# OBJECTIVO 14: SISTEMAS DE APRENDIZAGEM (MACHINE LEARNING - ML)

" A capacidade de aprender, de adaptar, de modificar o comportamento, é uma componente da inteligência humana"

"Poderemos construir máquinas realmente inteligentes, que não sejam capazes de aprender?"

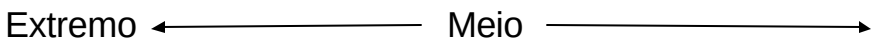
"Para muitos investigadores na área de ML, a aprendizagem é um pré-requisito para a verdadeira inteligência"

Os quatro principais paradigmas da aprendizagem:

| | |
|---|---|
| Indutivo | • aprendizagem simbólica<br>• indução da descrição de um conceito genérico a partir de exemplos e de contra-exemplos |
| Analitico | • método dedutivo em vez de indutivo<br>• utilização da experiência passada na resolução do problema para decidir qual o caminho a seguir na resolução de problemas novos<br>• formulação de regras de controlo da procura para aumentar a eficiência do conhecimento |
| Genético | • sistemas de classificação<br>• extremo empirico dos paradigmas de aprendizagem<br>• inspirado na analogia com as operações ocorridas na reprodução biológica e na selecção natural como descrita por Darwin onde sobrevivem os individuos mais fortes de cada nicho ecológico |
| Conectivista | • redes neuronais<br>• sistemas paralelos distribuidos |

Que Paradigma Escolher?

| Propriedades do domínio do problema | Melhor paradigma |
|---|---|
| Mapeamento sinal-símbolo<br>de sinais continuos em símbolos discretos | conectivista |
| Reconhecimento contínuo de padrões | conectivista<br>Indutivo e genético implicam que se resolva primeiro o mapeamento de sinal-símbolo |
| Reconhecimento discreto de padrões | indutivo<br>genético |
| Aquisição de descrições de novos conceitos | indutivo |
| Aquisição de regras para sistemas periciais | indutivo<br>genético |
| Aumento de eficiência de sistemas baseados em regras | analitico |
| Raciocínio simbólico | indutivo<br>analitico |




Conectivista

muitos exemplos de
para treino
reconhecimento em
domínios contínuos
não estruturados
características
escondidads

Indutivo e Genético

Analitico

domínios
conhecimento
estruturado
raciocínio mais
profundo
inferência em
múltiplos passos

OBJECTIVO 15: PROGRAMAS EVOLUTIVOS

- Sistemas de resolução de problemas baseados nos principios da evolução e hereditariedade: ALGORITMOS GENÉTICOS etc.
- Algoritmo probabílistico
- Mantém uma população de individuos P(t) = $\{x_1^t,...,x_n^t\}$ para a iteração t
- Cada individuo representa uma potencial solução para o problema
- Cada solução $x_i^t$ é avaliada para se ter uma medida do se desempenho
- Uma nova população é gerada a partir da anterior por selecção dos melhores individuos e aplicação de operadores genéticos

***PROGRAMA EVOLUTIVO***

***proc*** *PE º*

***seja*** *P(t) a população de individuos no ciclo de tempo t*

***seja*** *t o ciclo de tempo actual*

*t ¬ 0*

*inicializar P(0)*

*avaliar P(0)*

***enquanto*** *não(condição de terminação)* ***fazer***

*t ¬ t+1*

*seleccionar P(t) a partir de P(t-1)*

*recombinar P(t)*

*avaliar P(t)*

***fimenq***

## OPERADORES GENÉTICOS

### Selecção

A selecção baseia-se no método da roleta ("roulette wheel selection") é o operador genético utilizado para seleccionar os classificadores mais fortes para o processo de reprodução (companheiros e pais) [Goldberg, 1989]. Este método pode ser descrito pelo seguinte algoritmo:

1 - somar em *Sum* o valor *ST* de todos os *n* membros da população (os cromossomas);

$$Sum = \sum_{i=1}^{n} ST_i$$

2 - gerar um número aleatório *R* compreendido entre *0* e *Sum*;

3 - devolver o primeiro membro da população $m \in P(t)$, cujo valor $ST_m$, adicionado ao valor dos vários membros da população precedente, é maior ou igual a *R*:

$$\sum_{i=1}^{m} ST_i \geq R$$

O efeito deste método é a devolução, de forma aleatória, de um cromossoma seleccionado.

## RECOMBINAÇÃO

O processo de recombinação é baseado na aplicação de dois operadores: a mutação e cruzamento.

### Mutação

A mutação é uma operação realizada sobre uma string de caracteres A string é percorrida totalmente e, cada caracter que for encontrado é substituído por outro caracter, seleccionado de forma aleatória dentro do alfabeto definido, no caso de se ter ultrapassado um teste de probabilidade com sucesso. A probabilidade de ocorrer a mutação é $p_m$, um parâmetro do algoritmo genético.

$$0 \rightarrow 1 \rightarrow 0$$

Mutação de um bit:

**PROC** MUTAÇÃO ≡
***seja*** bit o bit a mutar
***seja*** mut a indição se deve ocorrrer a mutação
mut ← falso
***se*** $p_m$ = 1.0 ***então***
mut ← verdadeiro
***senão***
mut ← (random ≤ $p_m$)
***fimse***
***se*** mut ***então***

bit ← not bit

***fimse***

**Cruzamento**

- O operador de cruzamento num único ponto (“one point crossover”) é utilizado por forma a recombinar o material genético que as duas strings dos pais possuem para que se consiga obter dois filhos.
- Este método ocorre quando as partes das strings dos pais são trocadas após a selecção aleatória de um ponto ("sitecross"), criando-se assim dois filhos. Este operador é aplicado com uma probabilidade $p_c$.

Ponto de cruzamento na posição 3

| | | | |
|---|---|---|---|
| Cromossoma 1: 1 0 1 \| 1 0 1 | ⇒ | Filho 1: | 1 0 1 1 0 0 |
| Cromossoma 2: 0 0 1 \| 1 0 0 | ⇒ | Filho 2: | 0 0 1 1 0 1 |

CRUZAMENTO DE DOIS CROMOSSOMAS:

**PROC** CRUZAMENTO ≡
***seja*** crom1 e crom2 os cormossomas a cruzar de tamanho l
***seja*** cruz a indição se deve ocorrrer o cruzamento
cruz ← falso
***se*** $p_c$ = 1.0 ***então***
cruz ← verdadeiro
***senão***
cruz ← (random ≤ $p_c$)
***fimse***
***se*** cruz ***então***
sitecross ← random(1,l)
cruzar crom1 com crom2 no ponto sitecross
***fimse***

- Método adaptativo de procura num espaço de soluções por aplicação de operadores modelados de acordo com a herança e simulação da teoria “Darwiniana” da sobrevivência
- Geralmente, um algoritmo genético executa uma procura multi-direccional, e permite a formação de informação e troca entre essas direcções
- Mantém uma população de soluções propostas (cromossomas) para um dado problema
- Cada solução é representada num alfabeto fixado (normalmente binário) com um significado estabelecido.
- A população segue uma evolução simulada: as soluções relativamente ‘boas’ produzem descendentes, que subsequentemente substituem as ‘piores’
- A estimativa da qualidade de uma solução é baseada numa função de avaliação, que representa um dado ambiente

A especificação de um algoritmo genético para um problema particular envolve a descrição de um conjunto de componentes. Destes é de salientar:

- Uma representação genética das soluções potenciais para o problema, que também definem o espaço de soluções do problema (cromossomas).
- Um método de gerar a população inicial de soluções potenciais.
- Uma função de avaliação que simule o ambiente, e atribua valores relativos às soluções em termos da sua utilidade (valor) ou adaptabilidade ao ambiente.
- Operadores genéticos que alterem a composição dos cromossomas durante a recombinação.
- Valores para os vários parámetros que o algoritmo genético usa (tamanho da população, probabilidades associadas aos operadores genéticos, condição de terminação, etc.).

## Porque Funcionam Os Algoritmos Genéticos

Os fundamentos teóricos dos algoritmos genéticos baseiam-se na noção de esquema [Goldberg, 1989] - um formato de semelhança que permite uma exploração de semelhanças ao longo dos cromossomas. Um esquema *H* é construido pela introdução de um novo símbolo, o caracter universal # no alfabeto dos genes - tal esquema representa todas as strings (um hiperplano, ou sub-conjunto do espaço de procura) que unifica em todas as posições diferentes de #. Assumindo que o alfabeto é binário e uma população de tamanho *n*, existem entre $2^n$ e $n \cdot 2^n$ esquemas diferentes representados; pelo menos $n^3$ destes são processados ao mesmo tempo - Holland chamou a esta propriedade 'paralelismo implicito', uma vez que é conseguido sem necessidade de recorrer a memória ou processamento extras.

Vejamos então como é que os algoritmos genéticos propagam a construção de blocos. Para isso, temos de fazer algumas definições iniciais:

- *POP(t)* é uma população de n cromossomas na iteração *t*, e cada cromossoma é uma string de tamanho *k* no alfabeto binário *{0, 1}*.
- *H* é um esquema, ou seja, uma string no alfabeto $\{0, 1, \#\}^k$. Por exemplo, $H_1$=(0##1100##) é um esquema na população de strings de tamanho *k*=9. O esquema $H_1$ representa o conjunto de todas as strings que é possível obter substituindo # ora por 0 or por 1. Neste caso, $H_1$ representa um conjunto de $2^4$=16 strings.

- Ordem do esquema, *o(H)*, é o número de símbolos menos o número de posições # presentes numa string. Essencialmente, esta define a especificidade de um esquema. Neste exemplo, $o(H_1)$=5.
- Comprimento de um esquema, *l(H)*, é a distância entre os dois símbolos mais distantes no esquema que sejam diferentes de #. Define o grau de compactação da informação contida num esquema. Neste exemplo, $l(H_1)$=6.

Vamos agora definir o efeito que cada operador genético tem na construção de blocos. O resultado é chamado o teorema do esquema [Goldberg, 1989].

***Teorema 2.1 (Teorema do esquema)***. Este teorema também conhecido como teorema fundamental dos algoritmos genéticos, diz essencialmente que esuemas curtos com desempenhos acima da média (construção de blocos) aumentam em número com velocidade exponêncial. Pode ser traduzido pela fórmula:

$$m(H,t+1) = m(H,t)\frac{f(H,t)}{f(t)}\left[1 - p_c \frac{l(H)}{k-1}\right] o(H) p_m$$

onde *m(H,t)* é o número de esquemas *H* na população *P(t)*, *f(H,t)* é o desempenho médio do esquema *H* na iteração *t*, e *f(t)* o desempenho médio da população *POP(t)*:

$$\overline{f(t)} = \frac{\sum_{i=1}^{n} ST_i}{n}$$

onde $ST_i$ corresponde ao valor de um cromossoma *i* na população *POP(t)*. $p_c$ corresponde à probabilidade associada ao operador de cruzamento (uniformemente distribuída) e $p_m$ à probabilidade de mutação.

