Redes Neurais

Cristiane Neri Nobre

Redes Neurais Artificiais

O que são Redes Neurais Artificiais?

Redes Neurais Artificiais (RNA) são modelos de computação com propriedades particulares:

- Capacidade de adaptação (aprendizado)
- Capacidade de generalização

O que são RNAs?

Uma rede neural é um processador maciçamente paralelamente distribuído constituído de **unidades de processamento simples**, que têm a propensão natural para **armazenar conhecimento experimental** e torná-lo disponível para o uso. Ela se assemelha ao cérebro em dois aspectos:

1. O conhecimento é adquirido pela rede a partir de seu ambiente através de um processo de aprendizagem
2. Forças de conexão entre neurônios, conhecidas como pesos sinápticos, são utilizadas para armazenar o conhecimento adquirido.

Simon Haykin

O que são RNAs?

- RNA: estruturas distribuídas formadas por grande número de unidades de processamento conectadas entre si.
- Modelos inspirados no cérebro humano, compostos por várias unidades de processamento (“neurônios”)
- Interligadas por um grande número de conexões (“sinapses”)
- Eficientes onde métodos tradicionais têm se mostrado inadequados

Limitações das redes neurais:

- Não fornecem explicações: relação entre entrada/saída é obscura – efeito “caixa preta”.
- Falta de um formalismo na especificação e análise:
  - necessidade de realizar árduas simulações até encontrar parâmetros e topologia adequados.
- Tempo de treinamento grande
- Necessitam muitos exemplos de treinamento

Potenciais áreas de aplicação das RNAs

- Classificação de padrões
- *Clustering*/categorização
- Aproximação de funções
    - Previsão
    - Otimização
    - Mineração de dados
    - etc...

Breve histórico

Década de 40 : O começo

- (1943) McCulloch & Pitts
    - Concentrou em descrever um modelo artificial de um neurônio biológico e apresentar suas capacidades computacionais

- (1949) Donald Hebb desenvolve algoritmo para treinar RNA (aprendizado Hebbiano)
    - Se dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada

1950-1960: Anos de euforia

- (1958) Von Neumann mostra interesse em modelagem do cérebro (RNA)
  - The Computer and the Brain, Yale University Press

- (1958) Frank Rosenblatt implementa primeira RNA, a rede Perceptron
  - Ajuste iterativo de pesos
  - Prova teorema da convergência

Década de 70: Pouca atividade

- (1969) Minsky & Papert analisam Perceptron e mostram suas limitações
    - Não poderiam aprender a resolver problemas simples como o **OU-exclusivo**
    - Causou grande repercussão

Década de 70: Pouca atividade

- Na década de 70, a abordagem conexionista ficou adormecida (buraco negro), apesar de alguns poucos trabalhos:
    - (1971) Igor Aleksander propõe redes sem pesos
    - (1972) Kohonen e Anderson trabalham com RNAs associativas
    - (1975) Grossberg desenvolve a Teoria da Ressonância Adaptativa (redes ART)

Década de 80: A segunda onda

- (1982) Hopfield mostra que Redes Neurais podem ser tratadas como sistemas dinâmicos
- (1986) Hinton, Rumelhart e Williams, propõem algoritmo de aprendizagem para **redes multi-camadas**

Motivação para as RNAs: redes biológicas

- O cérebro humano contém em torno de 10^{11} neurônios, sua célula fundamental.
- Cada um desses neurônios processa e se comunica com milhares de outros continuamente e em paralelo.
- A estrutura individual dos nodos, a topologia de suas conexões e o comportamento conjunto destes nodos naturais formam a base para o estudo das RNAs.

Motivação para as RNAs: redes biológicas

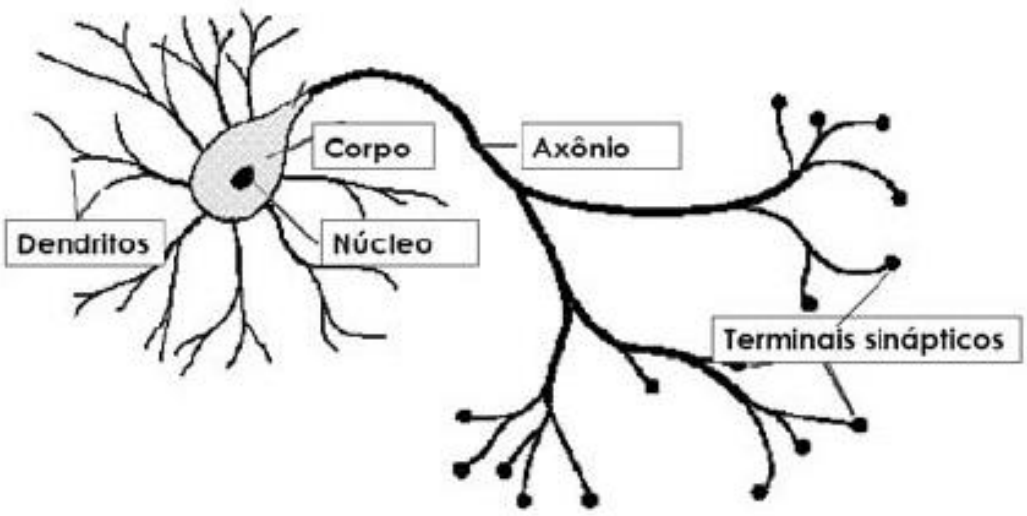
- O cérebro humano é responsável pelo que se chama de emoção, pensamento, percepção e cognição.

- Além disso, sua rede de nodos tem a capacidade de reconhecer padrões e relacioná-los, usar e armazenar conhecimento por experiência, além de interpretar observações.

- Assim, estruturas encontradas nos sistemas biológicos podem inspirar o desenvolvimento de novas arquiteturas para modelos de RNAs.

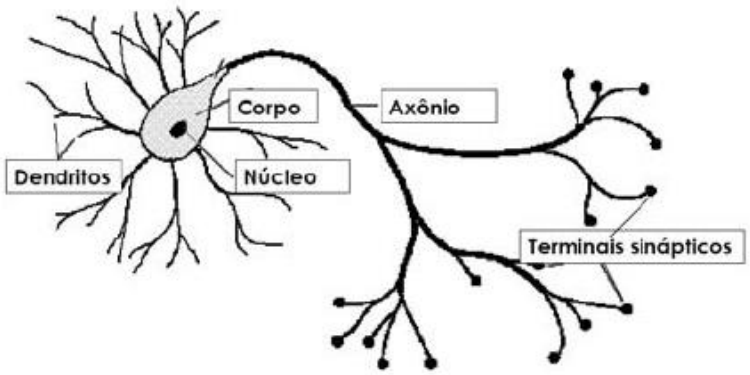
Neurônios biológicos

- Os neurônios são divididos em três seções: o **corpo da célula**, os **dentritos** e o **axônio**, cada um com funções específicas porém complementares.



Estrutura de um neurônio biológico

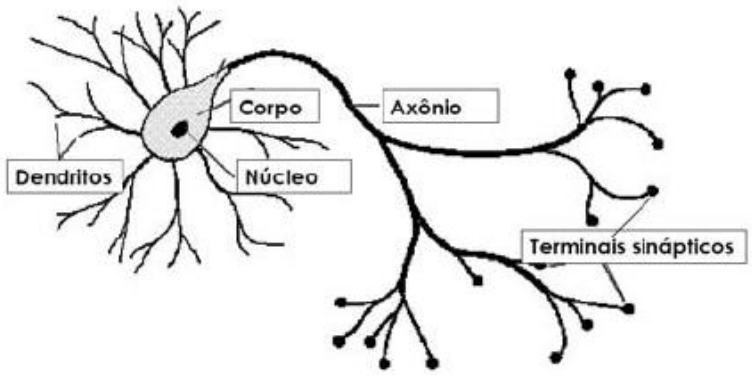
Neurônios biológicos



• Os **dentritos** têm por função receber as informações ou impulsos nervosos, oriundas de outros neurônios e conduzi-las até o **corpo celular**.

• No **corpo celular**, a informação é **processada**, e novos impulsos são gerados

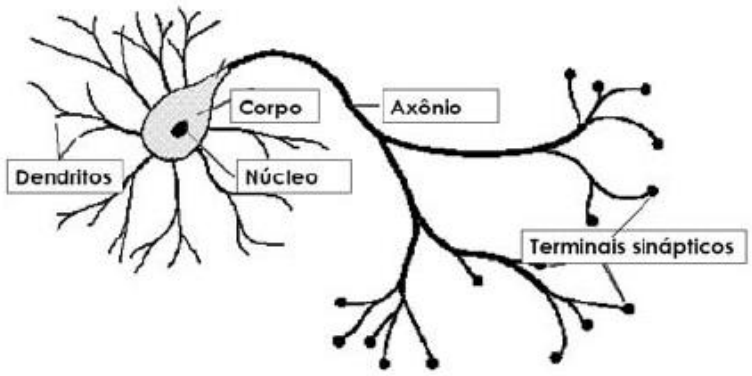
Neurônios biológicos



• Estes impulsos são transmitidos a outros neurônios, passando através do axônio até os dentritos dos neurônios seguintes.

• O ponto de contato entre a terminação axônica de um neurônio e o dendrito de outro é chamado de ***sinapses***. É pelas sinapses que os nodos se unem funcionalmente, formando redes neurais.

Neurônios biológicos



• As sinapses funcionam como **válvulas**, e são capazes de **controlar a transmissão de impulsos** – isto é, o fluxo de informação – entre os nodos na rede neural.

Conceitos básicos

- **Estrutura geral das RNAs:**
  - Unidades de processamento n_i (nós)
  - Conexões w_{ij}
  - Saída
  - Topologia

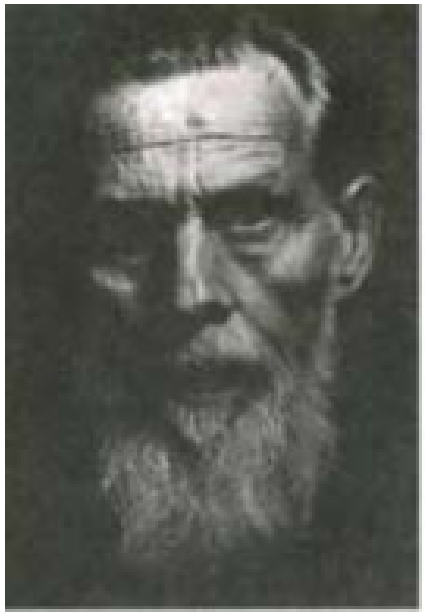
Conceitos básicos

- **Em linhas gerais o objetivo é**:

  ajustar os pesos da rede para **minimizar** alguma medida do erro no conjunto de treinamento.

