Inteligência Artificial

Aula 28- Aprendizagem de Máquina: Classificação por Redes Neurais [1]

Sílvia M.W. Moraes

Faculdade de Informática - PUCRS

June 13, 2017

[1] Este material não pode ser reproduzido ou utilizado de forma parcial sem a permissão dos autores.

Sinopse

- Nesta aula, continuamos a falar em **aprendizagem de máquina.**
- Este material foi construído com base nos capítulos:
  - 03 e 04 - Redes Neurais : Principios e Prática: Simon Haykin.
  - 07 - Inteligência Artificial: Uma abordagem de Aprendizagem de Máquina: Facelli e outros.
  - 11 do livro Inteligência Artificial: Luger
  - 19 do livro Artificial Intelligence – a Modern Approach: Russel & Norvig

Sumário

Aulas anteriores

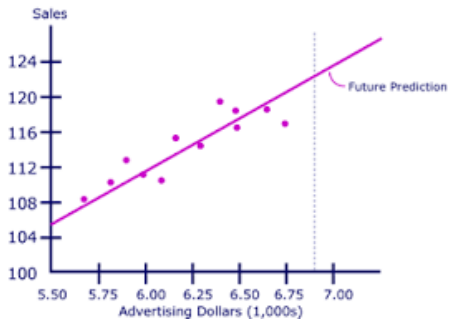
- Agente Reativos e Cognitivos
- Solução de Problemas: Algoritmos de busca
- Planejamento Clássico
- Introdução à Raciocínio Probabilistico
- Introdução à Aprendizagem de Máquina
  - Pré-processamento
  - Agrupamento: K-means
  - Classificação: k-NN

Paradigmas e Tarefas de Aprendizagem

- **Paradigma de aprendizagem** é definido pela natureza do problema. Tipo de realimentação usada pelo algoritmo para aprender.
  - Podem ser:
    - **Supervisionado**: aprendizagem de uma função h a partir de exemplos (amostras rotuladas), de entradas (x) e saídas correspondentes ($f(x)$). Com crítica referente ao erro da saída.
    - **Não-supervisionado**: aprendizagem a partir de as amostras não são rotuladas. Essa abordagem não usa os atributos de saída. Sem critica, usa regularidades e propriedades estatísticas dos dados.
    - **Por reforço**: processo de aprendizagem baseado em punição e recompensa. Reforça uma ação positiva e penaliza, uma negativa. Critica apenas de desempenho.

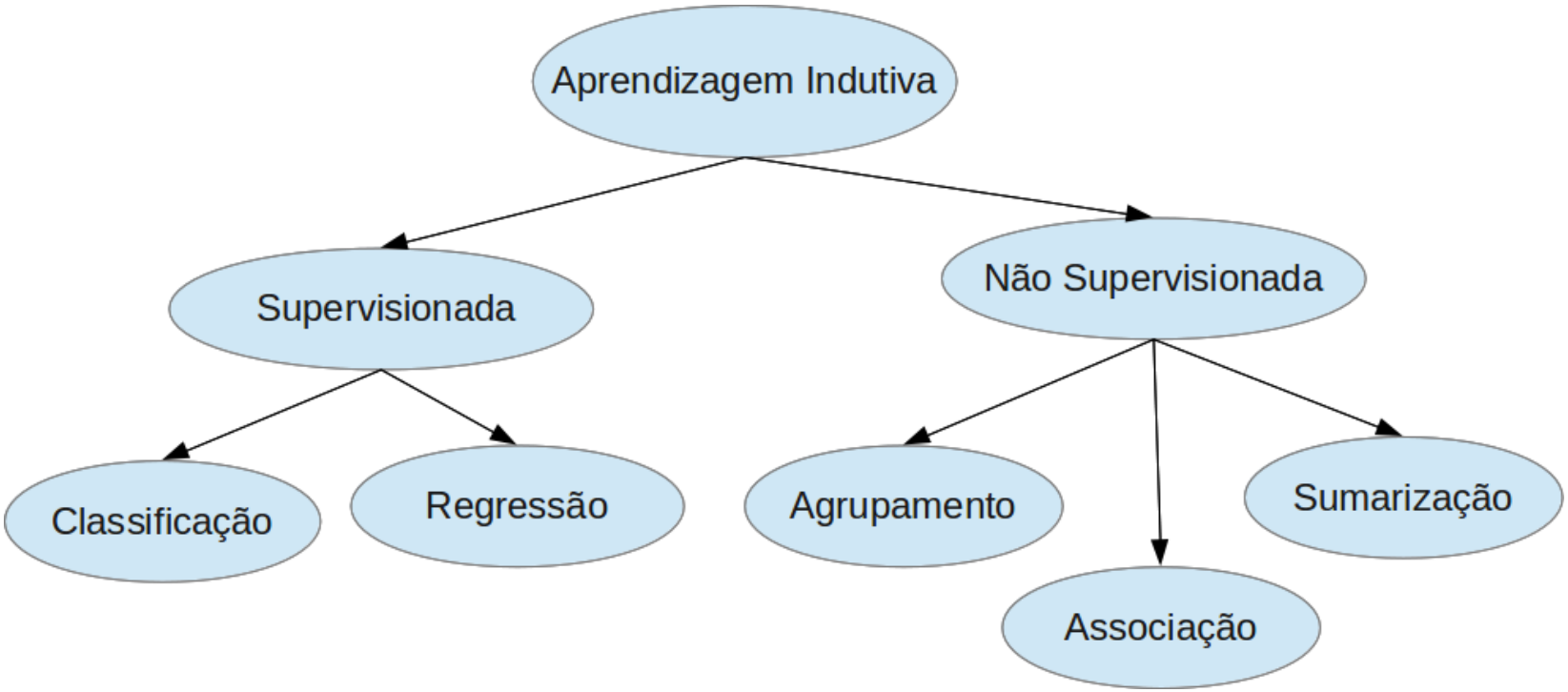
Paradigmas e Tarefas de Aprendizagem

- As **tarefas de aprendizagem** podem ser: **preditivas** ou **descritivas**
  - **preditivas:** tarefa supervisionada, sua meta é encontrar uma função (modelo ou hipótese) a partir dos dados de treino que possa ser usada para prever um rótulo (classe) ou valor de um novo exemplo.
    - Ex: **classificação** (rótulos discretos), **regressão** (rótulos contínuos)



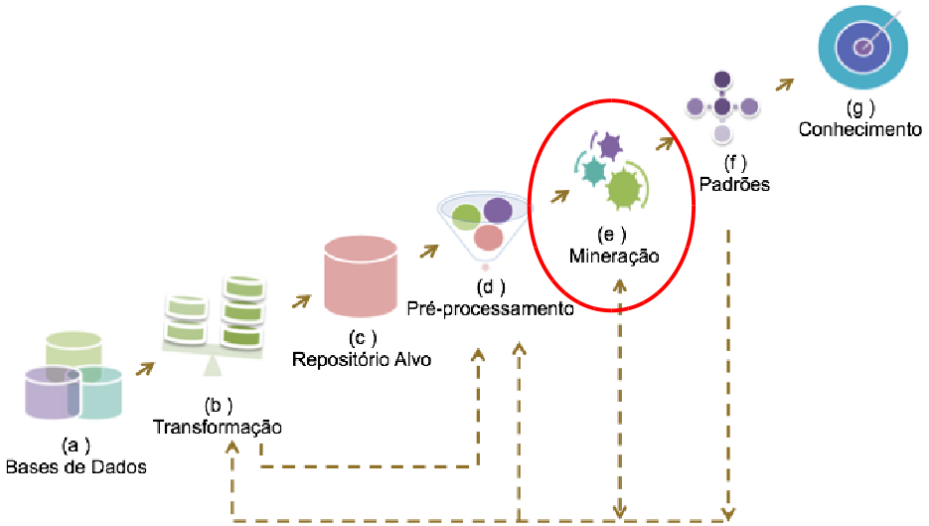
Paradigmas e Tarefas de Aprendizagem

- Resumo:



Processo de Descoberta de Conhecimento

- **Knowledge Discovery in Databases (KDD)**: consiste em uma série de passos bem definida cujo meta é transformar dados em conhecimento.
  (e) Mineração :
  - Usa Algoritmos de aprendizado de máquina
  - Análise de uma séries de dados para compreensão do domínio
  - Resultados compreensíveis e especialmente úteis



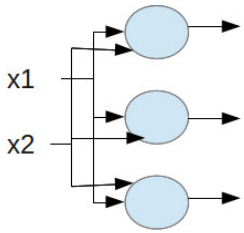
Classificação: Conceito

- **Objetivo: classificação de dados é o processo de automaticamente atribuir um (single label) ou mais rótulos (multi-label), ditos classes, aos dados.**

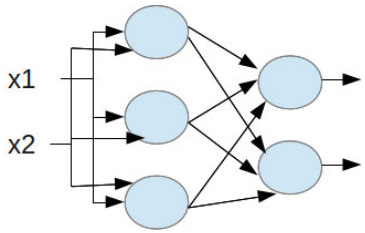
  - É uma **tarefa preditiva**, supervisionada que exige que os **dados usados para definir o modelo estejam rotulados**.
  - Os rótulos (classes) são pré-definidos.

Redes Neurais: Definição

- "Uma **rede neural** é um processador paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para uso." (Haykin,2001)

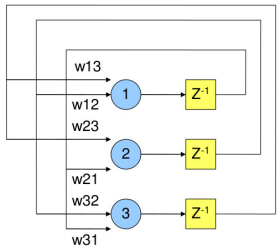


Perceptron



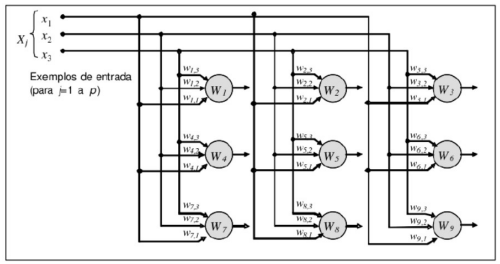
MultiLayer Perceptron

Classificação



HopField

Memorização



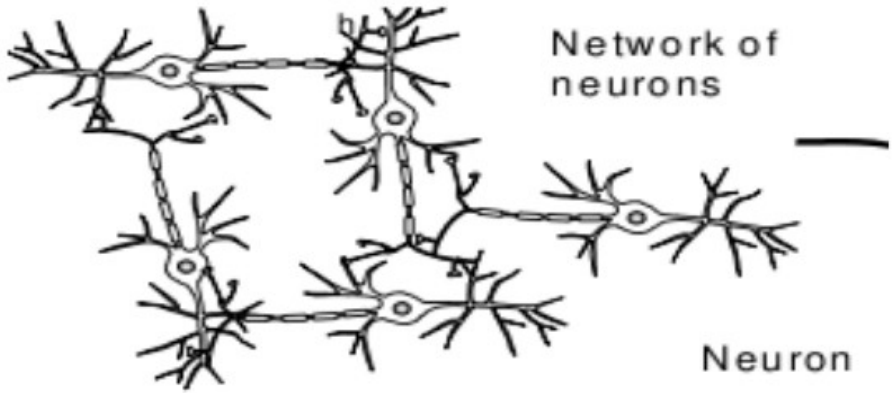
Self-Organizing Map (SOM)

Agrupamento

Redes Neurais: Aplicações

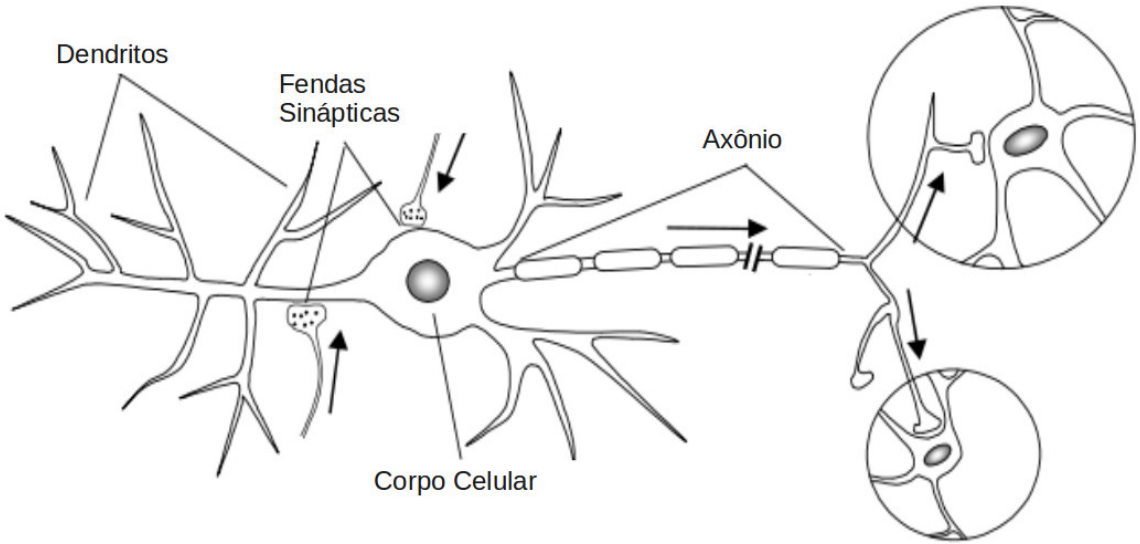
- Reconhecimento de Padrões (visão, voz, imagens, texto, ...)
- Classificação
- Clusterização (agrupamento)
- Memorização ...

Redes Neurais: Cérebro Humano



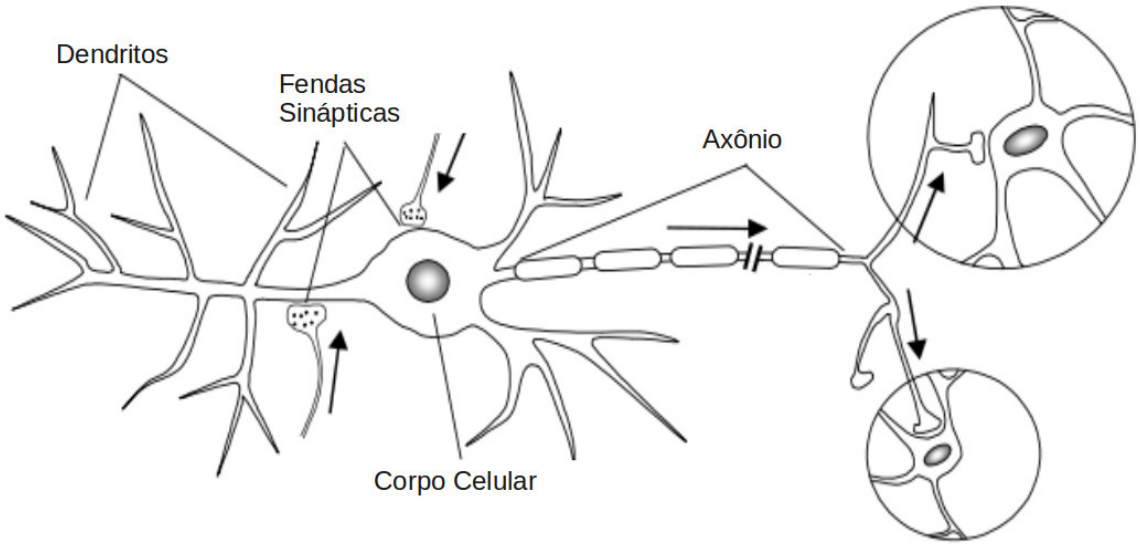
- Redes Neurais Artificiais (RNA) são modelos matemáticos inspirados no cérebro humano.
- O cérebro humano é um "processador" com bilhões de neurônios.
- Os neurônios estão conectados uns aos outros através de sinapses, formando uma grande rede NEURAL.

