Introdução a Redes Neurais Artificiais com a biblioteca Encog em Java

Apresentação

- Graduada em Sistemas de Informação – FAP/Parnaíba
- Mestranda do Programa de Pós-Graduação em Engenharia da Eletricidade - UFMA

Roteiro

- Introdução
- Redes Neurais Artificiais
- Neurônio Biológico
- Neurônio Artificial
- Funções de Ativação
- Estruturas das RNAs
- Aprendizado em RNAs
- Regras de Aprendizagem
- Rede Perceptron Simples
- A biblioteca Encog

Introdução

- O cérebro resolve problemas de maneira inata
- Desejo de construir artefatos de comportamento inteligente
- Dificuldades de tratar informações imprecisas e incompletas

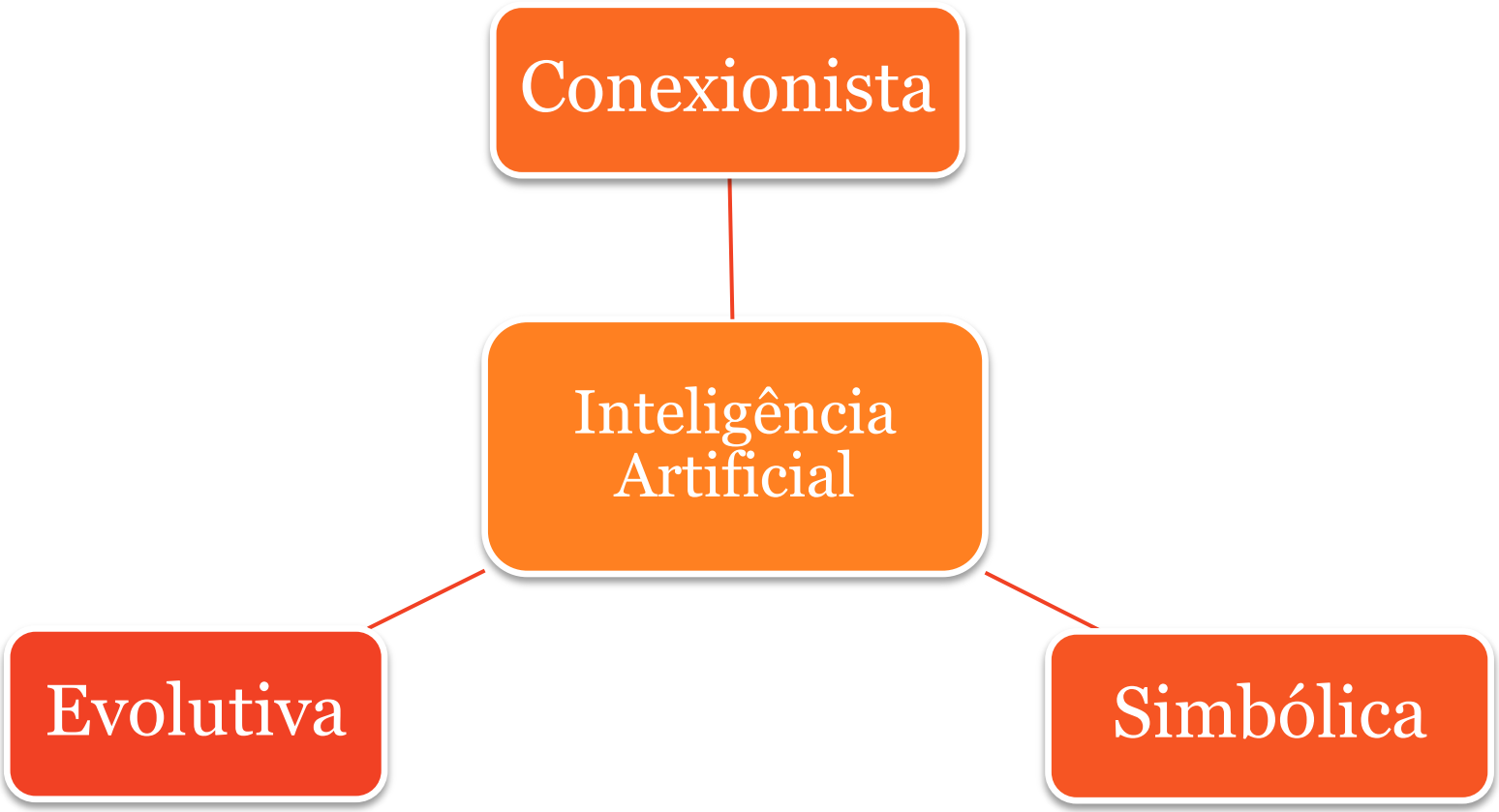
Redes Neurais Artificiais

"São modelos computacionais inspirados no sistema nervoso de seres vivos. Possuem a capacidade de aquisição e manutenção de conhecimento e podem ser definidas como um conjunto de unidades de processamento, que são interligados por um grande número de interconexões, sendo as mesmas representadas por vetores/matrizes de pesos sinápticos."Silva(2010)

Redes Neurais Artificiais

- Baseada no neurônio biológico
- Grande capacidade de aprendizado e generalização
- Processamento paralelo
- Técnica estatística não – linear
- Abordagem conexionista

Áreas relacionadas da IA



Principais tarefas realizadas pela RNAs

- Classificação
- Agrupamento
- Regressão numérica
- Predição
- Reconhecimento de Padrões

Interesse em usar RNAs

- Classificação e predição do câncer com base do perfil genético;
- Diagnóstico de doenças no coração;
- Sistemas de controle de tratamento da água;
- Previsão de ações do mercado financeiro;
- Classificação de fontes de corrente harmônico em sistemas de distribuição de energia;

Histórico das RNAs

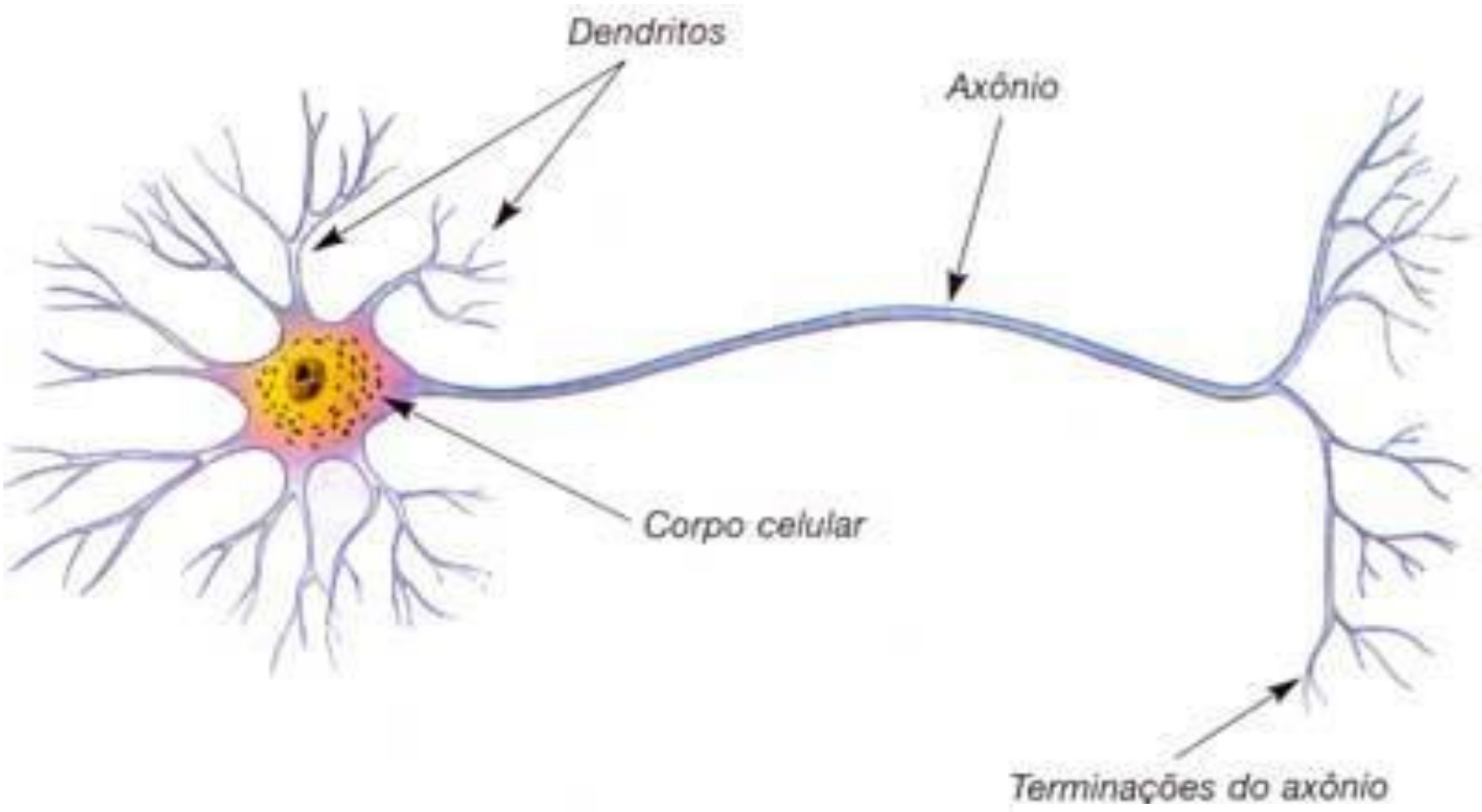
- Modelo neurônio artificial por McCulloch & Pitts (1943)
- Regra de aprendizado de Hebb (1949)
- O modelo Perceptron de Rosenblatt (1958)
- Descoberta da limitação do Perceptron por Minsky e Papert (1969)
- Abordagem de energia de Hopfield (1982)
- Algoritmo de aprendizagem Backpropagation para redes Perceptron Múltiplas Camadas por Rumelhart (1986)

Neurônio Biológico

Neurônio Biológico

- **Corpo celular ou Soma -** Produz um potencial de ativação que indicará se o neurônio poderá disparar um impulso elétrico ao longo do axônio.
- **Dendritos -** Responsáveis pela captação, de forma contínua, dos estímulos vindos de diversos outros neurônios.
- **Axônio -** Formado por um único prolongamento, têm por função conduzir os impulso elétricos para os outros neurônios conectores chegando até os dendritos.
- **Sinapses -** Que se configuram como as conexões que viabilizam a transferência de impulsos elétricos do axônio de um neurônio para os dendritos dos outros.