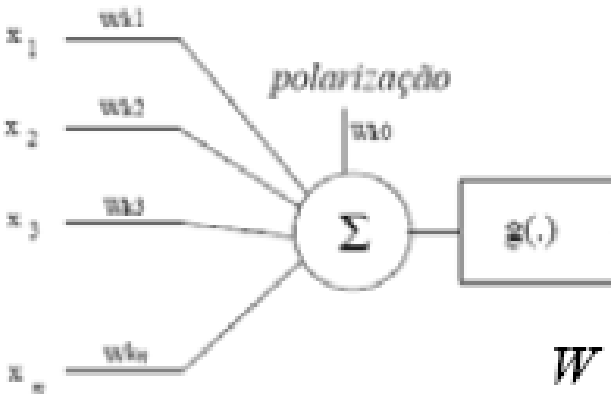
- Desse modo a aprendizagem é formulada como uma **busca de otimização** no **espaço de pesos**.

O neurônio de McCulloch-Pitts



Warren McCulloch

$x \in \{0,1\}$



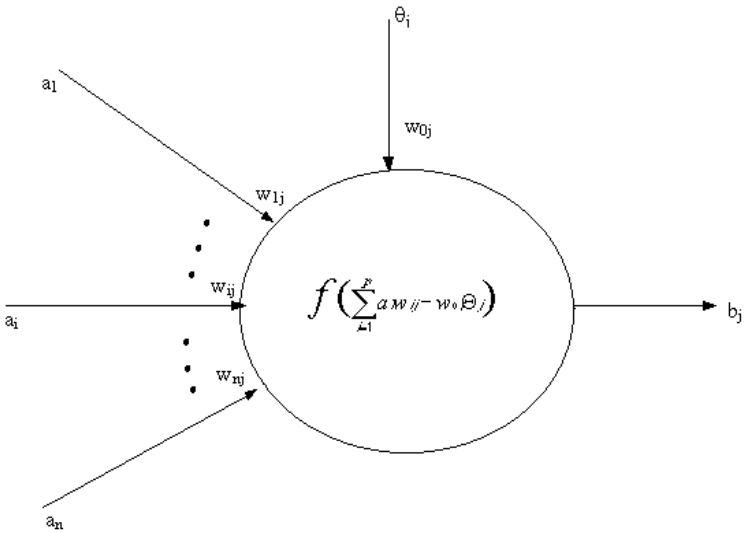
$y \in \{0,1\}$

$$W = \{w_1, w_2, \ldots, w_n\}$$
$$X = \{x_1, x_2, \ldots, x_n\}$$

- x_i é a excitação de entrada na sinapse i;
- y_k é a resposta (ou saída) do neurônio k;
- w_{ki} é o peso sináptico da entrada i do neurônio k;
- $g(.)$ é a função de ativação do neurônio.

Conceitos Fundamentais

- Modelo matemático de um neurônio



• Uma ligação da unidade j para unidade i serve para propagar a ativação a_j
desde j até i.

• Cada ligação tem um peso numérico $W_{j,i}$, que determina a sua intensidade.

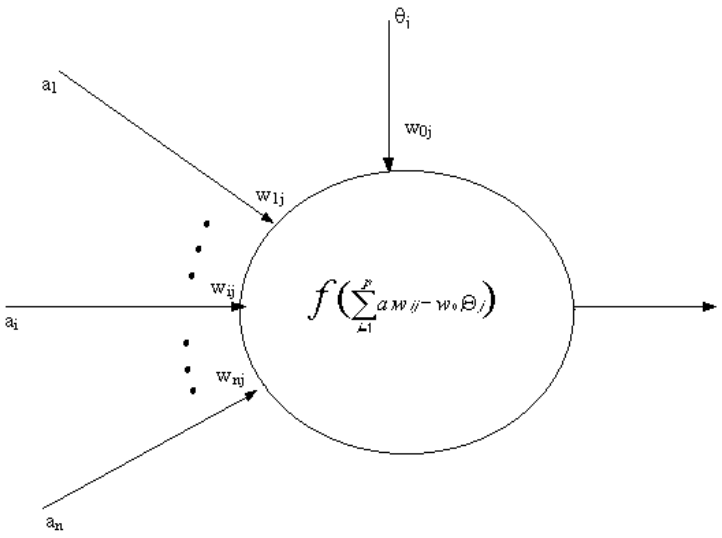
• Cada unidade calcula uma soma ponderada de suas entradas.

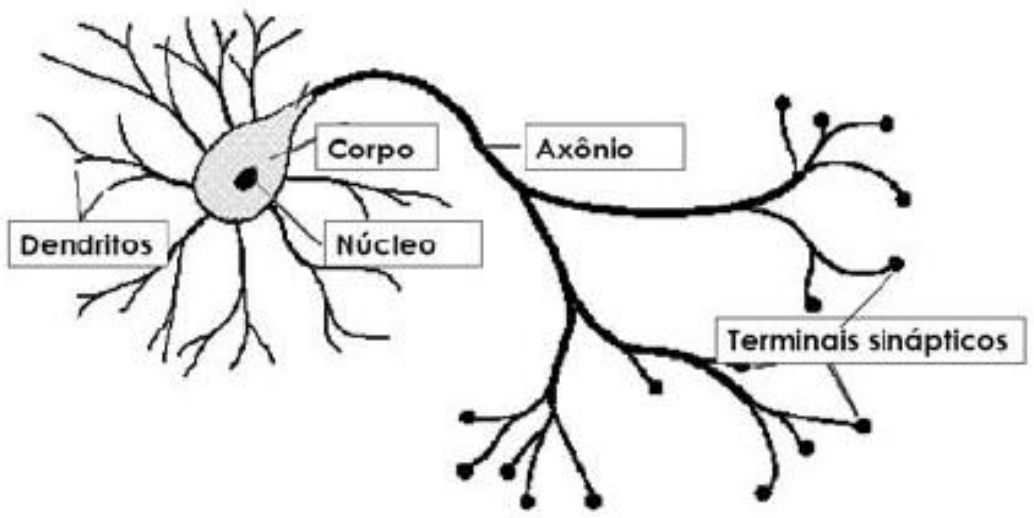
– Depois aplica uma função de ativação g a essa soma.

• O peso $W_{0,i}$ é conectado a uma entrada fixa a_0=-1 para permitir a representaçao de funções que não passem pela origem.

Conceitos Fundamentais

- Modelo matemático de um neurônio





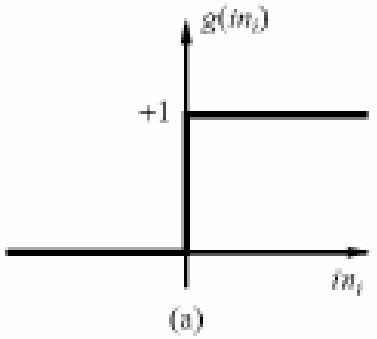
- Nesse modelo artificial, quem seriam as entradas (comparando-se com o modelo biológico)?
- Quem seria a saída?

Conceitos Fundamentais – Funções de ativação

A função de ativação tem que atender a dois critérios:

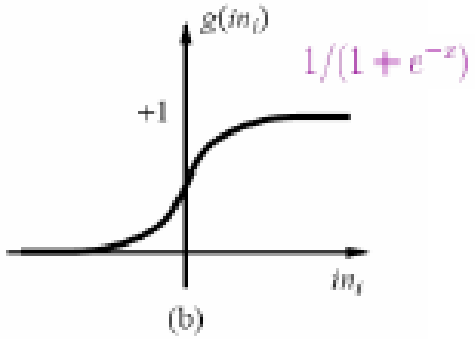
– Deve **estar ativa** (próxima de +1) para entradas "**corretas**" e **inativa** (próxima a 0) para entradas "**erradas**".

– Deve ser **não linear** para que a rede como um todo possa representar funções não-lineares.



(a)

Função de limiar

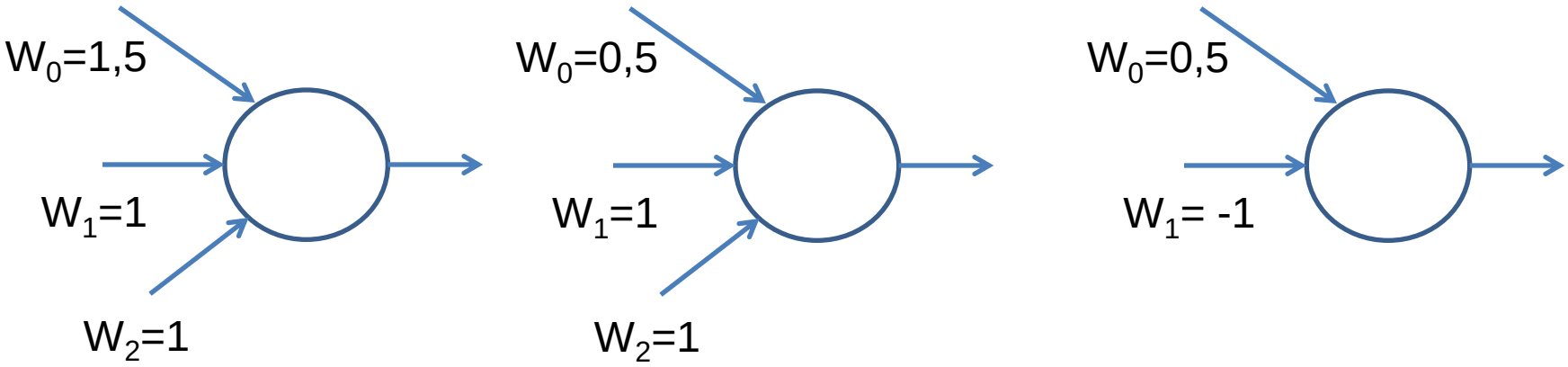


(b)

Função sigmóide

Implementando funções lógicas

Podemos utilizar as unidades para construir uma rede que calcule qualquer função booleana das entradas.

