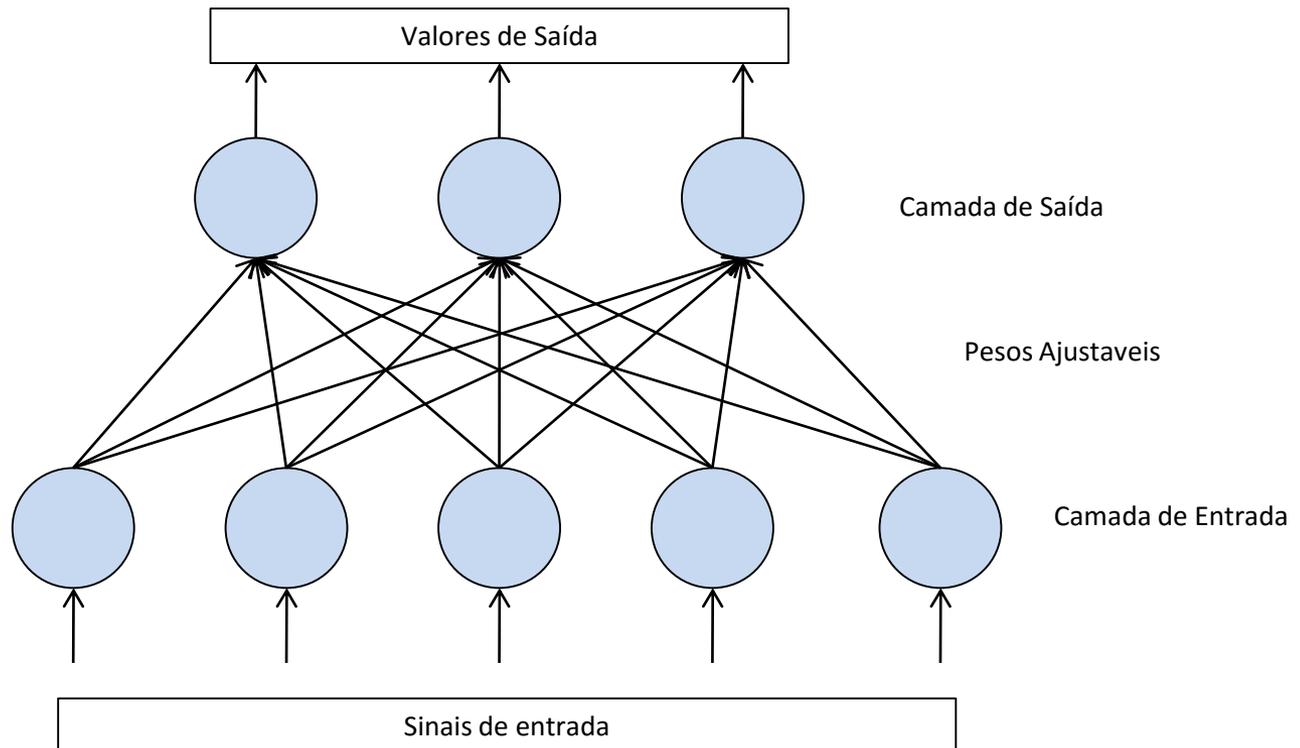
- Usa-se um conjunto de **exemplos de treinamento** que dão a saída desejada para uma unidade, dado um conjunto de entradas.
- O objetivo é **aprender pesos** sinápticos de tal forma que a unidade de saída produza a saída correta pra cada exemplo.
- O algoritmo faz atualizações iterativamente até chegar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

Aprendizado de Perceptrons

- Regra de aprendizado:

$$w_i = w_i + \Delta w_i$$

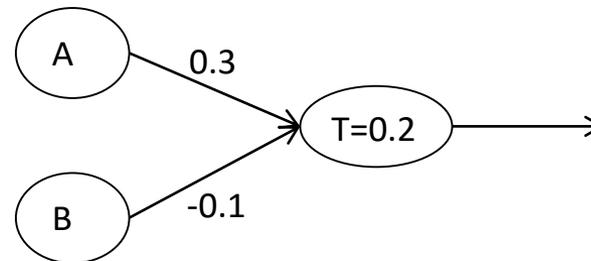
$$\Delta w_i = \eta(t - o)x_i$$

- Se a saída do perceptron não estiver correta ($t \neq o$):
 - 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