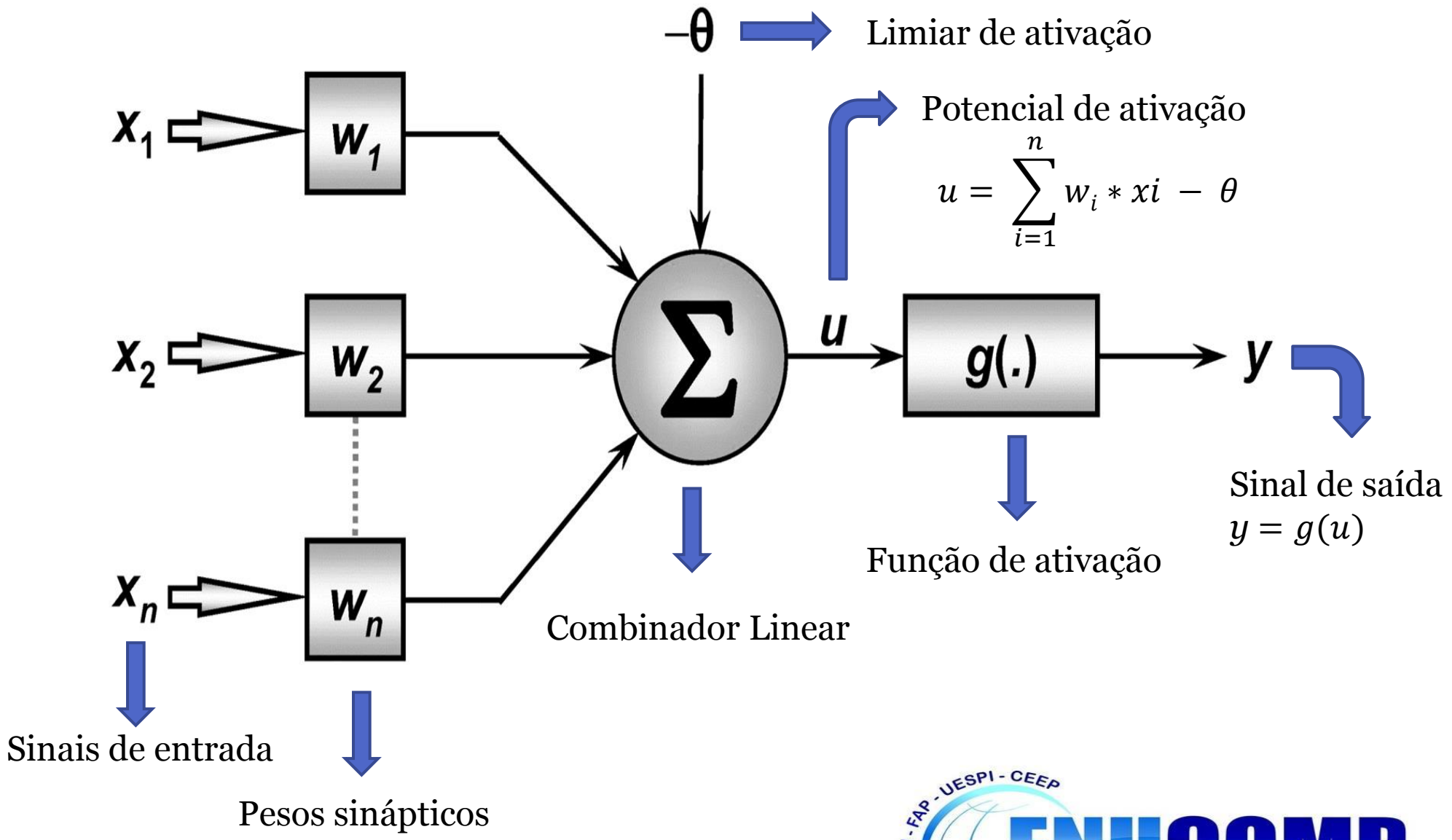
Neurônio Biológico



Neurônio Artificial

- Os **sinais de entrada** representados pelo conjunto $\{x_1, x_2, \ldots, x_n\}$, equivalem aos impulsos elétricos externos captados pelos dendritos.
- As **sinapses** são representadas pelas ponderações sinápticas ajustadas em $\{w_1, w_2, \ldots, w_n\}$.
- O **potencial de ativação** u determinado pela função soma, equivale ao corpo celular.
- A **saída** propagada pelo axônio é representada por y.