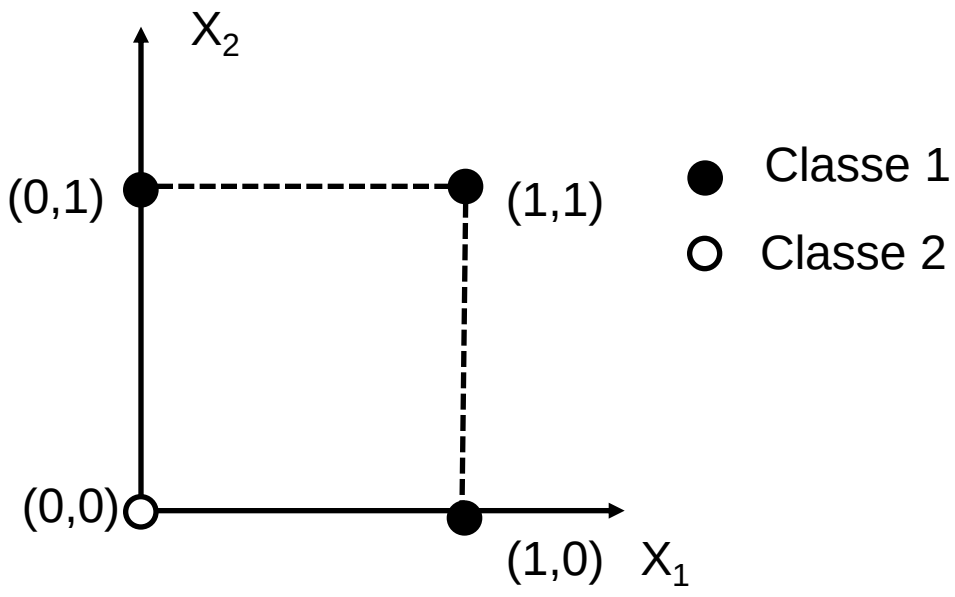


AND OR NOT

Implementando funções lógicas

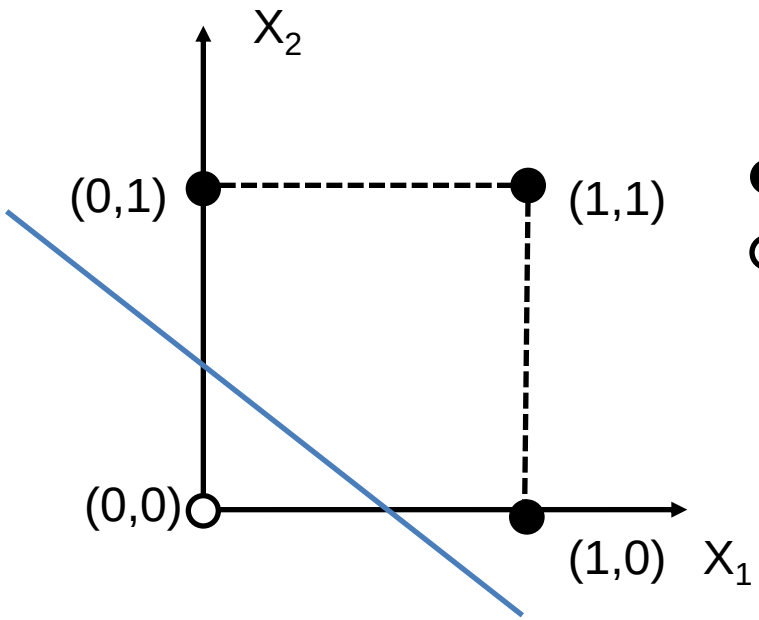
Exemplo 1: Implementando a função lógica OR

X_1	X_2	X_1 OR X_2
0	0	0
0	1	1
1	0	1
1	1	1



Implementando funções lógicas

Exemplo 1: É possível encontrar uma reta que separe os pontos da Classe 1 (y=1) dos da Classe 2 (y=0)?



Classe 1

Classe 2

Resposta:

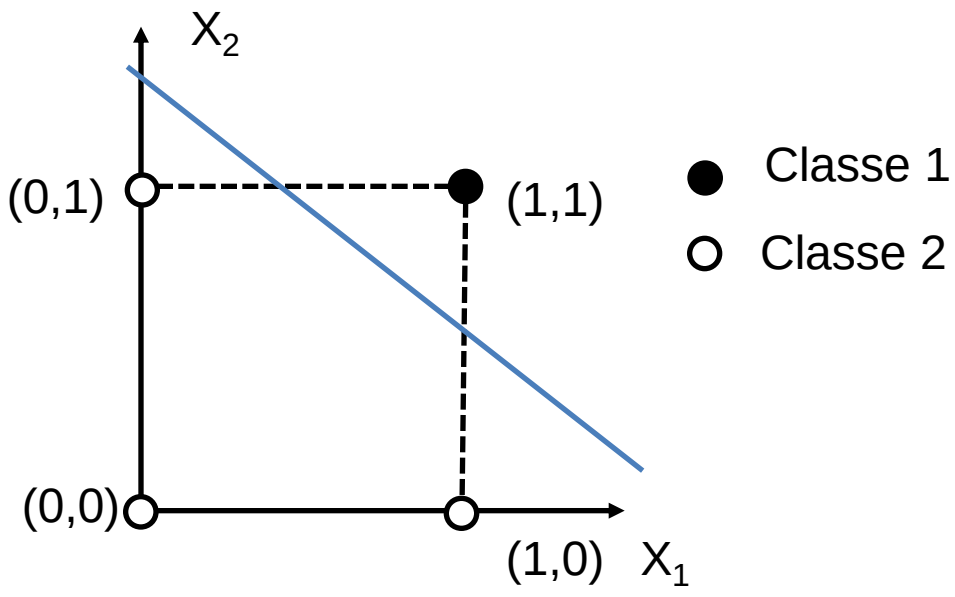
SIM!

Obs: Na verdade, é possível encontrar *infinitas* retas que separam as duas classes!

Implementando funções lógicas

Exemplo 2: E como ficaria a função **AND**?

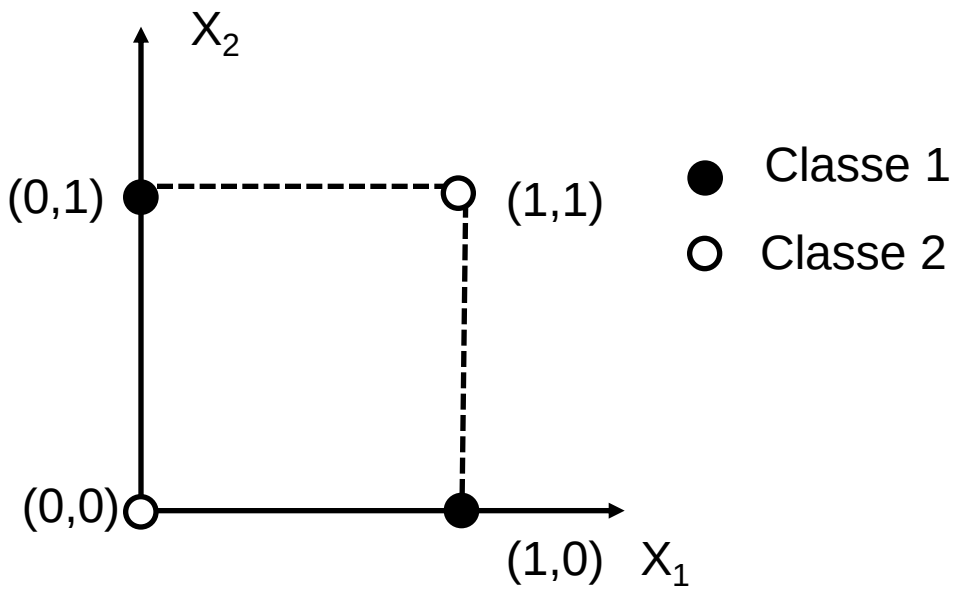
X_1	X_2	X_1 AND X_2
0	0	0
0	1	0
1	0	0
1	1	1



Implementando funções lógicas

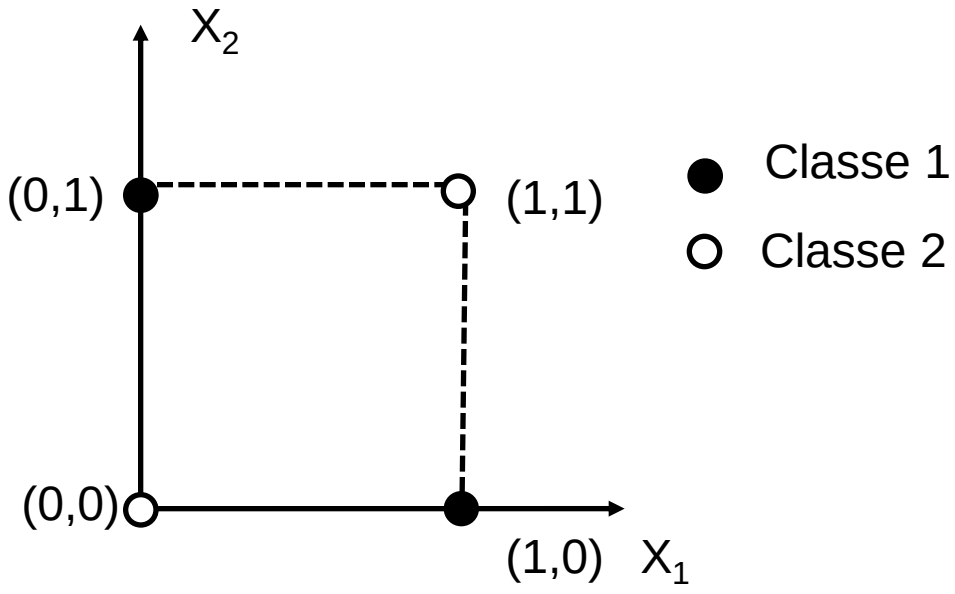
Exemplo 3: E como ficaria a função **XOR**?

X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0



Implementando funções lógicas

Exemplo 3: É possível encontrar uma reta que separe os pontos da Classe 1 (y=1) dos da Classe 2 (y=0)?

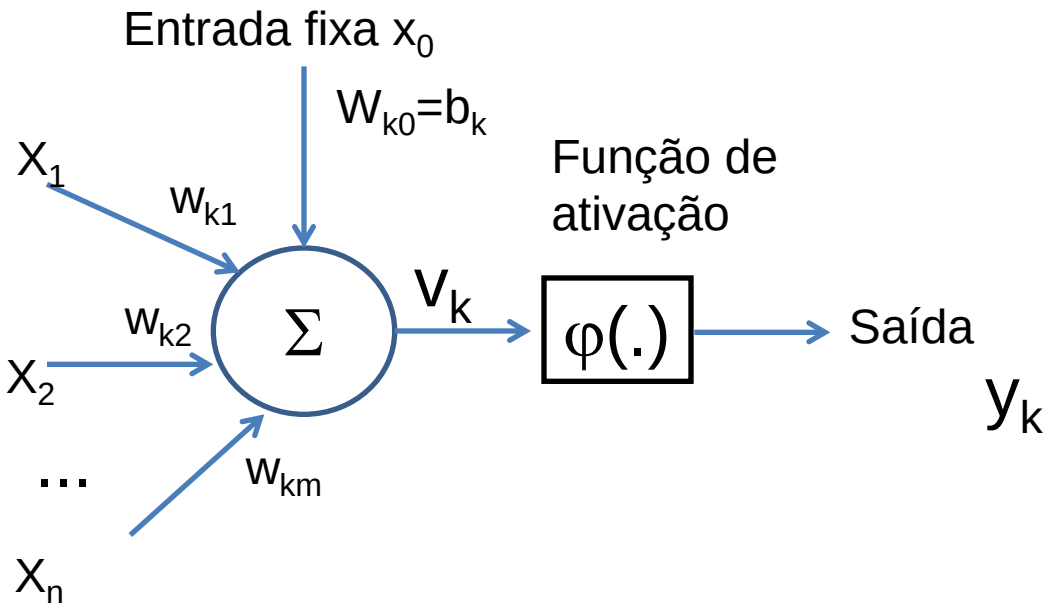


Resposta:

NÃO!