**Como Funcionam os Algoritmos Genéticos**

Vejamos um exemplo de um algoritmo genético em funcionamento (figura 2.4). Para isso consideremos uma função multimodal de uma variável *f(x)*, com um dado domínio $x \hat{I} [a,b]$. Usando o alfabeto binário, codificamos esta variável como um cromossoma. O número de bits nesta representação depende do domínio da variável *(b-a)* e da precisão desejada. Assumindo um comprimento *k*=10 bits, um cromossoma (e consequentemente, uma solução potencial) pode ter o seguinte aspecto:

0011101011

Claro que, para avaliar cada número binário, devemos primeiro convertê-lo para o seu correspondente valor decimal (235 neste caso) e então escalá-lo no domínio *(b-a)*. Por exemplo, o cromossoma anterior corresponde ao valor:

$$avaliação = 235 \cdot \frac{b-a}{2^{10}-1}$$

A seguir, devemos definir o tamanho da população (*n*=20 neste caso), e preenchê-la com cromossomas gerados aleatoriamente (*POP(0), t=0*). Esta inicialização corresponde a uma amostragem

do espaço de soluções. Avaliamos as amostras, e depois seleccionamos uma nova população composta pelos cromossomas de maior valor com a maior probabilidade de ocorrência, e aplicámos os operadores genéticos para produzir novos exemplares. Eis dois exemplos de uma possível mutação e cruzamento:

mutação: $0011\underline{1}101011 \rightarrow 001\underline{0}101011$

cruzamento: $00 \mid 11101011 \rightarrow 0000000010$

$11 \mid 00000010 \rightarrow 1111101011$

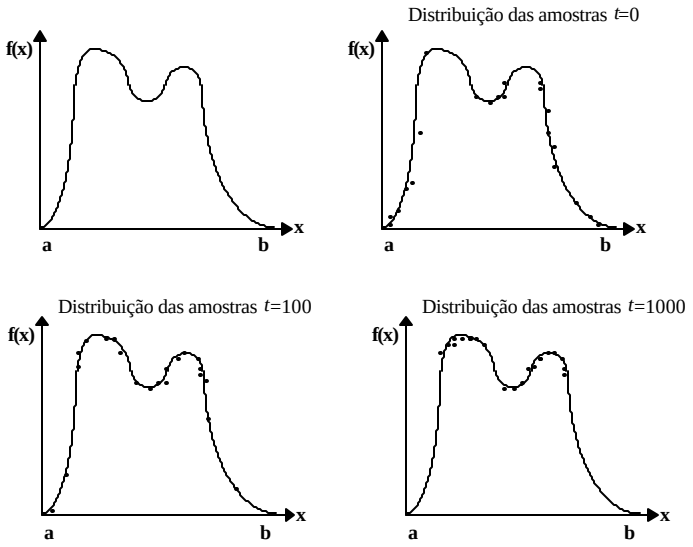



***Figura 2.4 Exemplo de execução de um algoritmo genético.***

Iteramos o processo de avaliação/selecção/recombinação durante um certo número de ciclos. Após algumas destas iterações, os exemplares concentram-se à volta do sub-espaço com maior valor associado (figura 2.4 após 100 e 1000 iterações). Com um número suficiente de iterações a solução pode ser encontrada (ou

aproximada, geralmente).

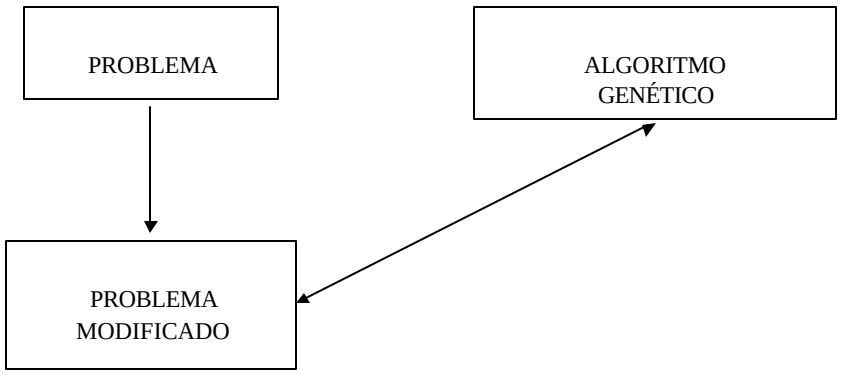
# RESOLUÇÃO DE PROBLEMAS EM INTELIGÊNCIA ARTIFICIAL

## ALGORITMOS GENÉTICOS / PROGRAMAS EVOLUTIVOS

## DIFERENÇAS CONCEPTUAIS

### ABORDAGEM GENÉTICA (ALGORITMO GENÉTICO)

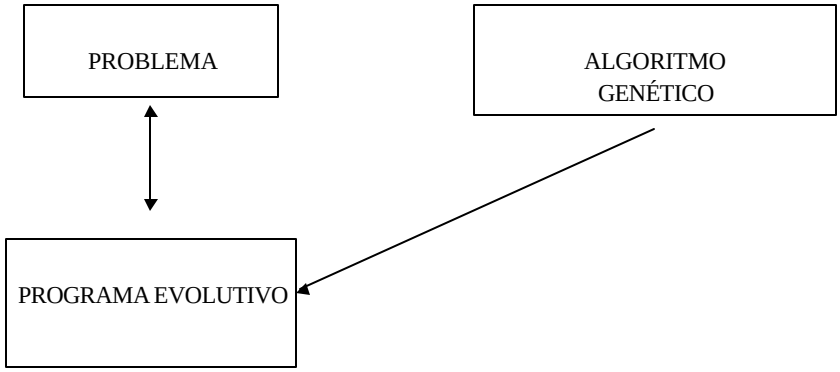
"o problema é adaptado para utilizar o algoritmo genético"




- Representação das soluções através de strings binárias
- Requere a modificação do problema para mapear as soluções em strings binárias, codificadores, algoritmos reparadores (soluções inválidas) etc.
- Geralmente uma tarefa difícil
- Não incorporam conhecimento específico acerca do problema
- Utilizam os operadores genéticos convencionais

## ABORDAGEM EVOLUTIVA (PROGRAMA EVOLUTIVO = ALGORITMO GENÉTICO ADAPTADO)

"o algoritmo genético é adaptado ao problema"




- Representação das soluções através de estruturas de dados adequadas ao problema
- Não requerem a modificação do problema
- Incorporam conhecimento específico acerca do problema utilizando operadores genéticos desenhados para o problema

A especificação de um programa evolutivo/algoritmo genético para um problema particular envolve a descrição de um conjunto de componentes. Destes é de salientar:

- Uma representação genética das soluções potenciais para o problema, que também definem o espaço de soluções do problema.

  Abordagem genética: desenvolver procedimentos para codificação das soluções em strings binárias (cromossomas) e sua descodificação
- Um método de gerar a população inicial de soluções potenciais.
- Uma função de avaliação que simule o ambiente, e atribua valores relativos às soluções em termos da sua utilidade (valor) ou adaptabilidade ao ambiente.
- Operadores genéticos que alterem a composição das soluções (cromossomas) durante a recombinação.
- Valores para os vários parámetros que o algoritmo usa (tamanho da população, probabilidades associadas aos operadores genéticos, condição de terminação, etc.).

# O PROBLEMA DE TRANSPORTE LINEAR

## ALGORITMO DE INICIALIZAÇÃO

***proc*** INI≡
***seja*** dest[k] o array de quantidades encomendadas
***seja*** sour[n] o array de quantidades fornecidas
***seja*** X[n,k] a matriz das quantidades a transportar
***seja*** visit[n*k] o array das posições de X já visitadas

atribuir a todas as posições do array visit o valor “não visitado”
***repetir***
    seleccionar aleatoriamente uma posição q ∈ [1, k*n] no array visit que não tenha sido visitada e marcar essa posição como visitada

    i ← round((q-1)/k + 1)
    j ← (q-1) mod k + 1
    val ← min(sour[i], dest[j])
    X[i,j] ← val
    sour[i] ← sour[i] - val
    dest[j] ← dest[j] - val
***até*** todas as posições do array visit terem sido visitadas

OPERAÇÃO DE MUTAÇÃO (MUTATION)

***proc*** MUTAÇÃO≡
***seja*** a matriz V[n,k] o parente para mutação
***seja*** dest[k] o array de quantidades encomendadas para V
***seja*** sour[n] o array de quantidades fornecidas para W
***seja*** W[p,q] uma sub-matriz de V
***seja*** $dest_W$[q] o array de quantidades encomendadas para W
***seja*** $sour_W$[p] o array de quantidades fornecidas para W

gerar aleatoriamente um número p ∈ [2, n]
gerar aleatoriamente um número q ∈ [2, k]
seleccionar aleatoriamente p linhas em V e copiá-las para W
seleccionar aleatoriamente q colunas em V e copiá-las para W
calcular os valores para

$sour_W[i] = \sum_{(j=1..q)} W_{i,j}$ i=1..p

$sour_W[j] = \sum_{(i=1..p)} W_{i,j}$ j=1..q

inicializar W com o proc INI (para respeitar as restrições)
substituir os elementos apropriados de V com os novos elementos de W

OPERAÇÃO DE CRUZAMENTO (CROSSOVER)

***proc*** CRUZAMENTO≡
***seja*** V1[n,k] e V2[n,k] as matrizes para cruzamento
***seja*** V3[n,k] e V4[n,k] as matrizes resultantes do cruzamento
***seja*** DIV[n,k] uma matriz com as médias de V1 e V2 arredondadas
***seja*** REM[n,k] uma matriz com as diferenças de arredondamento de DIV
***seja*** REM1[n,k] e REM2[n,k] matrizes tal que REM = REM1+REM2

preencher a matriz DIV da seguinte forma:

$div_{i,j} \leftarrow round(v1_{i,j} + v2_{i,j}) / 2)$

preencher a matriz REM da seguinte forma:

$rem_{i,j} \leftarrow (v1_{i,j} + v2_{i,j}) \bmod 2$

preencher as matrizes REM1 e REM2 distribuindo os 1's de REM igualmente por REM1 e REM2 (linha a linha ou coluna a coluna)

produzir as matrizes V3 e V4 da seguinte forma:

V3 = DIV + REM1
V4 = DIV + REM2

# OBJECTIVO 16: APRENDIZAGEM POR INDUÇÃO

## APRENDIZAGEM ATRAVÉS DE EXEMPLOS

- O professor dá exemplos para aprendizagem e o aluno faz generalizações acerca dos exemplos, i.e. tenta encontrar uma espécie de teoria inerente aos exemplos.

- O professor pode ajudar os alunos indicando quais são os bons exemplos e descrevendo estes exemplos numa linguagem que permita a formulação conveniente de regras genéricas.

- Exemplos:

  diagnóstico de uma doença num paciente;

  prever o tempo;

  prever o comportamento de novos compostos químicos;

  controlo de um sistema dinâmico;

  aquisição de conhecimento para um sistema pericial;

  etc.

# DESCRIÇÃO FORMAL DA APRENDIZAGEM DE CONCEITOS ATRAVÉS DE EXEMPLOS

## CONCEITOS COMO CONJUNTOS DE OBJECTOS

- Seja U o conjunto universal de objectos (que é possível encontrar).
- Um conceito C é um formalmente um subconjunto de objectos de U.
- Aprender o conceito C corresponde a reconhecer objectos em C, isto é para todos o objecto X em U reconhecer que X está em C.

## EXEMPLOS

O conceito de arco no mundo dos blocos:

- U é o conjunto das estruturas feitas por blocos do mundo dos blocos
- Arco é um subconjunto de U contendo todas as estruturas do tipo arco e nada mais

O conceito da multiplicação:

- U é o conjunto dos tupletos de números
- Multiplicação é o conjunto de todos os tripletos de números (a,b,c) em que a*b=c:

$$\text{Multiplicação} = \{(a,b,c) \mid a*b = c\}$$

O conceito de uma determinada doença:

- U é o conjunto de todas as descrições dadas pelos pacientes em termos de algum repertório de características preestabelecido.
- Doença é o conjunto de todas as descrições dos pacientes que sofrem da doença em questão.