Redes Neurais: Neurônio Biológico



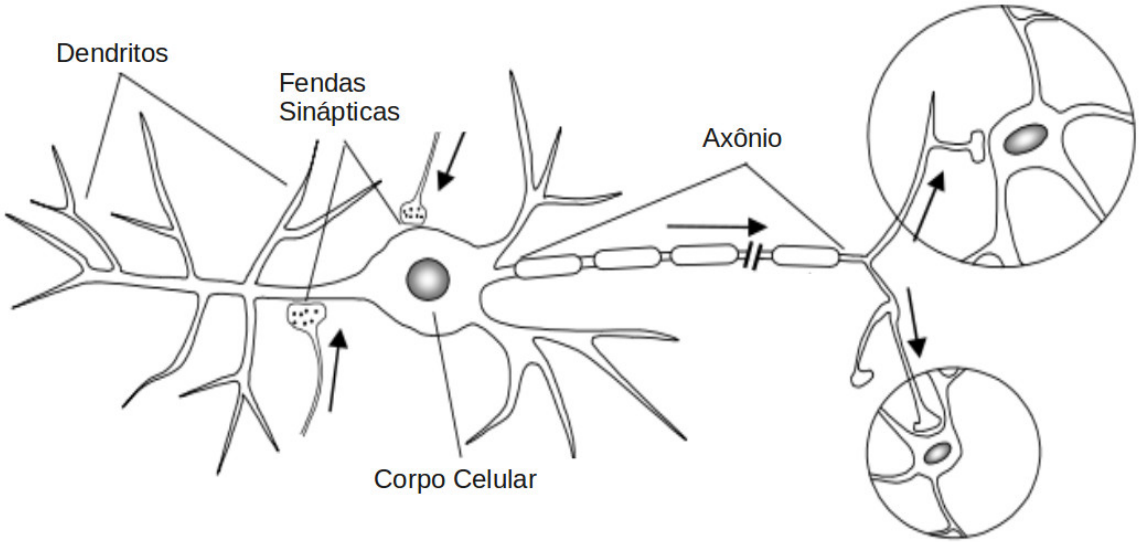
- Os principais componentes:
  - **Dentritos**: recebem estímulos de outros neurônios;
  - **Corpo celular** : coleta e combina informações vindas de outros neurônios;
  - ...

Redes Neurais: Neurônio Biológico



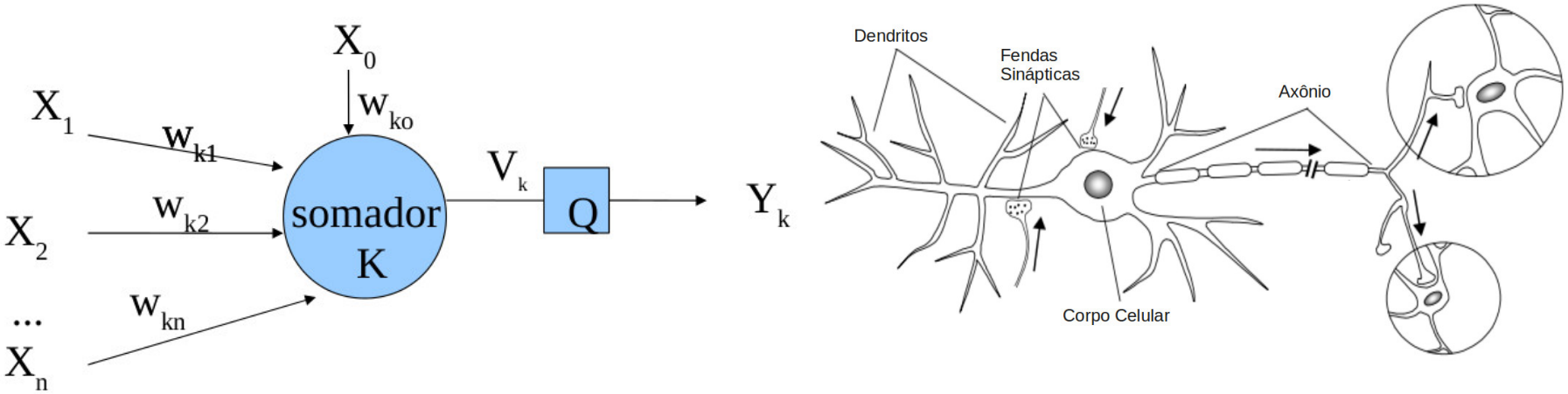
- Os principais componentes:
  - ...
  - **Axônio**: fibra tubular (pode alcançar até alguns metros) responsável por transmitir os estímulos para outras células.
  - **Sinapses**: pontos onde as extremidades de neurónios vizinhos se encontram.
    - transmitem estímulos através de diferentes concentrações de Na+ (Sódio) e K+ (Potássio).

Redes Neurais: Neurônio Biológico



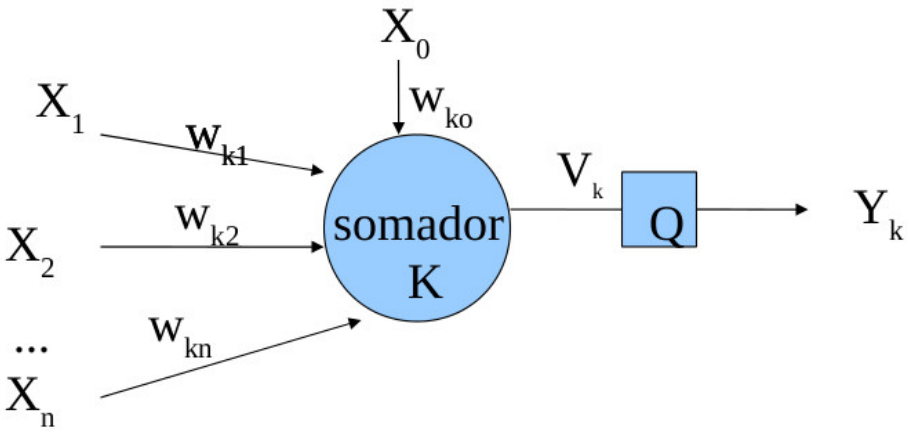
- Os neurônios se comunicam através de impulsos. O neurônio recebe e processa o impulso, disparando outro impulso, produzindo uma substância neurotransmissora que flui do corpo celular para o axônio.
- O neurônio que transmite o pulso pode controlar a freqüência de pulsos aumentando ou diminuindo a polaridade na membrana pós sináptica.