Neurônio de McCulloch-Pitts

O neurônio de McCulloch-Pitts pode ser modelado como um caso particular de um discriminador linear:



$$y_k = \begin{cases} 1 \; se \; v_k \geq 0 \\ 0 \; se \; v_k < 0 \end{cases}$$

$$v_k = \sum_{j=1}^{m} w_{kj} x_j - b_k$$

http://acroque.blogspot.com.br/2008/10/o-modelo-de-mcculloch-e-pitts-parte-1.html

Neurônio de McCulloch-Pitts

Podemos identificar três elementos básicos do modelo neural:

1. Um conjunto de *sinapses*

2. Um *somador* para somar os sinais de entrada, ponderados pelas respectivas sinapses do neurônio

3. Uma *função de ativação* para restringir a amplitude da saída de um neurônio. A função de ativação é também referida como função restritiva já que restringe (limita) o intervalo permissível de amplitude do sinal de saída a um valor finito.

Exercício

Considere um neurônio de McCulloch-Pitts de duas entradas (X_1 e X_2) e uma saída (y). Considere que todos os pesos **sinápticos w são iguais a 0,5**. Seja a função de ativação dada por:

$$y_k = \begin{cases} 1 \, se \, v_k > 0 \\ 0 \, se \, v_k \leq 0 \end{cases}$$

Determine os valores de limiar para que a rede implemente as funções booleanos:

a) AND
b) OR

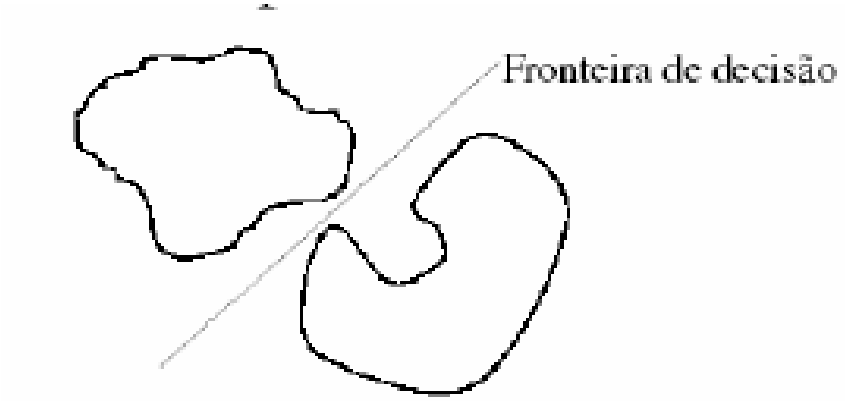
Solucão

$$v_k = \sum_{j=1}^{m} w_{kj} x_j - b_k$$

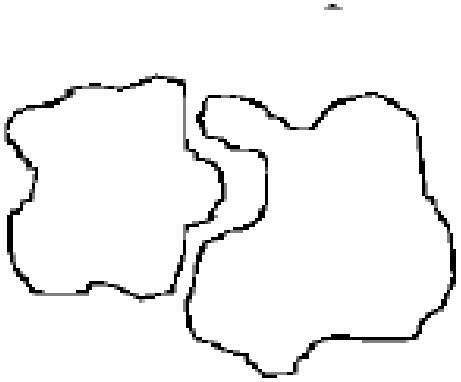
$$y_k = \begin{cases} 1 \; se \; v_k > 0 \\ 0 \; se \; v_k \leq 0 \end{cases}$$

	Função (X_1,X_2)	Somatória	Limiar	Y
AND	0,0 → 0 0,1 → 0 1,0 → 0 1,1 → 1	0*0,5 + 0*0,5 = 0 0*0,5 + 1*0,5 = 0,5 1*0,5 + 0*0,5 = 0,5 1*0,5 + 1*0,5 = 1	b_k = 0,75 (0,5 ≤ b_k < 1)	0 0 0 1
OR	0,0 → 0 0,1 → 1 1,0 → 1 1,1 → 1	0*0,5 + 0*0,5 = 0 0*0,5 + 1*0,5 = 0,5 1*0,5 + 0*0,5 = 0,5 1*0,5 + 1*0,5 = 1	b_k = 0,499 (0 ≤ b_k < 0,5)	0 1 1 1

Problemas linearmente separáveis e não linearmente separáveis



Padrões linearmente separáveis



Padrões não linearmente separáveis

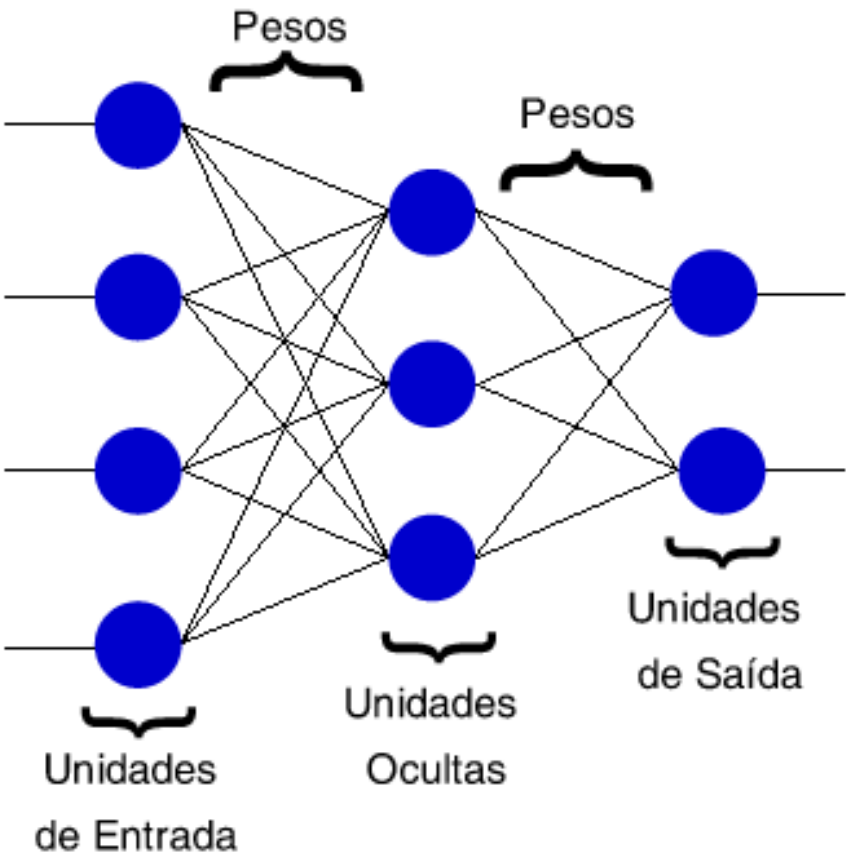
Um conjunto de treinamento *S* é **linearmente separável** se é possível separar os padrões das classes diferentes contidos no mesmo por pelo menos um hiperplano (Russel and Norvig, 1995).
http://www.lncc.br/~labinfo/tutorialRN/frm3_operacaoBooleana.htm

Problemas linearmente separáveis e não linearmente separáveis

Classificadores que separam os dados por meio de um hiperplano são **denominados lineares**, podendo ser definidos pela Equação:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

Esquema de uma Rede Neural Artificial



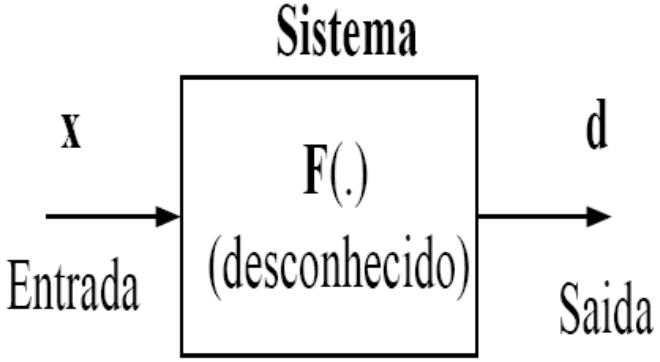
Em outras palavras...

Vamos assumir que existe uma lei matemática F(·), também chamada aqui de função ou mapeamento, que relaciona um vetor de entrada qualquer, com um vetor de saída.

Esta relação, representada genericamente na Figura abaixo, pode ser descrita matematicamente da seguinte forma:

$$d = F[x]$$

em que se assume que F(·) é totalmente desconhecida, ou seja, não sabemos de antemão quais são as fórmulas usadas para associar um vetor de entrada x com seu vetor de saída d correspondente.



Aprendizado

- Capacidade de aprender a partir de seu ambiente e melhorar seu desempenho com o tempo

- Parâmetros livres de uma RNA são adaptados através de estímulos fornecidos pelo ambiente
    - Processo iterativo de ajustes aplicado às sinapses
    - A RNA deve saber mais sobre seu ambiente após cada iteração

- RNA deve produzir para cada conjunto de entradas apresentado o conjunto de saídas adequado.

- Forma geral para o ajuste de pesos é:
  - $w_{ik}(n+1) = w_{ik}(n) + Dw_{ik}(n)$

Mecanismos de aprendizado

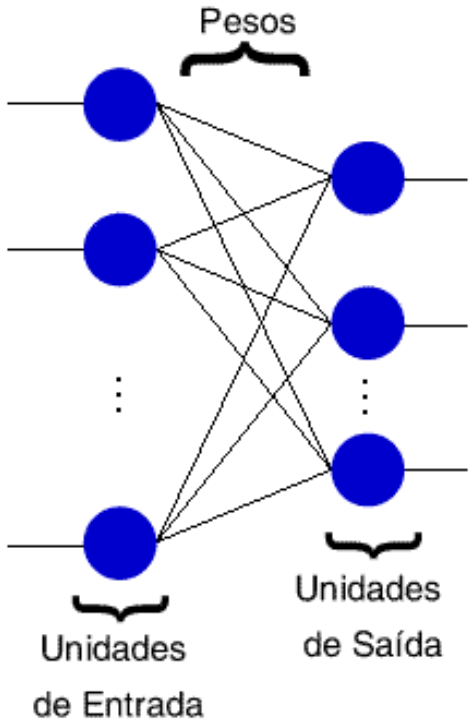
- Modificação de pesos (Dwij(n)) associados às conexões
- Acréscimo e/ou eliminação de conexões e/ou nodos

Perceptron

Perceptron Simples

Perceptron

O Perceptron, proposto por Rosenblatt, é composto pelo neurônio de McCulloch-Pitts, com Função de Limiar, e Aprendizado Supervisionado. Sua arquitetura consiste na entrada e uma camada de saída.