## LINGUAGENS PARA DESCRIÇÃO DE OBJECTOS E CONCEITOS

### Descrições relacionais (estruturadas)

Um objecto é descrito através das suas componentes e das relações entre estas.

Exemplo do arco:

estrutura com 3 componentes

(2 postes e 1 barra)

cada componente é um bloco

ambos os postes seguram a barra

os postes não se tocam

### Descrições através de atributos

Descrição de um objecto através das sua características como:

comprimento, altura, cor etc.

Algumas linguagens de representação de conhecimento que podem ser usadas em programas de aprendizagem:

- vectores de atributos para representar objectos
- regras do tipo se-então para representar conceitos
- árvores de decisão para representar conceitos
- redes semânticas
- lógica de predicados

Uma vez definidos os objectos e os conceitos necessitamos de regras de decisão que indiquem até que ponto um objecto pertence a um conceito.

## O PROBLEMA DE APRENDER A PARTIR DE EXEMPLOS

O objectivo da aprendizagem é a descrição de conceitos a partir de exemplos.

Dado um conjunto S de exemplos (conjunto de treino) encontrar uma fórmula F expressa através da linguagem de descrição de conceitos escolhida, tal que:

Para todos os objectos X

1. Se X é um exemplo positivo em S então X unifica F
2. Se X é um exemplo negativo em S então X não unifica F

F é aquilo que o sistema de aprendizagem percebeu acerca do conceito C.

Não existem garantias de que F corresponda a outros objectos novos (extra conjunto de treino)

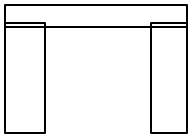
O principal objectivo é de classificar correctamente objectos não analisados.

A descrição de F deve ser mais genérica do que os exemplos de treino positivos em si.

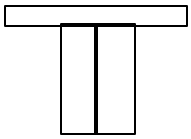
APRENDIZAGEM ACERCA DE ESTRUTURAS EM ARCO: UM EXEMPLO

OBJECTIVO: Aprender o conceito de arco a partir de exemplos e contra-exemplos

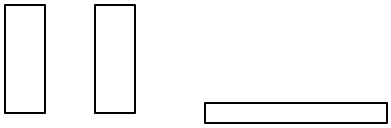
ARCO



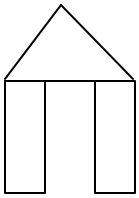
NÃO É ARCO



NÃO É ARCO



ARCO



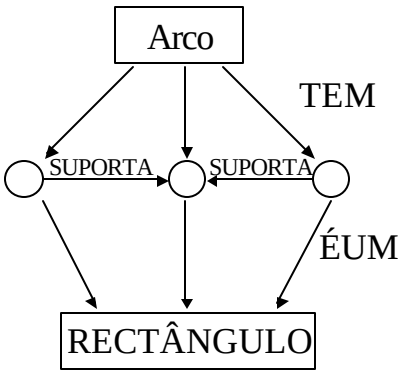
## ALGORITMO DE APRENDIZAGEM

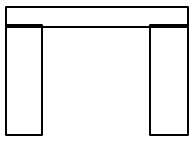
Para aprender o conceito C a partir de uma sequência de exemplos E1, E2,..., En (E1 deve ser um exemplo positivo)

1. Adoptar E1 como a hipótese inicial acerca de C
2. Processar todos os restantes exemplos Ei (i=2,...,n) fazendo para cada:
   - Unificar a hipótese corrente Hi-1 com Ei; o resultado é a descrição D das diferenças entre Hi-1 e Ei
   - Actuar em Hi-1 de acordo com D e com o facto de Ei ser um exemplo positivo ou negativo de C; o resultado é o refinamento de Hi acerca de C

## COMO REPRESENTAR ESTES OBJECTOS?

Por exemplo, através de redes semânticas



# APRENDIZAGEM DE DESCRIÇÕES SIMPLES DEFINIDAS POR ATRIBUTOS

Descrição de OBJECTOS e CONCEITOS através de conjuntos de atributos

Tipos de atributos:

Numéricos

Não-numéricos: ordenados

não-ordenados

EXEMPLO

Classificar objectos a partir da silhueta captada por uma câmara de filmar. Estas silhuetas são definidas em termos de um conjunto de atributos:

| **Atributo** | **Valores Possíveis** |
|---|---|
| tamanho | pequeno, grande |
| forma | longa, compacta, outra |
| buracos | 0, 1, 2, 3, muitos |

Os exemplos podem então ser representados em lógica de predicados da seguinte forma:

exemplo(Classe, [Atributo1 = Valor1, Atributo2 = Valor2,...]).

Imagens da câmara para alguns objectos

Representação destes exemplos:

exemplo(porca, [tamanho=pequeno, forma=compacta, buracos=1]).

exemplo(parafuso, [tamanho=pequeno, forma=longa, buracos=0]).

exemplo(chave, [tamanho=pequeno, forma=longa, buracos=1]).

exemplo(porca, [tamanho=pequeno, forma=compacta, buracos=1]).

exemplo(chave, [tamanho=grande, forma=longa, buracos=1]).

exemplo(**parafuso**,[tamanho=pequeno,forma=compacta,buracos=0])

.

exemplo(porca, [tamanho=pequeno, forma=compacta, buracos=1]).

exemplo(lápis, [tamanho=grande, forma=longa, buracos=0]).

exemplo(tesoura,[tamanho=grande, forma=longa, buracos=2]).

exemplo(lápis, [tamanho=grande, forma=longa, buracos=0]).

exemplo(tesoura,[tamanho=grande, forma=outra, buracos=2]).

exemplo(chave, [tamanho=pequeno, forma=outra, buracos=2]).

Se estas cláusulas forem comunicadas a um programa de aprendizagem, o resultado da aprendizagem será um conjunto de descrições de classes (conceitos) na forma de regras se...então que poderão ser usadas para classificar novos objectos

Classe <== [Conj1, Conj2,...]

onde Conj1, Conj2, etc. são listas de pares atributo=valor da forma:

[Atributo1=Valor1, Atributo2=Valor2,...]

Por exemplo as regras induzidas (conceitos) para porca e chave:

porca <== [[tamanho=pequeno, buracos=1]]
chave <== [[forma=longa, buracos=1], [forma=outra, buracos=2]]

UNIFICAÇÃO DE UM OJECTO COM UM CONCEITO

1. Um objecto unifica com o conceito se satisfaz pelo menos uma lista Conj1, Conj2,...
2. Um objecto satisfaz uma lista de valores de atributos Conj se todos os valores dos atributos em Conj estão tal e qual no objecto

Exemplo: o objecto descrito pela lista de atributos

[tamanho=pequeno, forma=longa, buracos=1]

unifica com o conceito de chave

Procedimento de unificação:

```
unifica(Objecto, Classe <== Descrição) ←
                                membro(Conj, Descrição),
                                não(  membro(Atrib=Val, Conj),
                                      membro(Atrib=ValX, Ojecto),
                                      ValX \== Val).
```

Os objectos podem ser parcialmente definidos, os valores de alguns atributos podem não ser definidos!

Exemplos:

```
← unifica([tamanho=pequeno, forma=longa, buracos=1],
          chave<==[[forma=longa,buracos=1],
                   [forma=outra,buracos=2]])
```

resposta: yes

```
← unifica([tamanho=pequeno, forma=longa, buracos=1],
          porca <== [[tamanho=pequeno, buracos=1]])
```

resposta: yes

## INDUÇÃO DOS CONCEITOS
## (DESCRIÇÃO DAS CLASSES DE OBJECTOS)

- No caso anterior (descrições relacionais) processamos um exemplo de cada vez sequencialmente
- Aqui vamos processar todos os exemplos de uma só vez!
- O objectivo principal é que a descrição de um conceito cubra (unifique com) todos os exemplos positivos pertencentes à classe e não cubra nenhum exemplo negativo

## ALGORITMO DE APRENDIZAGEM

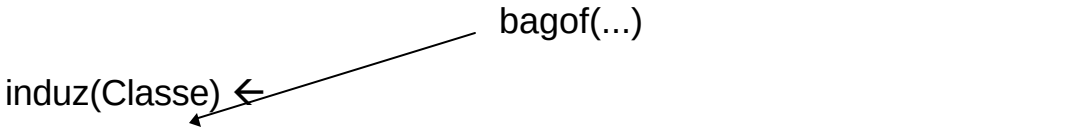
aprender(Exemplos, Classe, Descrição)

Para cobrir todos os exemplos de uma classe Classe a partir de Exemplos fazer:

se nenhum exemplo pertence a Classe
então
    Descrição = []
senão
    Descrição = [Conj | Conjs]  (menor nº de listas o mais pequenas possível)
    e obter Conj e Conjs da seguinte forma:
    1. construir uma lista Conj de valores de atributos que cubra pelo menos um exemplo positivo da Classe e nenhum exemplo das outras classes

2. remover dos Exemplos todos os objectos cobertos por Conj e cobrir os restantes objectos não cobertos pela descrição Conjs repetindo o passo 1

Procedimento de INDUÇÃO




soluções(exemplo(ClasX,Atr),exemplo(ClasX,Atr),Exemplos),
aprender(Exemplos, Classe, Descrição),
write(Classe), write('<=='), writelist(Descrição),
assert(Classe <== Descrição).

Exemplos:

← induz(tesoura)

Exemplos positivos

| tamanho | forma | buracos | colide? |
|---|---|---|---|
| Grande | longa | 2 | Não |
| Grande | outra | 2 | Não |

resposta: tesoura <== [tamanho=grande,buracos=2]

← induz(chave)

Exemplos positivos

| tamanho | forma | buracos | colide? |
|---|---|---|---|
| Pequeno | longa | 1 | Não |
| grande | longa | 1 | Não |

| | | | |
|---|---|---|---|
| pequeno | outra | 2 | Não |

resposta:

chave <== [[forma=longa,buracos=1],[forma=outra,buracos=2]]

Procedimento de CLASSIFICAÇÃO:

classifica(Objecto, Classe) ←

Classe <== Descrição,

unifica(Objecto, Classe <== Descrição).