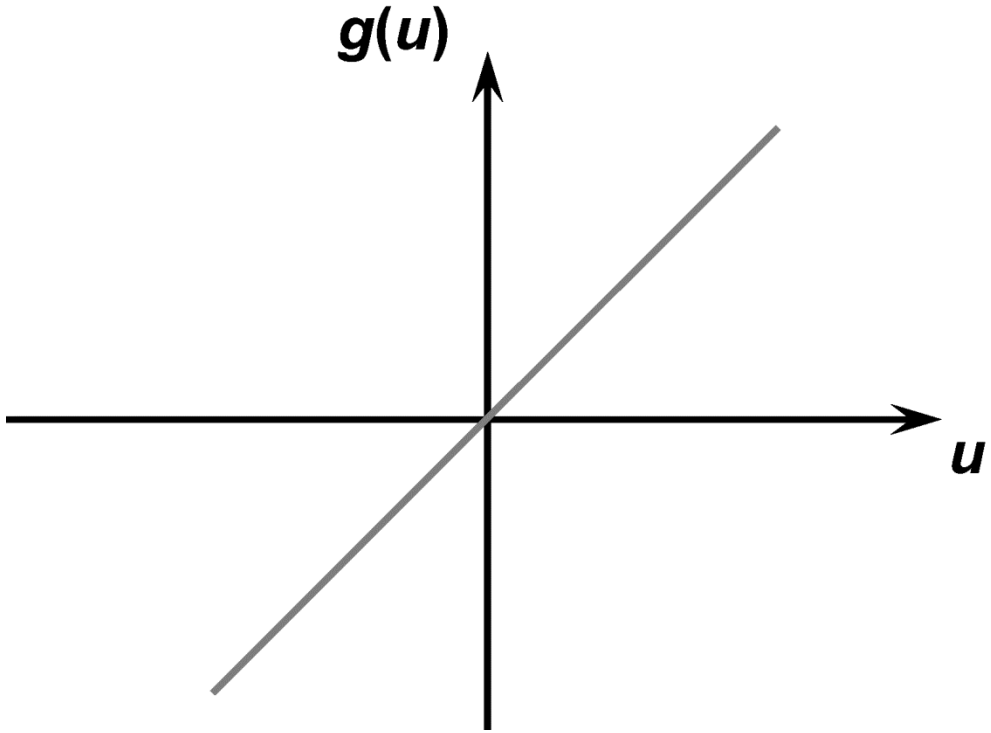
Neurônio Artificial



IFPI - FAP - UESPI - CEEP
ENUCOMP
ENCONTRO UNIFICADO DE COMPUTAÇÃO EM PARNAÍBA

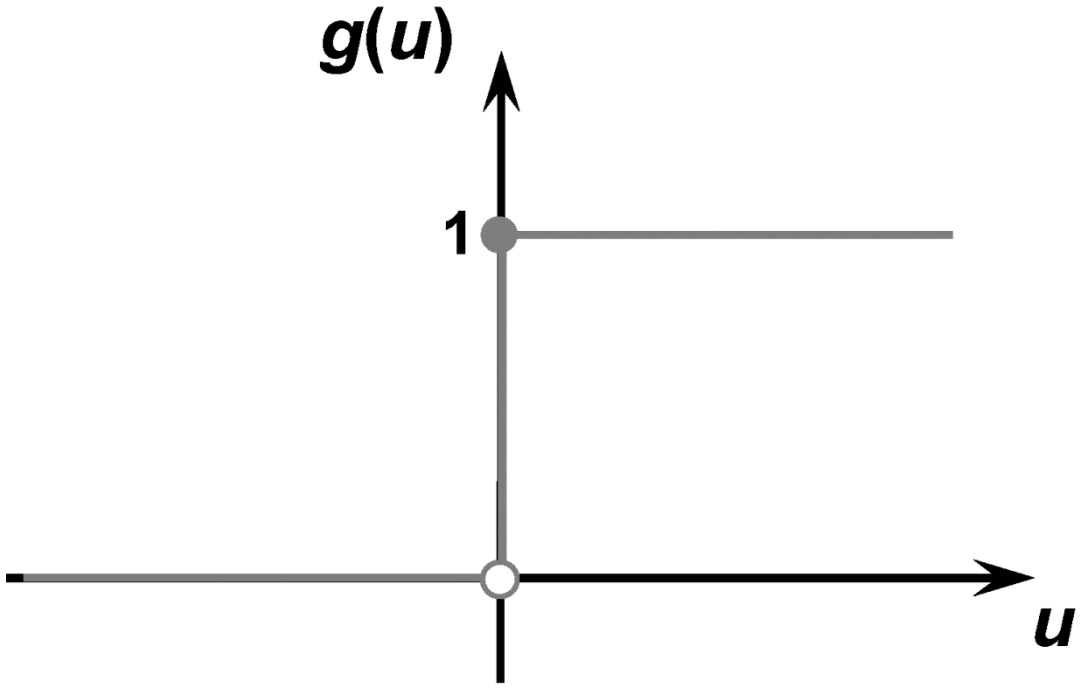
Funções de Ativação

- Função linear ⟶ $g(u) = u$



Funções de Ativação

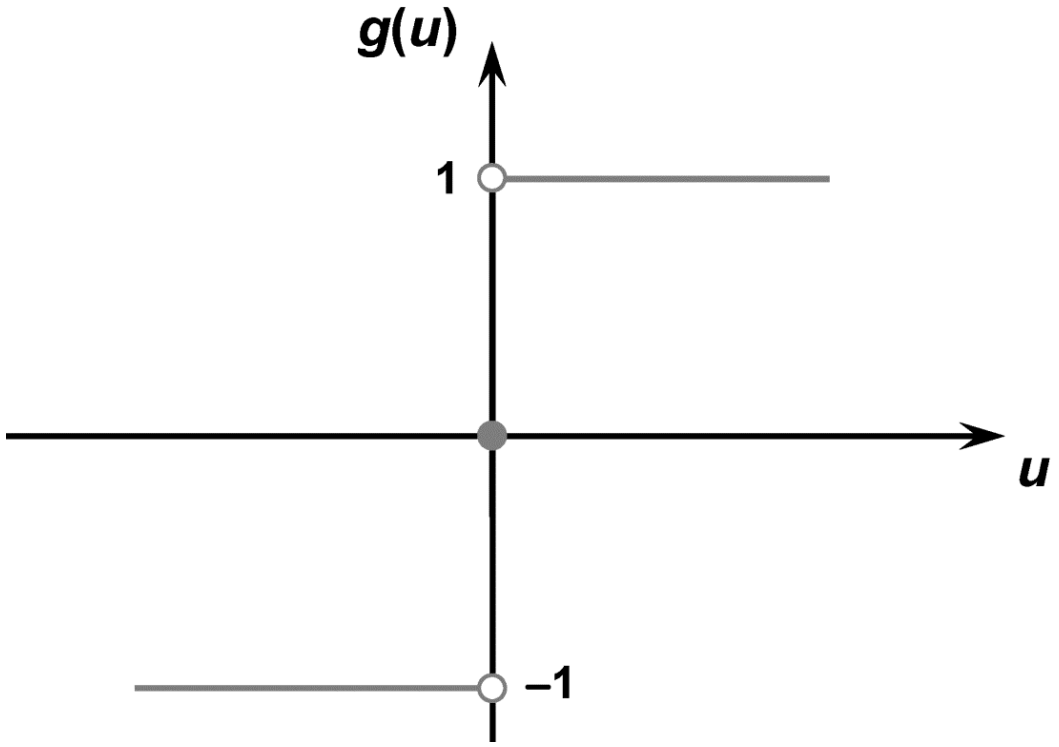
- Função degrau $\Rightarrow$ $g(u) = \begin{cases} 1 & se\ (u \geq 0) \\ 0 & se\ (u < 0) \end{cases}$



Funções de Ativação

- Função degrau bipolar

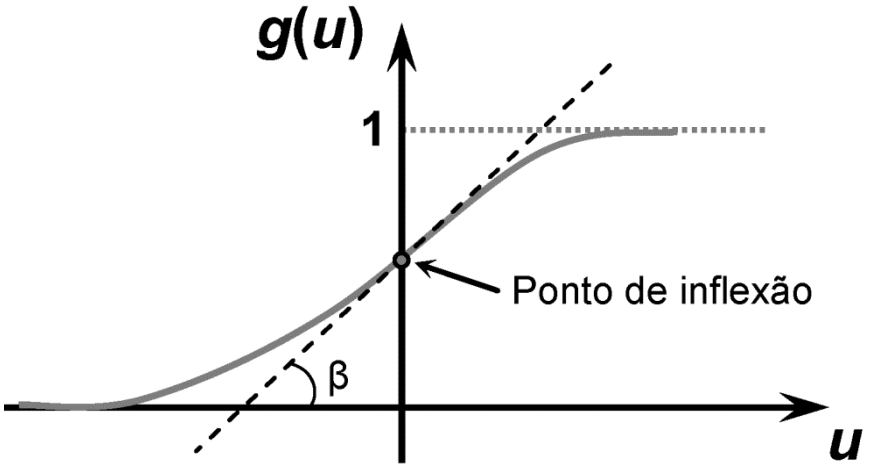
$$g(u) = \begin{cases} 1 & se\ (u > 0) \\ 0 & se\ (u = 0) \\ -1 & se\ (u < 0) \end{cases}$$

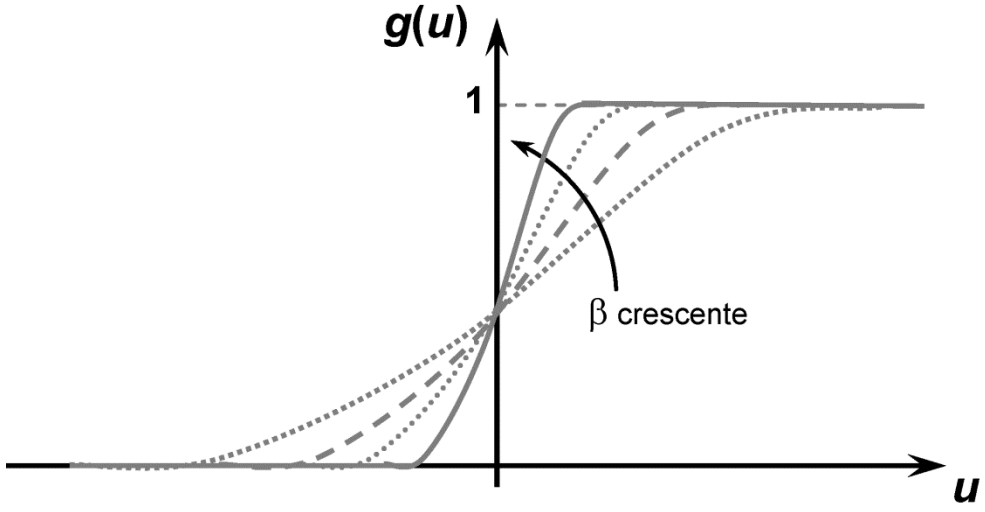


$$g(u) = \begin{cases} 1 & se\ (u \geq 0) \\ -1 & se\ (u < 0) \end{cases}$$

Funções de Ativação

- Função logística $\Rightarrow$ $g(u) = \dfrac{1}{1 + e^{-\beta u}}$



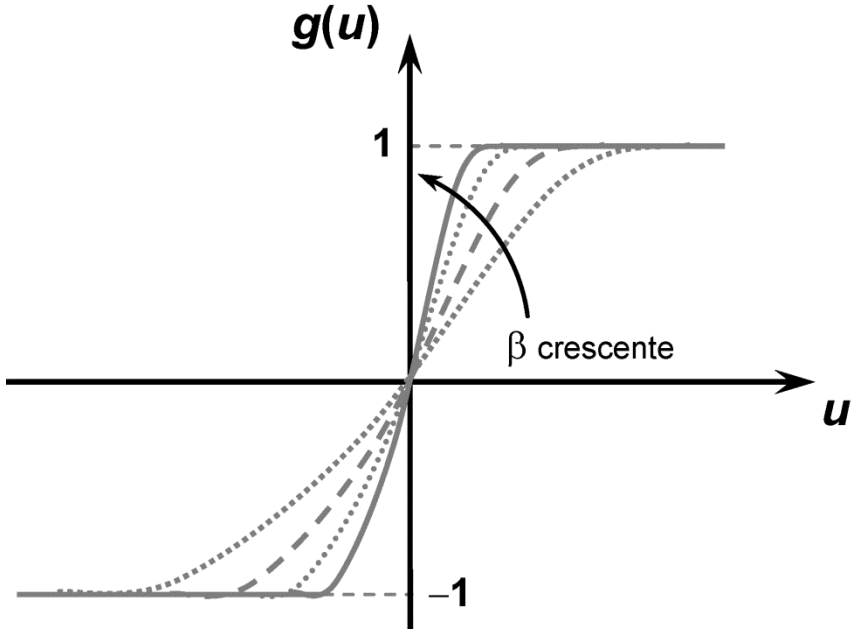


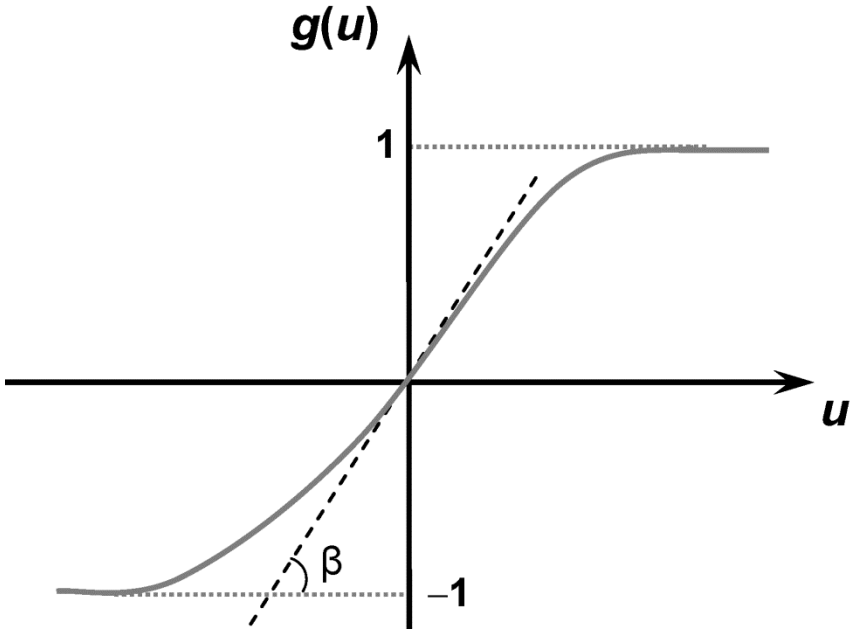
$e = 2{,}718281 \Rightarrow (Número\ de\ Euler)$