Perceptron - Características

- O Perceptron é usado para conjuntos de treinamento linearmente separáveis

- Inclusão de tendência (“bias”) $$S = \sum_{j=1}^{p} W_j . x_i - b$$

- No algoritmo de aprendizagem do Perceptron busca-se um vetor W que tenha projeção positiva (produto interno) com todos os exemplos positivos e projeção negativa com todos os exemplos negativos

- O algoritmo de aprendizagem converge em um número finito de passos que classifica corretamente um conjunto de treinamento linearmente separável.

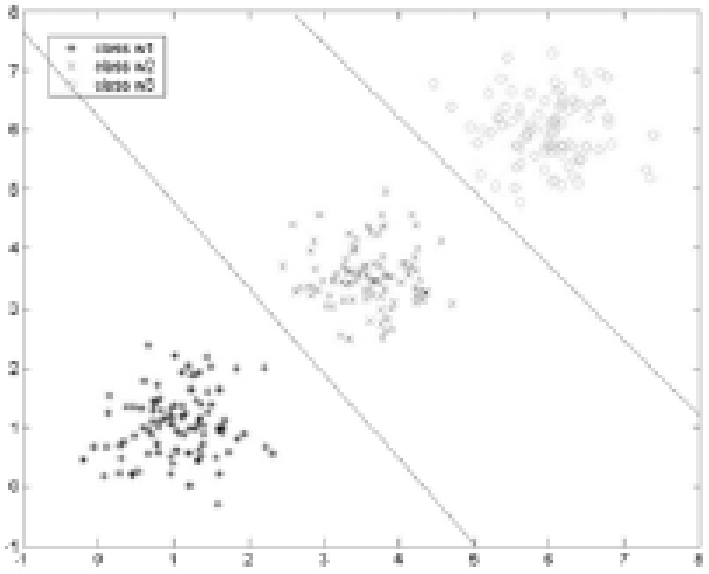
Perceptron - Características

- A superfície de decisão(curva de separação) forma um hiperplano, ou seja, para um dos lados está uma classe e para o outro lado está a outra classe.

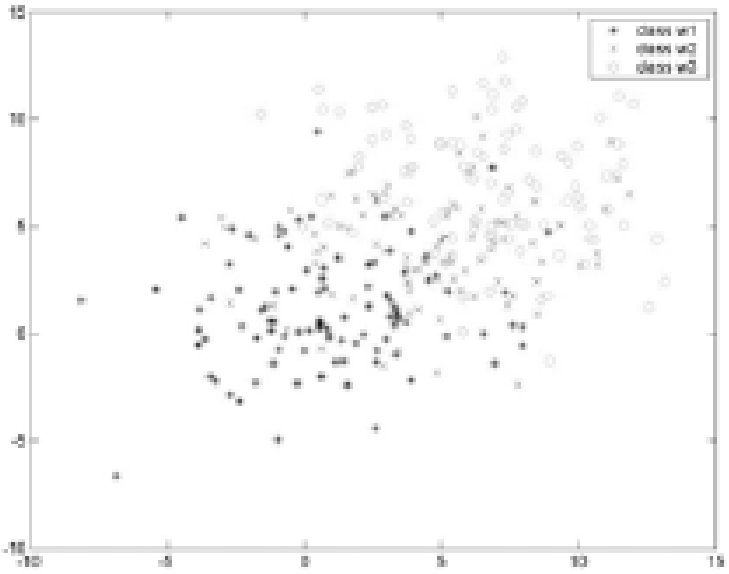
- Podemos "ver" o perceptron como uma superfície de separação em um espaço *N-dimensional* de instâncias.

- Um único perceptron consegue separar somente *conjuntos de exemplo linearmente separáveis.*

Perceptron - Características



Linearmente Separável



Linearmente Não–Separável

Perceptron - Características

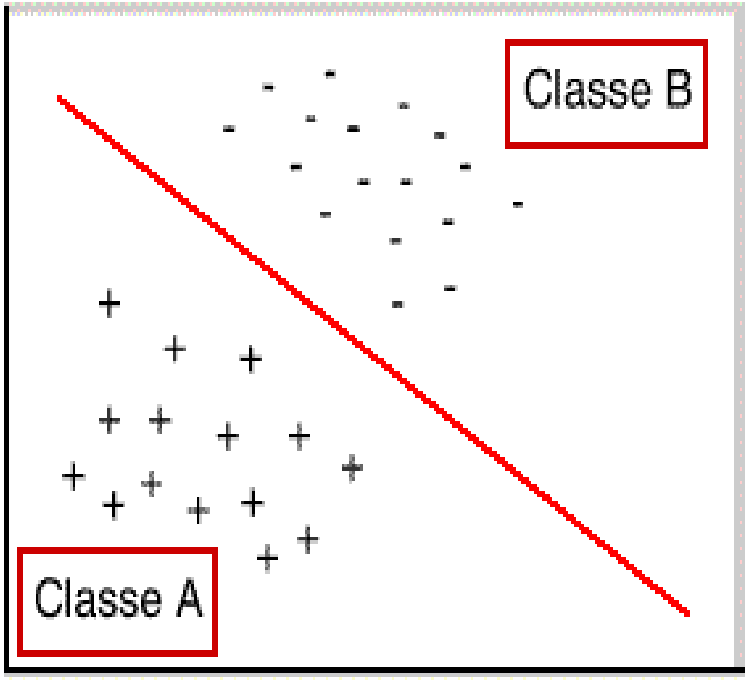


Fig. 1: Classes lineramente separáveis

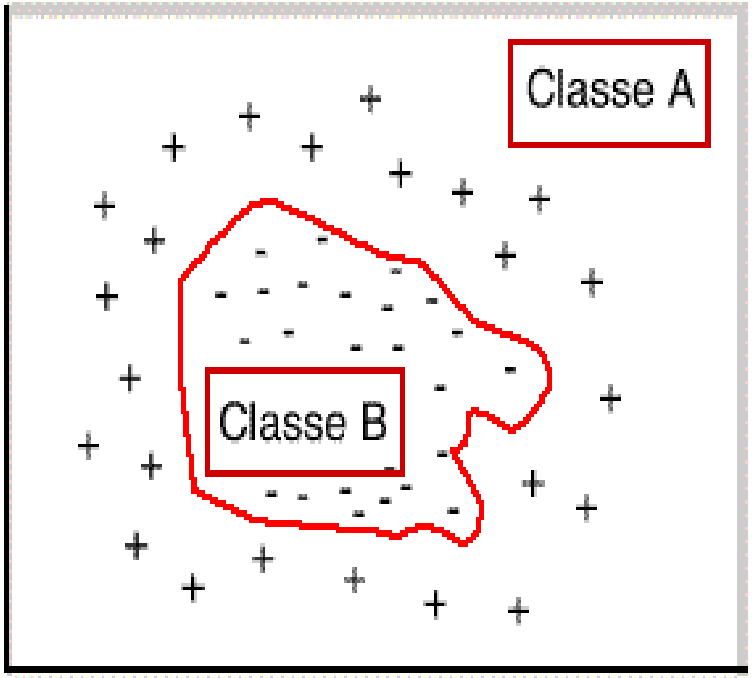


Fig. 2: Classes não lineramente separáveis

Algoritmo do Perceptron

Durante o processo de treinamento do Perceptron, busca-se encontrar um conjunto de pesos que defina uma reta que separe as diferentes classes, de forma que a Rede classifique corretamente as entradas apresentadas. Para que tal conjunto de pesos seja alcançado, ajustes são calculados segundo o algoritmo descrito a seguir:

Resumo do Algoritmo de Convergência do Perceptron

Variáveis e Parâmetros:

x(n) = vetor de entrada (m+1)-por-1
= [+1,x_1(n), x_2(n), ..., x_m(n)]^T

w(n) = vetor de pesos (m+1)-por-1
=[b(n), w_1(n), w_2(n), ..., w_m(n)]T

b(n) = bias;
A camada de entrada deve possuir uma unidade especial conhecida como bias, usada para aumentar os graus de liberdade, permitindo uma melhor adaptação, por parte da rede neural, ao conhecimento a ela fornecido.

y(n) = resposta real (quantizada)

d(n) = resposta desejada;

e(n) = erro na saída da unidade;

η = parâmetro da taxa de aprendizagem, uma constante positiva entre 0 e 1.

O valor da taxa de aprendizado define a magnitude do ajuste feito no valor de cada peso. Valores altos Fazem com que as variações sejam grandes, enquanto taxas pequenas implicam poucas variações nos pesos.
Esta magnitude vai definir a velocidade de convergência da rede.

Algoritmo do Perceptron

1 - *Inicialização:* Inicialize os pesos. Execute, então, os seguintes cálculos para os passos de tempo n = 1, 2, ...

2. *Ativação*. No caso de tempo n, ative o perceptron aplicando o vetor de entrada de valores contínuos $\mathbf{x}(n)$ e a resposta desejada d(n).

3. *Cálculo da Resposta Real*. Calcule a resposta real do perceptron:

$$y(n) = \text{função}[\mathbf{w}^{T}(n)\mathbf{x}(n)]$$

onde função(.) é a Função de Limiar.

4. *Adaptação do vetor de peso*. Atualize o vetor de peso do perceptron:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

onde

$$d(n) = \begin{cases} +1 \, se \, x(n) \; pertence \; à \; classe \; \delta_1 \\ -1 \, se \, x(n) \; pertence \; à \; classe \; \delta_2 \end{cases}$$

5. Continuação. Incremento o passo de tempo n em um e volte para o passo 2.

Algoritmo do Perceptron

- Quando devemos parar o treinamento, ou seja, parar a atualização dos pesos?
  - Escolha óbvia: continuar o treinamento até que o erro E seja menor que o valor pré-estabelecido.
  - Porém, isto implica em sobreajuste(*overfitting*).
  - O sobreajuste diminui a generalização da rede neural.