Exercício:

← induz(porca)

resposta: porca <== [tamanho=compacta,buracos=1]

# INDUÇÃO DE ÁRVORES DE DECISÃO

Método para representar conceitos definidos por atributos

Os conceitos são definidos por árvores de decisão em vez de regras se...então como no caso anterior

| | |
|---|---|
| Nodos internos = | atributos |
| Folhas = | classes/objectos (null - nenhum exemplo) |
| Ramos = | valores dos atributos |

Classificação de um objecto = percorrer a árvore

VANTAGENS:

- maior constrangimento (redução da complexidade combinatória da procura)
- maior eficácia na aprendizagem

DESVANTAGENS:

- representação mais complexa e mais extensa
- computacionalmente mais dificeis de implementar

Exercício: induzir a árvore de decisão para o seguinte conjunto de exemplos:

| Objecto | Buracos | Tamanho | Forma |
|---|---|---|---|
| lápis | 0 | grande | longa |
| lápis | 0 | grande | longa |
| parafuso | 0 | pequeno | longa |
| parafuso | 0 | pequeno | compacta |
| chave | 1 | grande | longa |
| porca | 1 | pequeno | compacta |
| chave | 1 | pequeno | longa |
| porca | 1 | pequeno | compacta |
| porca | 1 | pequeno | compacta |
| tesoura | 2 | grande | longa |
| tesoura | 2 | grande | outra |
| chave | 2 | pequeno | outra |

ALGORITMO DE INDUÇÃO

seja T a árvore de decisão a induzir

seja S o conjunto de exemplos para a aprendizagem

se todos os exemplos de S pertendem àmesma classe C

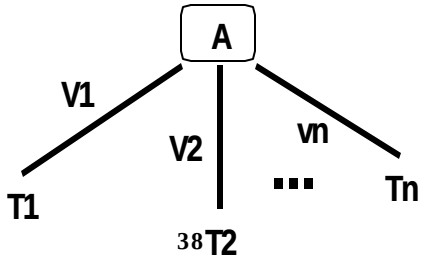
então a árvore tem um só nodo C

senão seleccionar o atributo A mais "informativo" cujos

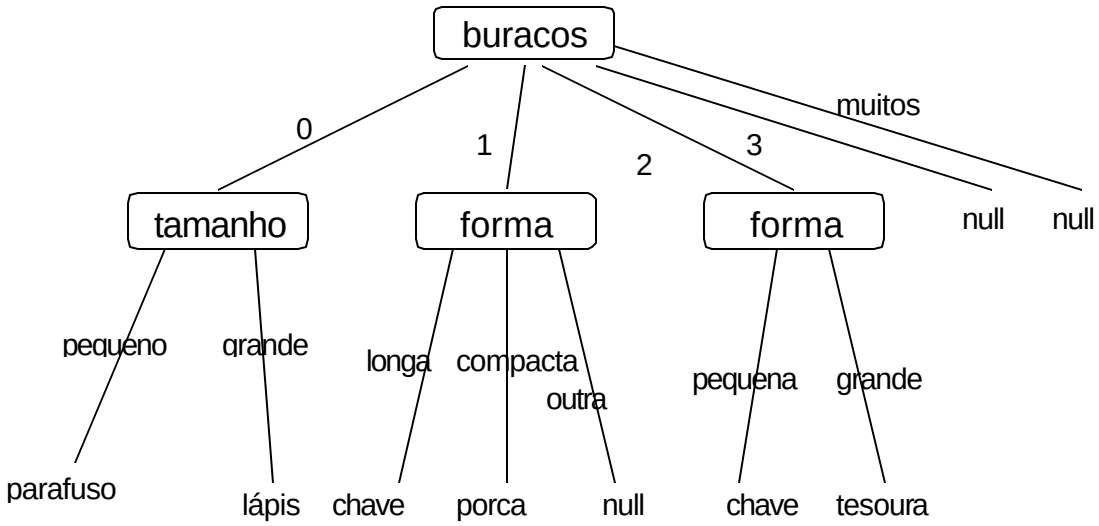
valores são v1, v2, ..., vn

particionar S em n sub-conjuntos S1, ..., Sn um para cada valor vi de A

construir (recursivamente) sub-árvores T1, ..., Tn para cada S1,..., Sn



Árvore de decisão (óptima) para os exemplos dados:



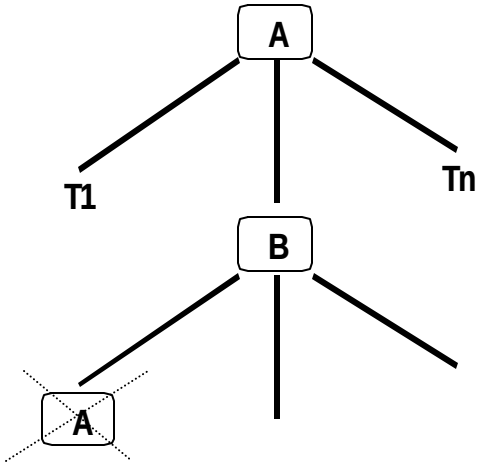

Exemplo:

o objecto [tamanho=pequeno, forma=compacta, buracos=1]

é classificado como uma porca

REFINAMENTOS:

1. Se S é vazio p resultado é uma árvore com um só nodo NULL
2. Cada vez que um novo atributo é seleccionado, são considerados apenas os atributos que não foram usados na parte superior da árvore



3. Se S não está vazio e não acontece que todos os exemplos pertençam à mesma classe C e não há mais atributos para escolher então o resultado é uma árvore com um só nodo. Neste nodo representam-se todas as classes presentes em S associando a cada uma a sua frequência relativa

| exemplo: Classe | Nº exemplos | Freq. relativa |
|---|---|---|
| A | 10 | 0.17 |
| B | 20 | 0.33 |
| C | 30 | 0.5 |

esta árvore é chamada árvore de probabilidades
existe 17% de probabilidade de um objecto pertencer a A
33 % de probabilidade de pertencer a B e
50% de probabilidade de pertencer a C

Conclusão:

- não foram definidos atributos suficientes para distinguir as classes de alguns objectos
- os objectos de diferentes classes possuem os mesmos valores e atributos

4. Como escolher o atributo mais “INFORMATIVO”?

CRITÉRIOS “TIRADOS” DA TEORIA DA INFORMAÇÃO...

1. MEDIDA DA ENTROPIA OU INFORMAÇÃO RESIDUAL

VOLUME DE INFORMAÇÃO:

aquela que é necessária para classificar um objecto

$$I = \sum_c p(c) \log_2 p(c)$$

I - Volume de informação médio para uma classe/objecto c

c - classe/objecto

p( c) - probabilidade de um objecto em S pertencer àclasse c

INFORMAÇÃO RESIDUAL

após termos aprendido um valor de um atributo A é aquela que sobra

corresponde àinformação contida nos sub-conjuntos

$$I_{res}(A) = -\sum_v p(v) \sum_c \frac{p(v,c)}{p(v)} \log_2 \left( \frac{p(v,c)}{p(v)} \right)$$

v - valores de A
p(v) - probabilidade do valor v no conjunto S
p(v, c) - probabilidade do valor v e a classe c ocorrerem no mesmo objecto em S

O ATRIBUTO MAIS INFORMATIVO É AQUELE QUE MINIMIZA A INFORMAÇÃO RESIDUAL

2. MEDIDA DA IMPUREZA (ÍNDICE DE GINI)

$$Gini(A) = \sum_{v} p(v) \sum_{i \neq j} p(i \mid v)\, p(j \mid v)$$

i, j - classes

A - atributo

v - valores do atributo A

p(v) - probabilidade do valor v ocorrer no conjunto S

p(i | v) - probabilidade de na classe i o atributo A Ter o valor v

O ATRIBUTO MAIS INFORMATIVO É AQUELE QUE TEM MENOR ÍNDICE DE IMPUREZA

Exercício:

Para o seguinte conjunto de exemplos, calcular o índice de impureza para os atributos tamanho, forma e buracos e desenhar a árvore de decisão correspondente:

Exemplos:

| Objecto | Buracos | Tamanho | Forma |
|---|---|---|---|
| **lápis** | **0** | **grande** | **longa** |
| **lápis** | **0** | **grande** | **longa** |
| **parafuso** | **0** | **pequeno** | **longa** |
| **parafuso** | **0** | **pequeno** | **compacta** |

APRENDIZAGEM A PARTIR DE DADOS INCORRECTOS

Erros nos valores dos atributos e nas classes (RUÍDO)

Abandona-se a regra de que os as descrições dos conceitos aprendidas unificam todos os exemplos positivos e nenhum dos negativos

Admite-se que possam ser feitas más classificações!

Isto tem duas implicações:

1. a árvore induzida pode classificar mal novos objectos
2. a árvore induzida é mais extensa e mais difícil de entender

Problema:

Se em 100 exemplos 99 pertencerem a uma classe C1 e 1 a uma classe C2 e se soubermos que existe ruído então podemos concluir que o exemplo pertencente a C2 está no conjunto de exemplos provavelmente por erro!

Solução:

Ignorar este exemplo e construir a árvore com um nodo correspondente à classe C1 e cortar a sub-árvore correspondente a C2 (PODA)

PODA DE ÁRVORES DE DECISÃO

Tipos de poda: durante a aprendizagem (forward prunning)
após a aprendizagem (post prunning)

PODA APÓS A APRENDIZAGEM

- Decidir se uma sub-árvore deve ou não ser podada.
- Permite minimizar o erro de classificação esperado na raíz da árvore ou seja o erro esperado de toda a árvore.
- Uma árvore podada está optimizada apenas em relação ao erro de classificação!

Algoritmo post prunning

Seja T uma árvore de decisão completa não podada

Calcular o ERRO ESTÁTICO para cada folha da árvore
Para todo o nodo que não seja uma folha no sentido das folhas para a raíz calcular:

ERRO ESTÁTICO e o
ERRO PROPAGADO
Se o ERRO ESTÁTICO < ERRO PROPAGADO
então podar a árvore passando esse nodo a ser uma folha

ERRO ESTÁTICO DE UM NODO

Erro de classificação de um nodo

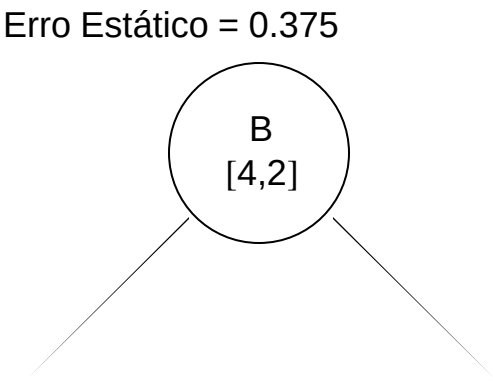
$$E_{Est}(Nodo) = \frac{N - n + k - 1}{N + k}$$

N - Número de exemplos classificados no nodo

n - Número de exemplos pertencentes àclasse maioritária (com mais exemplos)

k - Número de classes total do problema

Exemplo:




[n,m] significa n exemplos da classe C1 e m exemplos da classe C2

ERRO PROPAGADO DE UM NODO

Erro de classificação de um nodo devido à propagação de erros das subárvores

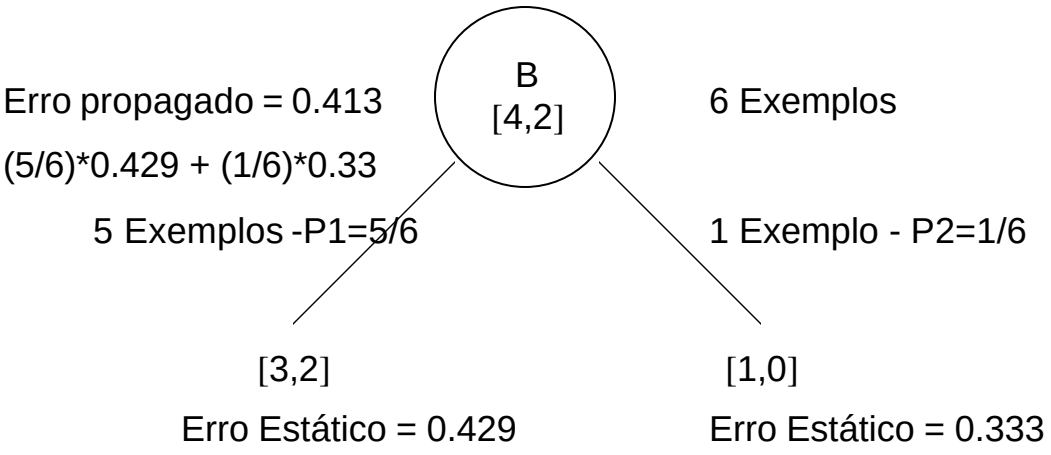
$$E_{Prop}(Nodo) = S_i\ Pi * E_{Est}(Nodo_i)$$