$\beta = constante\ de\ inclinação$

Funções de Ativação

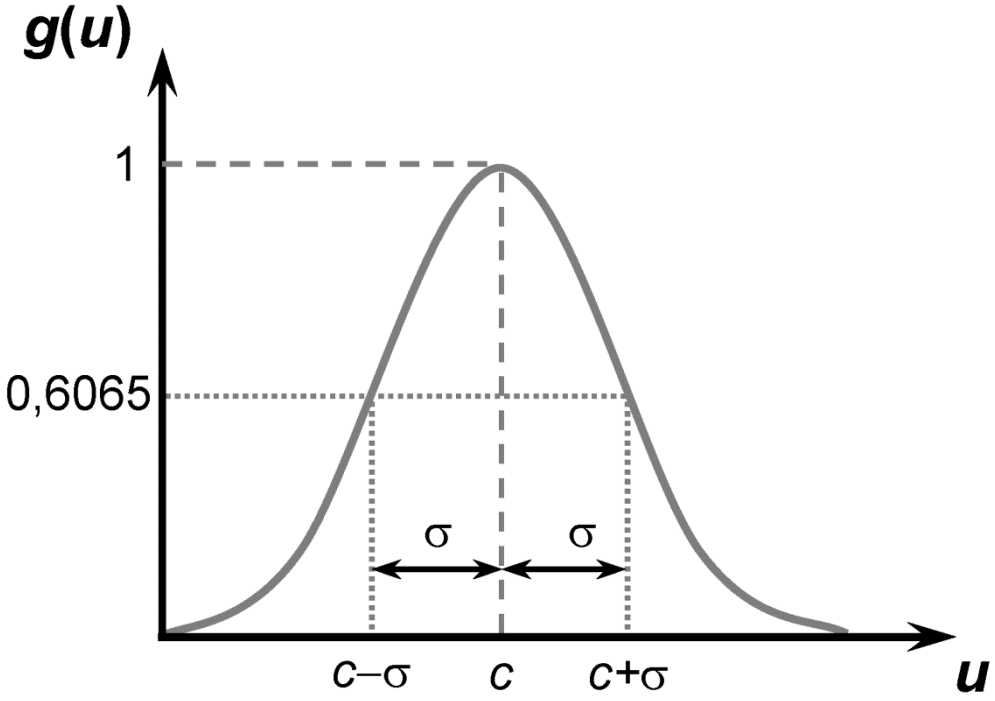
- Função tangente hiperbólica $\Rightarrow$ $g(u) = \dfrac{1 - e^{-\beta u}}{1 + e^{-\beta u}}$





Funções de Ativação

- Função gaussiana → $g(u) = e^{-\frac{(u-c)^2}{2\sigma^2}}$



$e = 2{,}718281 \Rightarrow Número\ de\ Euler$

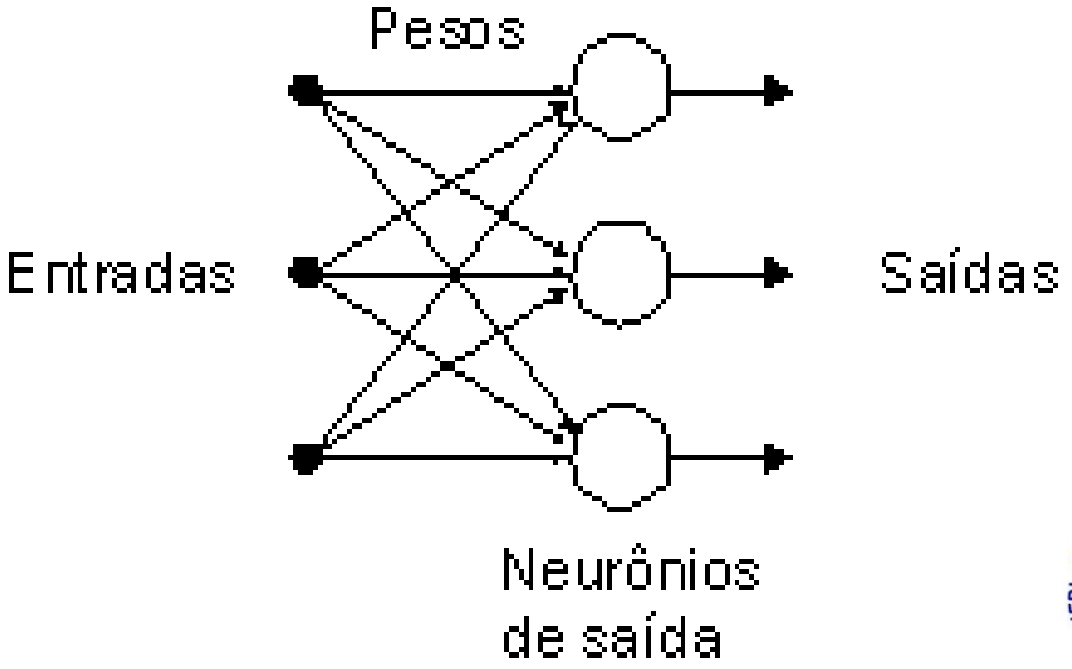
$\sigma \Rightarrow Desvio\ Padrão$

$c \Rightarrow Centro\ da\ Função\ Gaussiana$

Estruturas das RNAs

- **Redes feedforward de camada simples**

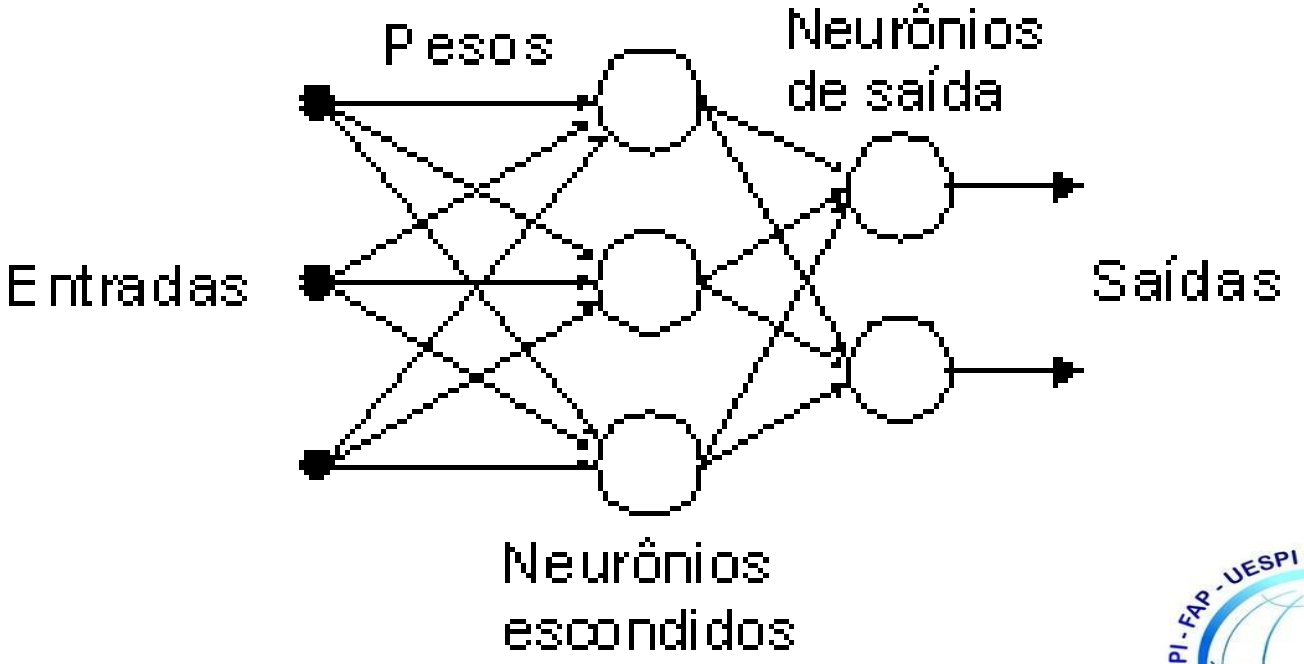
Este tipo de estrutura é formada por apenas uma camada de entrada e uma única camada de neurônios, que é a mesma camada de saída.



Estruturas das RNAs

- **Redes feedforward de camada múltipla**

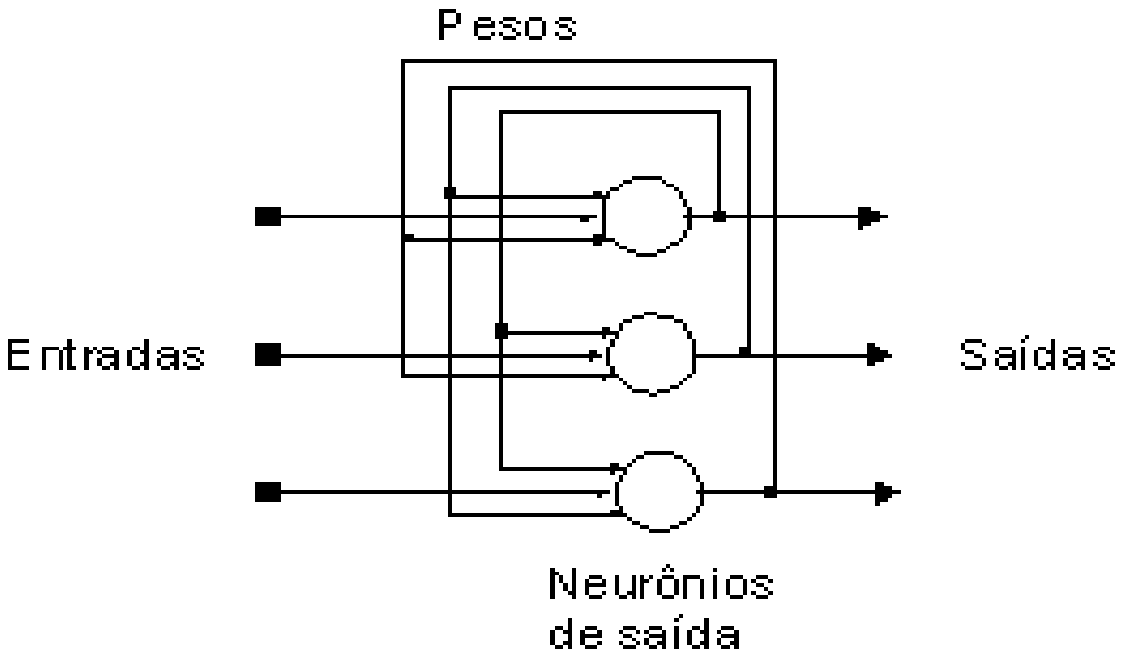
São constituídas pela presença de uma ou mais camadas escondidas de neurônios.



Estruturas das RNAs

- **Redes recorrentes**

São redes em que as saídas dos neurônios são realimentadas como sinais de entrada para outros neurônios.



Aprendizado em RNAs

"A aprendizagem é um processo pelo qual os parâmetros livres de uma rede neural são adaptados através de um processo de estimulação pelo ambiente no qual a rede está inserida. O tipo de aprendizagem é determinado pela maneira pela qual a modificação dos parâmetros ocorre." Haykin (2001)

Aprendizado em RNAs

- **Aprendizado supervisionado**

São fornecidos pares de entrada-saída para a rede;

A cada entrada fornecida, a saída é comparada com a saída correta.

Se a saída obtida for diferente da saída correta, os pesos sinápticos são corrigidos.