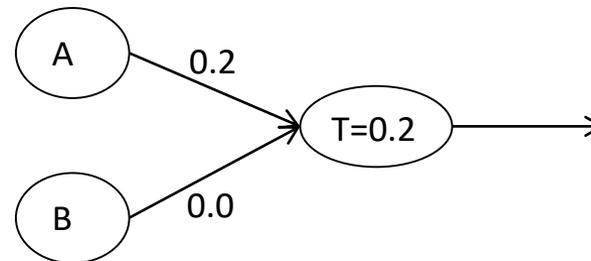
Learning Rate = 0.1

| 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

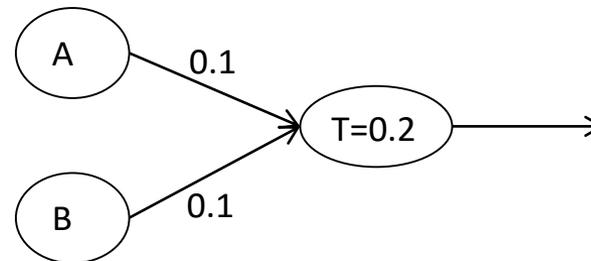
Learning Rate = 0.1

| 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 |



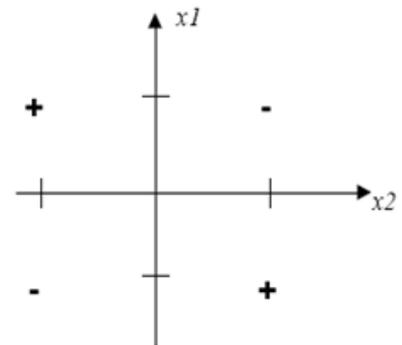
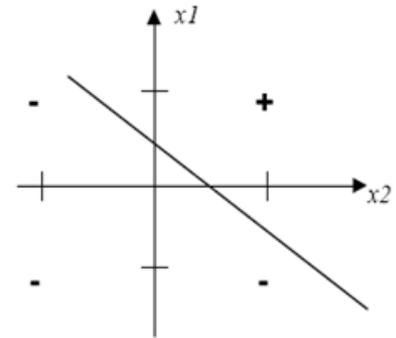
Threshold = 0.2

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.



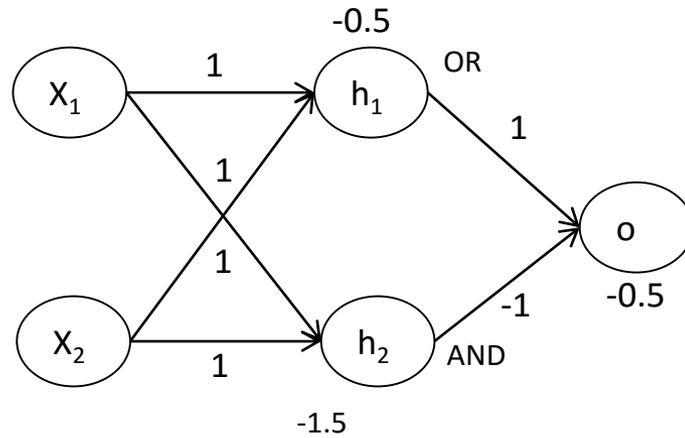
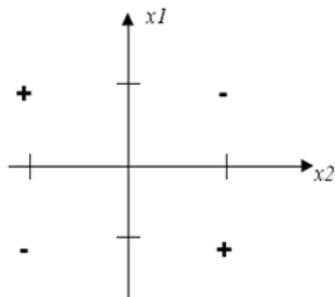
Redes Multicamadas

- 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.