i - ramo do Nodo

$P_i$ - frequência relativa dos exemplos no ramo i

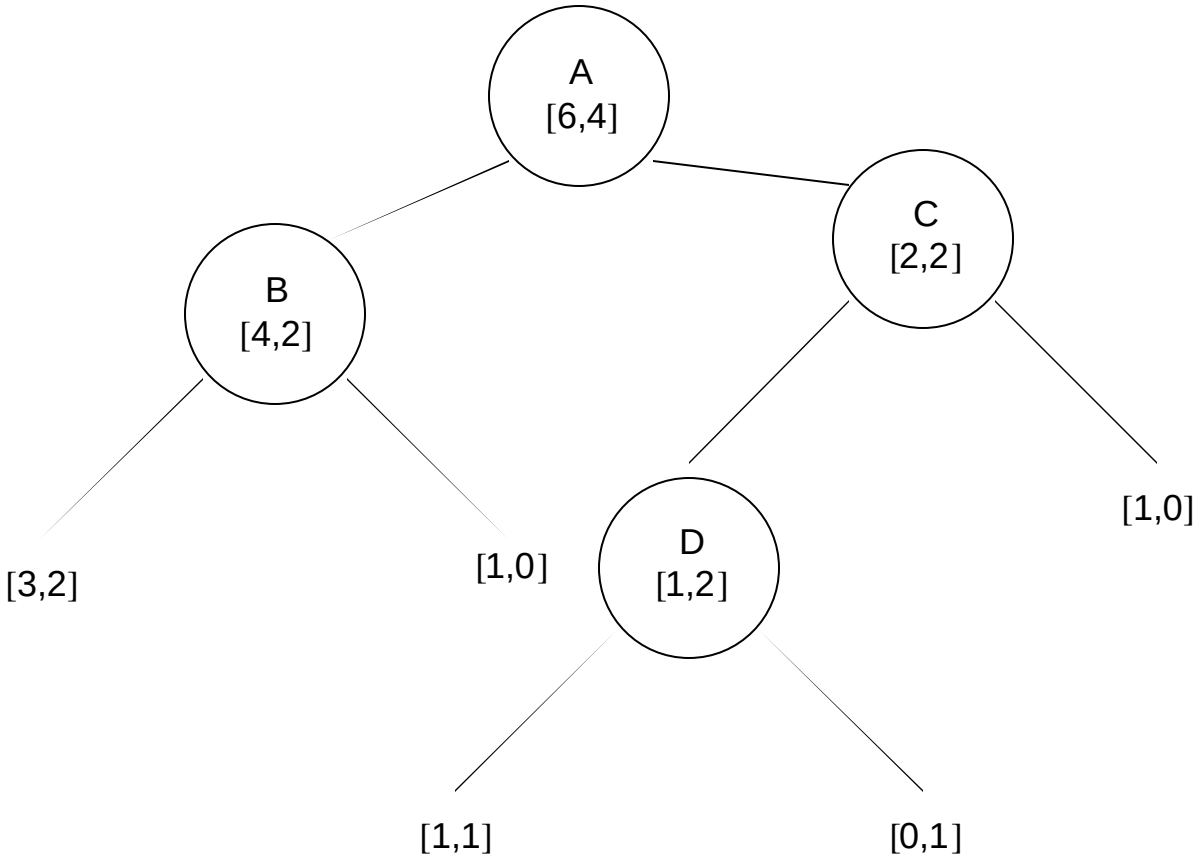
$Nodo_i$ - sub-nodo de Nodo

Exemplo:

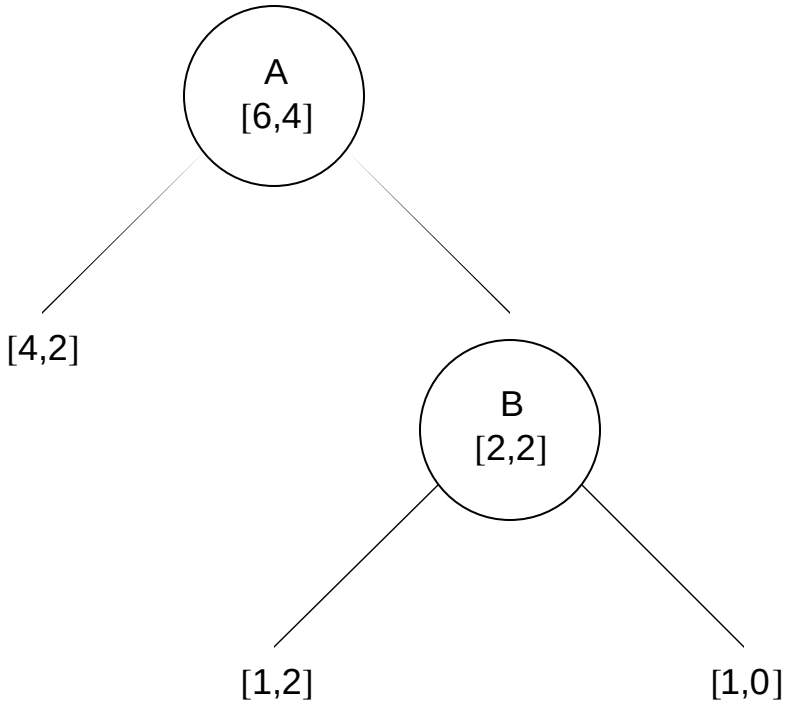



Esta sub-árvore deve ser PODADA!

Exercício: podar a seguinte árvore




Árvore podada:



## CRITÉRIOS DE SUCESSO DA APRENDIZAGEM

Acerto na classificação (% de objectos bem classificados para o conceio F):

classificação durante a aprendizagem

classificação na resolução de problemas

Transparência da descrição do conceito induzido (deve ser facilmente compreendido)

Complexidade computacional

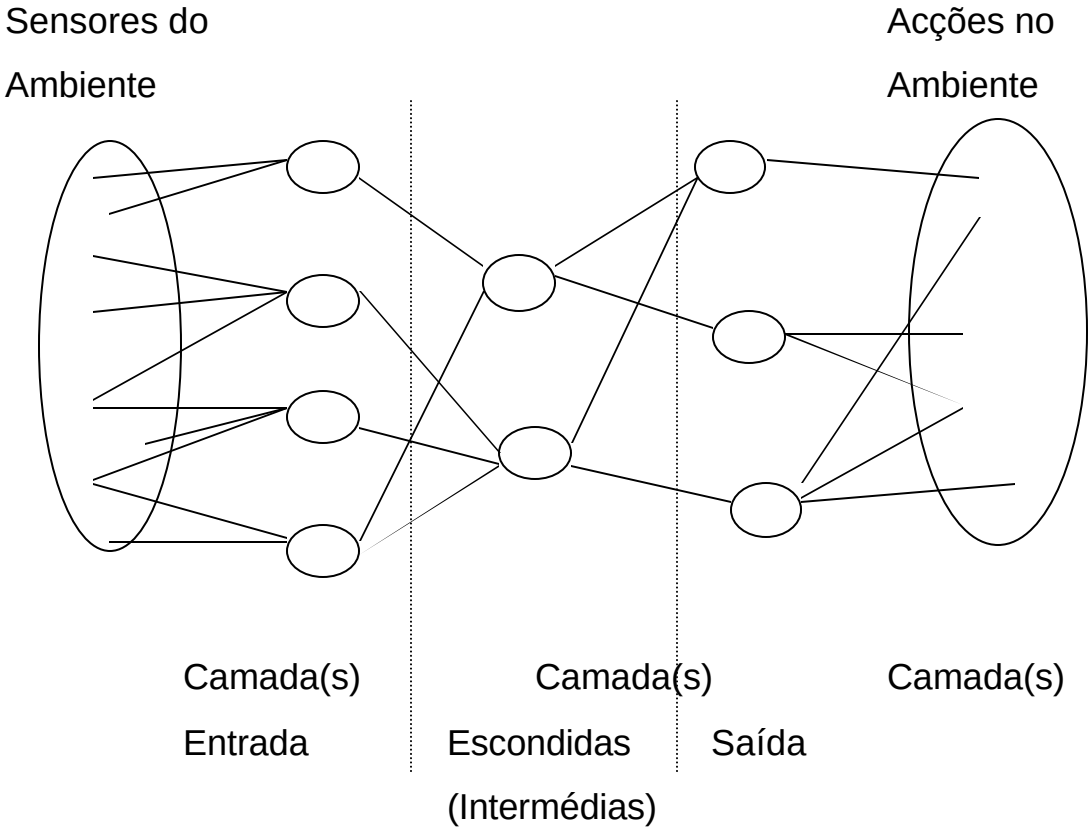
# OBJECTIVO 17: REDES NEURONAIS

Do ponto de vista da computação

Método de representar funções utilizando redes de elementos de computação simples, e métodos para aprender essa representação a partir de exemplos.

Do ponto de vista biológico

Modelo matemático para a operação do cérebro humano (similar às gates dos circuitos electrónicos).

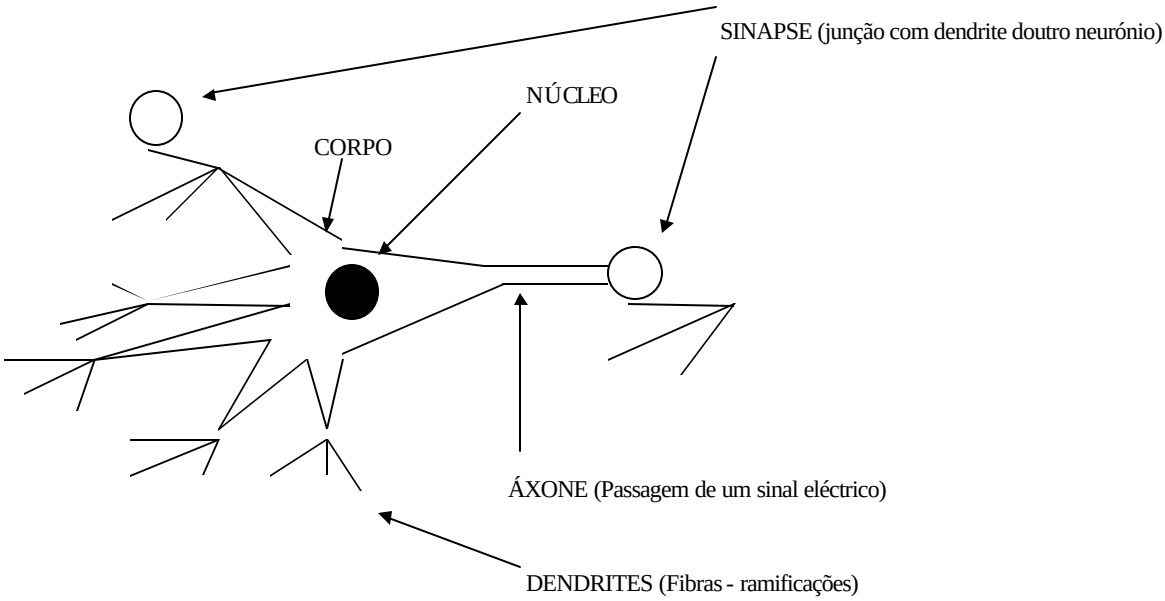



Aplicações das redes neuronais:

- Reconhecimento de voz
- Visão

- Problemas combinatórios (caixeiro viajante...)

Estrutura de um neurónio (Célula nervosa)




Cada neurónio recebe um conjunto de sinais eléctricos dos neurónios que estão ligados às suas dendrites e emite um sinal eléctrico para os outros aos quais este está ligado.