Aprendizado em RNAs

- **Aprendizado não-supervisionado**
  - Aprendizagem por reforço

É realizado através de um crítico que procura maximizar as boas ações executadas pela rede neural

  - Aprendizagem não-supervisionada

Não existe qualquer supervisor ou crítico, são dadas condições para a rede se auto organizar de acordo com uma medida de representação

Regras de Aprendizagem

- **Aprendizagem por correção de erro**

Seu princípio básico é usar o sinal de erro, para modificar os pesos gradualmente.

$$e_k(n) = d_k(n) - y_k(n)$$

- **Aprendizagem baseada em memória**

As experiências são armazenadas em uma grande memória de exemplos de entrada-saída

- **Aprendizagem competitiva**

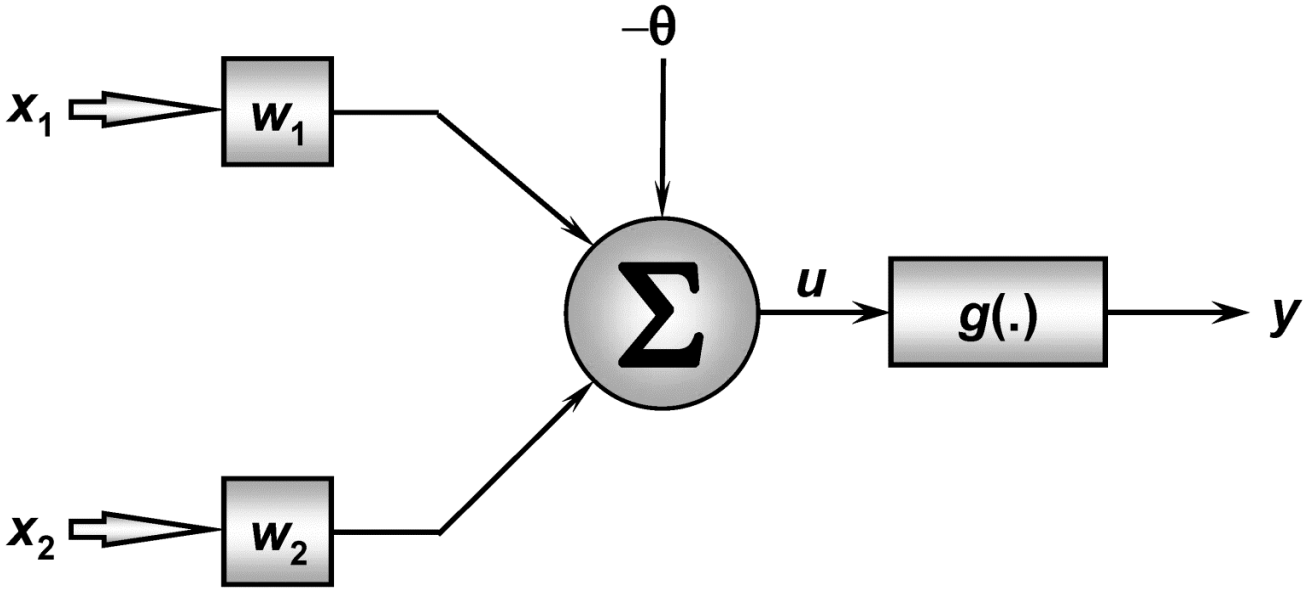
- Os neurônios disputam entre si, único vencedor de saída

Regras de Aprendizagem

- **Aprendizagem hebbiana**
  - Dois neurônios ativados sincronamente => Força sináptica ativada
  - Dois neurônios ativados assincronamente => Força sináptica enfraquecida
- **Aprendizagem de boltzman**
  - Máquina de boltzman derivado da mecânica estatística, distribuição de probabilidade desejada

Rede Perceptron Simples

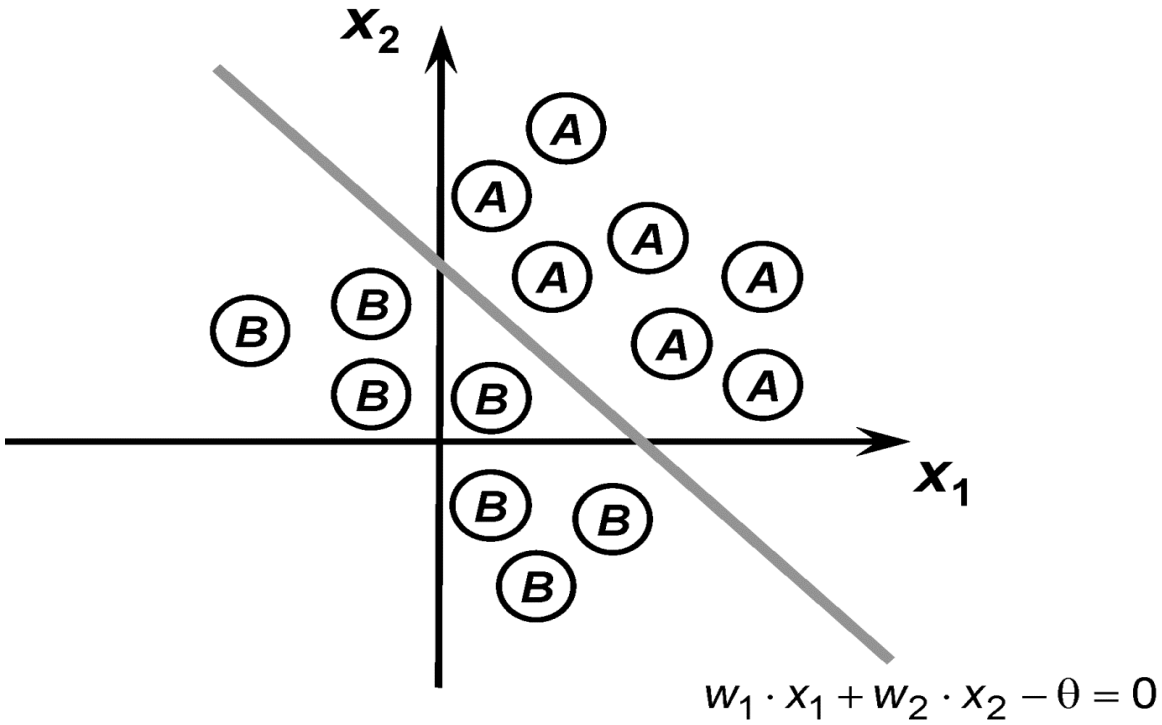
- **Estrutura da Rede**
  - Duas entradas



$$y = \begin{cases} 1, se \left(\sum_{i=1}^{n} w_i * xi - \theta \right) \geq 0 \leftrightarrow w_1 * x_1 + w_2 * x_2 - \theta \geq 0 \\ -1, se \left(\sum_{i=1}^{n} w_i * xi - \theta \right) < 0 \leftrightarrow w_1 * x_1 + w_2 * x_2 - \theta < 0 \end{cases}$$

Rede Perceptron Simples

- Classificador de padrões linearmente separável



Rede Perceptron Simples

- Processo de ajuste dos pesos
  - η ➔ Constante da taxa de aprendizado (0 < η < 1)
  - y ➔ Valor de saída produzida pelo *Perceptron*
  - $d^{(k)}$ ➔ Valor desejado para k-ésima amostra de treinamento
  - $x^{(k)}$ ➔ K-ésima amostra de treinamento
  - w ➔ Vetor contendo os pesos (inicialmente gerados aleatoriamente)

Notação matemática: $w^{Atual} = w^{Anterior} + \eta * \left(d^{(k)} - y\right) * x^{(k)}$

Notação algorítmica: $w = w + \eta * \left(d^{(k)} - y\right) * x^{(k)}$

Rede Perceptron Simples

- Algoritmo de aprendizagem

Início {Algoritmo *Perceptron* – Fase de Treinamento}

<1> Obter o conjunto de amostras de treinamento $\{ \boldsymbol{x}^{(k)} \}$;

<2> Associar a saída desejada $\{ d^{(k)} \}$ para cada amostra obtida;

<3> Iniciar o vetor $\boldsymbol{w}$ com valores aleatórios pequenos;

<4> Especificar a taxa de aprendizagem $\{\eta\}$;

<5> Iniciar o contador de número de épocas $\{época \leftarrow 0\}$;

<6> Repetir as instruções:

<6.1> *erro* ← "inexiste";

<6.2> Para todas as amostras de treinamento $\{ \boldsymbol{x}^{(k)}, d^{(k)} \}$, fazer:

<6.2.1> $u \leftarrow \boldsymbol{w}^T \cdot \boldsymbol{x}^{(k)}$;

<6.2.2> $y \leftarrow \text{sinal}(u)$;

<6.2.3> Se $y \neq d^{(k)}$

<6.2.3.1> Então
$$\begin{cases} \boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \cdot (d^{(k)} - y) \cdot \boldsymbol{x}^{(k)} \\ erro \leftarrow \text{"existe"} \end{cases}$$

<6.3> *época* ← *época* + 1;

Até que: *erro* ← "inexiste"

Fim {Algoritmo *Perceptron* – Fase de Treinamento}

Rede Perceptron Simples

- Algoritmo de aprendizagem

Início {Algoritmo *Perceptron* – Fase de Operação}

<1> Obter uma amostra a ser classificada { $\boldsymbol{x}$ };

<2> Utilizar o vetor $\boldsymbol{w}$ ajustado durante o treinamento;

<3> Executar as seguintes instruções:

<3.1> $u \leftarrow \boldsymbol{w}^T \cdot \boldsymbol{x}$;

<3.2> $y \leftarrow \text{sinal}(u)$;