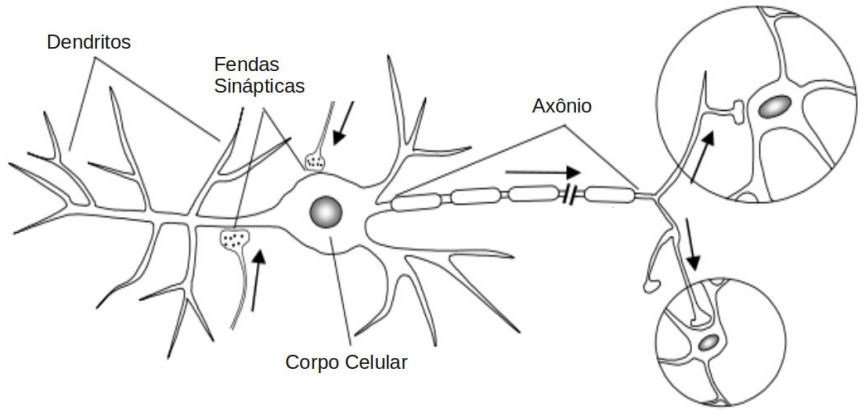
Redes Neurais: Neurônio Artificial



- Modelo desenvolvido por McCulloch & Pitts na década de 40.
- Elementos:
  - Entradas:
    - Simulam os dendritos.
    - x_1, .., x_n: valores do domínio.
    - x_0 : entrada extra, é sempre 1 (bias).

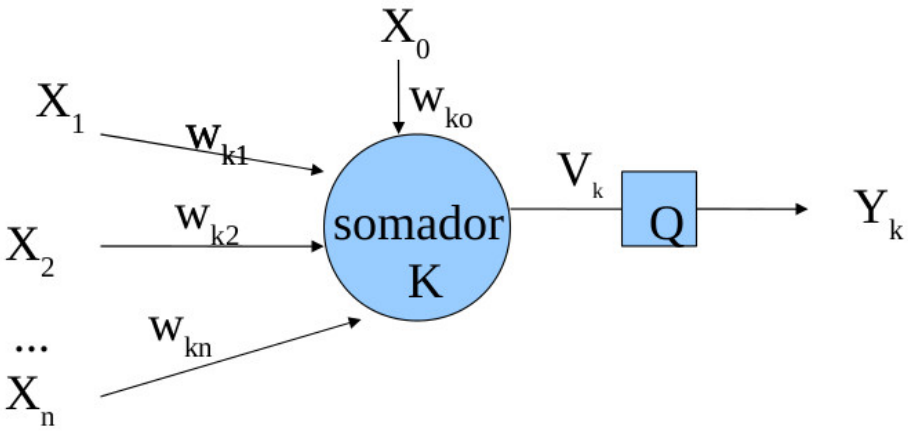
Redes Neurais: Neurônio Artificial

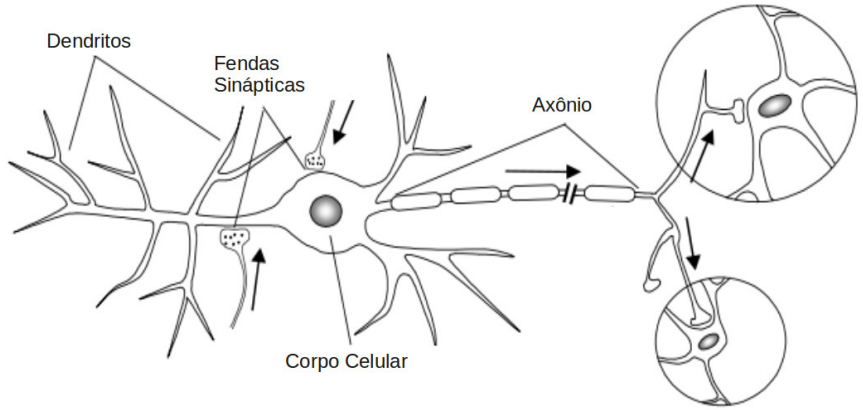




- Elementos: (continuação)
  - Pesos sinápticos:
    - simulam as sinapses (os pesos correspondem aos estímulos, que podem ser variados em intensidade).
    - armazenam o conhecimento adquirido pela rede.
    - w_{kn}, onde k é o neurônio e o n, a entrada.
    - w_{k0}: peso do bias

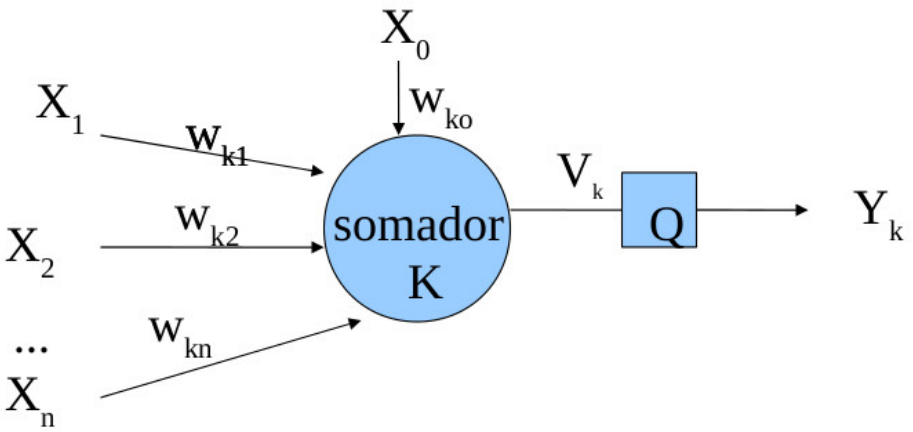
Redes Neurais: Neurônio Artificial

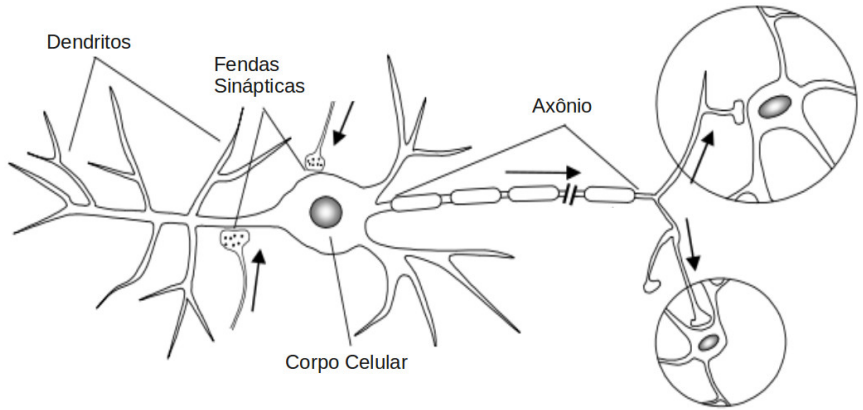




- Elementos (continuação):
  - v_k: campo local induzido
    - simula o corpo celular
    - corresponde a uma soma ponderada (pelos pesos) das entradas do neurônio