As sinapses são ligações que podem ser:

EXCITATÓRIAS - aumentam o potencial eléctrico
INIBITÓRIAS - diminuem o sinal eléctrico

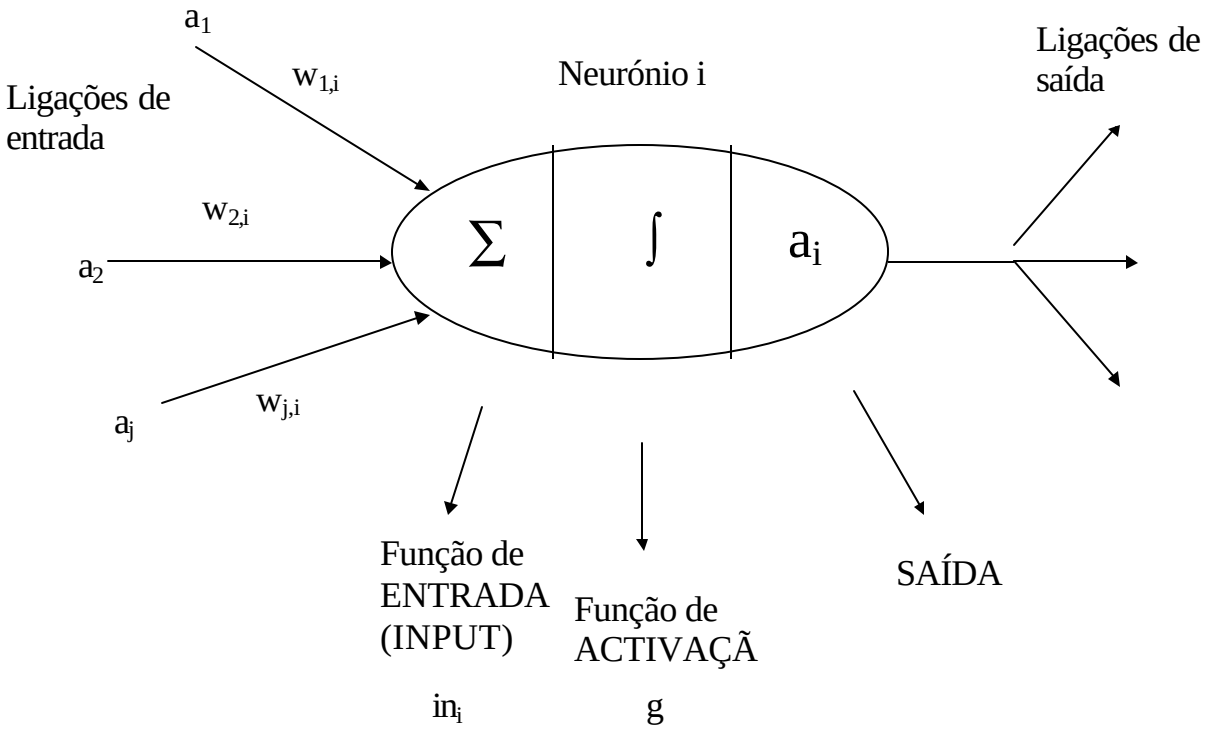
## APRENDIZAGEM NUMA REDE NEURONAL (Cérebro)

- O objectivo é formar a rede através de novas conecções entre os neurónios
- Por vezes grandes quantidades de neurónios migram de um lugar para outro no cérebro

## COMPARAÇÃO COMPUTADOR/CÉREBRO

| | Computador | Cérebro |
|---|---|---|
| Unidades Computacionais | 1 CPU $10^5$ gates | $10^{11}$ neurónios |
| Unidades de memória | $10^9$ bits RAM<br>$10^{10}$ bits DISCO | $10^{11}$ neurónios<br>$10^{14}$ sinapsis |
| Ciclo de Tempo | $10^{-8}$ Seg. | $10^{-3}$ Seg. |
| Largura de banda (comunicação) | $10^9$ bits/seg | $10^{14}$ bits/Seg. |
| Actualização de neurónios por Seg. | $10^5$ | $10^{14}$ |

CONSTITUIÇÃO DE UMA UNIDADE COMPUTACIOAL = NEURÓNIO ARTIFICIAL




NOTAÇÃO

j e i - neurónios

$a_i$ - valor de activação do neurónio i:

$$a_i = g(in_i)$$

g - função de activação (normalmente é igual para todos os neurónios)

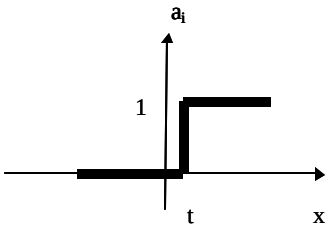
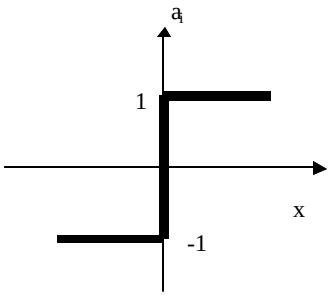
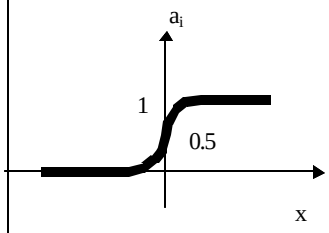
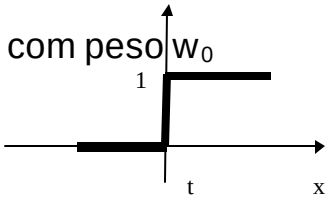
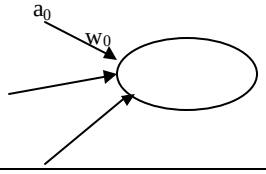
$in_i$ - soma pesada das entradas no neurónio i :

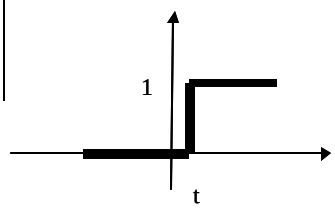
$$in_i = \sum_j a_j.w_{j,i}$$

$w_{j,i}$ - peso da ligação entre o neurónio j e i

$a_j$ - entrada j

FUNÇÕES DE ACTIVAÇÃO ($g(x)\ x = in_i$)

| STEP (Degrau) | SIGN (Sinal) | SIGMÓIDE |
|---|---|---|
|  |  |  |
| t é o ponto de começo (thresold) e é definido para cada neurónio<br><br>$g(x)=\begin{cases} 1 \text{ se } x \geq t \\ 0 \text{ se } x < t \end{cases}$<br><br>Pode eliminar-se t se definirmos uma entrada especial $a_0$ com peso $w_0$<br>  <br>  | $g(x)=\begin{cases} 1 \text{ se } x \geq 0 \\ -1 \text{ se } x < 0 \end{cases}$ | $g(x)=\dfrac{1}{1+e^{-x}}$ |


1
t

Requisitos para construir uma rede neuronal

1. Determinar o número de neurónios
2. Definir o tipo de neurónios (funções de activação)
3. Definir como é que estes devem ser conectados
4. Definir o processo de codificação da entrada na rede e descodificação da saída
5. Inicializar os pesos da rede estes é que codificam o conhecimento da rede
6. Treinar a rede: recalcular os pesos utilizando um algoritmo aplicado a um conjunto de exemplos (codificados) para a tarefa a desempenhar pela rede. Chama-se época ao processamento de todo o conjunto de exemplos.

## ESTRUTURAS DE REDES NEURONAIS

### Feed-Forward

- Ligações unidireccionais
- Não existem ciclos
- Grafos direccionados acíclicos
- Em redes Multi-nível cada unidade é ligada apenas às unidades da próxima camada, não existem ligações entre unidades da mesma camada, não existem ligações para a camada anterior, não se saltam camadas

### Recorrente

- Ligações podem formar qualquer topologia
- Mantém o estado interno por “realimentação”
- Mais próximas do modelo humano

### Uni-camada

### Multi-camada

### Modelo Hopfield

- Rede recorrente
- Ligações bidireccionais com pesos simétricos $w_{ij}=w_{ji}$
- Todas as unidades são de entrada e saída
- Função de activação SIGN com resultado +1 e -1
- Memória associativa
- Armazena 0.138N exemplos de treino, N= nº unidades da rede

Modelo Boltzman

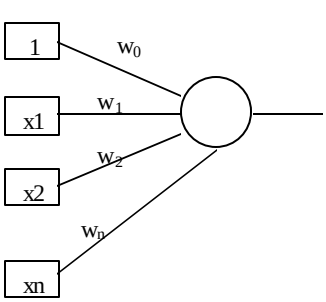
- Pesos simétricos
- Tem unidades que não são nem de entrada nem de saída (intermédias)

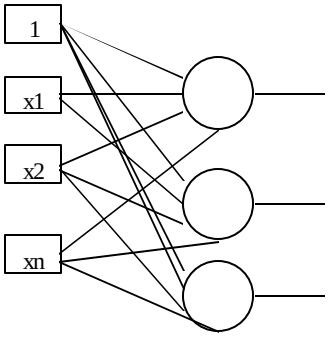
PROCURA DA ESTRUTURA ÓPTIMA

- Algoritmos genéticos
- Procura por “Hill Climbing”
- Aumento/Diminuição incremental da rede
- Algoritmo de Tiling (Diminuição)
- Procura do tamanho certo “Cross Validation”

# PERCEPTRON

- Rede com uma só camada
- Várias entradas e várias saídas

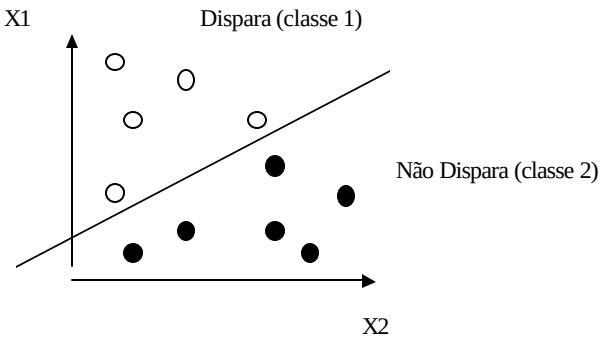



Várias entradas 1 saída  Várias entradas várias saídas

- Função de activação STEP  g(x) = 1 se x > 0 ;  0 se x <0
- A cada unidade é associada uma entrada extra de valor 1 e peso $w_0$
- São indepenedentes uns dos outros podendo por isso ser treinados separadamente
- Só são aplicáveis a problemas linearmente separáveis:

onde se pode desenhar uma linha que separe as classes de saída em função das entradas



Algoritmo de Aprendizagem (Incremento fixo)