<3.3> Se $y = -1$

<3.3.1> Então: amostra $\boldsymbol{x} \in$ *{Classe A}*

<3.4> Se $y = 1$

<3.4.1> Então: amostra $\boldsymbol{x} \in$ *{Classe B}*

Fim {Algoritmo *Perceptron* – Fase de Operação}

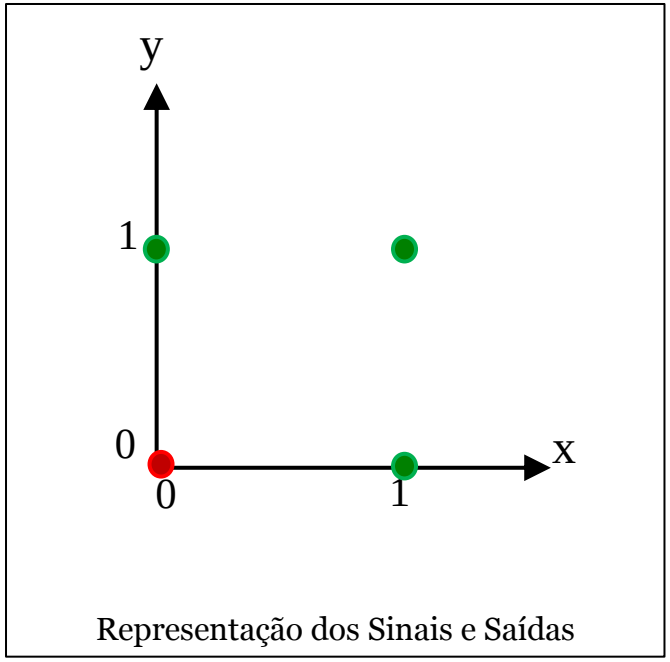
Rede Perceptron Simples

- **Exemplo de aplicação**
- Porta lógica OR

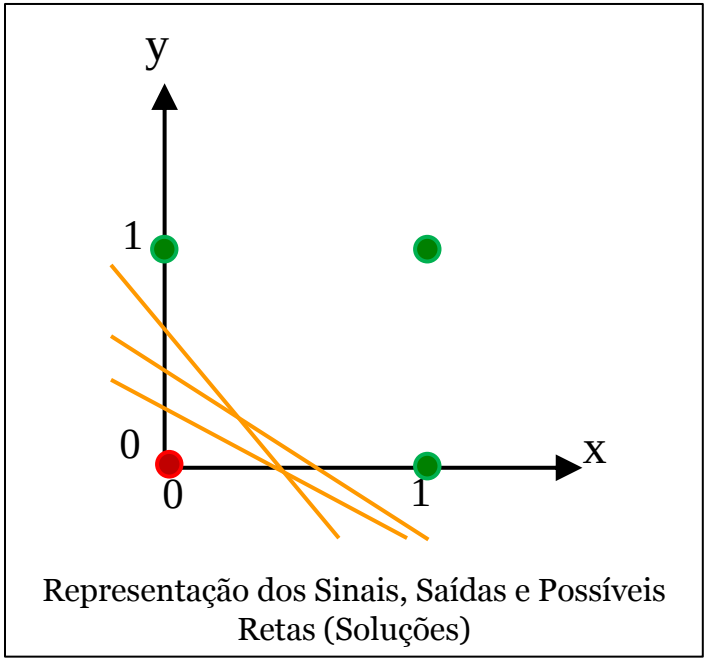
Sinal 1 (x)	Sinal 2 (y)	Saída (x *OR* y)
0	0	0 (Vermelho)
0	1	1 (Verde)
1	0	1 (Verde)
1	1	1 (Verde)

Rede Perceptron Simples

- **Exemplo de aplicação**
- Porta lógica OR



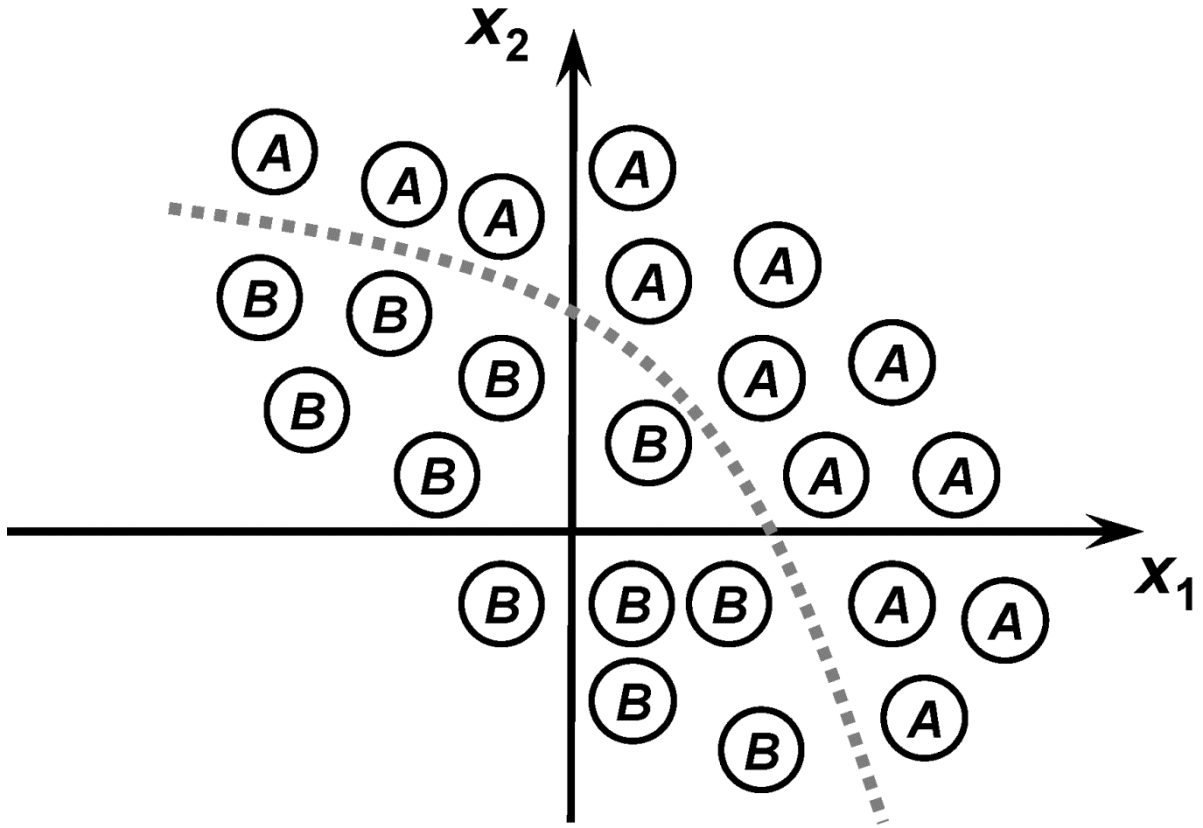
Representação dos Sinais e Saídas



Representação dos Sinais, Saídas e Possíveis Retas (Soluções)

Rede Perceptron Simples

- Não classifica (não linearmente separável)



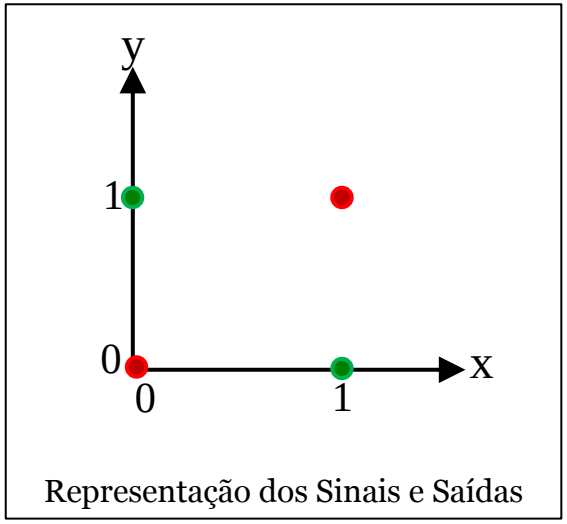
Rede Perceptron Simples

- XOR -> Não linearmente separável

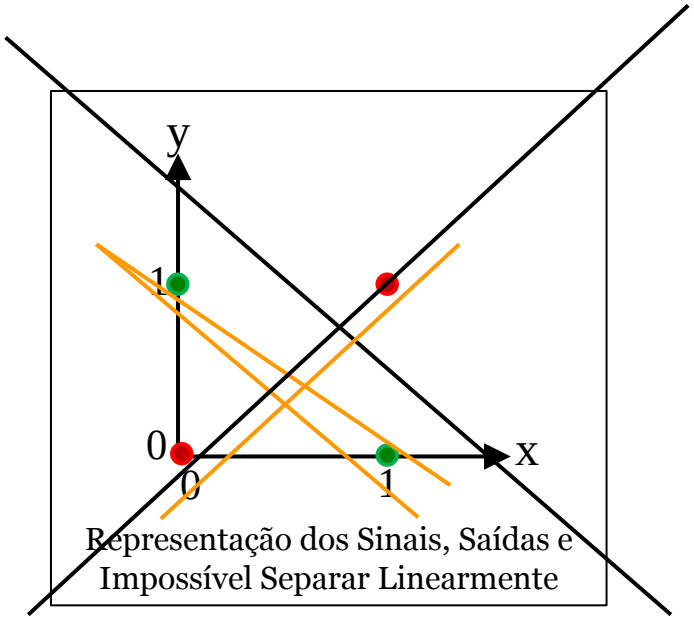
Sinal 1 (x)	Sinal 2 (y)	Saída (x *XOR* y)
0	0	0 (Vermelho)
0	1	1 (Verde)
1	0	1 (Verde)
1	1	0 (Vermelho)

Rede Perceptron Simples

- XOR -> Não linearmente separável



Representação dos Sinais e Saídas



Representação dos Sinais, Saídas e Impossível Separar Linearmente

Rede Perceptron Simples

- Exemplo

x_1 (Fósforo - mg)	x_2 (Acidez - mg)	x_3 (Cálcio - mg)	Classe (1 = tangerina) (-1 = laranja)
0,1	0,4	0,7	1
0,5	0,7	0,1	1
0,6	0,9	0,8	-1
0,3	0,7	0,2	-1

Rede Perceptron Simples

- Exemplo

$w = \begin{bmatrix} 0{,}34 \\ -0{,}23 \\ 0{,}94 \\ 0{,}05 \end{bmatrix}$	$\eta = 0{,}05$	$d^{(1)} = 1$	$x^{(1)} = \begin{bmatrix} -1 \\ 0{,}1 \\ 0{,}4 \\ 0{,}7 \end{bmatrix}$
		$d^{(2)} = 1$	$x^{(2)} = \begin{bmatrix} -1 \\ 0{,}5 \\ 0{,}7 \\ 0{,}1 \end{bmatrix}$
		$d^{(3)} = -1$	$x^{(3)} = \begin{bmatrix} -1 \\ 0{,}6 \\ 0{,}9 \\ 0{,}8 \end{bmatrix}$
		$d^{(4)} = -1$	$x^{(4)} = \begin{bmatrix} -1 \\ 0{,}3 \\ 0{,}7 \\ 0{,}2 \end{bmatrix}$