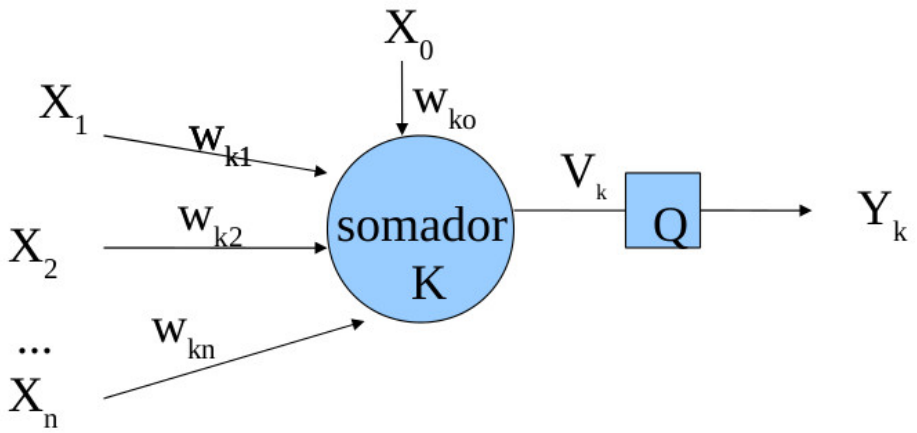
Redes Neurais: Neurônio Artificial





- Elementos (continuação):
  - Q: função de transferência ou ativação
    - simula a polaridade da membrana pós sináptica
    - é uma função matemática que em domínios discretos gera os valores 0 (inibição) e 1 (ativação)
  - y_k: saída do neurônio k
    - simula o axônio

Redes Neurais: Neurônio Artificial



- $v_k = \sum_{i=0}^{n} w_{ki} \times x_i$
- $y_k = Q(v_k)$, onde a função de transferência (ou de ativação) pode ser:
  - limiar: $Q(v_k) = \begin{matrix} 1 & se\ v_k \geq 0 \\ 0 & caso\ contrário \end{matrix}$

Redes Neurais x Cérebro Humano

- Uma **rede neural se assemelha ao cérebro** em dois aspectos (Haykin,2001):
  - o conhecimento é adquirido pela rede a partir do ambiente através de um processo chamado de aprendizagem.
  - forças de conexão entre os neurônios, conhecidas como pesos sinápticos, são usados para armazenar o conhecimento adquirido.

Redes Neurais: Etapas de Construção

- **Fase de Treinamento**
  - Pré-processamento dos dados de Treino
  - Definição da Topologia
  - Treinamento da rede (uso de um algoritmo)
- **Fase de Generalização**
  - Pré-processamento dos dados Teste (dados de entrada)
  - Generalização dos dados de Teste
  - Pós-processamento (saída da rede)
  - Análise dos resultados

Redes Neurais: Fase de Treinamento

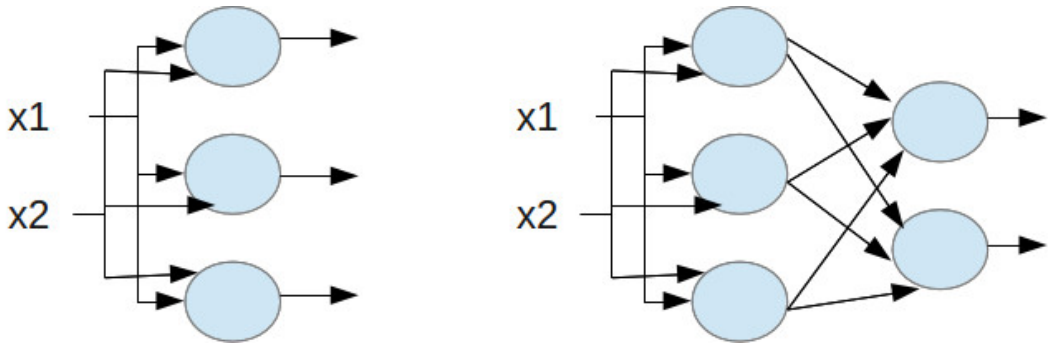
- **Conjunto de Dados (históricos):**
  - Nessa tarefa, **os dados são divididos inicialmente em 2 subconjuntos** disjuntos:
    - **Conjunto de treinamento:** é usado para treinar o algoritmo durante a etapa de fase aprendizagem.
      Este conjunto é subdividido em 2 novos conjuntos disjuntos:
      - Subconjunto **de estimação**: usado para selecionar o modelo;
      - Subconjunto **de validação**: usado para testar e validar o modelo.
    - **Conjunto de teste:** é usado para validar o modelo na fase de generalização (teste final).
  - No mínimo, 2 conjuntos: treinamento (~80% das amostras) e teste (~20% das amostras) .

Redes Neurais: Fase de Treinamento

- Pré-processamento
  - A rede **trabalha apenas com valores de entrada numéricos**, os quais devem pertencer ao intervalo [0;1].
  - O valor médio de cada valor de entrada deve ser pequeno se comparado ao desvio padrão.
  - Necessário usar técnicas vistas em aulas passadas:
    - Transformação de dados: Simbólico-Numérico
    - Normalização: Reescala ou padronização
      Ex: Reescala $c_i'' = \frac{c_i - c_{min}}{c_{max} - c_{min}}$, onde:
    - Em domínios com muitos atributos, usar técnicas de redução de dimensionabilidade.

Redes Neurais: Topologia - redes do tipo Perceptron

- Adequadas para classificação
  - **Perceptron:**
    - a primeira a surgir (1958)
    - Limitada: só classifica problemas linearmente separáveis
  - **MultiLayer Perceptron:**
    - extensão da anterior (~1987)



Topologias de uma rede Perceptron e de uma MultiLayer Perceptron (MLP)

Redes Neurais: Perceptron

- **Perceptron** é uma rede muito simples.
  - Quando constituída de apenas um neurônio é chamada de **perceptron elementar**.
  - Possui **apenas uma camada de neurônios**.
  - **Pode ter várias entradas e várias saídas**.
  - Trabalha com **valores discretos tanto para as entradas quanto para as saidas**.
  - Quando há **valores contínuos, as redes com essas características são ditas Adaline**.
  - **Só classifica dados linearmente separáveis.**

Redes Neurais: Perceptron - Fase de Treinamento

- Como determinar a **topologia** da rede ?
  - Exemplo:

cpf	nome	renda	dívida	classificação do cliente
111...	João	2000	1000	bom
222...	Maria	3000	2000	mau
333...	Pedro	1000	500	mau
444...	Carlos	3000	1500	bom

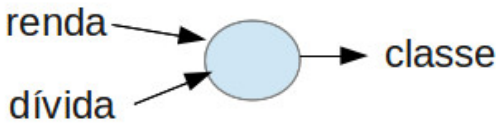
- **Entradas**:
  - Descrevem características significativas a partir dos quais a rede possa extrair padrões.
  - Quais são as entradas da rede ?
- **Saídas** :
  - A quantidade de saídas determina o número de neurônios (para redes "alimentadas para frente" de uma única camada) .
  - Quais são as saídas da rede ?

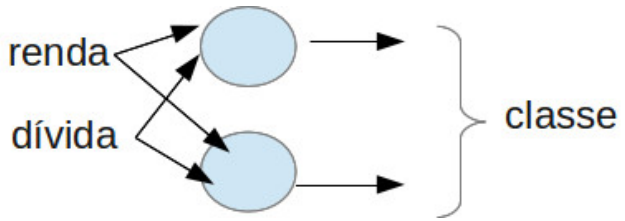
Redes Neurais: Perceptron - Fase de Treinamento

- Pré-processamento
  - Exemplo: (processamento simples: divisão pelo maior valor)

renda	dívida	classificação do cliente
0,66	0,5	1
1	1	0
0,33	0,25	0
1	0,75	1

- Duas topologias são viáveis:





Redes Neurais: Perceptron - Fase de Treinamento

- Os **ciclos de treinamento** de uma rede são medidos em **épocas**.
- Uma **época corresponde a passagem de todos os padrões do conjunto de treino uma vez pela rede**.
- Para treinar uma rede são necessárias várias épocas.