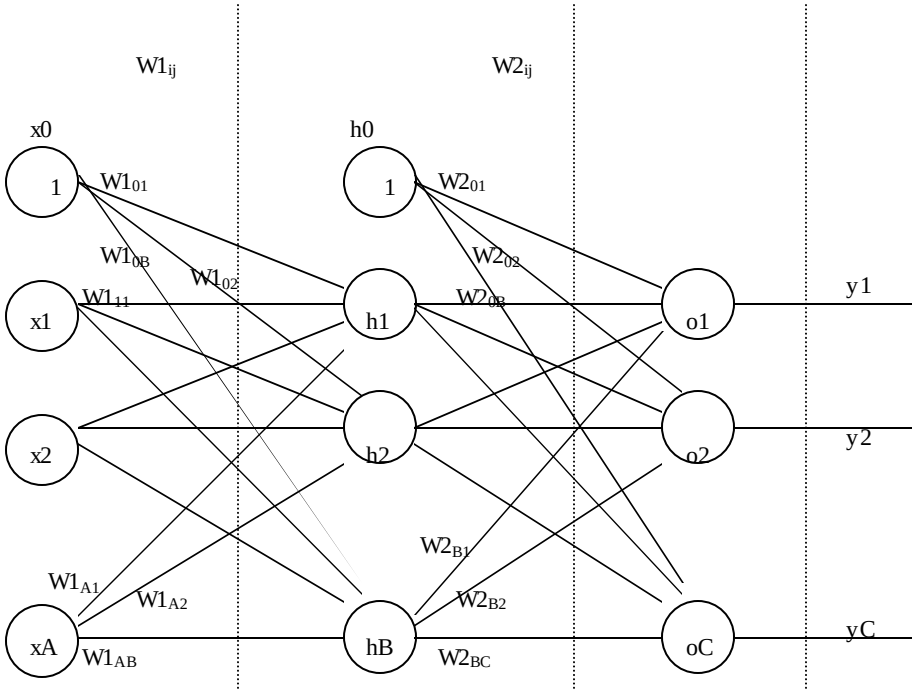
Dado um problema de classificação com n características de entrada (x1, x2, ...., xn) e duas classes de saída

Calcular um conjunto de pesos (W0, W1,..., Wn) que façam com que a rede dispare sempre que a entrada corresponda à primeira classe de saída

1. Criar uma rede com n+1 entradas e n+1 pesos onde a entrada extra x0 é sempre 1
2. Inicializar os pesos (W0, W1,...,Wn) com nºs gerados aleatoriamente $\in$ [-0.5, 0.5]
3. Iterar através do conjunto de treino, isolando todos os exemplos mal classificados pelo actual conjunto de pesos
4. Se todos os exemplos são correctamente classificados, está encontrado o conjunto de pesos, TERMINAR
5. Caso contrário, somar no vector S os vectores de entrada mal classificados (x1,...,xn). Considerar $\vec{-x}$ se o exemplo fez a rede disparar e não devia ter disparado, $\vec{x}$ se a rede não disparou e devia ter disparado. Multiplicar S por $\eta$ (Factor de aprendizagem).
6. Modificar os pesos (W0,...Wn) somando-lhes os elementos do vector S. Saltar para o passo 3

# REDES BACKPROPAGATION

- Rede completamente ligada;
- Multinível (normalmente possui 3 camadas)
- Feed-forward
- Função de activação sigmóide
- Generalização




| A unidades de ENTRADA | B unidades ESCONDIDAS (intermédias) | C unidades de SAÍDA | Saídas do Conjunto de Treino |
|---|---|---|---|

Algoritmo Backpropagation

Dado um conjunto de vectores de pares entrada-saída
Calcular um conjunto de pesos para uma rede de 3 camadas que mapeie as entradas nas saídas correspondentes

1. Seja
   - A o nº de unidades na camada de entrada = tamanho do vector de entrada
   - C o nº de unidades na camada de saída
   - Escolher o número de unidades na camada escondida (intermédia) B

2. Inicializar aleatoriamente os pesos da rede

$$W \in [-0.1, 0.1]$$

3. Inicializar as unidades $x_0$ e $h_0$

$$x_0 = 1.0$$
$$h_0 = 1.0$$

4. Escolher um par de entrada-saída:

$x_i$ - vector de entrada

$y_i$ - vector de saída

5. Propagar as activações desde a camada de entrada até à camada de saída usando a função de activação SIGMOIDE ($x_i \rightarrow h_i \rightarrow o_i \rightarrow y_i$)
6. Calcular os erros das unidades na camada de saída, de seguida para a(s) camada(s) intermédia(s)

$$\sigma 2_j = o_j(1-o_j)(yj-o_j) \qquad j=1,...,C$$
$$\sigma 1_j = h_j(1-h_j)\sum_{i=1..C} \sigma 2_j . W2_{ji} \qquad j=1,...,B$$

7. Ajustar os pesos desde a última camada escondida até à camada de entrada

$$\Delta W2_{ij} = \eta . \sigma 2_j . h_i \qquad i=0,..,B \qquad j=1,...,C$$
$$\Delta W1_{ij} = \eta . \sigma 1_j . x_i \qquad i=0,..,A \qquad j=1,...,B$$

$\eta$ - taxa de aprendizagem (0.35 é um bom valor)

8. Voltar ao passo 4. Quando todos os pares de entrada-saída do conjunto de treino estiverem processados passou-se uma ÉPOCA.
9. Repetir os passos 4..8 durante as épocas desejadas.

APRENDIZAGEM REFORÇADA

- Ao algoritmo de aprendizagem não é fornecida a saída exacta para cada exemplo
- Fornece-se uma recompensa/penalização pelas saídas (um valor real)

APRENDIZAGEM NÃO SUPERVISIONADA

Todos os exemplos anteriores correspondem a aprendizagem supervisionada (presença de um “professor”)

Na aprendizagem não supervisionada:

- Não é dado qualquer feedback para as saídas da rede
- Dado um conjunto de exemplos a rede tenta descobrir REGULARIDADES e RELAÇÕES entre as diferentes partes da entrada
- Descoberta

Exemplo:

Considere-se o conjunto de exemplos:

| | Tem cabelo? | Tem escamas? | Tem penas? | Voa? | Vive na água? | Põe ovos? |
|---|---|---|---|---|---|---|
| Cão | 1 | 0 | 0 | 0 | 0 | 0 |
| Gato | 1 | 0 | 0 | 0 | 0 | 0 |
| Morcego | 1 | 0 | 0 | 1 | 0 | 0 |
| Baleia | 1 | 0 | 0 | 0 | 1 | 0 |
| Canário | 0 | 0 | 1 | 1 | 0 | 1 |
| Robin | 0 | 0 | 1 | 1 | 0 | 1 |
| Avestruz | 0 | 0 | 1 | 1 | 0 | 1 |
| Cobra | 0 | 1 | 0 | 0 | 0 | 1 |
| Lagarto | 0 | 1 | 0 | 0 | 0 | 1 |
| Corcodilo | 0 | 1 | 0 | 0 | 1 | 1 |

ALGORITMO DE APRENDIZAGEM COMPETITIVA

Dada: uma rede com n entradas binárias directamente ligadas a qualquer número de unidades de saída

Produzir: um conjunto de pesos tal que as unidades de saída sejam activadas de acordo com alguma divisão natural nas entradas

1. Apresentar um vector de entrada (x1,..., xn)
2. Calcular a activação inicial para cada unidade de saída (soma pesada das entradas)

3. Permitir que as saídas compitam até que apenas uma esteja activa
4. Ajustar os pesos nas ligações de entrada para a unidade activa
5. Repetir os passos de 1 a 4 para todas as entradas durante muitas épocas

OBJECTIVO 18:

INTELIGÊNCIA ARTIFICIAL PARALELA E DISTRIBUÍDA

As arquitecturas paralelas e distribuídas podem contribuir para:

- implementação de modelos psicológicos (redes neuronais etc.)
- modelos socais (agentes)
- aumento de eficiência
- organização de sistemas de uma forma modular

PARALELISMO

Os programas de IA consomem muito tempo e recursos computacionais

ARQUITECTURAS PARALELAS

Podemos esperar ganhos de velocidade em vários níveis

Exemplo dos sistemas de produção:

Paralelismo ao nível da unificação — vários processadores são utilizados para aumento de velocidade nos ciclos de unificação/resolução

Ao nível das produções — análise simultânea de regras

Ao nível das condições — análise simultânea de condições duma mesma regra

Ao nível das acções

execução simultânea das acções de uma regra

Paralelismo ao nível da tarefa

execução simultânea de vários ciclos

O paralelismo ao nível da tarefa depende da natureza da tarefa

- Diagnóstico médico - cada regra depende da acção da anterior
- Diagnóstico médico de 5 pacientes - pode ser feito em paralelo

O paralelismo ao nível da unificação é mais aplicável

n regras - n processadores = aumento de velocidade n*?

não porque:

- apenas algumas regras são afectadas
- umas regras são mais complicadas que outras ficando estas àespera
- tempo para comunicação entre os processadores

PARALELIZAÇÃO DE LINGUAGENS DE IA

Escrever programas paralelos é uma tarefa difícil ➔ implementação paralela das linguagens

Exemplo do PROLOG (Concurrent PROLOG, PARLOG, Guarded Horn Clauses...)

```
tio(X,Y):- mãe(Z,Y), irmã(Z,X).
tio(X,Y):-pai(Z,Y), irmão(Z,X).

?- tio(António, manuel).
```

Paralelismo OR: estas duas cláusulas podem ser processadas em paralelo. Corresponde a investigar simultaneamente vários ramos da árvore de prova.

```
docente(X):- lecciona(X), licenciado(X).
```

Paralelismo AND: a prova de lecciona(X) e licenciado(X) pode ser feita simultaneamente. No exemplo anterior isto não é possível.

PARALELIZAÇÃO DOS ALGORITMOS DE IA

Alguns problemas são mais facilmente paralelizáveis do que outros!

Nem sempre é possível nem sempre resulta num aumento de velocidade

## HARDWARE PARALELO

O ideal seria desenvolver hardware específico para implementar os algoritmos paralelizados...

## SISTEMAS DISTRIBUÍDOS

Conjunto de módulos separados (também chamados AGENTES desde que actuem como uma entidade para resolução de problemas) e um conjunto de caminhos de COMUNICAÇÃO entre estes.

Granularidade: grau de distribuição - nível de raciocínio em cada módulo.
Granularidade fina = sistemas conectivistas em que cada nodo não desenvolve qualquer tipo de raciocínio

Vantagens sobre os modelos monolíticos:

- Modularidade - mais fácil de construir e manter um conjunto de módulos quase independentes
- Eficiência - Nem todo o conhecimento é necessário para todas as tarefas.

- Arquitecturas computacionais mais rápidas - como as paralelas são melhor aproveitadas pelos sistemas distribuídos
- Raciocínio heterogéneo - em cada módulo é aplicada a técnica mais conveniente
- Perspectivas múltiplas - o conhecimento pode não estar numa só pessoa mas sim em várias e com estruturas incoerentes
- Problemas distribuídos - alguns problemas são inerentemente distribuídos (dados disponíveis em diferentes localizações)
- Segurança - se um problema está a ser resolvido por vários agentes e um deles falha os outros continuam

Características uma ARQUITECTURA DISTRIBUÍDA

1. Um mecanismo de coordenação entre os vários agentes
2. Uma estrutura de comunicação entre os agentes
3. Versões distribuídas das técnicas necessárias

COORDENAÇÃO E COOPERAÇÃO

Um agente está em carga. O agente master divide o problema e entrega sub-problemas aos escravos que retornarão os resultados. Estes podem ainda recorrer a outros agentes.

Um agente está em carga e divide o problema mas negoceia com os outros agentes quem vai ficar com os sub-problemas.

Nenhum agente em carga, existe um objectivo partilhado por todos os agentes. Devem cooperar na planificação da resolução e na resolução propriamente dita.

Nenhum agente em carga, nem um objectivo partilhado. Os agentes competem uns com os outros.