Rede Perceptron Simples

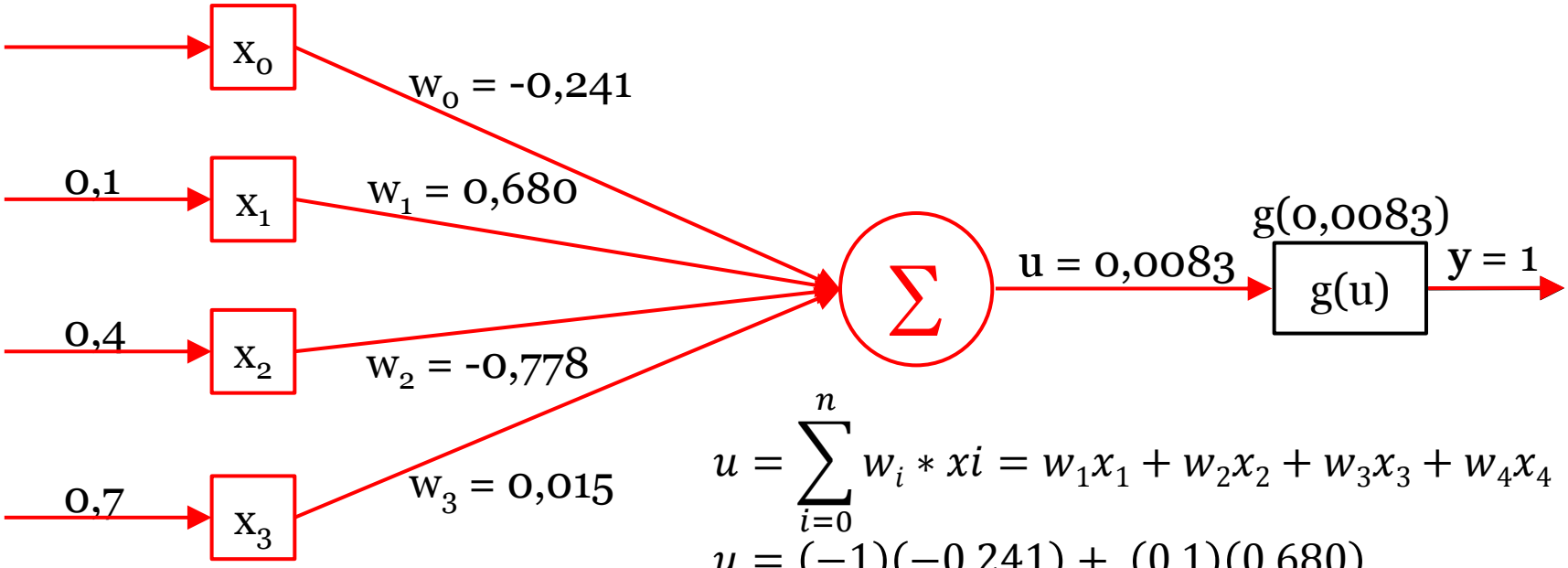
- Exemplo - Ajuste dos pesos

$$w = w + \eta * \left(d^{(k)} - y\right) * x^{(k)}$$

$$\mathrm{w} = \begin{bmatrix} 0{,}34 \\ -0{,}23 \\ 0{,}94 \\ 0{,}05 \end{bmatrix} + 0{,}05\ (1\ -1) \begin{bmatrix} -1 \\ 0{,}6 \\ 0{,}9 \\ 0{,}8 \end{bmatrix}$$

Rede Perceptron Simples

- Exemplo - Pesos ajustados



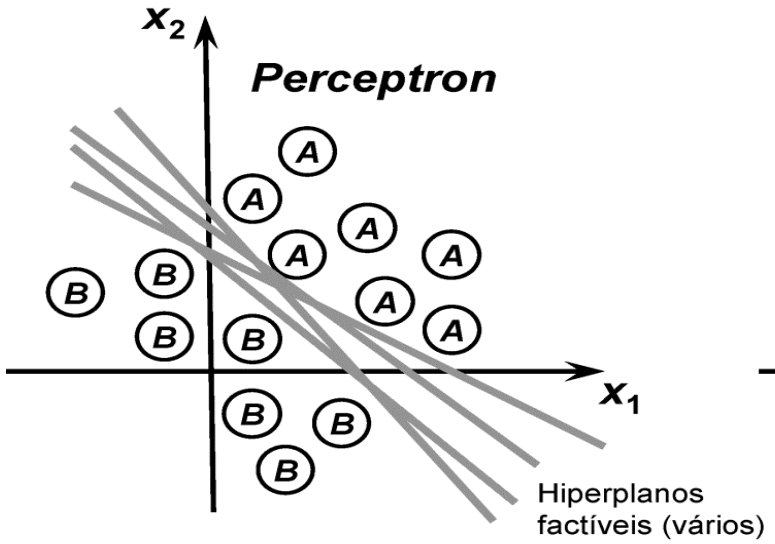
$$u = \sum_{i=0}^{n} w_i * xi = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

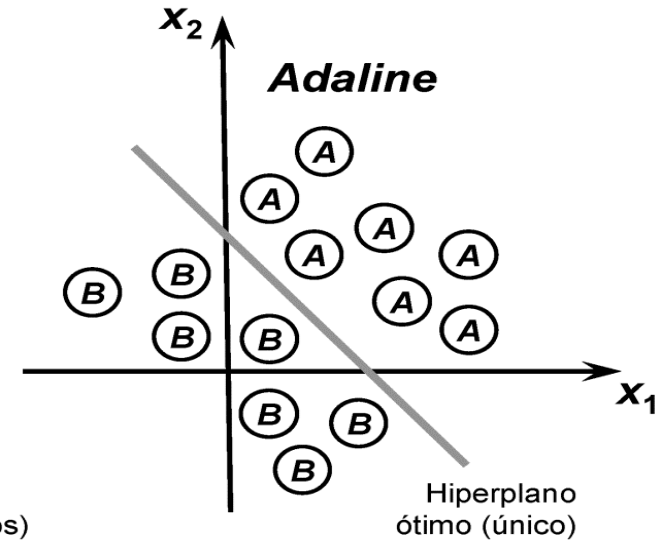
$$u = (-1)(-0{,}241) + (0{,}1)(0{,}680) + (0{,}4)(-0{,}778) + (0{,}7)(0{,}015)$$

$$u = 0{,}0083$$

Rede Adaline

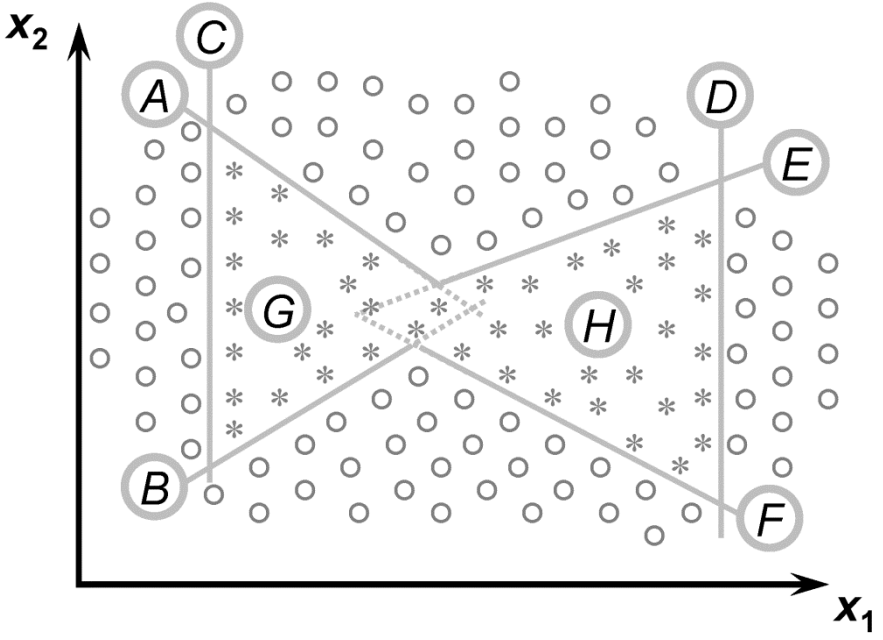
- Ajustado pelo método dos Mínimos Quadrados dos Erros (LMS – *least mean square*)
- Essência da regra Delta (ajuste de pesos de redes múltiplas camadas)

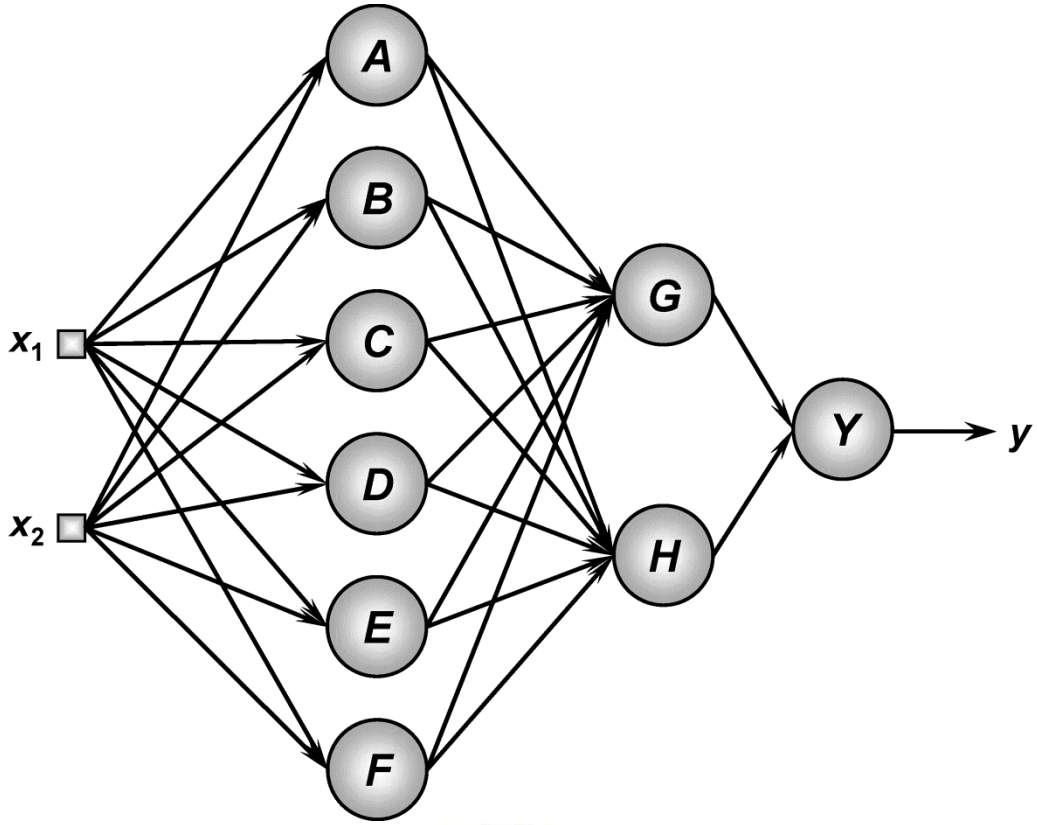




Rede Multiplas Camadas

- Usa regra delta generalizada ou método gradiente descendente.