Redes Neurais: Perceptron - Fase de Treinamento

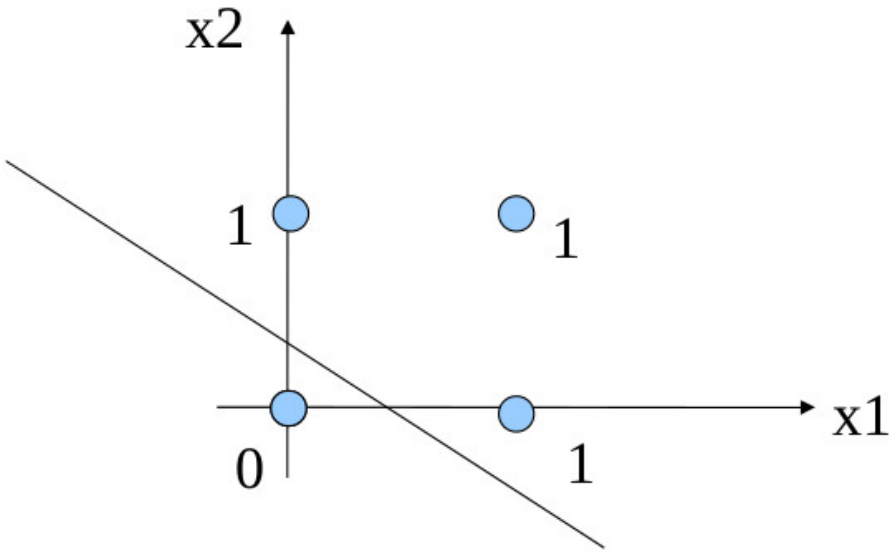
- **Rede Perceptron** (Adaline):
  - **Algoritmo de Treinamento: Regra Delta**
  - Sendo $X = \{(amostra_1, d_1), (amostra_2, d_2), ...\}$o conjunto de treino e η, a taxa de aprendizagem (deve ser positiva).

```
Inicializa os pesos w da rede com zero
 Repetir até encontrar erro zero para todas as amostras{
    epocas = epocas + 1
     Para cada par de X {
        Para cada atributo x_i da amostra, onde i = 1 a n{
            Para cada neurônio k da rede{
                v_k = w_ki * x_i
                y_k = Q(v_k)
            erro_k = d_k – y_k
               Δw_ki =  η * erro_k * x_i
               wki = w_ki + Δw_ki
                }
            }
        }
```

Redes Neurais: Perceptron - Fase de Treinamento

- **Rede Perceptron** (Adaline):
  - **Limitações**:
    - A rede só consegue convergir quando os dados de entrada são linearmente separáveis.
    - Exemplo: OR

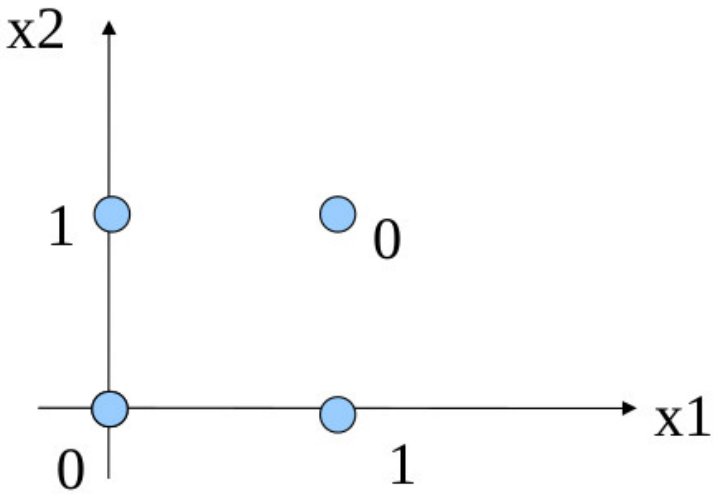
x1	x2	d
0	0	0
0	1	1
1	0	1
1	1	1



Redes Neurais: Perceptron - Fase de Treinamento

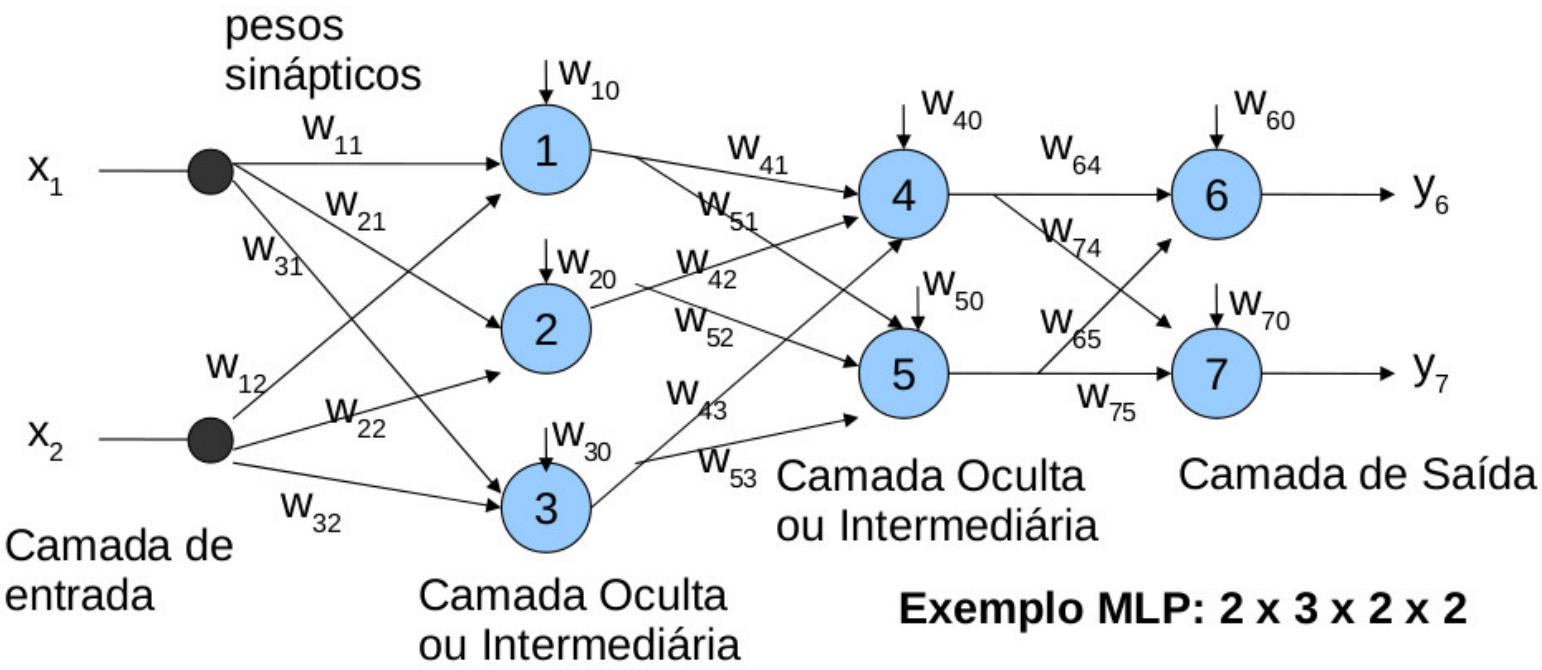
- **Rede Perceptron** (Adaline):
  - **Limitações**:
    - O XOR a rede não consegue resolver.
    - Não há como traçar uma linha (equação da reta) que separe as duas classes.

x1	x2	d
0	0	0
0	1	1
1	0	1
1	1	0



Redes Neurais: MultiLayer Perceptron (MLP)

- MLPs são redes **perceptron** de múltiplas camadas, alimentadas para frente (feed forward), **contendo uma ou mais camadas ocultas**.