Perceptron - Conclusões

- Se um conjunto de exemplos de treinamento **E é não-separável**, então por definição não existe um vetor de pesos W que classifique corretamente todos os exemplos de treinamento em **E** utilizando o algoritmo de aprendizagem do perceptron. A alternativa mais natural é encontrar um vetor de pesos W* que classifique tantos exemplos de treinamento quanto possível de **E**. Tal conjunto de pesos é chamado de *ótimo.*

Perceptron – Implementação em Matlab

- O Matlab tem algumas funções já prontas:

- **% Apply the Percepton learning to the AND function**

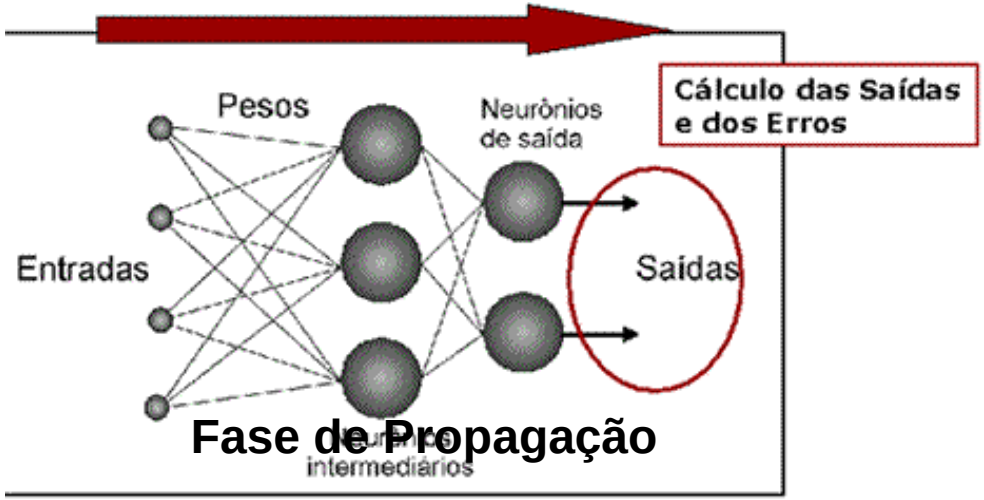
```
clear
clc
% Possible values of 2 variables in a matrix format
P = [0 0 1 1; 0 1 0 1];
%expected results
T = [0 0 0 1];

net = newp([0 1; 0 1],1);
net.trainParam.epochs = 300;
net = train(net,P,T);
simulation = sim(net,P)
```

Algoritmo Backpropagation - MLP

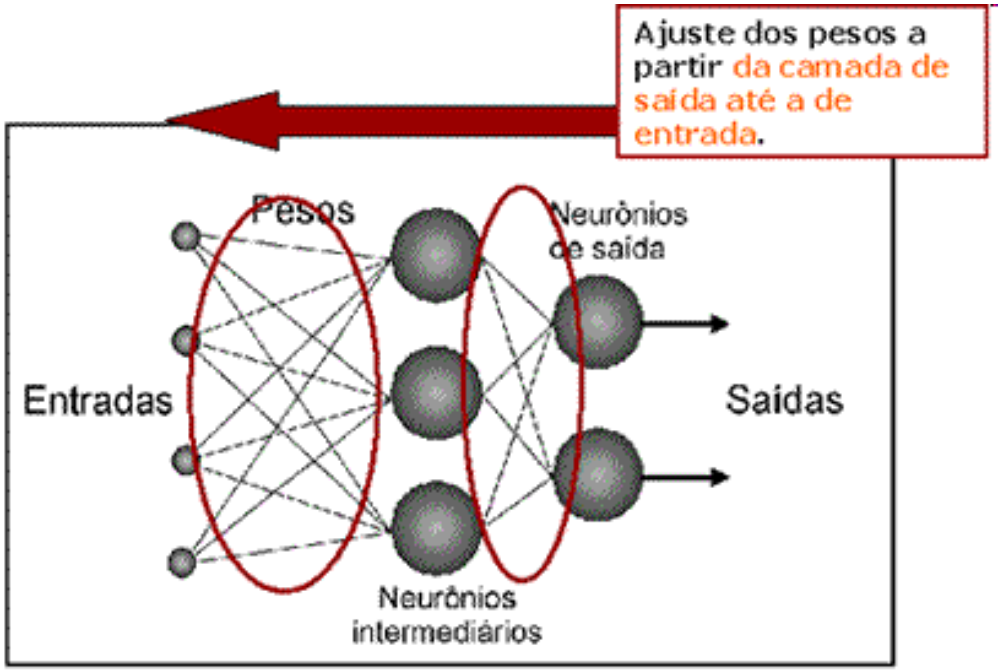
"**Backpropagation**" é um algoritmo para treinamento de Redes Multi-Camadas mais difundido. Baseia-se no **Aprendizado Supervisionado por Correção de Erros**, constituido de:

1° - Propagação: Depois de apresentado o padrão de entrada, a resposta de uma unidade é propagada como entrada para as unidades na camada seguinte, até a camada de saída, onde é obtida a resposta da rede e o erro é calculado;



Algoritmo Backpropagation - MLP

2° - Retropropagação ("backpropagation"): Desde a camada de saída até a camada de entrada, são feitas alterações nos pesos sinápticos.



Fase de Retropropagação

Algoritmo Backpropagation – MLP

Durante a **fase treinamento** devemos apresentar um conjunto formado pelo par: **entrada** para a rede e valor desejado para **saída**.

A saída será comparada ao **valor desejado** e será computado o **erro global da rede**, que influenciará na correção dos pesos no passo de retropropagação.

Apesar de **não haver garantias** que a rede forneça uma **solução ótima** para o problema, este processo é muito utilizado por apresentar uma boa solução para o treinamento de **Perceptrons Multi – Camadas**.

Fases do Backpropagation

O algoritmo **Backpropagation** é composto por duas fases:

Feed-Forward – as entradas são propagadas pela rede desde a camada de entrada até a camada de saída.

Feed-Backward – propagação dos erros obtidos desde a camada de saída da rede até à primeira camada escondida.

Backpropagation - Inicialização

1 - Inicialização: Inicialize os pesos sinápticos e os bias aleatoriamente

Backpropagation – Computação para frente

2 - Computação para frente (propagação):

Depois de apresentado o exemplo do conjunto de treinamento T = {(x(n),d(n)}, sendo x(n) a entrada apresentada à rede e d(n) a saída desejada, calcule o valor da ativação v_j e a saída para cada unidade da rede, da seguinte forma:

$$v_j = \sum_{i=1}^{m} w_{ji} x_i + b$$

para o cálculo do valor da ativação e

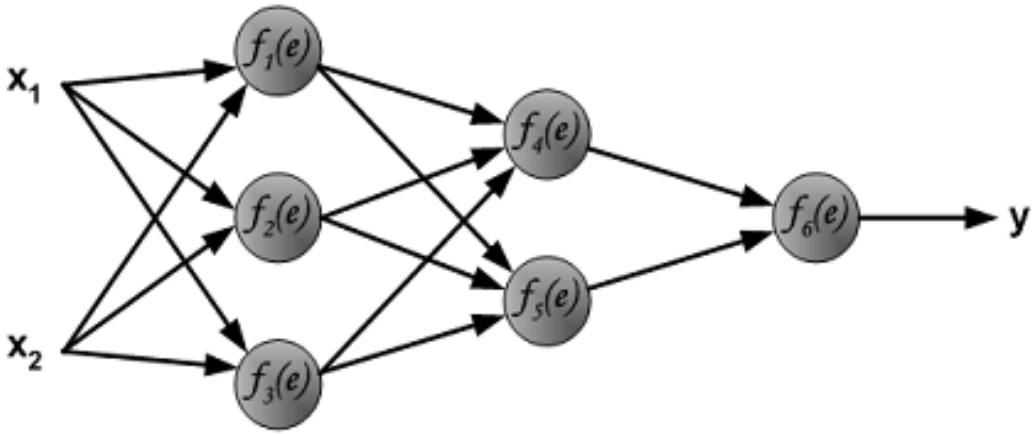
$$f(v) = \frac{1}{1+e^{(-av)}}$$

para o cálculo da saída *y* da unidade *k*, utilizando a função sigmóide, como no exemplo, ou uma outra função, se necessário.

Utilize a saída das unidades de uma camada como entradas para a seguinte, até a última camada. A saída das unidades da última camada será a resposta da rede.

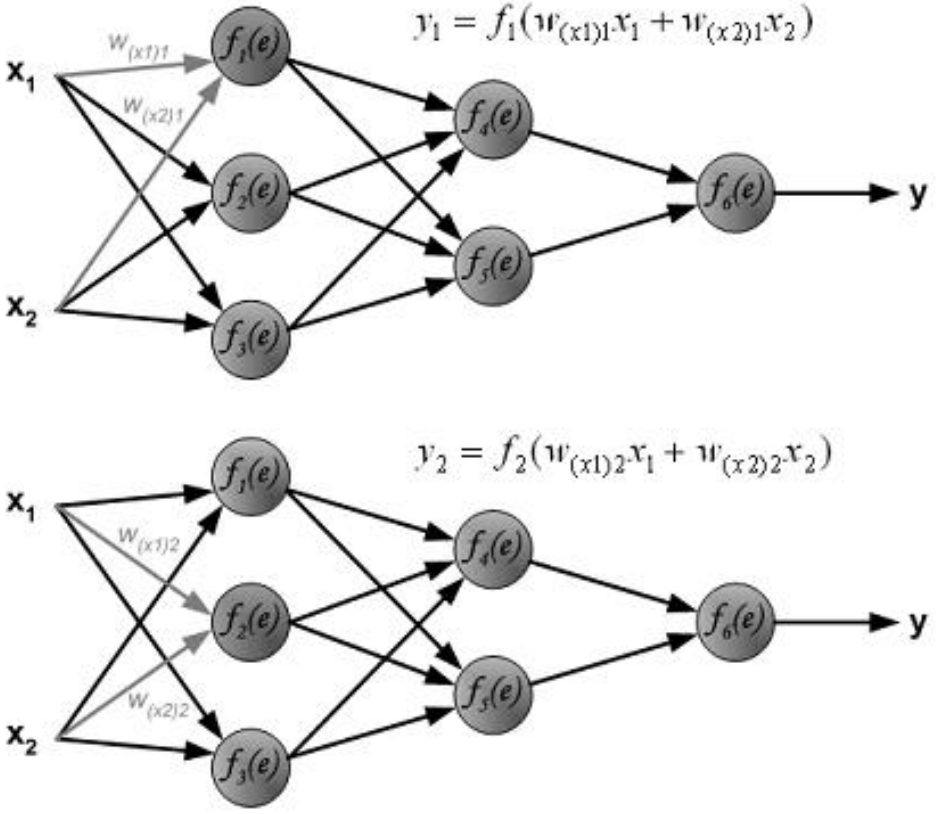
Backpropagation – Computação para frente

Exemplificação do passo 2:



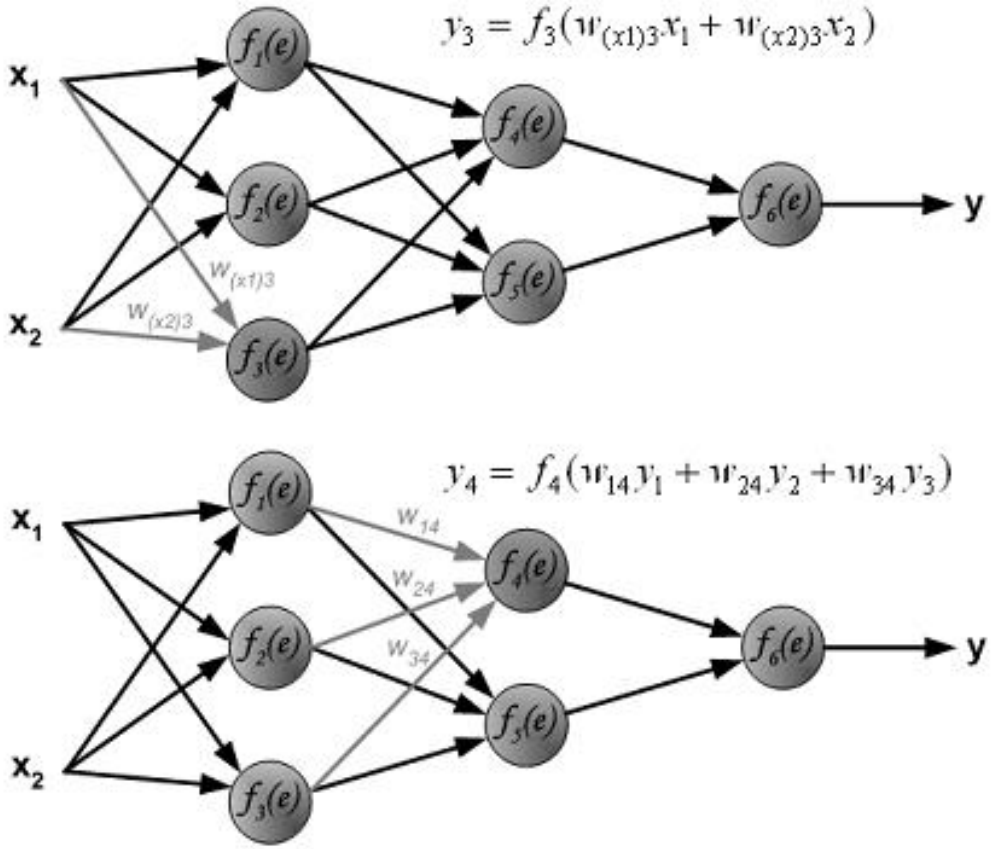
Backpropagation – Computação para frente

Exemplificação do passo 2:



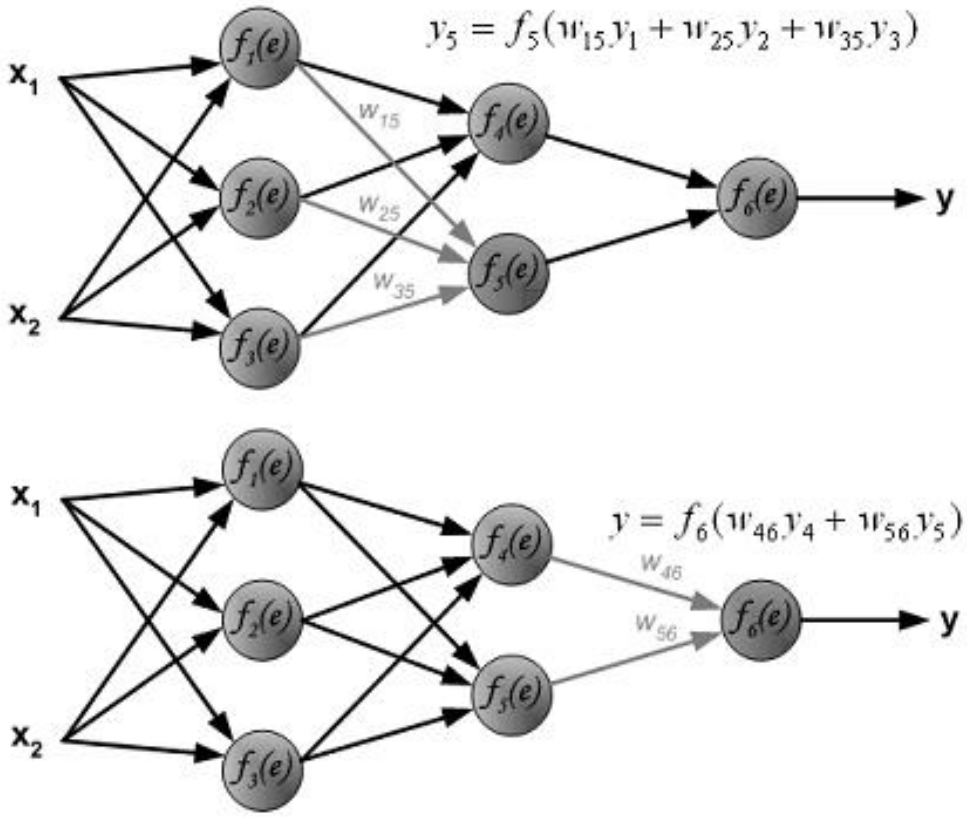
Backpropagation – Computação para frente

Exemplificação do passo 2:



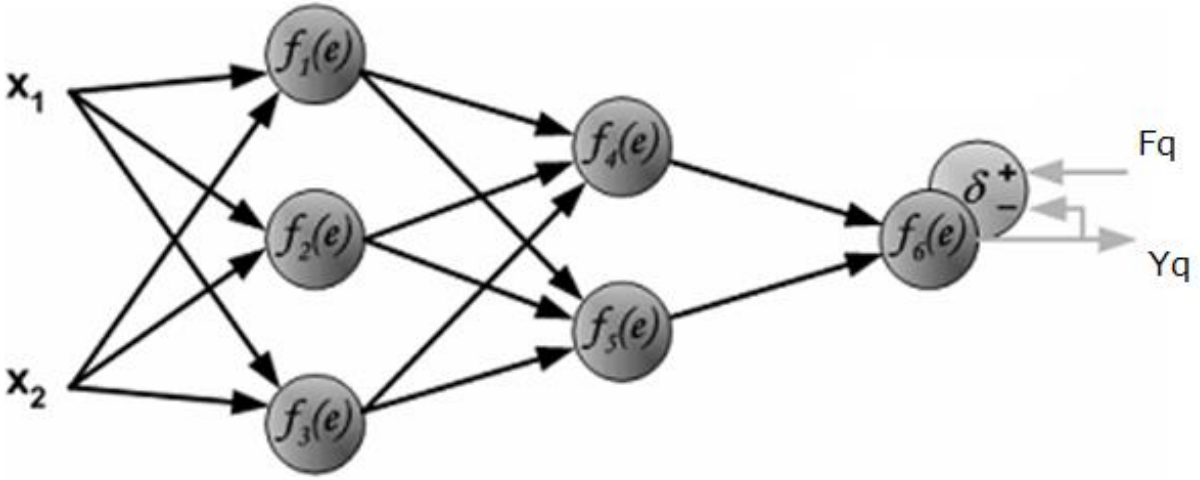
Backpropagation – Computação para frente

Exemplificação do passo 2:



Backpropagation – Computação para frente

Como calcular o erro?



Depende em qual camada o neurônio se encontra:

Se pertencer à camada de saída

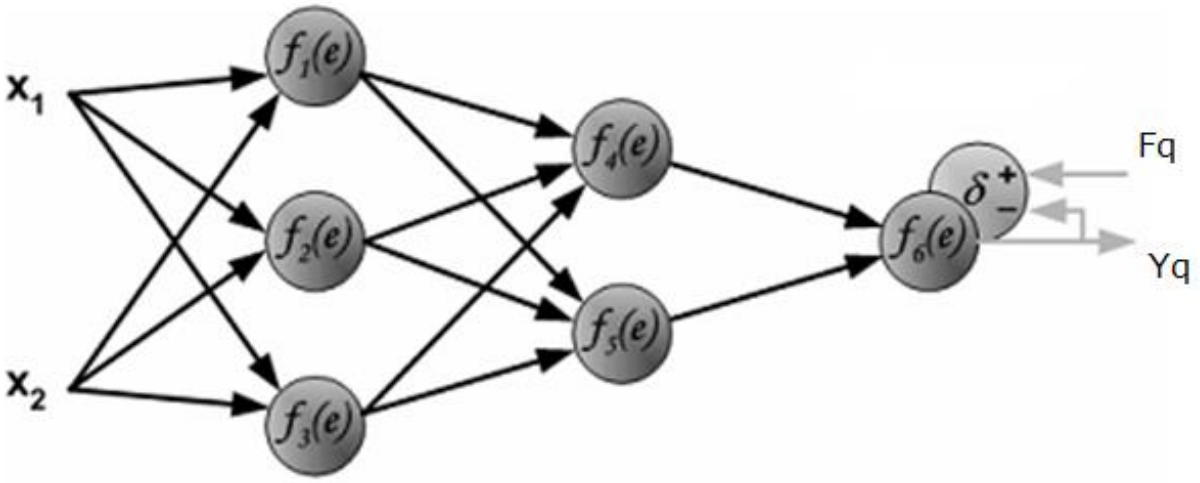
erro = derivada(função)*erro_quadrático

Sendo erro_quadrático, o erro cometido pelo neurônio de saída quando sua resposta é comparada à desejada

Erro_quadrático $= 0{,}5 * \sum_{q=1}^{k} \left(y_q - f_q\right)^2$

Backpropagation – Computação para frente

Como calcular o erro?