Etapas para desenvolver uma aplicação de RNA

- Coleta de dados
- Separação em conjuntos
- Configuração da rede
- Treinamento da rede
- Testes da rede

A biblioteca Encog

"O Encog é um framework da Inteligência Artificial para Java que suporta não só redes neurais como outras áreas da IA. É uma estrutura de aprendizagem de máquina avançada que suporta uma variedade de algoritmos avançados, bem como métodos de apoio para normalizar e processar dados. Algoritmos de aprendizagem de máquina, como Support Vector Machines, Redes Neurais Artificiais, redes Bayesianas e algoritmos genéticos são suportados." [Heaton 2011]

Instalação

- Para instalar é necessário baixar a última versão do Encog no seguinte link:

  http://www.heatonresearch.com/encog/

- Depois é só extrair os seguintes arquivos em um local desejado e incluir o caminho das bibliotecas ao usar uma IDE.

  The Encog Core

  The Encog Examples

  The Encog Workbench

Elementos básicos de construção da rede neural no Encog

- O Encog está estruturado em camadas, sinapses, propriedades, classe lógica neural e funções de ativação.

Elementos básicos de construção da rede neural no Encog

- Camada de entrada:

MLData – Interface usada para definir que classes mantém esses arrays de dados de entrada. O array pode ser convertido em um objeto **MLData**, através do seguinte código:

MLData data = new BasicMLData (input);

Elementos básicos de construção da rede neural no Encog

- Camada de entrada:
  - A classe **BasicMLData** implementa a interface **MLData**.
  - Existem outras classes que implementam o **MLData** para diferentes tipos de entrada de dados.
- Camada de saída:
  - Utiliza a mesma estrutura da camada de entrada, um array **BasicMLData**.

Elementos básicos de construção da rede neural no Encog

- Configuração da rede:

- A classe **BasicNetwork** implementa uma rede neural.
- Implementa as interfaces **MLProperties** e **MLMethod** que definem o tipo de aprendizado de máquina e suas propriedades.

Elementos básicos de construção da rede neural no Encog

- Configuração da rede:

- A classe **BasicLayer** implementa as funcionalidades básicas das camadas.
- Implementa a interface **Layer** que define todos os métodos necessários a uma camada.

Elementos básicos de construção da rede neural no Encog

- Treinamento da rede:

- A classe **MLDataSet** e **BasicMLDataSet** configuram as entradas e saídas ideais da rede.

MLDataSet trainingSet = new BasicMLDataSet(input, ideal);

Elementos básicos de construção da rede neural no Encog

- Treinamento da rede:

- Os métodos de treinamento podem ser utilizados através da interface **MLTrain**, aqui todos os métodos de treinamento implementam essa interface.

Elementos básicos de construção da rede neural no Encog

- Treinamento da rede:
  - Backpropagation
  - ResilientPropagation
  - ManhattanPropagation
  - QuickPropagation
  - ScaledConjugateGradient
  - LevenbergMarquardtTraining

Elementos básicos de construção da rede neural no Encog

- Teste da rede:

- O teste da rede ocorre através das classes **MLDataPair** que recebe os pares de entrada e saída do treinamento e a classe **MLData** que computa os respectivas saídas da rede, dado somente os dados de entrada e a rede treinada.

Implementação do BackPropagation - Problema do XOR

- Algoritmo simplicado:

1. Apresentação de um padrão X a rede, a qual fornece a saída;
2. Calculo do erro (diferença entre o valor desejado e a saída) para cada saída;
3. Determinação do erro pela rede associado a derivada parcial do erro quadrático;
4. Ajuste dos pesos de cada elemento;
5. Por fim, um novo padrão é apresentado a rede e o processo é repetido até que ocorra a convergência (erro < tolerância estabelecida) ou o número de interações atinja a um valor máximo estabelecido;

Implementação do BackPropagation - Problema do XOR

- Definição dos dados:

```
//Entrada necessária para o XOR
public static double XOR_INPUT[][] = {{1.0,0.0}, {0.0,0.0}, {0.0,1.0},
{1.0,1.0}};
// Dados ideais necessários para XOR
public static double XOR_IDEAL[][] = {{1.0}, {0.0}, {1.0}, {0.0}};
```

Implementação do BackPropagation - Problema do XOR

- Configuração da rede neural

```
//Cria a rede neural
BasicNetwork network = new BasicNetwork();
network.addLayer(new BasicLayer(null,true,2));
network.addLayer(new BasicLayer(new ActivationSigmoid(), true, 2));
network.addLayer(new BasicLayer(new ActivationSigmoid(),true,1));
network.getStructure().finalizeStructure();
network.reset();
```

Implementação do BackPropagation - Problema do XOR

- Treinamento da rede neural

```
//Cria dados de treinamento
MLDataSet trainingSet = new BasicMLDataSet(XOR_INPUT, XOR_IDEAL);
//Treinamento da rede neural
final Backpropagation train = new Backpropagation(network,trainingSet,
0.7, 0.3);
//Inicializo o numero de épocas
int epoch = 1;

//Interação de treinamento da rede
do {
  train.iteration();
  System.out.println("Epoch #" + epoch +"Error: " + train.getError());
  epoch++;
} while(train.getError() > 0.01);
```

Implementação do BackPropagation - Problema do XOR

- Teste da rede neural

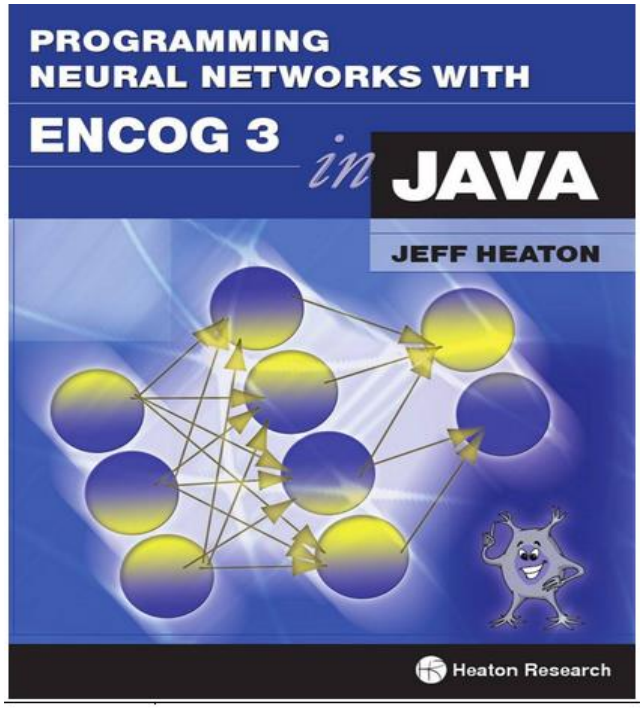
```
//Teste da rede neural
System.out.println("Neural Network Results:");
for (MLDataPair pair: trainingSet){
   final MLData output = network.compute(pair.getInput());
   System.out.println(pair.getInput().getData(0) + ","
                      + pair.getInput().getData(1) + ", actual="
                      + output.getData(0) + ", ideal="
                      + pair.getIdeal().getData(0));
}
```

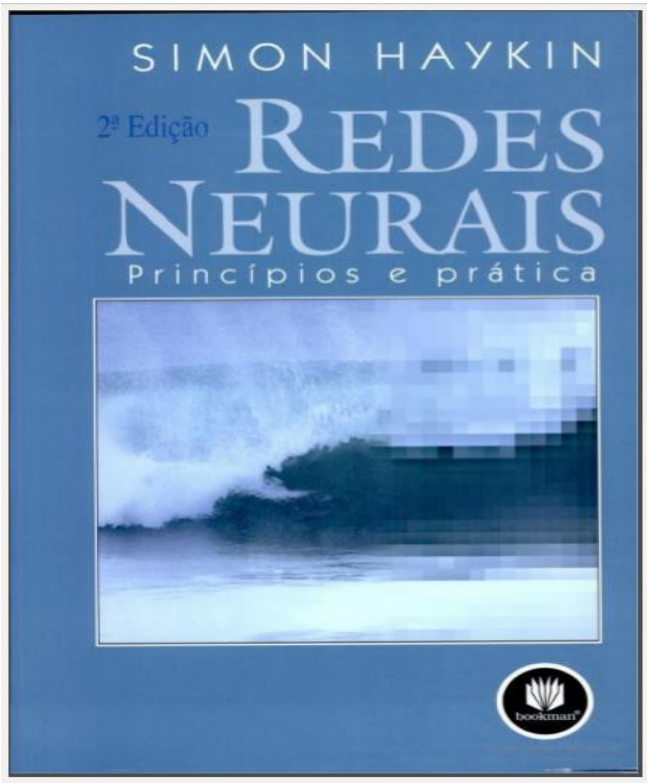
Bibliografia

SILVA, I. N.; Redes Neurais Artificiais para engenharia e ciências aplicadas.

HEATON, J.; Programming Neural Networks with Encog3 in Java.

HAYKIN, S.; Redes Neurais: Princípios e práticas





IFPI - FAP - UESPI - CEEP
ENUCOMP
ENCONTRO UNIFICADO DE COMPUTAÇÃO EM PARNAÍBA

Obrigada pela atenção!

- Contato:
- rachel.msousanet@gmail.com

IFPI - FAP - UESPI - CEEP
ENUCOMP
ENCONTRO UNIFICADO DE COMPUTAÇÃO EM PARNAÍBA