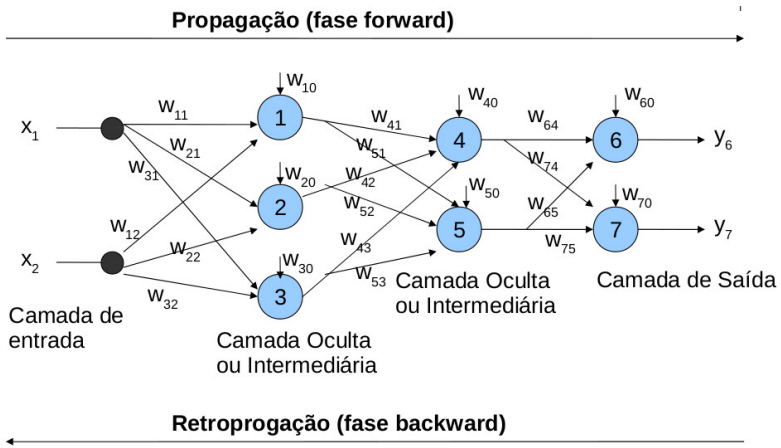


MultiLayer Perceptron (MLP): Topologia

- Para determinar o número de neurônios da camada de saída, basta determinar a quantidade de saídas possíveis da rede. Pode-se usar uma heurístuca:
  - um neurônio dedicado para cada classe
  - combinar as saídas de todos os neurônios para representar as classes (abordagem binária).
- Para estimar o número inicial de neurônios de uma camada oculta, pode-se usar a seguinte heurística:
  - $numeroNeuronios = \sqrt{entradas \times saidas}$,
  - Exemplo: MLP de duas camadas: 10 x ? x 2 (representação binária)
    - entradas: 10
    - saidas possiveis: 4
    - numNeuronios $= \sqrt{30} = 6.32 \approx 6$

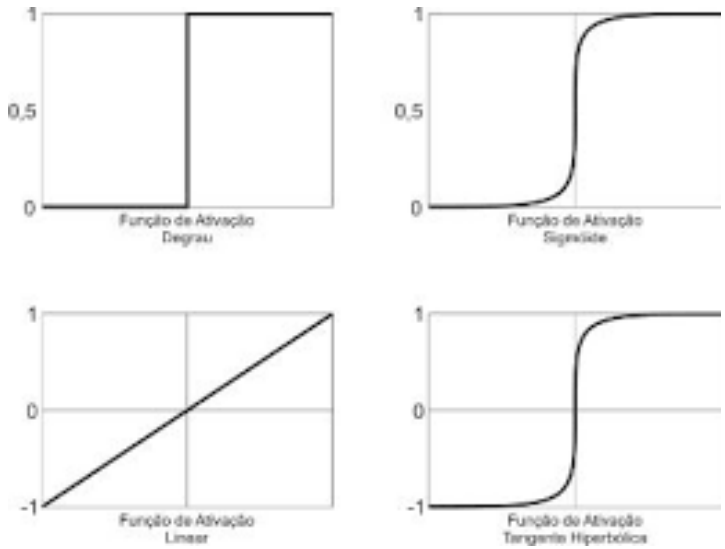
MultiLayer Perceptron (MLP): Treinamento

- O algoritmo **Error Backpropagation é usado para treinar a rede**.
- Ele possui 2 etapas: **forward** e **backward**.
- Em cada etapa a rede é percorrida em um sentido.
  - A fase forward (para frente) – **propagação** - é utilizada para definir a saída da rede para um dado padrão de entrada.
  - A fase backward (para trás) – **retropropagação** - utiliza a saída desejada e a saída gerada pela rede para atualizar os pesos de suas conexões.



MultiLayer Perceptron (MLP): Treinamento

- **Funções de transferência (ou de ativação) típicas**
  - Limiar (ou degrau): $Q(v_k) = 1$ se $v_k \geq 0$; 0 c.c.
  - Linear: $Q(v_k) = a \times v_k + b$
  - Sigmóide (Logistica): $Q(v_k) = \frac{1}{1+exp(-v_k)}$
  - Tangente Hiperbólica: $Q(v_k) = tanh(v_k)$

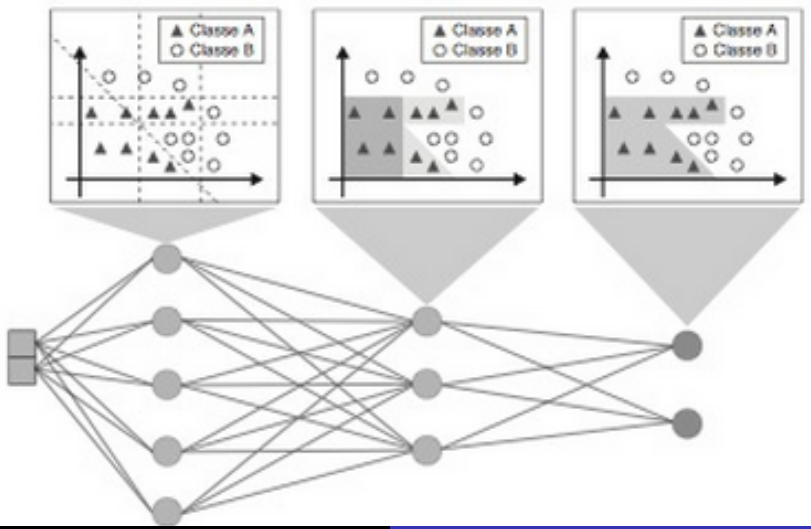


MultiLayer Perceptron (MLP): Treinamento

- **Funções de transferência** (em Java e C)
  - Logística:
    - valores entre [0;1]
    - $Q(v_k) = \frac{1}{1+exp(-v_k)}$
  - Tangente hiperbólica:
    - valores entre [-1;1]
    - $Q(v_k) = tanh(v_k)$

MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - Na primeira camada, cada neurônio aprende uma função que define um hiperplano, o qual divide o espaço de entrada em dois.
  - Os neurônios da camada seguinte, combinam hiperplanos gerados pelos neurônios da camada anterior e formam regiões convexas.
  - Os neurônios da próxima camada combinam um subconjunto das regiões convexas em regiões de formato arbitrário.
  - A combinação das funções desempenhadas por cada neurônio define a função associada à rede como um todo.



MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - Considere que a topologia da rede já está definida e que há um conjunto de Treino com N pares (X,D), onde:
    - X é o conjunto de entrada: $\{x_1, x_2, x_3, x_4, ..x_Z\}$, com Z entradas (atributos)
    - D é o conjunto de saídas desejadas:$\{d_1, d_2, d_3, d_4, ..x_M\}$, com M saídas (uma para cada neurônio da camada de saída)

MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - **Etapas :**
    1. Iniciar os pesos da rede **arbitrariamente com valores não nulos.**
    2. Apresentar cada padrãon de entrada do conjunto de treino e **propagá-lo** até a saída da rede (geração dos y's da camada de saída), onde $n = 1$até N.
    - **Propagação:** Para cada neurônio k da rede.
      - $v_k(n) = \sum_{i=0}^{z} (w_{ki}(n) \times x_i(n))$, onde k é o neurônio e i a entrada, para $i = 0$ até z (total de entradas, ou seja, total de atributos de um padrão n do conjunto de treino). Quando o neurônio for de uma camada oculta, x_i será y_i(saida da neurônio i da camada anterior).