PLANEAMENTO MULTI-AGENTE

Um agente:

1. Decompõe o objecto em sub-objectivos
2. Atribui os sub-objectivos aos outros agentes

Implicações:

- O agente master deve conhecer bem as capacidades dos vários escravos
- O agente master deve balancear as atribuições por forma a conseguir resolver o problema o mais rapidamente possível
- Após a distribuição das tarefas deve promover a sincronização entre os escravos a menos que as tarefas sejam completamente indepenedentes (modelo baseado em estados - os agentes partilham um modelo com o estado do sistema)

PLANEAMENTO E NEGOCIAÇÃO: REDES DE CONTRATAÇÃO

Um agente decompõe o problema mas depois negoceia com os outros para determinar quem vai ficar com as sub-tarefas.

Existem dois tipos de agentes:

- GESTOR, que decompõe o problema, negoceia, atribui e monitoriza os resultados
- Os CONTRATADOS, que executam as sub-tarefas - podem ser por sua vez gestores...

Processo de negociação (licitação)

1. Um gestor anuncia uma tarefa
2. Os contratados avaliam a tarefa e mediante as suas capacidades e recursos disponíveis
3. Os contratados fazem um oferta
4. O gestor escolhe um só contratado e espera pelo resultado

CONTROLO E COMUNICAÇÃO DISTRIBUÍDOS

O controlo pode ser:

- Centralizado num agente
- Descentralizado e distribuído pelos agentes

Os agentes (racionais - reagem de uma forma quase óptima na resolução do problema) podem:

- Comunicar entre si

- Não comunicar entre si, neste caso pode-se utilizar uma matriz de recompensas

COMUNICAÇÃO

As arquitecturas de comunicação entre agentes podem ser:

SISTEMAS BASEADOS EM QUADROS-NEGROS

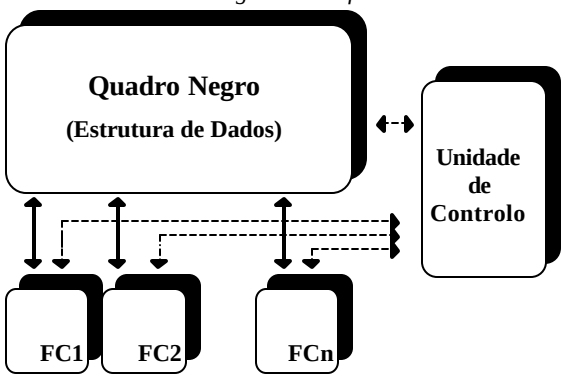
A comunicação é feita através de uma estrutura de conhecimento partilhada (pode ser distribuída) chamada quadro-negro. Os agentes colocam mensagens no quadro-negro e podem ler e actuar sobre mensagens que outros lá colocam.

SISTEMAS DE PASSAGEM DE MENSAGENS

Cada agente envia e recebe mensagens para/de outros agentes que este conhece bem. Um agente pode difundir (broadcast) uma mensagens para todos os outros, ou enviar uma mensagem especificamente para um outro agentes.

SISTEMAS BASEADOS EM QUADROS-NEGROS

1 - uma estrutura de dados, o quadro negro;
2 - as fontes de conhecimento; e
3 - a unidade de controlo.




CICLO DE ACTIVIDADE

1 - uma fonte de conhecimento faz modificações nos objectos contidos no quadro negro;
2 - cada fonte de conhecimento, de acordo com as suas capacidades, apresenta a sua possível contribuição que poderá originar um novo estado da solução;
3 - com a informação que tenha sido manipulada nos passos 1 e 2, a unidade de controlo define qual o seu ponto de interesse; e
4 - dependendo da informação que o ponto de interesse seleccionado envolve, a unidade de controlo prepara essa informação para execução.

OBJECTIVO 19: **SISTEMAS PERICIAIS**

Sistema computacional dotado de conhecimento numa área relativamente restrita, e que, com base num determinado problema, é capaz de:

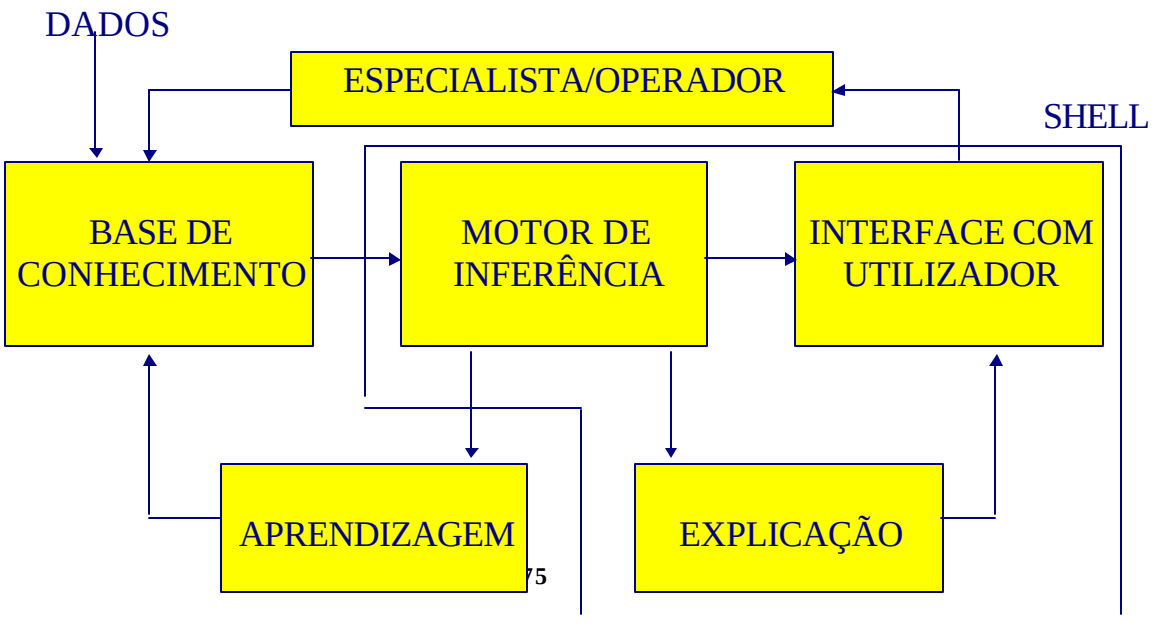
- produzir uma resposta e de
- explicar essa resposta

## CARACTERÍSTICAS PRINCIPAIS

Capacidade de processamento simbólico

- Capacidade de dedução
- Capacidade de explicação do raciocínio
- Tratamento do raciocínio incerto (incerteza) e conhecimento incompleto
- Capacidade de introspecção
- Capacidade de aquisição de conhecimento, quer retirando-o de um perito quer adquirindo-o através de casos conhecidos
- Capacidade de interacção com o utilizador

## ESTRUTURA:



BASE DE CONHECIMENTO (DEPENDE DO DOMÍNIO)

- Factos
- Regras (relações)
- Métodos (para resolver problemas no domínio)

SHELL (INDEPENDENTE DO DOMÍNIO - UNIVERSAL)

DOMÍNIOS DE APLICAÇÃO:

- Diagnóstico - causa de mau funcionamento (médico, industrial...)
- Localização de falhas
- Interpretação de dados de medida
- Escalonamento de pessoal
- Decisões de crédito

...

EXEMPLO (DESENVOLVIMENTO DE UMA SHELL)

1. Seleccionar e definir os detalhes do formalismo para representação do conhecimento
2. Desenho dos detalhes do mecanismo de inferência para o formalismo escolhido
3. Adicionar facilidades de interacção com o utilizador
4. Adicionar facilidades para manuseamento de incerteza

Problema: identificação de animais

## 1. REPRESENTAÇÃO DO CONHECIMENTO

### Regras se...então

**regra1:** **se**

**Animal tem cabelo**

**ou**

**Animal dá leite**

**então**

**Animal éum mamífero.**

**regra2:** **se**

**Animal tem penas**

**ou**

**Animal voa**

**ou**

**Animal põe ovos**

**então**

**Animal éum pássaro.**

**regra3:** **se**

**Animal éum mamífero e**

**(Animal come carne**

**ou**

**Animal tem 'dentes caninos' e**

**Animal tem garras e**

**Animal tem os 'olhos apontados para a frente')**

**então**

**Animal éum carnívero.**

**regra4:** **se**

**Animal éum carnívero e**

**Animal tem 'cor de laranja' e**

**Animal tem 'manchas negras'**

**então**

**Animal éum xita.**

**regra5: se**

**Animal éum carnívero e**

**Animal tem 'cor de laranja' e**

**Animal tem 'listas negras'**

**então**

**Animal éum tigre.**

**regra6: se**

**Animal éum pássaro e**

**Animal não voa e**

**Animal nada**

**então**

**Animal éum pinguim.**

**regra7: se**

**Animal éum pássaro**

**Animal éum 'bom voador'**

**então**

**Animal éum albatroz.**

**facto: X éum animal:- membro(X, [xita, tigre, pinguim, albatroz ] ).**

**questionavel(_ dá _, 'Animal' dá 'O quê').**

**questionavel(_ voa _, 'Animal ' voa).**

**questionavel(_ põe ovos, 'Animal' põe ovos).**

**questionavel(_ come _, 'Animal' come 'O quê').**

**questionavel(_ tem _, 'Animal' tem 'Alguma coisa').**

**questionavel(_ não _, 'Animal' não 'Faz alguma coisa').**

**questionavel(_ nada, 'Animal' nada).**

**questionavel(_ éum ´bom voador', 'Animal' éum 'bom voador').**

## 2. MOTOR DE INFERÊNCIA

| Sistema Pericial | Utilizador |
|---|---|
| **Questão por favor:** | manuel éum tigre. |
| **É verdade que: manuel tem cabelo?** | Sim. |
| **É verdade que: manuel come carne?** | Não. |
| **É verdade que: manuel tem dentes caninos?** | Sim. |
| **É verdade que: manuel tem garras?** | Porquê? |
| **Para investigar, pela regra3, manuel é carnívero** | Sim. |
| **Para investigar, pela regra5, manuel éum tigre** | |
| **É verdade que: manuel tem garras?** | |
| **É verdade que: manuel tem olhos apontados para a frente?** | Sim. |
| **É verdade que: manuel tem cor de laranja** | Sim. |
| **É verdade que: manuel tem faixas negras** | Sim. |
| **(manuel éum tigre) é verdade.** | Sim. |

**Deseja ver como?**

**manuel éum tigre**

**derivado da regra5 de**

**manuel é carnívero**

**derivado da regra3 de**

**manuel é mamífero**

**derivado de regra1 de**

**manuel tem cabelo**

**foi dito**

**e**

**manuel tem dentes caninos**

**foi dito**

**e**

**manuel tem garras**

**foi dito**

**e**

**manuel tem olhos**

**apontados para a frente**

**foi dito**

**e**

**manuel tem cor de laranja**

**foi dito**

**e**

**manuel tem listas negras**

**foi dito**

## PROCESSO DE RACIOCÍNIO

Para procurar uma resposta Resp para uma questão Q

1. Se Q é um facto na BC então Resp='Q é verdade'

2. Se existe uma regra na BC na forma

se Condição então Q

então explorar a Condição e usar o resultado para construir Resp

3. Se Q é uma questão questionável então perguntar acerca de Q

4. Se Q é da forma Q1 e Q2 então explorar Q1 e depois:

se Q1 é falso então Resp='Q é falso', senão explorar Q2 e combinar as respostas de Q1 e Q2 em Resp

5. Se Q é da forma Q1 ou Q2, explorar Q1 e:

se Q1 é verdadeira então Resp=´Q é verdade´, ou alternativamente explorar 2 e combinar as respostas de Q1 e Q2 em Resp