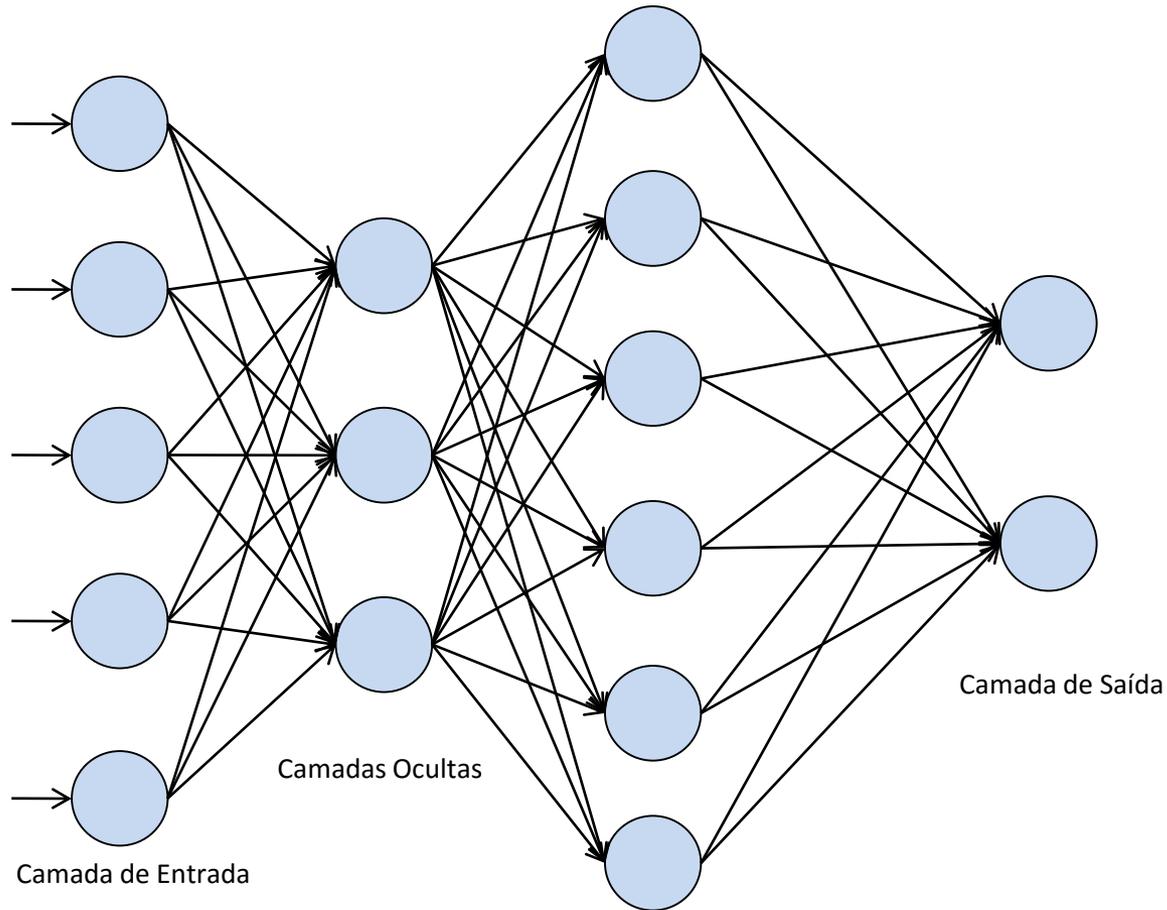
Operador XOR

Operador XOR

| A | B | Saída |
|---|---|-------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

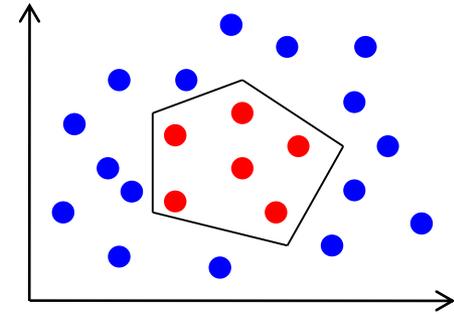


Redes Multicamadas

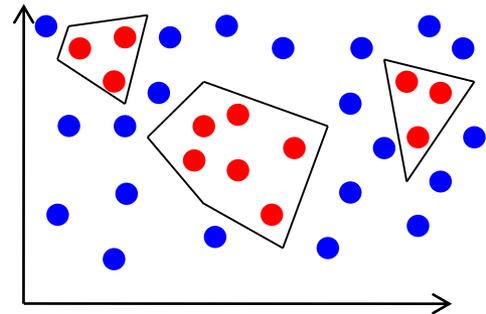


Redes Multicamadas

- 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

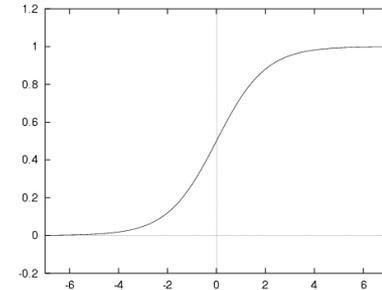
- **Unidades lineares** são capazes de gerar **funções lineares**, dessa forma a função de uma rede multicamada também será linear.
- Entretanto, existem muitas funções que **não podem ser modeladas por funções lineares**.
- Por esse motivo é necessário utilizar uma **outra função de ativação**.