Se pertencer à camada intermediária:

erro = derivada(função)*pesosdasconexões*ErroDaCamadaposterior

3 - Computação para Trás (Retropropagação) – Cálculo dos erros

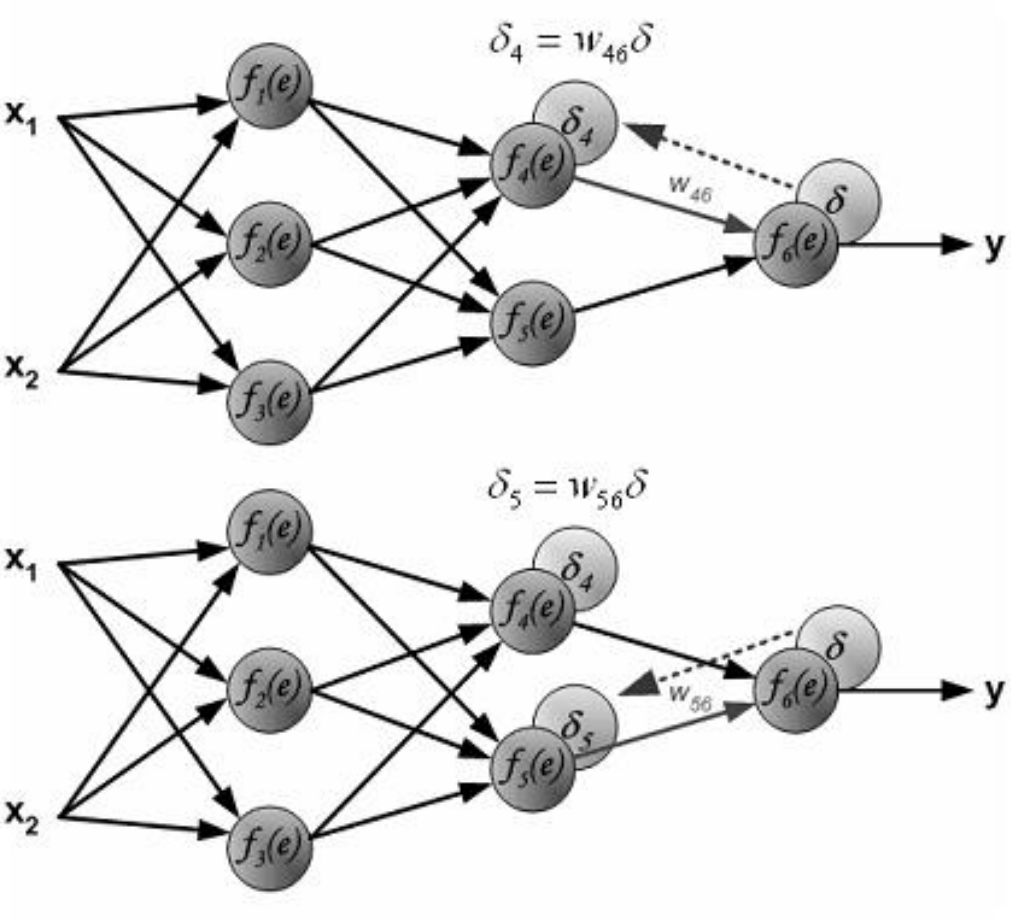
Determinar o erro nas camadas intermediárias. O erro é dado pelo soma ponderada dos erros da camada de saída multiplicado pelos pesos das respectivas ligações.

Propagar todos os erros da mesma forma para as restantes camadas intermediárias até à primeira camada escondida.

$$\delta_i = \sum_{i=1}^{N} \varpi_{ij} \delta_j$$

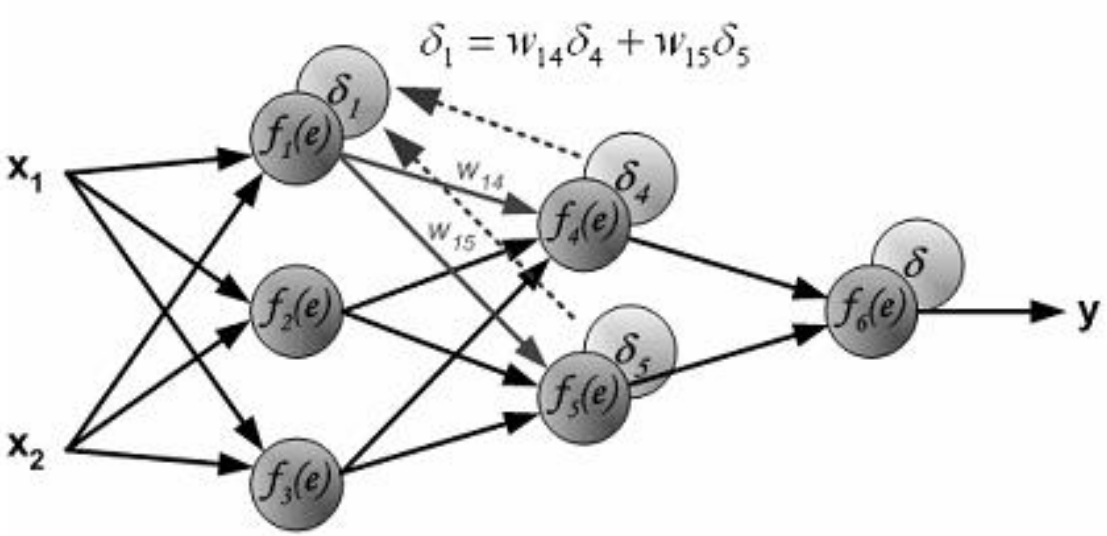
Backpropagation – propagação do erro

Exemplificação do passo 3 – propagação do erro:



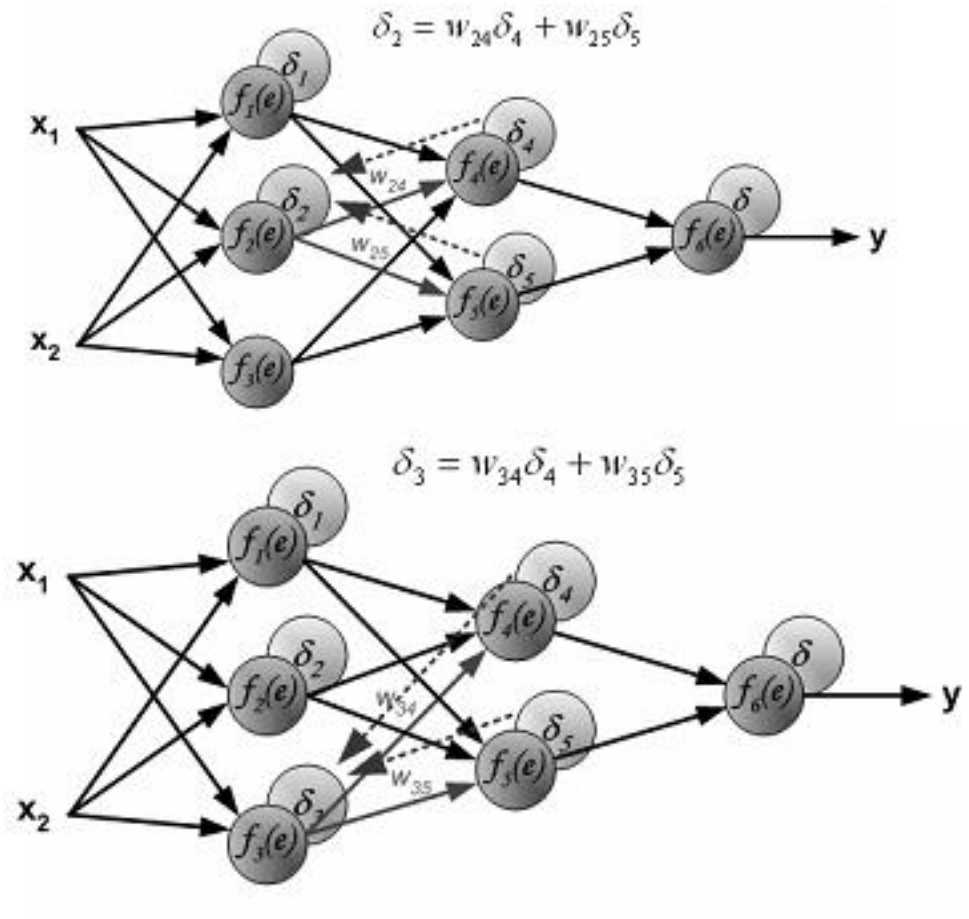
Backpropagation – propagação do erro

Exemplificação do passo 3 – propagação do erro:



Backpropagation – propagação do erro

Exemplificação do passo 3 – propagação do erro:



4 – Ajuste dos pesos

Após o cálculo dos erros é necessário corrigir os pesos das ligações entre os neurónios. O cálculo do novo peso é dado pela fórmula:

$$w_{jl}(t+1) = w_{jl}(t) + ta * xj^{*} erro_{l}$$

Onde,
W_{jl} representa o peso entre um neurônio i e o j-ésimo atributo de entrada ou a saída do j-ésimo neurônio da camada anterior

$Erro_{l}$ indica o erro associado ao l-ésimo neurônio

Xj indica a entrada recebida por esse neurônio (o j-ésimo atributo de entrada ou a Saída do j-ésimo neurônio da camada anterior

Métodos de treinamento como o *backpropagation* exigem que a função de ativação possua derivada.

Função sigmoide:

$$f(x) = \frac{1}{1+e^{-x}}$$

Derivada da função sigmoide:

$$f'(x) = x(1-x)$$

Função sigmoide:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Derivada da função sigmoide:

$$f'(x) = 1-x^2$$

Algoritmo Backpropagation - Considerações

- A atualização dos pesos é feita de forma individual para cada um dos vetores do conjunto de treinamento;

- Para cada vetor de treinamento, o sinal deve ser propagado das entradas para as saídas para que o erro possa então propagar em sentido contrário e permitir o treinamento;

- Cada apresentação completa de todos os elementos do conjunto de treinamento e consequente ajuste de pesos é chamada *epoch*;

- É aconselhável randomizar a sequência com que os vetores são apresentados à rede de uma *epoch* para a outra para acrescentar um componente estocástico ao treinamento e evitar ciclos limites indesejáveis na atualização dos pesos;

- Os pesos iniciais devem ser preferencialmente obtidos de uma distribuição uniforme (evita polarização)

Algumas regras para se pensar no número de neurônios:

- **Regra do valor médio -** De acordo com esta fórmula o número de neurônios da camada escondida é igual ao valor médio do número de entradas e o número de saídas da rede, ou seja:

$$q = \frac{p + M}{2}$$

Regra da raiz quadrada - De acordo com esta fórmula o número de neurônios da camada escondida é igual a raiz quadrada do produto do número de entradas pelo número de saídas da rede, ou seja:

$$q = \sqrt{p \cdot M}$$

Algumas regras para se pensar no número de neurônios:

- **Regra de Kolmogorov** - De acordo com esta fórmula o número de neurônios da camada escondida é igual a duas vezes o número de entradas da rede adicionado de 1, ou seja:

$$q = 2p + 1$$

Outras sugestões:

- Entre o número de neurônios nas camadas de entrada e saída
- 2/3 do tamanho da camada de entrada, somado ao tamanho da camada de saída
- Menor que duas vezes o tamanho da camada de entrada

Ferramentas

- http://www.nada.kth.se/~orre/snns-manual/
- Weka
- Knime
- Matlab
- Etc, etc

Bases de Dados

http://archive.ics.uci.edu/ml/

http://mldata.org/