MLP - Algoritmo de Treinamento

- **Algoritmo Error-Backpropagation (Retropropagação do Erro)**
  - **Etapas :**

    ..

    2. Apresentar cada padrãon de entrada do conjunto de treino e **propagá-lo** até a saída da rede (geração dos y's da camada de saída), onde $n = 1$até N.

    - **Propagação:** Para cada neurônio k da rede.
      - $v_k(n) = \sum_{i=0}^{z} (w_{ki}(n) \times x_i(n))$, ...
      - $y_k(n) = Q(v_k(n))$, a saída é gerada pela aplicação da função de transferência Q sobre o campo local induzido v.

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation : :**

  ..

  3. Iniciar a **Retropropagação**.

  a) Calcular o erro. Para cada neurônio k da camada de saída: $erro_k(n) = d_k(n) - y_k(n)$, onde d_k é a saída desejada, o rótulo (a classe) do padrão n.

  b) Calcular a energia do erro instantâneo para o padrão n propagado. $\xi(n) = \frac{1}{2} \sum_{k=1}^{s} erro_k(n)^2$, onde n identifica o padrão e $k = 1$até s (o número de neurônios da camada de saída). O erro instantâneo combina os erros de todos neurônios da camada de saída para o padrão então propagado.

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  c) Calcular os gradientes δ da camada de saída.

  - Para cada neurônio k da camada de saída.

  $\delta_k(n) = Q'(v_k(n)) \times erro_k(n)$

  d) Calcular o ajuste dos pesos de k:

  $\triangle w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$, onde é a η taxa de aprendizagem – intervalo típico (0;1].

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  e) Ajustar os pesos dos neurônios da camada de saída:
  $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n)$.
  - Pode-se usar ainda a **constante de momento** α, intervalo típico [0;1]: $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n) + \alpha \times w_{ki}(n-1)$

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  f) Calcular os gradientes δ das camadas ocultas: Para cada neurônio k de uma camada oculta:
  $\delta_k(n) = Q'(v_k(n)) * \sum_{j=1}^{t} (\delta_j(n) * w_{jk}(n+1))$, onde j são os neurônios com os quais o neurônio k tem conexão à direita. Como a camada à direita já sofreu retropropagação, seus pesos já foram atualizados, por isso aparece $n+1$.

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**

  ..

  3. Iniciar a **Retropropagação**.

  ...

  g) Da mesma forma calcular o ajuste dos pesos de k:
  $\triangle w_{ki}(n) = \delta_k(n) \times \eta \times y_i(n)$
  h) Também da mesma forma, ajustar os pesos dos neurônios das camadas ocultas:
  $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n)$.
  - Pode-se usar ainda a **constante de momento** α, intervalo típico [0;1]: $w_{ki}(n+1) = w_{ki}(n) + \triangle w_{ki}(n) + \alpha \times w_{ki}(n-1)$

MLP - Algoritmo de Treinamento

- **Etapas do Algoritmo Error-Backpropagation :**
  - ...
  - Ao final de uma época, **calcular o Erro Médio Quadrado (EMQ):** Média aritmética dos erros instântaneos.
    - $EMQ = (\sum_{i=1}^{N} \xi_i)/N$
    - É usado como critério de parada.
    - Pode oscilar no inicio da aprendizagem, mas deve decrescer ao longo do treinamento.

MLP - Algoritmo de Treinamento

- **Critérios de parada** do algoritmo backpropagation:
  - número pré-definido de épocas;
  - valor pré-definido como desejado para o erro médio quadrado;
  - variação do erro médio quadrado nas últimas x épocas inferior a um valor pré-definido (convergência);
  - número de padrões corretamente classificados não se alterar;
  - combinação desses critérios.

MultiLayer Perceptron (MLP): Treinamento

- Algumas **heurísticas para melhorar o treinamento**:
  - Taxas de aprendizagem η usadas: 0.01, 0.1, 0.3, 0.5 e 0.9;
    - Colocar em um gráfico: *taxa* $\times$ *erro*
    - A taxa que proporcionar a melhor curva de aprendizagem, determinada pelo erro médio, é a melhor escolha.
  - Atualização da taxa de aprendizagem ao longo da execução (vai sendo reduzida).
  - Algumas constantes de momento α usadas: 0, 0.1, 0.5 e 0.9;
  - Adição de constantes às funções de transferência:
    - $Q(v_k) = \frac{1}{1+exp(-a \times v_k)}$, onde $a = 1.7159$
    - $Q(v_k) = a \times tanh(b \times v_k)$, onde $a = 1.7159$ e $b = \frac{2}{3}$
  - Submeter os padrões do conjunto de treino aleatoriamente.
  - Submeter primeiramente o padrão que na época anterior gerou o maior erro.

MultiLayer Perceptron (MLP): Generalização

- **Generalização**:
  - Após o treinamento, os pesos que mapeiam os padrões de entrada nas saídas desejadas foram encontrados.
  - A generalização consistem em propagar pela rede, usando os pesos encontrados, os padrões pré-processados do conjunto de teste e analisar os resultados gerados pela rede.
  - Visto que as funções de transferência geram valores contínuos é comum um pós-processamento da saída gerada.
    - O pós-processamento é uma regra de decisão, geralmente baseada em algum valor limiar que auxilia a definir a classe do padrão de teste.
    - Ex: se y >= 0.8 então y=1

MultiLayer Perceptron (MLP): Validação

- Validação: Um dos principais problemas na construção de uma RNA é a sua configuração.
- Apesar de existirem algumas heurísticas, a busca da configuração adequada é na base de tentativa e erro.
- Faz parte da configuração da rede determinar:
  - Número de camadas da rede
  - Número de neurônios por camada
  - Valores adequados para η e α (constante de momento).
  - Funções de transferência para cada camada da rede.

MultiLayer Perceptron (MLP): Validação

- Para determinar a melhor configuração da rede é comum o uso de validação cruzada.
- Validação cruzada: consiste em subdividir o conjunto de treino e dois conjuntos disjuntos: estimação e validação.
  - Normalmente, o conjunto estimação possui 80% do conjunto de treino; e
  - os 20% restantes são do conjunto de validação.

MultiLayer Perceptron (MLP): Validação

- Define-se então algumas configurações possíveis para a rede, tais como:
  - topologia,
  - valores para η e α;
  - funções usadas
- Para cada configuração treina-se com o conjunto de estimação e testa-se com o conjunto de validação.
- A configuração que obtiver melhores resultados é usada na fase de generalização.

MultiLayer Perceptron (MLP): Validação

- Existem algumas variações como a validação cruzada múltipla, que consistem em variar os conjuntos de estimação e validação.
- Neste tipo de validação, o conjunto de treino é particionado em Nconjuntos disjuntos.
- Por exemplo, se o conjunto de treino possui 500 padrões e o valor de N for 10, serão gerados 10 subconjuntos: $C1, .., C10$.
  - Neste caso, serão feitos 10 combinações de conjuntos de estimação e validação.
    - estimação: $C2 \cup C3 \cup C4 ... C10$, validação: $C1$
    - estimação: $C1 \cup C3 \cup C4 ... C10$, validação: $C2$
    - estimação: $C1 \cup C2 \cup C4 ... C10$, validação: $C3$
    - ...
    - estimação: $C1 \cup C2 \cup C4 ... C9$, validação: $C10$

MultiLayer Perceptron (MLP): Problemas

- **Overfitting**
  - Provocada pelo excesso de neurônios
  - A rede memoriza o conjunto de treino
- **Underfitting**
  - Provocado pela falta de neurônios
  - A rede não encontra o modelo