Redes Multicamadas

- Funções de ativação mais comuns:

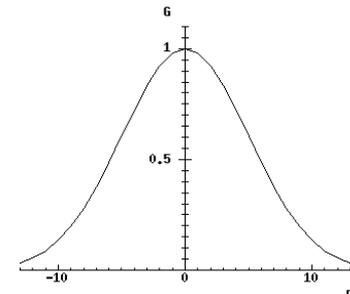
– Sigmoidal:

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



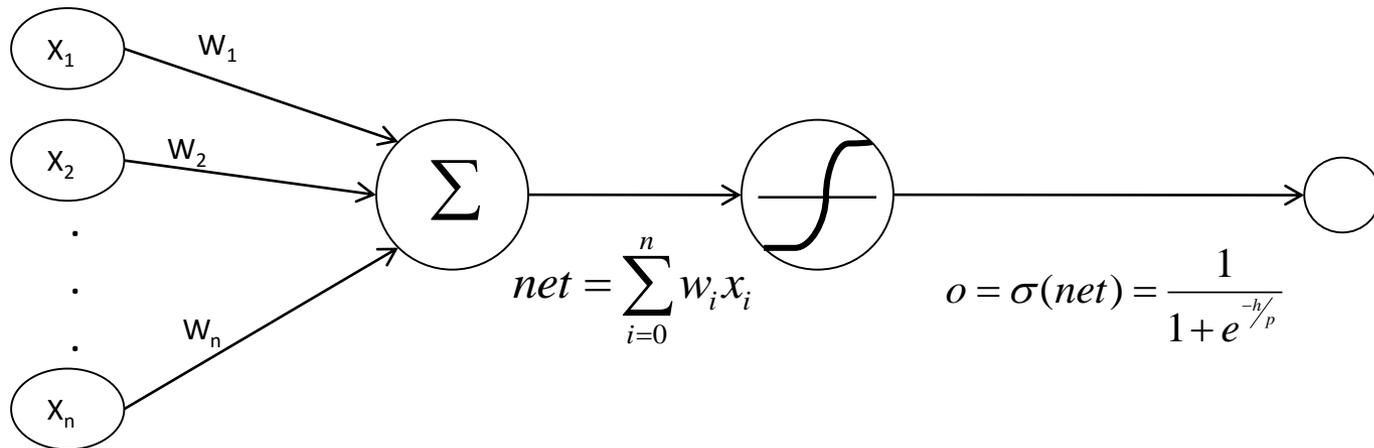
– Radial (Gausiana):

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



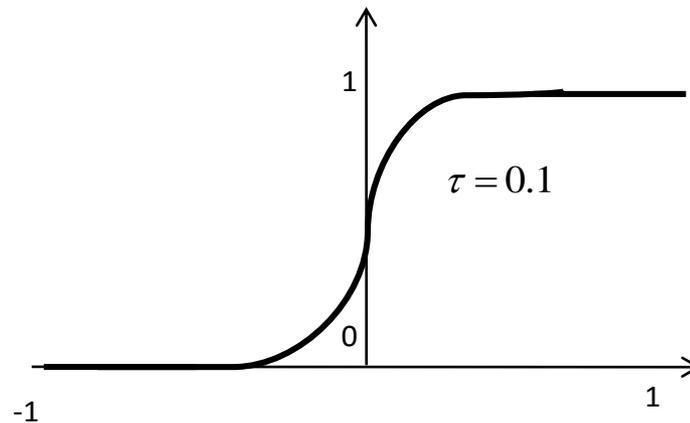
Redes Multicamadas

- **Unidade Sigmoid**



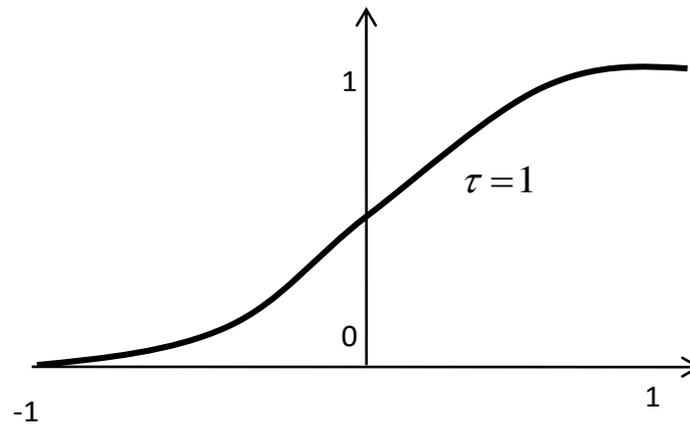
Função Sigmoidal

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



Função Sigmoidal

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

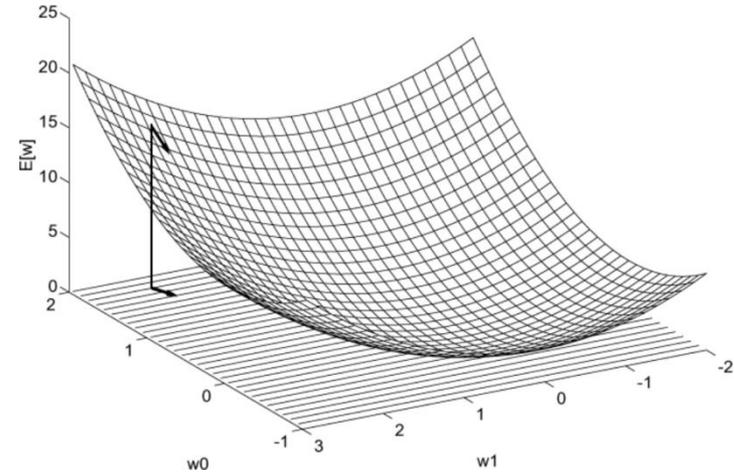


Backpropagation

- **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.

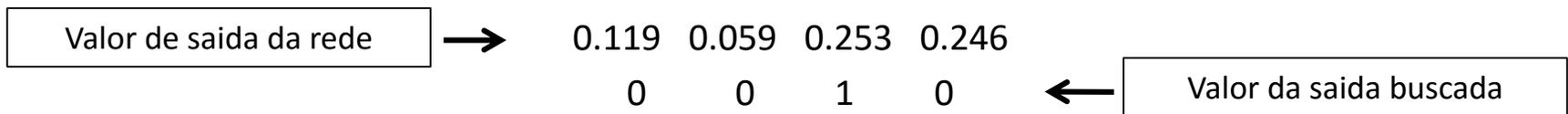
Descida do Gradiente

- 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.



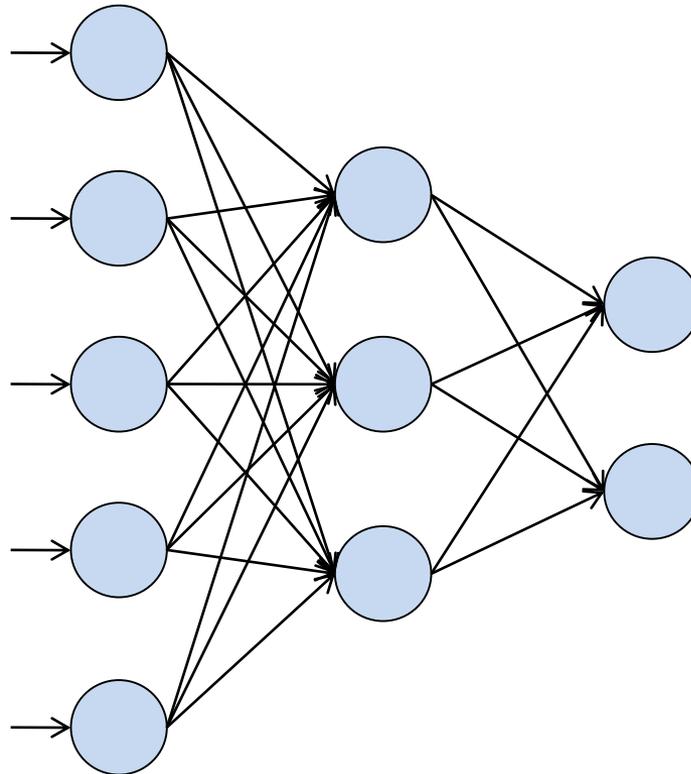
Backpropagation

- **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.



$$\text{Erro (E)} = (\text{Valor da saída buscada}) - (\text{Valor de saída da rede})$$

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**

 {

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

 }

Para cada peso w_j da rede **faça**

 {

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

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

 }

 }

}

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.

Leitura Complementar

- Mitchell, T. **Machine Learning**, McGraw–Hill Science/Engineering/Math, 1997.
- Duda, R., Hart, P., Stork, D., **Pattern Classification**, John Wiley & Sons, 2000

